взлом и защита программ

Спец Хакер

Спец Хакер

№08(57) АВГУСТ 2005

ЕЖЕМЕСЯЧНЫЙ ТЕМАТИЧЕСКИЙ КОМПЬЮТЕРНЫЙ ЖУРНАЛ

(anti)cracking

Взлом и защита программ

БОНУС

Тест Кулеров LGA775

НА CD ASPack 2.12 ■ ASProtect 1.23 DJ Java Decompiler (ver. 3.8.8.85) Hex Workshop 4.23 ■ Microsoft Spy++ ■ VBReFormer PE Tools v1.5.400.2003 Xmas Edition ■ StealthPE 2.1.1 PEiD v0.93 ■ RACEVB6 3.4 ■ SourceRescuer ■ REC 1.6

(game)land

ISSN 1609-1027

9 771609 102006 08>

Создай свою реальность

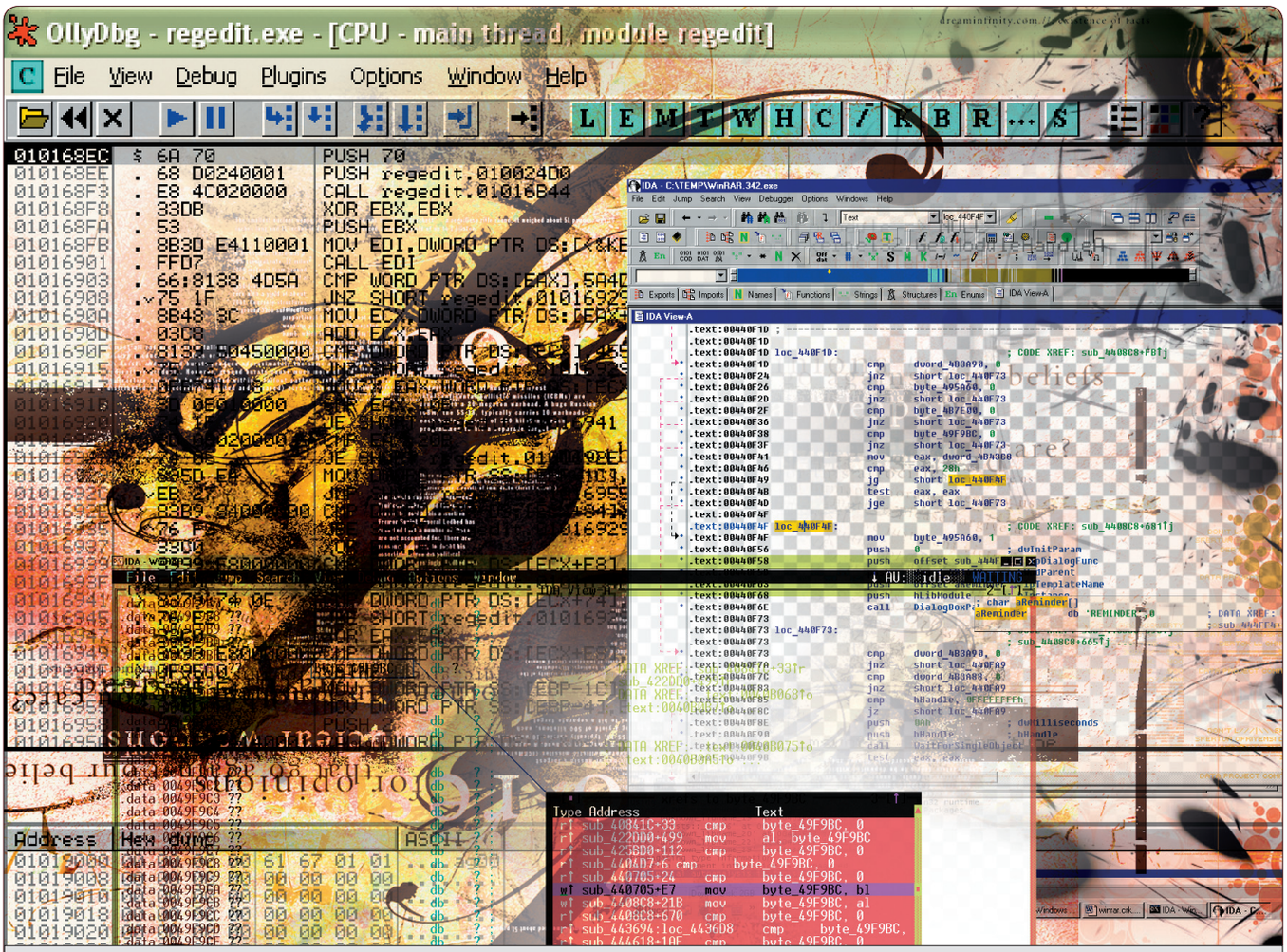
с компьютером DEPO Ego на базе процессора Intel® Pentium® 4 с технологией HT

Включи DEPO Ego — и перед тобой откроется новая реальность твоих любимых компьютерных игр. Наслаждайся быстротой реакции и скоростью, исследуй распахнувшийся перед тобой мир высококачественной компьютерной графики и настоящего экшена. Теперь эта цифровая реальность может стать твоей благодаря компьютеру DEPO Ego на базе процессора Intel® Pentium® 4 с технологией HT.

DEPO Ego 360 TV:

- процессоры Intel® Pentium® 4 с технологией HT серии 6xx (2Мб cash второго уровня)
- чипсет Intel® 925XE с улучшенной архитектурой
- сверхбыстрая память DDR2
- новые возможности графики PCI–Express
- реалистичный объемный 8–канальный звук

Компания DEPO Computers Тел./факс: (095) 969–2215, www.depo.ru



INTRO

Процесс любого исследования очень интересен и притягателен. Познание как таковое присуще человеку с древнейших времен. Именно благодаря познанию мы стали приобретать отличия от обезьян и превратились, согласно теории Дарвина, в людей. Одно из достижений эволюции человечества – мир информационных технологий. Многие справедливо считают компьютер одним из величайших изобретений в нашей истории. Сегодня информационные технологии – это целая индустрия, без которой немыслимы быт и бизнес современного человека.

А начиналось все с того самого кода: первые программисты упрямо набивали его, всматриваясь в монохромные экраны. Шли годы, появлялись новые технологии и языки, программные продукты становились все сложнее. Также благодаря жажде человека к познанию попытки постигнуть чужой код породили появление reverse engineering – обратной разработки, подразумевающей восстановление алгоритма программы путем анализа скомпилированного выполняемого файла. Так появились крэкеры, скоро они стали объединяться в крэк-команды. Сегодня в мире насчитывается уже более ста крупных крэкерских групп, reverse engineering перестает быть просто хобби и формирует отдельную культуру. Настоящий крэкер никогда не станет заниматься исследованием кода ради заработка – для него это прежде всего познание, которое ценнее работы и гораздо ценнее простого увлечения. Крэкинг – это стиль жизни.

Этот Спец полностью посвящен взлому и защите программ. Новичок найдет для себя много интересного и необходимого для занятий крэкингом, а профессионал - немало полезных советов.

Оганесян Ашот

СОДЕРЖАНИЕ № 08 (57)

РЕВЕРС

ВЗЛОМ

ЗАЩИТА

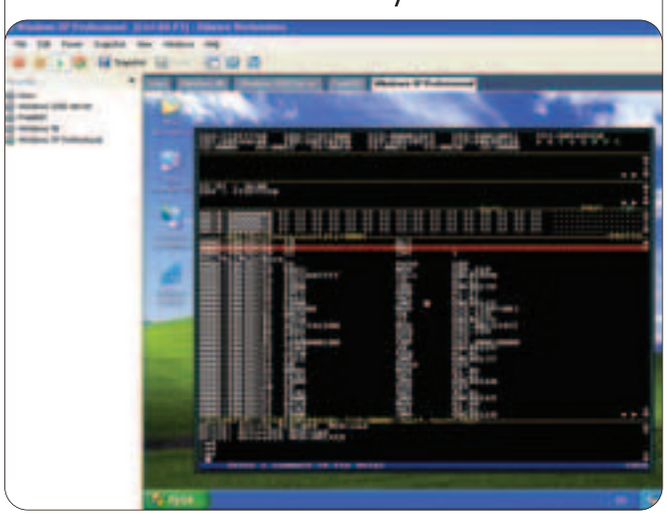
SPECial delivery

ЭКСПЕРТ НОМЕРА

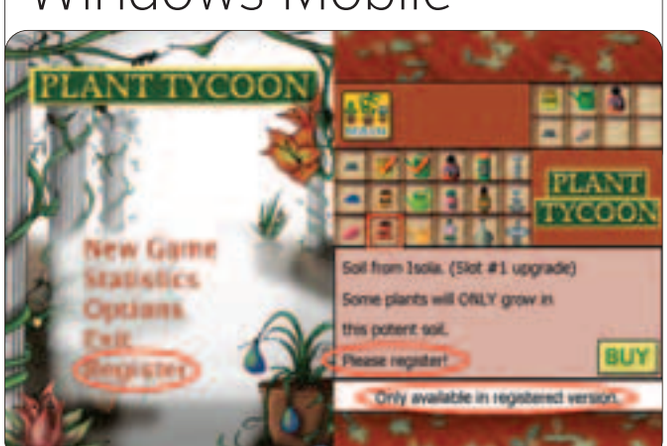
Крис Касперски aka мыщъх

Известный специалист в области IT-безопасности, автор многочисленных книг и статей

РЕВЕРС

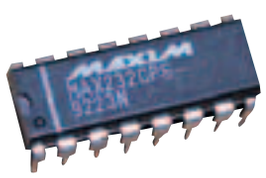


ВЗЛОМ



ОФФТОПИК

HARD



STORY

ЗАЩИТА

HARD

Редакция
» главный редактор
Николай «AvaLANche» Черепанов
(avalanche@real.xakep.ru)
» выпускающие редакторы
Ашот Оганесян
(ashot@real.xakep.ru),
Николай «GorluM» Андреев
(gorlum@real.xakep.ru)
» редакторы
Александр «Dr.Klouniz» Лозовский
(alexander@real.xakep.ru),
Андрей Каролик
(andrusha@real.xakep.ru)
» редактор CD и раздела ОФФТОПИК
Иван «SkyWriter» Касатенко
(sky@real.xakep.ru)
» литературный редактор, корректор
Валентина Иванова
(valy@real.xakep.ru)

Art
» арт-директор
Кирилл «KROt» Петров
(kerel@real.xakep.ru)
Дизайн-студия «100%КПД»
» верстальщик
Алексей Алексеев
» художник
Константин Комардин

Реклама
» директор по рекламе ИД (*game*)*land*
Игорь Пискунов (igor@gameland.ru)
» руководитель отдела рекламы цифровой и игровой группы
Ольга Басова (olga@gameland.ru)
» менеджеры отдела
Виктория Крымова (vika@gameland.ru)
Ольга Емельянцева
(olgaeml@gameland.ru)
» трафик-менеджер
Марья Алексеева
(alekseeva@gameland.ru)
тел.: **(095) 935.70.34**
факс: **(095) 780.88.24**

PR
» директор по PR цифровой группы
Глеб Лашков
(lashkov@gameland.ru)

Распространение
» директор отдела дистрибуции и маркетинга
Владимир Смирнов
(vladimir@gameland.ru)
» оптовое распространение
Андрей Степанов
(andrey@gameland.ru)
» региональное розничное распространение
Андрей Наседкин
(nasedkin@gameland.ru)
» подписка
Алексей Попов
(popov@gameland.ru)
тел.: **(095) 935.70.34**
факс: **(095) 780.88.24**

PUBLISHING
» издатель
Сергей Покровский
(pokrovsky@gameland.ru)
» учредитель
ООО «Гейм Лэнд»
» директор
Дмитрий Агарунов
(dmitri@gameland.ru)
» финансовый директор
Борис Скворцов
(boris@gameland.ru)

Горячая линия по подписке
тел.: 8 (800) 200.3.999
Бесплатно для звонящих из России

Для писем
101000, Москва,
Главпочтамт, а/я 652, Хакер Спец

Web-Site
http://www.xakep.ru

E-mail
spec@real.xakep.ru

Мнение редакции не всегда совпадает с мнением авторов. Все материалы этого номера представляют собой лишь информацию к размышлению. Редакция не несет ответственности за незаконные действия, совершенные с ее использованием, и возможный причиненный ущерб.
За перепечатку наших материалов без спроса - преследуем.

Отпечатано в типографии «**ScanWeb**», Финляндия

Зарегистрировано в Министерстве Российской Федерации по делам печати, телерадиовещанию и средствам массовых коммуникаций **ПИ № 77-12014** от 4 марта 2002 г.

Тираж **42 000** экземпляров.
Цена договорная.

Content:

Симонов Илья aka Shturmovik (nazi@ghOsts.org)

НАБОР ИНСТРУМЕНТОВ

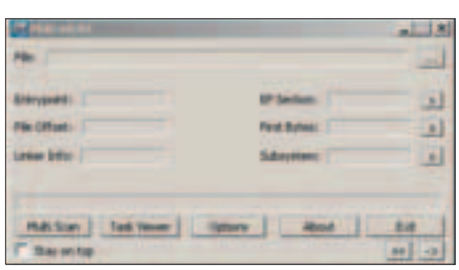
ОБЗОР ПОМОЩНИКОВ ИССЛЕДОВАТЕЛЯ

Твоя любимая программа запросила денег, или тебе просто интересно раскрывать чужие секреты? Torga, думаю, стоит заняться реверс-инженерингом, а еще лучше - крэкингом. Думаешь, с чего начать? Наверное, лучше начать с обучения использованию стандартных инструментов крэкера. Чем мы сегодня и займемся.

НАЧАЛО НАЧАЛ

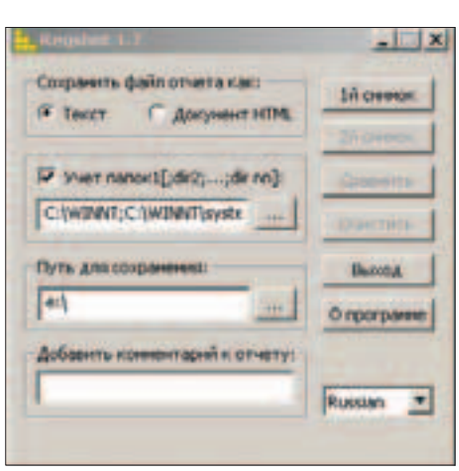
А начинать необходимо всегда с простого, чтобы понять сложное. Для начала узнаем, чем защищена и защищена ли вообще программа или CD с игрушкой, как она шаманит с файловой системой, где следит в реестре. В этом нелегком деле поможет целый набор уже готовых программ.

PEID



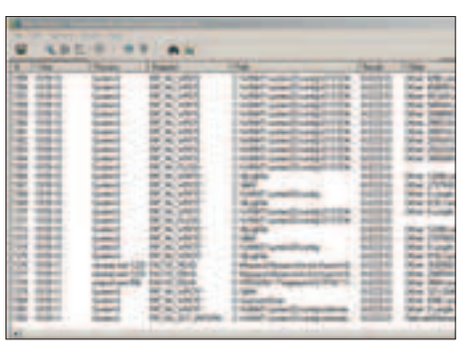
Первоочередное дело данной программы - показать, на чем написан исследуемый файл, чем он упакован, если упакован вообще. Также мы увидим информацию о секциях файла и другие атрибуты. Имеется встроенный дизассемблер начала кода. Если считать стандартные плагины, то тут и универсальный распаковщик, и восстановление импорта, и криптоанализ. В общем, это вещь №1 на крэкерском рабочем столе.

REGSHOT



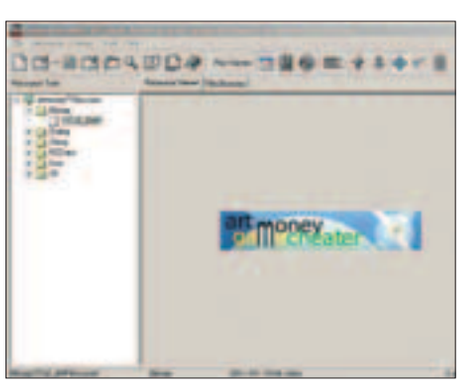
Что делать, когда нужно узнать, где в системном реестре шаманит платная программулина? Можно доверится REgMon'y или же ставить точки останова на обращение к реестру в отладчике. Но, по-моему, проще снять снимок реестра до и после определенной операции. Можно сделать снимок файловой системы и определенных папок. И, самое главное, можно сравнить все это, и тогда пред нами встанет отчет обо всех изменениях реестра и указанных папок. Думаю, больше слов не требуется - осталось запустить.

FILE MONITOR



Иногда, однако, наши пакостники за считанные секунды создают временные файлы, а затем удаляют их. Torga RegShot отходит на второй план и на его место вступает File monitor, который с частым обновлением показывает все обращения к файловой системе в данный момент. Главное - не упустить в быстро растущем списке свою программу и закрыть все, что только можно, перед запуском.

RESTORATOR



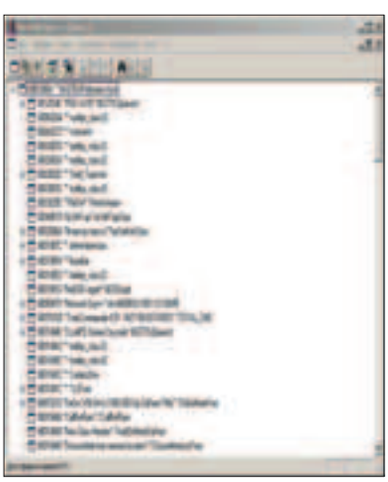
Кто из нас не любит присваивать чужое? Там поменять имя автора в любимой про-

грамме, или же, наоборот, достать интересную картинку из жертвы. К твоим услугам лучший редактор ресурсов для таких дел - Restorator. После установки он встраивается в оболочку, и исправить файл можно кликнув по нему правой кнопкой и выбрав open with restorator. Все иконки/картинки/строки и прочие ресурсы с возможностью исправления будут как на ладони.

PID

Что-то мы все о программах да о программах. Думаешь заняться пиратством, да вот какая-то защита не дает нормально скопировать диск? Хочешь узнать, что же это за напасть? Поручи дело PiD'y, простая, как PeiD - без комментариев.

MICROSOFT SPY++

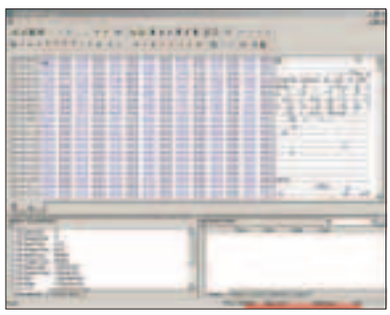


Замечательный шпион за API-вызовами и перехватчик сообщений между приложениями от создателей Windows. Этот простенький шпион покажет тебе все взаимные зависимости окон твоей программы. Есть дельный совет: при уборке NAG'a или баннера хорошо посмотреть размер вышесказанного этим шпионом. А далее в дизассемблер и искать помещение размеров в стек... вроде как мы нашли процедуру построения нехорошей рекламы, а дальше - дело времени.

HEX-РЕДАКТОРЫ

Итак, мы рассмотрели выдающиеся утилиты дознания и мониторинга. Пришло время углубиться внутрь. HEX-редакторы, как понятно из названия - редакторы шестнадцатеричного кода, каким представлены все данные, хранящиеся на компьютере. Вещь, необходимая для быстрого поиска и (если необходимо) правки байт в исполняемой программе.

HEXWORKSHOP

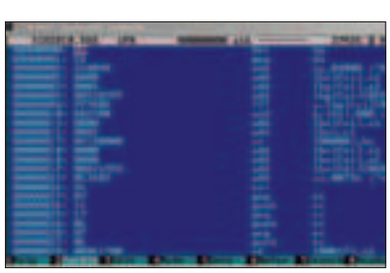


Лучший в своем роде HEX-редактор, с ним может сравниться только WinHEx. Редактирование, расширенный поиск, работа с любой кодировкой, встроенный HEX-калькулятор для быстрых расчетов. Профессиональная версия редактора HEX-кодов файла.

ДИЗАССЕМБЛЕРЫ

Целый класс программ, предназначенных для раскрытия исходного кода программы и представления его в виде ассемблерного листинга. Однако мы никогда не сможем получить достоверный и настоящий листинг программы, что связано хотя бы с тем, что дизассемблеры используют в качестве адресов вызовов процедур адреса памяти в текущее время, тем самым при таком подходе адреса могут быть заняты другой программой в данный момент, а также много отступлений, которые невозможно описать здесь. В целом же перед собой мы видим добротный ассемблерный код того, что делает программа. Без дизасма не обходится ни один крэкер нашего времени.

HIEW



Дизассемблер с возможностью редактирования кода, HEX-редактор с той же возможностью. Пожалуй, это основные отличительные черты данной DOS'овской утилиты. Порой редактирование кода в кустарных условиях - незаменимая вещь, да еще с распознаванием API-функций.

KWINDASM

Старый добрый дизассемблер. Новичкам вполне подойдет. Этот старик переживет еще многих своих собратьев. Отличительных черт не имеет за исключением просто хорошей, стабильной работы качественного дизассемблера.

IDA

Интерактивный дизассемблер, если исключить его незнание русского языка, то он еще и уникальный в своем роде продукт. Советую всем. Это туча функциональных настроек, подсветка различных секций исполняемого файла, работа с ресурсами и подгружаемыми библиотеками, встроенный отладчик, развороты функций и трассирование переходов. Интерактивность дает возможность распознавания полиморфного кода. По всем параметрам очень продвинутый дизассемблер, листинг которого читается очень приятно.

ДЕКОМПИЛЯТОРЫ

Среди языков программирования, некоторые (особенно те, что работают с виртуальными машинами), такие как Visual Basic, Borland C++ Builder, Delphi, оставляют в коде множество "мусора", меток, сообщений. Анализируя эти "остатки", декомпиляторы должны восстановить исходный код программы в первозданном виде. Это отнюдь не значит, что мы увидим все так, как было написано - такое невозможно. Но раскрыть код формы, показать обработчики событий, созданных программистом, и умение раскрывать программу на исходные файлы - это задача декомпилятора.

DEDE

Да, это так. Это действительно лучший декомпилятор Delphi-приложений. Если программист не думал о защите от декомпиляторов, то ты получишь не только дизассемблерный код программы. Код будет привязан к со- »

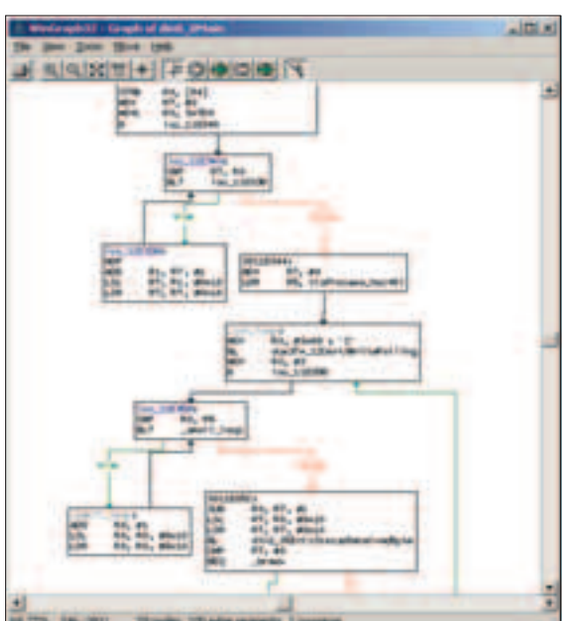
Реверсинг в разгаре

Количество подчеркиваний для меня является показателем уровня, на котором подфункция находится. Так, FormatDiskC - нулевой уровень, FormatDiskC_ - первый, FormatDiskC__ - второй.

❶. Если встречаю функцию, которая вызывает функцию нулевого уровня, но не делает ничего кроме этого (является переходником для преобразования типов), то добавляю подчеркивание перед именем. Это надфункция. Пример: _FormatDiskC, _DoDecrypt.

Имена методов объектов, как обычно, Object::Method.

Давать имена переменным часто бывает довольно сложно. Приходится переименовывать их по три-четыре раза, если попадется какая-нибудь временная переменная. Но главной проблемой всегда было именование копий переменных. К примеру, в функцию передается указатель, и дальше, чтобы не испортить его значение, он копируется в какую-то переменную. То есть сначала именуются аргументы, а дальше приходится именовать копии аргументов в локальных переменных. Локальным переменным - копиям аргументов - я даю имена с подчеркиванием в конце. То есть аргумент функции имеет чистое имя, а у локальных перемен-



Построение графа code-flow функции часто помогает понять пути выполнения кода

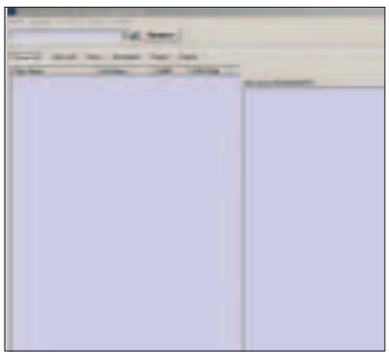
МНЕНИЕ ЭКСПЕРТА

■ **Крис Касперски aka мыщъх:** В дизассемблировании намного больше искусства, чем науки. Хотя машинный анализ делает огромные успехи, сердцем любого декомпилятора по-прежнему является человек. Отличие констант от смещений до сих пор остается фундаментальной проблемой дизассемблирования (так называемая "проблема offset'а"). В двоичном виде их представление идентично, но интерпретируются они по-разному. Нераспознанные смещения в свою очередь не позволяют отличать данные от команд, и это вторая фундаментальная проблема дизассемблирования. Наконец, дизассемблеры не справляются (точнее, практически не справляются) с анализом виртуальных функций, зашифрованного и самомодифицирующегося кода, поэтому приходится запрягать дизассемблер и отладчик в одну упряжку - дизассемблер реконструирует скелет алгоритма, а отладчик расшифровывает запакованные фрагменты кода и уточняет значение регистров ЦП и ячеек памяти в данной точке.

Дизассемблирование решает, по меньшей мере, две взаимоисключающие задачи: реконструирует алгоритм или создает ассемблерный листинг, пригодный к последующей трансляции. Пакетные дизассемблеры (к примеру, SOURCER) нацелены именно на генерацию листинга. Правда, качество генерации оставляет желать лучшего, и прежде чем исследуемая программа, наконец, заработает, над ним придется изрядно попыхтеть, исправляя константы, ошибочно принятые за указатели, или наоборот. К тому же значительная часть кода остается нераспознанной вообще и оформленной в виде массива данных знаменитой директивой DB. Самомодифицирующийся код обнаруживает третью фундаментальную проблему - одна и та же ассемблерная команда часто соответствует целому ансамблю ассемблерных инструкций (например, INC EAX может быть представлена либо как 40h, либо как FFh C0h), выбор которых ложится на плечи транслятора. Следовательно, ассемблирование даже идеального дизассемблерного листинга далеко не всегда дает идентичный двоичный файл.

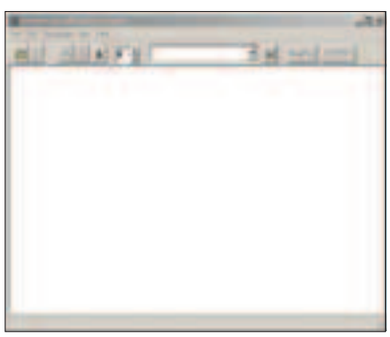
Интерактивные дизассемблеры нацелены именно на анализ алгоритма, для чего они содержат мощную систему навигации, распознаватель библиотечных функций, трассировщик потока управления и многие другие незаменимые инструменты. Приложения и драйверы дизассемблируются очень легко. Фактически все упирается во время. Прошивки - другое дело. Сначала требуется опознать процессор. IDA Pro с этим уже не справляется. Впрочем, опытный кодокопатель опознает тип процессора и самостоятельно. Проблема в другом: код прошивки представляет собой бессмысленный набор машинных команд, перемешанный с обращениями к портам. За что отвечает тот или иной порт - непонятно. Один и тот же набор байт может управлять двигателем, перемещать головку или выводить изображение на экран. Если описание чипсета отсутствует, прошивку приходится разбирать буквально по кусочкам. Это примерно то же самое, что разгадывать кроссворд на китайском языке, не имея под рукой даже словаря. В этом лабиринте своеобразной нитью Ариадны становятся отнюдь не машинные команды, а... структуры данных. Например, при исследовании прошивки пишущего привода нам встретятся ATAPI-команды, заголовки секторов, константы, отвечающие за подсчет EDC/ECC-кодов и т.д.

Дальнейшее развитие техники дизассемблирования немыслимо без финансовых вложений и притока свежих умов. К сожалению, мировая общественность относится к дизассемблированию очень настороженно, считая, что это криминальная методика, пригодная только для взлома. Конечно, без взломов тут не обходится, но если запретить дизассемблирование, как бороться с вирусами? Как отлаживать программы, взаимодействующие с другими приложениями, исходных текстов которых нет?



бытиям, сделанным в Delphi. Также ты сможешь увидеть исходный код формы и ее собственной персоной. Пробуй и наслаждайся.

WISDEC



Замечательный декомпилятор инсталляционных скриптов. Возможно, ты никогда и не будешь заниматься взломом Install Sheild, поскольку защита на этом уровне встречается все-таки довольно редко. Однако пройти мимо было бы кощунством. Программа разбирает скрипт инсталляции по полочкам, и в недрах выявленного кода тебе, может, посчастливится найти заветный серийный номер, если немного подумаешь.

ОТЛАДЧИКИ

■ Боевой софт крэкера. Про то, что отладчик также является дизассемблером, можно было и не упоминать. Первостепенная задача отладчика (естественно, отладка :)) - пошаговое выполнение анализируемого кода, возможность установки прерываний программы в связи с определенными событиями. Особенное отличие отладчика от остальных программ: отладчик работает с программным кодом в памяти компьютера, а не на диске. Оно, в принципе, и понятно: как-никак программа запущена.

NUMEGA SOFTICE

Великий и могучий, чье название внушает уважение всем, кто смог установить его и поработать с ним. Отладчик режима ядра, входящий в состав Driver Studio. Главные плюсы - это загрузка до основных модулей операционной системы и возможность отладки абсолютно всех модулей. Это нулевое кольцо, а значит, высочайшие привилегии для отладки. Перехват критических ошибок системы и последующая отладка, вплоть до перехвата синего окна смерти. Естественно, работает только в текстовом режиме.

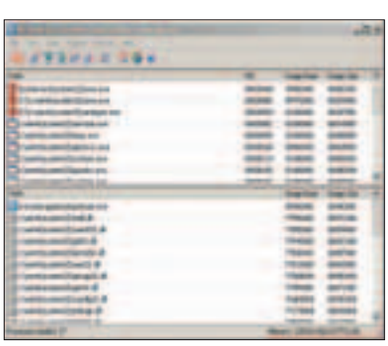
OLLYDBG

Да, это тоже отладчик, но только режима пользователя. Ты скажешь, что это не круто, и скажешь зря. Конечно, SoftIce лучше, но иногда важна простота и дружественный интерфейс, что явно преобладает у полюбившейся крэкерским кругам "Оли" :). Из отличий можно упомянуть, что Olly хорошо использовать как достойный диз-ассемблер. Явное отличие - возможность самостоятельно загружать для отладки не только exe-, но и dll-файлы, что добавит огромный плюс новичку.

УНИВЕРСАЛЬНЫЕ УТИЛИТЫ

■ Из универсальных утилит хочу выделить утилиты для получения снимков памяти, восстановления импорта, редакторов секций файлов и многих добавочных функций, чем все эти программы обладают в полной мере.

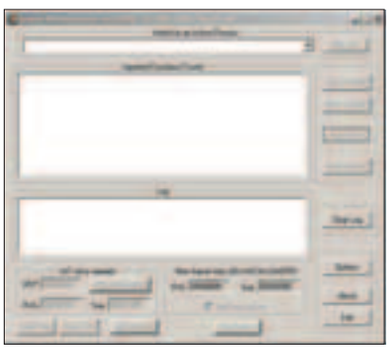
PETOOLS



Наконец-то ты нашел OEP, зациклил на нем программу. Остается лишь сохранить снимок памяти с распакованной программой. Обратимся за помощью к отличной универсальной утилите от нашего соотечественника PETools. Правый клик на нашем процессе - и Full dump! Кроме того, у программы есть еще много удивительно полезных функций, мимо которых пройти невозможно. Это Break&Enter - команда, помогающая прерваться в SoftIce на точке входа в программу. Это свой редактор PE-файлов. Мы можем менять названия секций, атрибуты и многое другое. Из подобных PETools можно выделить LordPE by Yoda.

В общем, смотри - все увидишь и поймешь сам. А если что-то непонятно, то не забывай об авторе.

IMPORT RECONSTRUCTOR



Наконец-то мы получили дамп распакованной программы. Вроде бы ничего не мешает нам исследовать ее, однако она даже не запускается. Да, дело в таблице импорта, которую при распаковке мы благополучно пр... потеряли :). Восстанавливать импорт программ призвана утилита Import Reconstructor. Открываем в нем запакованный файл, вводим найденные значения RVA и OEP, жмем Get Imports и (в зависимости от коэффициента кривизны рук), отсеяв невалидные записи, фиксируем их в дампе программы. Другой такой программы по восстановлению импорта, которая так же хорошо справляется с задачей, я назвать не могу. Уж извините, так сложились звезды, и его могущество - код.

НАПОСЛЕДОК

■ Как? Ты уже сломал программу? И даже сделал свой первый крэк. И он наверняка валяется пустым файликом без имени и фамилии. Torga

флаг тебе в руки, и NFOmaker, и ему подобные. С помощью такой замечательной программы упрощается создание .nfo- и .diz-файлов, которые, несомненно, должны присутствовать в архиве с крэком. Что там писать - уже твое дело. Почитай чужие, если сам придумать не сможешь. В общем, до новых встреч. Буду надеяться, что к тому времени ты уже напишешь свой инструмент.

Наслаждайся ***разнообразием!***

Экосистема кораллового рифа является наиболее разнообразной и сложно устроенной во всей биосфере. Коралловые рифы служат домом для многочисленных видов рыб, крабов, моллюсков, червей, губок и водорослей, обеспечивая их пищей и убежищем. Хотя коралловые рифы занимают менее 0.2% площади океанского дна, в их биоценозах обнаружена четверть всех известных животных и растений океана.

R-Style® Proxima® MC-e

на базе процессора Intel® Pentium® 4 560 с технологией HT

Разнообразие возможностей для отдыха, развлечений и самообразования дает **развлекательный центр R-Style® Proxima® MC-e.** Благодаря мощным процессорам Intel® Pentium® 4 560 с технологией HT, он заменит Вам музыкальный центр, DVD-рекодер и компьютер.

Краткие технические характеристики:

Процессор Intel® Pentium® 4 560 с технологией HT
Операционная система: Microsoft® Windows® XP Media Center Edition 2005
Звук: поддержка стандарта Dolby Digital 7.1 (до 8 каналов)
TV-тюнер: PAL/SECAM
Пульт дистанционного управления
Комплект беспроводных устройств: клавиатура, манипулятор «мышь»

Система качества проектирования, разработки и производства компании R-Style Computers сертифицирована по международному стандарту ISO 9001-2000.

Астрахань ТАН (8512) 394-254 **Братск** Байт (395-3) 411-121 **Владивосток** Эр-Стайл ДВ (4232) 205-410 **Воронеж** Элмар Трейд (0732) 512-018 **Екатеринбург** R-Style (3432) 616-086 **Калининград** Балтик Стайл +7(0112) 99-11-99, 99-11-98 **Кострома** ИТ-Проффесионал (0942) 626-903 **Кемерово** Конкорд ПРО (3842) 357-888 **Краснодар** ВСС Company (8612) 640-450 **Красноярск** ЛанСервис (3912) 75-12-91/92/93 **Москва** R-Style Trading (095) 514-14-14, Компания R-Style (095) 514-14-10, Профит-М (095) 786-77-37, Сибкон (095) 292-50-12 **Нижний Новгород** Эр-Стайл Волга (8312) 464-328, 461-622 **Новосибирск** Эр-Стайл Сибирь (383-2) 661-167 **Пермь** Эр-Стайл Кама (3422) 107-445 **Петрозаводск** Илвес (8142) 762-288 **Петропавловск-Камчатский** АМН (4152) 168-751 **Ростов-на-Дону** Эр-Стайл Дон (863) 252-48-13 **Санкт-Петербург** Эр-Стайл СПБ (812) 445-34-18/17 **Тамбов** Гитон (0752) 719-754 **Тула** ПитерСофт-НТ (0872) 355-500 **Уфа** Онлайн (3472) 248-228 **Хабаровск** Эр-Стайл ДВ регион (4212) 314-530 **Якутск** Эльф (4112) 457333

R·Style COMPUTERS

Оптовые поставки: ООО «Эр-Эс-Ай»: тел.: (095) **514-1419**
www.rsi.ru
Техническая поддержка: ЗАО «Эр-Стайл Компьютерс»: тел.: (095) **514-1417**; бесплатный телефон: **8-800-200-800-7**
www.r-style-computers.ru

Сделано в России. Сделано на совесть!

Иван Скляров (www.sklyaroff.com)

БЕССМЕРТНЫЙ ОТЛАДЧИК

САМЫЙ МОЩНЫЙ ОТЛАДЧИК ВО БЛАГО ЧЕЛОВЕЧЕСТВУ!

Стоит ли объяснять, что такое SoftIce? Даже люди, далекие от крэкинга, обязательно слышали об этом знаменитом отладчике от Compuware Corporation (раньше NuMega). Хотя уже появилось множество хороших отладчиков, таких как OllyDbg, ни одному из них не удалось вытеснить SoftIce.

SoftIce является частью Driver Studio (или Driver Siute). Driver Studio занимает большой объем - в архиве около 200 Мб! В связи с этим крэкерами были созданы маленькие инсталляторы, включающие только SoftIce с самыми важными файлами, извлеченными из Driver Studio. Таким инсталлятором является, например, архив от русского крэкера DeMoNiX, размером всего около 2 Мб (ищи на крэкерских сайтах) и имеющий две отдельных версии (для Win9x и WinNT/2000/XP). Такого "упрощенного" инсталлятора хватает практически для любых задач, и именно на его примере я расскажу о SoftIce.

УСТАНОВКА SOFTICE

Установка "упрощенного" инсталлятора от установки Driver Studio практически ничем не отличается. Самым важным является окно конфигурации. Здесь необходимо в разделе Videos установить видеодрайвер - рекомендуется по умолчанию оставлять "Universal Video Driver", иначе возникнут проблемы. В разделе Mouse нужно честно выбрать тип мыши, который установлен в твоей системе (PS/2, USB). Если мышь имеет более двух кнопок, можно поставить галочку Enchanced Mouse. Также очень важным является раздел Exports - здесь нужно включить необходимые библиотеки, чтобы можно было работать с именами функций WinAPI, иначе SoftIce просто не будет понимать их. Удобнее включать эти библиотеки не через кнопку Add, а с помощью редактирования файла winice.dat, который в NT/2000/XP стандартно устанавливается в папку \%systemroot%\system32\drivers\ (в Win9x он размещался в папке \WINDOWS). В этом файле можно обнаружить строки вида:

```
; EXP=\SystemRoot\System32\hal.dll
; EXP=\SystemRoot\System32\ntoskrnl.exe
; EXP=\SystemRoot\System32\ntdll.dll
```

Перед всеми ними нужно убрать знак комментария ";". Замечу, что инсталлятор от DeMoNiX устанавливает файл winice.dat с уже раскомментированными строками, поэтому ничего делать не нужно, но при установке Driver Studio это не так. Однако не все библиотеки, которые могут понадобиться в работе, упомянуты в winice.dat. Например, если отлаживать программу, написанную на Visual Basic 6.0, нужно будет добавить в список библиотеку msvbvm60.dll, иначе будет невозможна работа с именами функций VB.

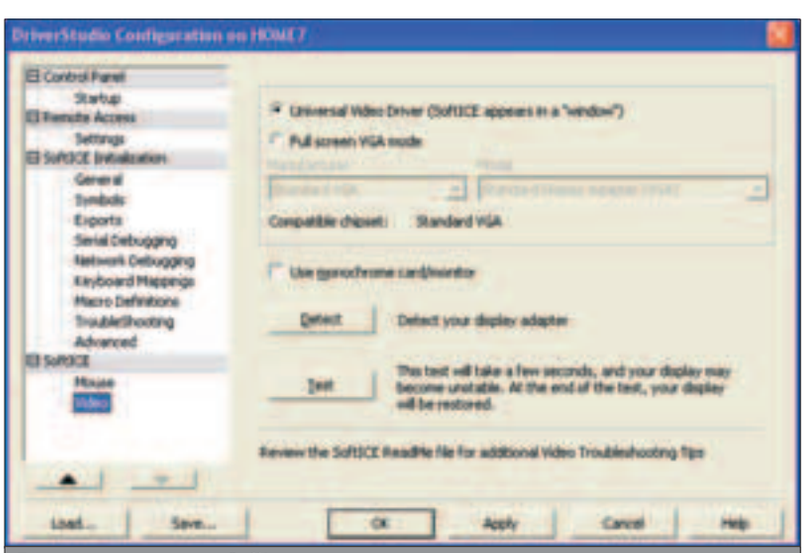
Далее на вкладке Startup, если установка осуществлялась под NT/2000/XP, рекомендуется выставить ручной запуск SoftIce (Manual) (а в Win9x нужно было выставлять режим Boot, который самостоятельно модифицировал файл autoexec.bat). После перезагрузки системы нужно запустить файл NTICE.BAT из папки, куда был установлен SoftIce. Этот файл содержит всего одну команду: "net start ntice", которая запускает в системе службу отладчика ntice.

Проверить работу SoftIce можно нажатием комбинации <Ctrl>+<D> (эта комбинация открывает окно отладчика, и она же закрывает его). Если черный экран с приглашением к вводу появился, то все OK. К сожалению, не все бывает так гладко и часто возникают проблемы с запуском. Конкретные рекомендации здесь дать сложно, так как они зависят от операционной системы, на которой осуществляется установка, а также от версии Driver Studio - может понадобиться установить дополнительный патч. Поэтому рекомендую статью "Установка отладчика SoftICE на Windows XP SP1, SP2" (автор Bad_guy), которая лежит по адресу www.cracklab.ru/art/softice.php, где рассмотрено большинство проблем, возможных после установки SoftIce, и способы их решения.

УСТАНОВКА SOFTICE ПОД VMWARE

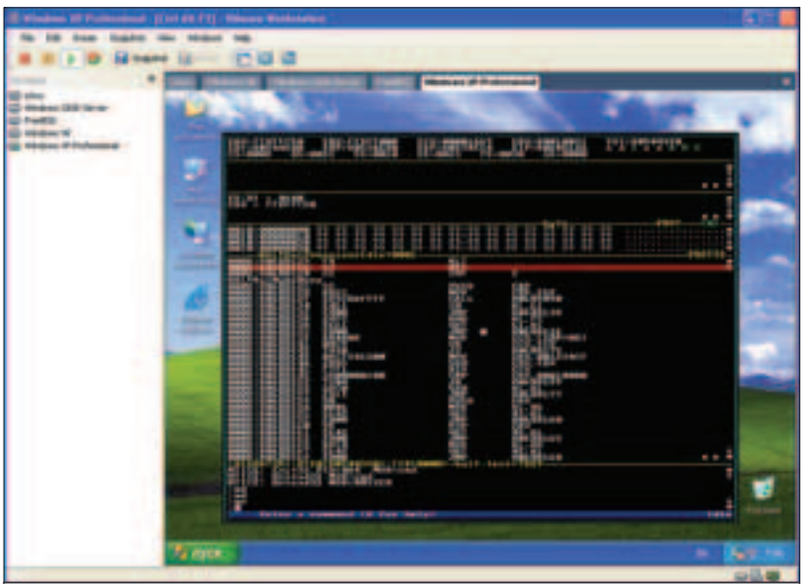
Лучше использовать самую последнюю версию VMware, так как безг-

Установка Driver Studio



Главное окно настроек SoftIce

лючная работа SoftIce во многом зависит от виртуальной машины. В целом установка отладчика в VMware осуществляется точно так же, как в обычной системе. Рекомендую до начала установки в VMware выполнить: File->Install VMware Tools. Но если на гостевой Windows 98SE SoftIce у меня прекрасно устанавливается и работает, то на Windows XP под VMware комбинация <Ctrl>+<D> просто "подвешивала" виртуальную систему. Те, кто ставил когда-либо пакет Driver Studio, знают, что в каталоге \DriverStudio\Books\ размещается документ Using SoftICE.pdf (в инсталляторе от DeMoNiX он отсутствует). В "Using SoftICE" целый раздел посвящен установке SoftIce под VMware: "Appendix E. SoftICE and VMware". В нем рекомендуется добавить в файл с расширением .vmx виртуальной машины следующие строчки:



SoftIce отлично работает под VMware

КОМАНДЫ SOFTICE

■ Основными командами SoftIce, без знания которых невозможна полноценная работа, являются:

BL - просмотреть список установленных бряков (каждый бряк имеет свой номер).
BC [номер бряка] - удалить бряк с определенным номером. Если вместо номера бряка указать звездочку, будут удалены все бряки. Пример: BC *.
BD [номер бряка] - сделать неактивным бряк с соответствующим номером (в списке он будет помечаться звездочкой). Пример: BD 5.
BE [номер бряка] – команда, обратная предыдущей.
Клавиша F8 (аналог команды P) - шаг трассировки с заходом в функцию.
Клавиша F10 (аналог команды T) - шаг трассировки без захода в функцию.
D [адрес], DW [адрес], DD [адрес] - показать содержимое памяти в окне данных (байт, слово, двойное слово) по указанному адресу.
E [адрес] [данные] (EW или ED) - изменить содержимое памяти (байт, слово, двойное слово) по указанному адресу. Пример: E edx 33 - записать байт 33 по адресу, хранящемуся в регистре edx.
R [регистр] [данные] - записать новое значение в регистр. Пример: R eip 50E5B8.
R FL [флаг] - инвертировать значение флага. Пример: R FL Z.
U [адрес] - показать ассемблерный листинг программы в окне кода начиная с указанного адреса. Пример: U 401000.
S [адрес L длина список-данных] - поиск данных в памяти. Пример: S 0 L -1 "pass" - искать слово "pass" в памяти с нулевого адреса до -1 (-1 = FFFFFFFF).

```
vmmouse.present = "FALSE"
svga.maxFullscreenRefreshTick = "2"
```

Эти строчки замечательно решают проблему, что можно подсмотреть на рисунке.

ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ SOFTICE

■ Экран SoftIce состоит как минимум из нескольких окон. В окне команд расположен курсор, позволяющий вводить команды. В окне кода показывается отлаживаемый код - окно можно открыть и закрыть командой WC. Команда WD открывает и закрывает окно данных, соответственно: WR - окно регистров, WS - окно стека, WW - окно слежения, WL - окно локальных переменных. Можно задавать размер любого окна (количество строк в окне) таким образом: WW 10. В окнах действуют обычные клавиши управления: стрелки, End, Home, PageUp, PageDown и пр. Для перемещения курсора в окно кода и обратно используется комбинация клавиш <Alt>+<C>. Комбинация <Alt>+<D> используется для перемещения в окно данных, <Alt>+<R> - в окно регистров, <Alt>+<S> - в окно стека и т.д.

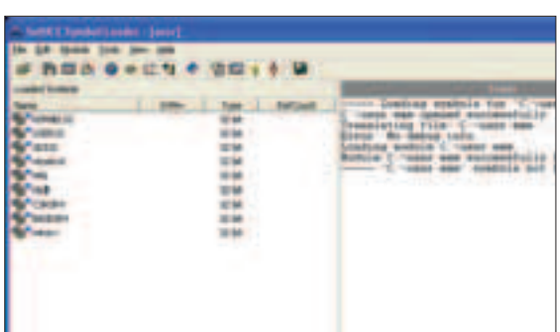
ЗАГРУЗКА КОДА В SOFTICE

■ Открыть программу для отладки в SoftIce можно, по крайней мере, двумя способами.

Способ 1. С помощью загрузчика

Для 32-разрядных программ используется загрузчик Symbol Loader (Пуск->Программы->Numega SoftIceNT-> Loader 32). В нем во вкладке Open нужно открыть отлаживаемую программу (при этом сервис SoftIce должен быть уже запущен с помощью NTICE.BAT). Затем выбрать пункт Load из меню Module. При этом загрузчик создаст символьный файл (если возможно) с расширением .NMS, в который заносится оттранслированная отладочная информация, загрузит символьные и исходные файлы, запустит саму программу и отобразит ее исходный текст в SoftIce. При этом точка прерывания устанавливается на стартовой функции программы (main, WinMain, DllMain).

Для 16-разрядных программ используется загрузчик wldr.exe, а для DOS- »



Symbol Loader используется для загрузки 32-разрядных файлов в SoftIce

программ - загрузчик dldr.exe. Оба находятся в папке UTIL16, где установлен SoftIce. Программу можно загружать прямо из командной строки:

```
/util16>dldr dosproga.exe
```

Способ 2. С помощью установки бряка

Простое открытие программы с помощью загрузчика мало интересно крэкерам. Самое важное, за что любят SoftIce, так это за его умение работать с бряками (англ. breakpoint - точка останова). Бряк позволяет отслеживать определенные события в системе и вызывать отладчик только по их наступлению. Алгоритм крэкера выглядит следующим образом: устанавливается бряк на нужную функцию (на сообщение, память, прерывание), закрывается SoftIce комбинацией <Ctrl>+<D> (или командой x), провоцируется вызов функции в программе (например в результате ввода пароля), после того как бряк сработает и активизируется отладчик, анализ показанного кода. SoftIce поддерживает следующие типы бряков:

BPX - бряк на функцию или адрес в программе. Пример: BPX MessageBoxA. После установки такого бряка, как только в системе произойдет вызов функции MessageBoxA, активизируется SoftIce и курсор в окне кода будет установлен на том коде, который вызвал эту функцию. BPX - наиболее часто используемый крэкерами бряк. Чаще всего он ставится на API-функции: GetWindowText и GetDlgItemText.

BPM - бряк на обращение к памяти. Пример: BPM 602380 - если произойдет обращение к памяти по адресу 602380, сработает SoftIce на том коде, который осуществил обращение по этому адресу. Эта команда имеет несколько разновидностей: BPMW - бряк на обращение к слову, BPMD - бряк на обращение к двойному слову. Также в командной строке можно задавать тип обращения: R - чтение памяти, W - запись в память, RW - чтение и запись, X - выполнение. Пример: BPMD 402438 R.

BPINT - бряк на прерывание, передаваемое через IDT. Пример: BPINT 6F - если произойдет программное прерывание, вызываемое командой INT 6F, то SoftIce покажет код, который вызвал его.

BPIO - бряк на прерывание ввода/вывода. Этот бряк перехватывает обращение к порту команд IN и OUT. Пример: BPIO 3FE - если произойдет прерывание 3FE, активизируется SoftIce, а текущей командой окажется инструкция, следующая за командой IN или OUT, вызвавшей прерывание. В командной строке можно задавать тип обращения: R - чтение из порта (команда IN), W - запись в порт (команда OUT), RW - чтение и запись. Пример: BPIO 2A W.

BMSG - бряк на сообщения Windows. Пример: BMSG 520D WM_GETTEXT - если будет послано сообщение WM_GETTEXT окну с дескриптором 520D, то SoftIce активизируется и покажет тот код, который послал это сообщение.

Любой из перечисленных бряков может быть задан с дополнительным условным выражением [IF выражение] и действием при срабатывании бряка [DO "команда1; команда2; ..."]. Пример: BPX 78C23155 IF (esp->8)==WM_GETTEXT DO "d edx". Если будет послано сообщение WM_GETTEXT процедурой окна 78C23155, то в SoftIce выполнится команда "d edx", которая показывает содержимое памяти в окне данных по адресу, содержащемуся в регистре edx.

Внизу экрана SoftIce всегда можно посмотреть подсказки по синтаксису команд.

ПРИМОЧКИ ДЛЯ SOFTICE

Очень полезно, а иногда даже необходимо установить дополнительные "примочки" (плагины) для SoftIce. Самыми известными такими примочками являются IceDump и IceExt (можно взять на сайте www.wasm.ru). Они умеют скрывать отладчик от антиотладочных приемов в программе, а также дополняют SoftIce многими полезными возможностями и командами. Вот лишь некоторые команды, которые добавляет плагин IceExt:

!BPR - прерывание на диапазон памяти.

!CP - устанавливает кодовую страницу (866 или 1251).

!DUMP - сбрасывает дамп памяти на диск.

!DUMPSCREEN - сбрасывает фотографию окна SoftICE на диск в RAW-формате (потом ее можно преобразовать в BMP-формат программой SiwRender.exe, которая идет в комплекте с IceExt).

!LOADFILE - загружает в память файл с диска.

!PROTECT - переводит SoftICE в режим, невидимый для обнаруживающих его программ.

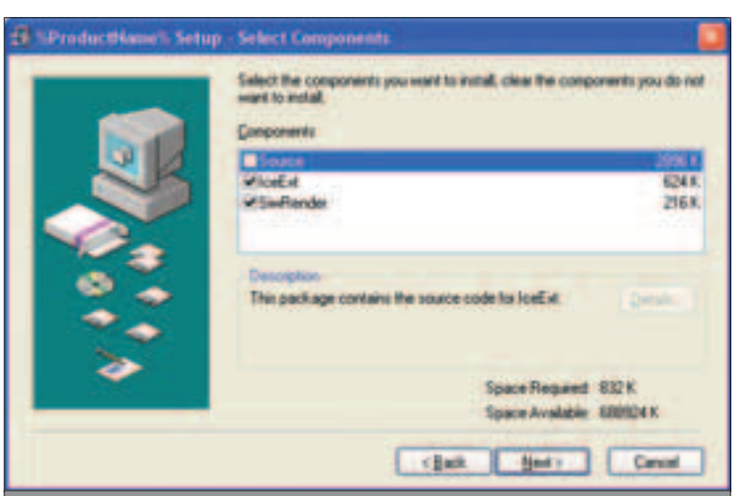
!TETRIS - можно поиграть в тетрис, не выходя из SoftIce.

Установка плагинов происходит довольно просто. Например, для установки IceExt нужно запустить его инсталлятор Setup.exe, по окончании установки выполнить файл iceext.bat (или Пуск->Start IceExt), который запускает сервис IceExt, после чего смело пользоваться командами IceExt в SoftIce.

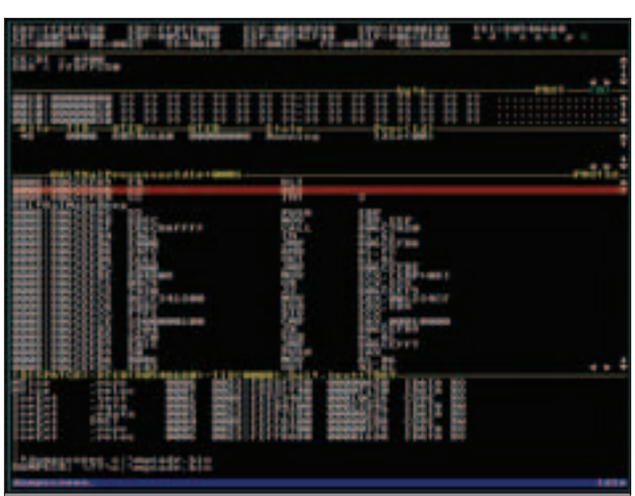
ЭТО ТОЛЬКО НАЧАЛО

Конечно, в такой маленькой статье сложно рассказать обо всех возможностях SoftIce. Поэтому смотри подробности в "Using SoftICE" (в интернете легко найти перевод на русский язык - "SoftICE руководство пользователя") и читай крэкерские сайты!

Самыми известными плагинами для SoftIce являются IceDump и IceExt.



Установка плагина IceExt



Этот скриншот экрана SoftIce сделан с помощью плагина IceExt

СПЕЦ ХАКЕР SMS СЕРВИС

Хочешь фирменный лого на свой сотовый?

Пришли код логотипа (к примеру "1001") на номер **4446**.

Что нового ты хочешь увидеть в SMS-сервисе? Присылай идеи и критику на sms@real.xakep.ru


1073

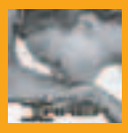

1074

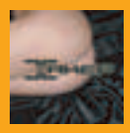

1071


1072

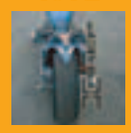

1065


1066


1067


1068


1069


1070


1059


1060


1061


1062


1063


1064


1045


1046


1040


1043

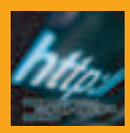

1044


1008


1000


1001


1002



1003


1005


1007

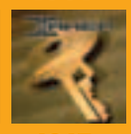

1009


1010

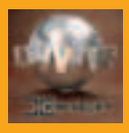

1011


1012


1014


1015

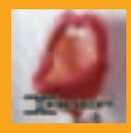

1016


1018


1020


1023


1035


1039

1025

1027

1030

1033

1034

1036

1049

1050

1051

1052

1053

1054

1055

1056

1057

1058

Пришли свой логотип! sms@real.xakep.ru

На диске к журналу есть новый СЮРПРИЗ, но он под паролем! Чтобы узнать пароль, пришли код w0168 на номер 4445.

Хочешь узнать, что значит термин?

Пришли код термина (к примеру "w0001") на номер **4444**.

идентификатор	(код w0008)	транслятор	(код w0092)
скрипт	(код w0009)	верификатор	(код w0093)
интерфейс	(код w0010)	спам	(код w0094)
терминал	(код w0011)	офшор	(код w0095)
библиотека	(код w0012)	крякер	(код w0096)
транзакция	(код w0013)	бета	(код w0097)
архитектура	(код w0014)	скин	(код w0098)
трассировка	(код w0015)	сертификация	(код w0099)
дистрибутив	(код w0016)	аутсорсинг	(код w0100)
утилита	(код w0017)	баннер	(код w0101)
брандмауэр	(код w0018)	локализация	(код w0102)
хост	(код w0019)	тестер	(код w0103)
подсеть	(код w0020)	дамп	(код w0104)
демон	(код w0021)	стек	(код w0105)
эксплойт	(код w0022)	исключение	(код w0106)
хостинг	(код w0023)	мидлет	(код w0107)
сервис пак	(код w0023)	обфускатор	(код w0108)
файрвол	(код w0025)	документация	(код w0109)
брутфорсер	(код w0026)	поток	(код w0110)
тэг	(код w0027)	хэширование	(код w0111)
парсер	(код w0028)	браузер	(код w0113)
инициализация	(код w0029)	инсталятор	(код w0114)
кодировка	(код w0030)	реестр	(код w0115)
визуализация	(код w0038)	аккаунт	(код w0116)
снифер	(код w0040)	домен	(код w0117)
кейлоггер	(код w0041)	девелопер	(код w0118)
троян	(код w0042)	флуд	(код w0119)
отладчик	(код w0043)	пиктограмма	(код w0120)
эмулятор	(код w0044)	архиватор	(код w0121)
хук	(код w0045)	экспозиция	(код w0128)
пиринг	(код w0047)	стробоскоп	(код w0129)
хаб	(код w0048)	бинарник	(код w0130)
фтп	(код w0049)	баг	(код w0131)
маппинг	(код w0050)	шлюз	(код w0132)
роутер	(код w0051)	шелл	(код w0133)
прокси	(код w0052)	блог	(код w0134)
редирект	(код w0053)	бэкап	(код w0135)
слот	(код w0054)	декодирование	(код w0136)
ник	(код w0055)	локалка	(код w0137)
биос	(код w0056)	бэкдор	(код w0138)
оболочка	(код w0057)	хомпага	(код w0139)
ядро	(код w0058)	сессия	(код w0140)
юстировка	(код w0059)	авторизация	(код w0141)
конвертер	(код w0060)	топик	(код w0142)
коаксиал	(код w0061)	профиль	(код w0143)
транспондер	(код w0062)	сегмент	(код w0144)
поляризация	(код w0063)	листинг	(код w0145)
патч	(код w0064)	алиас	(код w0146)
азимут	(код w0065)	свитч	(код w0147)
кодек	(код w0066)	спуфинг	(код w0148)
граббинг	(код w0067)	фрикинг	(код w0149)
мультифид	(код w0068)	крэкинг	(код w0150)
бод	(код w0069)	сиквел	(код w0151)
пиксел	(код w0070)	ретранслятор	(код w0152)
модератор	(код w0071)	коммутатор	(код w0153)
флейм	(код w0072)	аттач	(код w0154)
кряк	(код w0073)	плагин	(код w0155)
варез	(код w0074)	регистр	(код w0156)
сплиттер	(код w0075)	протокол	(код w0076)

Пришли свои термины на номер **4445** в виде **98 termini** (например "98 баг"). Не более 160 символов латиницей или 70 кириллицей.

Можно присылать свои термины

Подробности: www.i-free.ru, (095) 916-7253, (812) 118-4575, support@i-free.ru. Для заказа картинок включи услугу WAP/GPRS-доступа в Интернет (оплачивается согласно твоему тарифному плану). Проверить возможность закачки можно зайдя на wap-сайт http://4446.ru. В случае ошибки уточни настройки в службе поддержки твоего оператора. Стоимость запроса на номер 4444 – $0,30 без учета налогов, на номер 4445 – $0,60 без учета налогов, на номер 4446 – $0,90 без учета налогов, на номер 4449 – $3,00 без учета налогов. В случае ошибочного запроса услуга считается оказанной.

Антон "Hex" Кукоба (xtin@ua.fm)

ОБРАТНАЯ ИНЖЕНЕРИЯ

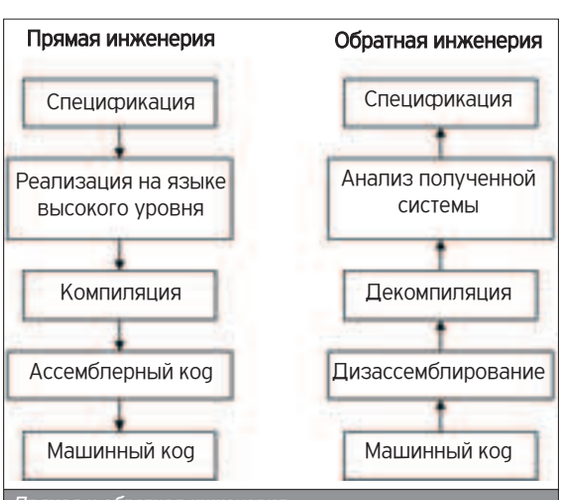
ПОСОБИЕ ПО РЕВЕРСИНГУ ДЛЯ НАЧИНАЮЩЕГО

Для многих реверсинг и крэкинг - слова-синонимы. На самом деле реверсинг - это всего лишь один из методов, которым пользуются крэкеры, да и то не все.
Из мудрых справочников ты узнаешь, что реверсивная инженерия - это "процесс восстановления спецификации из кода". Спецификация - техническое задание, бумажка, выдаваемая программистам, с указаниями о том, как должна выглядеть и работать система. А реверсивная инженерия в полном объеме - это восстановление всех знаний о системе. Например, полноценный реверсинг программного обеспечения подразумевает дизассемблирование, декомпиляцию, анализ функционирования системы и восстановление спецификации.

ДОКОПАТЬСЯ ДО ИСТИНЫ

Согласно широкому взгляду, реверсивная инженерия представляет собой подход к созданию новых систем на основе существующих. Часто ее используют в промышленном шпионаже, и это не слепое спионеривание, а анализ продукта, созданного конкурентами. Также реверсинг применяется для переработки собственных систем, если код программы уже настолько сложен, что проще восстановить всю систему, чем вчитываться в тонны документации. Тот же метод применяется в ре-инженерии (повторной инженерии), при анализе собственного продукта, чтобы определить узкие места реализации. Кто-то использует реверсивную инженерию программного обеспечения для написания антивирусов, кто-то реверсит антивирусы, чтобы писать не видимые для них вирусы. Плоха или хороша реверсивная инженерия, как обычно, зависит от контекста: для родины ты разведчик, для врагов - шпион. И тяга к ней у человека проявляется с раннего детства, когда он разламывает игрушки, чтобы понять, как они сделаны :).

В этой статье мы разберем два важных этапа реверсинга: дизассемблирование и декомпиляцию.



Прямая и обратная инженерия

ИНСТРУМЕНТЫ

Реверсить голыми руками у нас умеют только уникумы, которые, как в фильме "Хакеры", смотрят на шестнадцатеричный дамп и сразу все понимают. Большинству нормальных людей все-таки нужны инструменты. Твой выбор будет зависеть от поставленной задачи, я же приведу список утилит, нужных для реверсинга Win32-программ.

Первый, главный и незаменимый инструмент - это дизассемблер. Тут без вариантов: Interactive Disassembler Pro (дальше IDA Pro), на текущий момент единственный действительно интерактивный дизассемблер, просто незаменимая вещь. В нем проводится 95% работ.

Редактор PE-файлов - любой имеющийся у тебя. Главное, чтобы он также умел выводить список процессов и модулей: PE Tools, Lord PE, PEditor.

Редактор ресурсов PE-файлов: ResHacker или ExeScope. Бывает очень нужен, когда все текстовые строки вынесены из кода в секцию ресурсов или специальную DLL.

Декомпиляторы: Dede, Dj java decompiler, SWF Decompiler и т.д. Если ты занимаешься реверсингом большого продукта, будь готов к тому, что он окажется составленным из кучи EXE, DLL, ActiveX, причем все они будут написаны на разных, самых неожиданных языках.

Различные информационные инструменты: Filemon, Regmon, HDD Serial/USB Monitor, Greatis Windowse, Spy++, OLE/COM object viewer. Очень помогает плагин KANAL к PeID: он позволяет находить криптоалгоритмы, используемые в программе. В общем, нужно иметь как можно больше инструментов, которые предоставляют какую-то неочевидную информацию о программе.

Кажется, это все. Let's get it started!

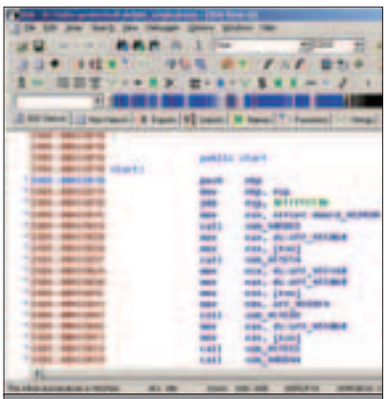
АНАЛИЗ СУЩЕСТВУЮЩЕЙ ИНФОРМАЦИИ

Итак, основная задача реверсера - разобраться в том, как работает чужой код. Чтобы сделать это быстрее, мы должны получить как можно больше сведений об исследуемом продукте, поэтому в первую очередь лезем на посвященный ему сайт. Там наверняка есть информация о технологиях, задействованных в программе. Например, описан способ шифрования, вынесены благодарности каким-нибудь компаниям за библиотеки и т.д. Какой именно протектор использовался, конечно же, не скажут. Однако, опять же, реверсинг - это не борьба с защитой, а самостоятельное исследование кода.

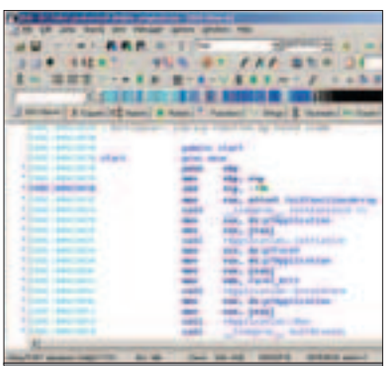
Программисты - люди, как правило, ленивые и рациональные, поэтому они очень любят использовать готовый код. А где обычно берут готовый код? Правильно, на SourceForge и иже с ним. Реверсер обязательно должен ознакомиться с библиотеками, которые использовались в исследуемой части программы, чтобы иметь о каждой из них хотя бы общее представление. Множество платных библиотек можно найти на варезных сайтах или в p2p-сетях. Но если уж так "повезло", программисты написали все сами и ничьим кодом не пользовались или код был куплен и его невозможно добыть из интернета на халяву, то есть другой путь - посмотреть, как реализованы аналогичные программы. Как ни причудлива человеческая фантазия, но схожие задачи люди обычно решают одинаково. Возьмем, к примеру, два менеджера закачек - reget и flashget. Ну и что? Разницу между ними можно найти только на высоком уровне, а внизу одни и те же API WinInet. А если тебе нужно разобраться с продуктом, написанным на VC, и у него есть аналог на Delphi. Ковыряй последний, потому что исследовать Delphi всегда проще.

Часто бывает полезно посмотреть предыдущие версии программы: ты можешь найти или отладочную (debug) версию, или незащищенную. В конце концов, в начальных версиях код проще.

Реверсинг невозможен без знания того, как компилятор реализует те или



Так выглядит код до применения сигнатур



А так выглядит код после применения сигнатур

иные конструкции языка высокого уровня. Поэтому если попадается программа, написанная на каком-нибудь малораспространенном компиляторе, сначала нужно изучить компилятор, сделать сигнатуры для IDA с его библиотек. Сигнатуры - это шаблоны кода функций, позволяющие IDA находить и именовать библиотечные функции внутри файлов. Поэтому стоит сделать сигнатуры и для библиотек, которые используются в исследуемом продукте, чтобы распознать их части в коде. Это сократит время анализа в несколько раз, поскольку не придется анализировать библиотечные функции. Для создания собственных сигнатур есть набор инструментов FLAIR, идущий с IDA. Только после этого стоит переходить к следующему этапу.

ДИЗАССЕМБЛИРОВАНИЕ

ПОИСК ИНФОРМАЦИИ В КОДЕ

■ Первым делом определим компилятор. Самый простой способ сделать это - применить все сигнатуры. Тогда по функциям из той или иной библиотеки все сразу станет ясно. Можно просто поискать оставленные компилятором следы вроде строк "Borland" и "Microsoft".

Основные сведения, которые могут пригодиться при исследовании, лежат прямо в файле программы. Это отладочная информация, копирайты, строки, обработка ошибок и т.д. Файл с отладочной информацией часто содержит имена функций, типы их параметров, имена структур и еще много всего. Поэтому всегда нужно искать отладочные варианты. Копирайты поведают о версии использованных библиотек и их именах. Программы на Delphi - вообще отдельная тема: в файлах остаются имена и классов, и модулей. Разные строки в коде, оставленные программистом для ведения лога и анализа ошибок - это просто кладезь информации! Особенно если программист делает вывод ошибки типа "Error at function MYFUNCTION () pointer to XXXXXX object == NULL". Сразу понятно, какая это функция и какие параметры что в нее передают.

Если в программе есть логирование, нужно обязательно найти и включить его. Программисты используют лог-файлы, чтобы видеть этапы работы программы, параметры ее запуска и результаты. В программах, в которых задействованы COM-объекты, дополнительной информацией являются TLB-файлы. По ним ты найдешь практически всю функциональность COM-объекта. Если удается найти PDB-файл, задача реверсинга упрощается в несколько раз, так как PDB-файл может содержать информацию об именах функций и типах параметров.

Если программа многоязычная или использует вместо строк/сообщений их коды, обязательно сделай таблицу для перекодирования. Тебе нужно будет найти таблицу соответствия этих кодов к сообщениям об ошибках. В IDA это можно организовать в enum. Такие таблицы обычно хранятся в ресурсах или вынесены в отдельный файл.

АНАЛИЗ И СИСТЕМАТИЗАЦИЯ

■ Общий подход. Перед реверсером всегда стоит дилемма: сверху или снизу? Подход сверху означает, что анализ начинается от обработчика какого-то события GUI, то есть реверсер понятия не имеет, как оно работает. Подход снизу означает анализ от одной из API. Такая проблема особенно актуальна, если исследуется какая-нибудь сложная система.

Пример подхода сверху. Например, есть программа, которая работает с нестандартной базой данных недокументированного формата. Данные в базе шифруются, и их можно просмотреть в открытом виде только где-то на среднем уровне. На верхнем уровне находятся просто какие-то объекты, каждый из которых имеет неизвестную структуру, и неведомо, какие параметры могут передаваться в его методы. Внизу лежат API-открытия, чтения и записи в файл. Посередине идут слои преобразования шифрования/дешифрования данных. Нужно научиться читать такие базы данных.

Для начала проведем разделение на уровни, чтобы знать, где мы находимся. Первый анализ должен быть поверхностным, сортировочного типа: "Тут у нас открытие, тут общение с GUI, тут шифрование, здесь реализация SQL".

Дальнейший анализ следует начинать с простых коротких операций, чтобы посмотреть, как работает GUI и как выглядит вся цепочка вызовов. Очень сильно помогает восстановление структур и классов. Анализируй функциональности проекта по мере их сложности, накапливая с каждым разом как можно больше информации о классах, структурах и типах данных. Анализ крупных проектов всегда связан с анализом взаимодействия классов, поэтому если в ООП не разбираешься, хочешь не хочешь, придется выучить.

Пример подхода снизу. Есть программа, работающая с драйвером, в котором и реализована функциональность. Нужно узнать, как можно работать с драйвером без программы. Или, например, нужно декомпилировать алгоритм работы драйвера для реализации в ring 3. Тут сразу видно, что есть точка старта внизу. Эта точка - обмен данными между драйвером и программой, то есть функция DeviceIoControl(). Анализ начинается с перехвата управляющей программы. Смотрим, какие IOCTL-коды нужны каждой из функциональностей. После этого дизассемблируем драйвер и исследуем код, обрабатывающий каждый из нужных IOCTL-кодов. Далее снова анализируем программу для определения параметров, передаваемых в драйвер, и т.д.

Для получения наилучшего эффекта оба подхода нужно комбинировать, выбирая тот или иной в зависимости от текущей подзадачи.

ОФОРМЛЕНИЕ DISASM-ЛИСТИНГА

■ Внутренней информации часто бывает мало. Функции низкого уровня могут быть чисто алгоритмического типа, без вызовов API. А может попасться release-вариант, где все вылизали и не оставили вообще никаких намеков. В таком случае придется придумывать свой подход для именования переменных/функций/структур. Единого стандарта тут нет, и я сам еще не до конца для себя решил, как их называть. Пока при именовании я ввожу подфункции и надфункции. С функциями, найденными по ходу реверсинга, я поступаю по следующему принципу:

❶. Если встречаю функцию, которая полностью реализует какую-то функциональность, я даю ей имя этой функциональности. Пример: FormatDiskC, DoDecrypt и т.д.

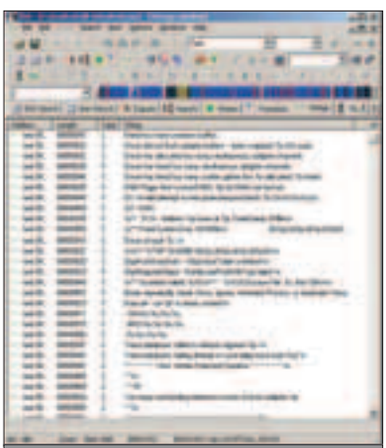
❼. Если внутри именованной функции встречаю вызов неименованной функции, которая выполняет конкретно реализацию функциональности, то даю ей имя с подчеркиванием в конце. Это подфункция. Пример: FormatDiskC_, DoDecrypt_. »

Если реверсинг для тебя только начинается, изучи все возможности IDA. В первую очередь иди на http://idapro.ru/faq.

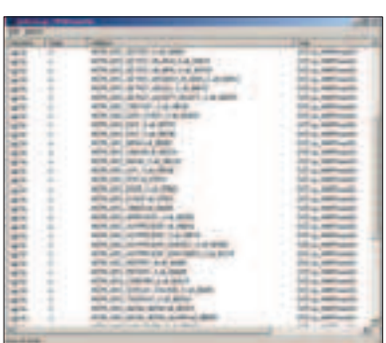
Информацию о декомпиляции можно найти в статьях Кристины Сифуентес по адресу www.cs.uq.edu.au/~cristina/pubs.html.

Горячо рекомендую прочесть книгу Криса Касперски "Техника и философия хакерских атак". Удели особое внимание главам не о взломе, а об анализе кода. Крис детально описал, как можно распознавать функции, классы, сложные механизмы ООП, как анализировать стек и многое другое.

Используя граф вызовов, намного проще выделить уровни абстракции, используемые в программе

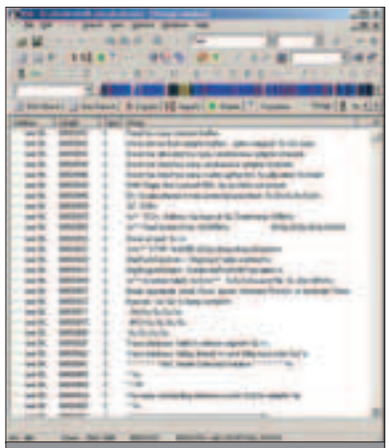


Самый удобный способ искать следы отладочной информации в списке строк



Перекрестные ссылки позволяют найти все места вызов функций или обращения к переменным

ных в именах есть подчеркивание. Еще при именовании аргументов я стараюсь выделять входные и выходные параметры, дописывая им суффиксы _in или _out. Обязательно ставлю префикс "p", если передается указатель. Называть структуры и классы сложно, имена часто приходится придумывать в зависимости от данных, которые они хранят в себе, но большей частью приходится проявлять чудеса воображения. Подход типа Struct1, Struct2... при появлении десяти и более структур запутывает напрочь. Иногда мне хочется, подобно астрономам, использовать кодовые имена. Но это неэффективно, потому что имя функции/структуры должно нести информацию о том, что она содержит и что делает.

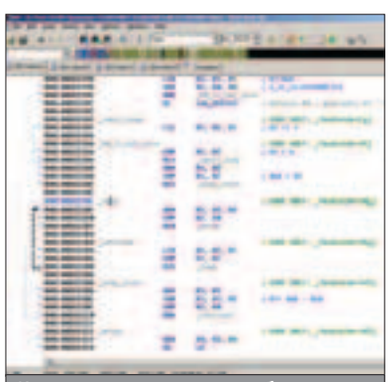


Самый удобный способ искать следы отладочной информации в списке строк

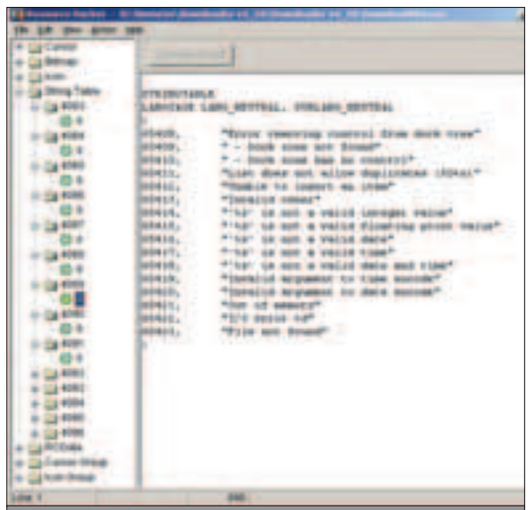
При анализе кода помни, что программисты - тоже люди, они тоже пытаются писать красивый и понятный код. И если где-то выделяется память, это значит, что где-то она очищается. Если создается объект, значит, он где-то используется и где-то уничтожается. Используя информацию о жизненном цикле, можно сначала отследить и отреверсить какой-то один несложный объект/структуру, а далее, основываясь на этом объекте, реверсить более сложные объекты, которые взаимодействуют с ним. Это намного эффективнее, чем охватывать все целиком. Анализ любой функции тоже производится с учетом известных данных.

Если в функции есть вызовы API, нужно поименовать все переменные и аргументы, исходя из прототипов функций. Старайся подписывать все, что только можешь: регистры, переменные, функции, точки переходов, пиши комментарии. Обязательно находи и именуй циклы: переменную-счетчик, точку возврата и точку выхода. Я обычно даю имена @Loop, @Break, @Continue. Очень сильно помогает именование точек переходов - мест, куда совершается переход при ошибке и при нормальном выполнении кода. Я называю их @Ok и @Error.

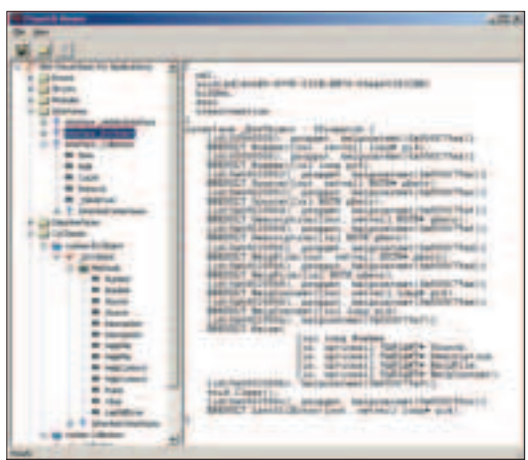
Если в коде есть вызовы виртуальных функций (регистровые вызовы), обязательно проверь стек - IDA не умеет анализировать его после вызова виртуальной функции.



Код лучше записывать, чтобы не запутаться



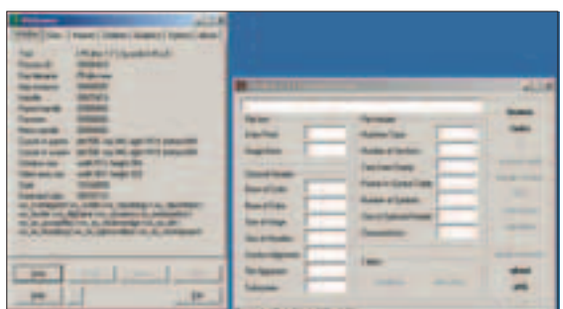
Те самые таблицы строк, которые вынесены в секцию ресурсов



COM/OLE Object viewer - незаменимый источник информации о COM-объектах

ЧТО ДЕЛАТЬ ДАЛЬШЕ

Когда уровни выделены и есть представление о классах и нужных API, уже можно извлекать код из программы и пытаться использовать его. Есть два подхода: декомпиляция вручную и переассемблирование ассемблерного листинга. Первый способ более трудоемкий, но при его использовании ты чист перед законом, поскольку не используешь чужой код, а пишешь свой. Второй способ дает очень быстрый результат. Ты сразу получаешь obj, который и линкуешь к программе. Этот кусок кода в твоей программе будет совпадать с оригиналом байт в байт, и для гневного заявления "Это плагиат!" будет много оснований.



Windows предоставляет полную информацию об окне

GPcH (admin@dotfix.net)

ДЕКОМПИЛЯТОРЫ

ОБЗОР СРЕДСТВ ДЛЯ ВОССТАНОВЛЕНИЯ ИСХОДНОГО КОДА ПРОГРАММ

Сложность кода, генерируемого компиляторами языков высокого уровня, использование виртуальных машин, новые хитрые форматы хранения данных VCL, Visual Basic, Java, FoxPro, .NET, наконец – все это ведет к тому, что обычных дизассемблеров native-кода перестало хватать для того, чтобы хоть как-то разобраться в работе программы. Реверсеру потребовались новые инструменты, которые смогли бы раскопать в миллионах байт исследуемой программы до боли знакомые ему строки языка, на котором она была написана. Вдруг выясняется, что стали необходимы декомпиляторы, и в этом обзоре пойдет речь именно о них.

Восстановление исходного кода программы во многом стало возможным из-за массы лишней информации об оригинальном коде в откомпилированном файле, а также из-за однотипности структур и операторов, используемых в языках высокого уровня. Декомпиляторы анализируют избыточную информацию, и, зная, как компилер того или иного языка любит представлять некоторые структуры, пытаются создать на базе сырого кода программы некоторое подобие исходника. У некоторых это даже неплохо получается. Лучше всего декомпилируются программы, которые выполняются не напрямую процессором, а виртуальной машиной (написанные на Java, Visual Basic, FoxPro, .NET и т.п.). Причина этого кроется в том, что инструкции виртуальных машин, как правило, высокоуровневые и объединяют сразу несколько машинных команд. Иными словами, это те же операторы и ключевые слова языка программирования, только записанные немного иначе, с некоторой оптимизацией инструкций языка разработки.

Вторыми по простоте декомпиляции идут программы, написанные на таких языках программирования, как Delphi и C++ Builder. Несмотря на то, что данные языки не имеют дела с виртуальными машинами (кроме разве что платформы .NET), а компилируют программы в нормальный native-код, они любят использовать собственные стандартные библиотеки вроде VCL и оставляют в EXE-файлах много лишней информации, используя которую можно также вполне успешно восстановить исходный код.

Что же касается новомодной среды разработки .NET, то благодаря действительно невероятному количеству избыточной информации, хранящейся в exe-файлах компилируемых программ, можно чуть ли не с 100% точностью восстановить исходный код, написанный кодером. Разработчики сего проекта вовремя опомнились и начали выдавать так называемые обфускаторы кода, способные вычищать лишнюю информацию из программ, чтобы хоть как-то препятствовать декомпиляции, однако об этом пока мало кто знает, а вот сама среда разработки используется уже вовсю. К чему это ведет? К росту количества кейгенов, конечно :).

Что ж, давай посмотрим, какие декомпилеры уже успели появиться на свет.

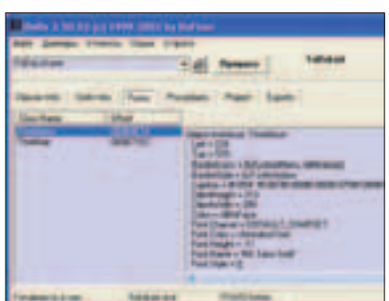
DEDE BY DAFIXER

Самый знаменитый декомпилятор Delphi. Работает с программами, скомпилированным любыми версиями Delphi, кроме восьмой (так как она создает .NET-код). Парень с ником DaFixer, написавший его, действительно молодец. Мало того, что он создал такой полезный инструмент, он еще и не пожалел для народа исходного кода одной из его старых версий! Подобный сорец может очень пригодиться тому, кто изучает код, генерируемый борландовскими монстрами.

Что же делает эта программа? Многое - разве что яичницу не жарит. Для начала предоставляет тебе все формы в оригинальном виде и дает возможность полазить по процедурам и функциям, имеющимся в программе. Помимо дизассемблерного листинга этих функций, программа пытается распознать стандартные операторы и типы Delphi и добавляет их в комментарии к ассемблерному коду. Особенно радует распознавание блоков типа:

```
try
..
except
..
end;
```

Кроме того, эта программа умеет создавать исходник, который можно открыть в Delphi. А если ты используешь в качестве дизассемблера WDasm (о ужас!), то тебе точно будет полезна возможность Dede экспортировать данные в понятный ему формат. В общем, весьма позитивный декомпилер. Главный минус Dede в том, что он не умеет выдергивать из exe'шника компоненты, используемые в программе. Из-за этого в сгенерированном исходнике присутствует множество нераспознанных типов данных. Но это не большая беда, так как заставить полученный код работать все равно не удастся, как ни крути. А исследовать его – пожалуйста.



Dede by DaFixer

Программу вместе с сорцами бери на www.wasm.ru.

SOURCERESCUER

Еще один декомпилер Delphi, но попроще. Умеет восстанавливать формы и генерировать заголовки pas-файлов. Главное отличие от Dede – мгновенная работа и более эргономичный интерфейс. Что ж, одной программе дано иметь крутой интерфейс, другой - крутые возможности, тут ничего не поделаешь. Из главных особенностей декомпилятора можно выделить то, что он может создавать шаблон исходника не только в формате Delphi, но и в формате Builder'а.

SourceRescuer

Фишка простая (после компиляции программы из Delphi и C++ Builder мало чем отличаются), но полезная. Распространяется в двух видах: GUI и консольном. Насколько я понял, требует регистрации.

Взять можно отсюда: www.ems-hitech.com.

REC BY GIAMPIERO CAPRINO

■ Полное название - the Reverse Engineering Compiler. Программа, которая с успехом преобразует в исходный код на C любой попавшийся ей на глаза исполняемый файл. Конечно, сорец получается мало похожим на оригинал, но разобраться в нем будет несложно. REC определяет функции исследуемого файла, основные блоки и структуры языка, такие как if, for, switch, вызовы API, и делает вообще все возможное, чтобы код стал понятным C-программисту. Однако не удивляйся, если увидишь в исходнике что-нибудь вроде:

```
eax++;
for(ecx=1000;ecx!=0;ecx--)
{
    ebx = ebx&ecx;
}
```

Это нормально. И, я думаю, лучше уж исследовать это, чем:

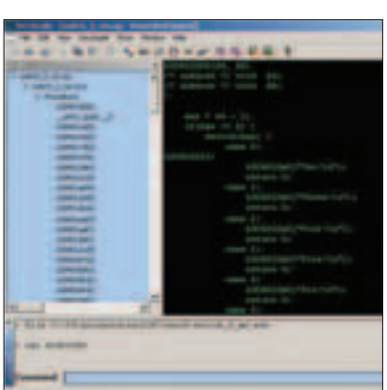
```
00400000:   inc eax
00400001:   mov ecx, 1000
00400006:   and ebx, ecx
00400008:   loop 00400006
```

Хотя на вкус и цвет... Некоторые знают ассемблер даже лучше, чем русский матерный. Из приятных мелочей: кроссплатформенность (кроме Windows, декомпилятор считает родными Linux, Mac OS X и Solaris); поддержка нескольких форматов исполняемых файлов (REC не ограничился одними PE и COFF, он умеет анализировать и ELF, и AOUT, и даже Playstation PS-X).

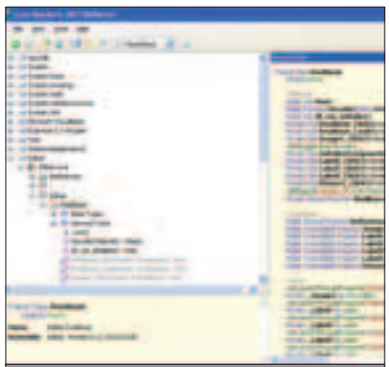
Любому C-ориентированному реверсеру must have. Качать тут: www.backerstreet.com/rec/recdload.htm.

.NET REFLECTOR BY LUTZ ROEDER

■ Динамично развивающийся бесплатный и мощный декомпилятор



REC by Giampiero Caprino



.NET Reflector by Lutz Roeder

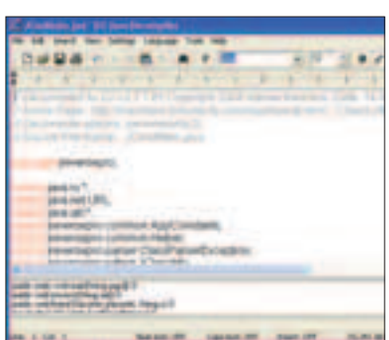
.NET-сборок. Практически любую программу, собранную по новой мелкомягкой технологии, он без проблем представит в виде полного исходника со всем деревом наследования классов. Подсветка синтаксиса, гиперссылки на объекты классов, удобный и приятный интерфейс - все это говорит лишь об одном: в декомпилировании .NET-сборок этому инструменту нет равных. Если ты исследуешь .NET-программы или просто хочешь взглянуть на свою разработку глазами реверсера, быстрее качай это чудо. Must have однозначно. Вместе с самим декомпилером можно скачать удобный инструментик для выдирания ресурсов из .NET-сборок.

За программами и их регулярными обновлениями лезь на www.aisto.com/roeder/dotnet.

DJ JAVA DECOMPILER

■ А это уже декомпилер Java-классов. Довольно прост и удобен. Открываешь в нем класс и сразу же видишь его исходник. Имеется неплохая подсветка синтаксиса, поиск и настройки. Также есть браузер классов и объектов. В общем, очень неплохой и интересный декомпилятор. Жаль только, что exe-файлы, написанные на Java, не декомпилирует.

Лежит здесь: http://members.fortunecity.com/neshkov/dj.html.



DJ Java Decompiler

ReFox by Jan Brebera

REFOX BY JAN BREBERA

■ Со слов людей, использующих его, это довольно мощный декомпилятор программ, созданных на FoxPro. Причем какая "Фокса" - не критично. Декомпилер берет и DOS-, и Windows-версии, причем не только стандартные, но и закриптованные модули. Даже если код был скомпилен под Macintosh, ReFox возьмется изучать его и даже портирует выходной вариант по DOS/Windows. Короче, если ты долго разрабатывал экономическую программу для своего отдела и ненароком потерял ее сорцы, обращайся к ReFox - он не оставит тебя в беде.

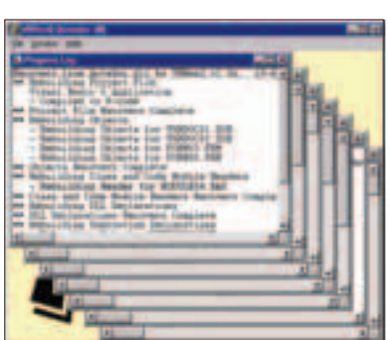
ДЕКОМПИЛЯТОРЫ VISUAL BASIC

■ Взлом приложений, созданных с помощью Visual Basic и скомпилированных в p-код, был мукой для взломщиков до тех пор, пока не стали появляться нормальные декомпиляторы этого языка. Сейчас их уже очень много, практически на любой вкус. Они делятся на три типа: декомпиляторы форм, редакторы форм и декомпиляторы p-кода. Также есть программы, сочетающие в себе сразу несколько возможностей, о них мы тоже не забудем.

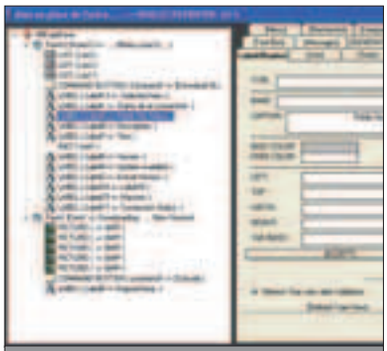
VBREZQ

■ Один из самых стабильных декомпиляторов форм. Хотя, кроме стабильности, он ничем больше и не при- »

Лучше всего декомпилируются программы, которые выполняются не напрямую процессором, а виртуальной машиной.



VBRezQ


VB Editor by HEXMAN

мечателен ;). Объявления API-функций делает без параметров, от чего пользы мало. Код не декомпилирует вообще. Имеет довольно подробную документацию и просит за свое использование немало зеленых президентов. В общем, его, наверное, стоит использовать, если тебе не жалко денег и нужно декомпилировать только элементы интерфейса.

Найти можно тут: www.vbrezq.com.

VB EDITOR BY HEXMAN

■ На этот раз абсолютно бесплатный редактор форм и лежащих на них объектов. Если ты занимаешься русификацией программ, но не можешь ничего поделать с теми, что написаны на VB (Restorator тут бессилен), смело бери в руки VB Editor. На основе сделанных изменений редактор может сгенерировать форму, поэтому он вполне достоин статуса достойной альтернативы упомянутому выше VBRezQ. Забавно, что из двух поддерживаемых языков интерфейса, английского и французского, по умолчанию грузится именно французский. Однако вряд ли у тебя возникнут большие сложности с ориентацией в программе даже в процессе сношений с меню на непривычном для тебя английском.

Качай отсюда: www.multimania.com/hexman.

VBREFORMER BY SYLVAIN BRUYERE

■ Еще одна работа французских программистов. Эта программа явно помощнее предыдущей. Помимо просмотра и изменения форм, предоставляет также возможность вы-

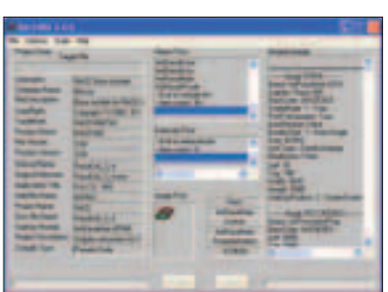

VBReFormer by Sylvain Bruyere

драть из exe'шника всякие изображения, которые обычно лежат в frx-файлах. Trial-версия позволяет только просматривать результаты. При этом не только запрещается сохранять результат, но и постоянно обнуляется буфер обмена, дабы исключить и возможность копирования. Отсюда, прежде чем запускать программулину, убедись, что в буфере нет нужных, но еще не сохраненных данных. Помимо всего прочего, VBReFormer предоставляет для обзора некоторые данные из заголовка exe-файла, потому адрес точки входа в программу и Image Base можно узнать прямо не отходя от кассы. Еще один плюс софтины - она умеет распознавать используемые в декомпилируемой программе ActiveX файлы и позволяет просмотреть все их свойства и методы. Жаль только, что не использует эту информацию при генерации форм - там все ActiveX'ы выглядят немного убого, без свойств и присвоенных им данных. При некотором желании эта тулза может обрыскать весь твой жесткий диск в поисках программ, написанных на VB. Зачем это нужно, не знаю. Наверное, для тестирования возможностей на разных exe'шниках.

Если программа тебе приглянулась, то trial-версию можешь взять здесь: www.decompiler-vb.tk. Думаю, ты сумеешь поладить с ней.

RACEEX6

■ А этот монстр пытается декомпилить и формы, и p-код, но представляет на экран всю информацию в таком убогом виде, что не поймешь, какие


RaceEx6

данные к чему относятся. Так же, как и предыдущая софтина, умеет дергать графику из программ, написанных на VB. Когда мучает p-код, декомпилит только методы - с передаваемыми в них параметрами туго. В общем, если довести интерфейс до ума, получилась бы вполне нормальная штука, но автору, судя по всему, этим заниматься лень.

Брать можешь отсюда: www.raceco.us.

EXDEC BY JOSEPHCO

■ Наверное, самый известный в среде крэкеров декомпилятор p-кода. Как говорится, старенький, но рабочий. Возможность у программы всего одна - декомпилить p-code в том ви-


exdec by josephco

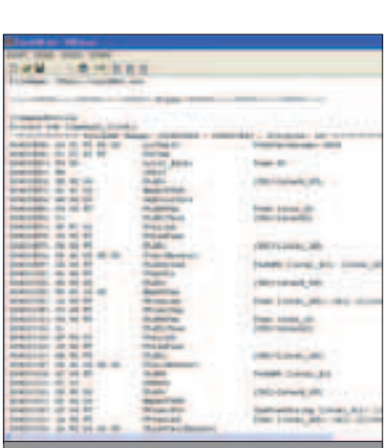
де, в каком он есть. То есть того кода, который писал программист на VB, ты не увидишь, а вот то, что сгенерировал компилятор - да, причем в довольно читабельном для профессионала виде. В комплекте с программой идут примеры и небольшой учебник. Вероятно, он поможет тебе хотя бы немного понять, что означает вся эта декомпилированная муть и как читать ее, поэтому, если ты разбираешься с p-кодом впервые, очень советую обратить внимание на этот tutorial.

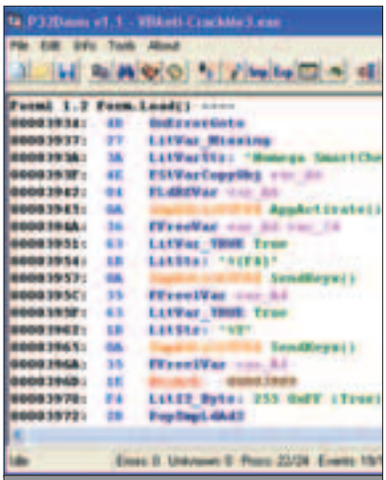
Найти родной сайт программы поможет только Google. Сам дистрибутив же можешь взять с www.wasm.ru или нашего диска.

VBPARSER

■ Почти полный аналог exdec, только написанный китайскими разработчиками. Результат своей работы не только выводит на экран, но и сохраняет в файле ParseVB.txt. На случай ее падения (а такое частенько случается) этот файл здорово пригодится.

Бери ее с диска - судя по всему, она не имеет родного сайта.


VBParser



p32Dasm by DARKER

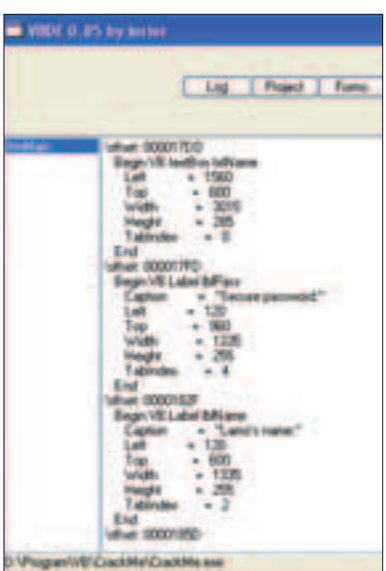
P32DASM BY DARKER

Лучшая альтернатива exdec'у и VBParser'у. В отличие от двух предыдущих, написана на VB (exdec и VBParser на C++) и достаточно неплохо развивается в последнее время. Имеет подсветку синтаксиса, калькулятор адресов, умеет декомпилировать с определенного смещения в файле, а также, подобно дизассемблерам, способна представить все строки и функции, используемые в программе, в удобном списке с возможностью мгновенно перейти на интересующую тебя позицию в листинге. Главный недостаток - нестабильность работы и медленная скорость, в остальном же программа стоит того, чтобы не только посмотреть ее, но и даже записать в джентльменский набор. Распространяется бесплатно.

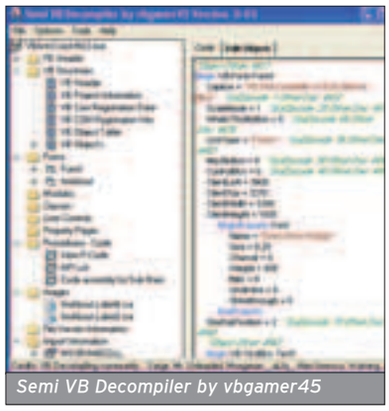
Брать отсюда: http://llamellik.webzdarma.cz/forumb/index.php.

VBDE BY IORIOR

Довольно неплохой декомпилятор, причем бесплатный. Декомпилит формы (правда, без ActiveX). Выдает адреса на все процедуры, а если это возможно, то выводит не только адрес процедуры, но и ее имя, что значительно упрощает анализ. Пытается декомпилировать native-код, однако в большинстве случаев, кроме операторов сложения, вычитания и вывода MessageBox'а, ничего декомпильнуть не может. Несмотря на все эти ужасы, этот зверь довольно удобный и стабильный, поэтому рекомендую всегда держать его под рукой.



VBDE by iorior



Semi VB Decompiler by vbgamer45

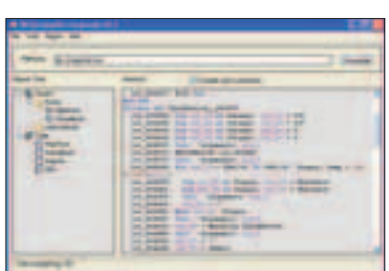
Сайта у программы нет - бери ее с диска.

SEMI VB DECOMPILER BY VBGAMER45

Довольно интересный проект. До недавнего времени распространялся в исходниках на VB, а теперь стал коммерческим. Несмотря на некоторую глючность, бесплатная версия умеет довольно многое. Декомпилятся формы, содержащаяся в них графика и названия процедур. Также определяются используемые в программе API-функции. Помимо этого, предоставляется информация из заголовков PE. Есть даже попытки декомпилировать p-код. В общем, создается довольно приятное впечатление. Огромный респект автору за сорцы. Думаю, многим будет полезно заглянуть в них, чтобы понять, как происходит анализ кода VB. Честности ради скажу, что исходники написаны жутко, и порой, когда смотришь на некоторые участки кода, хочется плакать, хотя код этот и выполняет заложенные в него функции. Но, как говорится, дареному коню в зубы не смотрят. Поэтому быстрее беги на http://pscode.com/vb/scripts/ShowCode.asp?txtCodeId=55935&lngWId=1, пока сорцы еще там, а если хочешь, купи у автора новую версию - поддержи парня.

VB DECOMPILER BY GPCH

А вот мы и дошли до моего собственного декомпилера VB. Я пытался внести в него как можно больше возможностей, при этом не загромождая интерфейс тем, что никогда не пригодится. В итоге имеем: декомпилятор форм с поддержкой ActiveX'ов, лежащих на них (при этом декомпилятся только общие для всех ActiveX'ов свойства), декомпилятор p-кода (причем, если в бесплатной Lite-версии он напоминает p32Dasm'овский, то в Pro-версии программа пытается перевести p-код в исходникоподобный читабельный вид, что часто помогает быстрее восстановить нужные куски кода) и декомпилятор ссылок на API (при этом они записываются уже в объявленном виде со всем списком параметров). Подсветка синтаксиса тоже есть. Также для каждого модуля с кодом есть свой список встречающихся там символьных строк с возможностью мгновенно перейти на участок кода их использования. Присутствует также поиск, помогающий найти нужный код в активном окне. Результат работы можно сохранять, причем вместе с кодом и формами сохраняются и frx-файлы с графикой и корректно прописываются ссылки на эти графические объекты в формах. В общем, если тебе не нужен детальный декомпиль p-code'а, вполне можешь ограничиться бесплатной Lite-версией. Если все же нужна Pro, пиши мне на мыло - договоримся о цене. Кстати, программа еще не выросла до стадии разработки и имеет некоторые баги (все-таки, на момент написания статьи, версия 0.2). И еще: декомпилер предназначен только для восстановления твоих исходников, если вдруг ты их потерял и у тебя есть только EXE. Если же ты роешься в чужой программе, помни: вся ответственность за это лежит на тебе, поэтому не забывай читать лицензионное соглашение, прежде чем лезть в чужой код :).



VB Decompiler by GPcH

Lite-версию можешь скачать отсюда: http://vbdecompiler.dotfix.net.

Здесь же лежат примеры работы Pro-версии.

ЗАКЛЮЧЕНИЕ

Как видишь, дефицита декомпиляторов не наблюдается. При этом очень хорошо чувствуется разница между профессиональными и любительскими разработками, она велика как в качестве, так и в цене. Несмотря на это, многие смогут обойтись бесплатными разработками. Если же ты исследователь защит (не по душе мне слово "крэкер", а более общее и правильное "реверсер" мне ласкает слух), то, думаю, для тебя не составит труда сделать даже коммерческие программы бесплатными для себя :). Главное - не переусердствовать. Не забывай, что закон существует, и, как ни странно, он един для всех. Удачи!

Крис Касперски aka мыщъх

ТЕХНИКА ОТЛАДКИ

КАК ПРАВИЛЬНО ОТЛАЖИВАТЬ ПРОГРАММЫ БЕЗ ИСХОДНЫХ КОДОВ

Практически все знают, что программы взламываются отладчиком, но не все знают, как именно. На самом деле ничего сложного в этом нет - достаточно выучить несколько простых приемов работы с ним, и уже можно начинать ломать.

ВВЕДЕНИЕ В ОТЛАДКУ

Дебаггер - невероятно мощный инструмент взломщика, однако к нему нужен свой подход. Большинство начинающих хакеров начинают отлаживать программу с точки входа и в итоге умирают в цикле выборки сообщений. Пошаговое исполнение программы (также называемое трассировкой) - слишком трудоемкий и крайне неэффективный процесс. Событийно-ориентированные (то есть практически все виндовые) приложения так не отлаживаются. Допустим, мы трассируем MFC-программу: доходим до вызова AfxWinMain и оказываемся глубоко внутри MFC42.DLL, откуда и должен вызываться весь пользовательский код, однако прежде чем трассировка доберется до него, мы успеем состариться!

Но отлаживать программу целиком совершенно не обязательно! Опытные хакеры трассируют только отдельные части защитного кода. Как мы найдем их в миллионах машинных инструкций исполняемого файла? Существует множество методик: точки останова, раскрутка стека, перекрестные ссылки, условная трассировка, прямой поиск паролей/серийных номеров в памяти и т.д. Расскажем обо всем этом поподробнее.

Испытания мы будем проводить над программой Drive LED от компании O&O Software, ее 30-дневную демонстрационную версию можно скачать с сайта www.oo-software.com/en/download/index.shtml.

ДИЗАССЕМБЛЕР И ОТЛАДЧИК В ОДНОЙ УПРЯЖКЕ

Дизассемблер содержится в каждом отладчике (мы же не собираемся отлаживать программу непосредственно в машинном коде, верно?), но те дизассемблеры, что находятся внутри SoftIce или OllyDbg, слишком примитивны. ИДА (IDA Pro) намного мощнее. Она автоматически распознает имена библиотечных функций, определяет типы локальных переменных и делает множество других полезных вещей, в частности, позволяет комментировать листинг и назначать символьные метки для инструкций и данных. Исследовать защищенные программы с ее помощью - настоящее удовольствие. Однако без дебаггера все равно никак. Вызовы типа call [ebx+64h] в дизасм-листинге приводят хакеров в бешенство, особенно если функция вызывается все время с разным EBX. На выяснение значения этого регистра в дизассемблере можно ухлопать целый день, а в отладчике просто "подсмотрел" его - и все!

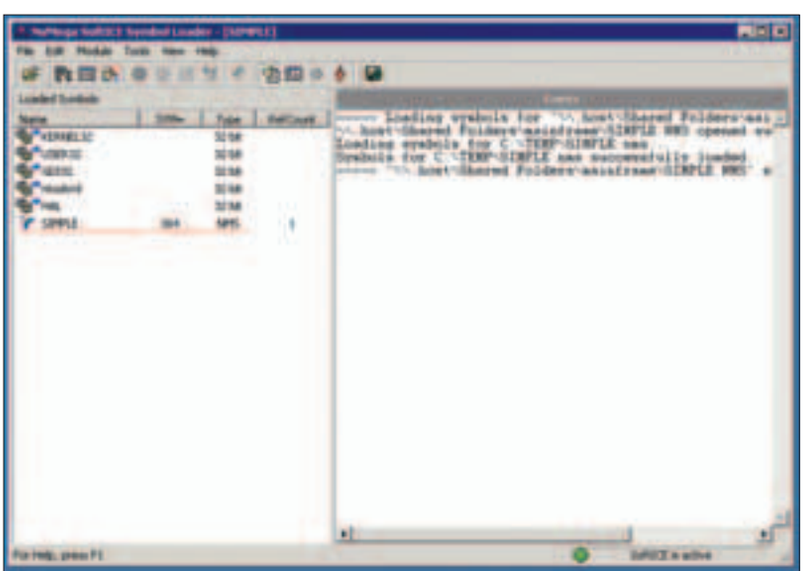
Или вот вызывается что-то по адресу 77E92B8D, лежащему где-то внутри операционной системы (при дизассемблировании дампов памяти такие адреса встречаются сплошь и рядом). В отладчике достаточно просто дать команду "u 77E92B8D", и мы тут же увидим, что это CreateFileA.

Бессмысленно спорить, кто круче: отладчик или дизассемблер. Эти инструменты взаимно дополняют друг друга. Реконструкцию алгоритмов лучше поручить дизассемблеру, а все непонятные места уточнять в отладчике.

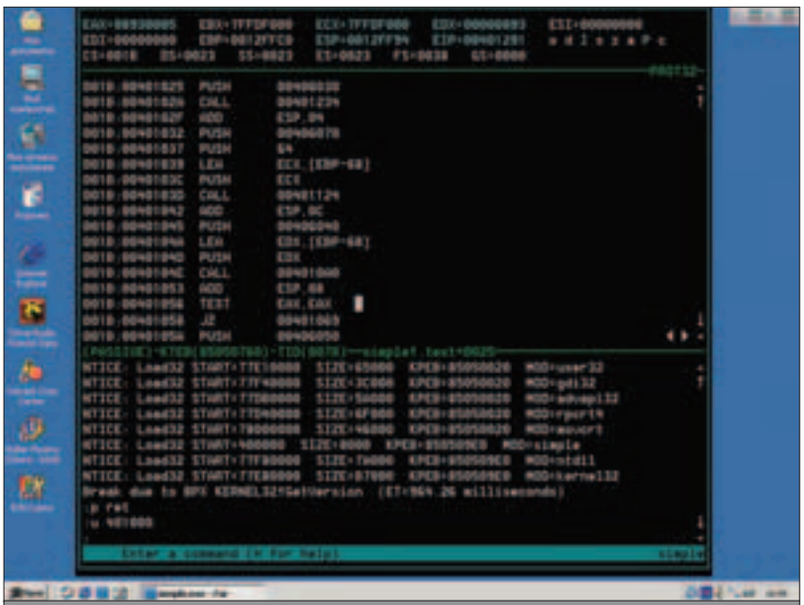
Загрузка символов в дебаггер осуществляется довольно неочевидным образом, на котором спотыкаются многие начинающие. Последовательность действий должна быть следующей.

Сначала исследуемый файл пропускаем через ИДУ. Затем в меню File выбираем пункт Produce output file-> Produce MAP file (причем имя MAP-файла должно совпадать с именем самого дизассемблируемого файла). В появившемся диалоговом окне взводим все три галочки: Segmentation information (информация о сегментах), Autogenerated names (автогенерируемые имена) и Demangle names (размагленные име-

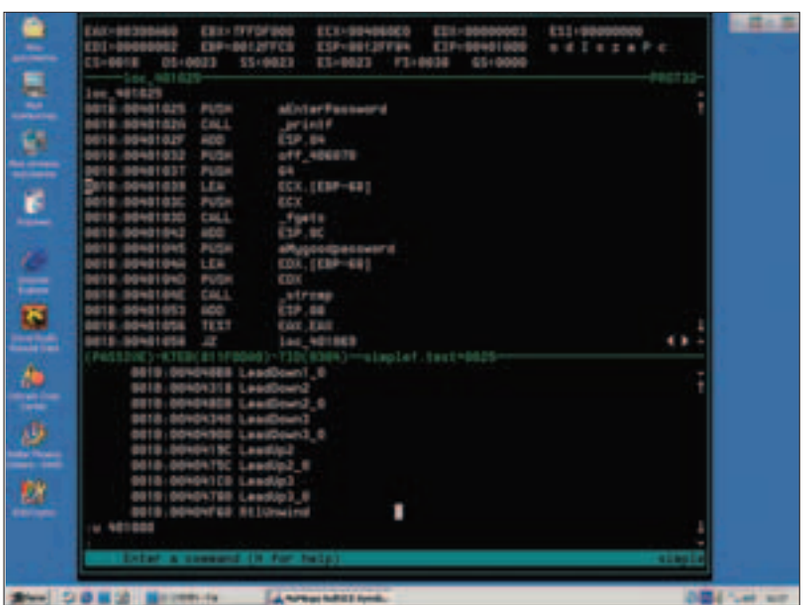
Бессмысленно спорить, кто круче: отладчик или дизассемблер. Эти инструменты взаимно дополняют друг друга.



Загрузка символьной информации в loader32



Отладка файла без символьной информации



Отладка файла с символьной информацией, сгенерированной IDA Pro

на). Полученный MAP-файл скармливаем утилите idasym (ее можно скачать с сайта www.idapro.com) и конвертируем в sym-формат. Далее с помощью утилиты nmsym, входящей в комплект поставки SoftIce, sym-файл преобразуем в nms. Уф! Половина работы сделана! Запускаем NuMega Symbol Loader, в его меню File выбираем пункт Open, открываем nms-файл и жмем Module->Load. Появившаяся надпись "Symbols for C:\TEMP\SIMPLE.NMS successfully loaded" говорит о том, что все прошло успешно. Осталось только открыть исполняемый файл (File->Open и Module->Load).

Посмотри на скриншоты и сравни, как выглядит экран отладчика с символами и без них.

Без символьной информации назначение функций 401234h и 401124h совсем не очевидно, и на их отладку можно было бы угробить несколько часов лучших лет своей жизни, а с символами все и так ясно. К тому же символьные имена можно использовать в точках останова, например: "bpx _fgets" (установить точку останова на функцию чтения пароля) или "bmp aMygoodpassword" (установить бряк на код, обращающийся к эталонному паролю).

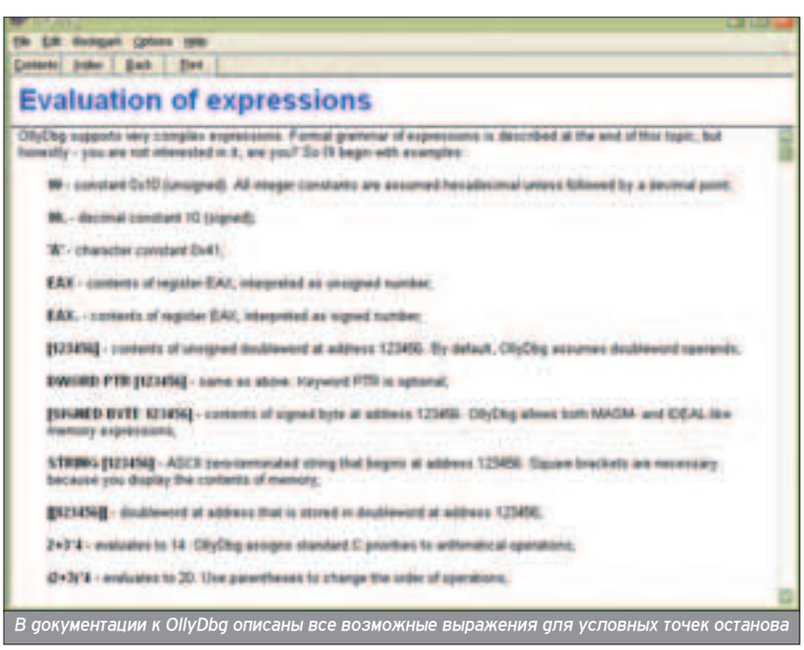
ТОЧКИ ОСТАНОВА НА ФУНКЦИИ

Точки останова (они же breakpoint'ы или просто бряки) - основное оружие хакера в борьбе с защитными механизмами. Наибольшей популярностью пользуются точки останова на API-функции. Чтение содержимого окна часто (но не всегда) осуществляется функцией GetWindowTextA, открытие файла - CreateFileA, загрузка динамической библиотеки - LoadLibraryA и т.д. С помощью установки бряка на эти функции мы можем локализовать защитный код, заставив отладчик всплывать всякий раз, когда защита пытается сделать что-то нехорошее.

Проблема в том, что API-функций очень много и угадать, каким именно способом защита манипулирует окном, не так-то просто. Обычно используется либо тупой перебор всех возможных вариантов один за другим, либо API-шпионы, показывающие, что происходит под капотом отлаживаемой программы.

Для установки brakepoint'а на какую-нибудь функцию достаточно нажать <Ctrl-D> и, дождавшись появления отладчика на экране, написать "bpx имя_функции". Однако нужно учитывать, что в SoftIce точки останова носят глобальный характер, и если мы установим бряк на функцию CreateFileA, отладчик будет вылезать при каждом открытии/создании любого файла в системе. Вот радость! Чтобы охладить пыл отладчика, необходимо использовать условные точки останова. Допустим, мы »

Испытания методик отладки мы проводим над программой Drive LED от компании O&O Software. Ее 30-дневную демонстрационную версию можно скачать с сайта www.oo-software.com/en/download/index.shtml.



В документации к OllyDbg описаны все возможные выражения для условных точек останова

хотим всплывать только на файле "keyfile.key". Открываем MSDN и смотрим прототип CreateFile. Видим, что lpFileName передается в крайнем левом аргументе. Поскольку аргументы API-функций заносятся в стек справа налево, указатель на имя открываемого файла окажется на вершине стека - ниже только адреса возврата.

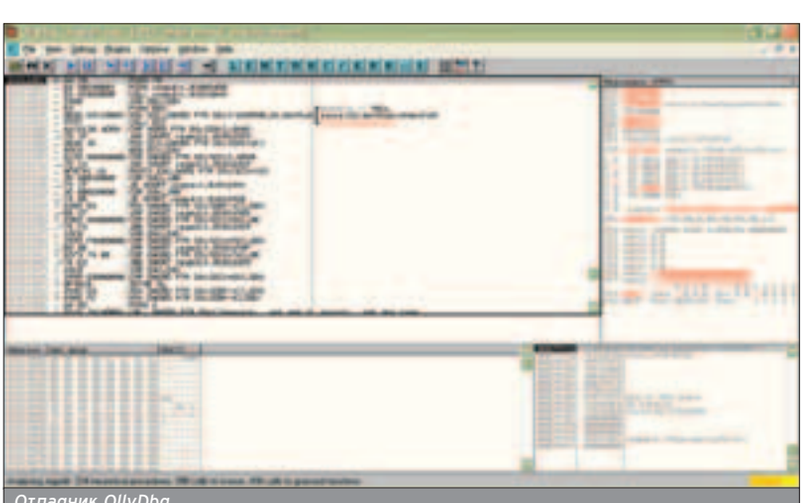
Таким образом, в момент вызова CreateFile lpFileName будет лежать по смещению 4, относительно ESP, и условная точка останова будет выглядеть так: "bpx CreateFileA if (*(esp >4)=='keyf')". Имя файла, заключенное в кавычки, автоматически преобразуется отладчиком в 32-разрядную константу, поэтому его длина не должна превышать четырех байт, причем отладчик чувствителен к регистру ('keyf' и 'Keyf' для него - не одно и то же), а вот файловая система - нет. В большинстве случаев частичного сравнения имени оказывается вполне достаточно. Если же будет недостаточно, можно прибегнуть к оператору AND и сравнивать несколько четырехбитных подстрок за раз. Синтаксис условных точек останова подробно описан в документации на SoftIce, так что не будем останавливаться на этом.

Многие защиты противостоят брякам, например, начиная выполнение API-функции не с первого байта. В таких случаях приходится прибегать к установке brakepoint'ов на native-API - своеобразному фундаменту операционной системы, ниже которого находятся только порты ввода/вывода и драйверы. Описание функций native-API можно найти в Interrupt List'e Ralf'a Brown'a или "The Undocumented Functions Microsoft Windows NT/2000" от Tomas'a Nowak'a. В частности, для создания/открытия файлов используется NtCreateFile.

Отладчик OllyDbg поддерживает намного более мощный механизм условных точек, позволяющий отслеживать практически любые ситуации. Например, выражение EAX == "mypswd" заставит дебаггер всплывать всякий раз, когда регистр EAX указывает на строку с паролем/серийным номером, который мы ввели при регистрации. Это универсальный способ взлома, подходящий практически ко всем защитам. Каким бы образом программа ни извлекала содержимое окна редактирования, в какой-то момент она неизбежно засунет указатель в регистр. Вот тут-то мы с отладчиком и всплывем! Процедура проверки соответствия пароля будет где-то неподалеку. Конечно, этим регистром не обязательно должен быть EAX. Вполне вероятно, что компилятор задействует EBX, ESI или что-то еще. Документация на OllyDbg заявляет о поддержке выражения R32 == "mypswd", где R32 – любой регистр общего назначения, однако в текущих версиях отладчика эта конструкция не работает и все регистры приходится перебирать вручную (благо можно написать свой плагин, автоматизирующий этот процесс).

Помимо API, можно брякать библиотечные функции. В приложениях, написанных на Delphi/Builder/MFC/Visual Basic, прямые вызовы API используются редко. И хотя никакое дело без API-функций, конечно же, не обходится, их анализ дает мало что, особенно если используется динамический обмен данных с окном (DDX) и другие навороченные технологии, обмазывающие API-функции несколькими мегабайтами кривого кода. Это же сдохнуть можно! Но мы не будем! Библиотечные функции легко опознаются ИДОЙ и брякаются как обычные API, только с той разницей, что точка останова носит локальный характер и воздействует лишь на отлаживаемое приложение. А это значит, что после нажатия <Ctrl-D> мы должны переключить контекст управления, чтобы попасть в адресное пространство отлаживаемого приложения. Это осуществляется либо командой "ADDR имя_процесса", либо установкой точки останова на любую API-функцию, вызываемую отлаживаемым приложением. Например SendMessageA. Жмем <Ctrl-D>, пишем "bpx SendMessageA", выходим из SoftIce, дожидаемся, пока он всплывет (если не всплывает, можно дернуть мышью или щелкнуть по отлаживаемому окну). Если в правом нижнем углу отладчика находится имя нашего процесса - все OK, в противном случае выходим из отладчика и ждем его всплытия опять.

> Библиотечные функции легко опознаются ИДОЙ и брякаются как обычные API.



Отладчик OllyDbg

Диалоговое окно, на которое мы поставим бряк

ТОЧКИ ОСТАНОВА НА СООБЩЕНИЯ

Допустим, у нас есть окно с несколькими элементами управления (меню, флажок или кнопка), нажатия на которые мы хотим отследить (см. картинку). Как это сделать? Очень просто! Установить точку останова на сообщение! В Windows весь интерфейс построен на сообщениях (об этом хорошо написал Петзолд в "Программировании для Windows 95"). В частности, при нажатии на элемент управления (или при изменении окна редактирования) окну посылается сообщение WM_COMMAND. Вот на него-то мы и поставим точку останова. Но сначала нам нужно определить дескриптор (handle) окна.

Это можно сделать либо любым Windows-шпионом (например Spyxx, входящим в состав Microsoft Visual Studio), либо средствами самого SoftIce, командой "HWND", выводящей список всех оконных элементов. Если в ответ на "HWND" SoftIce выплюнет "Unable to find a desktop window", необходимо переключить контекст командой "ADDR".

Самая левая колонка содержит дескрипторы оконных элементов, самая правая - имена модулей, которым эти элементы принадлежат. В данном случае диалог обрабатывается библио-

текой oodlrwrs, о чем можно узнать с помощью команды MOD.

Определение дескрипторов окон и элементов управления:

```
Handle Class WinProc TID Module
010098 VMDropTargetClass 0403810 138 VMwareUser
010096 VMDropTargetClass 00403810 138 VMwareUser
010094 VMDropTargetClass 00403810 138 VMwareUser
010090 VMDropTargetClass 00403810 138 VMwareUser
01001C NDDEAgnt 0100BC04 F8 winlogon
120124 #32770 (Dialog) 00F7BC5E 2BC comctl32
220132 #32770 (Dialog 00F7BC5E 2BC oodlrwrs
1F00FE Button 00F7BC5E 2BC oodlrwrs
200102 Button 00F7BC5E 2BC oodlrwrs
1B00F0 Button 00F7BC5E 2BC oodlrwrs
320130 Static 00F7BC5E 2BC oodlrwrs
210138 Static 77E19AA4 2BC oodlrwrs
230116 Static 77E19AA4 2BC oodlrwrs
24014C Static 77E19AA4 2BC oodlrwrs
1700F8 Static 00F7BC5E 2BC oodlrwrs
20013A Static 77E19AA4 2BC oodlrwrs
1F0122 Static 77E19AA4 2BC oodlrwrs
```

Мы видим, что наши три кнопки принадлежат диалогу #32770 с дескриптором 220132. В принципе, можно поставить точку останова и на 120124 - адрес оконной процедуры (WinProc) у них одинаков. Говорим "BMSG 220132 WM_COMMAND" и выходим из SoftIce. Нажимаем на кнопку "Далее>", и (ура!) отладчик послушно всплывает! Остается только немного протрассировать оконную процедуру в поисках кода, обрабатывающего это нажатие.

ТОЧКИ ОСТАНОВА НА ДАННЫЕ

Чаще всего бывает так, что ключевой файл или регистрационные данные извлекаются в одном месте, а обрабатываются совсем в другом. Установив точку останова на GetWindowTextA, мы перехватим код, считывающий введенный нами регистрационный номер, но как найти то место, где он сравнивается с оригиналом? Легко!

Открываем MSDN, смотрим прототип функции GetWindowText. Ага: указатель на возвращаемую строку находится во втором аргументе слева, значит, на момент вызова GetWindowTextA он будет располагаться по адресу ESP + 8 (четыре байта на hWnd и еще четыре на адрес возврата).

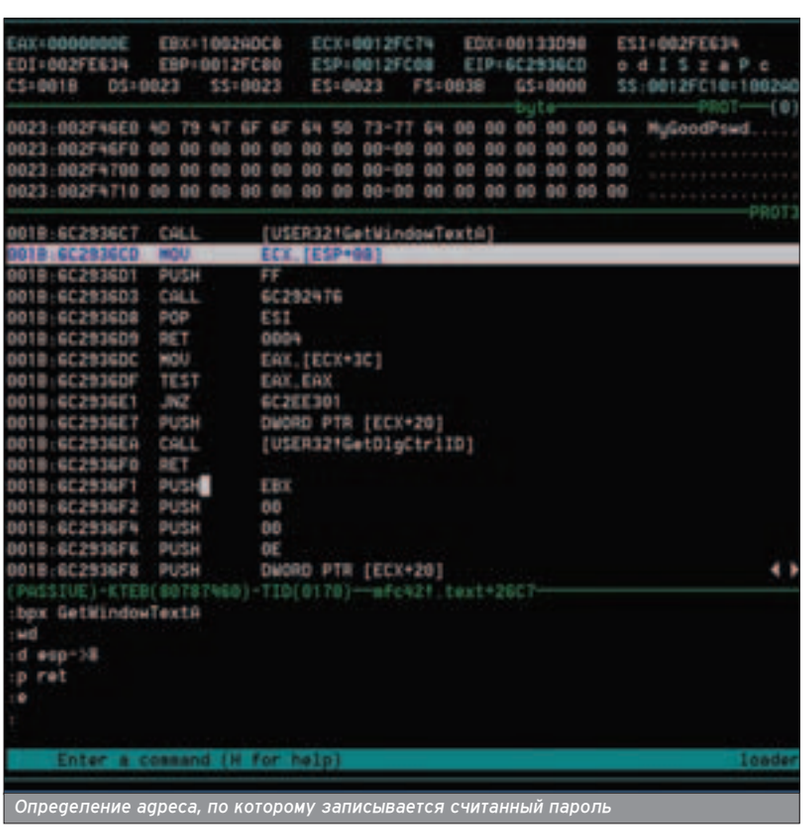
Говорим "bpx GetWindowTextA", выходим из отладчика, вводим серийный номер в окно редактирования, нажимаем OK - дебаггер всплывает (ну, будем считать, что всплывает, в действительности этого может и не произойти; все зависит от того, какую API-функцию использовал программист). Даем команду "d esp->8" (если окно дампа отключено, перед этим необходимо дать команду "wd"), а затем "p ret" - в окне появляется введенная нами строка.

Все, что нам нужно - это ее адрес, который в данном случае равен 2F46E0. Логично: чтобы сравнить пароль с оригиналом, защита должна считать его из памяти. В этот момент она себя и выдаст. Команда "bpm 2F46E0" установит точку останова на адрес 2F46E0, заставляя SoftIce всплывать при каждом чтении/записи этой ячейки памяти. Звучит прекрасно, но на практике срабатывает далеко не всегда. Вовсе не факт, что первое же всплытие отладчика выведет нас к защитному коду. Скорее всего, здесь будет библиотечная функция, копирующая пароль в локальный буфер, передаваемый по цепочке другим функциям. И хорошо если по ссылке! Часто буфер передается по значению, то есть копируется в другой буфер целиком. На каждый из таких буферов приходится ставить точку останова, а максимальное количество бряков на память равно четырем. И это не ограничение отладчика - просто у "Пня" такая архитектура.

Отсюда еще не следует, что точки останова на данные бесполезны - они сильны совсем в другой области. Вот, например, мы выяснили, что в переменной x содержится флаг регистрации. Каким образом выяснили, не суть важно. Допустим, встретили код типа: cmp [x],0/jz nag_screen (если переменная x равна нулю, вывести ругательный диалог). Как определить, где именно инициализируется этот x? В большинстве случаев перекрестные ссылки автоматически восстанавливаются ИДОЙ, однако разработчик защитного механизма может легко ослепить ее. Но едва ли он справится с командой "bpm x"!

А вот другой вариант: мы изменили пару байтиков в программе, а она, обнаружив факт своего взлома, отказалась работать. Чтобы найти процедуру проверки целостности кода, достаточно установить одну или несколько точек останова на модифицированные ячейки. Да много чего можно придумать, главное - фантазию иметь!

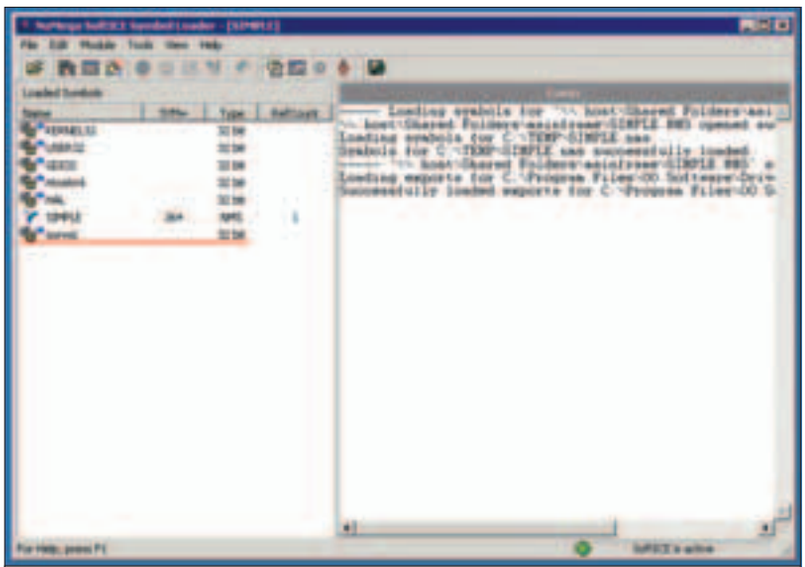
> Установив точку останова на GetWindowTextA, мы перехватим код, считывающий введенный нами регистрационный номер.



Определение адреса, по которому записывается считанный пароль

РАСКРУТКА СТЕКА

Внешние проявления защитного механизма засечь очень легко. Как правило, это либо окошко с надписью "trial expired", либо форма для ввода серийного номера. Установить точку останова на WM_COMMAND легко, но какая польза от этого? Мы окажемся внутри оконной процедуры, в недрах которой зарыт защитный код. Можно, конечно, и потрассировать, но сколько времени уйдет на это! Вот бы узнать, какие команды исполнялись до этого! Обратить выполнение программы вспять и посмотреть, какой именно код определяет факт регистрации программы. Некоторые отладчики поддерживают механизм обратной трассировки (back trace), запоминая »



Загрузка экспорта из динамических библиотек

"Чтобы найти процедуру проверки целостности кода, достаточно установить одну или несколько точек останова на модифицированные ячейки".

все выполняемые команды и складывая их в специальный буфер, однако это сильно замедляет выполнение программы и выводит антиотладочные приемы на оперативный простор. Мы поступим иначе. SoftIce поддерживает шикарную команду STACK, раскручивающую стек и выводящую адреса всех материнских функций. Не совсем равноценная замена обратной трассировки, но, как правило, ее вполне хватает.

В нашем случае ответ отладчика выглядит так:

```
:STACK
0012E138  77E155B5  oorwiz!.text+0001AC5E
0012E168  77E15A3B  USER32!DefWindowProcW+0105
0012E188  77E1FB52  USER32!SendMessageW+0043
0012E214  77E1E6C3  USER32!WINNLSGetIMEHotkey+0E15
0012E254  77E1E561  USER32!EditWndProc+0075
0012E278  77E198DF  USER32!ScrollWindow+0096
0012E29C  77E13EB0  USER32!ShowCursor+0057
0012E2BC  77E16469  USER32!SetTimer+0435
0012E2E0  77E164E5  USER32!SetRect+0065
0012E300  00F7A1B6  USER32!CallWindowProcW+0019
0012E320  00F7A403  oorwiz!.text+000191B6
0012E33C  00F7BC02  oorwiz!.text+00019403
; _AfxPostInitDialog
0012E39C  00F7BC92  oorwiz!.text+0001AC02
; AfxWndProc
0012E3BC  77E13EB0  oorwiz!.text+0001AC92
0012E3DC  77E1591B  USER32!SetTimer+0435
```

Десять первых вызовов относятся к библиотеке USER32.DLL и не представляют для нас никакого интереса (SoftIce неправильно определил принадлежность вызова 12E138h, приписав его к oorwiz, но oorwiz не может располагаться по адресам 77E155B5 - эта зона принадлежит USER32). А вот одиннадцатый вызов 12E320, ведущий к адресу F7A403, весьма интересен. Заглянув сюда дизассемблером, мы обнаружим следующий код:

```
.text:1001A3E6    call dword ptr [eax+10Ch]
.text:1001A3EC    test eax, eax
.text:1001A3EE    jnz short loc_1001A406
.text:1001A3F0    push [ebp+arg_8]
.text:1001A3F3    mov eax, [esi]
.text:1001A3F5    push [ebp+arg_4]
.text:1001A3F8    mov ecx, esi
.text:1001A3FA    push [ebp+arg_0]
.text:1001A3FD    call dword ptr [eax+110h]
.text:1001A403    mov [ebp+var_4], eax ; адрес возврата

.text:1001A406    loc_1001A406:
                      ; CODE XREF: CWnd::WindowProc()
.text:1001A406    mov eax, [ebp+var_4]
.text:1001A409    pop esi
.text:1001A40A    leave
.text:1001A40B    retn 0Ch
```

Вызов функции call dword ptr [eax+110h] - тот самый, к которому ведет адрес возврата. Именно он выводит противный регистрационный диалог. Прокручивая экран дизассемблера вверх, легко найти условный переход, расположенный по адресу 101AEE, который прыгает за диалог возврата. Изменив jnz на jmp short, мы навсегда уберем диалог с экрана. Конечно, такая мера еще не зарегистрирует программу, но это уже кое-что!

ОТЛАДКА ДИНАМИЧЕСКИХ БИБЛИОТЕК

Loader32 (символьный загрузчик SoftIce) как будто позволяет загружать динамические библиотеки, но отлаживать их в автономном режиме не дает, что, собственного говоря, и не удивительно: всякая такая библиотека - просто набор функций, вызываемых из основного процесса. Возьмем библиотеку oorwiz.dll, экспортирующую тройку функций с заманчивыми именами: RegWiz_InitReadOnly, RegWiz_InitTrial, RegWiz_InitLicMgr. Как отладить их?

Заходим в Loader32, выбираем пункт File->Load Export, указываем имя библиотеки (oorwiz.dll). В списке Loader Symbols немедленно появляется новое имя. Теперь загружаем основной исполняемый файл (в данном случае oodled.exe) и устанавливаем точки останова на интересующие нас функции ("bpx RegWiz_InitReadOnly", "bpx RegWiz_InitTrial", "bpx RegWiz_InitLicMgr"), заставляя отладчик всплывать при их вызове.

Поскольку динамические библиотеки перемещаемы, адреса в дизассемблере могут не совпадать с отладчиком. Вот, например, в oorwiz.dll IDA определяет адрес функции RegWiz_InitTrial как 10001D00h, а SoftIce - как F60000. Ну и как с этим жить? А вот как: базовый адрес загрузки (Imagebase) равен 10000000h, в чем IDA честно признается в начале файла. Но загрузить библиотеку по этому адресу не получается, и операционная система перемещает ее по адресу xxxx, о чем говорит команда "MOD" в SoftIce:

Просмотр базовых адресов загрузки командой MOD:

```
:mod
hMod     Base     Module Name     File Name

80400000 804000C8 ntoskrnl
         \WINNT\System32\ntoskrnl.exe
...
00400000 00400108 oodled
         \Program Files\00 Software\DriveLED2\oodled
00F30000 00F300B8 oodlrwrs \
         \Program Files\00 Software\DriveLED2\oodlrwrs
00F60000 00F600F8 oorwiz
         \Program Files\00 Software\DriveLED2\oorwiz
10000000 100000C0 oodledrs
         \Program Files\00 Software\DriveLED2\oodledrs
```

Разница базовых адресов составляет 10001000-F60000 == F0A1000, поэтому, чтобы перевести адрес из отладчика в дизассемблер, к нему необходимо добавить F0A1000, а из дизассемблера в отладчик - отнять.

ЗАКЛЮЧЕНИЕ

Отладчик - это сложный инструмент, и за один день его не освоить. Исследование машинных кодов - настоящее искусство, которому учатся всю жизнь. Так что не нужно огорчаться, если что-то не получается. Чем хитрее защита и чем труднее взлом, тем больше удовлетворения она приносит в конечном итоге! Кто-то сравнил это чувство с экстазом. Ничего подобного! Хакерство намного круче!

Под эгидой Министерства информационных технологий и связи РФ

5-я юбилейная международная выставка-форум

ИнфоКом-2005

инфокоммуникации России - XXI век

28 сентября-1 октября 2005 года

Москва Санкт-Петербург Нижний Новгород Ростов-на-Дону Екатеринбург Иркутск

Экспозиция "ИНФОКОМ"

На данной экспозиции будут представлены услуги, интересные широкому слою населения (B2C):

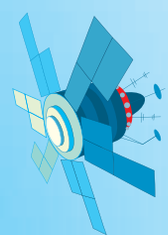
- Комплексные инфокоммуникационные услуги населению
- Беспроводная связь
  - Bluetooth
  - Услуги MMS, GPRS
  - Услуги доступа в Интернет
  - Wi-Fi
  - Мультимедийные услуги на базе Интернета
  - Услуги спутникового, кабельного и наземного телевидения
  - Интерактивные услуги в сетях подвижной связи
  - Мобильный Internet
  - Домашние сети
  - Услуги телемедицины
  - Видеоконференцсвязь
  - Услуги местной, междугородной и международной связи
  - Телефонные аппараты
  - Мультимедиа-продукты
  - Аксессуары
- Интеллектуальный дом (Consumer electronics)

Тематика разделов на стенде ФЦП "Электронная Россия":

- Электронное правительство (Человек и государство)
- Электронный бизнес (Человек и бизнес)
- Электронный мир (Человек в электронном мире)

Экспозиция "ИНФОКОМ ПРО" ориентирована на специалистов в области информатизации и связи (B2B) и предполагает следующие направления:

- Инфокоммуникационные услуги на базе интеграции средств связи и информатизации
- Информатизация и компьютерные сети
- Информационные системы
- Телекоммуникации
- Почтовые услуги
- Научные исследования и технологии
- Беспроводные технологии
- Электроника и электронные компоненты
- Интеллектуальный дом
- Интегрированные системы управления
- Мультирумные аудио/видеосистемы
- Презентационные системы для конференц-залов и ситуационных комнат



РЕСТЭК ИКТ

129223, г. Москва, пр. Мира, ВВЦ, стр.334
Тел.: (095) 544-38-31 Факс: (095) 181-64-30
E-mail: mail-ict@restec.ru

http://www.ict-expo.ru

Информационные спонсоры: Сети и системы связи, Спецхакер, Журнал Хакер, Мобильные компьютеры

Иван Скляров (www.sklyaroff.com)

АНАТОМИЯ ФАЙЛА

ПРОСТО, НО СО ВКУСОМ О PE-ФОРМАТЕ ФАЙЛОВ

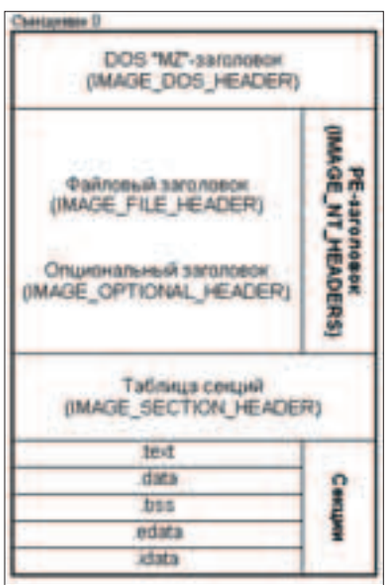
Зачем крэкеру знать все о PE-формате? Затем же, зачем хирургу знать устройство человеческого организма. PE-формат - это основной формат файлов Windows, с которым приходится работать крэкеру. Без знаний об этом формате невозможно полноценно заниматься крэкингом.

Полного описания PE-формата не существует в природе. Есть множество материалов в MSDN, есть отдельные статьи исследователей (например несколько статей от Мэтта Питтрека, которые присутствуют и в MSDN), но все это содержит недомолвки, неточности, ошибки. Впрочем, и эта статья никак не претендует на полное описание и организует лишь вводное знакомство с PE-форматом.

Все структуры, макросы и сигнатуры PE-формата содержатся в заголовочном файле winnt.h, на него и буду опираться в дальнейшем. Увы, этот файл довольно скудно откомментирован, так что о назначении некоторых полей можно лишь догадываться.

ОБЩЕЕ УСТРОЙСТВО PE-ФАЙЛА

■ В самом общем случае PE-файл состоит из четырех частей, которые идут в следующем порядке, начиная с нулевого адреса:



Общее устройство PE-файла

❶. DOS-заголовок (IMAGE_DOS_HEADER);
❷. PE-заголовок (IMAGE_NT_HEADERS);
❸. Таблица секций (IMAGE_SECTION_HEADER);
❹. секции.

В самом конце PE-файла за секциями вполне могут размещаться дополнительные данные, например какая-нибудь отладочная информация, но это носит необязательный характер.

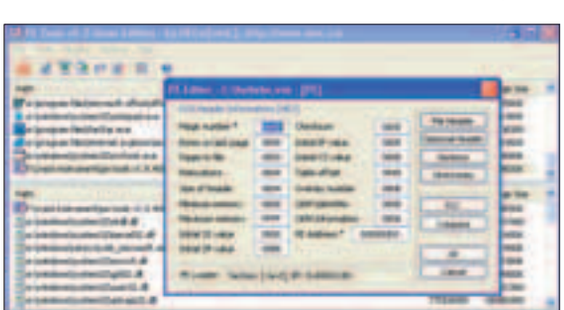
ЗАГОЛОВОК DOS

■ В файле winnt.h DOS-заголовок описан следующей структурой:

```
typedef struct _IMAGE_DOS_HEADER {  // DOS .EXE заголовок
    USHORT e_magic;       // Магическое число
    USHORT e_cblp;        // Количество байт на последней
                          // странице файла
    USHORT e_cp;          // Количество страниц в файле
    USHORT e_crlc;        // Relocations
    USHORT e_cparhdr;     // Размер заголовка в параграфах
    USHORT e_minalloc;    // Minimum extra paragraphs
                          // needed
    USHORT e_maxalloc;    // Maximum extra paragraphs
                          // needed
    USHORT e_ss;          // Начальное (относительное)
                          // значение регистра SS
    USHORT e_sp;          // Начальное значение регистра SP
    USHORT e_csum;        // Контрольная сумма
    USHORT e_ip;          // Начальное значение регистра IP
    USHORT e_cs;          // Начальное (относительное)
                          // значение регистра CS
    USHORT e_lfarlc;      // Адрес в файле на таблицу
                          // переадресации
    USHORT e_ovno;        // Количество оверлеев
    USHORT e_res[4];      // Зарезервировано
    USHORT e_oemid;       // OEM identifier (for e_oeminfo)
    USHORT e_oeminfo;     // OEM information; e_oemid specific
    USHORT e_res2[10];    // Зарезервировано
    LONG   e_lfanew;      // Адрес в файле
                          // нового .exe-заголовка
  } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
```

Самым важным здесь является поле e_lfanew, которое содержит 4-байтовое смещение от начала файла до PE-

Культовые статьи Мэтта Питтрека в MSDN. В интернете можно найти их перевод на русский язык



Просмотр значений полей DOS-заголовка реального PE-файла с помощью утилиты PE Tools

заголовка. Первое поле структуры e_magic содержит сигнатуру исполняемого файла. Все MS-DOS-совместимые исполняемые файлы имеют сигнатуру 0x54AD, которая в ASCII-символах представлена двумя символами MZ. По этой причине заголовок DOS часто называют MZ-заголовком.

PE-ЗАГОЛОВОК

■ Формат PE-заголовка представлен структурой IMAGE_NT_HEADERS, которая в 32-разрядных системах соответствует структуре IMAGE_NT_HEADERS32:

```
typedef struct _IMAGE_NT_HEADERS {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

Как видно, структура состоит из трех частей. Первая часть - это сигнатура, которая для PE-файла равна 0x00004550 или, в ASCII-символах, "PE00". Кстати, именно из-за этой сигнатуры исполняемый файл Windows и называют PE-файлом. Вторая часть - это файловый заголовок (FileHeader), а третья часть - опциональный заголовок (OptionalHeader). Рассмотрим отдельно оба заголовка.

ФАЙЛОВЫЙ ЗАГОЛОВОК

■ Файловый заголовок в winnt.h описан такой структурой:

```
typedef struct _IMAGE_FILE_HEADER {
    WORD    Machine;
    WORD    NumberOfSections;
    DWORD   TimeDateStamp;
    DWORD   PointerToSymbolTable;
    DWORD   NumberOfSymbols;
    WORD    SizeOfOptionalHeader;
    WORD    Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

#define IMAGE_SIZEOF_FILE_HEADER          20
```

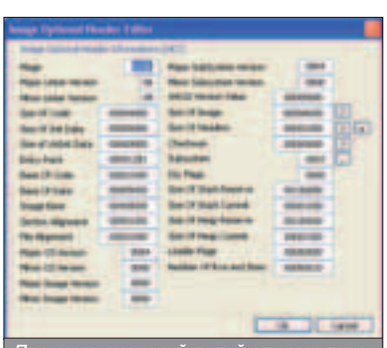
Как видишь, размер заголовка для удобства также определен в файле заголовков (значение 20). Первое поле, Machine, используется для идентификации типа процессора, для которого скомпилирован исполняемый файл: Alpha, MIPS, Intel и др. Поле NumberOfSections содержит количество секций в файле, или, точнее - количество заголовков секций и количество тел секций, имеющихся в исполнимом PE-файле. Это поле занимает одно слово, поэтому максимальное число секций в файле составляет 65536. В третьем поле, TimeDateStamp, записано число секунд, которые прошли с момента 16-ти часов 1969 года до момента создания файла. В поле PointerToSymbolTable содержится смещение таблицы символов, о которой будет рассказано ниже, а в поле NumberOfSymbols указано количество символов в таблице символов отладочной информации. SizeOfOptionalHeader хранит размер опционального заголовка. И последнее поле, Characteristics, содержит специфические характеристики файла, например, IMAGE_FILE_DEBUG_STRIPPED указывает на то, что из файла была удалена отладочная информация, а IMAGE_FILE_DLL говорит о том, что файл является DLL, а не исполняемым файлом. Все характеристики с комментариями можно увидеть в файле winnt.h сразу после описания файлового заголовка.

Просмотр значений полей файлового заголовка с помощью PE Tools

ОПЦИОНАЛЬНЫЙ ЗАГОЛОВОК

■ Теперь перейдем к опциональному заголовку. Вот его описание из заголовочного файла winnt.h:

```
typedef struct _IMAGE_OPTIONAL_HEADER {
    //
    // Standard fields.
    //

    WORD    Magic;
    BYTE    MajorLinkerVersion;
    BYTE    MinorLinkerVersion;
    DWORD   SizeOfCode;
    DWORD   SizeOfInitializedData;
    DWORD   SizeOfUninitializedData;
    DWORD   AddressOfEntryPoint;
    DWORD   BaseOfCode;
    DWORD   BaseOfData;

    //
    // NT additional fields.
    //

    DWORD   ImageBase;
    DWORD   SectionAlignment;
    DWORD   FileAlignment;
    WORD    MajorOperatingSystemVersion;
    WORD    MinorOperatingSystemVersion;
    WORD    MajorImageVersion;
    WORD    MinorImageVersion;
    WORD    MajorSubsystemVersion;
    WORD    MinorSubsystemVersion;
    DWORD   Win32VersionValue;
    DWORD   SizeOfImage;
    DWORD   SizeOfHeaders;
    DWORD   CheckSum;
    WORD    Subsystem;
    WORD    DllCharacteristics;
    DWORD   SizeOfStackReserve;
    DWORD   SizeOfStackCommit;
    DWORD   SizeOfHeapReserve;
    DWORD   SizeOfHeapCommit;
    DWORD   LoaderFlags;
    DWORD   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUM-
BEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEAD-
ER32;
```



Просмотр значений полей опционального заголовка с помощью PE Tools

А сейчас жизненно важное замечание: структура условно поделена на "стандартные поля" и "дополнительные поля NT". Стандартные поля присутствовали в формате файлов COFF, который положен в основу формата PE. Но, несмотря на то, что стандартные поля имеют такие же названия, как определено в COFF, в действительности Windows использует некоторые из них для целей, совершенно противоположных предписанным COFF.

Поле Magic не используется совсем. В полях MajorLinkerVersion и MinorLinkerVersion записана соответственно старшая и младшая часть номера версии линкера, который был использован при создании файла.

SizeOfCode - размер исполняемого кода, округленный к верхней границе.

SizeOfInitializedData - размер инициализированных данных.

SizeOfUninitializedData - размер неинициализированных данных (BSS).

AddressOfEntryPoint - одно из самых интересных полей для крэкера: относительный виртуальный адрес (RVA - Relative Virtual Address) точки входа в программу. Многие поля в PE-файле задаются с помощью RVA. RVA - это смещение в памяти по отношению к базовому адресу ImageBase (о нем ниже). Чтобы получить линейный адрес, необходимо сложить RVA и ImageBase.

BaseOfCode - относительное смещение сегмента кода (.text) в загруженном файле.

BaseOfData - относительное смещение сегмента неинициализированных данных (.bss) в загруженном файле.

ImageBase - базовый адрес, начиная с которого в память будет отображен образ исполняемого файла. Обычно по умолчанию он равен 400000h, однако это значение не постоянно и может изменяться.

SectionAlignment - граница выравнивания секций в памяти.

FileAlignment - граница выравнивания секций в файле на диске, то есть до загрузки в память.

MajorOperatingSystemVersion и **MinorOperatingSystemVersion** - старшая и младшая часть номера самой старой версии операционной системы, которая позволяет запускать данный файл.

»

MajorImageVersion и **MinorImageVersion** - старшая и младшая часть номера версии файла.

MajorSubsystemVersion и **MinorSubsystemVersion** - старшая и младшая часть номера версии Win32-подсистемы.

Win32VersionValue - один из параметров, назначение которого никому не известно.

SizeOfImage - общий размер файла, загруженного в память от базового адреса загрузки (ImageBase) до адреса конца последней секции. Значение этого поля должно быть кратно значению SectionAlignment.

SizeOfHeaders - содержит суммарный размер места, занимаемого всеми заголовками файла, включая заголовок MS-DOS и размер таблицы секций.

CheckSum - контрольная сумма, используемая для проверки целостности исполняемого файла во время загрузки. Это поле устанавливается и проверяется линковщиком.

Subsystem - подсистема, используемая файлом в качестве интерфейса с пользователем. Например, IMAGE_SUBSYSTEM_WINDOWS_GUI гласит, что программа использует графический пользовательский интерфейс Windows. Значения всех возможных подсистем содержатся в файле заголовков winnt.h после описания структуры опционального заголовка.

DllCharacteristics - характеристики DLL, также описанные в winnt.h.

SizeOfStackReserve, **SizeOfStackCommit**, **SizeOfHeapReserve**, **SizeOfHeapCommit** - эти поля определяют объем адресного пространства, зарезервированного и выделенного для стека и кучи. По умолчанию и для стека, и для кучи зарезервировано 16 страниц, на что указывает запись в winnt.h:

```
#define IMAGE_NUMBEROF_DIRECTORY_ENTRIES    16
```

LoaderFlags - в настоящее время не используется.

NumberOfRvaAndSizes - содержит размер массива DataDirectory, расположенного далее.

DataDirectory - массив указателей на различные важные компоненты PE-файла. Элементами этого массива являются структуры типа IMAGE_DATA_DIRECTORY:

```
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

Каждая из этих структур содержит виртуальный адрес определенной таблицы и размер этой таблицы. Информация о таблицах всегда располагается в строгом порядке, например, информация о таблице ресурсов (IMAGE_DIRECTORY_ENTRY_RESOURCE) всегда идет вторым номером (если начинать отсчет с нуля).

ТАБЛИЦА СЕКЦИЙ

Таблица секций - это заголовки секций, в которых описано их размещение на диске и в памяти. Все заголовки секции имеют длину 40 байт и располагаются без выравнивания. Заголовок секции определяется следующей структурой:

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME];
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;

#define IMAGE_SIZEOF_SECTION_HEADER         40

/* Name[IMAGE_SIZEOF_SHORT_NAME] - имя секции.
В файле winnt.h указано, что длина секции должна
быть не больше восьми символов: */

#define IMAGE_SIZEOF_SHORT_NAME             8
```

Misc - объединение (union), которое принимает значение PhysicalAddress, если файл является объектным, или VirtualSize, если файл является исполняемым. В исполняемом файле это поле содержит неокругленный размер раздела.

VirtualAddress - смещение, по которому загрузчик должен отобразить секцию в адресном пространстве процесса. Оно вычисляется относительно базового адреса загрузки (ImageBase).

SizeOfRawData - размер секции, округленный в большую сторону до значения FileAlignment.

PointerToRelocations, **PointerToLinenumbers**, **NumberOfRelocations**, **NumberOfLinenumbers** - эти поля не используются и обычно устанавливаются в ноль.

Characteristics - характеристики секции, которые описаны в файле winnt.h. Из них я выбрал основные и привел их в отдельной таблице.

СЕКЦИИ

За всеми заголовками в файле следуют тела секций. В Microsoft исторически сложились такие устойчивые названия секций: ".text", ".bss", ".data", ".rdata", ".edata", ".idata", ".rsrc", ".debug" и некоторые другие. Однако совсем не обязательно, чтобы секции имели такие названия. Например, компиляторы от фирмы Inprise (бывшая Borland) присваивают секциям имена вида CODE, DATA и т.п. Кроме того, программист может создавать дополнительные секции со своими названиями. Наличие точки в начале секции также необязательно. Секции расположены вплотную друг к другу без промежутков, но это совсем не значит, что в самих секциях не бывает пустых мест. Каждая стандартная секция имеет определенное назначение, например:

- ".text" распложен код программы;
- ".bss" размещаются неинициализированные данные;
- ".data" - инициализированные данные;
- ".edata" – функции, экспортируемые файлом;
- ".idata" - таблицы импортируемых функций;
- ".rsrc" – ресурсы;

В сегменте ".rdata" - данные только для чтения (строки, константы, информация отладочного каталога).

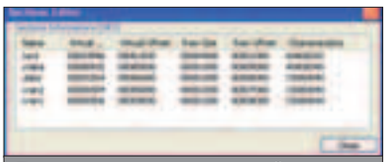
Но это назначение секций не строгое, например, ".text" вполне может быть начинен обычными данными, а также экспортируемыми функциями. Некоторые секции имеют особый формат, сейчас внимание сосредоточу на них.

СЕКЦИЯ ЭКСПОРТА (.EDATA)

В секции .edata представлен перечень функций и данных, которые PE-файл экспортирует для использования сторонними модулями. Сегмент экспорта стандартно состоит из следующих частей:

Определение	Значение	Описание
IMAGE_SCN_CNT_CODE	0x00000020	Секция содержит код
IMAGE_SCN_CNT_INITIALIZED_DATA	0x00000040	Секция содержит инициализированные данные
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080	Секция содержит неинициализированные данные
IMAGE_SCN_LNK_NRELOC_OVFL	0x01000000	Секция содержит расширенные поправки
IMAGE_SCN_MEM_DISCARDABLE	0x02000000	Секция может быть игнорирована
IMAGE_SCN_MEM_NOT_CACHED	0x04000000	Секция не кешируема
IMAGE_SCN_MEM_NOT_PAGED	0x08000000	Секция не сбрасывается в страничный файл
IMAGE_SCN_MEM_SHARED	0x10000000	Общая секция
IMAGE_SCN_MEM_EXECUTE	0x20000000	Секция выполняемая
IMAGE_SCN_MEM_READ	0x40000000	Секция для чтения
IMAGE_SCN_MEM_WRITE	0x80000000	Секция для записи

Таблица с некоторыми характеристиками секций (Characteristics) из файла winnt.h



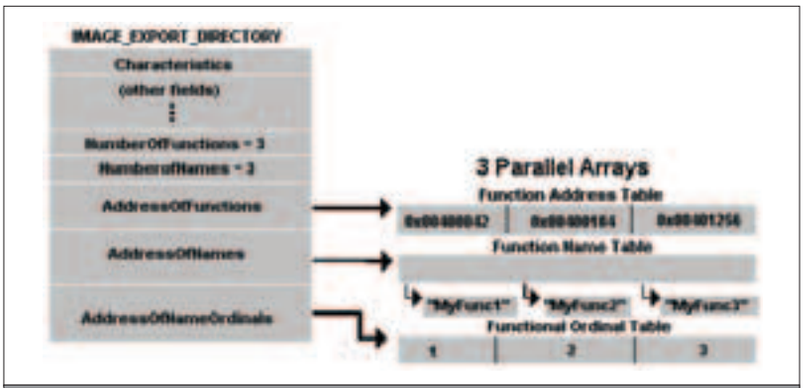
Просмотр информации секций реального файла. Заметно, что присутствуют две нестандартных секции .ivan1 и .ivan2

❶. Оглавление;
❷. Таблица указателей на экспортируемые имена;
❸. Таблица порядковых номеров функций;
❹. Таблица экспортируемых имен.

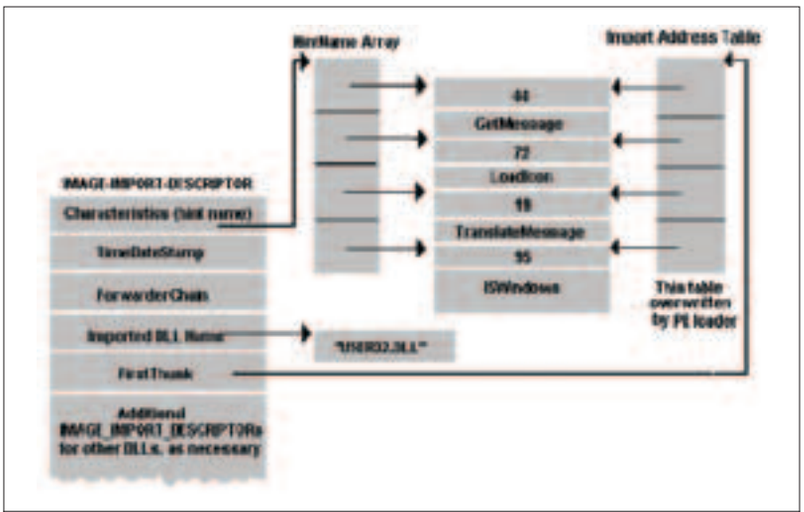
Оглавление представлено вот такой структурой:

```
typedef struct _IMAGE_EXPORT_DIRECTORY {
  DWORD   Characteristics;
  DWORD   TimeDateStamp;
  WORD    MajorVersion;
  WORD    MinorVersion;
  DWORD   Name;
  DWORD   Base;
  DWORD   NumberOfFunctions;
  DWORD   NumberOfNames;
  DWORD   AddressOfFunctions;   // RVA from base of image
  DWORD   AddressOfNames;       // RVA from base of image
  DWORD   AddressOfNameOrdinals;  // RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

Characteristics - в настоящее время не используется.
TimeDateStamp - время создания таблицы экспорта.
MajorVersion и **MinorVersion** - старший и младший номер версии файла.
Name - имя файла.
Base - начальный порядковый номер для экспортируемых функций (обычно 1).
NumberOfFunctions и **NumberOfNames** - содержат количество функций и имен функций, экспортируемых из файла.
AddressOfFunctions - относительный виртуальный адрес (RVA) на таблицу указателей экспортируемых функций.
AddressOfNames - это относительный виртуальный адрес (RVA) на таблицу экспортируемых имен.
AddressOfNameOrdinals - это относительный виртуальный адрес (RVA) на таблицу порядковых номеров функций.
Все три таблицы связаны между собой особым образом.



Устройство секций экспорта (рисунок взят из статьи Мэтта Питтрека "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format")



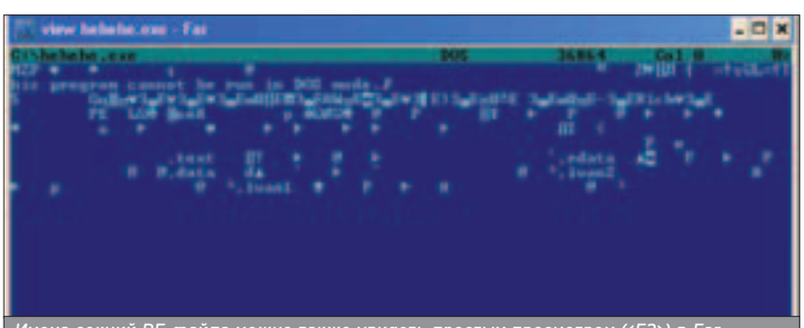
Устройство секций импорта (рисунок из статьи Мэтта Питтрека "Peering Inside the PE: A Tour of the Win32 Portable Executable File Format")

СЕКЦИЯ ИМПОРТА (.IDATA)

Секция импорта содержит информацию о функциях и данных, которые файл импортирует из библиотек DLL. Секция .idata состоит из последовательности таблиц импорта, каждая из которых представлена структурой IMAGE_IMPORT_DESCRIPTOR. Каждая такая структура соответствует одной DLL, из которой программа импортирует функции. Последняя структура в последовательности имеет нулевые поля. Вот формат этой структуры из файла winnt.h:

```
typedef struct _IMAGE_IMPORT_DESCRIPTOR {
    union {
        DWORD   Characteristics;  // 0 for terminating null
                                  // import descriptor
        DWORD   OriginalFirstThunk;  // RVA to original
                // unbound IAT (PIMAGE_THUNK_DATA)
    };
    DWORD   TimeDateStamp; // 0 if not bound,
                // -1 if bound, and real date\time stamp

//     in IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT (new BIND)
// O.W. date/time stamp of DLL bound to (Old BIND)

    DWORD   ForwarderChain;  // -1 if no forwarders
    DWORD   Name;
    DWORD   FirstThunk;  // RVA to IAT (if bound this IAT has
                                  // actual addresses)
} IMAGE_IMPORT_DESCRIPTOR;
typedef IMAGE_IMPORT_DESCRIPTOR UNALIGNED
*PIMAGE_IMPORT_DESCRIPTOR;
```

Первое поле является объединением (union) и принимает такие значения: либо 0 (Characteristics), если структура является последней в последовательности, либо указатель (OriginalFirstThunk) на последовательность структур типа IMAGE_THUNK_DATA (о ней читай ниже).
TimeDateStamp - время создания файла, из которого импортируются функции (чаще всего имеет нулевое значение).
ForwarderChain - переадресация к другой DLL, если возникает такая необходимость.
Name - имя библиотеки, которой соответствует текущая структура IMAGE_IMPORT_DESCRIPTOR.



Имена секций PE-файла можно также увидеть простым просмотром (<F3>) в Far

»

FirstThunk - еще один указатель на структуру типа IMAGE_THUNK_DATA.

```
#include "poppack.h"            // Back to 4 byte packing

typedef struct _IMAGE_THUNK_DATA32 {
    union {
        PBYTE  ForwarderString;
        PDWORD Function;
        DWORD Ordinal;
        PIMAGE_IMPORT_BY_NAME AddressOfData;
    } u1;
} IMAGE_THUNK_DATA32;
typedef IMAGE_THUNK_DATA32 * PIMAGE_THUNK_DATA32;
```

Как видно, структура IMAGE_THUNK_DATA содержит в себе объединение. Если функция импортируется по порядковому номеру, значение объединения расценивается как Ordinal. Если функция импортируется по имени, объединение интерпретируется как указатель на структуру типа IMAGE_IMPORT_BY_NAME (AddressOfData). Эта структура в файле winnt.h описана следующим образом:

```
typedef struct _IMAGE_IMPORT_BY_NAME {
    WORD    Hint;
    BYTE    Name[1];
} IMAGE_IMPORT_BY_NAME, *PIMAGE_IMPORT_BY_NAME;
```

Здесь поле Hint является подсказкой, облегчающей поиск указателя на функцию.

Name - указатель на строку с именем импортируемой функции.

СЕГМЕНТ РЕСУРСОВ (.RSRC)

Ресурсы имеют довольно сложную иерархическую структуру. В начале секции ресурсов расположено оглавление, которое представлено вот такой структурой:

```
typedef struct _IMAGE_RESOURCE_DIRECTORY {
    DWORD   Characteristics;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    WORD    NumberOfNamedEntries;
    WORD    NumberOfIdEntries;
//  IMAGE_RESOURCE_DIRECTORY_ENTRY
DirectoryEntries[];
} IMAGE_RESOURCE_DIRECTORY,
*PIMAGE_RESOURCE_DIRECTORY;
```

Здесь интересны только два поля: NumberOfNamedEntries и NumberOfIdEntries, которые показывают число ресурсов, идентифицируемых по имени, и число ресурсов, идентифицируемых по номеру. Сразу за оглавлением расположена таблица ресурсов, каждая строка которой представлена структурой:

```
typedef struct _IMAGE_RESOURCE_DIRECTORY_ENTRY {
    union {
        struct {
            DWORD NameOffset:31;
            DWORD NameIsString:1;
        };
        DWORD   Name;
        WORD    Id;
    };
    union {
        DWORD   OffsetToData;
        struct {
            DWORD   OffsetToDirectory:31;
            DWORD   DataIsDirectory:1;
        };
    };
} IMAGE_RESOURCE_DIRECTORY_ENTRY,
*PIMAGE_RESOURCE_DIRECTORY_ENTRY;
```

Видно, что каждая строка таблицы состоит из двух объединений. Первое из них определяет, как идентифицируется ресурс: по имени или по номеру. Если старший бит первого объединения установлен в единицу, то ресурс идентифицируется по имени. Если в 0 - то по ID. Поле OffsetToData всегда используется для указания на потомка либо в ветви дерева, либо в конечном узле. Если первый бит второго объединения установлен в единицу, то OffsetToData указывает на потомка в ветви дерева. Если в 0, то на конечный узел. Конечные узлы - это низшие узлы в дереве ресурсов, которые определяют размер и местоположение непосредственно данных ресурса. Каждый конечный узел представляет собой структуру IMAGE_RESOURCE_DATA_ENTRY:

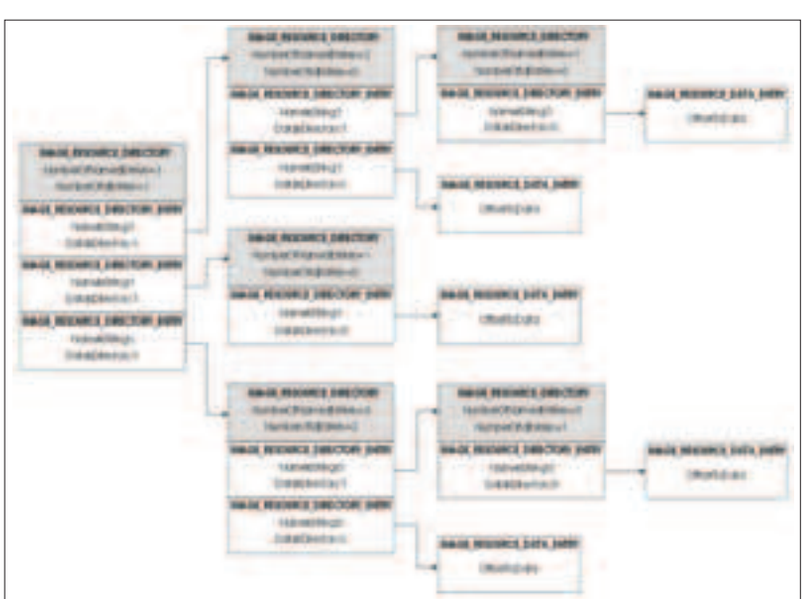
```
typedef struct _IMAGE_RESOURCE_DATA_ENTRY {
    ULONG   OffsetToData;
    ULONG   Size;
    ULONG   CodePage;
    ULONG   Reserved;
} IMAGE_RESOURCE_DATA_ENTRY,
*PIMAGE_RESOURCE_DATA_ENTRY;
```

OffsetToData и Size указывают местоположение и размер непосредственно данных ресурса. CodePage - номер кодовой страницы, которая должна использоваться при декодировании данных. Последнее поле в структуре зарезервировано.

На рисунке можно увидеть примерную иерархию ресурсов в файле, а за рассматриванием рисунка нужно запомнить, что данные каждого отдельного ресурса (меню, диалоги, иконки, кнопки и т.д.), на которые указывает OffsetToData конечного узла, также могут иметь собственную особую структуру.

Стандартные идентификаторы ресурсов можно найти в заголовочном файле winuser.h:

```
/*
 * Предопределенные типы ресурсов
 */
#define RT_CURSOR          MAKEINTRESOURCE(1)
#define RT_BITMAP          MAKEINTRESOURCE(2)
#define RT_ICON            MAKEINTRESOURCE(3)
#define RT_MENU            MAKEINTRESOURCE(4)
#define RT_DIALOG          MAKEINTRESOURCE(5)
#define RT_STRING          MAKEINTRESOURCE(6)
#define RT_FONTDIR         MAKEINTRESOURCE(7)
#define RT_FONT            MAKEINTRESOURCE(8)
#define RT_ACCELERATOR     MAKEINTRESOURCE(9)
#define RT_RCDATA          MAKEINTRESOURCE(10)
#define RT_MESSAGETABLE    MAKEINTRESOURCE(11)
```



Приблизительная иерархия ресурсов в PE-файле

РЕЗЮМЕ

Я рассказал не обо всех существующих полях PE-формата, но даже моя статья демонстрирует то, какую громоздкую структуру имеет этот формат. Если кто-то возьмется описывать форматы всех известных ресурсов, то, вполне может быть, посвятит этому целую книгу!

Существует множество утилит для просмотра и редактирования полей PE-файла. PE Tools от отечественных разработчиков не знает равных себе по мощности в этой области. На некоторых скриншотах к этой статье значения полей заголовков и секций реального PE-файла показаны именно благодаря PE Tools.

MIMS 2005

9-ая Московская Международная Автомобильная Выставка

9th Moscow International Motor Show

24 – 28 августа 2005

24 – 28 August 2005

Выставочный комплекс ЗАО "Экспоцентр" на Красной Пресне, Москва

Exhibition Complex of Expocentr, Krasnaya Presnya, Moscow, Russia

Организаторы / Organisers:

ITE Group Plc
105 Salisbury Road
London, NW6 6RG, UK
Tel: +44 (0) 20 7950 5177
Fax: +44 (0) 20 7950 5150
Website: www.motorshows-ite.com

ITE LLC
ул. Щепкина 42, Строение 2а
129110 Москва, Россия
Тел: +7 095 935 7350
Факс: +7 095 935 7351
Вебсайт: www.motorshow.ru

При поддержке / upported by

Министерства промышленности и энергетики РФ

Правительства Москвы

При содействии / Assisted by:

ЗАО Экспоцентр

Content:

ВЗЛОМ

Симонов Илья aka Shturmovik (nazi@ghOsts.org)

РАСПАКОВКА ВРУЧНУЮ

«СНЯТИЕ» УПАКОВЩИКОВ ПРИЛОЖЕНИЙ

Для человека, вставшего на путь исследования программ и их защит, первым барьером к вкусному сердцу кода станет упаковщик исполняемых файлов. Что это такое и как обойти его, я и расскажу в этой статье.

РАЗВЕДКА БОЕМ

Начнем, как обычно, с начала и продолжим до конца :). А сначала опишу принцип работы упаковщиков и его общие особенности. Упаковщик - программа для сжатия исполняемого файла, которая отличается от архиватора лишь тем, что добавляет код распаковщика в тело программы. При запуске управление передается этому коду вплоть до полной распаковки программы в память. Определение грубое, но, как ни крути, общий смысл выразил. Задача крэкера - слить дамп памяти (не пугайся слов - еще успею объяснить их) распакованной программы, а затем, конечно, восстановить импорт, но все по порядку. Итак, поехали.

Как ты, наверное, уже предполагаешь, а может быть, давно знаешь, существует множество различных пакеров. Естественно, способы распаковки у каждого особенные. Сначала нам предстоит разведка, то есть определение упаковщика, которым сжата программа. Хочу сразу отметить, что это не столь важная деталь, как может показаться. Действительно, когда распакуешь вручную парочку пакеров, тебе уже будет безразлично то, как называется пакер, ранее бывший неизвестным для тебя. Принцип их действия, по сути, одинаков: если узнал, как работает один, значит, ты понял, как работают все. Распознавание языка, на котором написана программа, также то, каким упаковщиком она ужата, осуществляется сигнатурным способом, который так полюбился нашим антивирусным конторам. На эту тему существуют замечательные статьи Alex'a на не менее замечательном портале cracklab.ru. Конечно, если ты сейчас с интересом читаешь эту статью, значит, ты еще многого не знаешь, и специально для новичков изобретена отличная в своем роде программа-распознаватель PEiD, которая не только покажет, чем упакован наш файл, но и расскажет еще много интересного о жертве.

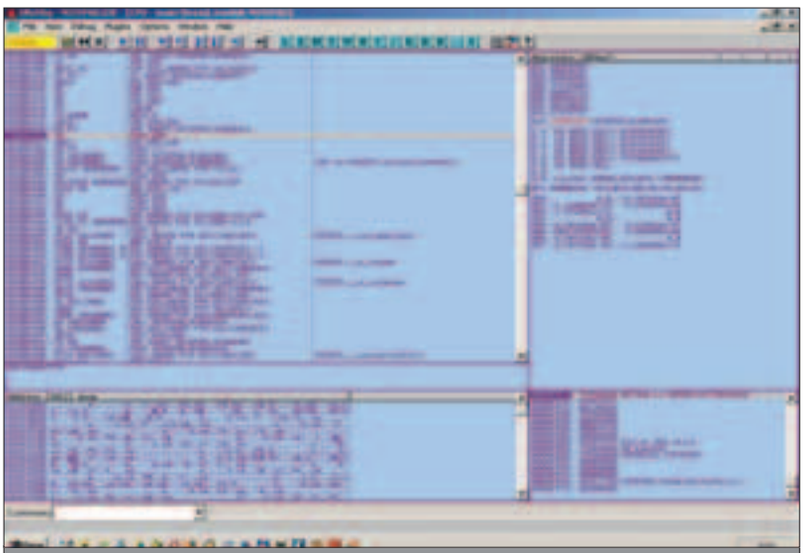
Обзор всех инструментах, необходимых реверсеру, также есть в этом номере, поэтому время поговорить о программах еще будет. А сейчас мы окунемся в прошлое и предадимся классике. Внимание! Распаковываем блокнот, упакованный UPX. И что бы там нам ни говорили, что якобы веселее распаковывать FSG, блокнот и UXP - это классика, от которой никуда не деться. И наш урок начнется именно с нахождения Оригинальной точки входа в сжатую программу.

Для общего разъяснения скажу, что после запуска программы с физического носителя операционная система проецирует образ программы в оперативную память (точнее, в определенное адресное пространство памяти). В этом случае сам код располагается в памяти с началом по адресу так называемого значения Image base (то есть Image Base - адрес в памяти, начиная с которого идет код программы). Однако начало проекции совсем не означает начало исполняемого кода. И вот уже в проекции существует Точка входа (EP - Entry Point) в программу - то место, откуда начинается выполнение исходника. Поскольку я уже упоминал, что упаковщик добавляет свой код в тело программы, то на сцене появляется понятие Оригинальная точка входа. Как уже можно было догадаться, это точка входа в изначальную программу, если бы она не была запакована. Естественно, нам необходимо найти это место, снять дамп памяти (dump - снимок/образ памяти) исходной программы и сохранить его на диск. Мне не хочется делать из тебя бездумного робота-шаблонщика, поэтому попытаюсь донести до глубин твоего сознания суть процесса.

НАХОЖДЕНИЕ OEP. ТЕОРИЯ

Как мы уже знаем, код распаковщика предшествует коду ужатой программы, следовательно, нам необходимо первым шагом найти переход на OEP. Если ты знаком с языком ассемблер (на первое время достаточно азов), то тут будет понятнее. Для перехода на OEP в стандартных пакерах применяются, как правило, несколько способов: безусловный переход на OEP, который используется в большинстве простых упаковщиках, предназначенных для простого сжатия программ, но никак не защита от взлома. Например, наш UPX делает именно вот так и никак не иначе.

```
POPAD
JMP NOTEPAD.01006420
DB 00
DB 00
DB 00
DB 00
```



После прыжка на OEP

В ASPack'е от нашего соотечественника используется несколько другой метод.

```
POPAD
JNZ SHORT NOTEPAD1.010103BA
MOV EAX,1
RETN 0C
PUSH NOTEPAD1.01006420
RETN
```

Как видишь, адрес OEP кладется в стек PUSH NOTEPAD1.01006420, а затем командой RET производится по адресу в стеке. Что и требовалось доказать, друзья.

Теперь поподробнее: как я нашел переходы и почему в большинстве случаев перед вызовом OEP находится команда POPAD, что заметили все начинающие реверсеры, хотя мало кто из них придал значение этому факту.

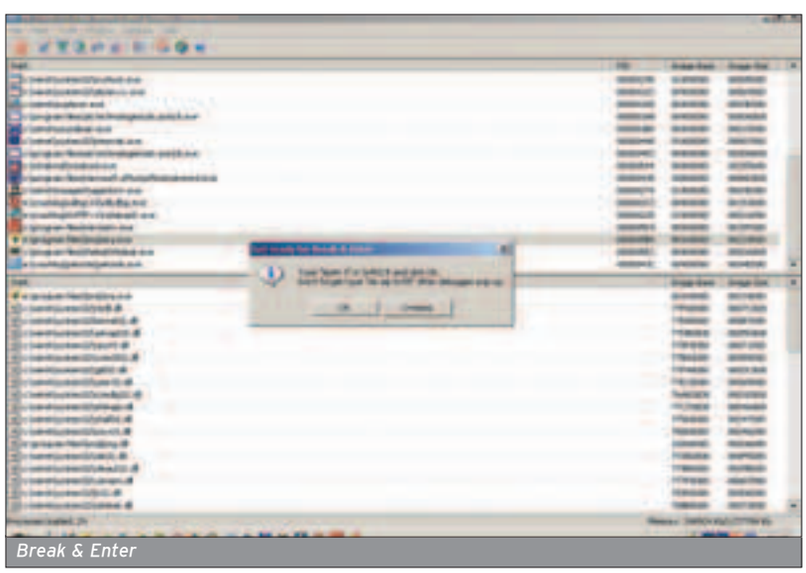
Дело в том, что для корректной работы программы важным и необходимым условием является гармония адресного пространства. После прохождения распаковки и до передачи управления исходной программы код пакера должен вернуть значения регистров, а главное - стека в первоначальное состояние. Именно поэтому в начале кода распаковщика всегда стоит (как правило!) команда PUSHAD, которая указывает процессору поместить значения регистров в стек. Команда POPAD существует для противоположной операции. Это можно наглядно посмотреть в программе, упакованной UPX: после возвращения стека в первозданное состояние происходит прыжок на OEP. Если просто смотреть листинг дизассемблера, то после перехода на OEP мы увидим непонятные наборы символов (назвать их кодом сложновато) либо лишь пустую область.

Однако этот способ действует только на очень простых пакерах наподобие UPX. В случае с ASPack'ом, как мы видим, такое уже не подойдет. Поэтому лучше перейдем к практике распаковки простейших упаковщиков. Нам понадобится установленный SoftIce. Если у тебя по какой-либо причине нет вышеприведенного отладчика, то можешь пользоваться любым другим - ход действий остается таким же. Мне SoftIce ближе к сердцу, и советую пользоваться именно им или OllyDbg, которым я пользовался, как ты уже понял, только ради скриншотов и наглядности кода.

НАХОЖДЕНИЕ OEP. ПРАКТИКА

Итак, приступим. Для тех, кто работает с OllyDbg, остановиться на EP программы не составит труда - нужно просто открыть файл в отладчике. Но поскольку мы работаем с SoftIce'ом, тут необходимо знать некоторые тонкости. Итак, сейчас мы научимся прерываться на точке входа в SoftIce с помощью великолепной программы PETools, о которой можно прочесть в другой статье этого номера. Этот способ широко известен всем: лично я научился ему после прочтения статьи хорошо известного в своих кругах MozgC (TSRh), за что ему отдельное спасибо. Теперь следим за рукой: запускаем PETools, далее в меню Tools выбираем Break&Enter и указываем нужный нам файл. Затем появляется простое окошко с сообщением, что необходимо запустить SoftIce, ввести команду bpint3, после ее прохождения нажать кнопку OK в окошке и ввести следующую команду типа e eip 0xXX. На самом деле все намного проще, чем кажется. При использовании Break&Enter PETools заменяет первый байт точки входа на байт CCh (то есть специальное отладочное прерывание int3, которое генерирует исключение), мы уже ставим в отладчике бряк (breakpoint - точка остановки) на это прерывание (bpint3), то есть, грубо говоря, мы помечаем по-своему точку входа. Когда отладчик останавливает программу на этом месте, команда "eb eip 0xXX" (где XX - определенные байты, которые были на точке входа) записывает оригинальные байты точки входа, тем самым приводя все в первоначальный вариант. Остается только снять все бряки командой "bc *" и работать дальше.

А дальше мы должны рассуждать, как найти OEP. Если думать трезво и вспомнить уроки учителей, то мы узнаем, что после исполнения своего долга упаковщик, как истинный мужчина, восстанавливает все в первоначальное состояние. При стандартном начале Windows-программ указатель на верхнюю часть стека один и тот же и равен значению регистра esp. Когда пакер передает управление первоначальной программе, указатель восстанавливается, но перед этим, как правило, упаковщик считывает из стека значение esp-4. Поскольку 98% простых упаковщиков одинаковы в своих принципах, мы имеем право утверждать, что перед переходом на OEP будет взято значение из стека, равное esp-4. Следовательно, вызываем отладчик и пишем bpm esp-4. После нажатия <F5> в случае с UPX мы сразу же прерываемся на команде POPAD и последующем переходе на OEP. Не всегда первая остановка будет верна: тут необходимо смотреть переходы, и если они есть, значит, мы пришли правильно. И вот мы на заветном прыжке на OEP. Это означает то, что сейчас в памяти лежит распакованная программа с точ- »



Break & Enter

кой входа, которую мы только что нашли. Остается только слить дамп на диск и восстановить импорт. Наверное, все уже поняли, что для того чтобы нормально слить дамп, нужно остановить программу на месте прыжка на OEP.

СНЯТИЕ ДАМПА

Для тех, кто пользуется OllyDbg, никакого труда не составит поставить breakpoint на команде перехода (всего лишь нажав <F2>) и запустить программу до срабатывания точки останова. Любителям SoftIce придется немного поколдовать. Тут мы уже просто зациклим программу на этом месте путем замены байт, да SoftIce и это может :). Итак, двигаемся на команду прыжка и вводим команду "a", что означает ввод ассемблерных команд, а затем пишем jmp eip. Применяем магическим <Enter>'ом, и теперь наша команда делает бесконечные прыжки на себя. Как видишь, ничего сложного нет. Вообще самое сложное в распаковке упаковщиков - это нахождение OEP.

Отладчики запущены, программы висят на прыжках на точку входа, остается только снимать дамп. Вопрос - чем? Конечно, сейчас есть уйма плагинов как для SoftIce, так и для OllyDbg, однако ими я не пользуюсь, ибо все это от лукавого. Существуют две замечательные программы PETools и LordPE, первую сегодня мы уже использовали, значит, снимать дамп будем второй. Вообще принципиальной разницы в этих программах нет, но нужно учиться всему и знакомиться со всеми. После запуска LordPE находим в списке наш зацикленный процесс, правый клик на нем и full dump - что может быть проще? Сохраняем дамп на диск. Ура! Наконец-то наш файл готов... но что это? Он не запускается и выдает ошибку. А это потому, что наш новоявленный блокнот не знает, какие же функции ему использовать. Нам предстоит еще восстановить таблицу импорта. Кстати, если мы уже сдампили программу, то можно закрывать отладчики. Для тех, у кого отладчик ядра в LordPE, выбираем программу и в контекстном меню нажимаем burn process.

ВОССТАНОВЛЕНИЕ ИМПОРТА

Импорт мы восстановим с помощью еще одной замечательной программы Import Recontructor. Запустим упакованный блокнот, затем найдем его в листе процессов ImpRec'a. Теперь нам необходимо указать RVA OEP (в ImpRec он просто OEP). Не пугайся слов: на врезке есть объяснение всему. Вот формула "RVA OEP = VA OEP - ImageBase". Ее нужно запомнить крепко. Image Base мы сможем найти нажав на кнопку PE Editor в LordPE. В нашем случае RVA = 01006420 – 01000000 = 6420. Вводим это значение в поле OEP и жмем автопоиск, то есть IAT AutoSearch. После этого нажимаем Get Imports. Мы должны увидеть строки с функциями и надписью YES напротив. Если все так, а так и должно быть, то кнопка Fix Dump открывает нам дверь на путь истинный. Останется только указать наш дамп и после нажатия на заветную кнопку OK наслаждаться распакованным блокнотом.

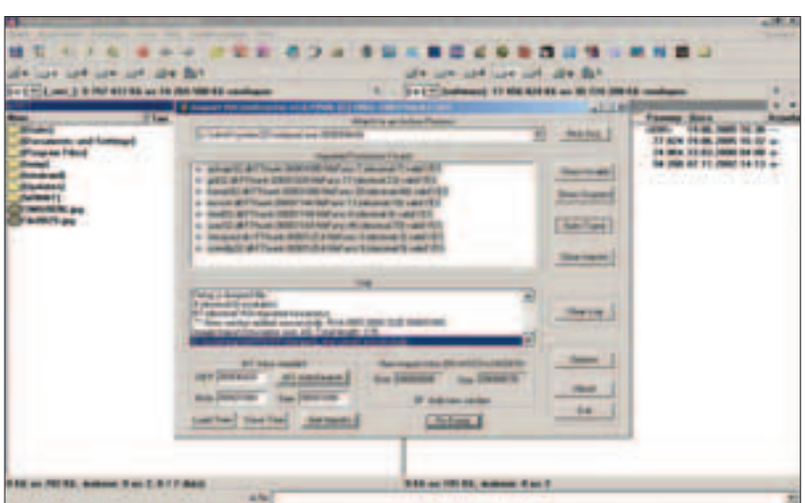
Однако не всегда все так просто и не всегда автопоиск спасает. Поэтому напоследок упомяну некоторую хитрость, которая уже давно используется у реверсеров. Если мы попробуем таким же образом поступить с файлом, упакованным упаковщиком с небольшой степенью защиты, то функции мы не обнаружим, в этом случае придется искать RVA самостоятельно. Как известно, функции в исполняемом файле находятся в виде адресов. Упаковщик сохраняет всю таблицу. Следовательно, по адресам в простом HEX-редакторе мы сможем обнаружить таблицу импорта в упакованном файле. Но для этого нам необходимо знать хотя бы одну функцию, которая присутствует в файле, и ее адрес. GetModuleHandleA встречается практически во всех Windows-программах, а адрес функции легко узнается с помощью отладчика. Достаточно ввести в SoftIce "exp GetModuleHandleA" и получить ответ 77E87D93 (или в OllyDbg просто поставить бряк на эту функцию, дойти до нее и прочитать внизу DS:[01001094]=77E87D93 (KERNEL32.GetModuleHandleA) =)). При поиске в HEX-редакторе не забывай, что все адреса хранятся в обратном порядке, то есть искать мы будем 937DE877. У тебя, возможно, будет другое число. Уверяю, что ты точно не пропустишь место таблицы импорта. Запускаем опять PE Editor, выбираем наш дамп и жмем FLC. Затем вычисляем RVA, для этого нажимаем на кнопку offset и вводим найденный нами адрес. После нажатия кнопки DO в поле RVA будет то, что мы искали. На глаз прикинем размер таблицы импорта. Теперь все так же, только без автопоиска. Если мы были правы (а мы были правы!), то появится уйма функций. Удалим те, что с надписью NO, и фиксируем снимок (Fix Dump). Наслаждаемся и читаем врезки.

ВМЕСТО ЗАКЛЮЧЕНИЯ

Надеюсь, чтение этой статьи утяжелило твой багаж знаний и ты встал на истинный путь реверсера. Может быть, ты, наоборот, вспомнил молодость, когда еще стоял на распутье, и теперь с ностальгией читаешь слова заключения, а на твоем винте уже лежат рабочие дампы армы.

УЧИТЬ НАИЗУСТЬ!

EP (Entry Point) - точка входа в программу, располагающуюся в адресном пространстве.
OEP (Original Entry Point) - точка входа в исходную программу, логично только когда программа упакована. Тогда EP - точка входа в код упаковщика.
Image Base - адрес начала размещения программы в памяти. Не путать с точкой входа! Точка входа - это адрес старта программы, а Image Base - адрес начала всего кода.
RVA (Relative Virtual Address) - относительный виртуальный адрес, адрес смещения относительно Image Base.
VA (Virtual Address) - виртуальный адрес, адрес в памяти, который нам показывает отладчик.



Вот он - наш импорт

GL#OM (gl00m-crk@yandex.ru)

«ВРЕМЕННАЯ» ЗАЩИТА

МЕТОДЫ РАБОТЫ TRIAL-ЗАЩИТ

Первый вопрос, который обычно задают новички крэкинга: "Как побороть ту или иную trial-защиту?". И это совершенно естественно. Как известно, это самый популярный вид защиты. Прочитав эту статью, ты научишься довольно быстро справляться с trial-защитой.

TRIAL-АЗБУКА

Trial ("испытание", "пробный") - это один из видов защиты программ, распространяемых по лицензии shareware. Его принцип заключается в том, что пользователю предоставляется для ознакомления полностью функциональная версия программы, но ограниченная по времени использования. Для того чтобы убрать это ограничение, необходимо купить программу... или воспользоваться народными методами ;).

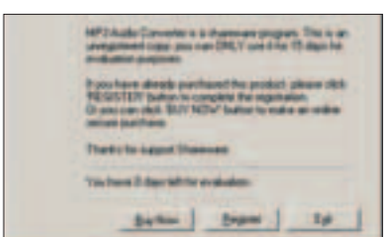
Вспомни, как ты, запустив свою любимую программу, с негодованием смотрел на всплывающее окно с просьбой зарегистрироваться за энное количество у.е. и демонстрирующий себя тебе trial-счетчик, который навевал мысли о приближении конца срока использования. И вот этот прекрасный день настал: при запуске программы ты со вселенским ужасом в глазах лицезрел финальное NAG-окно с ужасной фразой "Trial license has expired." Ты наверняка не растерялся и пошел на свой любимый сайт в поиске "лекарства" и, я уверен, нашел crack, который благополучно не подошел, сославшись на неверный размер или CRC целевого файла. Секундное недоумение, и у тебя возникла новая мысль, а именно - найти серийный номер. И, о чудо! Он нашелся и подошел ;). Могу лишь представить, как ты был ошарашен и расстроен, когда, через некоторое время, а может быть, даже при следующем запуске регистрация и работа с программой были заблокированы :((в интернете сидел?). Только теперь, зайдя на все тот же warez'ный сайт, где был найден регистрационный ключ, ты обнаружил сообщение, предупреждающее всех, что в программе предусмотрен online-тест. Там же наверняка приложено подробное руководство по разблокированию программы и дан "мудрый" совет запретить программе какое-либо общение с интернетом. Это не наш путь. Крэкеры мы или нет? Да =)! Значит, будем зрить в корень, а точнее в ассемблерный листинг.

ОТ СЛОВ К ДЕЛУ

Итак, существует несколько особенностей trial-защит, на каждую из которых есть свой метод противодействия. Собственно, о самых стандартных я сейчас и попытаюсь рассказать.

Trial-счетчик

Да, это самая главная особенность. Какой, скажите мне, может быть trial без счетчика =)? Trial-счетчик - это обычный счетчик, связанный с какой-либо скрытой переменной. Его увеличение/уменьшение зависит от каких-либо системных особенностей, обычно от даты. По достижении определенного числа нормальная работа с программой блокируется.



"Trial has expired" - пора что-то срочно предпринять!

Суть, я думаю, ясна. Скрытая переменная может находиться либо в файле, либо в реестре. Поэтому вспомним стандартный набор Windows API, которые обычно используются при этом, а также рассмотрим примеры исследования:

❶. Для работы с реестром: RegQueryValueA, RegQueryValueExA, RegOpenKeyA и т.д.

MP3 Audio Converter

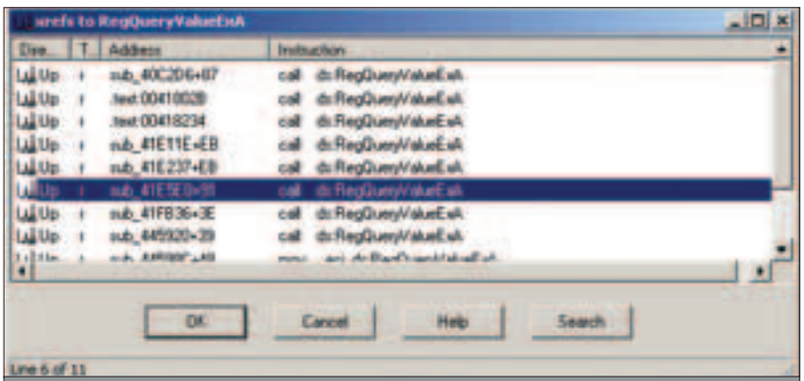
Пример: в качестве цели мы возьмем программу MP3 Audio Converter, которая живет по адресу www.ezsoft-magic.com.

Ничем не запакована, написана на Microsoft Visual C++, trial - 15 дней. Откроем ее в IDA и после завершения дисассемблирования пойдем в закладку Imports. Ищем там одну из API-функций для работы с реестром, например RegQueryValueExA, щелкаем по ней два раза и поочередно просматриваем все участки кода, вызывающие ее.

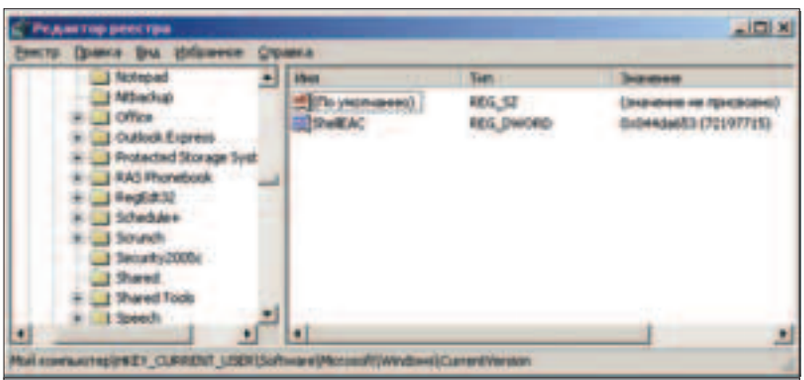
Через некоторое время мы заметим интересный код:

```
push 0 ; lpReserved
mov eax, [ebp+lpValueName]
push eax ; lpValueName
mov ecx, [ebp+hKey]
push ecx ; hKey
call ds:RegQueryValueExA
mov [ebp+var_8], eax
cmp [ebp+var_8], 0
jnz loc_41E74A
mov eax, dword ptr [ebp+Data]
xor edx, edx
mov ecx, 64h
div ecx
mov [ebp+var_20], edx
mov eax, dword ptr [ebp+Data]
xor edx, edx
mov ecx, 64h
div ecx
mov [ebp+var_1C], eax
cmp [ebp+var_20], 0
jbe short loc_41E6AE
cmp [ebp+var_20], 0Fh<- 0Fh = 15
jbe short loc_41E6C4
```

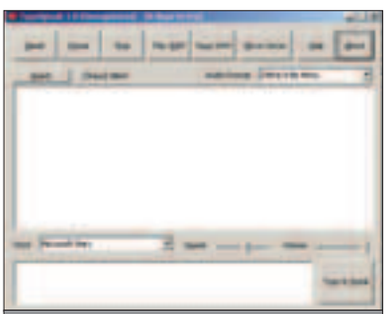
Странное совпадение, не правда ли? Для проверки нашего предположения откроем цель в OllyDbg и поставим точку останова на адрес начала данной процедуры - 0041E5E0. В результате мы обнаружим, что скрытая переменная находится в ключе раздела по адресу HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion и называется ShellEAC. После ее удаления trial обнуляется. Есть море способов



IDA Xreferences



Ключ реестра ShellEAC



Программа TypeSpeak

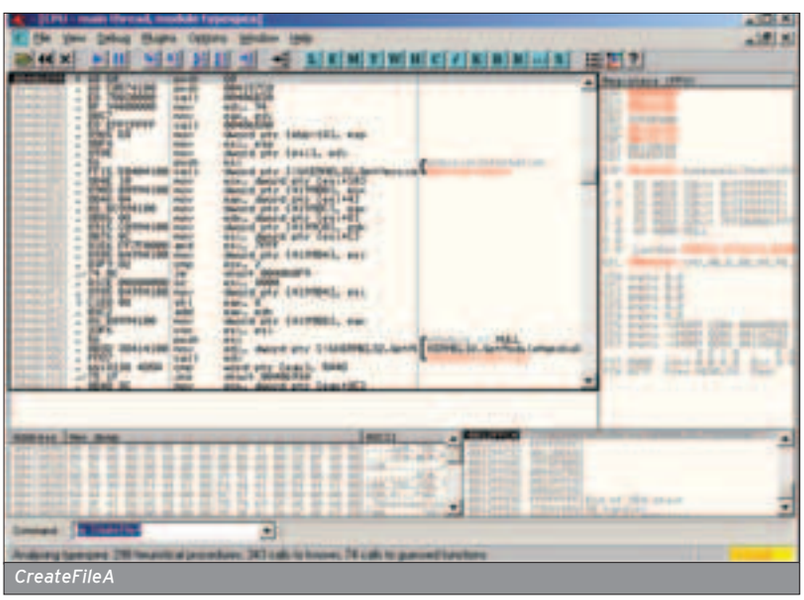
сделать его бесконечным, пусть это будет твоим домашним заданием ;).

❼. Для чтения из файла: ReadFile, ReadFileEx, CreateFileA, SetFilePointer, SetFilePointerEx и т.д.

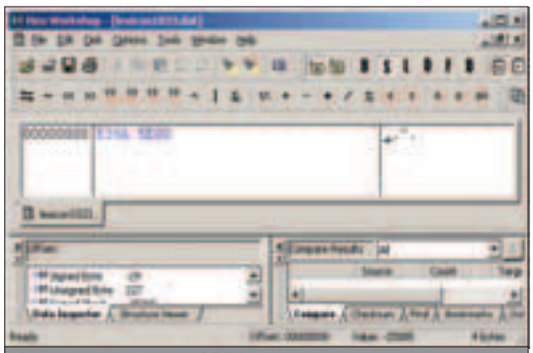
Можно попытаться найти место проверки счетчика в IDA, то есть произвести действия, аналогичные тем, что были проделаны в предыдущем методе борьбы (это, естественно, предпочтительнее, особенно для новичков, так как этим самым наращивается опыт), но на этот раз мы сразу откроем цель в OllyDbg. Так или иначе, мы должны искать вызов API для работы с файлом, например CreateFileA. Ставим точку останова на эту API.

Как ни странно, вызов всего один:

```
push 004147AC ; More = "lexicon1033.dat"
lea eax, dword ptr [esp+554] ;
push eax; Path = "C:\WINNT\System32"
call dword ptr [<&SHLWAPI.PathAppendA>] ;
push edi; hTemplateFile
```



CreateFileA



Содержимое lexicon1033.dat

```
push 80 ; Attributes = NORMAL
push 3; Mode = OPEN_EXISTING
push edi; pSecurity
push edi; ShareMode
push C0000000 ; Access = GENERIC_READ|GENERIC_WRITE
lea ecx, dword ptr [esp+568] ;
push ecx; FileName
call dword ptr [<&KERNEL32.CreateFileA>]; CreateFileA
mov ebx, eax
cmp ebx, -1
je 004053B3
```

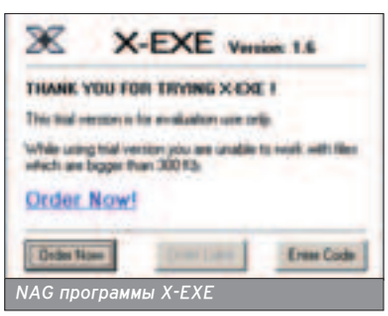
Происходит открытие файла с очень странным названием lexicon1033.dat. Не спеши удалять его. Это ничего хорошего не даст. Лучше взглянуть на него при помощи Hex Workshop.

Файл содержит всего четыре байта, они-то и отвечают за trial =). Узнать это можно при помощи отладчики или еще как-нибудь... Я определил методом научного тыка =). Эти байты никогда не изменяются, они записываются лишь однажды - при установке программы. При запуске программы идет их сверка с системной датой. Самое интересное, что я заметил: если забить этот файл нулями, то программа станет полностью зарегистрированной %)!

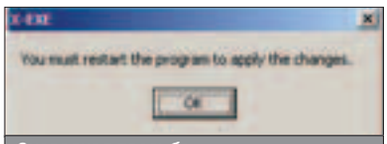
❽. Для чтения из ini-файла: GetPrivateProfileStringA, GetPrivateProfileIntA.

Пример: именно для проверки счетчика примера не нашлось, точнее, я не смог вспомнить ;). Ну, не в этом суть, главное - порядок действий. Цель - X-EXE. Сайт - www.softeza.com.

Trial'а как такового нет, только NAG с таймером ожидания. Наша задача - зарегистрировать это чудо мысли ;). Нажимаем на кнопку Enter Code и вводим в появившееся окно любую чушь, нажимаем на кнопку OK. Спустя пару секунд увидим сообщение с просьбой перезапустить программу для завершения регистрации. »



NAG программы X-EXE



Очередная просьба...

Твоя первая мысль: "Программа куда-то записала введенный код..." Да, ты абсолютно прав. Взглянем на содержимое директории, в которую была установлена программа, а точнее на файл Settings.ini:

```
[Settings]
1=11112222 <- хехе, я именно это и вводил
2=0
```

Теперь откроем нашу цель в OllyDbg и поставим точку останова на GetPrivateProfileStringA. Первый же вызов наш.

```
push eax
call 00406908 ; jmp to kernel32.GetPrivateProfileStringA
mov ecx, eax
ea edx, dword ptr [ebp-800] <- EDX = указатель на код
mov eax, dword ptr [ebp+8]
call00404690
pop edi
pop esi
pop ebx
mov esp, ebp
pop ebp
retn 8
```

Пройдя по ret, а затем еще чуть ниже, замечаем следующее:

```
mov eax, dword ptr [ebp-10] <= EAX = указатель на код
mov edx, 004786E8; ASCII "1254960154494"
call 004049AC<= процедура сравнения
jnz short 00478513
mov eax, 00478700; ASCII "Thank you for registration X-EXE!"
call0042CAFC
```

Не ведись на эту уловку! Это заблокированный ключ, который не принесет должного результата. Смотрим дальше:

```
mov eax, dword ptr [ebp-10] <= указатель на введенный
                               серийный номер
call 00404860 <= функция получения длины строки
cmp eax, 0C
jl 004785B5 <= длина должна быть больше 11
ea eax, dword ptr [ebp-16C]
push eax
mov ecx, 9 <= сколько байт
mov edx, 4 <= с какого байта начинать
mov eax, dword ptr [ebp-10]
call 00404AC0 <= функция копирования подстроки
mov eax, dword ptr [ebp-16C] <= EAX = указатель на
подстроку
push eax
..... Код создания строковой константы
00478597 8B95 90FEFFFFmov edx, dword ptr [ebp-170]
<= EDX = указатель на константу
pop eax
call 004049AC <= процедура сравнения
jnz short 004785B5
```

В результате имеем код: xxx124651914 (где xxx - любые символы). Ура! Задача выполнена =).

На этом со счетчиками покончим... Хотя нет, мне хотелось бы упомянуть еще одну API-функцию - GetSystemTime =). Как уже все заметили (в первых двух пунктах), ее вызов происходит как раз перед вызовом рассмотренных API. И, что самое приятное, он единственный! "Так зачем же мы тут мучались?!" - спросишь ты. Решать тебе, но, по-моему, приведенный метод надежнее ;).

NAG-окно с просьбой зарегистрироваться

Понятия "борьба с NAG-окнами" для меня не существует, по крайней мере,

Круто, что сказать...

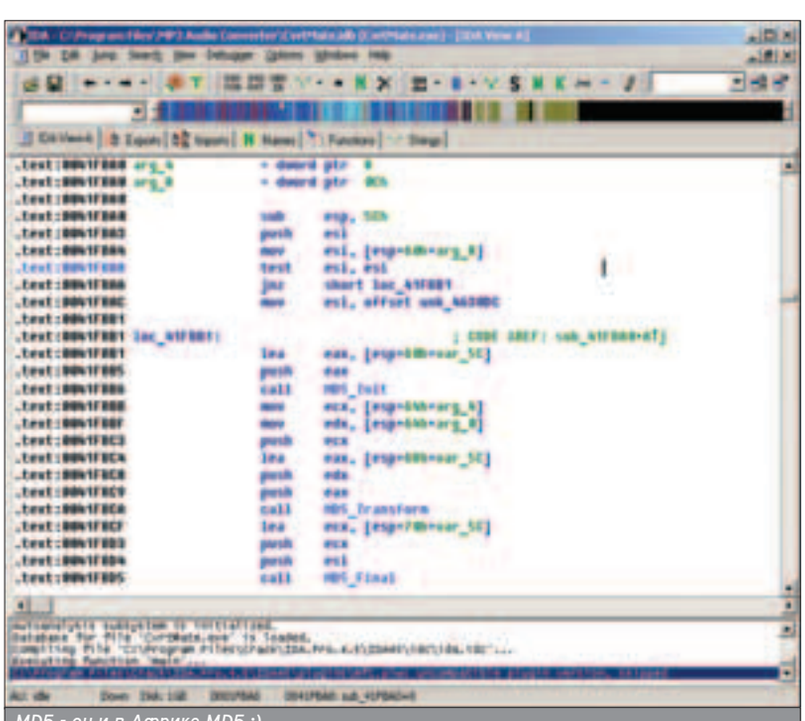
при исследовании trial-защит. Потому что единственный правильный путь - это разобрать непосредственно процедуру проверки серийного номера или, на худой конец, подправить ее, в 80% случаев получим полностью зарегистрированную программу. А что нам даст "убийство" NAG-окна? Окно пропало, а ограничения остались. Поэтому смотреть надо в сторону всех вышеперечисленных API, а также: GetWindowTextA, GetDlgItemTextA и т.д.

Пример: рассмотрим все тот же MP3 Audio Converter, но теперь на предмет регистрации.

Загружаем его в OllyDbg и ставим точку останова на RegQueryValueExA. Второй вызов наш:

```
push 104; Arg4 = 00000104
lea ecx, dword ptr [ebp-104] ;
push ecx; Arg3
push 3; Arg2 = 00000003
push 0044E7E0 ; Arg1 = 0044E7E0 ASCII "6+E"
call 0041D220 ; CvrtMate.0041D220
add esp, 10
lea edx, dword ptr [ebp-214]
push edx; pHandle
lea eax, dword ptr [ebp-210] ;
push eax ; Subkey =
"SOFTWARE\EZSoftMagic\AudioConverter\SN"
push 80000002 ; hKey = HKEY_LOCAL_MACHINE
call dword ptr [<&ADVAPI32.RegOpenKeyA>];
mov dword ptr [ebp-10C], eax
cmp dword ptr [ebp-10C], 0
je short 0041E303
xor eax, eax
jmp short 0041E34B
lea ecx, dword ptr [ebp+C]
push ecx; pBufSize
mov edx, dword ptr [ebp+8] ;
push edx; Buffer
lea eax, dword ptr [ebp-108] ;
push eax; pValueType
push 0; Reserved = NULL
lea ecx, dword ptr [ebp-104] ;
```

Единственный верный путь при взломе NAG - разобрать процедуру проверки серийного номера.


MD5 - он и в Африке MD5 ;)

```
push ecx; ValueName
mov edx, dword ptr [ebp-214] ;
push edx; hKey
call dword ptr [<&ADVAPI32.RegQueryValueExA>] ;
```

Пройдя еще немного, мы увидим:

```
lea ecx, dword ptr [ebp-418]
push ecx; Arg5
lea edx, dword ptr [ebp-41C] ;
push edx; Arg4
lea eax, dword ptr [ebp-424] ;
push eax; Arg3
lea ecx, dword ptr [ebp-10];
push ecx; Arg2
lea edx, dword ptr [ebp-410] ;
push edx; Arg1 = 0012FB00 ASCII "125-1-125125"
call 0041D7C0 ; CvrtMate.0041D7C0
add esp, 14
mov eax, dword ptr [ebp-10]
xor eax, dword ptr [ebp-424]
cmp dword ptr [ebp-41C], eax
jnz short 00401931
```

Даже невооруженному глазу заметно, что call 0041D7C0 - процедура проверки серийного номера ;). Немного подправив ее, мы получим полнофункциональную версию. "А почему бы не найти верный серийный номер?" - спросишь ты. Дело в том, что в самой последней проверке действительности нашего номера от него берется хэш MD5, а затем поочередно сверяется с элементами массива хэшей, который хранится в теле программы =/. Перебрать хоть один из них за разумное время не представляется возможным из-за размера ключа.

Остается только один, он же идеальный, вариант:

❶. разобрать структуру серийного номера;

❷. взять от него хэш MD5;

❸. заменить полученной хэш-суммой один из элементов массива...;

❹. дерзать ;)!

❸. Online-проверка

Вот чего я не люблю, так это Online-тест. Найдешь верный серийный номер, вроде бы все отлично, так нет же - еще эту чертову проверку патчить. Ничего не поделаешь...

EzDNS

Если кто-нибудь не в курсе, online-тест - это проверка достоверности регистрационных данных на сервере производителя. Другими словами, во время работы программа может в любой момент проверить наш серийный номер, и, если, например, его нет в базе на сервере производителя, работа с программой будет заблокирована.

Итак, я лично встречал два типа online-проверок:

❶. с помощью API-функций: InternetGetConnectedStateEx, InternetOpenUrlA;

❷. посредством MFC-функции - CHttpFile::SendRequest.

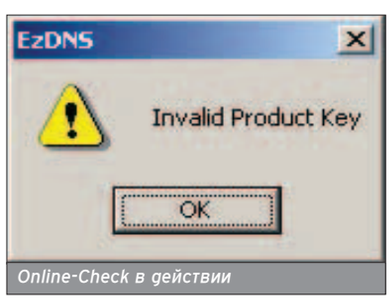
На первом способе я останавливаться не буду, потому что не нашел подходящей цели... Одна упакована, вторая написана на VB и откомпилирована в псевдокод. Пришлось бы рассказать еще и про распаковку или, еще хуже, про исследование VB-PCode, а это выходит за рамки статьи =/. Поэтому покажу метод борьбы лишь со вторым типом.

В качестве примера рассмотрим программу EzDNS, которую можно скачать по адресу: www.ezdns.com. Для ее регистрации вводится имя GL#0M и номер 4BB19HWLA7FF6JFP. Теперь у нас вроде бы полностью рабочая программа... Ах нет, не пройдет и трех секунд, как вылетит MessageBox с надписью "Invalid Product Key" и программа закроется.

OK, загрузим цель в IDA и найдем все вызовы функции ?SendRequest@CHttpFile. Их всего-то два, вот один из них:

```
push 0
push 0
push 0
push 0
mov ecx, esi
call ?SendRequest@CHttpFile@@QAEHPBDKPAXK@Z ;
lea ecx, [ebp+var_7C]
```

Теперь следовало бы разобраться, что же он запрашивает, но, по-моему, это лишнее: легче забить оба вызова опкодом NOP и идти пить пиво. Ах да, чуть не забыл. Последнее домашнее задание - разобрать алгоритм проверки серийного номера в этой программе ;).



Online-Check в действии

MC707 (mc707@mail.ru)

ЭФФЕКТИВНЫЙ ПАТЧИНГ

КОЕ-ЧТО О ТОМ, КАК МОЖНО ПАТЧИТЬ ПРИЛОЖЕНИЯ

Патчинг - это чуть ли не основной способ взлома программного обеспечения. Для того чтобы с его помощью убить в целевой программе ту или иную гадкую функцию (к примеру требование зарегистрироваться), достаточно найти некоторый код, отвечающий за эту функцию, и модифицировать его одним из возможных методов. О том, как найти этот код, и о том, какими методами его можно модифицировать, пойдет речь в этом материале.

ИЩЕМ, ЧТО ПАТЧИТЬ

Если посмотреть на проблему с точки зрения крэкера, можно сразу сориентироваться: код, который мы ищем - это любой код, отвечающий за регистрацию программы и ее нормальное функционирование. Существует несколько несложных приемов, для того чтобы найти его.

Первый, самый простой, распространенный и довольно эффективный - нахождение в коде строк, имеющих какое-либо отношение к регистрации. Благо сообщения об успешной/неуспешной активации программы, о количестве оставшихся до окончания срока функционирования дней, содержимое NAG-окна, записи в About и т.п. - все хранится в программе и, как правило, в виде plain-текста, поэтому их поиск в программе будет не очень сложен. После успешного нахождения строки остается только поймать в программе код, который использует ее. Это можно сделать либо с помощью любого дизассемблера, либо с помощью OllyDbg. Обычно такие строчки встречаются в коде в виде инструкций наподобие "mov eax, prog.004FB613" или "push prog.004FB613", где по адресу 004FB613 как раз и лежит искомая строка. Далее путем статического или динамического метода (визуальный метод или метод трассировки) определяется, является ли найденный код важным для взлома.

Второй прием, кстати, не менее эффективный - останов на API-функциях, вызываемых в критичных для взлома участках. Для его применения нужны довольно глубокие познания в области набора API-функций для конкретной версии Windows, поэтому перед употреблением советую хорошенько изучить MSDN последней версии. Для того чтобы воспользоваться этим приемом, нужно хотя бы примерно представлять себе, что делает программа, пытаясь стрясти с тебя некоторую сумму денег за регистрацию. Как правило, она просит ввести что-нибудь вроде имени/рег. кода/e-mail. В этом случае нужно ловить место регистрации по API-функциям GetDlgItem, GetDlgItemTextA, GetWindowTextA.

Если тебе повезло и ты поймал программу в процессе ввода серийника на одной из этих API, то, выйдя из дебрей системных библиотек и немного потрассировав код, ты, скорее всего, найдешь место проверки или какой-нибудь другой манипуляции введенных тобой данных. Можно также ловить место регистрации функциями ShowWindow, MessageBoxA, MessageBoxExA, MessageBoxIndirectA и недокументированной MessageBoxTimeoutA, отвечающими за выводы различных окошек с сообщениями. Соответственно, если выдаются сообщения вида "Вы ввели неправильный код" или что-то очень похожее, то, когда вылезешь из системных дебрей, посмотри на код, находящийся выше/раньше вызова этого сообщения, чтобы найти код, критичный для взлома.

Также программа может издавать характерный звук при выводе ошибки - тут можно попробовать отловить код на MessageBeep. В случае неудачи в первых двух случаях можно попробовать поискать места чтения/записи значений из реестра, так как программисты порой очень любят хранить там регистрационные данные своей программы. Здесь тебе помогут API RegOpenKeyA, RegQueryValueA, RegQueryValueExA, RegCreateKeyA, RegSetValueA и RegSetValueExA. При анализе данных, передаваемых в реестр, всегда есть вероятность, что ты наткнешься на критичный код. Данный способ немного муторный, так как программы обычно считывают множество параметров реестра, и чем больше программа, тем больше нагоняется трафика, который нужно анализировать. Кстати, иногда программисты записывают регистрационные данные в файл. Здесь все немного проще. Существует замечательная API CreateFileA, вызываемая всегда - как при открытии какого-либо файла, так и при его создании. Аналогично, анализируя параметры вызываемой CreateFileA, можно нарваться на место для будущей модификации.

Если же программа проверяет, запустили ее с оригинального диска или нет, то, как правило, бывает достаточно брякнуться на API GetDriveTypeA. Эта функция просто проверяет тип заданного диска (в данном случае диска, с которого запущена программа). Если возвращенное значение равно пяти, значит это CD/DVD-привод. После запуска этой функции должны идти разные проверки на соответствие метки диска, наличия какого-нибудь файла и т.п. Их и нужно патчить.

Естественно, это не все приемы поиска важного для взломщика кода - лишь основные. Подробности в этом номере журнала.

НАШЛИ? ПАТЧИМ!

Существует несколько методов патчинга. Результаты их применения не отличаются: в любом случае будут модифицированы одни и те же байты и программа перестанет напоминать о регистрации, однако реализация методов различается. Разберемся с каждым.

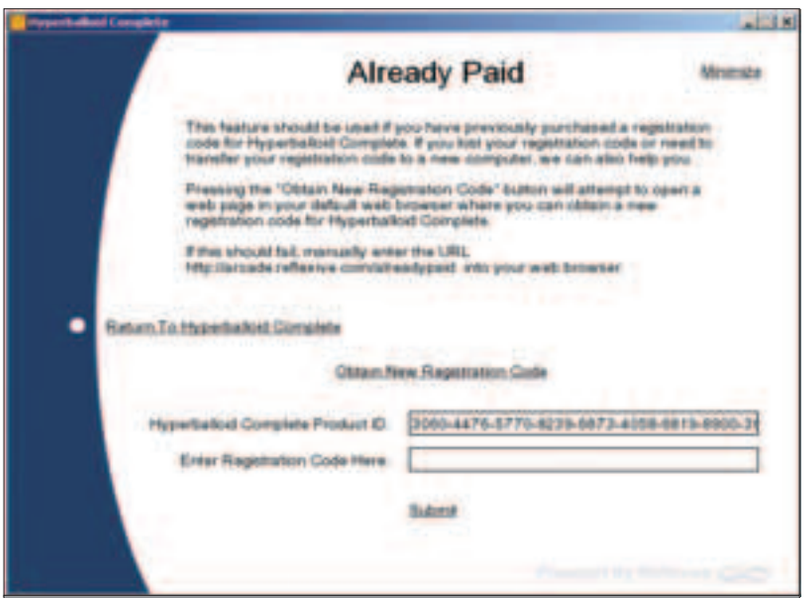
ПРЯМОЙ ПАТЧИНГ

Это самый простой и распространенный метод. Он заключается в простом модифицировании критичного кода. Есть, к примеру, запакованная программа, и ты хочешь написать к ней крэк. При использовании этого метода план действий должен быть таким:

1. Распаковать программу (это, думаю, не вызовет трудностей);
2. Найти код, ответственный за регистрацию;
3. Прямо в распакованной программе модифицировать найденный код определенным образом.

В итоге распакованная и модифицированная программа - это, по сути, и есть крэк. Вернее, программа, просто взломанная прямым патчингом. Как видишь, все зло сведено к минимуму, сделать такой крэк очень просто даже

NAG-окно игры



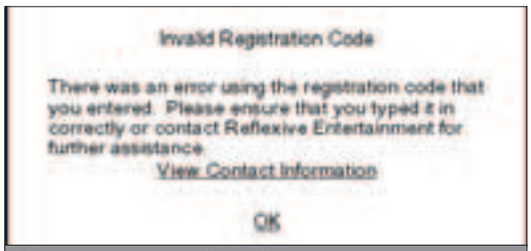
Ввод регистрационного кода

без особых затрат времени. Чтобы прояснить, как искать критичный код и как патчить его, разберемся со всем этим делом, как говорится, на живом примере. Исследуем и взломаем реальную программу - игру HyperBalloid Complete Edition 1.20, которую можно скачать с сайта www.reflexive.net. В процессе патчинга будем пользоваться только отладчиком OllyDbg. Запускаем программу и видим NAG-окно с любезным предложением зарегистрироваться и указанием количества минут, оставшихся от trial-периода.

Сразу же попытаемся отловить процедуру регистрации, поставив бряки на описанные в начале статьи API-функции. Итак, жмем на кнопку Already Paid в NAG'е и видим окно с приглашением ввести регистрационный код.

Переходим в отладчик и ставим точки останова сразу на все указанные API: GetDlgItem, GetDlgItemTextA, GetWindowTextA, MessageBoxA, MessageBoxExA, MessageBoxIndirectA, MessageBoxTimeoutA, ShowWindow, вводя bp <имя_API_функции> в поле Command. Введем какой-нибудь, неважно какой, серийник, нажмем Submit, и, как это ни странно, увидим сообщение - якобы неправильно набран номер ;).

Отсюда сделаем вывод, что, если мы не остановились ни на одной из функций, то в игре используются иные методы взятия введенной информации и вывода результата. Что ж, не будем отчаиваться. Перезапустим программу и пойдем по первому указанному мной методу - посмотрим наличие строк в коде, имеющих отношение к регистрации. Нажав правой кнопкой мыши по любому участку кода и выбрав пункт Search for->All referenced text strings, ты сможешь увидеть окно со списком всех строк, встречающихся в программе, и с информацией о коде, который использует эти строки. Честно говоря, найти строки из NAG-скрина вряд ли повезет. При размере exe-файла 144 Кб вряд ли



Введен неправильный серийник

в нем будут находиться процедура регистрации и сам код игры. Скорее всего, в этом случае весь код вынесен из основного модуля приложения в динамически подгружаемые библиотеки.

Итак, анализируя выведенные строки (благо из-за размера exe-файла их там не очень много), я наткнулся на подозрительную:

```
0040631B PUSH game.0041DAA8 ASCII
                "radll_HasTheProductBeenPurchased"
```

Очень похоже на вызов функции из библиотеки, проверяющий, приобретена ли программа. Поставим точку останова на этот PUSH, то есть на адрес 0040631B, выделив строку и нажав <F2>. Запустим игру по <F9> и, как это ни странно, еще до появления каких-либо окон остановимся на этом адресе. И вот показался очень важный код.

Не нужно быть reverse engineer'ом, чтобы, взглянув на инструкцию call esi и на

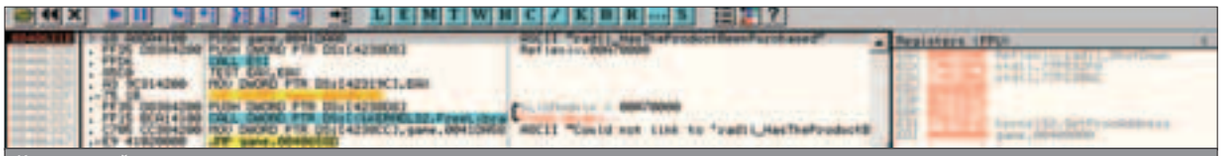
```
esi = 77E7B332 kernel32.GetProcAddress,
```

сообразить, что из какой-то библиотеки берется адрес функции radll_HasTheProductBeenPurchased и он записывается в некоторую переменную по адресу 0042319C. Если посмотреть на строку Reflexiv.00A70000, можно сделать вывод, что эта функция берется из библиотеки ReflexiveArcade.dll. Ее мы обнаружим в папке игры в директории ReflexiveArcade.

Чтобы отучить игру от вредной привычки просить зарегистрироваться, достаточно пропатчить функцию с длинным названием в найденной библиотеке так, чтобы она все время утверждала, что программа успешно зарегистрирована. Но зачем патчить DLL, если можно пойти более изящным путем: просто записать по адресу 0042319C адрес не radll_HasTheProductBeenPurchased, а адрес своей функции, которая всегда возвращала бы единицу, означающую, что игра зарегистрирована.

Первое, чего нам в этом случае не хватает - это своя функция. Нет ничего проще! Берем бегунок прокрутки и прокручиваем код программы в самый низ, пока не встретим там пустое место для записи нашего кода.

Находим его довольно быстро - нули начинаются с адреса 004198AE. Чтобы было проще запомнить, спустимся еще чуть ниже до адреса, кратного »



Критический код игры

Ищем пустое место для записи нашего кода

100h - 00419900, который и сделаем адресом нашей функции. Выделим стоку 00419900 и нажмем пробел для ввода кода по этому адресу. Вобьем в появившемся окошке mov eax, 1 и нажмем <Enter>. В регистре eax, как ты и сам знаешь, обычно содержится значение, возвращаемое функцией. В данном случае этим значением может быть только 1. Так как мы пишем функцию, а не просто кусок кода, мы должны позаботиться о том, чтобы код вернулся на то место, откуда был запущен. Поэтому нужна еще одна инструкция - ret. Вбиваем ее и жмем <Enter>.

Все. Нажмем Cancel для отмены дальнейшего ввода кода. Получена мини-функция из шести байт. Теперь вернемся к месту, где записывался адрес функции radll_HasTheProductBeenPurchased. Для этого выделим в окне регистров EIP, тыкнем по нему правой кнопкой мыши и выберем Origin. Окажемся по адресу 0040631B. В принципе, весь местный код нужно вырезать совсем: нам ни на что не сдался этот GetProcAddress. Поэтому, стоя на адресе 0040631B, нажмем пробел и введем MOV EAX,419900, то есть подставим вместо оригинального адреса функции свой. Остальные команды нам не нужны, поэтому вводим далее инструкции nop до адреса 00406329 включительно.

Нам остается только сохранить все изменения в программе и протестировать ее. Выделяем весь код с 00401000 по 00419FFF, выбираем в контекстном меню Copy to executable-> Selection и указываем в появившемся окне файл, куда хотим сохранить пропатченную версию игры. После этого можно закрывать отладчик и пробовать запустить игру. Вуаля! Она прекрасно запустилась и, обращаю на это твое внимание, без всяких приглашений зарегистрироваться. При выходе из игры нас мило благодарят за приобретение.

"Нет, нет, что вы! Вам спасибо". Сделаем некоторые выводы. Мало того, что лентяи программисты из Reflexive не делают дополнительных проверок, так они еще и называют экспортируемые (!) функции из dll как radll_HasTheProductBeenPurchased. Крайне безответственно с их стороны. Ну что ж, их лень - наши сэкономленные деньги.

Кстати, не могу не заметить, что подобным образом ломается любая игра с сайта www.reflexive.net.

ПАТЧИНГ ЗАГРУЗЧИКОМ

Малораспространенный и не самый авторитетный метод, но он реализуется довольно просто. Используется, как правило, для программ, запакованных чем-нибудь хитрым, например протектором. Нет смысла писать загрузчики для программ, не запакованных вообще, а для запакованных пакерами проще сделать прямой патчинг, то есть распаковать, или, на худой конец - inline-патч.

Суть метода заключается в следующем: уже после того, как был обнаружен код для патчинга, пишется некоторая специальная утилитка-загрузчик (лоадер), которая запускает программу, ждет, пока она распакуется, и проверяет код на целостность, после чего патчит код. Использование этого метода позволяет, во-первых, обходить разнообразные противные проверки, работающие в начале программы, а во-вторых, уменьшить размер крэка до минимума.

Но написание лоадера - занятие довольно неблагодарное. Мало того, что тут требуются знания работы ОС, основы управления памятью и умение программировать, так еще и результат может работать совершенно как ему заблагорассудится (глючить). Это связано с тем, что лоадер может модифицировать байты программы, когда протектор еще не распаковал основной код, что очень критично для стабильной работы программы. Скорее всего, программа просто упадет с критичной ошибкой. В связи с этим не советую увлекаться лоадерами. Правда, иногда деваться некуда, и проще написать загрузчик, чем возиться с другими методами. Есть несколько способов реализации лоадеров, во врезке я привел самый простой и распространенный.

В нем, я думаю, тебе будет все понятно. Просто запускаем процесс с помощью CreateProcess и модифицируем после некоторой паузы и нескольких проверок код программы,

"Спасибо за регистрацию"

Пропатченный код

ответственный за регистрацию с помощью WriteProcessMemory. Кстати, по такому же принципу пишутся трейнеры к играм.

На случай если писать loader очень лень, но очень нужно, существуют автоматизированные loader-мэйкеры, которые по специальному скрипту создают полноценный загрузчик. Останется лишь указать имя программы и то, по каким адресам и какие байты нужно модифицировать - все! Loader-мэйкер сделает по этому сценарию loader, который можно будет запустить с чувством собственного достоинства.

INLINE-ПАТЧИНГ

Это довольно сложный метод. Возможно, он чем-то напомнит тебе прямой патчинг, но они схожи только по подходу к проблеме, а отличаются и принципами работы, и большинством случаев в области применения: inline-патчинг делается для программ, круто обработанных пакерами/протекторами, которые, вероятно, даже невозможно распаковать с ходу. Смысл метода в том, что в место программы, которое передает управление на OEP (на оригинальный Entry Point), то есть в место, получающее управление, когда вся программа уже распакована, встраивается код, который уже будет модифицировать некоторые байты, отучать от регистрации - собственно, производить взлом. Отработав свое, патчащий код возвращает управление на OEP уже взломанной программе. Для чего нужен этот метод? Ну, хотя бы для того, чтобы уменьшить размер крэка. Если ломаешь программу прямым патчингом, приходится раздавать крэк либо в виде взломанного exe'шника, что ужасно, особенно если вспомнить многомегабайтные Delphi-монстры, либо в виде программы-патча вместе с распаковщиком, что тоже, скорее всего, будет весить немало. Сложность метода inline-патчинга, как ты понимаешь, заключается в том, чтобы вычислить адрес прыжка на OEP и грамотно создать код, модифицирующий программу. Подробно о том, как реализуется этот метод, вместе с его примерами читай на www.cracklab.ru.

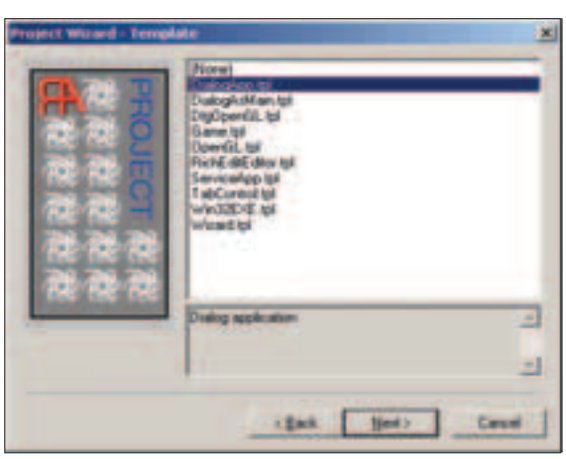
НА ДОРОЖКУ

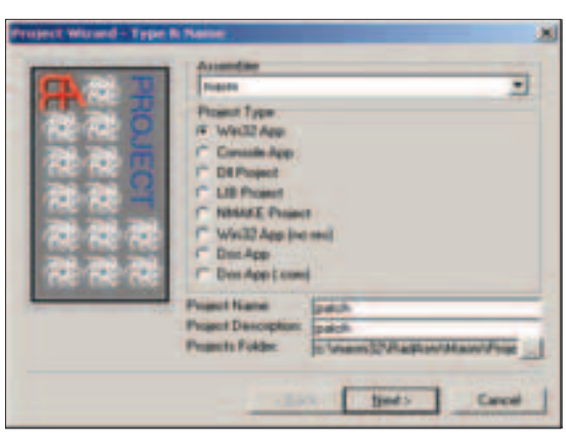
Как видишь, модифицировать найденный критичный код - это далеко не такая очевидно решаемая задача, как может показаться. Можно решать ее разными способами, и каждый из них представит особый интерес и принесет особую пользу. Вопрос о том, какой способ выбрать, можно решить только на конкретном деле. Все будет зависеть от сложности системы защиты, крутости пакера/протектора и т.п.

На этом я завершаю свой опус. Если возникли вопросы, пиши - постараюсь помочь. Удачного патчинга!

ИСХОДНЫЙ КОД ЗАГРУЗЧИКА

```
.data
; заголовок окна с сообщением об ошибке
Msg             db "Fatal Error", 0
; сообщение об ошибке
Error           db "Program not found",0
; имя файла программы
program     db "victim.exe",0
; записываемый в память процесса байт
write_buffer db 90h
; адрес, по которому будет
; осуществляться считывание/запись
check_addr  DWORD 401050h

.data?
; переменная, в которую производится
; считывание байта процесса
buffer          dw ?
; структура информации о процессе
process_info PROCESS_INFORMATION <>
; структура информации о параметрах
; создающегося процесса
startup_info  STARTUPINFO <>

.code
start: ;начало программы
; запускаем нужные нам программы.
invoke      CreateProcess,addr program, NULL, NULL, NULL, FALSE,
            CREATE_NEW_CONSOLE OR NORMAL_PRIORITY_CLASS, NULL, NULL, addr startup_info, addr process_info
; если результат выполнения равен 0,
; то программа не найдена и не запустилась
.if eax == 0
; информируем об ошибке
invoke      MessageBox, NULL, addr Error, addr Msg, MB_OK
; и выходим
invoke      ExitProcess,0
.endif
; главный цикл
.while true
; считываем память процесса по
; адресу check_addr в буфер buffer размером в 1 байт
invoke      ReadProcessMemory, rocess_info.hProcess, check_addr, addr buffer, 1, NULL
; проверка на успешность считывания
.if eax != 0
; проверка на распакованность
; программы по этому адресу
.if bufr != 00h
;ждем проверку целостности кода
invoke      Sleep,300
; приостанавливаем процесс
invoke      SuspendThread, addr process_info.hThread
; записываем 1 байт write_buffer
; по адресу check_addr
invoke      WriteProcessMemory, process_info.hProcess, check_addr, addr
write_buffer, 1, NULL
; продолжаем выполнение программы
invoke      ResumeThread, addr process_info.hThread
; закрываем хэндл процесса и
; завершаем свой процесс
invoke      CloseHandle, process_info.hThread
invoke      ExitProcess, 0
.endif
.endif
.endw
; конец кода
end start
```





GL#OM (gl00m-crk@yandex.ru)

КЕЙГЕН СВОИМИ РУКАМИ

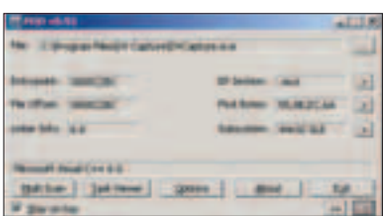
ИССЛЕДОВАНИЕ ПРОГРАММЫ MOOGEAR DV CAPTURE 1.0

Ни для кого не секрет, что статей на русском языке на тему создания ключегенераторов к программам, защита которых основана на криптоалгоритмах, ничтожно мало. Собственно, этот факт и побудил меня написать эту статью. А в качестве жертвы был выбран MooGear DV Capture v1.0.

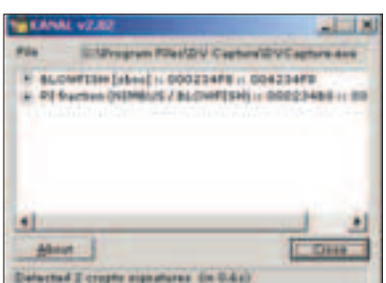
Сразу предупрежу, что я не собираюсь вдаваться в подробности каждого этапа работы криптоалгоритма, так как об этом написано немало отличных книг (например "Прикладная криптография" Брюса Шнайера). С их прочтения советую начинать. Плюс без опыта исследования программных защит и создания ключегенераторов (к простым защитам) тоже будет трудно осмыслить все нижесказанное. Для полноценной работы понадобятся PEiD, OllyDbg, IDA и MASM32. Теперь приступим к делу.

Цель нашего исследования, как показал PEiD, написана на Microsoft Visual C++ 6.0. Это хорошо, потому что компилятор данного языка генерирует более компактный и легче распознаваемый код, в отличие от того же Delphi, код которого переполнен тоннами ненужных проверок и процедурами, при виде вложенности которых меня охватывает ужас.

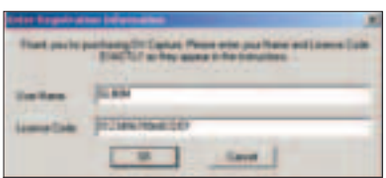
Не помешает также применить к нашей жертве какой-нибудь поисковик криптосигнатур. Лучшим, на мой взгляд, является KANAL (PEid plugin),



PEiD в деле



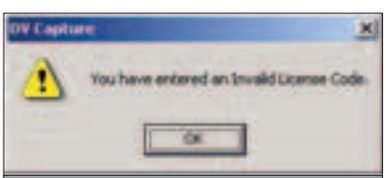
KANAL - отличный поисковик криптосигнатур



Форма регистрации программы

поэтому воспользуемся именно им... Хех, найдено две сигнатуры, и обе относятся к алгоритму Blowfish.

В таких случаях, то есть если обнаруживается какой-либо криптоалгоритм, я обычно сразу же загружаю файл в IDA и от адреса, указанного анализатором, выхожу на процедуру регистрации, попутно распознавая и называя элементы (процедуры, их параметры и переменные) криптоалгоритма более понятными именами. Но такой подход не всегда уместен... Например, этот криптоалгоритм может вообще не использоваться при проверке ключа, а быть лишь для нашего устрашения или использоваться какой-нибудь процедурой программы, совершенно не относящейся к регистрации. Поэтому поступим иначе, а точнее "дедовским" способом =). Запустим программу и откроем форму регистрации. Вводим любую чушь в поля регистрации и нажимаем OK.



Сообщение о неверном лицензионном коде

Как и следовало ожидать, мы ввели неверный серийный номер, на что и получили соответствующее сообщение.

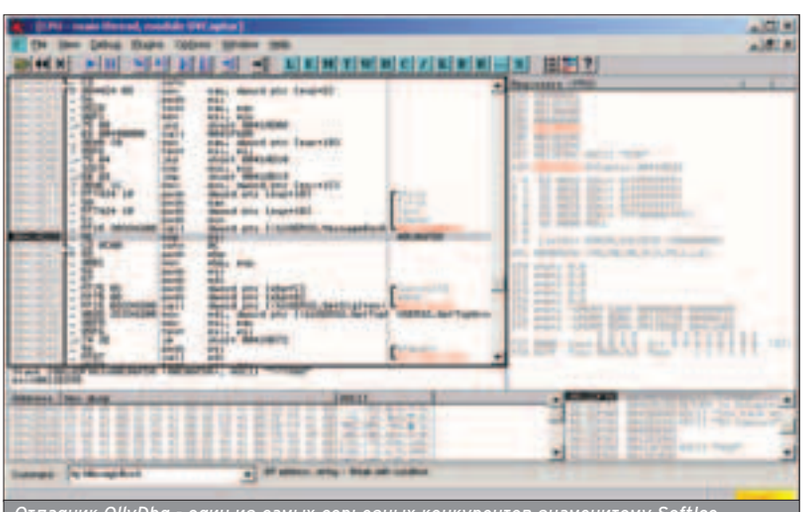
Как все уже заметили, это обычное сообщение об ошибке, и, вернее всего, оно вызывается посредством стандартной функции, а именно - MessageBoxA. Поэтому загрузим нашу цель в OllyDbg и поставим точку останова на эту API.

Korga OllyDbg остановится, нажимаем <Ctrl>+<F9>... Мы тут:

```
push dword ptr [esp+10h] ; Style
push eax ; Title
push dword ptr [esp+10h] ; Text
push ecx ; hOwner
call dword ptr [<&USER32.MessageBoxA> ; \MessageBoxA
pop esi
retn 0Ch
```

Это лишь процедура показа сообщения... Проходим ret:

```
push 30h
```



Отладчик OllyDbg - один из самых серьезных конкурентов знаменитому SoftIce

```
push 0042B334h; "DV Capture"
push 0042BAA0h; "You entered an Invalid License Code."
mov ecx, ebp
call_MessageBoxA
Cmov ecx, dword ptr [esp+1Ch] <= мы тут
```

Теперь попробуем проанализировать, какие же действия произвела программа для проверки действительности введенных данных. Для этих целей больше подойдет дизассемблер IDA, поэтому загрузим нашу цель в него и перейдем на начало данной процедуры.

С первых же строк мы замечаем следующее:

```
 mov eax, [ecx-8]; EAX = длина введенного имени
cmp eax,
jge short loc_409AAE
.push 30h
push offset aDvCapture ; "DV Capture"
.push offset aYouMustEnterAU ; "You must enter a User
Name before you c"...
mov ecx, ebp
call_MessageBoxA
```

❶. Длина имени должна быть больше единицы или равна ей.

```
mov eax, [esi-8]; EAX = длина введенного серийного
                                              номера
cmp eax,1
jge short loc_409AF3
push 30h
push offset aDvCapture ; "DV Capture"
push offset aYouMustEnterAL ; "You must enter a
License Code before yo"...
jmp loc_409E24
```

❷. Длина серийного номера должна быть больше единицы или равна ей.

Если предыдущие проверки пройдены успешно, то далее мы увидим немного странные операции над длиной серийного номера и следующие за ними проверки, поэтому я прокомментирую их более подробно:

```
mov ecx, eax ; ECX = длина введенного серийного
                                              номера
.text:00409AF5 and ecx, 80000001h ; проверка на
                                              четность
jns short loc_409B02 ; если нет знака (у нас его быть
                                              не может)
dec ecx
orecx, 0FFFFFFFEh
inc ecx
loc_409B02: ; CODE XREF: sub_409A10+EB j
jnz loc_409E18 ; если не четно, то на ошибку
mov edx, eax ; EDX = длина введенного серийного
                                              номера
 and edx, 80000007h ; проверка делимости на восемь
                                              без остатка
jns short loc_409B17 ; проверка на знак
dec edx
oredx, 0FFFFFFF8h
inc edx
loc_409B17: ; CODE XREF: sub_409A10+100 j
jnz loc_409E18 ; если не делится, то на ошибку
cmp eax, 10h
jlloc_409E18
```

❸. Длина лицензионного кода должна быть четной, делиться на 8 и быть больше 15.

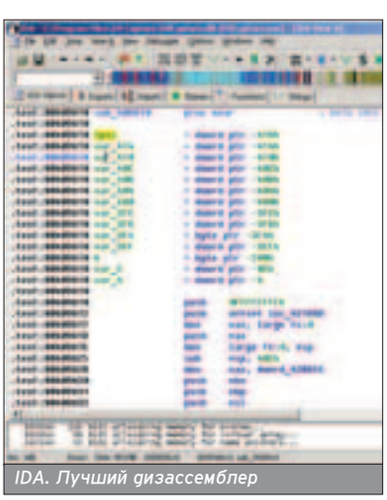
Например:

```
User Name: GL#0M
License Code: 0123456789ABCDEF
```

Если все условия соблюдены, то:

```
mov eax, [esi-8]; eax = 16 (длина лицензионного кода)
cdq
sub eax, edx
sar eax, 1; беззнаковое деление на 2 ;)
push eax ; cbeax = 8
lea eax, [esp+42Ch+b]
push eax ; lpb eax = указатель на буфер результата
push esi ; lps esi = указатель на лицензионный код
callHexFromHexStr
```

Компилятор Visual C++ генерирует более компактный код, чем Delphi.



IDA. Лучший дизассемблер

HexFromHexStr - преобразует указанную часть строки шестнадцатеричных цифр в бинарный вид. Преобразование происходит по два байта, поэтому третий параметр равен частному от деления длины лицензионного кода на два. Вот ее код:

```
push ebx
mov ebx, [esp+cb] ; EBX = длина, деленная на 2
push esi
xor esi, esi
test ebx, ebx
jle short loc_401F75
push ebp
mov ebp, [esp+8+lpb]; EBP = указатель на буфер
                                              результата
push edi
mov edi, [esp+0Ch+lps]; EDI = указатель на буфер дан-
ных
 CODE XREF: HexFromHexStr+31 j
lea eax, [esp+0Ch+cb]; EAX = указатель на буфер
                                  для байта результата
push eax
push edi
call sub_401E80; преобразует два байта строки
                                              в hex-число
mov cl, byte ptr [esp+14h+cb] ; CL = результирующий
байт
add esp, 8
mov [esi+ebp], cl; заносим его в буфер результата
inc esi
add edi, 2
cmp esi, ebx
jlshort loc_401F56
pop edi
pop ebp
loc_401F75: ; CODE XREF: HexFromHexStr+A j
pop esi
pop ebx
retn
```

Дальше идет собственно то, ради чего я все это затеял, поэтому приведу теоретические выдержки из книги "Прикладная криптография".

Blowfish - это 64-битный блочный шифр с ключом переменной длины.

Алгоритм включает два этапа:

❶. SetKey - развертывание ключа;

❷. Encrypt/Decrypt - шифровка/дешифровка данных.

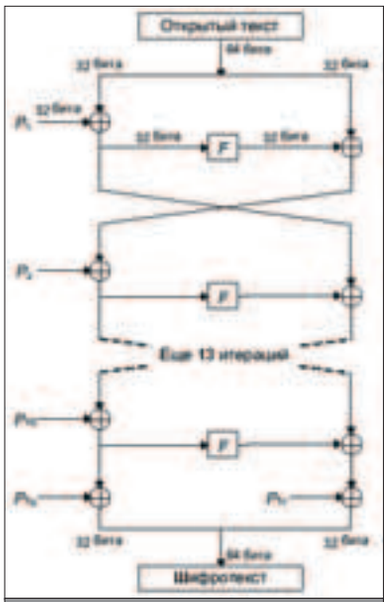
Развертывание ключа преобразует ключ длиной до 56 байт в несколько массивов подключей общим объемом 4168 байт.

Шифрование данных состоит из простой функции, последовательно выполняемой 16 раз. Каждый этап состоит из зависимой от ключа перестановки и зависимой от ключа и данных подстановки. Используются только сложения и XOR двойных слов. Единственными дополнительными операциями на каждом этапе являются четыре извлечения данных из индексированного массива.

В Blowfish используется много подключей. Так называемые массивы PBox и SBox. Массив PBox состоит из 18-ти подключей (двойных слов). Каж-

»

"Прикладная криптография". Брюс Шнайер



Blowfish

дый из четырех массивов SBox содержит 256 подключей (двойных слов).

Дешифрование выполняется точно так же, как и шифрование, но подключи PBox используются в обратном порядке.

Для дальнейшего понимания кода, я думаю, достаточно. Теперь мы без труда можем распознать Blowfish, а уж если вооружиться его исходным кодом, то тем более ;). (Исходные коды Blowfish на C++ найдешь на диске.)

Продолжим...

```
push 1058h
call Alloc
mov edx, eax; edx = указатель на выделенную память
add esp, 10h
```

Странный размер... Ничего не напоминает =)? Правильно! Эта память будет предназначена для массивов подключей развернутого ключа. Сразу предупрежу, что этого выделения памяти здесь могло и не быть, это лишь частный случай.

Далее мы можем видеть обычный для программ, написанных на Microsoft Visual C++, способ получения длины строки.

```
lea ecx, dword ptr [esp+24h]; ecx = указатель на
                              вспомогательный буфер
or esi, 0FFFFFFFFh
push ecx ; 3 параметр
mov edi, _KEY ; edi = "A02DD91A-C700-47b7-82D8-
10E68082B4C0"
mov ecx, esi
xor eax, eax
repne scas byte ptr es:[edi]
not ecx
dec ecx ; длина строки = 36 ;)
mov dword ptr [esp+28h], ebx; 1 dword буфера = ebx = 0
push ecx ; 2 параметр
push_KEY; 1 параметр
mov ecx, edx; ecx = указатель на выделенную память
mov dword ptr [esp+34h], ebx; 2 dword
                    вспомогательного буфера = ebx = 0
call Blowfish_SetKey
mov edi, eax; edi = указатель на развернутый ключ
```

❶. A02DD91A-C700-47b7-82D8-10E68082B4C0 - это не что иное, как ключ шифрования ;).

❷. Вспомогательный буфер используется в качестве счетчика циклов в данной реализации Blowfish.

❸. Blowfish_SetKey - функция развертывание ключа.

Почему я решил, что это именно Blowfish_SetKey? Естественно, не с потолка взял =). Пришлось пройти ее всю под отладчиком и понять, что она делает. А иначе никак... Хотя в данном случае есть некоторые моменты, которые мне хотелось бы выделить:

```
mov ebx, [esp+54h+keylength]; EBX = длина ключа
.............
cmp ebx, 1
.............
jnb short loc_40104F
lea eax, [esp+54h+zerobuf]
lea ecx, [esp+54h+var_44]
push eax
mov [esp+58h+zerobuf], offset aIncorrectKeyLe ;
"Incorrect key length"
call??0exception@@QAE@ABQBD@Z ; exception::except-
tion(char const * const &)
```

Текст ошибки ("Incorrect key length") сразу же выдает назначение данной функции. Наши догадки подтверждает следующий код:

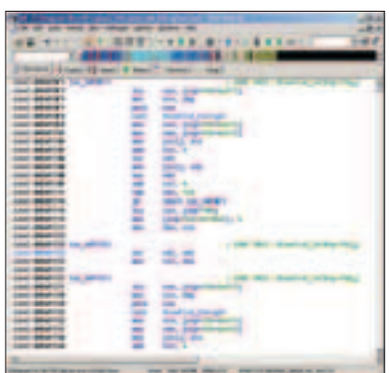
```
cmp ebx, 38h<= сравнение длины ключа с 56
jbe short 00401059h
mov ebx, 38h
```

Вспоминаем фразу "Преобразует ключ длиной до 56-ти байт" ;).

В недрах функции мы можем увидеть работу с массивами, до боли напоминающими PBox и SBox =).

```
mov ecx, 12h ; "PBox состоит из 18 подключей"
mov esi, offset PBox
mov edi, edx
mov [esp+5Ch+keylength], 12h
rep movsd
ea edi, [ebp+58h]
mov ecx, 400h; "Каждый из четырех массивов SBox
                              содержит 256 подключей"
mov esi, offset SBox
rep movsd
```

Также можно заметить внутри функции два цикла с участием довольно объемной функции - это Blowfish_Encrypt. Наличие этих циклов также обязательно для Blowfish_SetKey.



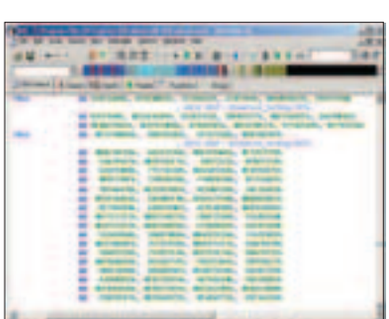
Часть функции Blowfish_SetKey

Одним словом, нет сомнений, что это именно функция развертывания ключа.

Что дальше? А дальше, как уже все догадались, идет Blowfish_Encrypt... Или Blowfish_Decrypt? Хех, смотрим:

```
push ebx ; 4 параметр (ebx = 0, флаг режима)
lea ecx, dword ptr [esp+22Ch] ; ecx = указатель на
                        лицензионный код (бинарный вид)
mov eax, dword ptr [edx-8h] ; eax = 16 (длина лицензи-
онного кода)
cdq
sub eax, edx
sar eax, 1h
push eax ; 3 параметр (eax = 8)
lea eax, dword ptr [esp+3Ch]; eax = указатель на буфер
результата
50 push eax ; 2 параметр
```

Дешифрование выполняется так же, как и шифрование, но подключи PBox используются в обратном порядке.

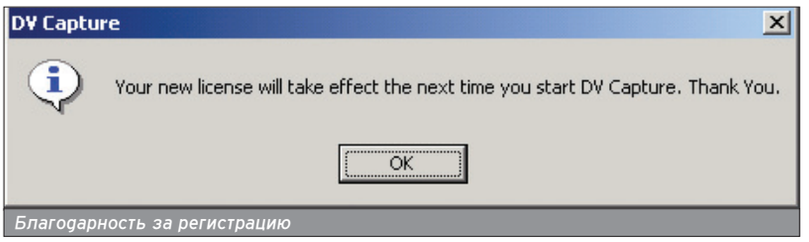


Массив PBox и часть первого массива SBox

```
push ecx ; 1 параметр
mov ecx, edi; ecx = указатель на развернутый ключ
call Blowfish_DecryptMode
```

На вход подается указатель на введенный нами лицензионный код (преобразованный функцией HexFromHexStr в бинарный вид). Явный Decrypt =). Хотя, конечно, не помешает убедиться... (Все по аналогии с Blowfish_SetKey.)

Кстати, ты мог бы подумать, почему я назвал функцию Blowfish_DecryptMode, а не



Благодарность за регистрацию

ны. Дело в том, что Blowfish_DecryptMode - это функция, содержащая несколько разных режимов Blowfish_Decrypt. За выбор отвечает четвертый параметр функции Blowfish_DecryptMode, у нас он равен 0, что соответствует стандартному режиму. В этом тоже пришлось разобраться =).

Ну вот мы и подошли к завершающей стадии.

```
mov eax, dword ptr [esp+14h]; User Name
mov ecx, dword ptr [esp+18h]; Расшифрованные данные
push eax
push ecx
calll strcmpi
add esp, 8h
test eax, eax
push edi
jnz short _WrongSerial
```

Здесь мы видим сравнение введенного User Name с расшифрованными данными, то есть, для того чтобы регистрация прошла успешно, нужно, чтобы они были равны. И что теперь? Догадался? Правильно! Нужно зашифровать наше имя. Полученный результат будет считаться действительным лицензионным кодом.

Для этого нам необходима функция Blowfish_Encrypt. Где ее взять? Вариантов море...

Вот некоторые из них:

❶. Написать самостоятельно - это полезнее ;).

❷. Взять прямо из кода нашей цели. В этом нам поможет IDA, а точнее ее функция сохранения дизассемблерного листинга в asm-файл. Для этого нужно выделить нужный участок и нажать комбинацию клавиш <Alt>+<F10>.

❸. Взять из кода Blowfish_Decrypt и видоизменить ее так, чтобы получился Blowfish_Encrypt.

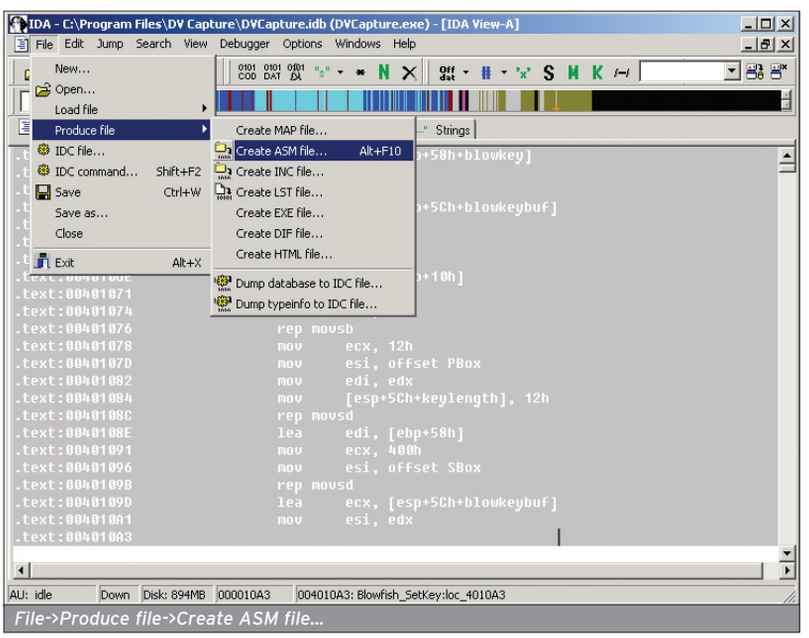
Это несложно - нужно только применить смекалку. Ответ кроется в теории этого алгоритма ;).

❹. Взять готовую реализацию на каком-либо языке.

Я выбрал второй вариант. Пришлось даже исправить баг Blowfish??crypt, из-за которого правильно шифровались данные только до девяти байт =). В общем, смотри исходники!

Все! Надеюсь, этот опус хоть как-то может помочь в освоении интересной темы Reverse Engineering.

Эта память будет предназначена для массивов подключей развернутого ключа.



File->Produce file->Create ASM file...

Крис Касперски aka мыщъх

ПРИМЕР ВЗЛОМА: WINRAR

НА ПРОСТОМ ПРИМЕРЕ УЧИМСЯ ВЗЛОМУ ПРИЛОЖЕНИЙ

Практическую часть обучения взлому программ нужно начинать с чего-то простого, возможно, даже банального. Архиватор WinRAR – идеальный выбор. Его несложная защита лучше всего подойдет для проверки твоих свежеприобретенных крэкерских навыков.

Только что скачанная версия WinRAR'a нормально работает 40 дней, после чего начинает вопить, как ненормальная, о регистрации, выплевывая противный NAG-screen через несколько секунд после запуска. Это очень раздражает, и возникает естественное желание отломать NAG.

Мы научимся взламывать версию 3.42 - последнюю стабильную на момент работы над номером, на которую ведет ссылка www.rarsoft.com/rar/wrar342.exe. Все остальные будут ломаться аналогичным образом, разве что только смещения "защитных" байт будут другими.

Помимо самого архиватора-жертвы, нам понадобится любой нормальный HEX-редактор (например HIEW), API-шпион Kerberos, дизассемблер IDA Pro и редактор ресурсов (Microsoft Visual Studio подойдет). В разных версиях HIEW раскладки горячих клавиш отличаются, поэтому, чтобы не создавать путаницы, определимся: мы будем использовать бесплатную версию 6.04 без функциональных ограничений. Последние версии этого редактора распространяются на коммерческой основе, а коммерция и хакерство несовместимы!

ПОДАВЛЕНИЕ NAG'A

Все диалоги в системе, сам понимаешь, выводятся не сами по себе, а с помощью некоторых API-функций. Если нам удастся перехватить функцию, выводящую NAG, мы сможем дизассемблировать защитный код, который вызывает ее, и проанализировать условия, из-за которых на экране появляется приглашение зарегистрироваться.

Но функций, связанных с диалогами, множество: это и CreateDialog, и DialogBox, и MessageBox, и целая куча других. Какую из них использовал разработчик RAR'a? Чтобы не гадать, воспользуемся API-шпионом. Он все покажет. Только сначала настроим фильтр, чтобы Kerberos отбрасывал малоинформативные API-вызовы, захламляющие файл отчета. Открываем ke_spy.txt и комментируем следующие функции (достаточно перед их именами поставить знак ';'): TlsGetValue, DefWindowProcA, DispatchMessageA, GetFocus, GetMessageA, SendMessageA, SendMessageW, TranslateAcceleratorA, TranslateAcceleratorW и TranslateMessage. Для улучшения фильтрации имеет смысл зайти в "Опции" (кнопка Options) и взвести флажок report only .exe calls, чтобы собирать API-вызовы только из winrar.exe, но не из загружаемых им DLL. Если этого не сделать, ничего страшного не произойдет, но файл отчета получится слишком большим и удручающе ненаглядным.

Теперь нажимаем Browse, указываем путь к нашему архиватору и давим Inject. Дождавшись появления NAG'a на экране, выходим из rar'a и открываем файл отчета WinRAR.rep, находящийся в одном каталоге с exe'шником.

Фрагмент файла отчета:

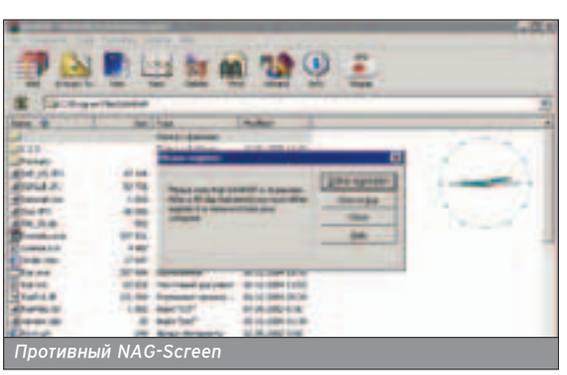
```
WinRAR.exe|0044B030|LoadAcceleratorsA(00400000,
00496BA8: "VIEWACC") returns: 001E006F

WinRAR.exe|00440F73|DialogBoxParamA(400000,495FE1:"
REMINDER",70094,444FF4,0) returns:0

WinRAR.exe|00440F9B|WaitForSingleObject(00000110,
0000000A) returns: 00000102
```

Изучение файла-отчета лучше начинать с конца (так как NAG-screen появляется в последнюю очередь, когда основной интерфейс уже инициализирован). Только слепой не обнаружит вызов функции DialogBoxParamA, создающей диалог с грозным именем "REMINDER" (то есть "напоминатель"). Это и есть наш NAG!

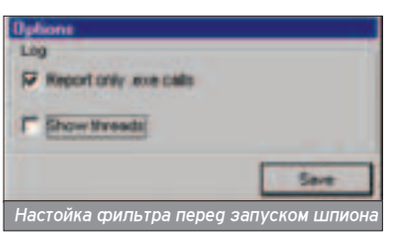
Kerberos (вот умница!) даже сообщает адрес возврата из функции - 440A73h, ведущий прямо к защитному коду. Заглянем сюда дизассемблером? Загружаем winrar.exe в IDA PRO и давим <G> (Jump to address), "440A73", <Enter>.

Тут отчетливо виден вызов DialogBoxParamA, выше которого находится следующий код:

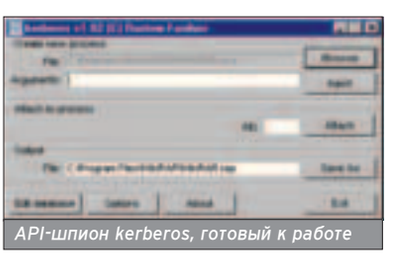
```
00440F1D    cmp dword_4B3A90, 0
00440F24    jnz short loc_440F73
00440F26    cmp byte_495A60, 0
00440F2D    jnz short loc_440F73
00440F2F    cmp byte_4B7E00, 0
00440F36    jnz short loc_440F73
00440F38    cmp byte_49F9BC, 0
00440F3F    jnz  loc_440F73
00440F41    mov eax, dword_004B43C8
00440F46    cmp eax, 28h
00440F49    jg short loc_440F4F
00440F4B    test eax, eax
00440F4D    jge short loc_440F73
00440F4F loc_440F4F:
            ; CODE XREF: sub_4408C8
00440F4F    mov byte_495A60, 1
00440F56    push 0 ; dwInitParam
00440F58    push offset sub_444FF4 ; lpDialogFunc
00440F5D    push dword_4B161C ; hWndParent
```



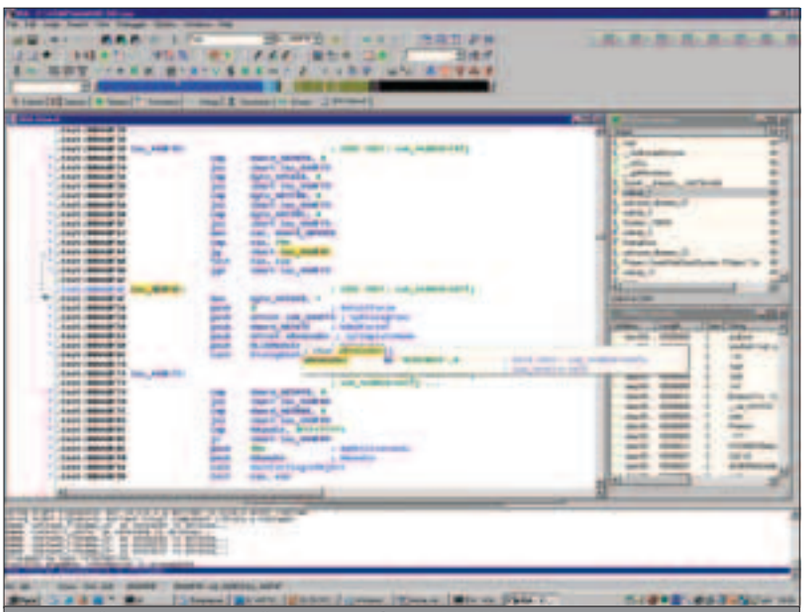
Противный NAG-Screen



Настойка фильтра перед запуском шпиона



API-шпион kerberos, готовый к работе



Защитный код, исследуемый в дизассемблере IDA PRO

```
00440F63      push offset aReminder ; lpTemplateName
00440F68      push hLibModule ; hInstance
00440F6E      call DialogBoxParamA
00440F73 loc_440F73:
                                    ; CODE XREF: sub_4408C8
00440F73      cmp dword_4B3A90, 0
```

Можно заметить, что функция DialogBoxParamA вызывается тогда, когда выполняется условный переход: cmp eax, 28h/jg loc_440F4F (прыжок, если eax > 28h). В десятичной системе 28h равно 40. Это и есть срок демонстрационного периода, положенный нам по праву. Здесь сразу становится ясен "физический" смысл переменной dword_004B43C8: она содержит количество дней, прошедших с момента установки программы.

В общем, можно радоваться! Штабквартира защитного механизма найдена! Как мы будем действовать теперь? Чтобы заблокировать NAG, можно, например, изменить cmp eax,28h (83 F8 28) на xor eax,eax/nop (33 C0/90). В этом случае eax всегда будет равен нулю независимо от того, какой день сейчас за окном. Команда nop нужна тут для того, чтобы скомпенсировать уменьшение длины инструкции (cmp занимает три байта, а xor - только два).

Запускаем HIEW, загружаем winrar.exe, дважды нажимаем на <ENTER>, чтобы перейти в ассемблерный режим, давим <F5> (goto) и пишем ".440F46" - адрес инструкции cmp. Точка здесь затем, чтобы сообщить HEX-редактору, что это именно адрес, а не смещение в файле. Нажимаем <F3> для перехода в режим редактирования (edit), а затем <ENTER> для ввода ассемблерной инструкции. В появившемся диалоговом окне пишем "xor eax,eax" <ENTER> "nop" <ESC>. Сохраняем все изменения в файле нажатием <F9> и выходим.

Запускаем WinRAR. Теперь NAG уже не выводится! Весь взлом не занял и десяти минут! Как вариант можно заменить mov eax, dword_004B43C8 (A1 C8 43 4B 00) на mov eax, 6 (B8 06 00 00 00), и тогда бедный архиватор будет всегда считать, что с момента регистрации прошло ровно шесть дней. Почему именно шесть? Ну, не шесть, так девять. Какая нам разница?! Главное - чтобы не больше 40! А еще можно заменить jg short loc_440F4F (7F 04) на jmp short loc_440F73 (EB 28), тогда безусловный переход будет перескакивать диалог независимо от текущего времени.

Наиболее красивым взломом считается тот, в результате которого в файл было введено минимум исправлений. Двумя байтами мы смогли отучить программу от надоедливого диалога, однако можно было управиться и с помощью одного. Найти его, скорее всего, не составит большого труда.

ПРИНУДИТЕЛЬНАЯ РЕГИСТРАЦИЯ

Несмотря на то, что раздражающий NAG успешно ликвидирован, программа остается незарегистрированной и честно пишет в заголовке окна: "evolution copy". А если нажать About, мы увидим "40 days trial copy". И хотя никаких ограничений в демонстрационной версии нет, чисто психологически работать с зарегистрированной копией намного приятнее.

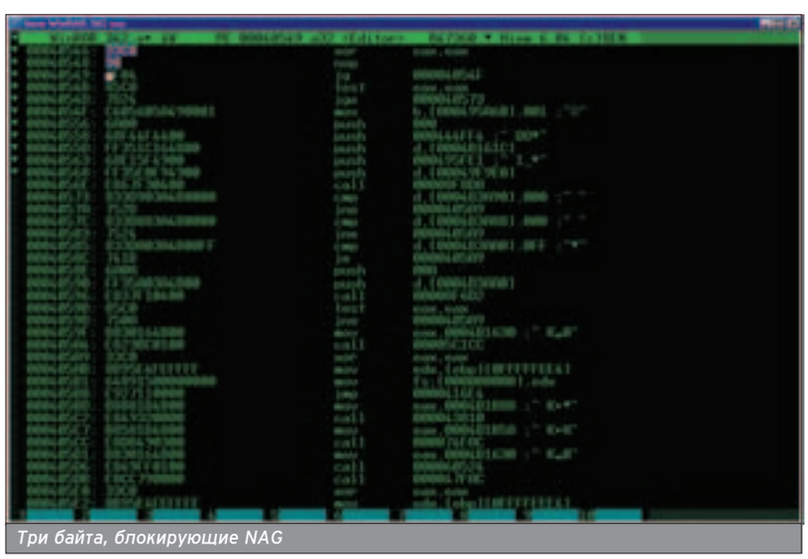
Известно, что регистрация осуществляется с помощью ключевого файла с электронной подписью, сгенерированной на криптографической основе с таким расчетом, чтобы подделка ключа была невозможной. Все это так, но нам же не нужен ключ! Мы хотим установить флаг регистрации! А как его найти? Вернемся к защитному механизму.

Выше уже известной нам инструкции "cmp eax, 28h" ополчилась целая серия условных переходов, при определенных обстоятельствах перепрыгивающих через этот противный диалог. Очевидно, один из них принадлежит флагу регистрации (у зарегистрированных пользователей NAG не выводится), но как определить, какой именно?

Разберемся со всеми по порядку. Назначение переменной byte_495A60 определяется сразу. При выводе диалога сюда записывается 1, то есть диалог уже выведен и выводить его повторно не нужно. С переменной dword_4B3A90 все гораздо сложнее. Чтобы узнать, кем она используется и для чего, необходимо просмотреть перекрестные ссылки. Подводим курсор к имени переменной, вызываем контекстное меню и выбираем пункт "jump to xref to operand" или просто нажимаем <X>. Появляется окошко с кучей информации.

Фу! Куча перекрестных ссылок по чтению (r) и записи (w), разбросанных по всему телу программы, среди которых доминируют dec и inc. На флаг регистрации это мало похоже. Скорее, это какой-то дикий семафор, используемый для организации взаимоблокировок. В общем, запчасть от интерфейса.

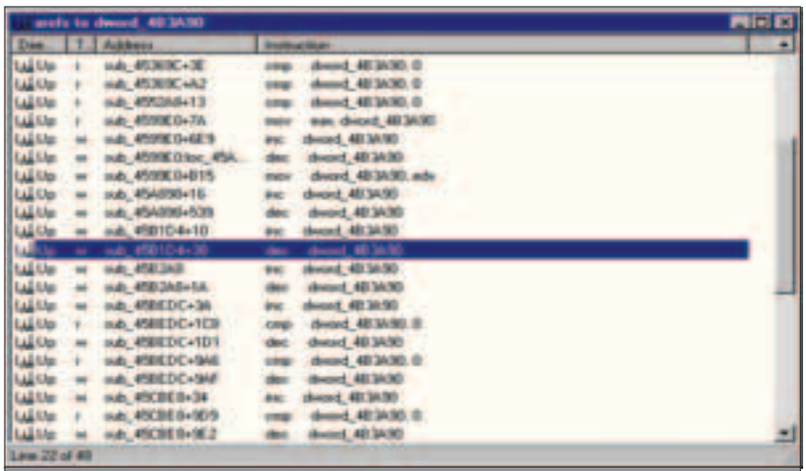
К переменной byte_4B7E00, ведут три ссылки, две из которых находятся в непосредственной близости от »



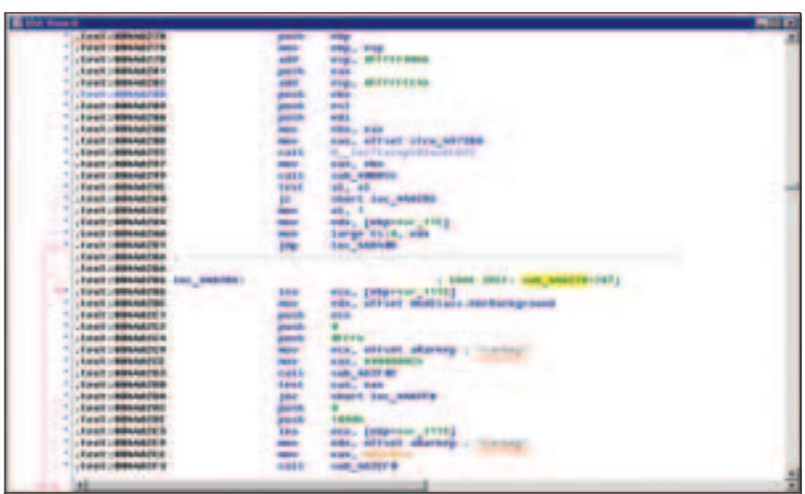
Три байта, блокирующие NAG

Подопытный WinRAR можно скачать по адресу www.rarsoft.com/rar/wrar342.exe.

На диске ты можешь найти как сам WinRAR, так и утилиты, необходимые для его взлома.



Исследование перекрестных ссылок



Дизассемблерный листинг функции DoRegister, выдающей себя текстовыми строками "rarkey"

функции DoDragDrop, так что их можно откинуть сразу.

А вот переменная byte_49F9BC - это настоящий клад. К ней ведет множество перекрестных ссылок на чтение и запись, но все записываемые значения возвращаются либо функцией sub_40DB5C, либо sub_44A278. При первом же взгляде на sub_44A278 бросаются в глаза текстовые строки "rarkey", заботливо оформленные дизассемблером как комментарии. Ага! Похоже, это и есть процедура, ответственная за регистрацию. Подводим курсор к ее началу, нажимаем <N> и переименовываем ее в "DoRegister".

С функцией sub_40DB5C разобраться тоже несложно. Достаточно проанализировать код, находящийся в самом начале DoRegister:

```
DoRegister    proc near
...
0044A299      call sub_40DB5C
0044A29E      test al, al
0044A2A0      jz short loc_44A2B6
              ; продолжение регистрации
0044A2A2      mov al, 1
0044A2A4      mov edx, [ebp+var_11C]
0044A2AA      mov large fs:0, edx
0044A2B1      jmp loc_44A40D
              ; на вход из функции
```

Если sub_40DB5C возвращает ноль, функция DoRegister продолжает регистрацию. Ненулевое значение приводит к немедленному выходу из функции. Логично предположить, что sub_40DB5C просто сообщает статус регистрации: ноль - не зарегистрирован, не ноль - зарегистрирован. Подведем курсор к началу sub_40DB5C и переименуем ее в "IsRegistered".

А давай заставим IsRegistered всегда возвращать ненулевое значение! Тогда программа будет признана зарегистрированной, несмотря на то, что ключевого файла, заверенного электронной подписью, у нас нет (да и откуда бы ему взяться)!

Запускаем HIEW, дважды нажимаем <ENTER> для перехода в дизассемблерный режим, давим <F5>, вводим ".40DB5C" (адрес функции IsRegistered), затем <F3> для перехода в режим редактирования и <ENTER> xor eax,eax <ENTER> inc eax <ENTER> retn <ESC> (обнулить регистр eax, тут же увеличить его на единицу и свалить из функции). Записываем изменения клавишей <F9> и выходим из редактора.

Надпись "evaluation copy" в заголовке окна послушно исчезает, а в окне About появляется строка "Registered to".

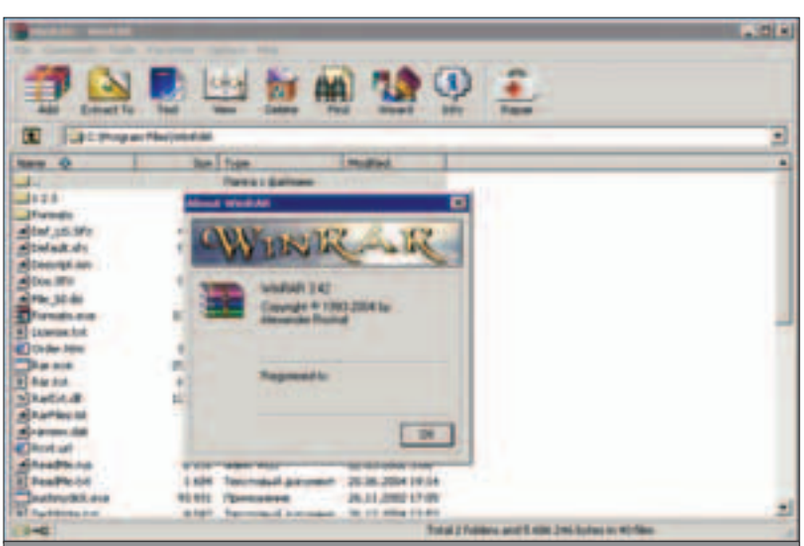
УКРОЩЕНИЕ ABOUT'A

Надпись "Registered to" - это, конечно, хорошо, только непонятно, на кого именно зарегистрирована программа. Первое, что приходит на ум - найти этот "Registered to" в программе (он там находится по смещению 50DBA4h) и заменить его на "hacked by KPNC", однако более длинный ник вместить уже не удастся, поскольку предельно допустимая длина строки жестко ограничена. Лучше найдем тот код, который выводит эту строку, и немного подкорректируем его!

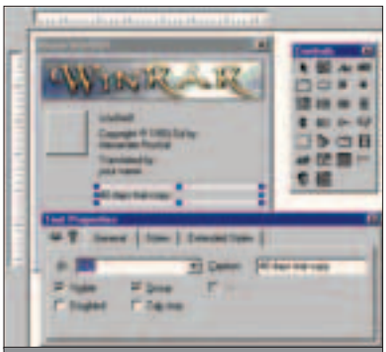
Запускаем Kerberos, загружаем winrar.exe, открываем "About", закрываем winrar.exe и лезем в протокол, в конце которого содержится строка DialogBoxParamA(400000, 496005: "ABOUTRARDLG", 001200AA, 00444618, 00000000), вызываемая по адресу 441D1Ch. Ага, это наш About Rar Dialog и есть! Возвращаемся в ИДУ и переходим по указанному адресу.

Код, создающий About-диалог:

```
00441D01    push offset sub_444618 ; lpDialogFunc
00441D06    push dword_4B161C
00441D0C    push offset aAboutrardlg
00441D11    push hLibModule
00441D17    call DialogBoxParamA
```



Как демонстрация версия стала зарегистрированной



Определение идентификатора поля вывода в Microsoft Visual Studio

Функция sub_444618, как и подсказывает IDA, представляет собой процедуру, ответственную за вывод диалога. Заглянем, что там? Ой-ой-ой, сколько всяких вызовов! Это же крышей поехать можно, пока разберешься, что к чему! Мы видим множество вызовов SetDlgItemTextA. Какой из них наш? Чтобы ответить на этот вопрос, требуется выяснить идентификатор соответствующего элемента управления.

Запускаем Microsoft Visual Studio (или любой другой редактор ресурсов), говорим "open file", в "Типе файлов" выбираем "Все файлы", а в Open as - Resources (если этого не сделать, файл будет открыт как двоичный, что совсем не входит в наши планы). В дереве ресурсов находим ветку "Dialogs", а в ней "ABOUTRARDLG". Дважды щелкаем по нему мышью или просто жмем <ENTER>. Запустится редактор ресурсов. Находим строку "40 days trial copy", на месте которой в зарегистрированной версии выводится "Registered to", и, вызвав контекстное меню, определяем ее ID. В нашем случае он равен 102 (или 66 в HEX-представлении).

Просматривая дизассемблерный листинг, ищем такую функцию SetDlgItemTextA, чьим аргументом будет идентификатор 66h. В конечном счете мы находим ее по адресу 4447ECh:

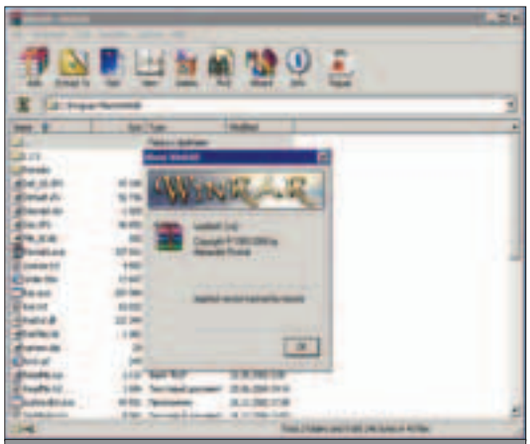
```
004447E6    call sub_4113DC
004447EB    push eax ; lpString
004447EC    push 66h ; nIDDlgItem
004447EE    push [ebp+hDlg]
004447F1    call SetDlgItemTextA
```

Функция sub_4113DC возвращает указатель на выводимую строку, которая тут же передается SetDlgItemTextA. Исследовать саму sub_4113DC мы не будем. Имя зарегистрированного пользователя берется из ключевого файла, над которым можно просидеть всю оставшуюся жизнь. Лучше внедрить свою строку в исполняемый файл и подменить указатель. Внедряться будем в секцию данных, в хвосте которой практически всегда имеется свободное место. Размещать выводимую строку в секции кода нельзя, поскольку RAR требует, чтобы она была доступна на запись.

Открываем HIEW, переходим в HEX-режим, давим <F8> для отображения заголовка файла и вызываем таблицу объектов (object table) клавишей <F6>. За секцией .data расположена секция .tls. Подгоняем сюда курсор и нажимаем на <ENTER>, а затем перемещаемся на несколько строк вверх, следя за тем, чтобы не залезть в значимые данные, которые начинаются там, где кончается цепочка нолей. В нашем случае это будет адрес 49D7B0h (хотя при желании также можно выбрать 49D7AEh, 49D7AFh и т.д.). Нажимаем <F3> для перехода в режим редактирования и записываем "registered version hacked by nezumi" (nezumi - это "мыщъх" по-японски).

Каталог секций

Создание подложной строки с именем зарегистрированного пользователя



Полностью хакнутая версия, зарегистрированная на мыщъх'а

версия	адрес IsRegistered	адрес lpString
3.0 stable (rus)	40BA4C	439740
3.42 stable (eng)	40DB5C	4,45E+09
3.50 beta 5 (eng)	40DE2C	4457B0
3.42 stable (rus)	40DB5C	4,45E+09

Адреса хакаемых байт в различных версиях WinRAR'а

Теперь переходим по адресу 4447E6h, возвращаясь к нашей диалоговой процедуре, и заменяем call sub_4113DC (E8 F1 CB FC FF) на mov eax, 49D7B0 (B8 B0 D7 49 00), где 49D7B0h - адрес хакнутой строки. Сохраняем изменения в файле, и... работает!

Теперь взломанная версия ничем не отличается от легально зарегистрированной! Разумеется, это еще не означает, что теперь RAR'ом можно пользоваться и ничего за это не платить (а законов никто не отменял), поэтому сразу же после экспериментов взломанный файл должен быть удален с жесткого диска.

ЗАКЛЮЧЕНИЕ

Многие разработчики используют электронные подписи и прочие криптографические механизмы, надеясь, что так уберегутся от зловредных хакеров. Как бы не так! Криптография - конечно, мощная штука, но к ней нужен свой подход. Если программа опирается на флаг регистрации (а так поступает большинство программ), она элементарно взламывается правкой нескольких байт, на поиск которых уходит совсем немного времени.

Зачем искать в Сети крэки, которые еще не факт что заработают. Гораздо интереснее и быстрее взламывать программы самостоятельно!

Ara (ara@clteam.net)

ПРИМЕР ВЗЛОМА: SOURCEFORMATX

ВЗЛОМ ПРОГРАММ С НЕВЕРОЯТНО ГАДКОЙ СИСТЕМОЙ ЗАЩИТЫ

Порой встречаются программы, которые как только заметят, что их ломают, начинают делать разные очень неприятные вещи. Ты только представь, каково будет удивление хакера, если при очередном взломе он вдруг лишится всех бесценных данных на своем жестком диске. Впрочем, вряд ли будет только удивление - тут и разрыдаться можно. Ниже будет рассказано, что я предпринял, когда наткнулся на подобную программу, как ломал ее, с какими трудностями столкнулся. Надеюсь, тебе будет полезно посмотреть, как другие крэкеры воюют с софтом. Enjoy!

Однажды на форуме cracklab.ru кто-то попросил меня помочь со взломом программы SourceFormatX версии 2.56, использующейся при форматировании исходных кодов. Она постоянно выводила NAG-окно с предложением зарегистрироваться, и все самые интересные функции форматирования исходников в ней были недоступны. В тот момент я был свободен и взялся посмотреть на защиту. После некоторых манипуляций, проведенных в отладчике, у меня вдруг стала самопроизвольно открываться папка "Мои документы". Уже открылось примерно 50 окон, прежде чем мне удалось убить процесс. И тут я с удивлением обнаружил, что не могу запустить ни одну программу на своей машине, а иконки на рабочем столе стали однообразно стандартными. Работали только программы, которые были запущены. Моя система умерла.

Тотчас на IRC-канале cracklab'a я попросил сообщить на форуме о таком поведении программы, чтобы другие были осторожны при ее взломе. И все равно некоторых постигла та же участь. Видимо, поэтому программа и осталась невзломанной - немногие захотели рисковать своей системой.

И вот через некоторое время, вооружившись необходимыми утилитами, я снова вернулся к этой злосчастной программе, чтобы, наконец, разобраться с ней и поделиться опытом с тобой. Несомненно, имея под рукой статьи подобного рода, тебе будет легче постигнуть трудную, но увлекательную науку взлома программ.

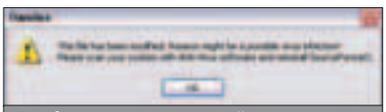
ИНСТРУМЕНТАРИЙ

Для работы нам будут необходимы следующие инструменты: отладчик OllyDbg, желательно последней версии с набором плагинов к нему (CommandLine или CommandBar, OllyDump), ImportREConstructor 1.6 Final и PEiD. Также рекомендую замечательную утилиту ShadowUser, найти ее не составит труда. Она отменяет ВСЕ изменения в системе, сделанные после перезагрузки. С ее помощью мы будем защищать системы от краха. В процессе взлома мне приходилось много раз видеть, как погибает моя ОС, но благодаря ShadowUser при перезапуске все возвращалось на свои места. Пользоваться ей очень легко: нужно кликнуть мышью в трее на ее значке и выбрать режим Enable. Программа попросит перезагрузку, на которую нужно согласиться, и после рестарта она загрузится уже активной. Теперь, чтобы мы ни делали, все изменения при следующем запуске системы будут возвращены на прежние места. Я проверял ее так: удалил несколько папок с диска C, прописал один файл нулями и очистил корзину. Потом деактивировал программу точно таким же образом (правым кликом) и убедился, что все стоит на своих местах. Итак, защита работает, активируем ее снова и приступаем к работе.

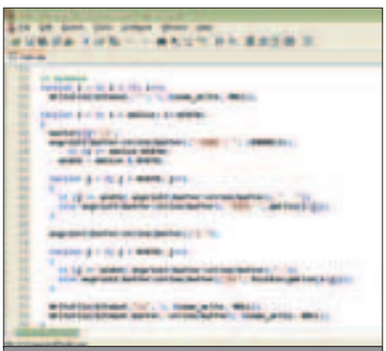
РАСПАКОВКА

При помощи PeiD определим, чем запакована программа. PEiD говорит, что это "PECompact 1.68 - 1.84->Jeremy Collake". Что ж, поверим. Загружаем программу в отладчик, вызвав плагин CommandLine клавишами <Alt>+<F1> или написав в командной строке команду hr esp-04 (установка hardware-прерывания), и пару раз нажимаем <F9> (Run). Остановимся прямиком на OEP (адрес - 00573258). Теперь можно снять дамп. Я воспользуюсь OllyDump, а ты - любым привычным тебе дампером - LordPE, PETools и т.п. Для восстановления таблицы импорта лучше всего подойдет ImpREC. Выбираем нашу программу в списке процессов, выставляем найденное OEP (573258-400000=173258), жмем GetImports. ImpREC определит все функции, поэтому кликаем Fix Dump и находим полученный ранее дамп. Все, программа распакована. Проверим правильность наших действий, запустив ее. Сразу видим сообщение, что программа заражена вирусом или модифицирована.

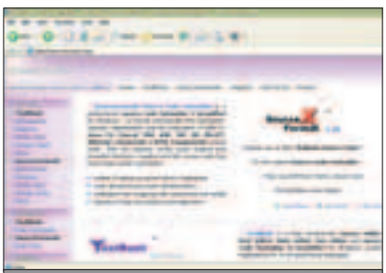
Однако несколько минут назад никакого вируса не было - наша система регулярно проверяется антивирусными средствами. Поэтому логично предположить, что таким образом программа проверяет себя на распакованность, тем самым сопротивляясь взлому. Что делать? Далее будем приводить программу в нормальный вид, устранив всячес-



Сообщение о зараженной программе

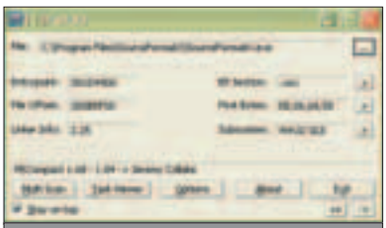


Злосчастная программа

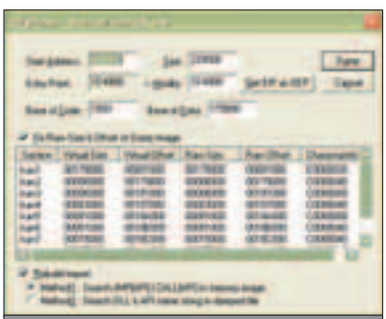


Сайт разработчиков SourceFormatX

Утилита ShadowUser



PeiD определил упаковщик как "PECompact 1.68 - 1.84->Jeremy Collake"



Плагин к OllyDbg, OllyDump

Восстановление импорта с помощью ImpREC

кие проверки и получив корректно работающий эк-земпляр.

УСТРАНЕНИЕ ЗАЩИТЫ ОТ ВЗЛОМА

Запускаем программу в отладчике (клавиша <F9>), и она благополучно закрывается. Причем вместе с отладчиком и безо всяких сообщений. Вывод тут один: имеет место проверка на наличие отладчика, что может осуществляться множеством разных способов. Начнем искать самые популярные: поиск окна по заданному классу с помощью FindWindows и получение хэндла процесса OpenProcess'ом. Ставим точки останова на эти функции:

```
bp FindWindow
bp OpenProcessA
```

Запускаем программулину, и нам попадется такой код:

```
kan1:00556144     push eax
kan1:00556145     call FindWindowA  ; поиск окна
kan1:0055614A     test eax, eax
kan1:0055614C     jz   short loc_556171
                ; переход, если не найдено
kan1:0055614E     lea  edx, [ebp+dwProcessId]
kan1:00556151     push edx
kan1:00556152     push eax ; hWnd
kan1:00556153     call GetWindowThreadProcessId
kan1:00556158     mov  eax, [ebp+dwProcessId]
kan1:0055615B     push eax   dwProcessId
kan1:0055615C     push 0     ; bInheritHandle
kan1:0055615E     push 1     ; dwDesiredAccess
kan1:00556160     call OpenProcess
kan1:00556165     test eax, eax
kan1:00556167     jz   short loc_556171
kan1:00556169     push 0     ; uExitCode
kan1:0055616B     push eax  ; hProcess
kan1:0055616C     call TerminateProcess
```

Вырезку кода я снабдил комментариями. Хотя и так понятно, что программа ищет окно с классом OLLYDBG и, если такое имеется, просто завершает процесс, то есть закрывает отладчик. Обойти это довольно просто: поменять условный переход по адресу 0055614C на безусловный (JMP), что можно сделать прямо в отладчике - двойной щелчок по нужной команде, правка команды и <ENTER>. Теперь можно сохранить сделанные изменения: выделив их, вызвать контекстное меню и выбрать пункт Copy to executable-Selection. Затем там же - Save to file.

Как вариант, можно пропатчить OllyDbg.exe. Просто изменить ему класс окна и заголовок.

Едем дальше. Теперь наш отладчик для исследуемой программы не виден - можно запустить ее и проверить. Увидим уже знакомое сообщение о вирусе и модификации. Сначала я просто попытался обойти вывод сообщения. Поставил bp MessageBoxA и посмотрел чуть выше.

```
kan1:005723A5     call sub_40929C
kan1:005723AA     test eax, eax
kan1:005723AC     jmp short loc_572403
                ; если 0, то переход
...
kan1:005723F7     push 0
kan1:005723F9     call MessageBoxA_0
```

Поменял переход и, сохранив изменения, запустил софтину. В итоге получил 50 открытых папок "Мои документы" и "убитую" ОС. По частой смене экрана я догадался, что происходит перезапуск explorer.exe (как-то я писал маленькую программку для удаления трояна со своего компьютера, там мне пришлось временно убивать explorer.exe - и было такое же мигание).

Раз уж происходит запуск, попробуем прерваться на выполнении API WinExec. Прописываем в командной строке bp WinExec и запускаем программу.

Мы остановимся в очень интересном месте:

```
kan1:00517A61     call SetFilePointer_0
kan1:00517A66     cmp  eax, 11F9DFh
kan1:00517A6B     jmp short loc_517A88
kan1:00517A6D     call sub_555B58
kan1:00517A72      mov  eax, [ebp-4]
kan1:00517A75     call sub_516710
kan1:00517A7A
kan1:00517A7A loc_517A7A:
                                ; CODE XREF: kan1:00517A86
kan1:00517A7A     push 3
kan1:00517A7C     push offset dword_5194F8
kan1:00517A81     call WinExec
kan1:00517A86     jmp  short loc_517A7A
kan1:00517A88 loc_517A88:
                                ; CODE XREF: kan1:00517A6B
kan1:00517A88     xor eax, eax
```

»

> Стоит запустить программу в отладчике, как она благополучно закрывается. Причем вместе сотладчиком.

Если вдруг в этой статье тебе что-то непонятно, а листать журнал лень, лезь на сайт cracklab.ru. Там ты сможешь найти множество статей по этой и по другим интересующим тебя темам, взять инструменты, а также проконсультироваться со специалистами на форуме.

Это же бесконечный цикл! Нужно изменить условный переход по адресу 00517A6B на безусловный, чтобы никогда больше не попадать в это кошмарное место. Внимание! Перед циклом вызывается функция SetFilePointer, и результат ее работы сравнивается с неким числом. Даже без отладчика можно понять, что это не что иное, как проверка размера файла, то есть проверка на распакованность, которая делается следующим образом: вызывается функция SetFilePointer, устанавливающая указатель на конец файла. Возвращенный результат (текущая позиция указателя) как раз и будет размером файла, так как указатель находится в самом его конце. Полученное число сверяется с зашитым в программе. И если первое больше второго, то вызывается цикл. Будем исправлять такое безобразие. Сначала я вызвал из контекстного меню отладчика функцию Seargh for-All intermodular calls и поставил прерывания на все API WinExec (всего 27 вызовов). Это было сделано для удобства их поиска в коде, прерываться на них мы больше не будем. Теперь, тыкая в букву "B" на панели инструментов OllyDbg, можно посмотреть все установленные бряки. Кликая на каждый из них по очереди, поменяем все находящиеся выше условные переходы на безусловные. Кроме первого (там вызывается блокнот, возможно, это нужный вызов).

Попадаются интересные места вроде:

```
0051B526    PUSH 1_0051BD04
                        FileName = "\\.\NTICE"
0051B52B    CALL <JMP.&kernel32.CreateFileA>
                        CreateFileA
```

Оказывается, программа обнаруживает не только OllyDbg, но и SoftIce. Посмотришь сам: встроено и обнаружение некоторых инструментов взломщика - DeDe, RegMonitor и т.п.

И вот, кстати, занимательное место перед бесконечным циклом:

```
0055F10B    call <JMP.&kernel32.GetFileSize>
                        ; GetFileSize
0055F110    cmp eax,1242A8
0055F115    jle short 1_0055F136
```

Еще одна проверка на распаковку. Запомним это.

Исправив все переходы, нужно сохранить результат исправлений. Почему-то у меня некорректно работает функция сохранения всех сделанных изменений в OllyDbg, поэтому я делаю так: перемещаю указатель на начало секции кода (обычно это адрес 401000), затем в конец секции и, удерживая клавишу <Shift>, выделяю любую строку. Выделяется весь код, и его можно сохранять так же, как мы делали ранее при одном изменении.

Однако мы еще не до конца убрали проверку на распакованность, и радоваться нам рано. При обходе бесконечного цикла, который мы проделали ранее, было замечено, что перед циклом сначала вызывались функции GetFileSize и SetFilePointer. Некоторые из них мы уже обошли. Но у нас нет возможности проанализировать код: все строки в файле зашифрованы и расшифровываются по мере необходимости. Используя дизассемблер, мы не можем точно сказать, для каких именно файлов вызываются эти функции: может вычисляться, к примеру, размер форматируемого исходника, тогда этот участок нам обходить нельзя. Чтобы не попортить нужные участки кода, будем проводить анализ динамически, то есть с помощью отладчика. Конечно, тут возможны ошибки, какие-то места мы можем и пропустить, но делать нечего. В командной строке ставим точки останова:

```
bp GetFileSize
bp SetFilePointer
```

Жмем <F9> и смотрим. В тех местах, где полученный размер будет сравниваться с константой, мы будем менять условные переходы на безусловные. Подправив все необходимые места и сохранив изменения, мы можем нормально запустить свою программу. Теперь можно заняться собственно процессом взлома - радуемся.

УСТРАНЕНИЕ ОГРАНИЧЕНИЙ

Сначала проверим, как работает наша модифицированная программа. Откроем какой-нибудь исходник и попробуем отформатировать его кнопкой Format. Получаем сообщение с предложением о регистрации. Это не для нас - отказываемся и видим свой отформатированный исходник. Только форматирование у него какое-то странное: вместо красивых рядов кода лишь одна строчка с непонятными значками, крякозябрами. Откроем еще один исходник и выберем в меню File пункт Obfuskate. В итоге та же строка с крякозябрами. Теперь попробуем отформатировать сразу два открытых исходника, выбрав пункт Format All. Появляется сообщение о недоступности данной функции в незарегистрированной версии, то есть форматировать исходники совсем отказываются.

Начнем исследование с нахождения процедуры обработки нажатия кнопки Format. Ты, наверно, уже заметил, что программа написана на Delphi, а если нет, то убедись в этом с помощью PEiD. При анализе программ, написанных на Delphi, может помочь декомпилятор DeDe (полную версию можно скачать из раздела "Инструменты" сайта cracklab.ru или взять с диска). Запускаем исследуемую программу, затем сам DeDe, выбираем в нем пункт "Декомпиль активный процесс", указываем нашу программу в списке процессов и жмем "Дамп". Подождем, пока декомпилятор закончит работу, и начинаем искать нужный код. Сначала посмотрим процедуры, которые находятся в TmainForm. Их там оказывается довольно много, однако среди них легко выделить одну важную - FormatBtnClick, ее название говорит само за себя. Адрес начала процедуры - 0056C15C. Теперь можно глянуть программу в отладчике: загружаем ее в OllyDbg, переходим на адрес 0056C15C, ставим прерывание (<F2>) и запускаем программу. Теперь, если мы откроем исходник и нажмем Format, то остановимся как раз на начале процедуры форматирования. Здесь видим только одну инструкцию CALL, а за ней сразу RET. Придется немного потрейсить программу.

Это производится с заходом в процедуры (клавиша <F7>) или без (<F8>). Другими словами, если мы будем трейсить по <F8>, то выполнится вся процедура, в том числе вложенные, что не позволит посмотреть выполнение кода. Поэтому заходим в процедуру по <F7>, а дальше уже обходим все процедуры по <F8>. Чтобы не терять времени, я сначала прогнал все процедуры без захода в них и нашел ту, которая вызывает сообщение с предложением зарегистрироваться. Чуть выше по коду стоит условный переход, который обходит эту процедуру в зарегистрированной версии. Меняем его на безусловный.

По ходу можно отметить, что программа ищет занятный файл License.dat. Думаю, не надо объяснять, зачем он нужен. Для эксперимента я создал пустой файл с таким именем в каталоге программы, в итоге надпись "Unregistered user" в About пропала. Больше ничего интересного при первом просмотре не попалось.

Ну что же? Если мы точно уверены, что все дело в том, что программа распакована, что именно из-за этого

> Чтобы не попортить нужные участки кода, будем проводить анализ динамически, то есть с помощью отладчика.

Стек в OllyDbg при остановке на вызове ReadFile

исходник не хочет нормально форматироваться, поищем по-другому. Все проверки типа SetFilePointer и GetFileSize мы уже устранили ранее. Теперь я решил посмотреть вызовы функции ReadFile (чтение из файла). Перезапускаем программу в отладчике, прописываем в командной строке bp ReadFile и снова пытаемся отформатировать злосчастный исходник. Всякий раз брякаясь, будем смотреть стек (правая нижняя секция отладчика). Как известно, параметры при вызове процедур обычно передаются именно через стек. Нас интересует хэндл открытого файла и буфер, куда будут занесены считанные байты.

Нажимаем кнопку "H" на панели инструментов отладчика и смотрим, какой хэндл соответствует открытому файлу нашей программы. Далее ждем момента, когда программа брякнется и в стеке будет лежать именно это подсмотренное значение. Другие файлы нас не интересуют, поэтому на остальных всплытиях отладчика жмем <F9>.

Наконец, программа останавливается и в стеке лежит нужное значение. В коде рядом при этом читается 360 байт из нашего файла. Находим параметр Buffer, кликаем по нему правой кнопкой мыши и выбираем из появившегося контекстного меню пункт Follow in Dump. В окне, где отображается дамп памяти (правое нижнее) видим, что находится в данный момент в памяти, начиная с адреса параметра. Немного прогнав код, проследим, как заполнятся ячейки считанными из файла байтами. Отметим, что считался заголовок нашего файла - в памяти ясно видны имена секций. Похоже, это очередная уловка разработчиков - подсчитать контрольную сумму заголовка, так как при распаковке изменения в нем неизбежны. Чтобы отловить момент вычисления этой контрольной суммы, поставим точку останова на обращение к памяти, для чего выделим несколько первых байт памяти, вызовем контекстное меню и выберем пункт Breakpoint. Теперь можно продолжить выполнение программы (<F9>). Оно остановится как раз на месте, где и происходит искомая нами калькуляция.

```
0045A00D    movzx esi, byte ptr ds:[ebx]
0045A010    movzx ecx, al
0045A013    xor ecx, esi
0045A015    shr eax,8
0045A018    xor eax, dword ptr ds:[ecx*4+579BC0]
0045A01F    inc ebx
0045A020    dec edx
0045A021    jnz short 1003.0045A00D
0045A023    pop esi
```

Разбираться в алгоритме мы не будем, отметим только, какое значение получается в итоге - оно будет в регистре EAX. У меня на выходе из процедуры EAX = 69E70672. Естественно, эта контрольная сумма будет неверной. Чтобы узнать правильную, будем смотреть, что должно быть в оригинальной программе. Потом попробуем подставить нужное значение в распакованную.

Не попадаясь на антиотладочные приемы, запускаем программу и аттачимся к своему процессу. Для этого запускаем оригинальную версию программы, открываем в отладчике меню File->Attach и выбираем наш процесс. Нажимаем <F9>, ставим бряк на 0045A026 (остановившись на этом адресе, можно будет увидеть верную контрольную сумму), открываем любой пример, жмем Format и смотрим правильную контрольную сумму. Должно быть 1DFF122A.

Поменяем немного код распакованной жертвы. Тут можно делать все, на что хватит фантазии. Я сделал таким образом:

```
0045A009 mov eax, 1DFF122A
0045A00E jmp short 1003.0045A023
...
0045A024 nop
0045A025 nop
```

Сохраняем изменения, перезапускаем программу и проверяем работу - теперь все форматируется как положено. К тому же работает кнопка Obfuscate. Очень хорошо, можно заняться функцией Format All, которая вообще не хочет работать в незарегистрированной версии. Опять обратимся к помощи DeDe. Теперь наша процедура будет называться FormatAllBtnClick. Таким же способом, как и раньше, ставим бряк на адрес начала процедуры и, немного потрейсив, находим адрес вызова сообщения. Снова чуть выше вызова поменяем условный переход на безусловный.

```
0056468B    cmp byte ptr ss:[ebp-35], 0
0056468F    jnz short 1003.005646B2
00564691    mov eax, 0D9
00564696    call 1003.005589B0
0056469B    mov eax, 1003.00565680
005646A0    call 1003.005570D0
```

Сохраняем изменения и любуемся работой взломанной программы. Нет никаких нудных сообщений, NAG-окон, неработающих функций и т.п. Дело сделано, можно радоваться.

ВМЕСТО ЗАКЛЮЧЕНИЯ

Стремление разработчиков причинить взломщику максимальное беспокойство чаще всего бесплодно. Наоборот, это только прибавит упорства второй стороне. При взломе этой программы мы рассмотрели лишь небольшую часть приемов, которыми разработчики стремятся всячески испортить нам жизнь. К тому же ничего оригинального они изобрести не смогли, поэтому нет им оправдания за загубленные системы и потерянные файлы. Что ж, надеюсь, моя статья хоть как-то поможет тебе в нашем нелегком деле и ты не наступишь на те же грабли, что и я.

(ANTI)CRACKING

Ms-Rem (Ms-Rem@yandex.ru)

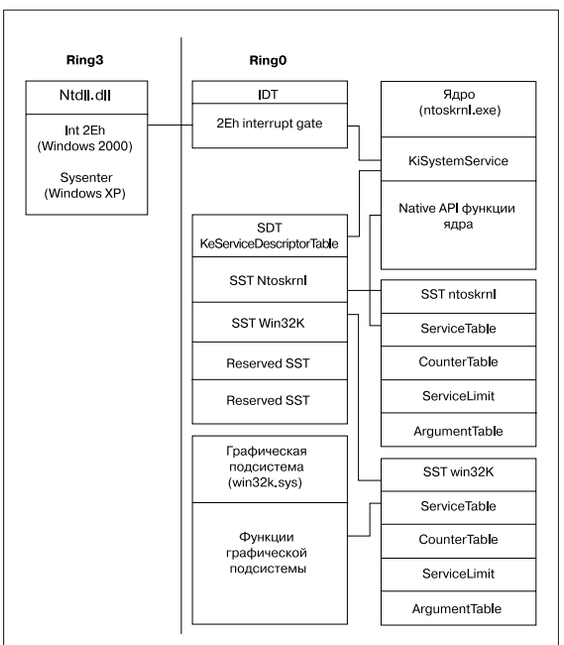
КРУТОЙ ПРОТЕКТОР – НЕ БЕДА

ТЕХНОЛОГИИ ВЗЛОМА СЛОЖНЫХ ПРОГРАММНЫХ ЗАЩИТ

Как ломаются протекторы? Находим OEP, снимаем дамп, восстанавливаем импорт – все. И это почти стандарт. Конечно, ты уже хорошо освоил это и умеешь распаковывать всякие аспры с закрытыми глазами. Инструменты, которыми ты пользуешься - это SoftIce, IceExt, IDA, OllyDbg, PeTools, LordPE, ImpRec, PEiD. Для снятия большинства протекторов их вполне достаточно. Но каждый крэкер когда-нибудь сталкивается с такой защитой, перед которой все инструменты пасуют. В этой статье я постараюсь рассказать, почему происходит так и что делать в таком случае. Опишу трюки, которые выделывают в самых сложных защитах и, главное, объясню, как обходить их.

Действительно крутых защит в мире немного. Это не ASProtect, не Armadillo, не SVKP, не EXE Stealth, не MoleBox и уж точно не какой-нибудь упаковщик или UPX Scrambler. Справляться со всем этим ты уже, несомненно, научился. Я говорю о защитах уровня hi-end, против которых бессильны стандартные методы и подходы. То есть о StarForce, XtremeProtector (Themida), ExeCryptor и о тех, что привязывают софт к аппаратным ключам (Hasp Envelope, Guardant и т.п.). Об этих защитах и об используемых ими приемах и поговорим.

Все протекторы можно разделить на два класса: это Ring3-протекторы, код которых исполняется только в третьем кольце защиты процессора, и Ring0-протекторы, которые имеют драйверы режима ядра, позволяющие влиять на работу ОС и многократно расширяющие список применяемых защитных приемов. Из hi-end протекторов единственным работающим в Ring3 остается ExeCryptor. Это делает его в некотором роде уникальным, но принципы его работы имеют массу неприятных недостатков. Из новых приемов защиты в hi-end протекторах можно выделить: влияние на работу операционной системы (на API-функции, в частности), применение псевдокода, исполняемого на виртуальных машинах, самомодифицирующийся и перекрывающийся код, недокументированные инструкции процессора (дизассемблирующиеся неправильно), масса размытого и мусорного кода, блокировка отладочных возможностей процессора. И всем этим арсенал hi-end протекторов, естественно, не исчерпывается. Теперь подробнее.



Интерфейс системных вызовов Windows NT

RING0-ПРОТЕКТОРЫ

К этому классу относится большинство hi-end протекторов. Обычно ни отладчики, ни различные дамперы ничего не могут сделать с ними – в этом главная неприятность этих протекторов. Попробуем, например, сдампить с помощью PeTools программу, защищенную xprot'ом. Очень странно, но защищенный процесс отображается в PeTools как [System Idle Process], поэтому заглянуть внутрь и сдампить его у нас не получается. В OllyDbg при попытке приаттачиться к процессу мы вообще не обнаруживаем его в списке (OllyDbg, как и любые другие Ring3-отладчики, непригоден для анализа Ring0-протекторов, так что в дальнейшем мы будем использовать только SoftIce). Большинство начинающих крэкеров, видя такую ситуацию, сразу же бросают ломать программу, считая, что она безнадежна. Но мы не маленькие - нас такими трюками не испугаешь. Если немного подумать головой, то можно прийти к выводу, что, скорее всего, перехвачена одна из функций, имеющих дело с памятью процессов. Для дамперов очень важна работоспособность Native API функций ZwOpenProcess, ZwReadVirtualMemory и ZwWriteWirtualMemory, так как на них построено большинство высокоуровневых функций работы с процессами, в число которых входят OpenProcess, ReadProcessMemory, WriteProcessMemory и EnumProcessModules. Возможен, конечно, перехват этих функций в третьем кольце, но это просто глупо (так как его легко обойти), и я такого еще ни в одной защите не встречал, поэтому лезем сразу в ядро системы. Только для начала следует уяснить, как работает интерфейс системных вызовов Windows NT.

Из схемы, которая расположилась где-то рядом, следует, что вызов функции ядра до его передачи соответствующей Native API проходит довольно сложную обработку. Сначала, в третьем кольце, вызывается соответствующая функция библиотеки ntdll, где в регистр EAX помещается номер вызываемого системного сервиса, а в регистр EDX - указатель на передаваемые параметры. Затем вызывается прерывание 2Eh (в Windows XP - команда sysenter), и происходит переход процесса в нулевое кольцо, где управление передается согласно записанному в IDT шлюзу прерывания. В этом месте происходит переключение окружения третьего кольца на нулевое. Выполняется смена пользовательского стека на стек ядра. Также осуществляется перезагрузка сегментного регистра FS, который в нулевом кольце указывает на совершенно иные структуры, чем в третьем кольце. Затем управление передается обработчику прерывания 2Eh - функции ядра KiSystemService. Она копирует в стек ядра передаваемые системному сервису параметры и производит вызов Native API функции ядра, согласно содержимому ServiceDescriptorTable (SDT). Эта таблица находится в памяти ядра и представляет собой структуру, содержащую четыре таблицы системных сервисов (SST). Первая из этих таблиц описывает сервисы, экспортируемые ядром (ntoskrnl.exe), вторая - графической подсистемой (win32k.sys), а остальные две зарезервированы на будущее и сейчас не используются.

Самый простой способ перехвата Native API в ядре - замена адреса обработчика нужной функции в SDT на "свой". При вызове перехваченной функции происходит проверка параметров, и если производится (например как в протекторе) попытка чтения памяти защищенного процесса, то возвращается отказ в доступе, иначе просто вызывается оригинальный обработчик функции. Просмотреть содержимое SDT мы можем командой SoftIce'a NTCALL, при этом отладчик выводит номер функции, ее имя (если установлены отладочные символы), адрес и имя функции обработчика. Если вместо Ntoskrnl!NtOpenProcess мы видим что-то типа Xprot!.text + 1234h, то функция перехвачена защитой.

Однако патчинг SDT - не единственный способ перехвата Native API. Также может быть перехвачено прерывание int 2Eh в win2k (о чем можно узнать командой SoftIce'a IDT) либо изменен обработчик системного вызова через sysenter в winxp и выше. Установка\снятие обработчика прерывания int 2Eh будет выглядеть так:

```
void Set2kSyscallHook()
{
    TIdt Idt;
    _asm
    {
        cli
        sidt [Idt]
        mov esi, NewSyscall
        mov ebx, Idt.Base
        xchg [ebx + 0x170], si
        rol esi, 0x10
        xchg [ebx + 0x176], si
        ror esi, 0x10
        mov OldSyscall, esi
        sti
    }
}
```

Для многопроцессорных систем, в том числе для процессоров Hyper Threading, этот код существенно усложняется, но рассматривать его сейчас не будем. Если интересно, залезь в исходники IceExt.

Адрес обработчика sysenter вызова определяется содержимым 32-битного MSR-регистра с номером 176h. Читать или записывать в такие регистры мы можем с помощью команд RDMSR/WRMSR, предварительно поместив в ECX номер регистра, а EAX используя как источник или приемник нового значения. Код, перехватывающий обработчик sysenter, будет выглядеть примерно так:

```
void SetXpSyscallHook()
{
    _asm
    {
        mov ecx, 0x176
        rdmsr
        mov OldSyscall, eax
        mov eax, NewSyscall
        wrmsr
    }
}
```

Перехват также может быть осуществлен методом сплайсинга (замена участка кода перехватываемой функции), что легко обнаружить при трассировке перехваченного участка.

Что ж, с методами перехвата разобрались. Осталось научиться бороться с ними. Тут все очень просто. Нужно определить адреса оригинальных обработчиков и самостоятельно пропатчить SDT, убрав перехват, либо заменить участок кода, измененный при сплайсинге, аналогичным участком из оригинального файла ядра системы (ntoskrnl.exe). При этом ты можешь столкнуться с проверкой наличия перехвата протектором, и если ты не знаешь, как бороться с такими вещами, значит, тебе рано браться за ring0-проекторы - потренируйся лучше на аспаке.

Второе, с чем ты обязательно столкнешься в ring0-протекторах - это перехват отладочных прерываний (int 1 и int 3). Этот прием не дает нам трассировать код в отладчике и ставить бряки. Реакция защит на срабатывание этих прерываний банальна - синий экран! Нужно что-то делать с этим, если ты хочешь пользоваться отладчиком.

Если отладочные прерывания просто заблокированы, но не несут при этом никакой смысловой нагрузки, то можно восстановить их оригинальные векторы, отключить проверки целостности перехватов (если они будут) и радоваться жизни. Но, к сожалению, в современных протекторах отладочные прерывания не просто вырублены, а используются для какой-либо работы, без которой защищенная программа функционировать не будет. Здесь есть три варианта: 1) перевести протектор на другие прерывания; 2) перевести отладчик на другие прерывания; 3) заставить и протектор, и отладчик работать на общих прерываниях.

В первом случае нужно разобраться в том, как протектор использует отладочные прерывания, и попробовать заменить их другими, не конфликтующими с отладчиком. Например, если в коде протектора встречается int 1, то заменим его на int 20 и модифицируем вектор 20 прерывания так, чтобы он указывал на обработчик протектора, после этого int 1 можно будет использовать для отладчика.

Второй подход несколько сложнее, так как аппаратная трассировка и точки останова работают только на стандартных отладочных прерываниях. Поэтому нам придется писать плагин к SoftIce'y, который будет вешать всплытие отладчика на свободное прерывание (например 20), а при установке бряка будет пихать в код не CC, а CD20.

Третий подход предполагает написание драйвера, который будет перехватывать отладочные прерывания сам и передавать их отладчику, после чего ты решишь, передавать прерывание защите или нет. Естественно, это тоже потребует написания плагина к SoftIce'y. Я обычно придерживаюсь этого подхода, он особенно удобен при взломе Star Force, который использует int 3 для вызова своей виртуальной машины в ring0. Этот вызов легко отделяется от срабатывания точки останова, что позволяет легко распределять возникающие прерывания между защитой и отладчиком.

А вообще это еще цветочки. В ring0-протекторах можно встретить кучу других более опасных приемов. Например, попробуй трассировать ring0-код при ESP = 0 - получишь синий экран. Это связано с тем, что при возникновении отладочного прерывания адрес возврата заносится в ring0-стек потока, а если по этому адресу не оказывается памяти, то возникает исключение.

Или, например, можно встретить отключение аппаратных прерываний с помощью перепрограммирования чипсетного контроллера прерываний. Если отладчик всплывает при вырубленных прерываниях, то остается только нажать reset, так как компьютер зависнет намертво. Бороться с этим можно опять же написанием драйвера, который включает прерывания перед вызовом отладчика.

К сожалению, такими "сюрпризами" богаты все современные ring0-протекторы, поэтому нужно уметь вовремя распознать и обойти их. Арсенал антиотладочных приемов в ring0 постоянно расширяется, и от них спасет »

RDMSR—Read from Model Specific Register

Opcode	Instruction	Description
0F 32	RDMSR	Load MSR specified by ECX into EDX:EAX

Description

Loads the contents of a 64-bit model specific register (MSR) specified in the ECX register into registers EDX:EAX. The input value loaded into the ECX register is the address of the MSR to be read. The EDX register is loaded with the high-order 32 bits of the MSR and the EAX register is loaded with the low-order 32 bits. If less than 64 bits are implemented in the MSR being read, the values returned to EDX:EAX in unimplemented bit locations are undefined.

This instruction must be executed at privilege level 0 or in real-address mode; otherwise, a general protection exception #GP(0) will be generated. Specifying a reserved or unimplemented MSR address in ECX will also cause a general protection exception.

IA-32 Intel Architecture Software Developer's Manual - очень полезная вещь для начинающего программиста

только хорошее знание защищенного режима работы процессора и внимательный анализ кода протектора.

Если захочешь потренироваться во взломе ring0-протекторов, я написал небольшой crackme, который использует все описанные здесь приемы. Распаковывается он просто, но код, проверяющий серийник, спрятан с помощью ring0-штучек. Закейгенить этот пример пока еще никто не смог, так что у тебя есть шанс стать первым. Crackme найдешь на диске с журналом.

МЕТАМОРФ И ПОЛИМОРФ

Вот пример легко понятного кода:

```
push eax
push 0
push 0
push ebx
call MessageBoxA
```

А если вместо этой гениальной простоты ты встречаешь путаницу вроде

```
xchg [edi], dl
add al, 30h
xlat
call 1234h
call 3456h
or al, 4a
```

знай, что ты встретился с метаморфом или полиморфом в защите. Полиморф обычно просто добавляет в код мусорные инструкции, чтобы затруднить дизассемблирование и анализ кода, а метаморф старается целиком изменить вид кода, сохраняя при этом оригинальный алгоритм его работы, для чего он заменяет инструкции их синонимами, состоящими в свою очередь из одной или нескольких других инструкций. Большая часть нового кода, производимого метаморфом, обычно нужна для работы программы, доля мусорного кода у него весьма мала. У полиморфа все наоборот. Естественно, декодировать метаморф значительно сложнее. Прежде чем делать это, нужно разобраться, как он работает.

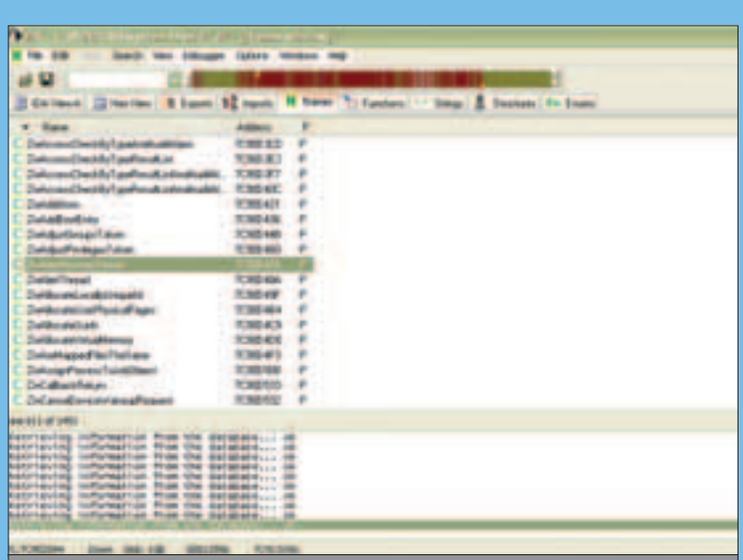
Сердцем любого метаморфа обязательно является дизассемблер. С его помощью происходит разделение защищаемого кода, после чего, как уже говорилось выше, производится замена всех инструкций на синонимы или небольшие куски кода, несущие тот же смысл. Причем замена может делаться неоднократно. Число проходов (циклов замены) морфера называется глубиной морфинга. Чем она больше, тем более запутанным будет выходной код. После морфинга инструкции компилируются обратно в машинный код. Авторы метаморфов считают, что большая глубина морфинга осложнит анализ кода, но мой опыт подсказывает мне обратное. Сложность декодирования метаморфа целиком зависит от первичного алгоритма морфера и от того, сколько комбинаций инструкций он способен выдать на одну оригинальную инструкцию. Увеличение же числа этих комбинаций повышением глубины морфинга ничего хорошего не дает: все равно после написания анализатора можно будет снять метаморф в несколько проходов так же, как он и накладывался.

Для начала рассмотрим простой полиморф и подумаем, как декодировать его. В качестве подопытной программы я взял написанный на ассемблере Hello World и защитил его с помощью DotFix FakeSigner 2.8. В IDA получившийся код выглядит так:

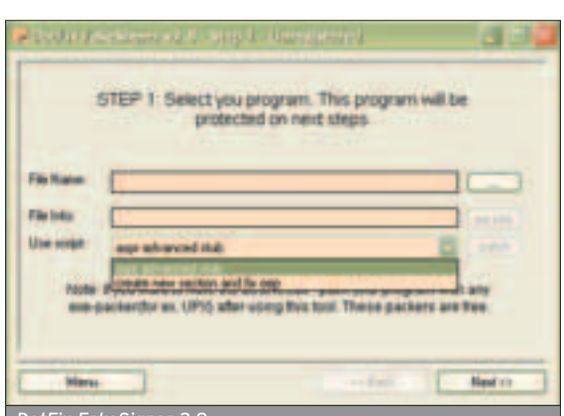
```
jmp short loc_902003
db 0D8h
jmp short loc_902006
db 0BFh
finit
fprem
inc eax
dec eax
inc eax
dec eax
lea ebx, [ebx+0]
jmp short loc_902015
db 0F6h
xor edx, edx
jmp short loc_902026
```

NATIVE API

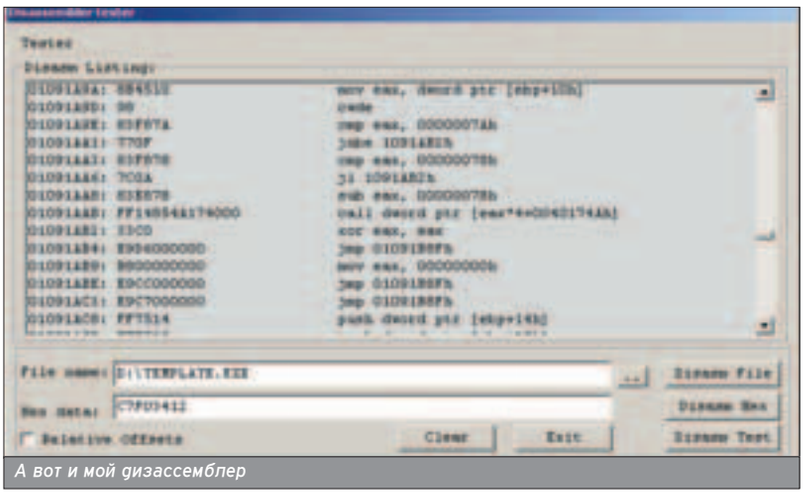
Функции Native API являются базовыми для системы, на них построена работа более высокоуровневого слоя функций kernel32. Они доступны из пользовательского режима через ntdll.dll, но на самом деле функции ntdll являются только переходниками, которые через интерфейс системных вызовов обращаются к соответствующим функциям ядра. При программировании драйверов мы можем использовать те же Native API функции, что и в приложениях третьего кольца, но API более высокого уровня тут недоступен. Также на этом уровне в наши руки попадают многие функции, экспортируемые ядром и предназначенные для использования только в драйверах (прослойка Kernel API). В Native API пользовательского уровня есть пары аналогичных функций, отличающихся только префиксами Zw и Nt. Там они имеют разные названия, но одну и ту же точку входа. На уровне ядра также существуют аналогичные пары функций, но между ними имеется одно различие: функции с префиксом Zw производят перед выполнением действия проверки системы безопасности (прав пользователя), а функции с префиксом Nt - нет.



Native API функции в ntdll.dll



DotFix FakeSigner 2.8

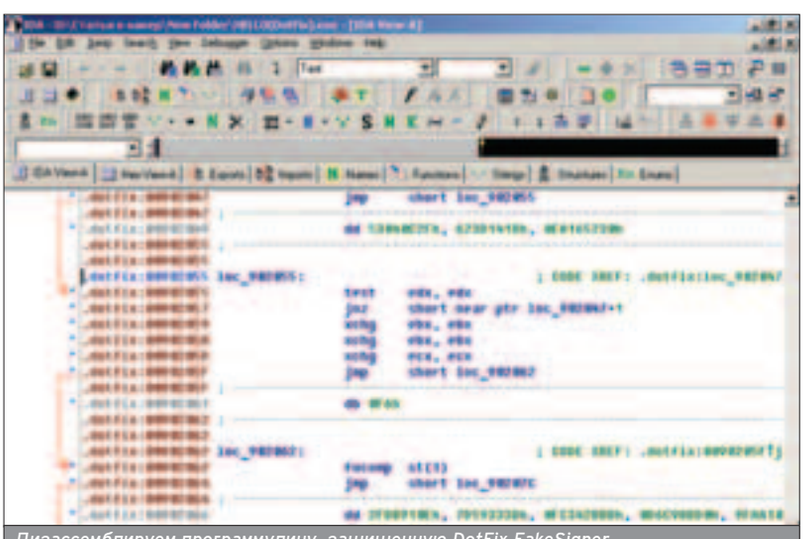


А вот и мой дизассемблер

Здесь мы видим пример использования простейшего полиморфа. Инструкции программы разносятся на разное расстояние, соединяются с помощью jmp, а промежутки заполняются мусорными инструкциями. В данном случае весь приведенный код представляет собой мусор, о чем нетрудно догадаться, однако начинающих крэкеров это может сбить с толку. Еще в этом полиморфе можно встретить редкие SEH, но автор даже не пытался их замаскировать - их не увидит только слепой. Для декодирования этого полиморфа необязательно использовать полноценный дизассемблер - достаточно дизассемблера длин и таблицы мусорных шаблонов. Алгоритм такой: проходим дизассемблером длин по инструкциям программы, переходим по jmp и собираем весь код (кроме jmp) где-нибудь в памяти. После этого по шаблонам удаляем мусорные инструкции. Вот и все - полиморф декодирован. FakeSigner, кстати, предназначен для затруднения определения упаковщика, которым сжат исполняемый файл, и, по идее, нельзя определить наличие самого дотфикса. К сожалению, код этого полиморфа настолько специфичен, что написать программу, определяющую его, будет проще простого. Если хочешь потренироваться в разборе полиморфов, рекомендую начать именно с этого.

Несомненно, при взломе сложных защит тебе придется столкнуться с метаморфом, а это штука даже неприятнее, чем простой полиморф. Для декодирования метаморфа нужно написать сложный анализатор, который содержит в себе дизассемблер, разбирающий инструкции, и деморфер, реализующий алгоритм, обратный морферу. Простейший алгоритм разбора метаморфа состоит в анализе близкорасположенных инструкций и определении того, как они в целом повлияют на регистры и память. После этого преобразование повторяется до получения кода наименьшего размера. К примеру, группу add eax, 5/sub eax, 4 можно превратить в inc eax. Этот метод называется многопроходной оптимизацией и применяется для реверсинга простых метаморфов. В более сложных случаях зависимые команды могут быть отделены кучей мусора или перемешаны с кодом протектора. И тут на помощь придет только частичная эмуляция участков кода на виртуальной машине и формирование кода на основе изменений состояния ВМ на определенных отрезках кода.

Формируется таблица измененных состояний и зависимостей между ними, после чего производится перегруппировка элементов таблицы таким образом, чтобы независимые элементы находились в одном месте. Далее к таблице применяется метод многопроходной оптимизации, который значительно сокращает число ее элементов. По сокращенной таблице строится цепочка инструкций, соответствующая изменениям, записанным в ней, и эти инструкции компилируются в машинный код. Это один из алгоритмов, которые можно использовать при анализе действительно сложного метаморфа. Можно поступить и по-другому: писать не эмулятор, а трассировщик, анализирующий путь исполнения программы и строящий таблицу изменений на основе этих данных. Как показывает практика, написать полноценный эмулятор чрезвычайно сложно, поэтому приходится комбинировать эти способы. В некоторых полиморфах и метаморфах можно встретить весьма хитрые методы запутывания кода, связанные с нестандартными опкодами инструкций. Большинство современных дизассемблеров и отладчиков могут неправильно дизассемблировать код, содержащий множественные префиксы либо неверное значение в поле Reg байта MOD r/m для Extended группы опкодов. В качестве примера можешь взять байты C7 F0 34 12. Большинство дизассемблеров (IDA в том числе) дизассемблируют это как mov eax, 1234h, а на самом же деле это INVALID OPCODE. Такие инструкции могут использоваться в защите для неявного вызова обработчика SEH, поэтому при создании дизассемблера следует учесть их. Для анализа таких инструкций в качестве образцового можно использовать мой дизассемблер, который лежит на диске (если найдешь в моем детище ошибку, обязательно сообщи мне).



Дизассемблируем программулину, защищенную DotFix FakeSigner

Качественный метаморф ты можешь найти в ExeCryptor'e. Он имеет большой разброс генерируемых инструкций, самомодифицирующийся код и много SEH. Все это приправлено немалой глубиной морфинга. Декодировать его сложно, но в этом нет ничего невозможного. Попробуй.

АНАЛИЗ ВИРТУАЛЬНЫХ МАШИН

В самых крутых защитах (в частности StarForce 3 Pro) можно встретить следующий прием: часть кода защищаемой программы и самой защиты переводится в псевдокод и исполняется на виртуальной машине. Для полного снятия такой защиты нужно восстановить этот код, а значит, придется исследовать ВМ и разбираться в алгоритмах ее работы. В общем случае ВМ, применяемые в защитах, делятся на два типа: эмулирующие исполнение »

псевдокода и использующие вызовы функций ВМ из native-кода.

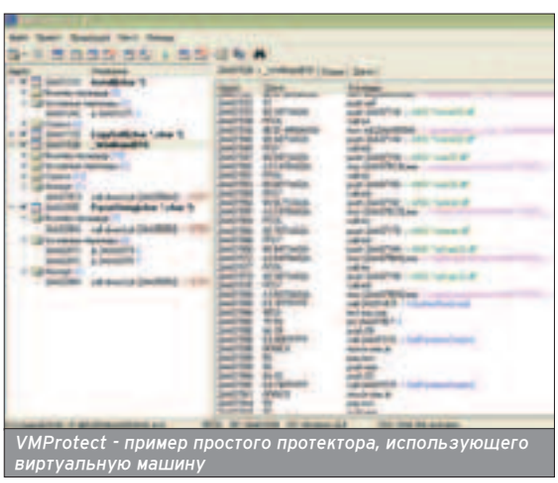
В первом случае виртуальная машина получает код операции и по своим таблицам определяет производимые действия. Исполнение псевдокода полностью эмулировано в цикле работы машины.

Во втором случае формируется код, который для выполнения каких-либо действий напрямую вызывает функцию ВМ. То есть mov eax, ebx/mov ecx, edx может превратиться в нечто подобное mov eax, 10h/call vm_mov/mov eax, 20h/call vm_mov. Трудно сказать, какой их этих типов ВМ сложнее для взлома, так как все зависит от конкретной реализации. Я считаю, что самой сложной будет комбинированная реализация, при которой часть кода полностью эмулируется ВМ, часть превращается в вызовы ее функций, а остаток подвергается метаморфному преобразованию. По общим методам анализа ВМ никаких рекомендаций тоже не дам, потому что все целиком и полностью зависит от ее реализации. Единственный метод взлома - долгая и упорная медитация над кодом машины и осмысление всего происходящего в ней. И большой запас терпения.

Можешь попрактиковаться, например на VMProtect - простом протекторе. Если захочешь чего-нибудь посложнее, бери Star Force 3 Pro – защиту с ВМ комбинированного типа, которая к тому же работает и в ring3, и в ring0.

ДИНАМИЧЕСКАЯ РАСШИФРОВКА КОДА

Несомненно, самый старый, но до сих пор применяемый прием - это динамическая расшифровка кода. Его суть в том, что код защищаемой программы расшифровывается не сразу, а по мере его исполнения. Например, Armadillo со включенным CopyMem расшифрует только первую страницу программы и передаст ей управление. Когда та обратится к еще не расшифрованным данным, возникнет исключение, по которому протектор расшифрует еще часть кода, зашифровав старый. Таким образом, весь код программы никогда не присутствует в памяти, а значит, его нельзя сдампить.



VMProtect - пример простого протектора, использующего виртуальную машину

Armadillo в переводе с английского – броненосец

Для борьбы с армадилой многим приходилось реверсировать весь код протектора, чтобы заставить его расшифровывать защищенную программу полностью. Но это весьма долгий и тяжелый путь, поэтому я ломал армадилу по-другому. Если процесс нельзя сдампить снаружи, почему не сделать это изнутри? Для этого нужно записать код дампера в адресное пространство процесса с помощью WriteProcessMemory и выполнить его, перенаправив контекст одной из нитей процесса (с помощью SetThreadContext) на наш код.

Дальше внедренный в защищаемую программу дампер должен просто пройтись по секции кода и сбросить ее на диск. На CopyMem можно смело забить болт, так как при отсутствии страницы в памяти протектор, думая, что к ней обратилась сама защищенная программа, будет вынужден расшифровать ее и отдать нашему дамперу в готовом виде, так что Armadillo is fucking yourself. К сожалению, не все протекторы ломаются так же просто, как армадилла. Например, ExtremeProtector расшифровывает код не по обращению к занимаемой им памяти, а по исполнению этого кода, что сильно затрудняет его нахождение. Здесь хочешь не хочешь, но приходится лезть в сам алгоритм работы протектора.

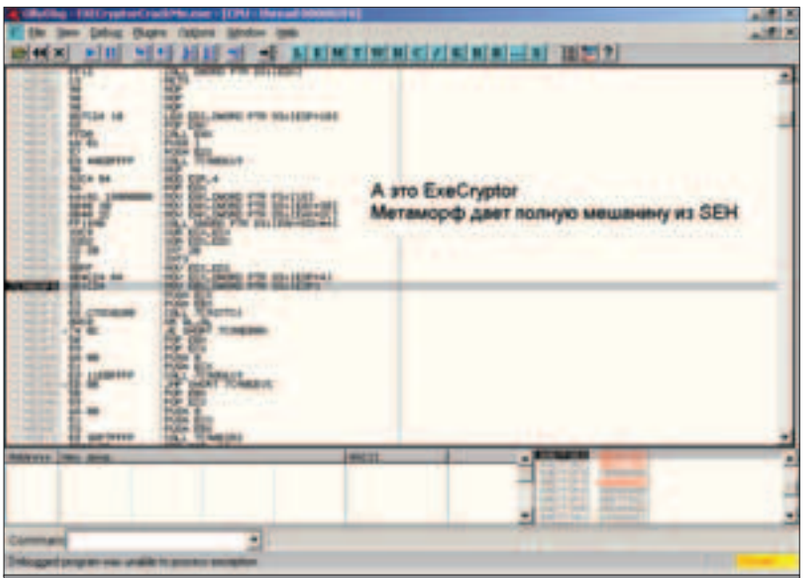
АППАРАТНЫЕ КЛЮЧИ

Существует определенный класс программных продуктов, предназначенных для узкого круга пользователей и при этом очень дорогих. Это, например, корпоративные системы для работы с базами данных, биллинговые системы для провайдеров, специфические программы, применяемые на заводах, в научных лабораториях и т.д. Эти программы стоят десятки или даже сотни тысяч долла-

SMM

SMM (System Managment Mode) - режим работы процессора, появившийся еще в x86-процессорах. Особенности этого режима не документированы и зависят в основном не от процессора, а от чипсета материнской платы. Процессор может войти в этот режим только аппаратно, по сигналу от чипсета, при этом обработчик SMM недоступен даже программам нулевого кольца. По уровню привилегированности этот режим самый высокий из всех, которые вообще возможны на x86-процессорах.

Одна из возможностей SMM, обеспеченных чипсетами - это вызов обработчика при обращении к портам ввода/вывода. Эта возможность предназначена именно для программной эмуляции оборудования, и ее можно использовать для эмуляции аппаратных ключей. Но, к сожалению, на различных чипсетах эта возможность реализована по-своему, а на каких-то просто отсутствует. На некоторых чипсетах (например nForce фирмы Nvidia) вся информация про SMM засекречена и никому не выдается. По умолчанию обработчик SMM устанавливается BIOS'ом и управляет работой кнопки Power на системном блоке. К сожалению, некоторые BIOS'ы блокируют возможность установки своего обработчика SMM, поэтому для работы эмулятора на таких компьютерах потребуется модификация BIOS (о ней читай в июньском "Хакере").

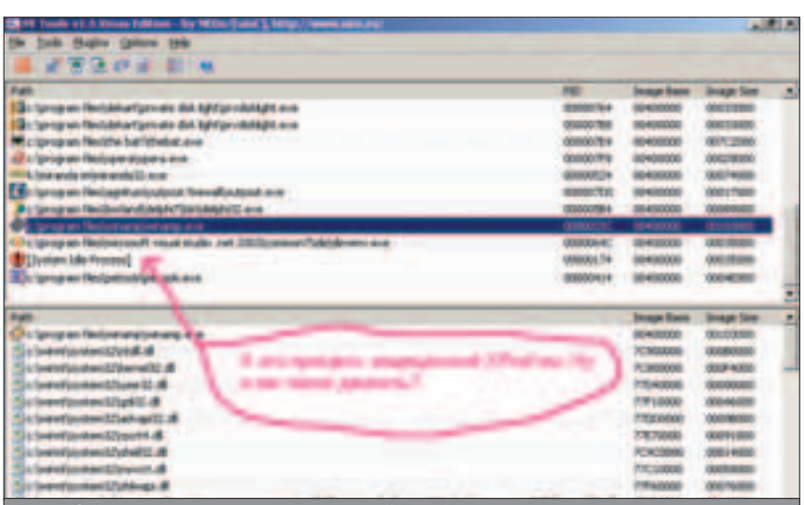


Полиморф ExeCryptor'а превращает код программы в кучу SEH

ров. Убытки автора обычных шароварок от взлома весьма велики. Но у этого класса программ круг пользователей настолько узок, что в условиях российского пиратства и при наличии крэков у авторов не было бы возможности продать даже одну копию. В таких случаях обычно прибегают к аппаратным ключам. Они обеспечивают привязку программы к специальному устройству, подключающемуся к USB или LPT, и при правильной реализации могут сильно усложнить взлом защиты. Несомненно, лидером продаж сейчас являются аппаратные ключи HASP. Фирма Aladdin Software Security выпускает целую линейку ключей, предназначенных как для USB, так и для LPT и отличающихся принципами своей работы.

Так как же работают аппаратные ключи? В простейшем случае к программе подключается модуль, который проверяет наличие ключа и сообщает программе результат. Кроме проверки наличия, могут также проверяться значения чисел, хранимых в нем, либо они будут использоваться для расшифровки кода программы. Как же бороться с этим? Если применяется простая проверка наличия ключа, эта борьба будет не напряженнее, чем с проверкой серийника: нужно просто найти код, вызывающий проверку, и отключить его.

В более сложных случаях можно заменить код библиотеки, которая используется для работы с ключом, на свой, в котором позже проэмулируешь ответы ключа. Перед этим нужно перехватить исполнение этого кода, получить все значения запрос/ответ, которыми обмениваются ключ и программа, занести их в таблицу, по которой потом будет идти эмуляция. Этот метод называется методом табличной эмуляции. Программы с плохо навешенным хаспом очень хорошо поддаются ему. Но, к сожалению, в реале встречаются далеко не такие простые случаи. Иногда защиту делают программисты, которые в этом деле действительно понимают (это можно увидеть на примере защиты 1С). Копать саму программу, защищенную такими профи, действительно трудно, и проще всего подойти к делу с другой стороны - эмулировать ключ на низком уровне.



Попробуй сдампить процесс, защищенный Xprot'ом

Электронный ключ HASP для USB

С ключом обычно идет драйвер, который обрабатывает запросы программы и возвращает ответы ключа. Этот драйвер тоже неплохо защищен, поэтому нужно работать ниже - непосредственно с самим оборудованием. Если попался USB-ключ, то можно считать, что тебе очень повезло, так как защита с USB работает не напрямую, а доверяет это дело системным драйверам. А незащищенные системные драйверы - это хорошее место для внедрения нашего эмулятора. Проще всего будет заменить точки входа в обработчики IRP-драйвера USB так, чтобы направляемые ему запросы попадали нам, после чего можно фильтровать их и эмулировать ответы ключа. Для этого тебе придется изучить работу ядра системы и научиться программировать драйверы - от этого никуда не денешься.

Более сложный случай - LPT-ключ. Драйверы защиты работают с ним напрямую через порты, что сильно затрудняет эмуляцию. Я пока знаю только два рабочих способа эмуляции такого ключа: это изменение кода защитного драйвера и перехват обращений к портам с помощью SMM. Первый способ проще, но требует хорошего знания методов реверсинга, а второй позволяет сделать универсальный эмулятор, но здесь есть неприятность: он будет сильно зависеть от используемого железа. Самое трудное, что может попасться - это последнее поколение аппаратных ключей, которые хранят в себе часть кода защищенной программы, и достать его оттуда программным способом не представляется возможным. Единственное, что делает взлом таких ключей возможным - это низкая скорость работы такого кода. Это означает, что в ключ будут помещать код не критичной для работы программы, а выполняющий какие-либо вспомогательные действия. Такие куски кода можно просто написать самостоятельно.

ПАРА СЛОВ НА ПРОЩАНИЕ

Забавно, но во взломе сложных защит практически нет ничего действительно сложного - нужен лишь подход ко всему с умом. Надеюсь, после прочтения этой статьи ты уже не будешь тормозить всякий раз, встретив что-то посложнее аспака, а спокойно и хладнокровно возьмешься ломать. Пробуй, тренируйся, если что - пиши.

GPcH (admin@dotfix.net)

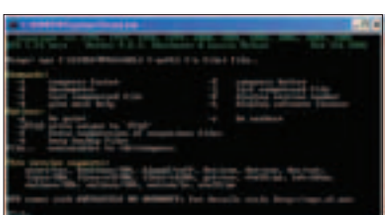
УПАКУЕМ ЗА РАЗ!

ОБЗОР УПАКОВЩИКОВ

Какеров наплодилось немало. Почти каждый кодер, разобравшийся в формате PE, считает своим долгом написать свой собственный упаковщик, превосходящий по тем или иным характеристикам все существующие. Порой он добавляет в пакер простые антиотладочные механизмы, придумывает что-нибудь новенькое неприятное для крэкера, и его упаковщик уже называют протектором. В общем, этот обзор поможет тебе разобраться в том, какие пакеры и когда лучше использовать, как при необходимости быстро снять их.

UPX BY MARKUS OBERHUMER & LASZLO MOLNAR

Данный пакер, думаю, знают все. Его последняя версия - 1.93. Довольно неплохо пакует, здорово развивается. На сайте разработчика лежит почти



UPX by Markus Oberhumer & Laszlo Molnar

полный исходный код упаковщика (без библиотеки NRV). И сам пакер, и упакованные им EXE-файлы запускаются под всеми версиями Windows. Если говорить о степени сжатия, то программа поддерживает девять режимов упаковки. Тот, который пакует быстрее, сжимает хуже, и наоборот. Но даже на максимальном режиме UPX не удерживает первенство по крутости упаковки, хотя вполне на уровне. В плане распаковки все отлично, пакер сам поддерживает эту возможность и без труда может восстановить до исходного состояния обработанный им же файл (ключ командной строки "upx -d <имя файла>").
URL: http://upx.sourceforge.net

ASPACK BY ALEXEY SOLODOVNIKOV

Тоже знаменитый пакер. Последняя версия - 2.12. Развитие этого упаковщика прекратилось довольно давно, он уже несколько лет не обновлялся, хотя по-прежнему используется многими. Пакует средне. Лучше всего жмет Delphi-программы. Имеет два режима сжатия: "обычное" и "максимальное". Разработчики не обделили его и многоязычным интерфейсом, а также некоторыми функциями для настройки. Ни о каких исходниках речи не идет: программа распространяется по шароварному типу и

ASPack by Alexey Solodovnikov

требует денег для регистрации. Что касается распаковки, на сегодняшний день мне известно более десяти универсальных распаковщиков ASPack'a, лучшие из которых, на мой взгляд, это Caspr и Stripper v2.7 (v2.11 и старше не предназначены для снятия ASPack'a).
URL: www.aspack.com

EXE32PACK BY STEELBYTES

Довольно редкий пакер. Последняя версия, которую мне удалось найти - 1.38. Упаковщик несколько лет не обновлялся, но при этом довольно неплохо пакует и оптимизирует файлы. Интерфейс программы консольный. Как я понял, разработка коммерческая, но куда перечислять деньги для покупки, непонятно. Степень сжатия иногда даже больше, чем у UPX. Совместимость тоже ничего: все тестируемые EXE-файлы работали у меня и на линейке Win9x, и под XP. Распаковщиков для данного пакера я не встречал.

FSG BY BART/XT

Упаковщик, популярный в кругах крэкеров и демомейкеров. Последняя версия - 2.0. С прошлого года не обновлялся, хотя, как мне кажется,

FSG by bart/xt

лишь потому что не было повода для этого. Отличное решение для упаковки маленьких программ, написанных на ассемблере или C\C++. Наверное, главная особенность данного пакера - размер его загрузчика, составляющий всего 158 байт, которых реально хватает для полноценной распаковки и создания импорта. Такую оптимизацию можно увидеть, наверное, только в FSG. Степень сжатия изменять нельзя, при этом установленная по умолчанию обходит почти все пакеры из данного обзора, а иногда и вообще все имеющиеся. Совместимость на очень высоком уровне. Распаковщиков специально для последней версии не видел, но generic unpacker'ы распаковывают его на ура. Движок упаковщика - aplib.
URL: www.xtreeme.prv.pl

MEW BY NORTHFOX

Малоизвестный в программерской среде, хорошо развивающийся пакер. Последняя версия - 11.1.2. Программа имеет отличную степень сжатия. Главная особенность и крутость этого упаковщика заключается в использовании одновременно двух движков упаковки - aplib и lzma. Последний из них применяется в архиваторе 7z и уже давно известен своей мощностью сжатия. Но этот движок медленный, поэтому запакованные программы хоть и имеют меньший, если сравнивать с показателями других пакеров, размер, зато тормозят при за-

MEW by Northfox

грузке, что, несомненно, минус. Однако можно отключить lzma в настройках пакера - грузиться будет быстрее, но и размер увеличится. Выбирай сам, что для тебя важнее - размер или скорость. Распаковщика для последней версии я не видел, но generic unpacker'ы опять на высоте, да и руками распаковать несложно, главное - уметь.
URL: http://northfox.uw.hu

NEOLITE BY NEOWORX INC

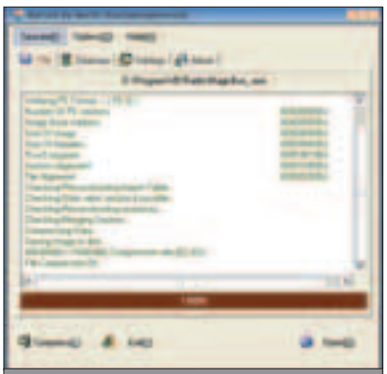
Еще один малоизвестный пакер. Последняя версия - 2.0. Не обновлялся уже пять с лишним лет, и все о нем уже забыли. Работает неплохо, имеет много настроек, умеет распаковывать упакованные им же файлы. Главная особенность - умение сжимать не только код, но и экстраданные, что пригодится при паковке программ, на-

NeoLite by NeoWorx inc

писанных на FoxPro, и роликов, сделанных на Flash'e. Программа коммерческая, поэтому все упакованные программы при запуске выдают диалог с сообщением о незарегистрированности. Защищенные Demo-версией программы работают под всеми Windows.
URL: www.neoworx.com

NSPACK BY LIU XING PING

На мой взгляд, лучший пакер и по сжатию, и по совместимости. Последняя бесплатная версия - 1.3, последняя платная - 2.3. Его автор вовремя понял, что такой классный компрессор нельзя распространять бесплатно. Пакер динамично развивается, хотя кроме изменения интерфейса я мало что могу сказать об улучшениях платной версии - бесплатный v1.3 ничуть не хуже. Все это благодаря не

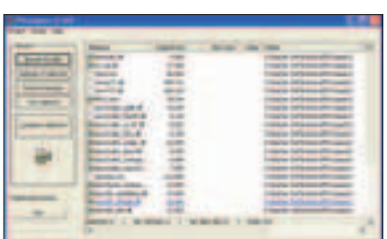


nSPack by Liu Xing Ping

только использованию двух движков упаковки, aplib и lzma, но и их оптимальному применению. В общем, must have. Специальных распаковщиков нет, однако generic берут его без проблем.
URL: www.nsdsn.com

PECOMPACT BY BITSUM TECHNOLOGIES

Довольно известный пакер. Последняя версия - 2.5. Имеет немало интересных фишек, к примеру пакетное сжатие. Ты можешь выбрать для сжатия сразу несколько программ и запаковать их в момент. Еще одной отличительной чертой является широкий выбор движков сжатия данных. Среди них есть и уже известные aplib и lzma, и свой - ffce. Совместимость замечательная: программы нормально запускаются под всеми поддерживаемыми Windows. Главный минус, который ограничивает использование этого пакера - его платность. А платить любят не все. Что касается распаковщиков, есть UnPecompact by yoda/f2f. Также неплохо с задачей справляется универсальный quick unpack v0.7.
URL: www.bitsum.com



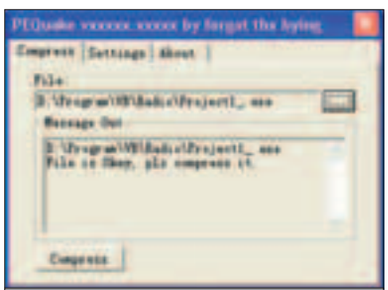
PECompact by Bitsum Technologies

ЧТО ТАКОЕ GENERIC UNPACKER?

Generic Unpacker'ы - это универсальные распаковщики, которые, в принципе, могут кое-как распаковать любой пакер и восстановить таблицу импорта. Ясен пень, такие распаковщики не ребилдят ресурсы, поэтому если ты хочешь распаковать программу для ее последующей русификации в Restorator'e, то они не помогут, хотя сэкономят время для взломщика программных защит: ему на ресурсы наплевать, лишь бы код был распакован. Лучшими представителями таких распаковщиков являются Quick Unpack by FEUERRADER, GUW by Christoph Gabler, а также плагин для распаковки программ, встроенный в файловый анализатор PEiD. Какой из них круче, решишь сам.

PEQUAKE BY FORGAT/UNKNOWN

Прикольный китайский упаковщик. О версии ничего сказать не могу, так как он релизился всего один раз и то не на сайте разработчика, а на форуме. Почему я решил описать его? Да потому что это интересный пакер, хотя сжимает он средне. Главным приколом является то, что автор обо-



PEQuake by fORGAT/Unknown

шел тормознутость движка сжатия statusbar'ом, который каждый раз пробегает по экрану при загрузке EXE-файла. Забавно, что сказать. Распаковщик к нему тоже есть, но в виде скрипта к Olly Debugger'y, который написал Mario555. Программ, упакованных этим пакером, не встречал.

PESPIN BY CYBERBOB

Новый здорово развивающийся пакер. Последняя public-версия - 1.1. 1.2 уже есть, но автор ее не распространяет. Упаковщик бесплатен и работает, как и большинство других, под всеми версиями Windows. Главный минус: запакованные программы жутко тормозят даже на четвертом пне.

PESpin by cyberbob

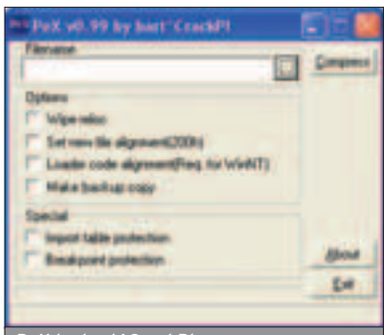
Используемый движок - aplib. Распаковщика к новой версии я не видел, но ребята с форумов уже давно распаковали сам пакер, отсюда вывод - нужно уметь распаковывать программы руками, а не ждать, пока выйдут автораспаковщики.
URL: http://pespin.w.interia.pl

PEX BY BART^CRACKPL

Практически никому не известный пакер. Последняя версия, которую я видел - 0.99 beta. Он меня очень порадовал. Пакует не сильно, зато распространяется с исходником на ассемблере, поэтому, если знаешь ассемблер, можешь смело доработать пакер »

Немало новых пакеров ты сможешь найти на www.wasm.ru в разделе "Инструменты".

Множество старых упаковщиков откапывай тут: www.exetools.com.



PeX by bart^CrackPl

под себя и пользоваться. Упаковщик умеет протектить код от распаковки, из-за чего не могу не порекомендовать его тем, кому нужен бесплатный пакер/протектор с возможностью доработки. В качестве движка, как обычно, используется aplib. Распаковщиков не видел, как и программ, упакованных данной утилитой.
Если вдруг будешь пользоваться им, не забудь поставить в настройках галку "Loader code alignment(Req. for WinNT)", иначе приложения, пакованные под NT, не будут работать.

PKLITE32 BY PKWARE

Наверное, старейший упаковщик, известный еще со времен MS-DOS. С переходом на Win32 утилита стала шароварной. Последняя версия, которую я видел, датирована 1999 годом,

PKLITE32 by PKWARE

и с тех пор пакер не обновлялся. Рекомендую посмотреть его лишь чтобы знать, с чего все начиналось. Особых настроек и крутого сжатия тут нет, но какой крутизны можно ждать от этого старого средства? Распаковщики есть, но толку от них я не вижу: пакер все равно не используется.
www.pkware.com

Очень рекомендую сходить на FTP нашей команды download.int3.net или напрямую на www.int3.net. Здесь немало и новых, и старых упаковщиков, плюс ко всему это, наверное, единственный ftp, где лежат почти все версии UPX.

TELOCK BY TE!

Довольно известный пакер/протектор. Время от времени впадает в развитие. Особенно известны его версии 0.71 и 0.98. Кроме паковки, tElock умеет неплохо защищать код от распаковки, хотя распаковщики все равно были написаны, и найти их на прос-

У тебя есть шанс найти на диске все программы обзора - рискни ;).

tElock by tE!

МНЕНИЕ ЭКСПЕРТА

Упаковка исполняемых файлов преследует две цели: уменьшить размеры файла на диске и затруднить исследование машинного кода. За это приходится платить замедлением загрузки и ростом потребностей в оперативной памяти. Неупакованный файл Windows автоматически подгружает с диска по мере необходимости, а при вытеснении из памяти просто передает страницы памяти другому процессу (разумеется, предварительно обнулив их). Упакованный файл грузится в память целиком, распаковывается, потребляя еще больше памяти, и, что самое неприятное, при его вытеснении Windows вынуждена сбрасывать страницы памяти в файл подкачки, так как взять их непосредственно с самого файла уже не получится - он же упакован!

Остается только затруднение анализа. Однако для всех популярных упаковщиков существуют автоматические распаковщики, а если и не существуют, файл нетрудно распаковать и вручную. Главное - найти точку входа. Это несложно. В начале каждой нормальной программы присутствует библиотечный код, который легко опознать по сигнатуре или по вызову API-функции GetModuleHandleA.

Основную сложность представляет восстановление таблицы импорта (многие упаковщики затирают ее после распаковки), PE-заголовка с таблицей секций (упаковщики издеваются над ней тоже) и, возможно, атрибутов страниц памяти, которые модифицируются упаковщиками, как им заблагорассудится. К счастью, все эти операции нетрудно автоматизировать, и уже существует множество утилит, освобождающих хакеров от бремени ручного труда. К тому же для исследования программы иметь работоспособный дамп совершенно необязательно! Достаточно просто снять полный дамп процесса и загрузить его в дизассемблер.

Чтобы IDA Pro смогла распознать имена функций, соответствующая библиотека сигнатур должна быть загружена вручную. А для распознания API-функций следует подключить map-файл каждой из DLL. В остальном же дизассемблирование дампов памяти ничем не отличается от анализа обычных приложений. Конечно, упакованный файл не может быть модифицирован, и заменить 7x на EB у нас вряд ли получится, поэтому приходится либо писать генераторы регистрационных номеров/ключевых файлов, либо прибегать к помощи онлайновых патчеров (лоадеров), правящих образ файла в памяти на лету.

Некоторые упаковщики внедряют в файл различные антиотладочные механизмы, перекомпилируют защитные процедуры в p-код, используют динамическую шифровку и делают множество других нехороших вещей, в результате чего размер файла только возрастает, а упаковщики превращаются в протекторы. Затрудняет ли это анализ? В какой-то мере да. Во всяком случае, лобовая атака становится невозможной, и хакеру приходится искать обходные пути. Однако редкий протектор обходится без ошибок, и у легальных пользователей появляются проблемы. Программа отказывает в регистрации или конфликтует с другими приложениями. А этого допускать ни в коем случае нельзя! Конфликтные издержки часто превышают ущерб, нанесенный хакерами. Любую популярную программу все равно взломают и выложат крэк. Защита лишь чуть-чуть увеличивает время от выпуска программы в свет до появления крэка, но редко подстегивает продажи.

В общем, к упаковке приложений следует подходить основательно или вообще не подходить. То же самое можно сказать и о распаковке. Некорректная распаковка проявит себя глюками, выскакивающими в самых неожиданных местах. Правильные хакеры пользуются только генераторами или правкой в памяти.

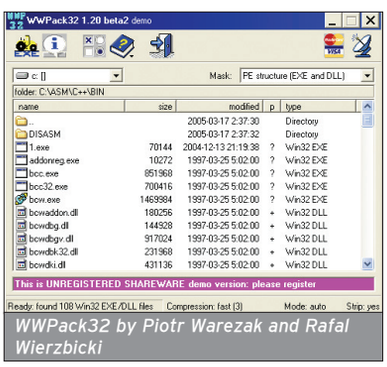
торах Сети не составит труда. В качестве движка автором был использован так любимый всеми aplib. Интерфейс удобный, настроек много, freeware. В общем, советую скачать хотя бы для того, чтобы посмотреть.
Пакер легко ищется поисковиками, несмотря на то, что сайта у него нет. Если будешь искать распаковщики, то обрати внимание на WKT_tElock_Dumper. Если не поможет, поищи tEunlock - тоже довольно неплохо распаковывает. Если и он не справится, то знай, что скриптов для Olly для распаковки написано немало, в том числе для этой утилиты.

UPACK BY DWING

Новый, динамично развивающийся пакер. Уже начинает обходить nSpack. Последняя версия на момент написания статьи - 0.22 beta. Действительно мощно жмет и оптимизирует файл. Одно то, что импорт кладется в заголовок, уже заставляет задуматься над тем, насколько хорошо автор решил провести оптимизацию. В качестве движка используется только lzma (наверное, поэтому и жмет хорошо). Совместимость тоже на уровне, хотя автор пишет: "So if it does't pack an exe-file, try UPX first. If UPX can pack it normally, send the original exe-file to me. Remember that it can't pack some weird exe-files. So you'd better backup your exe-file before packing it.". Настроек мало, но они и не нужны, так как все пакуется быстро и качественно. Пакер консольный, и это особенно удобно: прописал его один раз в батник, компилящий программу, и забыл.
URL: http://dwing.spymac.net

WWPACK32 BY PIOTR WAREZAK AND RAFAL WIERZBICKI

Еще один старенький шароварный пакер с документацией на 2 Мб. Зачем такая дока? Видимо, маркетологи решили, что так можно будет повысить объем продаж. Что ж, когда пакеры только зарождались, на этом реально можно было заработать. Сейчас же есть куча бесплатных упаковщиков, и заработать в условиях такой конкуренции очень сложно.
Возможности пакера весьма стандартны, хотя есть и интересные моменты. Например, в программе присутствует встроенный проводник, из которого можно выбрать файлы для упаковки. Кроме всего прочего, имеется целых 20 режимов сжатия. Не обольщайся: nSpack и Upack компрессят EXE-файлы все равно круче, хотя и не имеют всех этих наворотов. В общем, посмотреть пакер стоит, но пользоваться - вряд ли: он старовat для сегодняшних запросов к степени сжатия.
URL: www.webmedia.pl/wwpack32



WWPack32 by Piotr Warezak and Rafal Wierzbicki

GHF PROTECTOR BY GPCH

Да, это не опечатка: автор этого пакера/протектора действительно я. Он был написан мной на основе двух opensource-движков. В качестве упаковщика используется AHPack (его в свою очередь использует aplib), а в качестве протектора кода от распаковки - Morphine. Несмотря на то, что движки разные, я все-таки смог совместить их и написать нормальную утилиту. Естественно, программа бесплатная и распространяется с исходниками на Delphi. Так что смело можешь переписывать ее под себя, главное - не забывай о лицензии GNU GPL, если решишь распространять свою доработку. С помощью моего пакера можно и упаковать, и защитить, можно сделать и то, и другое. В опциях это изменяется легко, так же, как и возможность не удалять из ресурсов программы иконку и XP Manifest. Совместимость на уровне 75%: иногда бывают косяки с некоторыми EXE-файлами, но в общей массе этого не заметно. К моменту написания статьи нашелся человек, который написал распаковщик к данному пакеру, который называется GHF UnProtector и лежит там же, где и сам упаковщик.
Ну а чтобы не было лишних вопросов к названию, объясню: программа была названа по первым буквам ников его разработчиков GPcH, Holy_Father (автор Morphine'a), FEUERRADER (автор AHPAck'a).
URL: http://reversing.dotfix.net

ВЫБЕРИ СЕБЕ УПАКОВЩИК

"Какой же пакер выбрать," - спросишь ты? Вопрос сложный. Из проверенных и хороших рекомендую UPX и FSG. Если нужно экстрасжатие, хватай nSpack или Upack. Первый тестировался мной на многих осях - глюков не замечено, очень советую. Если есть желание не только запаковать, но и немного защитить файл, причем бесплатно, бери tElock или GHF Protector. Если нужен пакер, легко дорабатывающийся под себя и распространяющийся в исходниках, то тут все зависит от того языка программирования, который ты знаешь. Если это Delphi, то используй GHF Protector, если это ассемблер - тебе наверняка придется по душе PeX. Свой выбор я уже давно сделал в пользу nSpack, однако меня не сильно порадовало, что он стал платным и теперь продается по $30 штука.

Gelios

ПОРТАТИВНЫЙ ВЗЛОМ

ЛОМАЕМ КПК НА WINDOWS MOBILE

Большинство из нас имеют хотя бы элементарное представление о процессе взлома программ для персональных компьютеров. Но наверняка не каждый знает о том, как то же самое можно проделать с софтом, предназначенным для различных мобильных устройств.

Весь материал, содержащийся в данной статье, приведен исключительно в учебных целях. Использование материалов статьи в преступных целях в первую очередь подпадает под статьи 146 и 273 УК РФ. Автор не несет ответственности за возможное применение этой информации на практике.

Итак, к мобильным устройствам, конечно же, можно отнести КПК на Windows Mobile, Palm'ы а также смартфоны и другие устройства. Общий подход к их взлому такой же, как и для ПК. Есть одно существенное отличие, которое как раз и затрудняет исследование программ под различные платформы на начальном этапе: все устройства используют различные типы процессоров. А это значит, что в каждом устройстве применяется свой ассемблер и своя система команд. Поэтому далее ограничусь рассмотрением процесса взлома программ для КПК на основе платформы Windows.

ОБЩИЕ СВЕДЕНИЯ

Для начала немного теории. Многие характеристики и свойства, которые я опишу, относятся не только к карманным компьютерам на платформе Windows Mobile (ранее она называлась Pocket PC). В отличие от персональных компьютеров, КПК на Windows Mobile используют так называемые ARM-процессоры, которые основаны на RISC-архитектуре (Reduced Instruction Set Computer), в то время как в "обычных" процессорах используется CISC (Complex Instruction Set Computer). Ну и зачем я пишу все это? Положим на место подробности архитектуры и рассмотрим только те свойства, которые пригодятся в дальнейшем. Во-первых, эти отличия означают, что мнемоника и формат ассемблерных инструкций для таких процессоров различны. Тем не менее, в большинстве случаев названия инструкций совпадают, поэтому очень желательно знание системы команд "стандартного" ассемблера. Во-вторых, RISC-архитектура содержит больше регистров, которые используются намного шире, чем в ПК. Для сравнения: в процессорах линейки Pentium свободно можно использовать четыре регистра общего назначения (eax, ebx, ecx, edx), остальные же, такие как esp или ebp, трогать не рекомендуется, в то время как в ARM-процессорах можно практически свободно использовать до 10-13-ти регистров. Остальные важные отличия будут отмечены, когда мы непосредственно столкнемся с ними на конкретном примере.

По сути, карманный компьютер - это такой же ПК, только малогабаритный. У него имеется центральный процессор, оперативная и постоянная память, аудио- и видеоподсистемы, а также различные слоты расширения. Логично было бы предположить, что используемые в этих устройствах операционные системы тоже имеют общие черты. Действительно, так оно и есть. КПК на Pocket PC используют ОС Windows CE, недавно Microsoft переименовала ОС и платформу в Windows Mobile, но смысл от этого не изменился. Структура этой ОС во многом схожа с версиями Windows, которые используются в настольных ПК.

И опишу две важные для нас особенности Windows CE. Во-первых, Windows CE также имеет в своем распоряжении системный реестр. Поэтому, например, информация о настройках программы часто хранится именно в реестре. Во-вторых, Windows CE и версии Windows для ПК используют абсолютно те же имена функций WinAPI, и, более того, во всех (или почти всех) случаях эти API принимают одни и те же параметры. Следовательно, структура программ для рассматриваемых карманных компьютеров является такой же, как и для ПК, что значительно упрощает исследование программ.

ЧТО НАМ ПОНАДОБИТСЯ?

Для исследования программного кода, конечно же, понадобятся какие-то инструменты. Некоторые являются специфическими, некоторые можно взять из "стандартного" набора программ, используемых для анализа софта ПК.

Первый и главный инструмент, без которого не обойтись при исследовании программ всех без исключения КПК и телефонов - это, конечно же, дизассемблер IDA. Он поддерживает все типы процессоров, которые могут встретиться. Здесь важно, что программы для КПК используют свои DLL, то есть файлы типа user32.dll, используемые в обычных ПК, просто не подойдут. В последней версии IDA Pro 4.8 эти библиотеки получают в наличие сразу после установки, однако во всех предыдущих версиях их нет. Ищи их на сайте www.ka0s.net, посвященном исследованию КПК - тут можно найти много полезных статей и документов.

Далее понадобится какой-нибудь шестнадцатеричный редактор, например HIEW. Также будет очень нужен отладчик. Вот здесь и возникает проблема: дебаггеров для отладки программ под КПК почти нет! Вообще-то можно обойтись и без дебаггера, использовать только дизассемблер, но это работает далеко не всегда. Однако Microsoft о нас все-таки не забыли :). Они не поленились и написали среду разработки программ для мобильных устройств - eMbedded Visual C++, которая, к всеобщему счастью, содержит встроенный отладчик. Последнюю версию этой среды, а также Service Pack (его желательно поставить) можно найти и бесплатно скачать с www.microsoft.com/windowsmobile. Очень рекомендую скачать там же Microsoft Pocket PC 2003 SDK - пригодится при отладке.

Может пригодиться и какой-нибудь редактор ресурсов. Спешу еще раз предупредить: формат исполняемых файлов для многих типов КПК и для персональных компьютеров один и тот же. Иными словами, такие файлы без проблем воспринимаются стандартными программами типа Resource Hacker, PETools и т.п.

Последнее, что понадобится - это сам КПК, если, конечно, ты собираешься отлаживать программу, а не пы-

таться анализировать код только с помощью IDA. Также желательно иметь достаточно интернета, чтобы скачать все описанные инструменты :).

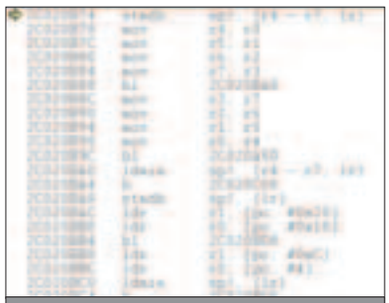
ЭТАП ПЕРВЫЙ. СНИМАЕМ ОСНОВНЫЕ ОГРАНИЧЕНИЯ

Когда все необходимые инструменты установлены и находятся под рукой, можно приступать непосредственно к исследованию программ. Для примера возьмем игру Plant Tycoon версии 1.0, и этот выбор не случаен. Во-первых, игра далеко не новая, поэтому взломана давно, так что, надеюсь, мы уже не нанесем материального ущерба ее автору. А во-вторых, на этой программе можно наглядно показать процесс создания патча. Смысл игры состоит в выращивании и скрещивании разных растений - что-то вроде тамагочи.

Итак, подключаем КПК к компьютеру, устанавливаем подопытную программу и копируем исполняемый файл на ПК, чтобы можно было работать с ним. Затем запускаем IDA: пока она будет анализировать файл, будем разбираться с другими инструментами. Пока открыто лишь пустое окно дизассемблера, поэтому выбираем пункт Open... и указываем наш скопированный файл. Появляется окно, в котором указываются различные опции. Здесь выставлены опции по умолчанию, но нужно проанализировать программу под КПК, поэтому наш главный интерес - поменять тип процессора. В данном случае нас заинтересовал ARM-процессор. В списке их несколько, однако выбираем тип "ARM processors: ARM", так как он является наиболее универсальным. Жмем OK и, если используется IDA версии ниже 4.8, указываем пути к запрашиваемым библиотекам. Когда начнется анализ, можно свернуть окно дизассемблера и заняться отладчиком.

Запускаем eMbedded Visual C++ и также через Open открываем exe'шник. Сразу же над рабочей областью находятся несколько выпадающих списков, значения в которых определяют режим отладки. Так как

Выбираем тип процессора

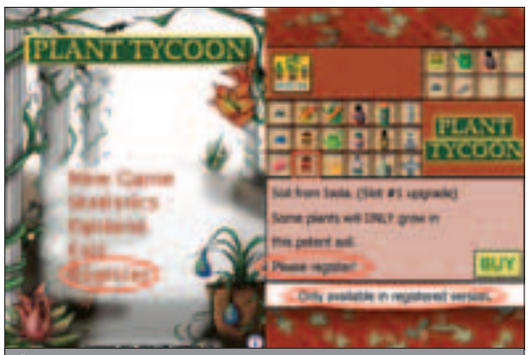


Точка входа в exe

установлен Pocket PC 2003 SDK, в списках должны быть соответствующие значения. В самом правом из них выставлен режим эмулятора, который требуется изменить на режим реального устройства, так как при отладке будет использоваться сам КПК. Нужно сказать, что можно было бы использовать и эмулятор, но он может попросту отказаться работать. Теперь все готово, осталось только запустить программу, поэтому жмем <F11> (запуск программы в режиме пошагового исполнения). Пока происходит соединение с устройством и загрузка файла, отладчик может попросить указать пути к некоторым библиотекам. Здесь можно просто жать Cancel, так как отладка от этого зависеть не будет. Если все прошло успешно, откроется окно с командами, а мы будем стоять на точке входа (Entry Point) в программу.

Как показывает картинка, многие команды схожи с командами ассемблера, используемого на обычном ПК. Единственное отличие - названия и количество операндов. Подробная документация по системе команд для ARM-процессоров лежит на сайте www.ka0s.net. Теперь, чтобы можно было приступить к взлому, хорошо было бы познакомиться с ограничениями программы, для чего нажимаем <F5> и наблюдаем, как запускается игра. В описании сказано, что в незарегистрированной версии в магазине игры нельзя покупать почти ничего - вот такая неприятность. Помимо этого, становится не очень радостно, когда в меню находится пункт Register. Это все и предстоит каким-нибудь образом отключить.

Начнем с ограничений при покупке вещей. Для этого открываем IDA, которая наверняка уже успела проанализировать файл, и в окне Strings window ищем строку "Please register!". Как оказалось, такая строка не одна - их целых 12! Ну что ж, попробуем пойти по какой-нибудь из них, например по той, чей вызов происходит по адресу 24EF4 (это адрес смещения не в файле, а в памяти после загрузки программы). Интересно, что все эти строки расположены рядом со строками, содержащими описание продукта в магазине. Причем все эти описания расположены в файле друг за другом, а количество продуктов, которые нельзя купить в магазине незарегистрированной версии игры, равно числу

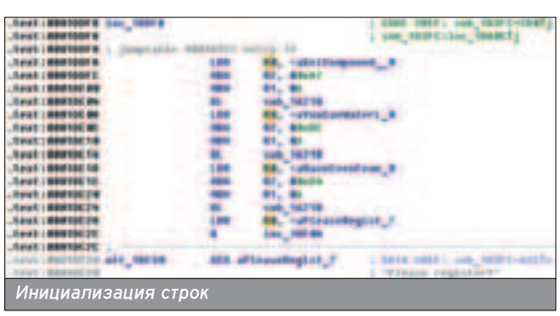


Основные ограничения программы

этих описаний. Очень хорошо. Посмотрим, где в коде имеются обращения к этим строкам: выделяем имя переменной, которая соответствует какой-нибудь из строк, из контекстного меню правой кнопки мыши выбираем Jump to xref to operand или просто жмем кнопку x. IDA находит только одно обращение. Что ж, это хорошо. Что находится по найденному адресу? Всего лишь какой-то указатель на строку, то есть не совсем то, что нужно. Если посмотреть код прямо над этими строками, то можно увидеть, что как раз там и происходит обращение к найденному указателю. Эта функция начинается по адресу 1DDF8, и, похоже, именно в ней происходит инициализация строк.

Теперь также попробуем найти, где происходит вызов данной функции, для этого выделяем строчку, содержащую loc_1DDF8, и применяем поиск xref (cross reference), как было описано выше. На этот раз IDA выдала два места, которые так или иначе связаны с вызовом функции. Для начала попробуем пойти по первому обращению, находящемуся по адресу 1D9CC. И тут мы видим некую структуру, похожую на switch в C++, то есть условный переход, зависящий от значения какой-то переменной. В данном случае выясняется, что таких состояний 18, что соответствует количеству предметов в магазине игры. Из всего делаем вывод, что выбор какого-либо предмета обрабатывается отдельной функцией.

Здесь, опять же, видно, что переход к этому коду осуществляется условно где-то выше, о чем говорит пунктирная стрелка слева от надписи loc_1D9C8. Опять смотрим обращения к этому коду и находим только одно место по адресу 1D484. Вот это уже больше похоже на то, что мы ищем, а именно переход на switch в зависи- »



Инициализация строк

мости от какого-то условия с помощью условного перехода BEQ. Действительно ли это одно из условий, выводящих строку о просьбе зарегистрироваться? Чтобы проверить, открываем окно отладчика и жмем <CTRL>+<G> - появляется окно, в котором нужно указать адрес, в него мы и хотим перейти. Но тут вводится абсолютный адрес в памяти, а не смещение, поэтому первые три байта нужно посмотреть в основном окне. В данном случае это 2C0 (на разных машинах возможны разные значения), поэтому вводим 2C01D484, жмем Go To и оказываемся на том коде, что мы видели в IDA. Нажимаем правую кнопку мыши на адресе 2C01D484 и выбираем пункт Insert/Remove Breakpoint, после этого пробуем уже на самом КПК, войдя в магазин игры (shop), выбрать какой-нибудь предмет. И вот тут-то срабатывает breakpoint.

Теперь в окне регистров (его можно открыть, нажав на соответствующую кнопку на панели инструментов) смотрим значение флага Zero, который отвечает за условный переход BEQ. Здесь он равен единице, то есть переход будет осуществлен. Два раза кликнем на флаге Zero, чтобы поменять его значение на ноль, нажмем <F5> для продолжения работы программы и посмотрим результат. Отлично! Теперь вместо строки "Please register!" можно посмотреть цену предмета. Останавливаем программу в отладчике (<Shift>+<F5>). С помощью hex-редактора открываем на компьютере "подопытный" файл и переходим по адресу 1D484. Если используется hiew, то вводим ".1D484", так как это смещение не в файле, а в памяти, как уже упоминалось. По этому адресу находятся байты 4F 01 00 0A, и, чтобы убрать условный переход в этом месте, введем предыдущие четыре байта FF 30 10 E2, которые соответствуют команде "ANDS R3, R0, 0xFF".

Таким изменением мы просто продублируем команду, идущую до условного перехода, причем это никак не повлияет на содержание регистров. Сохраняем изменения и выходим из редактора. Теперь можно снова запустить этот файл под отладчиком (нажать в нем <F5>) и посмотреть результаты. Возможна такая неприятность, при которой на КПК не закачивается измененный файл, поэтому сначала оттуда удаляется предыдущий exe'шник (отладчик записывает его в корневую папку на КПК).

Все бы хорошо, но теперь при нажатии кнопки BUY появляется надпись "Only available in registered version", что вряд ли позволит что-то купить. Попробуем отключить и это. Таким же способом, который уже был описан, доходим до функции, начинающейся с адреса 17834 и содержащей switch, похожий на найденный раньше. Однако если посмотреть, откуда вызывается эта функция, то IDA выдаст 17 таких мест, так что без отладчика не обойтись. Наша программа уже запущена под дебаггером, поэтому переходим (<Ctrl>+<G>) там по адресу 2C017834 (помни, что адрес может начинаться и с других байт, а не только с 2C0), ставим туда breakpoint и в игре жмем кнопку BUY. После срабатывания breakpoint'а в отладчике открывается окно Call Stack (соответствующая кнопка находится на той же панели инструментов, что и кнопка открытия окна регистров). Там можно видеть два адреса: первый - на котором мы сейчас стоим, второй - куда будет произведен возврат после выполнения данной функции. В данном случае это 2C015F28, поэтому открываем IDA, нажав кнопку <G>, вводим 15F28 и смотрим код выше этого адреса. Как теперь дойти до функции по адресу 15F24? Точнее, как сделать так, чтобы этот вызов никогда не происходил ;)? IDA говорит, что первым местом, откуда мог быть произведен переход к этому коду - это адрес 15EF4. Но выполнение этого кода в свою очередь зависит от того, выполняется ли условие по адресу 15EE4. Туда и поставим breakpoint в отладчике. И что произошло?

После установки breakpoint'а опять жмем <F5>, так как мы до сих пор стоим на прошлой точке останова, а в игре нажимаем на BUY. Тогда отправит нас по адресу 2C015EE4, где мы и хотели остановиться. Так же, как и в предыдущий раз, смотрим состояние флага Zero. Теперь он равен нулю, значит, условный переход по этому адресу не осуществляется, и таким образом выполнение вполне может дойти до вызова функции вывода строки о регистрации. Меняем флаг на единицу и наблюдаем за результатом. Надо же! Теперь вместо той строки показалась надпись "You can't

НЕКОТОРЫЕ КОМАНДЫ АССЕМБЛЕРА

B - Branch, безусловный переход, соответствует команде jmp
Bxx - условный переход; частный случай - BEQ, Branch if EQual (флаг Z=1)
BL - Branch with Link, вызов функции, соответствует команде call
LDRB - загрузка в первый операнд байта, находящийся по адресу, определяемому вторым операндом
CMP - compare, сравнение

Байты, отвечающие за первый условный переход

Часто бывает легче написать кейген, а не патчить программу.

Окно регистров

Код, отвечающий за второй условный переход

После внесения изменений

afford to buy this yet!", следовательно, нам просто не хватает денег и данное ограничение мы успешно обошли. Осталось только поменять в самом файле условный переход на безусловный, для чего байт по адресу 15EE7 меняем с DA на EA. И снова останавливаем отладку, изменяем файл, стираем старый файл с КПК, запускаем отладчик.

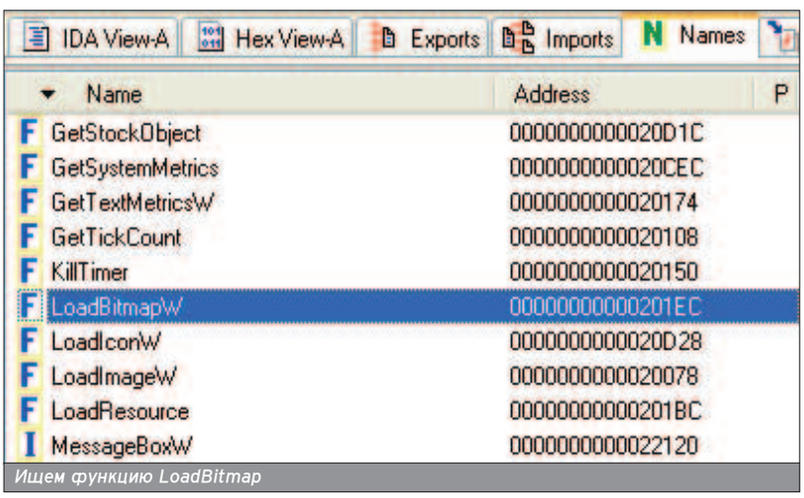
ЭТАП ВТОРОЙ. ПРАВИМ ГЛАВНОЕ МЕНЮ

Теперь нужно как-то убрать пункт главного меню Register. Для этого загружаем файл в Resource Hacker и в разделе Bitmap пробуем найти картинку с этой надписью. В данном случае номер этого ресурса равен 389 или 185 в шестнадцатеричной системе. Это изображение, скорее всего, загружается функцией LoadBitmap, поэтому ищем в IDA в окне Names Window эту функцию. Смотрим, по какому адресу она вызывается - в отладчике ставим на этот адрес breakpoint. Далее вызываем главное меню и ждем, когда сработает точка останова. У LoadBitmap вторым параметром является номер загружаемого ресурса. Тут нужно сказать, что в отличие от ПК, где параметры передаются через стек, КПК передает их через регистры начиная с R0. То есть нам нужно остановиться, когда регистр R1 будет равен 185, а до этих пор просто жмем <F5>. Итак, доходим до этого момента и смотрим Call Stack - функция была вызвана с адреса 12348, а это какая-то функция, которая, если посмотреть, вызывается из многих мест. Поэтому ставим точку останова на конец этой функции (на адрес 12410) и жмем <F5>. Далее один раз <F11>, чтобы выйти из функции, и оказываемся на 16FD8. Смотрим в IDA, что находится по этим адресам, и видим, что по адресу 16FC0 находится условный переход.

```
.text:00016FB8                  LDRB    R0, [R4,#0x2E8]
.text:00016FBC                  ANDS    R3, R0, #0xFF
.text:00016FC0                  BNE     loc_16FEC
.text:00016FC4                  MOV     R0, #0x184
.text:00016FC8                  MOV     R2, #0xEB
.text:00016FCC                  MOV     R1, #0x1D
.text:00016FD0                  ORR     R0, R0, #1
.text:00016FD4                  BL      sub_122F8
.text:00016FD8                  MOV     R0, #0x184
.text:00016FDC                  MOV     R2, #0xEB
.text:00016FE0                  MOV     R1, #0x1D
.text:00016FE4                  ORR     R0, R0, #1
.text:00016FE8                  BL      sub_1256C
.text:00016FEC
.text:00016FEC loc_16FEC
.text:00016FEC                  LDMFD   SP!, {R4,PC}
.text:00016FEC ; End of function sub_16F64
```

Код, отвечающий за вывод пункта меню Register

Если он не выполняется, то вызывается функция загрузки изображения, поэтому нужно всего лишь сделать этот переход безусловным, то есть поменять байт по адресу 16FC3 с 1A на EA. Делаем это, как и ранее в hex-редакторе, обновляем файл и смотрим результат. Отлично - пункта меню больше нет ;). В принципе, это все: можно играть, и ничто не напомнит о регистрации.



Ищем функцию LoadBitmap

НАПОСЛЕДОК

Часто бывает легче написать кейген, а не патчить программы, что, кстати, касается и нашего примера. Но, к сожалению, объем статьи ограничен и невозможно написать обо всем. Кейгены, без сомнения, предпочтительнее патчей, однако ради них необходимо более подробно изучить систему команд. У тебя будет чем заняться в свободное время :).

Content:

Крис Касперски aka мыщъх

ПИШИ БЕЗОПАСНО

ЗАЩИЩАЕМ КОД НА ЭТАПЕ ПРОГРАММИРОВАНИЯ

Если программист хочет выжить в нашем агрессивном мире, он должен писать свои программы так, чтобы их взлом вызывал у хакеров как можно больше отрицательных эмоций. Еще на этапе программирования программист должен задуматься о критичных для взлома местах и оформить их соответствующим образом. Глупо надеяться на мощь навесных защит - программисту нужно взять процесс обороны кода в свои руки и как следует наподдать взломщику с дизассемблером в руках.

Использование ненадежных приемов программирования и навесных протекторов вроде Extreme Protector или Armadillo иногда создает проблем гораздо больше, чем решает. Порой защищенная программа становится неуключей, тормозной, конфликтной и вообще всячески нестабильной. В ней появляются многочисленные критические ошибки, вылезают голубые экраны смерти, в результате чего программист получает весьма подмоченную репутацию. Ну и кому это надо? Никаких недокументированных возможностей! Никакой привязки к операционной системе! Никаких приемов нетрадиционного программирования! Защита должна быть простой и надежной, как индуистский слон! Минимум усилий, максимум эффективности!

СОВЕТ №1: ШИФРУЙСЯ!

Шифровка - простой, но весьма эффективный способ борьбы с дизассемблером. Естественно, она обязана быть динамической: крохотные порции кода/данных должны расшифровываться по мере необходимости, а после употребления зашифровываться обратно. Статические шифровальщики, расшифровывающие все тело программы за один раз, уже неактуальны. Достаточно снять дамп с работающей программы и - вуаля! Многие протекторы гробят таблицу импорта, корежат атрибуты секций, в общем, пакостят по всему мясокомбинату, в результате чего снятый дамп оказывается неработоспособен, но для дизассемблирования он подходит вполне, и такие меры защиты ни от чего не спасают!

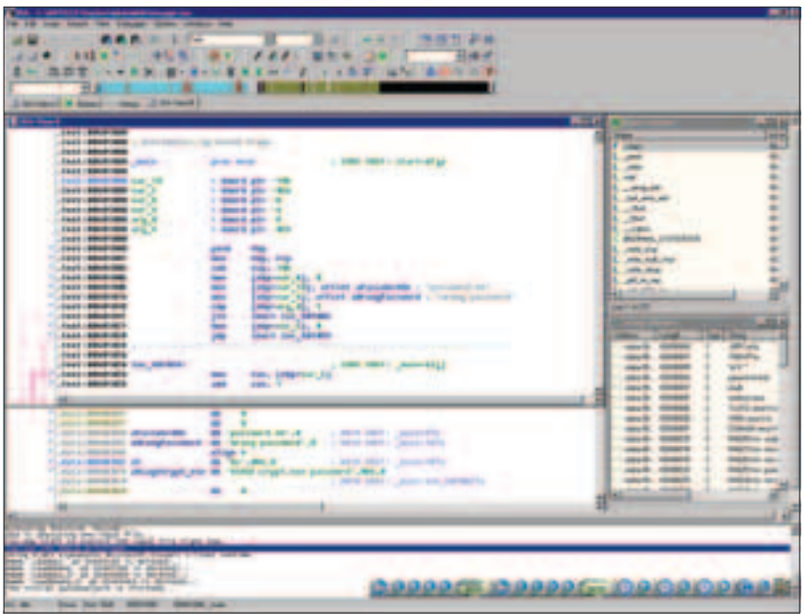
В реализации динамического шифровщика есть множество тонкостей. Если реализовать его в виде автономной процедуры типа crypt(void *p, int N), хакер сможет расшифровать любой желанный для него фрагмент простым вызовом crypt с соответствующими аргументами. Чтобы воспрепятствовать этому, различные части программы должны расшифровываться различными функциями. Некоторые "эксперты по безопасности" предлагают использовать асимметричную криптографию, полагая, что это убережет программу от модификации (то есть от взлома). Действительно, зашифровать модифицированную программу назад уже не получится - для этого нужно знать ключ, который хранится у разработчика и отсутствует в самой программе. Однако ничего не стоит дописать к концу распаковщика несколько машинных команд, ломающих программу на лету прямо в памяти.

Шифровать можно как код, так и данные, причем с кодом все намного сложнее. Во-первых, приходится предварительно манипулировать атрибутами страниц, выдавая разрешение на запись, а во-вторых, учитывать такую штуку, как перемещаемые элементы - специальные ячейки памяти, в которые в процессе загрузки файла операционная система прописывает фактические адреса. Впрочем, в большинстве случаев шифровки данных оказывается вполне достаточно. Главное - не дать хакеру обнаружить в дампе текстовые строки с ругательными сообщениями (типа "trial expired"), на которые легко поставить точку останова или подсмотреть перекрестную ссылку.

Давай посмотрим, как осуществляется шифровка текстовых строк на практике. Для начала возьмем простую программу, считывающую из командной строки пароль и выводящую в зависимости от ввода "password ok" или "wrong password". Один из вариантов ее реализации выглядит так:

```
// контрольная сумма пароля nezumi
#define _CRC_ 0x98

main(int c, char **v)
{
    int a;
    int CRC=0;
    char *goods="password ok";
    char *wrong="wrong password";

    if (c>1)
    {
        for (a=0; a<strlen(v[1]); a++)
            CRC=(CRC+v[1][a]) & 0xFF;
            // для отладки
            // printf("%x\n",CRC);

            // проверка CRC и вывод текстовых строк
            if (CRC-_CRC_)
                goods=wrong;printf("%s\n",goods);
            return 0;
        }
        printf("USAGE:crypt.exe password\n");
}
```

Для усиления защиты сравнивается не сам пароль, а его контрольная сумма (CRC).



Текстовые строки, хранящиеся открытым текстом, и перекрестные ссылки, ведущие к ним

Эталонный пароль нигде не хранится, и хакер при всем своем желании не может подсмотреть его. Оторвать мыщъх'у хвост, если это не так! Но как же мы узнаем CRC эталонного пароля? Да очень просто! Достаточно внедрить в отладочную версию программы строку printf("%x",CRC), распечатывающую контрольную сумму введенного пароля, и ввести эталонный пароль. Например, CRC слова "nezumi" равна 98h.

Откомпилировав полученную программу (естественно, предварительно убрав отладочную печать из финальной версии) и пропустив ее через дизассемблер, мы получим нечто вроде того, что на скриншоте.

Текстовые строки "password ok" и "wrong password" хранятся открытым текстом и легко обнаруживаются даже при беглом просмотре листинга (обычно для этого используются программы-фильтры, отсеивающие все читабельные текстовые последовательности). Что сделает хакер? Установив точку останова на начало "wrong password", он легко перехватит код, выводящий эту строку на экран, после чего ему останется найти тот условный переход, который его выводит. Весь взлом не займет и десяти минут!

Поэтому, как уже было сказано, текстовые строки необходимо хранить в зашифрованном виде, что можно сделать, например, так:

```
// контрольная сумма пароля nezumi
#define _CRC_ 0x98
// ключ шифрования
#define _KEY_ 0xFF

main(int c, char **v)
{
    int a;
    int CRC=0;
    char buf[1024];
    // зашифрованные текстовые строки

    char *goods = "\x8F\x9E\x8C\x8C\x88
\x90\x8D\x9B\xDF\x90\x94";
    // "password ok";
    char *wrong = "\x88\x8D\x90\x91\x98
\xDF\x8F\x9E\x8C\x8C\x88\x90\x8D\x9B";
    //wrong

    if (c>1)
    {
        for (a=0; a<strlen(v[1]); a++)
            CRC=(CRC+v[1][a]) & 0xFF;
        // проверка CRC и расшифровка строк

        if (CRC-_CRC_) // пароль ок
            for (a=0;a<strlen(wrong);a++)
                buf[a]=wrong[a]^_KEY_;
        else // пароль не ок
            for (a=0;a<strlen(goods);a++)
                buf[a]=~goods[a];

        // формирование завершающего
        // нуля и вывод строки на экран
        buf[a]=0; printf("%s\n",buf);
        return 0;
    }
    printf("USAGE:crypt.exe password\n");
}
```

И тут все обратили внимание на то, что строки расшифровываются не по месту хранения, а помещаются в специальный буфер, чтобы затруднить взлом. В противном случае хакер сможет установить точку останова на функцию printf (которая и выводит эту строку), что позволит ему определить смещение строки в секции данных и проанализировать перекрестные ссылки, ведущие к защитному коду.

Для шифровки не обязательно использовать крутые криптостойкие алгоритмы, такие как RC4 или DES. Сойдет и обычный XOR. Необходимо только убедиться, что ни один символ шифруемой строки не обращается в ноль: С трактует ноль как конец строки. Выражение x XOR y == 0 становится истинным тогда и только тогда, когда x == y, то есть ключ совпадает с одним из символов шифруемой строки. Значение FFh ни разу не встречается ни в одной из двух наших строк, поэтому оно вполне подойдет на роль ключа.

Единственная проблема - как зашифровать строки. В принципе, это можно сделать и после компиляции, воспользовавшись любым hex-редактором (например HIEW), однако при каждом ребилде эту процедуру придется повторять вновь и вновь, что очень достает.

Мы напишем для этой цели специальный отдельный шифратор, захватывающий исходную строку и выплевывающий зашифрованную последовательность, оформленную по всем правилам языка С, после чего нам останется только вставить ее в исходный код.

Его макет может выглядеть так:

```
main(int c, char **v)
{
    int a;

    printf("char *var_name=\"");
    for(a=0;a<strlen(v[1]);a++)
    { // шифровка по XOR
        printf("\\x%02X",v[1][a] ^ atol(v[2]));
        printf("\";");
    }
}
```

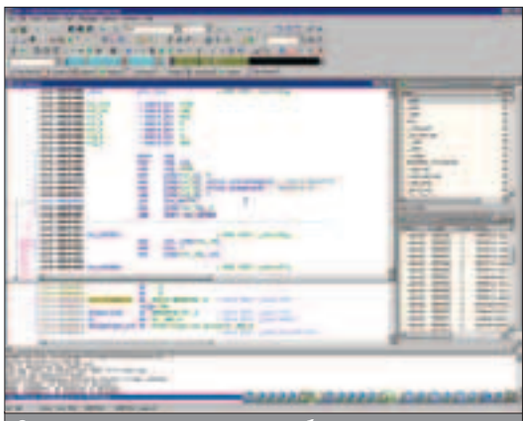
Если сделать все верно, текстовые строки "password ok/wrong password" исчезнут из откомпилированной программы. Теперь для взлома защиты хакеру придется потратить намного больше времени и усилий. В данном случае разница не так уж заметна, но в программах, состоящих из десятков тысяч строк, все будет пучком!

СОВЕТ №2: НЕ ДАВАЙ ПЕРЕМЕННЫМ ГОВОРЯЩИХ ИМЕН

■ Ни в коем случае не назначай защитным компонентам никаких осмысленных имен, особенно при программировании в Delphi и Builder - они попадут в исполняемый файл. Вроде бы очевидный совет (меня даже высмеяли за него пару раз), однако его очевидность не влияет на программистов, и они продолжают наступать на грабли. Взгляни, например, на результат декомпиляции программы Etlin HTTP Proxy на картинке.

Хакер с ходу видит юнит fRegister с процедурой bOkClick, обрабатывающей нажатие кнопки "OK", расположенной по адресу 48D2DCh. Все! Защитный механизм успешно локализован! Самая сложная часть взлома позади!

Просто поразительно, сколько информации можно извлечь, просматривая текстовые строки, открытым текстом лежащие в рядовой программе. Если бы программисты использовали для имен переменных какую-ни- »


Зашифрованные строки уже не бросаются в глаза

будь бессмысленную абракадабру, взлом занимал бы гораздо больше времени!

СОВЕТ №3: ИСПОЛЬЗУЙ ВИРТУАЛЬНЫЕ ФУНКЦИИ

■ Активное использование виртуальных функций существенно затрудняет дизассемблирование. Виртуальные функции вызываются по косвенным ссылкам, и над вычислением эффективных адресов приходится основательно потрудиться.

Вот пример:

```
class Base{
public:
    virtual void demo_1(void)
    { printf("BASE DEMO 2\n"); };
    virtual void demo_2(void) = 0;
};

class Derived:public Base{
public:
    virtual void demo_1(void)
    { printf("DERIVED\n"); };
    virtual void demo_2(void)
    { printf("DERIVED DEMO 2\n"); };
};

main() {

    Base *p = new Derived;
    p->demo();
}
```

А вот его дизассемблерный фрагмент:

```
.text:0040101B      mov eax, [esi]
.text:0040101D      mov ecx, esi
.text:0040101F      call dword ptr [eax]
; вызов виртуальной функции 1
.text:00401021      mov edx, [esi]
.text:00401023      mov ecx, esi
.text:00401025      call dword ptr [edx+4]
; вызов виртуальной функции 2
.text:00401028      push offset aNonvirtual;
.text:0040102D      call sub_4010B0
; вызов невиртуальной функции
```

Что мы видим? Вызов невиртуальной функции осуществляется по непосредственному значению (константе), равной в данном случае 4010B0h. А с виртуальными функциями все намного сложнее. Команда call dword prt [eax] передает управление по адресу, хранящемуся в двойном слове, на которое указывает регистр EAX, загружающийся, в свою очередь, из двойного слова, на которое указывает регистр ESI, а сам ESI... И такие цепочки могут быть длинными, очень длинными, причем исходный указатель инициализируется совсем в другом месте (как правило, в пра-пра-праматеринской функции). В общем, с виртуальными функциями современные дизассемблеры еще не справляются (к Delphi и Builder это не относится - для них существует множество отличных декомпиляторов).

Только необходимо помнить, что оптимизирующие компиляторы при первой же возможности заменят виртуальную функцию на статическую. Просто объявить функцию как виртуальную недостаточно - нужно ИСПОЛЬЗОВАТЬ ее как виртуальную.

СОВЕТ №4: ОСЛЕПЛЯЙ FLIRT

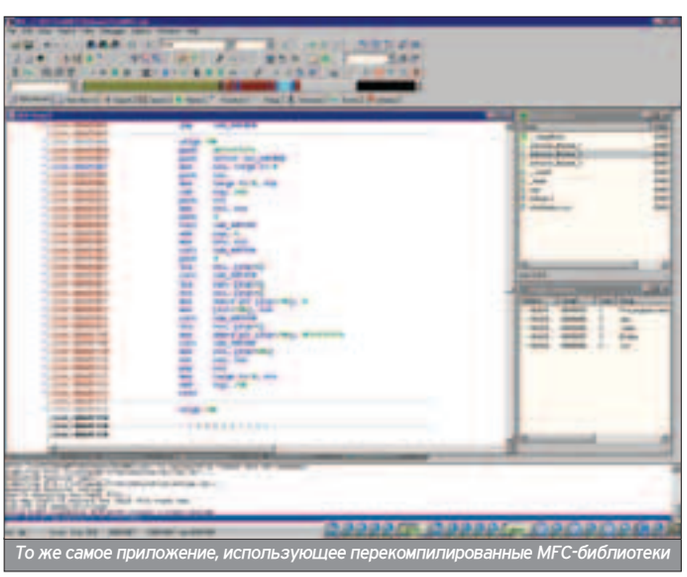
■ Типичная программа наполовину состоит из библиотечных функций, анализ которых занимает море времени и усилий (особенно если это какой-нибудь интерфейсный компонент). IDA PRO поддерживает шикарную технологию FLIRT (Fast Library Identification and Recognition Technology), автоматически распознающую функции большинства популярных библиотек, в результате чего задача хакера существенно упрощается.

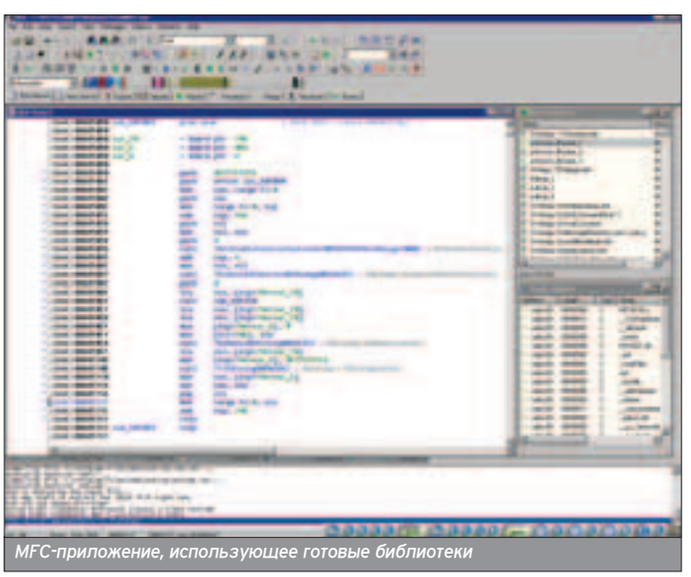
Вот фрагмент защитного механизма, считывающий имя пользователя и серийный номер из окна редактирования функцией TControl::GetText:

```
mov eax, [ebx+328h]
call @TControl@GetText$qqrv
; TControl::GetText(void)
...
mov eax, [ebx+320h]
call @TControl@GetText$qqrv
; TControl::GetText(void)
```

Размер защитного кода несопоставим с размером библиотечных функций, которые он использует (библиотечные функции на порядок "жирнее"). Если бы не FLIRT, взлом затянулся бы надолго. А так пришел, увидел, отломил. Тем более что большинство хакеров предпочитает ставить точки останова не на GetWindowTextA, а на @TControl@GetText$qqrv, что намного удобнее. Как этому помешать? Оказывается, чтобы ослепить ИДУ, достаточно изменить всего нес-

Типичная программа наполовину состоит из библиотечных функций, анализ которых занимает море времени и усилий.


То же самое приложение, использующее перекомпилированные MFC-библиотеки


MFC-приложение, использующее готовые библиотеки

колько байтов в начале каждой библиотечной функции.

Например, @TControl@GetText$qqrv:

```
push ebx
push esi
push edi
...
pop edi
pop esi
pop ebx
retn
```

Если заменить push ebx/push esi/pop esi/pop ebx на push esi/push ebx//pop ebx/pop esi, IDA не сможет узнать эту функцию и хакеру придется основательно попыхтеть над реконструкцией защитного механизма.

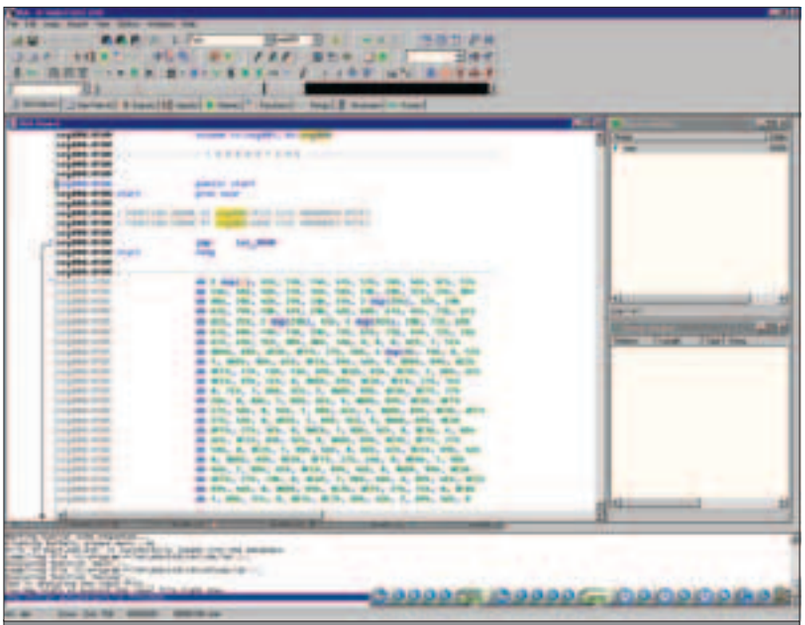
```
CODE:0048D2F7    mov eax, [ebx+328h]
CODE:0048D2FD    call sub_4410B8
...
CODE:0048D309    mov eax, [ebx+320h]
CODE:0048D30F    call sub_4410B8
```

Конечно, править все функции вручную - медленный и крайне неперспективный путь, однако знатоки ассемблера за несколько вечеров напишут автоматический пермутатор или возьмут уже готовый (благо полиморфных движков в Сети предостаточно). Сделать замену достаточно всего один раз - в библиотеке, а потом просто линковать ее к защищенным программам! Кстати говоря, исходные тексты большинства стандартных библиотек открыты, а это значит, что, перекомпилировав их с другими ключами оптимизации (или другим компилятором), мы точно ослепим FLIRT.

СОВЕТ №5: ПРОГРАММИРУЙ НА VISUAL BASIC ИЛИ FORTH

Как это ни смешно, но программы, написанные на Visual Basic, ломаются значительно труднее (особенно если использовать трансляцию в p-kod). Еще сложнее ломается Forth. Кто сталкивался - тот поймет. Кто не сталкивался – тот еще не в психушке. Forth – это вообще-то интерпретатор, но довольно своеобразный, совсем не похожий на остальные языки. Чисто технически можно написать Forth-декомпилятор или найти уже готовый, но для этого хакеру потребуется изучить и сам Forth, а это значит, что взлом программы затянется надолго. Разумеется, никто не предлагает писать приложение на Forth'е целиком. Достаточно запрограммировать на нем несколько ключевых защитных процедур (генерация серийного номера, расшифровщик и т.д.)

Еще можно использовать Microsoft Visual Studio .NET - она тоже позволяет генерировать p-код, правда, для него уже появилось множество декомпиляторов, да и сам формат p-кода стандартизован и тщательно специфицирован. Лучше откопать какой-нибудь редкоземельный интерпретатор, например Haskel или LISP. Во-первых, знакомство с новыми языками расширяет кругозор, а во-вторых, многие начинающие хакеры своими познаниями в языках программирования обычно не выпезают за пределы C/Pascal/ASM - тот же LISP с ходу они вряд ли взломают. Скорее всего, исследуемая программа будет заброшена на полку до лучших времен (то есть навсегда).



Дизассемблерный листинг программы на языке Forth

```
: IS 0 DO I . LOOP ;
: AS 0 DO CR 1 - DUP IS LOOP ;
```

СОВЕТ №6: ИСПОЛЬЗУЙ ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

Существует такой миф, что глобальные переменные – это плохо. Рассказывается множество ужасных историй, к примеру, как пара придурков гоняла бага целый день (месяц, неделю), и все из-за глобальных переменных! Это верно! Обращение к ним может происходить из разных мест, и кто угодно может затереть чужое значение со всеми вытекающими отсюда последствиями. Локальные переменные в этом отношении намного нагляднее и проще. Их любят все - и программисты, и хакеры.

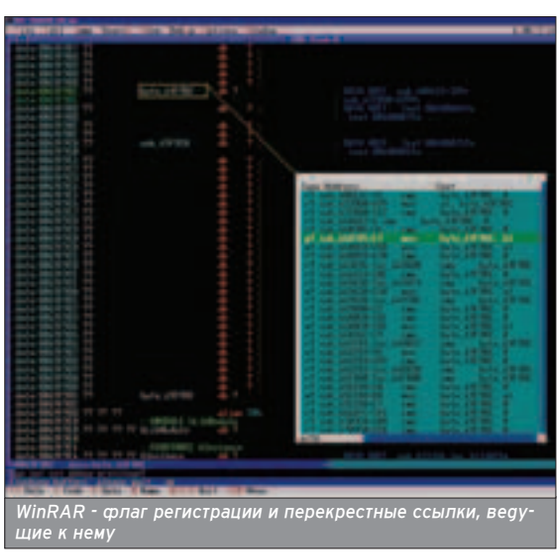
Флаг регистрации, расположенный в глобальной переменной, ломается на ура, поскольку к нему ведут перекрестные ссылки, сразу показывающие, какой код обращается к нему и когда. Однако если использовать одну и ту же ячейку памяти для хранения нескольких типов данных сразу, можно сильно усложнить жизнь взломщику.

Допустим, на стадии инициализации некоторая ячейка используется как флаг регистрации, затем флаг регистрации временно копируется в другое место, а сюда помещается флаг, например, ошибок. Один раз за определенный промежуток времени процедура обмена повторяется вновь. Конечно, это упрощенная схема, но общий смысл, думаю, понятен.

Хакер будет видеть множество перекрестных ссылок, ведущих в разные части программы. Проанализировав несколько из них, он быстро найдет флаг ошибок, переименует переменную в f_error и тут же потеряет к ней всякий интерес. А даже если не потеряет, ему будет очень трудно разобраться, в какой момент эта переменная является флагом ошибок, а в какой - флагом регистрации.

ЗАКЛЮЧЕНИЕ

Рассмотренные приемы, несмотря на свою простоту, служат отличным оружием против хакеров. Конечно, взломать можно все, но так или иначе на это потребуется какое-то время. Именно его всегда недостает хакерам. В первую очередь ломают простые программы. Сложные оставляют на потом. Надеюсь, эта статья поможет тебе сделать так, чтобы твоя программа попала в список сложных.



WinRAR - флаг регистрации и перекрестные ссылки, ведущие к нему

Bagie (bagie@bk.ru)

ЖИЗНЬ ПОСЛЕ КОМПИЛЯЦИИ

ВСЕ О ПРОТЕКТОРАХ И УПАКОВЩИКАХ

После компиляции нужно защищать свои программы всеми возможными способами. Один из самых популярных способов – использование протекторов. Соответственно, возникает вопрос: а какой протектор использовать, и стоит ли вообще делать это? Об этом и многом другом поговорим в этой статье.

НЕМНОГО ТЕОРИИ

В наше передовое время существует множество различных методов защиты приложений от посягательства со стороны других лиц. Давным-давно, когда о навесных методах защиты и не подозревали, жизнь программиста была намного сложнее, так как он должен был заботиться обо всем самостоятельно. С приходом первых упаковщиков исполняемых файлов все кардинально изменилось. Только появившись на свет, пакеры стали выполнять помимо своей основной функции, сокращения размера исполняемых модулей, еще и функцию защиты, так как после упаковки нельзя непосредственно редактировать файлы: для этого нужно распаковать файл, отредактировать его должным образом, а затем, по надобности, еще и упаковать. Конечно, сейчас мало кого удивишь таким "очень оригинальным методом защиты", поэтому становится популярной другая разновидность упаковщиков - протекторы. Протектор - это программа, которая помимо функции упаковки (хотя в некоторых протекторах упаковки нет) еще и модифицирует исполняемые файлы таким образом, что распаковать их после этого становится намного сложнее, чем при использовании любого другого пакера.

Те, кто не знает, что такое упаковщик исполняемых файлов, должно быть, думают, что все это бред. Мол, возьми архиватор, упакуй программу, а затем, когда понадобится, распакуй и пользуйся. На самом деле пакеры работают немного не так: во-первых, они не распаковывают программу на диск и затем не запускают ее. Декомпрессия происходит непосредственно в памяти программы, так что дополнительных дисковых ресурсов не потребуется. Во-вторых, алгоритмы сжатия, применяемые в пакерах, не столь требовательны к ресурсам системы, как алгоритмы сжатия, используемые в современных архиваторах. И, в-третьих, нам нужно защитить программу от посягательств с недоброжелательной к нам стороны. Как же защититься, если программа будет упакована WinRAR'ом, например? Можно, конечно, установить пароль на архив, но тогда что это такое получится :).

В реале простой упаковщик не обеспечивает должную защиту программам, так как распаковать такие упаковщики, которые не используют никаких защитных приемов, дело нескольких секунд. Для этих целей и были придуманы протекторы - они и упакуют, и закриптуют код программы каким-нибудь особым способом. Испортят таблицу импортов в файле. Будут использовать защиту против отладки. Будут проверять контрольную сумму, а некоторые - даже хэш-сумму запакованного файла, дабы исключить возможность подредактирования программы. Будут дико приставать с окошками, содержащими информацию о том, что программа не зарегистрирована или что срок действия демо-версии истек. И еще много чего будут. И действительно, набор функций для различных протекторов разный. Отличаются и уровни защиты, которые обеспечивают протекторы. И снова возникает вопрос: а какой протектор использовать, и стоит ли вообще делать это?

На форуме reversing.net ты найдешь много полезного

Cracklab.ru – настоящий рай ;-)

Свои программы по мере возможности и необходимости следует защищать всеми возможными способами. Внешний уровень защиты – это пограничный слой, который всегда первым принимает на себя вражеские атаки. Естественно, то, чем была упакована или закриптована программа, является своего рода "лицом", которое предстает перед взглядом взломщика в начале его пути. В данном типе защиты особенно важен психологический аспект, так как, например, неопытный взломщик, увидев действительно реальную и известную защиту, просто откажется от затеи украсть чужую интеллектуальную собственность. В подтверждение этому можно отметить, что многие протекторы пытаются, иногда успешно, маскироваться либо под простейшие упаковщики, к примеру UPX, либо под какое-нибудь малоизвестные, либо под очень кру-

тые защиты. В любом случае, неправильно определив тип протектора, взломать программу с полпинка, скорее всего, не удастся.

Также можно отметить, что по возможности необходимо обеспечивать связь всех уровней защиты, чтобы своевременно реагировать на любые изменения. Например, у некоторых протекторов есть одна очень полезная функция: если они обнаруживают попытку взлома (к примеру происходит отладка приложения), то, не предпринимая никаких видимых действий, протектор передает информацию на нижние уровни программы, где возможна более гибкая обработка возникшего события. Пусть, к примеру, нас отлаживают, пытаясь перехватить кусок кода, где происходит проверка правильности серийного номера. Протектор определяет это событие, затем сообщает приложению о том, что идет отладка. А ты в свою очередь можешь в этом месте написать процедуру, которая затирала бы целиком весь код, в котором происходит проверка, не подавая никаких демаскирующих признаков жизни. Получаем следующее: если неизвестно, как работает защита, отловить место, в котором решается судьба пользователя - довольно сложное дело. Некоторые современные протекторы поддерживают эту столь полезную функцию.

Чтобы создать действительно хорошую защиту, также необходимо уделять внимание внутреннему устройству программы. Особо надеяться на протектор нельзя, пусть даже он самый дорогой и известный из всех. Злоумышленник не пожалеет сил, чтобы обойти или подавить навесную защиту - и спасение от таких неприятных ситуаций тоже нужно обеспечить.

Вдогонку напишу, что многие крипторы, особенно платные, позволяют организовывать внешнюю регистрацию. Другими словами, это обстоятельство освобождает тебя от обязанности писать защиту: вся проверка и регистрация может осуществляться протектором. Но никогда не используй эту возможность, потому что стоит просто снять защиту - и приложение останется абсолютно "голым" и просто никаким образом не будет просить зарегистрироваться.

ПРОТЕКТОРЫ НА ПРАКТИКЕ

Мы подбираемся к самому интересному - к некоторым известным и не очень протекторам исполняемых файлов, которые подходят для защиты приложений.

ASPROTECT

Сайт: www.aspack.com

На сегодняшний день один из самых лучших протекторов для защиты исполняемых файлов. Эффективными и простыми методами обеспечивает защиту программы от взлома и нелегального распространения. Помимо

ASProtect обладает очень удобным интерфейсом

обычных приложений, с помощью ASProtect можно защитить и различные компоненты, например библиотеки ActiveX.

На момент написания статьи последней версией был ASProtect 2.1 - очень хороший и удобный в использовании протектор, сочетающий множество различных типов защит в различных комбинациях. При желании можно осуществить и внешнюю схему регистрации. В качестве стандартных защит ASProtect дает следующее:

- хорошая антиотладочная система затрудняет запуск программ, защищенных ASProtect под отладчиком.
- защита исполняемого файла контрольной суммой не позволит взломщику редактировать код программы.
- защита таблицы импортов не допустит, чтобы злоумышленник восстановил весь импорт до исходного состояния.
- сжатие файла перед зашифровкой не только уменьшит размер исполняемого файла, но и значительно усложнит модификацию программы.

Кроме того, используя ASProtect, можно осуществлять дополнительные виды защит, сочетая их в различных комбинациях. Например, можно заставить программу запускаться только после ввода пароля или использовать ключи активации. Интересна возможность отправить некоторые ключи в "черный список", если они получили широкое незаконное распространение. Также при использовании ключей активации возможна их привязка к конкретному оборудованию, установленному в системе. При генерации ключей используются сложные криптостойкие алгоритмы, поэтому просто "угадать" ключ абсолютно невозможно. Имеется также старая возможность прекращения работы программы через указанное число запусков или по истечении указанного строка. А при желании можно просто надоедать пользователям сообщениями с просьбами зарегистрироваться.

Замечу, что старые версии протектора (до версии 2.0) не обеспечивают защиту на достойном уровне, так как в настоящее время их могут снять даже не очень опытные взломщики, благодаря утилитам автоматической распаковки. Если нужна реальная защита, эти версии не для тебя.

Самый большой недостаток ASProtect, впрочем, как и других коммерческих протекторов - это высокая цена, которая может достигать $299. Поэтому, если нет возможности приобрести эту программу, то, скорее всего, ты отправишься на поиски других протекторов.

OBSIDIUM

Сайт: www.obsidium.de

Коммерческий протектор, который позволяет защитить программы от взлома, а также может с легкостью осуществить простую и надежную систему лицензирования. Многое, что есть в протекторе, встречается и в других крипторах, но авторы Obsidium предлагают довольно интересные решения. Чего стоят одни антиотладочные приемы? А оригинальный runtime patching, хитро сделанный импорт, умная кража байтов с OEP (Original Entry Point)? Шифровка данных блоками, зашифровка ресурсов и множество SEH-фреймов умножают достоинства этого протектора в плане защиты. Его ключевые особенности следующие:

- шифрование и сжатие кода и данных программы;
- защита от дизассемблирования, отладки, снятия дампа, а также защита API-функций и изменения кода программы в процессе работы;
- автоматизированная система лицензирования, поддерживающая размер открытого ключа до 2048 бит;
- возможность регистрации на основе ключевых файлов или текстовых ключей;
- поддержка черного списка для ключевых файлов;
- возможность защиты паролем, а также создание trial-версий программы.

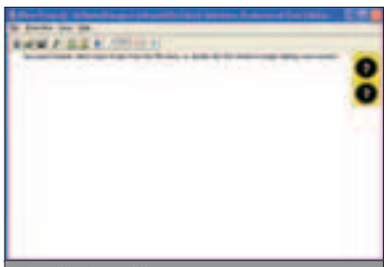
Окно Obsidium немного напоминает окошко какого-нибудь "мастера"

В общем, этот криптор для тех, кто действительно ценит свою интеллектуальную собственность и хочет надежно защитить свои программы от взломщиков.

ARMADILLO

Сайт: www.siliconrealms.com/armadillo.shtml

Продолжает коммерческую линейку протекторов класса middle end. Очень »



Дизайн Armadillo оставляет желает лучшего

хороший криптор, который в состоянии выполнять все стандартные функции: сжатие и шифрование кода и данных программы, использование антиотладочных и антидамповых приемов защиты и др. Но самой главной отличительной особенностью программы является принцип работы, который несколько отличается от принципов работы других протекторов. Вся суть в том, что Armadillo активно использует отладочные средства Win32 (Debug API). То, что мы запускаем, по сути, является отладчиком, который в процессе отладки улавливает и исправляет специально созданные им же ошибки в отлаживаемой программе, которую он запускает сам. В дополнение к этому протектор также динамически шифрует и дешифрует код программы. Данная технология получила название CopyMem II, суть которой заключается в том, что в памяти в один момент времени присутствует лишь несколько страниц кода программы, нужных для ее работы, а остальные страницы помечаются как отсутствующие в памяти. Если обратиться к несуществующей странице, то возникает исключение, которое обрабатывается отладчиком, и он в свою очередь производит расшифровку куска кода. После того как код становится ненужным для работы, Armadillo заново шифрует и помечает текущую страницу в памяти программы как неиспользуемую.

Благодаря такой системе усложняется снятие защиты, и это под силу не каждому взломщику. Следовательно, этот протектор является одним из самых лучших решений для защиты программ.

К недостаткам протектора можно отнести то, что при его использовании резко падает производительность, и из этого делаем вывод, что не следует использовать Armadillo для защиты программ, занимающихся, к примеру, кодированием аудио или видео, игр или других программ, интенсивно использующих оперативную память.

Цена профессиональной версии криптора может быть больше, чем на ASProtect, так что лучше хорошо подумать, прежде чем воспользоваться данной защитой.

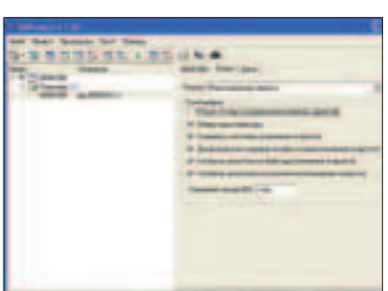
VMPROTECT

Сайт: www.polytech.ural.ru

Особенный протектор, не похожий ни на один другой. Эта программа, в отличие от других, не работает по предписанной схеме - не шифрует и/или пакует файл, а в процессе работы производит обратные действия. Все эти ухищрения в большинстве случаев устаиваются впустую: всегда можно перехватить то место, где происходит завершение работы криптора и управление передается программе-носителю. Вместо этого VMProtect работает как виртуальная машина, то есть преобразует участки кода, указанные разработчиком, в собственный формат команд. И в процессе выполнения этого кода не производится расшифровка, а непосредственно работает как интерпретатор процессорных команд. Поэтому для снятия этого протектора необходимо тщательно изучить структуру команд VMProtect, что является довольно трудоемкой и сложной задачей. Единственный недостаток, которой присутствует во всех интерпретаторах - это низкая скорость работы. Но если важна не скорость, а достойный уровень защиты, то этот протектор пригодится.

Из-за небольшой тонкости в использовании программы придется указывать адреса процедур, которые нужно зашифровать. Эти адреса можно посмотреть отправив программу в дизассемблер. Процедур для зашифровки может быть несколько, и рекомендуется защищать в первую очередь важные участки кода: точка входа в программу (здесь можно усложнить вероятное определение OEP, а также скрыть местоположение таблицы импортов), куски кода, где происходит, например, проверка серийного номера, код, содержащий ключевые строки и т.д.

Так как криптор работает по совершенно другому принципу, после этого было бы неплохо упаковать файлы, обработанные VMProtect, каким-либо упаковщиком, например можно взять UPX, ASPack, Packman, PCShrinker или любой другой, с которым программа будет работать нормально. Конечно, можно попробовать повесить поверх какой-нибудь протектор, но в этом случае, скорее всего, криптор необходимо будет подбирать для каждого конкретного случая.



Поначалу VMProtect слегка озадачивает, зато потом радуешься, что все очень удобно

Разработчики VMProtect денег за свое детище не берут

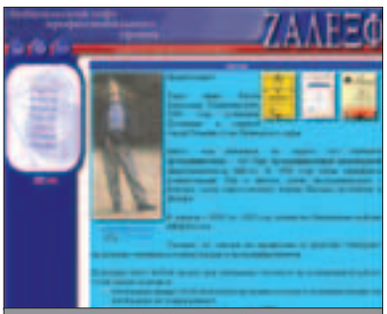
ORIEN

Сайт: www.zalexf.narod.ru

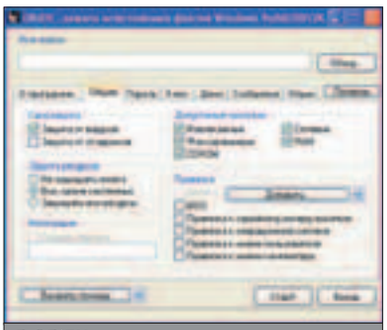
Еще одна альтернатива для защиты исполняемых файлов от взлома и ограничения допуска. По умолчанию защита работает всего лишь как обычный пакер, но у программы существует множество опций, благодаря которым можно настраивать уровень и тип защиты, а также и некоторые другие дополнительные и очень полезные функции. В качестве основных возможностей ORiEN предлагает следующее:

- 4096-битное шифрование (стандарт DES);
- упаковка защищаемых данных (используется aPlib);
- полиморфное тело защиты (движок Polystack);
- защита от модификации кода;
- защита от отладки;
- защита по паролю (до четырех паролей);
- защита ключевым файлом;
- использование различных привязок (HASP, BIOS и т.д.);
- возможность использовать демонстрационные ограничения;
- интеграция с защищаемой программой;
- полная совместимость со всей линейкой операционных систем Windows и поддержка форматов EXE, DLL, SCR, OCX и т.д., а также компиляторов C/C++, Delphi, VB, Visual FoxPro, Sybase PowerBuilder и т.д.

Важно, что защита может интегрироваться с самой программой. Суть данного метода заключается в том, что протектор при загрузке создает мьютекс – специальный объект, используемый для синхронизации между процессами. В защищаемой программе мы должны проверить существование этого объекта. Если в этом случае распаковать программу, то мьютекс, естественно, не будет создан, что и послужит поводом для дальнейших действий. Хотя данный метод не нов, а также присутствует во многих других протекторах, им ни в коем случае не следует пренебрегать – поможет в борьбе хотя бы с не очень опытными взломщиками. Также необходимо помнить, что если не используется ни защита ключом, паролем, привязкой или ограничением на число запусков, то данный протектор работает как самый обычный пакер и весь смысл защиты исчезает.



На ресурсе zalexf'a ты найдешь неплохую подборку специализированного софта



В ORiEN защита может интегрироваться с самой программой

Пожалуй, самый интересный момент в крипторе – его цена, которую устанавливает автор, то есть вполне возможно найти полностью функциональную версию программы за те деньги, которые считаешь приемлемыми.

KRYPTON

Сайт: www.lockless.com

Довольно перспективный криптор для исполняемых файлов. И... за его использование не нужно платить! Протектор не сжимает данные, перед тем как зашифровать их, следовательно, приходится заботиться об этом самостоятельно, хотя сразу скажу, что я не смог найти пакер, с которым Krypton заработал бы нормально. К тому же Krypton – довольно редкий протектор, и в первую очередь из-за того, что он весьма капризный, так что, подсовывая ему файлы на шифровку, будь готов к тому, что они вдруг просто-напросто упадут при запуске. Хотя это справедливо только для максимального уровня защиты (всего их три), а на минимальном уровне все должно работать отлично. Антиотладка также выглядит ужасно, однако, несмотря на это, криптор хорошо обрабатывает импорты в файле, после чего восстановить их бывает даже порой довольно сложно, что, несомненно, является его жирным и главным преимуществом.

Сама по себе структура протектора выглядит довольно сыро, но в совокупности с другими защитами может дать очень неплохой результат. Например, стоит попытаться укрыть OEP, чтобы усложнить распаковку. Обмануть OEP-детекторов проще

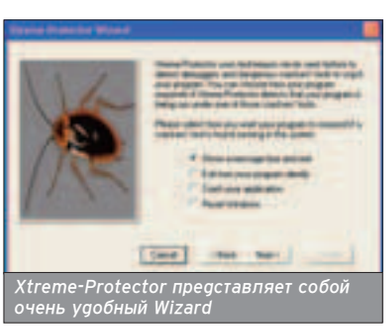
Krypton очень хорошо обрабатывает импорты в файле

всего кражей нескольких первых байт из точки входа. Для этого просто нужно написать небольшую процедуру, которая выполнялась бы при загрузке программы и затирала примерно 10 байт на OEP. Напоследок скажу, что испробовать данную программу на деле никто не запрещает, так что пробуй быстрее.

XTREME-PROTECTOR

Сайт: www.oreans.com/xprotector

Относится к классу high end, то есть его можно назвать почти самым сложным и трудноломаемым протектором исполняемых файлов. Это лишний раз подтверждается тем, что после выхода в свет криптор продержался почти целый год, что уже говорит об уровне защиты, которую он обеспечивает. Протектор отличается не только прекрасной антиотладочной системой и умелым сокрытием импорта и кода защищаемой программы, но и тем, что Xtreme-Protector частично использует код на привилегированном уровне Ring0 в виде драйвера. Все это, несомненно, очень сильно осложняет исследование и взлом защиты, а для начинающих и даже довольно опытных взломщиков и вовсе не оставляет шансов. Однако, несмотря на все преимущества, протектор не очень стабилен в работе, и вполне может случиться так, что после защиты программы тебе придется встретить резкий и очень неприятный ребут (достаточно посмотреть на название – и все станет ясно), и если действительно происходит так, то лучше отказаться от его использования. Ну и чтобы совсем испортить картину, по



Xtreme-Protector представляет собой очень удобный Wizard

секрету скажу, что авторы данной программы ненавязчиво спрашивают денежное вознаграждение.

Вот мы и рассмотрели некоторые протекторы. В действительности их намного больше, и даже больше, чем может вообразить себе психически здоровый человек, но большинство из них либо не заслуживают внимания, либо не доступны для публичного обозрения. Здесь были приведены наиболее интересные, на мой взгляд, крипторы. Однако это вовсе не означает, что нужно ограничиться этим набором. Для своих разработок можно использовать и другие навесные защиты, некоторые из которых можно комбинировать в различных сочетаниях.

ТЕСТИРОВАНИЕ ЗАЩИТЫ

Очень важно провести некоторое тестирование после установки навесной защиты. Суть тестирования заключается, прежде всего, в проверке полной работоспособности программы. После этой стадии также необходимо в некоторой мере протестировать защиту на соответствие уровню. Иначе говоря, стоит немного времени пожить в шкуре злоумышленника и попытаться всеми доступными средствами снять протектор. Как минимум, защита хотя бы должна быть стойкой к различным автоматическим распаковщикам. К универсальным распаковщикам пакеров и некоторых протекторов класса low end можно отнести GenericUnpacker и Quick Unpack. Следовательно, стоит попытаться снять защиту с помощью данных утилит, но с действительно хорошими протекторами типа Armadillo, ASProtect, Obsidium и gp. можно и не пытаться сделать это, так как все равно ничего хорошего не получится.

Многие протекторы используют защиту таблицы импорта в файле, так что можно использовать программу Import REConstructor для ее восстановления. Несмотря на то, что взломщик может и не знать реальной точки входа защищенной программы, в ImpREC'e указывай свой реальный OEP (EP программы, не упакованной ничем), и даже если таблицу импорта восстановить не удалось, испробуй трейсер для ее нахождения. Теперь, если после всех этих операций импорт оказался восстановленным до первоначального состояния, можно судить о том, насколько крута выбранная защита.

Неплохо также проверить криптор в режиме отладки, дабы убедиться в работоспособности его антиотладочных премудростей, а те, кто знаком с принципами взлома приложений, могут пойти дальше и попробовать распаковать программу вручную.

Многие виды навесных защит, анонсируемые сегодня на рынке, попросту не могут представить ничего особенного. Их документация гласит, что »

якобы именно ЭТА защита самая лучшая и никоим образом взломать ее нельзя. Не доверяй первой попавшейся документации, а ответ на вопрос о том, какие защиты действительно стоит попробовать, лучше поискать на различных крэкерских сайтах, так как те, кто ломают, в тысячу раз лучше знают, насколько хорош тот или иной протектор и для каких целей он сгодится.

УПАКОВКА ИМЕЕТ ЗНАЧЕНИЕ

К вышесказанному добавлю, что многие бесплатные протекторы и упаковщики созданы как раз руками крэкеров, вирусописателей и еще бог знает кого. Но это связано вовсе не с тем, что названные лица пытаются осложнить себе жизнь - скорее, наоборот. В современном мире уберечься от вирусов намного проще, чем самим вирусам от различных антивирусных мониторов. Другими словами, стоит новенькому почтовому червю появиться в свет, как его уже поджидают с широко распростертыми объятиями антивирусники. Да и пользователь пошел подозрительный, и прежде чем запустить только что скачанную из интернета программу, тут же отправляет ее на проверку. Но выход есть: стоит упаковать программу чем-либо, и вот оно - чудо техники. Антивирусы не видят исходного кода и часто падают на этом факте, а точнее падали, так как почти все современные защиты от вирусов умеют также и распаковывать многие пакеры и протекторы, которые повешены сверху программы. Несмотря на это, объять все навесные защиты просто невозможно, следовательно, нужно знать, чем "правильно" упаковать хороший троян ;-).

UPX

Самый популярный пакер всех времен и народов, мимо которого также не прошли стороной шаловливые ручки вирусописателей. Множество современных вирусов для уменьшения их размера отправляется на компрессию именно к нему. А почему к нему? Да потому, что UPX зарекомендовал себя, прежде всего, как самый стабильный, поддерживающий множество форматов, а также обладающий хорошей степенью сжатия упаковщик исполняемых файлов. Скачать UPX можно по адресу http://upx.sourceforge.net.

UPX пользуется огромной популярностью среди многих вирусописателей

FSG

Пакер, очень простой в использовании. Сжимает исполняемые файлы чуть хуже, чем UPX, однако у начинающих взломщиков частенько возникают проблемы с распаковкой FGS вручную. Особенно это осложняется тем, что пакер неплохо скрывает таблицу импорта в запакованном файле, поэтому Generic-распаковщики с ним не справляются. FSG также очень часто используют для упаковки различных вирусов. Качай FSG отсюда: www.cracksearch.ru/soft/soft7.htm.

MEW

Пакер класса "очень хорошая штука". Он относится именно к этому классу, потому что упаковывает даже лучше, чем UPX. Пусть и не намного, но все равно приятно. Такое сжатие достигается благодаря кодеку LZMA. С распаковкой такого пакера проблем возникнуть не должно, но у начинающих крэкеров бывают обидные обломы, когда они встречаются один на один с MEW. Приметно, что частенько трояны упаковывают именно MEW, для того чтобы довольно сильно уменьшить размер исполняемого файла. MEW можно взять с CrackLab: www.cracklab.ru/download/list.php?l=8.

Трояны очень часто упаковываются именно с помощью MEW

(WIN)UPACK

Король плотной упаковки исполняемых файлов. Награжден этим статусом, потому что сжимает даже лучше, чем UPX или MEW. К сожалению, в пакере пока имеется поддержка только одного формата PE(Win32) в виде EXE-файлов. В общем, это, наверное, самый перспективный упаковщик для "нехороших программ". Программу можешь найти по адресу http://dwing.go.nease.net.

MORPHINE

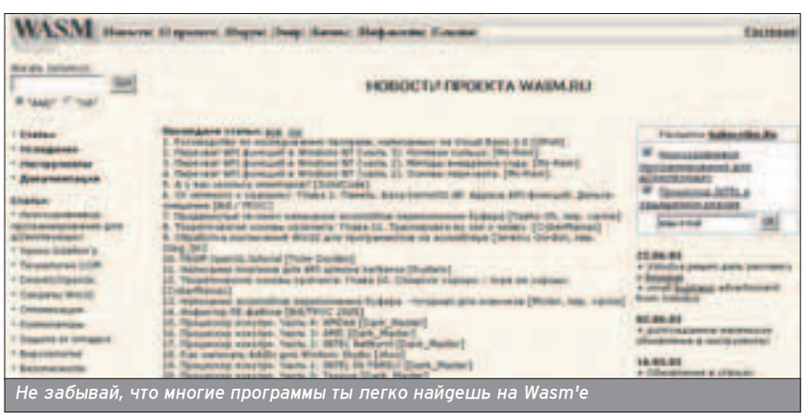
Ан нет, это вовсе и не пакер. Название программы говорит само за себя. Morphine - это полиморфный протектор. С точки зрения защиты он не представляет собой чего-то особенного, и распаковать это чудо можно секунд за 30. К тому же для протектора есть автоматический распаковщик, но пользоваться им для снятия защиты с вирусов вряд ли кто захочет, так как он запускает жертву на выполнение и затем снимает дамп. Не очень-то приятно, когда на твоей машине хозяйничает всякая "зараза". Очень хорошо повесить Morphine поверх любого упаковщика, например из тех, что приведены выше.

И напоследок скажу, что большинство ухищрений, связанных с упаковкой вирусов, могут оказаться бесполезными. В этом случае, возможно, поможет один метод: запакованную, например, с помощью UPX программу модифицируют таким образом, что при запуске получает управление код, который расшифровывал бы байты на EP упакованного вируса простым XOR, а затем прыгал на этот EP. Естественно, сами байты должны быть до этого зашифрованы. Некоторые антивирусы после этого и вовсе не видят, что программа чем-либо упакована, и никаких вирусов в ней, скорее всего, не обнаружат. Качай Morphine с http://wasm.ru/tools/12/Morphine.zip.

ВМЕСТО ЗАКЛЮЧЕНИЯ

Мы рассмотрели некоторые принципы защиты приложений после компиляции. Были изучены основные концепции упаковщиков и протекторов, а также приведены некоторые примеры самых достойных с краткой, но максимально информативной характеристикой. Некоторые принципы "легкого" тестирования защит тоже не были оставлены без внимания, как и использование навесной защиты в вирусах.

Звучит пессимистично, но если захотят - тебя все равно сломают, что бы ты ни придумал и ни реализовал в виде навесной защиты.



Не забывай, что многие программы ты легко найдешь на Wasm'e

ТОВАРЫ В СТИЛЕ

ПРИСОЕДИНЯЙСЯ!

ЭКСКЛЮЗИВНАЯ КОЛЛЕКЦИЯ ОДЕЖДЫ И АКСЕССУАРОВ ОТ ЖУРНАЛОВ ХАКЕР И ХУЛИГАН

* Футболки, толстовки, куртки, бейсболки,
* Кружки, зажигалки, брелки,
* Часы и многое другое

Тел.: (095) 780-8825
Факс. (095) 780-8824
www.gamepost.ru

GAMEPOST

Крис Касперски aka мыщъх

МАНУАЛЬНАЯ ТЕРАПИЯ

УЧИМСЯ ЛЕГКО ОБХОДИТЬ ТОЧКИ ОСТАНОВА

Если защита не будет препятствовать модификации своего собственного кода, ее немедленно хакнут, а если воспрепятствует - хакнут тем более. Контроль утрачивает свою силу, когда становится явным. Лобовые решения здесь бесполезны! Чтобы выжить, защитный механизм должен хитрить, используя все преимущества страничной организации виртуальной памяти. В этой статье автор делится с читателями передовыми идеями, выдранными из лучших защитных механизмов, которые только существуют.

Традиционные методики контроля целостности в большинстве своем сводятся либо к подсчету контрольной суммы, либо к проверке какого-то конкретного байта. Оба способа просты в реализации, но слишком гуманны. Во всех этих случаях происходит явное обращение к некоторой ячейке памяти, что не есть хорошо! Всякий хакер, хотя бы однажды видевший SoftIce, немедленно нажмет <Ctrl-D> и поставит бряк на эту ячейку, чтобы узнать, какая зараза ее контролирует. Конечно, защитный механизм может активно противодействовать отладчику, но это будет уже перебор. Грамотная антиотладка отнимает уйму времени и усилий, а безграмотная отламывается в два счета.

Необходимо найти такой способ самоконтроля, который никак не препятствует отладке, но волшебным образом обходит все точки останова, даже при запуске под отладчиком-эмулятором. И такие способы есть! Рассмотрим один из них.

Для этого нам понадобится hex-редактор HTE, который можно бесплатно скачать с сервера http://hte.sourceforge.net. Это могучий и послушный инструмент, намного более функциональный, чем HIEW, и к тому же распространяемый в исходных текстах, что позволило портировать его под множество платформ. Правда, набор горячих клавиш не совпадает с HIEW, что поначалу очень раздражает (как ни крути, многолетняя привычка к HIEW берет свое). Впрочем, раскладку горячих клавиш изменить нетрудно. Однако во избежание никому не нужной путаницы в этой статье будет использоваться оригинальная раскладка.

ИДЕЯ

Страничная организация памяти, используемая в 386+-процессорах, абстрагирует нас от физических адресов. Один и тот же физический регион памяти может проецироваться по нескольким виртуальным адресам, но ни отладчик, ни дизассемблер об этом даже не догадываются, поскольку опираются исключительно на виртуальные адреса.

Спроецируем физическую страницу F по виртуальным адресам A и B. Тогда, при записи в страницу A, все изменения немедленно отобразятся в странице B, поскольку в действительности это одна и та же страница! Как можно использовать это на практике? Создадим две переменных var_a и var_b. Первую из них мы разместим на странице A, а вторую - на странице B. Образуется что-то вроде нуль-пространственного туннеля, пригодного для контрабандной переброски данных из одного конца программы в другой. Убийственный трюк, не правда ли?! Допустим, переменная var_a отвечает за флаг регистрации. Дизассемблер покажет множество перекрестных ссылок, ведущих к разным частям защитного кода, но... среди них не будет ни одной перекрестной ссылки на var_b, а значит, часть проверок регистрационного кода останется незамеченной, и хакер будет долго ломать голову, как это так. Самое главное: точка останова, установленная на запись/чтение переменной var_a, при обращении к переменной var_b не вызовет всплытия отладчика!

К сожалению, с прикладного уровня манипуляции со страницами невозможны. Ну, практически невозможны. Небольшая лазейка все-таки есть. Возьмем PE-файл. Как известно, он состоит из секций, то есть непрерывных фрагментов произвольной длины. С каждой секцией связан ряд атрибутов: name - имя; raw offset или просто offset - физическое смещение секции в файле; raw size - размер секции на диске; virtual address, или, сокращенно, va – адрес, по которому секция проецируется в память; virtual size, или v_sz - размер секции в памяти. Есть и другие атрибуты, но эти самые важные.

Весь фокус в том, что один и тот же участок файла может быть спроецирован по нескольким виртуальным адресам! Как раз то, что нам нужно! К сожалению, сразу же после проецирования страница забывает о своем происхождении, то есть соорудить нуль-транспортер на основе секций у нас не получится, однако проконтролировать целостность защитного кода мы вполне сможем. Но для начала - маленький ликбез.

Имя секции может быть любым, операционная система все равно игнорирует его. А вот хакеры реагируют на нестандартные имена вполне адекватно. Чтобы не выделяться, лучше использовать имена вроде ".tsl" (Thread Local Storage - локальные данные потока) или ".rsrc" (сокращение от resource - ресурсы). Тот факт, что содержимое секции не совпадает с ее названием, ничуть не смущает операционную систему, зато на бдительность хакера воздействует самым усыпительным образом.

Физическое смещение секции в файле должно быть кратно степени выравнивания, прописанной в заголовке файла в поле File Alignment. Линкер от Microsoft по умолчанию использует выравнивание в 1000h, а минимальная кратность выравнивания составляет 200h (и хотя Windows NT поддерживает гораздо меньшие значения в 20h и даже 10h, такой файл не сможет работать в Windows 9x, поэтому такое выравнивание в природе практически не встречается).

Виртуальный адрес секции должен быть выравнен на величину Section Alignment, также указанную в заголовке. По умолчанию она чаще всего равна 1000h или 4000h.

Если физический размер секции меньше виртуального, то она исправно грузится в память, а оставшийся хвост заполняется нулями. Если виртуальный размер меньше физического, проекция секции в память автоматически расширяется до физического размера. Короче говоря, из двух размеров всегда выбирается наибольший, и он автоматически округляется до ближайшего Section Aliment в большую сторону.

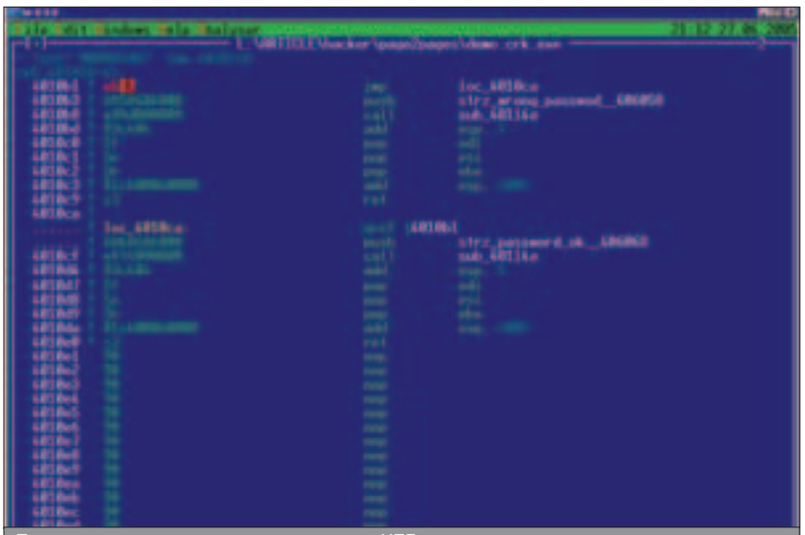
Виртуальный образ не может содержать никаких "дыр". Другими словами, на всем своем протяжении он дол-

жен быть непрерывен. Если мы попытаемся спроецировать секцию по произвольному виртуальному адресу, операционная система жестоко обломает нас. Поэтому виртуальные адреса секций лучше не трогать. Лучше (и безопаснее) оперировать с физическими. Так мы и поступим.

ПЕРВЫЕ ЭКСПЕРИМЕНТЫ

Рассмотрим простейшую программу, которая запрашивает пароль и контролирует целостность своего кода. Ключевой фрагмент исходного текста может выглядеть, например, так (полный вариант можно найти на компакт-диске в файле demo.c):

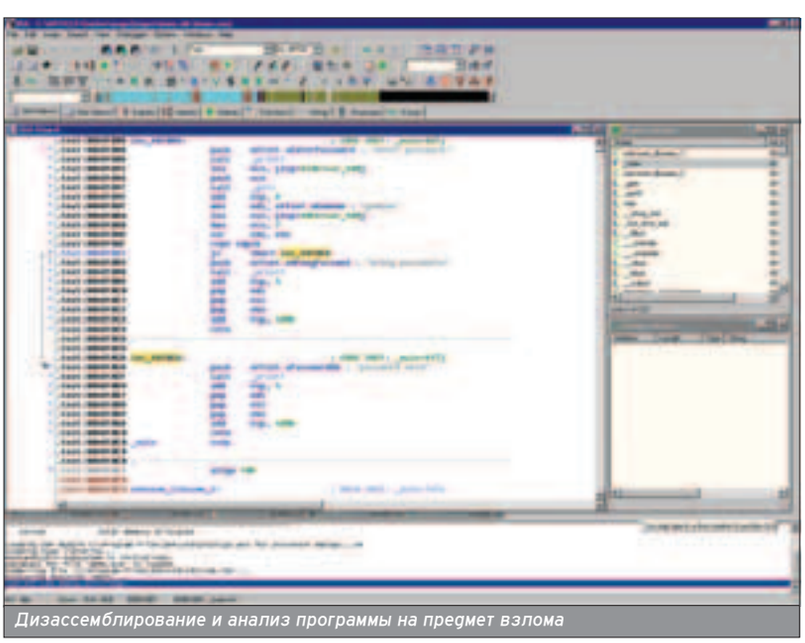
```
#define _PSW_ "godown"     // оригинальный пароль

// начало охранной зоны
begin(){ return 0;}endA();

main(int c, char **v)
{
        int CRC=0;char buf[1024];unsigned char *a;

        // переменная должна быть статической,
        // потому что локальные переменные
        // инициируются прямо в коде, изменяя его CRC
        static int _CRC_= -1;

        // подсчет CRC
        for(a=(unsigned char*) begin;
            a<(unsigned char *)endA;a++)
            CRC = (CRC + *a);

        // отладочная дверь под служебные нужды
        if ((c>1) && !strcmp(v[1],"--debug"))
              printf("%X\n",CRC);

        // если CRC не совпадают, выйти не прощаясь!
        // внимание! нельзя выводить никаких
        // ругательств, иначе нас засекут
        if (CRC ^ _CRC_) return 0;

        // проверка пароля
        printf("enter password:"); gets(buf);
        if (strcmp(buf, _PSW_))
                printf("wrong passwod\n");
        else
                printf("password ok\n");
}

// конец охранной зоны
endA(){ return 0;}
```



Правка условного перехода в редакторе HTE

Откомпилировав программу своим любимым транслятором (например Microsoft Visual C++), запустим ее с ключом "--debug" и посмотрим, какую контрольную сумму она напишет (в моем случае это 47h). Присвоим это значение переменной _CRC_ и перекомпилируем исходный код. Конечно, оставлять отладочный механизм в теле готовой программы нехорошо, однако в качестве демонстрационного примера такой трюк вполне сойдет. Не будем обращать внимание на то, что оригинальный пароль лежит открытым текстом и кто угодно может подсмотреть его. Сосредоточимся исключительно на механизме самоконтроля.

Дизассемблирование защитного механизма сразу же показывает тот ветный условный переход, который рулит всем. Вот он - лежит по адресу 4010B1h (см. скрин).



Дизассемблирование и анализ программы на предмет взлома

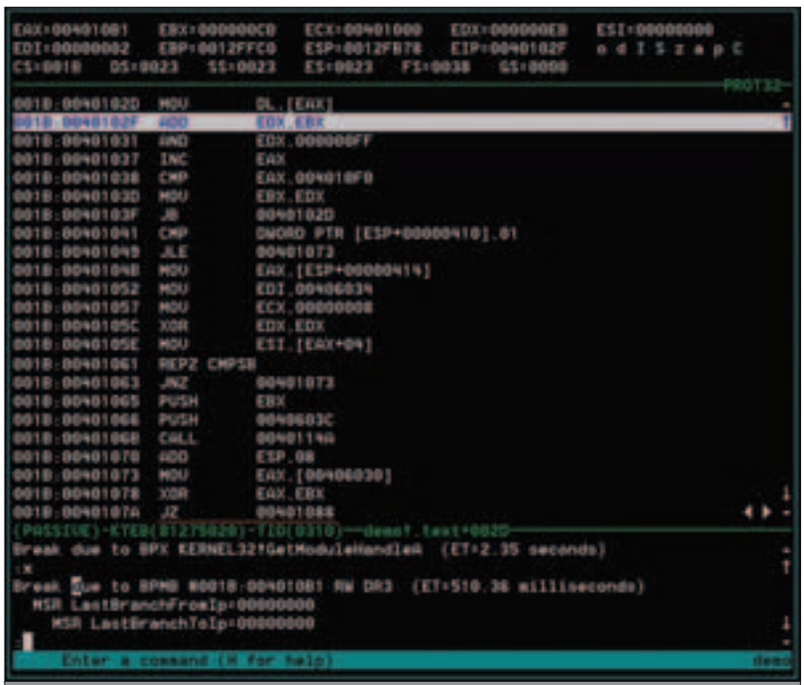
Как видно, он ведет к той ветке кода, которая выводит "password ok", поэтому, если мы заменим jz loc_4010CA на jmp short loc_4010CA, любой пароль будет воспринят как правильный. Для этого необходимо заменить 74 17 (машинный код jz loc_4010CA) на EB 17 (jmp short loc_4010CA). Берем любой hex-редактор, например HTE, создаем копию ломаемого файла (я называл ее demo.crk.exe), загружаем в редактор, нажимаем <F6> (mode), выбираем режим pe/image, давим <F5> (goto), вводим адрес условного перехода (4010B1h), затем нажимаем <F4> (edit) и заменяем "74" на "EB". Клавиша <F2> сохраняет все изменения в файле, а <F10> выводит нас из редактора.

Запускаем хакнутую программу и... опс! Она не запускается! Так где там наш старый добрый SoftIce? Загружаем программу в Symbol Loader, даем команду "bpm 4010B1" (установить точку доступа на обращение к ячейке 4010B1h, в которой расположен хакнутый условный переход) и по <Ctrl-D> выходим из отладчика, возвращая управление программе.

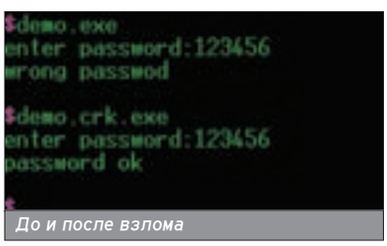
Отладчик тут же всплывает и, словно волшебная лампа Аладдина, мгновенно переносит нас в штаб-квартиру защитного механизма, контролирующего целостность машинного кода.

Так-так-так, контрольная сумма накапливается в регистре EDX (MOV DL, [EAX]/ADD EDX, EBX), который тут же пересылается в регистр EBX (MOV EBX,EDX), а спустя некоторое время EBX сравнивается с какой-то ячейкой памяти (MOV EAX, [406030]/XOR EAX,EBX). Если они идентичны друг другу, выполняется условный переход по адресу 401088h (JZ 401088). Вот этот переход и портит всю малину, препятствуя нормальной работе хакнутой программы. Если изменить XOR EAX,EBX (33 C3) на XOR EBX,EBX (33 DB или 31 DB), программа будет взломана окончательно.

»



Отладчик SoftIce, обнаруживающий стандартный механизм самоконтроля в исследуемой программе



До и после взлома

В редакторе HTE это делается так: загружаем файл, давим <F6> (mode), выбираем pe/image, жмем <F5> (goto) и вводим адрес перехода (401078h), говорим <F4> (edit), а затем <Ctrl-A> (Assemble). Вводим "XOR EBX,EBX", и... HTE запрашивает, каким именно образом мы хотим ассемблировать ее. Этой возможности нет ни у одного другого известного мне hex-редактора! Выбрав любой вариант (оба они двухбайтные), нажимаем <F2> (save) для сохранения изменений и по <F10> выходим из редактора.

ЗАКОНЧЕННАЯ РЕАЛИЗАЦИЯ

Попробуем усилить защищенность механизма самоконтроля. Добавим в начале программы следующие строки (полный вариант можно найти в файле demo.protected.c):

```
// начало нестандартной кодовой секции с именем .tsl
#pragma code_seg(".tsl")

// начало охранной зоны
begin(){ return 0;}

// фиктивная функция, чтобы секция не была пустой
demo(){}

// конец нестандартной секции
#pragma code_seg()

// конец охранной зоны
endA(){ return 0;}
```

Прагма code_seg(имя_секции) предписывает линкеру размещать весь последующий код в секции с именем ".tsl", что он и делает. На самом деле, как мы уже говорили, это никакой не .tsl, а вполне законная секция кода, только с другим названием. Чтобы линкер не отбраковал секцию как ненужную, мы создает фиктивную функцию demo() и окружаем ее "охранной зоной".

Прагма code_seg() отменяет действие предыдущей прагмы, и весь последующий код ложится линкером в стандартную секцию .text или CODE. Поскольку минимальный размер секции составляет 1000h (вспомним про выравнивание), то, расположив endA() после code_seg(), мы получим в свое распоряжение 1000h байт. Только не перепутай их местами, иначе ничего не получится!

Откомпилировав полученный пример, загрузим его в HTE и перейдем в режим отображения заголовка (<F6>, "PE-header"). Мы видим секцию .tls, содержащую фиктивную функцию demo, и секцию .text с подлинным кодом программы. <ENTER> распахивает содержимое атрибутов секций, а <F4> позволяет редактировать их. Очень хорошо!

Секция .tls отстоит на 1000h байт от начала файла, а .text - на все 2000h. Чтобы отобразить первые 1000h байт секции .text на два региона адресного пространства, необходимо изменить raw offset первой секции, передвинув ее на 1000h байт вглубь файла. Подводим курсор к строке offset (здесь должно быть записано 1000h), нажимаем <F4> (edit) и изменяем ее на 2000h. Сохраняемся по <F2> и выходим. Запустив отредактированный файл с ключом --debug, мы видим, что его контрольная сумма не изменилась, значит, все было сделано правильно!

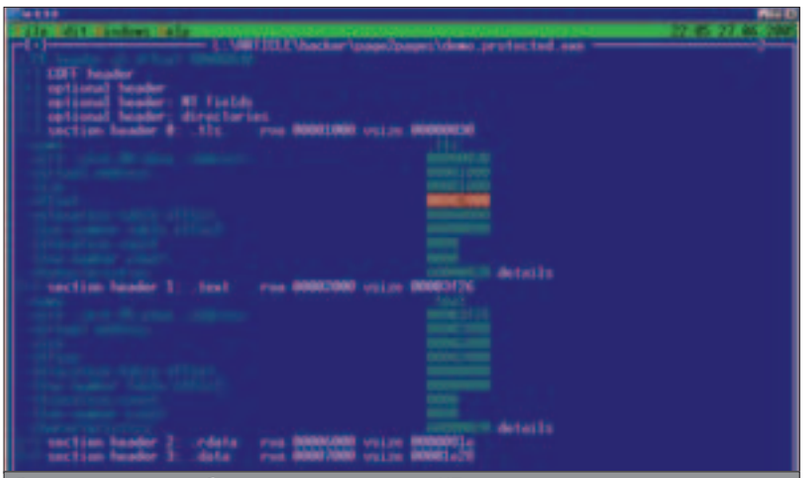
Проверим, сумеем ли мы взломать это? Дизассемблер легко обнаруживает условный переход, определяющий правильность ввода пароля (в моем случае он расположен по адресу 4020A1h). Меняем jz xxx на jmp short xxx. Программа видит, что контрольная сумма изменилась, и уже не срабатывает. Запускаем SoftIce и устанавливаем точку останова по адресу 4020A1h.

Черт возьми, она не срабатывает! Ну не срабатывает и все, хоть ты тресни! Хакер до отупения может перебирать все отладчики один за другим, но от этого ничего не изменится, даже если запустить программу под эмулятором. Вот так прием!

Встроенный ассемблер редактора HTE

НЕСКОЛЬКО ПОЛЕЗНЫХ СОВЕТОВ

В нашем случае охраняемый условный переход располагался в первых 1000h байтах от начала секции .text, поэтому этот трюк и сработал. Но ведь так бывает не всегда, правда?



Редактирование атрибутов секции в редакторе HT

Допустим, защитный механизм расположен по адресу 403069h, что тогда?

Сначала нам необходимо определить относительный виртуальный адрес защитного механизма. Берем 403069h, вычитаем базовый адрес загрузки PE-файла (HTE показывает его в поле image base в разделе "optional header: NT fields", и в нашем случае он равен 400000h), получаем 3069h. Округляем по величине выравнивания File Alignment в меньшую сторону. Получаем 3000h. Это и есть та величина, которую необходимо записать в поле "offset" секции .tls. Тогда в защитную зону попадают все байты, расположенные между адресами 3000h и 4000h.

А если нужно проконтролировать более 1000h байт? Проще всего изменить кратность выравнивания секций (у MS Link за это отвечает ключ /ALING, например /ALING:16384 дает нам 4000h байт). Правда, и размер потребляемой памяти при этом возрастает. Но кто запрещает нам создать несколько подложных секций по 1000h байт каждая? Кончено, слишком большое количество секций обязательно вызовет у хакера подозрения, и он сможет сообразить, что к чему.

В принципе, можно создать секцию данных, разместить в ней статический массив заданного размера и отобразить поверх этого массива контролируемый код, однако тут все не так просто. Гнусный MS link и некоторые другие линкеры насильно комбинируют нестандартные секции с основной секцией данных и никак не дают обойти это ограничение. Имеются и другие проблемы, но не будет углубляться во все посторонние тонкости, а удовлетворимся тем, что есть.

КАК ЭТО ЛОМАЮТ?

Разумеется, предложенная защита - не панацея от всех бед, и она легко может быть взломана, особенно если хакер с ней уже познакомился. Достаточно лишь проанализировать таблицу секций, и мы сразу обнаружим, что один и тот же участок файла отображается по нескольким виртуальным адресам.

Вернемся к нашему примеру demo.protected.exe. Загрузим программу в SoftIce и установим точку останова не на 4020A1h, а на 4010A1h. Она сработает! Хакер может обнаружить этот адрес даже не заглядывая в таблицу секций. Достаточно посмотреть, какие команды расположены в окрестностях модифицируемой ячейки, и отыскать их в дизассемблерном листинге. Они повторяются! А все потому что IDA Pro (и другие правильные дизассемблеры) эмулируют загрузку файла, что разоблачает защитный механизм с головой.

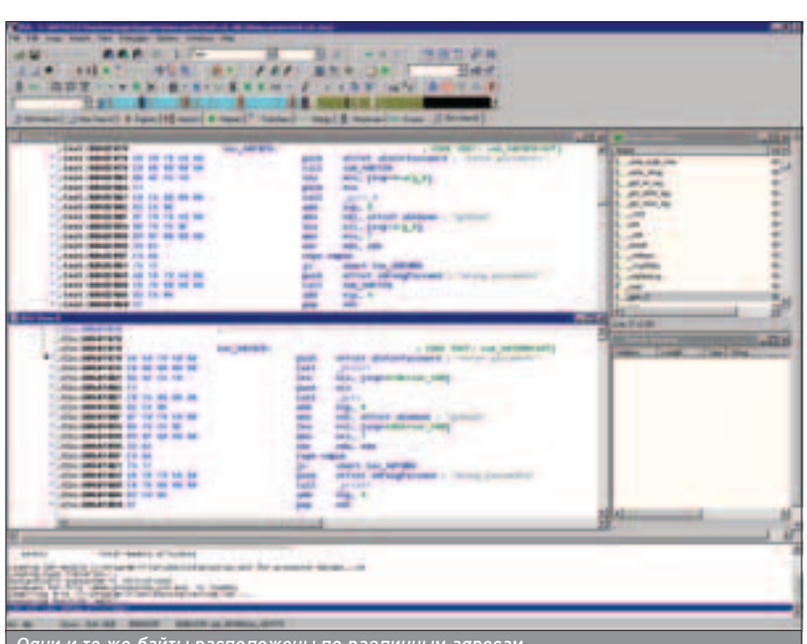
К тому же защита этого типа легко обходится простым онлайновым патчем, то есть изменением байтов не в файле, а памяти: как уже было сказано выше, при проецировании PE-файла в память все связи между родственными страницами утрачиваются, и потому изменение содержимого одной секции уже не вызывает немедленной реакции в другой. Вообще-то онлайновому патчу можно и противостать, поскольку он, как правило, базируется на довольно нежизнеспособной функции WriteProcessMemory из библиотеки TOOLHELP32, интегрированной в ядро Windows 2000 и XP. Но стоит ли? Если хакер всерьез разъярится, он вообще не будет править никаких байтов, а напишет генератор серийных номеров, регистрационных ключей или прочей мути, которая отличает правильную программу от неправильной.

Тем не менее, написание генераторов отнимает порядочное время, и на это решаются немногие. Поэтому абсолютно бесполезным наш трюк никак не назовешь. В комбинации с другими защитными приемами его действие только усиливается, а затраты на реализацию стремятся к нулю. К тому же он абсолютно законен с точки зрения операционной системы, следовательно, побочные эффекты практически исключены. Кстати говоря, обработка защищенной программы упаковщиками исполняемых файлов на ее самочувствие никак не влияет. Но довольно саморекламы, пускай каждый решает самостоятельно, использовать этот трюк или нет.

ЗАКЛЮЧЕНИЕ

У любого защитного механизма есть свои сильные и слабые стороны, и любой из них может быть взломан. Не стоит строить иллюзий. Абсолютно надежных щитов не существует. Идеально острых мечей, кстати говоря, тоже. Война с хакерами породила настоящую гонку вооружений, которая совершенствует как орудия нападения, так и инструменты для отражения атак. В этой игре нет ни правил, ни ограничений. Выигрывает не тот, кто сильнее, а тот, кто сумеет применить неожиданный прием, сбить с толку противника или выкинуть что-то еще.

До тех пор, пока описанная защита не станет популярной, она будет работать на благо наших программ.



Одни и те же байты расположены по различным адресам

Mario555

БОРЬБА С ОТЛАДЧИКОМ

ОСНОВНЫЕ АНТИОТЛАДОЧНЫЕ ФИШКИ В USER MODE

Многим программам, особенно упакованным, не очень нравится, когда их запускают под отладчиком. От них можно ждать чего угодно: могут ограничиться выводом сообщения вроде "Debugger detected", могут завершиться вместе с отладчиком, а могут и жесткий диск попортить. В этой статье я хочу рассказать, как обычно реализуется обнаружение user mode отладчиков (OllyDbg, к примеру) и как избежать этого обнаружения.

АНТИОТЛАДКА

Использование API-функции IsDebuggerPresent

Пожалуй, древнейший способ обнаружения отладчика. "The IsDebuggerPresent function indicates whether the calling process is running under the context of a debugger". Тут все просто: запускаем, смотрим результат, в зависимости от него делаем какую-нибудь гадость. Такое встречается почти в любом протекторе. Минус этой проверки в том, что поймать ее легче легкого: ставим бряк на функцию, подменяем результат - и все, антиотладка идет лесом. Избежать такого простого обхода проверки можно, если разобраться в том, откуда IsDebuggerPresent берет информацию о наличии отладчика. Лезем дизассемблером в библиотеку и видим ее код:

```
77E72740    mov eax, dword ptr fs:[18]    ; TEB
77E72746    mov eax, dword ptr [eax+30]
            ; eax <- адрес PEB из TEB
77E72749    movzx eax, byte ptr [eax+2]
            ; eax <- BeingDebugged
77E7274D    retn
```

PEB содержит информацию о некоторых параметрах процесса. Если посмотреть описание этой структуры, можно заметить, что третий байт в ней - это BeingDebugged, то есть флаг присутствия отладчика. Это значит, что для обнаружения отладчика можно не вызывать API, а просто где угодно в коде проверять значение BeginDebugging. Обход этой штуки напрашивается сам собой: сразу после загрузки в отладчик обнулить флаг BeingDebugged. Какое нападение, такая и защита :).

Поиск окна (или класса окна) отладчика

FindWindow вернет хэндл окна, если оно найдено, либо null, если оно не найдено. С помощью этой замечательной функции можно искать, естественно, не только отладчик, но и любую оконную программу, которой пользуются при взломе (это могут быть, к примеру, мониторы FileMon и RegMon). Такая проверка может осуществляться в отдельном трейде, тогда поиск получится не только до OEP запакованной программы, но и после него, то есть не выйдет даже запустить отладчик одновременно с программой (так, например, прибивают некоторые наиболее распространенные дампера). Соответственно, обходом этого защитного приема будет подмена результата вызова FindWindow, либо, более удобный вариант - замена имени и класса окна отладчика. Используется поиск окна во многих протекторах: ActiveMark, ACProtect (не путать с ASProtect), SoftDefender и др.

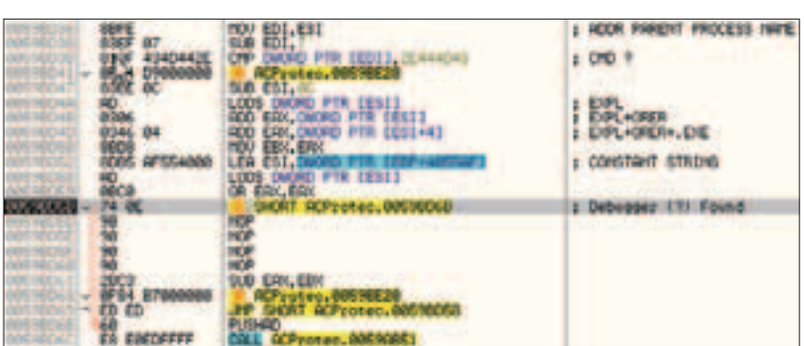
Поиск по имени процесса

Производится с помощью ToolHlp API, функций CreateToolhelp32Snapshot, Process32First и Process32Next, которые перечисляют все доступные процессы и получают для них структуру PROCESSENTRY32, содержащую полезную информацию о процессе. Получив список можно, например, закрыть непонравившийся процесс (скажем, если его имя равно имени процесса какого-нибудь известного отладчика). Еще есть параноидальная проверка у ACProtect: он считает допустимым свой запуск только от explorer.exe и еще пары учтенных программ. Проверка работает элементарно. В структуре PROCESSENTRY32 есть поле DWORD th32ParentProcessID, в котором указан PID процесса-родителя. Если вдруг это поле равно идентификатору неучтенного протектором процесса, то защита просто-напросто убивает своего родителя (печальная история, особенно когда хочешь запустить программу из-под какого-нибудь не очень популярного файлового менеджера).

Хорошо, что авторы ACProtect все-таки образумились и в последних версиях такую проверку убрали, однако ACPR - не единственный протектор, в котором она была :). Обход - переименовать отладчик, к примеру, OllyDbg.exe в explorer.exe. Правда, тут возникает одна проблемка: плагины будут искать именно OllyDbg.exe, так как там находятся функции, которые они импортируют, поэтому в папке должен остаться OllyDbg.exe, будем переименовывать и запускать его копию.

Поиск сигнатур в памяти процесса

Фактически это модификация предыдущего способа, только хитрее - тут уже переименование exe'шника отладчика не поможет. Так же, как и в предыдущем способе, перечисляются процессы с помощью ToolHlp API, потом каждый открывают с помощью OpenProcess и по некоторым адресам ищут сигнатуры неугодных программ, используя ReadProcessMemory. Чтобы защититься от такого обнаружения, можно либо отлавливать вызовы вышеназванных API и подменять значения, которые они возвращают, либо



Проверка имени процесса-родителя в ACProtect

попробовать убрать из exe'шника сигнатуры, по которым его могут найти. Последнее позволит запускать целевую программу (например дампер или импрек), даже если ее поиск осуществляется в отдельном трейде. Применяется в Execryptor2, ActiveMark.

Проверка присутствия отладчика с использованием SetUnhandledExceptionFilter и CheckRemoteDebuggerPresent

Довольно-таки интересная штука. До детального рассмотрения она казалась мне особенно страшной и труднообходимой.

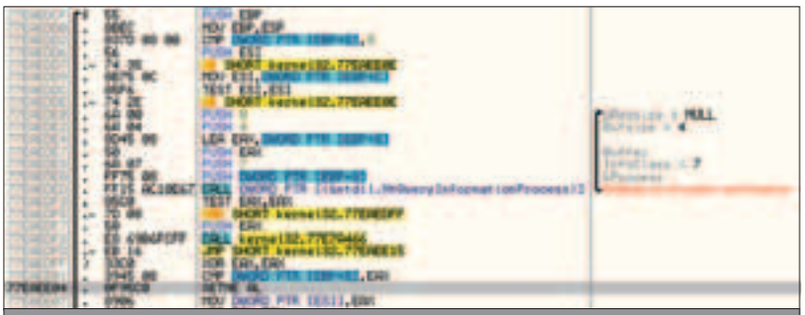
Из справочника по API: "After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the Win32 unhandled exception filter, that filter will call the exception filter function specified by the lpTopLevelExceptionFilter parameter". Следовательно, API поразному ведет себя в присутствии отладчика и без него. И, что хуже, это не изменение возвращаемого значения, которое можно было бы легко подменять, а обработка исключения. Если процесс не отлаживают, то на финальном исключении (не обработанном SEH), которое намеренно делает протектор (к примеру деление на ноль), управление переходит на установленный с помощью SetUnhandledExceptionFilter обработчик, после которого программа спокойно продолжает выполнять свои действия. Если же это происходит во время отладки, то управление передастся не на финальный обработчик, а отладчику, и программа падает, так как отладчик не знает, что делать. Посмотрим на код SetUnhandledExceptionFilter:

```
77E7E5A1    mov ecx, dword ptr [esp+4]
77E7E5A5    mov eax, dword ptr [77ED73B4]
77E7E5AA    mov dword ptr [77ED73B4], ecx
77E7E5B0    retn 4
```

Смотреть, получается, почти не на что: в переменную записывается адрес финального обработчика, она будет читаться в функции UnhandledExceptionFilter. Самое интересное место функции кода можно посмотреть на скриншоте.

Ниже адреса 77E93114 лежит процедура, передающая управление обработчику, установленному SetUnhandledExceptionFilter. Под отладкой этот код не получает управление, так как осуществляется переход по адресу 77E9310E. Соответственно, если занопить этот переход, то управление

UnhandledExceptionFilter



CheckRemoteDebuggerPresent

Под отладчиком программа выполняется медленнее, чем без него.

будет передаваться обработчику, как если бы отладчика не было. Здорово!

Теперь о CheckRemoteDebugger-Present. Ее применение я впервые увидел в Obsidium 1.2, там же был и трюк с SetUnhandledExceptionFilter.

```
BOOL CheckRemoteDebuggerPresent(
 HANDLE hProcess,
 PBOOL pbDebuggerPresent)
```

Эта функция, в принципе, аналогична IsDebuggerPresent, но только по возвращаемым результатам. А ее начинка не похожа на IsDebuggerPresent совсем. Она, как видно, может проверять и другой процесс на наличие отладки, и способ проверки отличается от простого просмотра байта в PEB. Впервые она появилась в Windows XP SP1, поэтому ее использование не универсально. Функцию CheckRemoteDebuggerPresent можно, как обычно, перехватывать, а возвращаемые ею значения - подменять. Однако не остановимся на достигнутом и копнем поглубже, чтобы найти способы обхода. Смотрим код функции на скриншоте.

И опять видно использование NtQueryInformationProcess, как и в UnhandledExceptionFilter. Именно она определяет наличие отладчика, информацию о котором берет из ring0. Вызов NtQueryInformationProcess с параметром InfoClass = 7 (7 = ProcessDebugPort из структуры EPROCESS) является еще одним способом определить, отлаживают ли нас. Если устранить эту проверку с помощью перехвата данной API, антиотладочные трюки и с SetUnhandledExceptionFilter, и с CheckRemoteDebuggerPresent будут обезврежены. В чистом виде проверка через NtQueryInformationProcess есть, например, в Safedisk и SoftDefender, а косвенные (то есть с использованием двух вышеупомянутых API) есть в Obsidium и Execryptor2.

Баг в OllyDbg при работе с OutputDebugString

В Windows есть несколько функций API, предназначенных для взаимодействия с отладчиком, одна из них - OutputDebugString. Она посылает отладчику строку, которую тот в свою очередь показывает пользователю. В отсутствие отладчика OutputDebugString ничего не делает. Баг заключается в неправильной обработке посылаемой строки в OllyDbg. Если эта строка будет вида "%s%s%s%s...", то Olly упадет с ошибкой чтения по адресу 00000001h. Соответственно, сдохнет и отлаживаемая программа. Чтобы не дать уронить Olly, нужно не дать выполниться OutputDebugString с такой строкой либо пропатчить Olly, чтобы он вообще не реагировал на посылаемые строки. Мне больше нравится последнее, тем более что это нетрудно сделать. Этот антиотладочный трюк применяется в Armadillo, Execryptor2.

Проверки времени (RDTSC, GetTickCount и т.п.)

Под отладчиком программа выполняется медленнее, чем без него - на этом и основывается следующий метод антиотладки. Команду RDTSC обычно используют для определения тактовой частоты процессора и для измерения малых интервалов времени, она считает число тактов процессора с момента нажатия RESET. Замерив, сколько времени исполняется код без отладчика, можно внести его как эталон, и если время выполнения будет намного больше него, делаем вывод, что программу отлаживают. Таким же образом используют функцию GetTickCount, которая возвращает количество миллисекунд, прошедших с момента запуска Windows. Бороться с такими трюками сложно. Если GetTickCount еще можно перехватывать, то за RDTSC зацепиться трудно: приходится искать ее в коде и смотреть, где именно происходит про- »

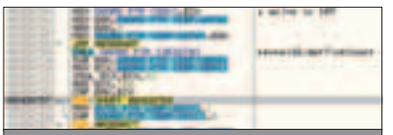
верка, которая может быть не единственной и далеко не единственной. Но, к счастью, использование таких трюков крайне ненадежно, так как процессоры бывают разные и скорость выполнения на них тоже бывает разная, да и многозадачность идет не на пользу таким проверкам. Поэтому для обнаружения отладчика они применяются все реже и реже. Однако более мягкие проверки, нацеленные на обнаружение остановок во время выполнения, являются довольно надежными (задан относительно большой допустимый интервал времени) и применяются часто, но об этом позже. Жесткие проверки особенно извращенно применяются в SoftDefender и еще некоторых малоизвестных пакерах, авторы которых не слишком заботятся о совместимости. SoftDefender - это глюковатый китайский протектор, который ты вряд ли когда-нибудь встретишь на скачанной шароваре. Но с точки зрения антиотладки в нем есть некоторые интересные вещи, и об одной из них сейчас расскажу.

Вот необычная проверка времени. Ее суть не в измерении времени выполнения кода, а в том, что при загрузке программы в отладчик мы тратим много времени (по процессорным меркам), даже перед нажатием RUN в отладчике. Протектор, используя NtQueryInformationProcess с InfoClass = ProcessTimes, получает время, когда процесс был создан. Потом через GetSystemTimeAsFileTime получает текущее системное время, далее следует вычитание из одного другого и сравнение с заданным допустимым интервалом времени, который выбран относительно большим, поэтому ложных срабатываний на медленных процессорах не будет, но в то же время этот интервал меньше, чем время, за которое ты успеешь запустить программу в дебаггере.

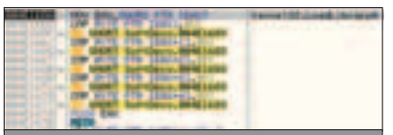
Еще один способ обнаружить отладчик – искать по тем изменениям, которые происходят в программе при запуске под ним (но не являются четко документированными признаками отладки, как, например, байт BeingDebugged в PEB). Я бы вообще промолчал о них (так как доподлинно не известно, надежны они или нет), если бы не их применение в ExeCryptor, который порядочно распространился в последнее время. Там проверяется значение NtGlobalFlag из PEB: без отладчика оно равно нулю, с отладчиком и по умолчанию в XP SP1 оно равно 70h, и DWORD по смещению + 10h в ProcessHeap (взятом из PEB) - без отладчика там ноль, с отладчиком по умолчанию в XP SP1 там 40000060h.

Проверка NtGlobalFlag в ExeCryptor'e

Проверка Heap'a в ExeCryptor'e



Проверка времени выполнения кода в Armadillo



Стандартная проверка наличия байта CCh в SoftDefender

> Когда исследуешь протектор, ставить бряки на начало API-функции, мягко говоря, опрометчиво.

Это были, в принципе, все наиболее распространенные трюки, используемые ring3-протекторами для обнаружения ring3-отладчиков. Существуют также ненадежные антиотладочные трюки вроде проверок, которые могут быть применены только при определенных условиях (версия Windows (9x/Nt) или наличие администраторских привилегий).

АНТИТРАССИРОВКА

Антитрассировкой я буду называть процесс, который мешает отлаживать (читай "исследовать/взламывать") программу, но не запрещает простой запуск программы под отладчиком, то есть противостоит действиям взломщика, а не отладке вообще.

Использование замеров времени

Об этом я уже рассказывал выше, поэтому здесь приведу конкретный пример. Для получения чистой IAT во многих протекторах нужно править что-либо в процедуре заполнения этой самой IAT. В Armadillo нужно занопить всего один условный переход, но для этого ты должен предварительно найти его (даже если знаешь, где искать, все равно потратишь время) плюс поставить бряк на конец процедуры (чтобы восстановить измененные байты). Короче, затратить промежуток времени, очень большой по сравнению с тем, сколько эта процедура выполняется без вмешательств. И Arma проверяет это время.

На куске кода видно, что по адресу 3DA7BA вызывается GetTickCount, а несколько ниже по адресу 3DA7D7 происходит сравнение результата с допустимым значением: если меньше или равно, то JBE выполняется и все ОК; если больше, то не выполняется и в переменную [EBP-26CC] записывается единица, которая сигнализирует о том, что некто копается в процедуре обработки импорта. При этом протектор не будет выдавать никаких сообщений об отладчике и т.п., а под шумок заведомо ошибочно выполнит заполнение IAT, и придется потратить время, чтобы найти эту проверку.

Защита от обычных CC-бряков

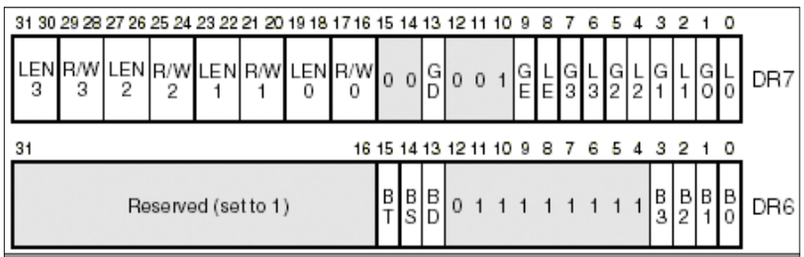
Когда исследуешь протектор, ставить бряки на начало API-функции (как советуют в некоторых статьях bp addr_api), мягко говоря, опрометчиво. Почти все защиты проверяют, по крайней мере, исследуется первый байт вызываемой функции на присутствие байта CCh.

Есть и простенькие проверки, как, например, в SoftDefender.

Тупо проверяются первые пять байт. Взломщику в этом случае нужно ставить бряк на середину или на конец (ret) функции. Однако бывают и сложные проверки (например в ExeCryptor2 или ASProtect), когда почти вся основная ветвь вызываемой API дизассемблируется и, соответственно, проверяется целиком на наличие CCh. В таких случаях нужно либо ставить точку останова куда-нибудь вглубь API-функции (в call'ы), либо воспользоваться другим типом бряка, например на доступ к памяти. Но байт CCh ищут не только в API. Когда ты выполняешь Step over (<F8> в Olly) через какой-нибудь call, на следующую за ним команду отладчик ставит CCh-бряк. И он может быть легко обнаружен в подпрограмме внутри call'а, поэтому в таких случаях взломщику предстоит использовать не Step over, а вручную ставить бряк куда-нибудь после call, но не на первую после него команду, и выполнять RUN (<F9>).

Защита от точек останова на доступ к памяти (BPM)

Такие бряки очень удобны для поиска OEP в запакованных программах, если протектор не украл начальные байты. Поэтому во многих статьях по относительно простому софту советуют ставить BreakPoint memory on access на секцию кода. Эти бряки основаны на изменении атрибутов доступа страницы памяти, к которой отно-



Формат регистров DR6 и DR7

CONTEXT содержит информацию о текущем состоянии потока, то есть значения регистров и т.п., включая отладочные DRx-регистры.

сится адрес, на который установлен BPM. Когда ставится memory on access, Olly, используя функцию VirtualProtect, меняет доступ к странице на PAGE_NO_ACCESS, а когда memory on write, то на PAGE_EXECUTE_READ. Соответственно, при нарушении прав доступа к странице (попытке чтения, выполнения или записи) управление переходит к отладчику, который уже смотрит, к тому ли адресу происходит обращение. Минимальный размер страницы памяти в Windows равен 1000h. Если ты поставил BPM на доступ всего к одному байту, то отладчик все равно будет внутренне обрабатывать обращение ко всей странице, на которой расположен этот байт, поэтому BPM, особенно on access, может сильно замедлить выполнение программы. Такой отладочный метод обнаруживается проверкой атрибутов доступа с помощью VirtualQueryEx. Соответственно, если обнаружено, что доступ к странице не такой, каким должен быть, то протектор, используя VirtualProtect, либо меняет его (что фактически означает отключение установленного BMP - не стоит сильно удивляться, если он не сработает), либо выдаст сообщение об обнаруженном отладчике и т.п.

Защита от Hardware Breakpoints (HW)

HW-бряки, пожалуй, самые удобные при исследовании программ. Они работают на отладочных DRx-регистрах, прямой доступ к которым из ринг3 невозможен. Установленные HW-бряки не меняют код программы и доступ к страницам памяти, но все равно обнаружить их очень просто... Для начала немного информации о DRx. Есть всего шесть доступных регистров отладки: DR0-DR3 хранят адреса установленных бряков (отсюда и ограничение количества оных - всего четыре штуки), DR6 и DR7 предназначены для контроля и задания параметров первым четырем, DR4 и DR5 - не используются (зарезервированы). Формат регистров DR6 и DR7 ты можешь увидеть на картинке.

DR6 показывает текущее состояние бряков, поэтому не слишком интересен, а вот DR7, можно сказать, управляет ими: L0-L3 - биты, означающие, активирован или деактивирован соответствующий DR0-3 в контексте данного потока; RW0-RW3 - биты, определяющие условие соответствующего бряка DR0-3:

00 - on execute,
01 - on write,
10 - обращение к порту ввода-вывода,
11 - on access;

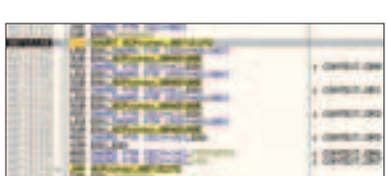
LEN0-LEN3 - длина бряка:

00 - byte,
01 - word,
10 - не определено,
11 - dword.

Видно, что в DRx заключены широкие возможности для отладки программы, и ко всем этим возможностям имеет доступ сама программа! Дело в том, что в обработчике исключений она получает доступ к структуре CONTEXT (которой, кстати, и пользуется отладчик через функции SetThreadContext и GetThreadContext). CONTEXT содержит информацию о текущем состоянии потока, то есть значения регистров и т.п., включая отладочные DRx-регистры. Если изменить значение какого-либо регистра в CONTEXT, то при возвращении из обработчика исключений оно окажется в самом этом регистре. Таким образом, программа может косвенно читать и писать в DRx. Поэтому почти во всех протекторах практически бессмысленно устанавливать такой удобный для нахождения OEP (или начала краденых байт) HW-бряк на восстановление стека [esp-4] на EP - протекторы в обработчиках искусственно сделанных исключений записывают в CONTEXT.DRx нули или мусор.

Но запись нулей - это еще не самое интересное. Иногда применяется самотрассировка: с помощью CONTEXT.DRx протектор расставляет в своем коде HW-бряки, и если какой-то из них не сработает, то выполнение пойдет неверным путем и программа упадет. Такая самотрассировка применена, например, в ACProtect.

Код одного из обработчиков исключений в obsidium'e



Кусок кода одного из обработчиков исключений ACProtect

Running line

Структуру CONTEXT в протекторах используют не только для управления DRx-регистрами. Еще одним примером антитрассировочного трюка с ее использованием является так называемая running line-трассировка. Заключается она в следующем: сначала флаг T (trace) устанавливается равным единице следующим кодом:

```
pushfd
pop eax
push eax
or ah, 1
push eax
popfd ; T=1
nop
```

При установленном флаге трассировки T=1 выполняется одна следующая команда, после которой флаг T сбрасывается в ноль и возникает исключение Single step event. В приведенном коде это происходит после команды NOP, за которой следуют специфически зашифрованные команды. К примеру, в каждой из них поксорен первый байт (поэтому они выглядят как куча мусора). На возникшем исключении Single step event управление передается обработчику, в котором первый байт команды расксоривается, первый байт предыдущей команды ксорится и модифицируется значение регистра флагов Eflags в структуре CONTEXT (выставляется T=1). Соответственно, после выполнения следующей (только что расшифрованной) команды происходит то же самое и т.д. Разбирать такой живой код под отладчиком очень неудобно. Применяется running line в XtreamLok (включен в SoftWrap).

THE END

Вот я и подошел к концу. Надеюсь, эта статья помогла тебе разобраться, как работают самые простые, но популярные антиотладочные приемы в user mode. Может быть, информация будет полезна тебе независимо от того, собрался ты писать защиту или ломать ее.

Bit-hack (bit-hack@mail.ru)

НАВЕСНАЯ ЗАЩИТА

РАЗБОР ПОЛЕТОВ СРЕДИ СИСТЕМ ЗАЩИТЫ И УПАКОВКИ WIN32 PE-ФАЙЛОВ

Что такое навесная защита, каковы ее достоинства и недостатки, чем упаковщики отличаются от протекторов и зачем нужны скрэмблеры – обо всем этом ты узнаешь из этой статьи. Кроме того, сможешь научиться писать скрэмблер UPX самостоятельно.

НЕМНОГО ТЕОРИИ

Для начала кратко опишу принципы работы упаковщиков. Обычно они "навешиваются" на уже скомпилированные PE-файлы (win32 Portable Executable), уменьшая их размер и частично обеспечивая защиту исполняемого кода и некоторых других составляющих PE-файла. Стоит, впрочем, отметить, что существуют упаковщики, которые выполнены, например компоненты Delphi, но такие типы упаковщиков мы сегодня рассматривать не будем, так как нам не требуется привязка к одному языку.

Итак, принцип работы рассматриваемых упаковщиков (операции с файлом, по шагам):

❶. проверка файла (на принадлежность файла к win32 PE, на упакованность, на возможность сжатия);
❷. упаковка кода и некоторых других частей файла;
❸. добавление распаковщика;
❹. правка некоторых полей PE-заголовка (для дальнейшей работоспособности файла).

Но иногда попадаются упаковщики, которые еще сильнее изменяют файл, например UPX, перестраивающий весь файл во внутренний формат. В этом случае запуск происходит по еще более простой схеме (для упаковщиков):

❶. Выполнение распаковщика.
❷. Переход на OEP (оригинальная точка входа в программу), то есть на тот код, с которого и начиналась незапакованная программа. На ассемблере такой переход выглядит примерно так (пример на UPX):

```
POPAD ; Восстановили все процессорные регистры
JMP 01012475 ; Перешли на OEP
```

❶. Сохранение всех процессорных регистров и (иногда) флагов.
❷. Сбор служебной информации.
❸. Подготовка протектора к работе (заполнение необходимых таблиц, значений).
❹. Выполнение кода для обнаружения отладчиков (редко помогает, так как у реверс-инженеров уже написаны средства для борьбы с таким кодом, точнее не с кодом, а с тем, как этот код определяет отладчик).
❺. Проверка CRC (циклический контроль избыточности). CRC используется для проверки файла на изменение.
❻. Выполнения кода расшифровщика (часто расшифровщик не один), напичканного антиотладочным кодом и мусором.
❼. Создание таблицы импорта (обычно она сильно видоизменяется, создаются переходники на оригинальные функции, также функции эмулируются, копируются).
❽. Стирание всего кода протектора (это не всегда - зависит от протектора).
❾. Продолжение нормального выполнения программы.

Некоторые упаковщики (не протекторы!) содержат в себе и распаковщики (для своего же упаковщика). Пример такого упаковщика, опять же - UPX, распаковщик которого не только распаковывает файл, но и перестраивает структуру файла в первозданный вид. Но, как всегда, программисты решили защищать свои творения упаковщиком, а в результате появились так называемые скрэмблеры, которые слегка изменяют файл, чтобы тот не был опознан распаковщиком и принят как непригодный для распаковки... Но и воюющая с ними сторона не стала складывать руки, и были написаны программы для восстановления структуры файлов, чтобы распаковщик смог сделать свое "черное дело". В некоторых случаях эти программы сами не справляются, так что после них приходится заканчивать восстановление руками.

А для тех упаковщиков, которые не содержат в себе распаковщиков, пишутся (естественно, другими автора-

Оригинальный размер файла	21504	6750208	7619	846336
Упаковщики				
ASPACK 2.12	17408	2683904	10240	387584
EXE32PACK 1.38	18230	3287498	7108	477548
EZIP 1.0	47585	4679082	39180	735467
FSG 2.0	11789	3185885	4337	439673
JDPACK 1.01	17920	3887104	8704	470528
MEW10 1.1	12526	3161991	4278	463181
MEW11 SE 1.0	11273	2508897	4215	363849
PACKLITE	15360	4121088	6144	560128
PCSHRINKER 0.71	14336	3872256	6144	438784
PECOMPACT 2.38(Aplib)	14336	3068928	7168	413696
PECOMPACT 2.40(BriefLZ)	16384	3630592	8192	475648
PECOMPACT 2.40(FFCE)	14336	2876928	7680	405504
PECOMPACT 2.40(LZMA)	14848	2452480	8192	367104
PEDIMINISHER 0.1	16922	3204122	6682	560154
PEPACK 1.0	15872	3814400	7680	425984
UPX 1.92 beta	12800	2774016	5632	395776

Сравнительная таблица упаковщиков файлов

```
      File size         Ratio      Format      Name
--------------------   ------   -----------   -----------
upx: calc.exe: NotPackedException: not packed by UPX

Unpacked 0 files.

D:\Unpack\upx sc>
```

Файл, упакованный UPX и обработанный DotFix FakeSigner, при попытке автоматической распаковки (upx.exe -d filename). UPX просто не узнал файл

Dotfix.net – настоятельно рекомендуется к прочтению

ми, обычно реверс-инженерами) автораспаковщики, которые исполняют свои обязанности пусть и хуже, чем распаковщики от авторов (например, недавно мной был написан распаковщик для UPX), но распаковывают файлы, после чего можно начать исследовать код с целью его "заимствования" или просто взлома программы. Опытные люди распаковывают файлы "руками", берется отладчик, дампер процессов и программа для восстановления импорта, после чего все "смешивается в нужных пропорциях". При ручной распаковке во время запуска любое приложение оказывается распакованным (когда этот код уже распакован и способен нормально работать, отслеживается передача управления из кода протектора/упаковщика в код оригинальной программы, после чего весь файл скидывается из памяти на жесткий диск и правится).

ДЛЯ ТЕХ, КТО МОЖЕТ

Люди, имеющие деньги, защищают свои программы с помощью другого подкласса упаковщиков – протекторов. Эти защитные системы специализируются на защите, а не на упаковке (хотя тоже упаковывают). Многие программисты считают, что защитные механизмы, написанные реверс-инженерами (пусть даже бывшими) принесут только зло и что реверс-инженеры будут их ломать сами... Это большая ошибка: тот, кто занимался или занимается реверсингом, лучше всех знают про всевозможные уловки, секреты и многое другое, что помогает создать действительно надежные защитные системы. А взлом программ, упакованных своим же протектором – неблагодарное дело, из-за которого популярность протектора может упасть навсегда. Надеюсь, эта статья развеет такие заблуждения. Какие же защитные системы распространены шире и популярнее остальных?

❶. **AsProtect** – протектор, написанный отечественным программистом – Алексеем Солодовниковым (когда-то занимался взломом). Он создал действительно мощную систему защиты. Основные преимущества этого протектора:

❶.❶. Защита кода программы с помощью очень стойких криптоалгоритмов.

❶.❷. Защита таблицы импорта (в последних версиях протектора защита позаимствована из Obsidium).

❶.❸. Предоставление служебных функций для проверки регистрационных ключей и для многого другого.

❶.❹. Использование виртуальной машины, при помощи которой защищается главная ветвь кода программы.

❶.❺. Упаковка файла.

❶.❻. Хорошая техническая поддержка.

❶.❼. Сильная интеграция в код программы, очень серьезно мешающая при распаковке – приходится изощряться любыми способами и восстанавливать недостающий код.

❷. **Armadillo** – довольно старый протектор, содержащий некоторые революционные технологии. Его основные преимущества:

❷.❶. Защита кода программы с применением очень стойких криптоалгоритмов.

❷.❷. Защита таблицы импорта.

Помимо знаменитого AsProtect, Алексей Солодовников написал также упаковщик AsPack

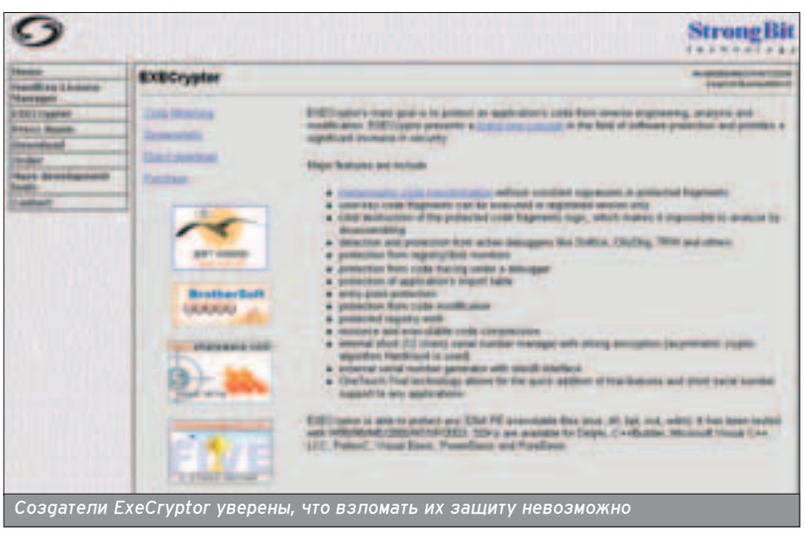
❷.❸. Предоставление служебных функций для проверки регистрационных ключей (в данном протекторе используется плохой алгоритм - были найдены "дыры", а после этого были "закейгенены" приложения, защищенные Armadillo и использующие функции проверки регистрационных ключей) и многих других.

❷.❹. Упаковка файла.

❷.❺. Применение технологии CopyMem II, которая ни за что не позволит реверс-инженеру получить весь код программы (сразу :), а только если докопировать недостающие куски программы по мере надобности.

❷.❻. Применение технологии Nanomites. При защите эта опция является самой мощной. Очень немногие реверс-инженеры, умеющие снимать Armadillo почти со всеми опциями, умеют снимать с этой опцией. Эта опция затирает все переходы в программе (пределы помечены программистом), заменяя из специальной отладочной командой int 3, и при попытке их выполнения защита вставляет их обратно (затирая, когда нужда в них пропадает). Снятие этой защиты со всеми опциями занимает примерно один-два часа... Armadillo – моя любимая защита.

❸. **ExeCryptor 2** – по словам авторов, данная защита неломаема :). Как на-



Создатели ExeCryptor уверены, что взломать их защиту невозможно »

ивно... Как говорил ORC, все, что можно запустить, можно и взломать. Эти слова подтверждаются и в данном случае, о чем мне недавно заявил очень влиятельный человек. Взлом этой защиты является очень затратным (люди, написавшие инструмент для снятия защиты, просят за него от $1000). Однако на эти деньги можно получить неплохую гарантию того, что программа не будет взломана. Преимущества:

❸.❶. Применение виртуальной машины.

❸.❼. Обработка кода метаморфным движком (это главное преимущество).

❸.❽. Защита кода программы с применением очень стойких криптоалгоритмов.

❸.❿. Защита таблицы импорта.

❶. **Obsidium** - относительно новый протектор, средний по стойкости, не имеет особых возможностей для надежной защиты. Преимущества:

❶.❶. Хорошая защита импорта. Эта технология оригинальна: реверс-инженеру, не знакомому ни с чем подобным, будет трудно разобраться с защитой.

❶.❼. Защита кода программы с применением очень стойких криптоалгоритмов.

❶.❽. Упаковка файла.

❶.❿. Предоставление служебных функций для проверки регистрационных ключей и многое другое.

❾. **VmProtect** - защита, основанная на превращении нужного кода в псевдокод (код, который интерпретируется виртуальной машиной), который затем интерпретируется мощной виртуальной машиной. Достоинства:

❾.❶. Применение мощной виртуальной машины, для которой очень сложно написать декомпилятор (преобразователь из псевдокода в стандартный машинный код) псевдокода.

❾.❼. Защита бесплатна.

Среди клиентов StarForce числятся такие компании, как 1С и "МедиаХаус"

СКРЭМБЛЕР UPX СВОИМИ РУКАМИ

Одной из функций PeStubOEP является скрэмблер сигнатур различных упаковщиков. Попробуем написать скрэмблер UPX. Если кто-то забыл, то напомню, что если запустить UPX с ключом -d, он распакует упакованный файл, но только в случае если сигнатура не повреждена. Также мы научимся изменять имена секций, так как не очень-то весело видеть подобное:

```
.UPX0   сюда UPX кладет информацию из .tls
.UPX1   здесь располагается весь упакованный код и данные программы
.rsrc   ресурсы
```

Нам понадобится только информация о расположении PE-заголовка и адрес начала секций. Так вот, несмотря на то, что многие любят ломать DOS-заголовок, по смещению &H3C всегда находится адрес на начало PE-заголовка (его там нет только если программа была написана под DOS). Теперь, когда уже найден адрес начала PE-заголовка, найдем адрес начала описания секций. Как ни странно, описание секций всегда начинается со смещения (PeOffset+&HF9), где PeOffset - адрес начала PE-заголовка. Осталось только определить число секций, а оно хранится по смещению (PeOffset+6) в основном PE-заголовке. Теперь, когда все найдено, не мешало бы разобраться со структурой описания секций. Приведу ее вариант для VB:

```
Public Type pe_section_header
    section_name As String * 8 ' Имя секции [8 байт]
    section_size As Long       ' Размер секции в памяти
    section_rva As Long        ' Адрес загрузки секции в памяти
    section_size2 As Long      ' Размер секции в файле
    section_start As Long      ' Смещение начала секции в файле
    reserved As String * 12
    section_flags As Long      ' Флаги секции
End Type
```

Структуры, подобные приведенной выше, идут одна за другой, а их количество равно числу секций в файле.
Нижеприведенная структура идентична для UPX 1.24 и 1.90, а также вряд ли поменяется в следующих версиях упаковщика, и я думаю, что можно смело пользоваться данной структурой. Так вот, сама структура содержит в себе поля, необходимые для нормальной распаковки файлов UPX'ом и всегда начинается с "UPX!". Этим мы и воспользуемся для нахождения адреса начала сигнатуры.

```
Public Type UPX_STRUCT
    upxMagic As String * 4    ' Символы "UPX!"
    upxVersion As Byte        ' Версия UPX'a (например: 0C значит 1.24, 0D - 1.90)
    upxFormat As Byte         ' Определяет формат файла (PE, ELF, DOS и т.д.) PE - 09
    upxMethod As Byte         ' Метод сжатия (если NRV или UCL, то 02)
    upxLevel As Byte          ' Степень сжатия (от 0 до 10)
    upxU_adler As Long        ' CRC части экзешника в распакованном виде
    upxC_adler As Long        ' CRC части экзешника в запакованном виде
    upxU_len As Long          ' Размер части экзешника в распакованном виде
    upxC_len As Long          ' Размер части экзешника в запакованном виде
    upxU_file_size As Long    ' Размер распакованного экзешника.
    upxFilter As Integer      ' Метод распаковки
    upxCRC As Byte            ' CRC сигнатуры
End Type
```

СКРЭМБЛЕР UPX СВОИМИ РУКАМИ (ПРОДОЛЖЕНИЕ)

Теперь у нас есть все необходимое для написания собственного скрэмблера. Полученных знаний вполне достаточно, чтобы переименовать все секции в файле, например на ".sux", и стереть все данные из UPX-сигнатуры. Для начала запустим Visual Basic 6.0, создадим новый проект, удалим из него форму и добавим модуль (наша программа будет работать с командной строкой). Ниже я приведу простейшую реализацию этой задачи. Для того чтобы не повторятся, я убрал объявления вышеописанных структур:

```
Public sSections() As pe_section_header
Private Sub Main()
sFileName = Command$
If sFileName = "" Then MsgBox "Вы не передали имя файла в командной строке", _
vbCritical, "Ошибка": Exit Sub
If Left$(sFileName, 1) = Chr(34) Then sFileName = Mid$(sFileName, 2, _
Len(sFileName) - 2)
Dim sPeOffset As Long, sFindUPX As String * 5000, sUPX As UPX_STRUCT, _
sNumberSections As Integer
FileCopy sFileName, Left$(sFileName, Len(sFileName) - 3) & "bak"
sFile = FreeFile
Open sFileName For Binary As #sFile
Get #sFile, &H3C + 1, sPeOffset
Get #sFile, sPeOffset + 7, sNumberSections
sSectionsStart = sPeOffset + &HF9 'ищем сигнатуру UPX в первых 5000 байт программы
Get #sFile, 1, sFindUPX
sFind = InStr(1, sFindUPX, "UPX!")
If sFind > 0 Then
  Get #sFile, sFind, sUPX  'заполняем все поля ерундой
  sUPX.upxMagic = "GPcH": sUPX.upxVersion = 0
  sUPX.upxC_adler = 0: sUPX.upxC_len = 0
  sUPX.upxCRC = &HFF: sUPX.upxFilter = 0
  sUPX.upxFormat = 0: sUPX.upxMethod = 0
  sUPX.upxU_adler = 0: sUPX.upxU_file_size = 0
  sUPX.upxU_len = 0: sUPX.upxVersion = 11
  Put #sFile, sFind, sUPX
Else
  MsgBox "Ошибка: файл '" & sFileName & "' не содержит сигнатуру UPX", _
  vbCritical, "Скрамблер UPX": Close #sFile: Exit Sub
End If
ReDim sSections(sNumberSections - 1)
Get #sFile, sSectionsStart, sSections
For i = 0 To sNumberSections - 1
  sSections(i).section_name = ".sux"
Next
Put #sFile, sSectionsStart, sSections
Close #sFile
MsgBox "Файл '" & sFileName & "' успешно обработан", vbInformation, "Скрамблер UPX"
End Sub
```

Вот, собственно, и все. Не удивляйся, что к каждому смещению при считывании прибавляется единица - это особенности Visual Basic'a.

GPcH

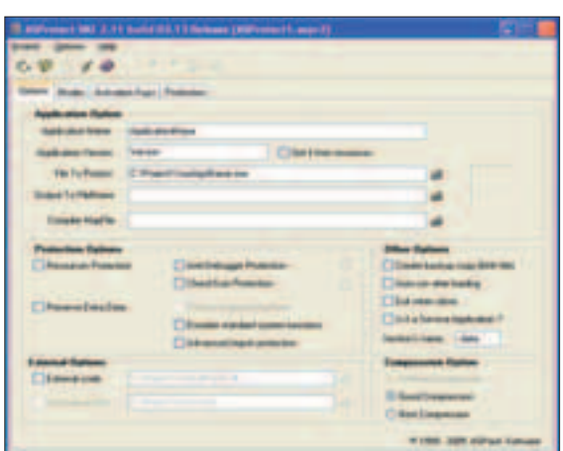
На нашем диске ты найдешь все необходимые исходники.

❻. **XProtector** - защита, основанная на применение драйвера, который очень усложняет распаковку. Недостатки этого протектора - скорость запуска приложения и очень низкая стабильность (часто перезагружает компьютер без видимых причин). Преимущества:
❻.❶. Применение драйвера.
❻.❼. Хорошая защита импорта.
❻.❽. Упаковка файла.
❻.❿. Защита кода программы с применением очень стойких криптоалгоритмов.

❼. **StarForce** - старая защита, написанная старыми реверс-инженерами России. В ней применяется драйвер защиты. В основном используется для защиты игр (без диска игра не расшифровывается). Но недавно появилась обычная защита (без привязки к диску), которую снимает очень низкий процент реверс-инженеров - эта защита для элиты. Однажды нашли ее очень серьезный недостаток: защита сжигает USB-приводы. Зато она может и порадовать:
❼.❶. Одно название защиты будит в душе реверс-инженера чувство безнадежности :).
❼.❼. Применяется хорошо отлаженный драйвер, который не дает подступиться к защите.
❼.❽. Применяется виртуальная машина, до которой практически не возможно добраться при попытке отладки.
❼.❿. Защита кода программы с применением очень стойких криптоалгоритмов.

Все преимущества этой защиты трудно перечислить, так как они мало кому известны :).

Самой перспективной (по моему мнению) защитой является AsProtect. Так как Алексей с каждым новым билдом добавляет что-то новое и очень неприятное, защита очень сильно "поднялась" со времен AsProtect 1.23. Рекомендую ее или Armadillo (не со всеми опциями, желательно без Copy Mem II, так как из-за нее очень сильно падает производительность программы, а толку с нее мало) всем, кто »



Главное окно AsProtect

действительно хочет защитить свой программный продукт.

ЗАЩИТА НА ПРАКТИКЕ

О! Это целое искусство. Существует множество аспектов, от которых зависит качество защиты приложения. Для каждого протектора они индивидуальны. Разберем их с несколькими из описанных протекторов.

❶. AsProtect

В главном окне AsProtect советую установить галочки: Resources Protection, Preserve Extra Data, Anti-Debugger Protection, CheckSum Protection, Protect Original EntryPoint(!), Emulate standard system functions(!), Advanced Import protection(!), Best Compression, Create backup copy (BAK-file). После этого указать путь к map-файлу. С этими опциями AsProtect максимально защитит программу и создаст резервную копию файла (она нужна - мало ли что случится). В следующих закладках выбирай сам: они не влияют на защиту приложения серьезно, так как отвечают за регистрационные ключи и trial. Я не советую использовать некоторые api-функции, предоставляемые AsProtect'ом: GetRegistrationInformation, так как после снятия AsProtect'а эта функция "эмулируется", то есть пишется код, который всегда возвращает, что программа зарегистрирована. Соответственно, твоя защита быстро нейтрализуется.

❷. Armadillo

С такими опциями, как показано на рисунке, защита будет на максимально возможном уровне. Но может случиться упадок производительности, если они покажутся слишком большими, и тогда передвинь шарик на одну позицию вниз. Служебные функции, предоставляемые защитой, использовать не советую (кроме функций проверки ключей). И вот почему. Однажды судьба вынудила меня взяться за взламывание программы, на которой "висела" Armadillo и в которой применялись функции проверки регистрации. Я решил сделать по-хитрому: написал dll-библиотеку, содержащую необходимые функции, которые возвращали мой ник, после чего программа стала зарегистрированной... Халява всегда такая ;-).

AHTeam UPX Mutanter прячет сигнатуру, заменяя ее другими

СКРЭМБЛЕРЫ

И вот мы добрались до долгожданных скрэмблеров. Я уже объяснил, что это такое и зачем оно нужно, но не сказал, что существуют разные скрэмблеры: одни просто исправляют файл для невозможности дальнейшей автораспаковки, а другие могут и добавлять свой код во имя практически полной невозможности восстановления автораспаковки.

Stealth PE "убивает" утилиты для взлома

Кроме Stealth PE, советую попробовать Hide PE от того же автора

Скрэмблер PeStubOEP от GPcH также широко известен

❶. AHTeam UPX Mutanter

Этот скрэмблер прячет сингатуру (устойчивую последовательность байтов с точки входа) UPX, заменяя ее различными другими, после чего UPX определяется как какой-либо протектор и добавляет небольшой расшифровщик. И в результате восстановить возможность автораспаковки трудно.

Возможность подмены сигнатур есть и в AHTeam EP Protector

❷. Stealth PE by BGCorp

Скрэмблер, написанный моим другом, является еще одной программой для подмены сингатуры. Программа, обработанная этим скрэмблером, после запуска "убивает" некоторые утилиты для взлома.

❸. AHTeam EP Protector

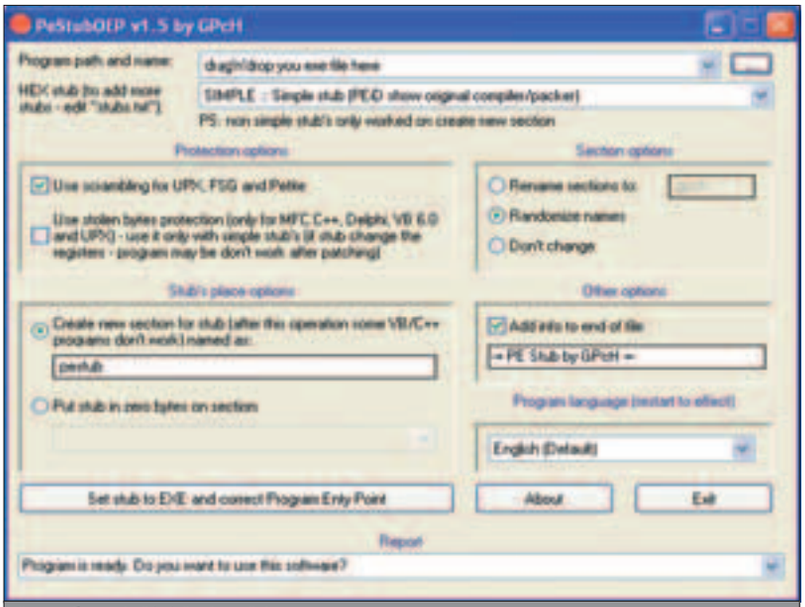
Тоже весьма неплохой скрэмблер. Умеет подменять сигнатуры и добавлять защиту от автоматических поисковиков оригинальной точки входа в программу.

❹. PeStubOEP

Программа предназначена для защиты EXE-файлов от определения их компилятора/упаковщика снифферами типа PEiD и PE Sniffer. Поддерживает практически все упаковщики, кроме FSG. Также, если EXE-файл не упакован и написан на MFC C++, Delphi 6.0/7.0 или Visual Basic 5.0/6.0, то PeStubOEP прячет в stub'е некоторые байты из OEP вперемешку с мусорными байтами, что усложняет восстановление OEP. Очень удобно, что программа ведет детальный лог своей работы.

❺ DotFix FakeSigner by GPcH

Очередная защита. Основные возможности:



Скрэмблер PeStubOEP от GPcH также широко известен

❼.❶. Добавление в код программы дополнительных кусков кода, затрудняющих трассировку и распаковку.

❼.❷. После обработки программы данным протектором снифферы скажут, что программа защищена довольно навороченным протектором. Хотя, возможно, скоро снифферы начнут определять данный протектор, но это никак не поможет крэкерам выявить реальный упаковщик.

❼.❸. При загрузке защищенных программ в некоторых дизассемблерах те начинают глючить и не дизассемблируют код.

❼.❹. Возможность изменения имен секций (как на одинаковые, так и на имена различных упаковщиков).

❼.❺. Специализированный скрэмблер UPX 0.6, UPX 1.24, UPX 1.9, FSG, Petite.

❼.❻. Программа ведет детальный лог своей работы.

❼.❼. Имеется возможность выбора внедряемой сигнатуры, обманывающей PEiD и подобные снифферы.

❼.❽. Присутствует также возможность выбора метода защиты.

❼.❾. Возможность шифровки точки входа и секции кода программы (только в зарегистрированной версии).

❼.❿. Есть превосходный генератор мусорных команд.

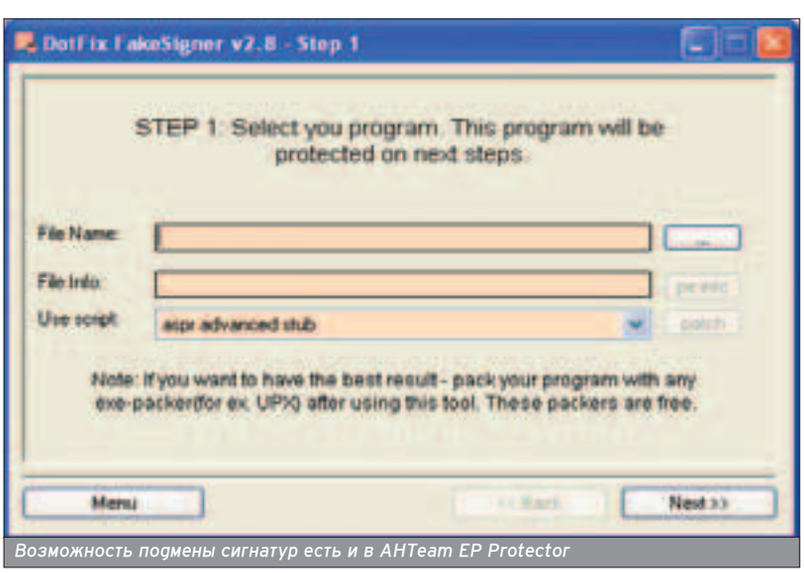
❼.⓫. Программа регистрируется в контекстном меню Windows. Теперь для более быстрой защиты программ щелкни по ним правой кнопкой и выбери пункт Protect with DotFix FakeSigner.

❼.⓬. DotFix уже имеет встроенный снифер. Отныне снифферы не нужны для определения, чем упакована программа - эту задачу может выполнять DotFix.

❼.⓭. DotFix позволяет узнать информацию из PE-заголовков EXE-файла, а также информацию о секциях.

❼.⓮. Защита от трассировки и корректная обработка или удаление Bound Import Directory.

Заканчивая обзор скрэмблеров, я могу посоветовать: Stealth PE, Morphine, AHTeam EP Protector. Дерзай.



Возможность подмены сигнатур есть и в AHTeam EP Protector

Chingachguk/HI-TECH (chingachguk@newmail.ru)

КЛЮЧИК К СЕРДЦУ

ВСЕ ОБ АППАРАТНЫХ КЛЮЧАХ ЗАЩИТЫ

Существует множество аппаратных ключей защиты. По заявлениям производителей, "с помощью ключей HASP защищено более 2/3 коммерческих программ, разработанных и продаваемых на территории СНГ и стран Балтии". Тем не менее, и этот способ защиты имеет ряд существенных недостатков, которыми можно воспользоваться.

ГОНКА ВООРУЖЕНИЙ

Итак, dongle (ключ), компонента многих защит, используется, как правило, для защиты от незаконного использования программ, но также может (по утверждениям разработчиков) использоваться для авторизации/защиты баз данных и т.п. Втыкается в LPT-порт (это уже в прошлом) или в USB. Для нормальной работы может потребоваться установка дополнительных драйверов, не связанных с комплексом защиты. В общем и целом dongle представляет собой часть алгоритма защиты, вынесенную в аппаратную область, проще говоря - в "железо". Сейчас объясню почему. Разработчики защит уже давно приметили, что уровень исследования исполняемых кодов защищенных программ страшно возрос (его неуклонный рост начался еще во времена написания первых DOS-вирусов и крэков) и невозможно дать гарантий на то, что взломщик не сможет полностью изучить код защиты. Полное исследование проведено, исходный код программы фактически имеется перед глазами - крэка при таких делах не напишет только ангел небесный. Разработчики защит изощрялись и порождали на свет все новые и новые приемы, призванные помешать взломщику: антиотладочные процедуры, расшифровываемый при запуске код и многие другие техники. Но и в далеком прошлом, и сейчас на исследование кода защиты требуется намного меньше времени, чем на написание хорошей защиты.

Почему так? Допустим, обычный программист решает написать, как ему кажется, "защиту" и встраивает в свой код диалоговое окно ввода ключевой информации, а потом проверяет валидность введенной информации. Большинству таких программистов среди сотен килобайт исполняемого кода сложно найти собственную ошибку, даже обладая исходниками, поэтому им опять-таки только кажется, что невозможно найти то самое место, в котором проверяется валидность ключевой информации:

```
{ "Прячем" настоящий пароль, чтобы его не нашли по
                                F4 в far/total/etc }
Str(123,Key1);
Str(691,Key2);
MyBestPassword:=Key1+Key2+... { И другой подобный
                                код ... }
if MyBestPassword <> UserPassword then ...
```

Однако время обнаружения такой процедуры ничтожно мало, например десять минут. Реверсер установит после ввода абстрактного ключа активации breakpoint на доступ к нему из любого кода и, разумеется, тут же увидит использование введенной информации и критичный код. Конечно, разработчик защиты может пойти дальше. Допустим, он попытается вырвать основной вид оружия из рук реверсера - отладчик SoftIce, которым тот может выполнить вышеописанные действия:

USB-ключи - одни из самых популярных в мире

Несмотря на популярность USB-устройств, до сих пор встречаются и LPT-ключи

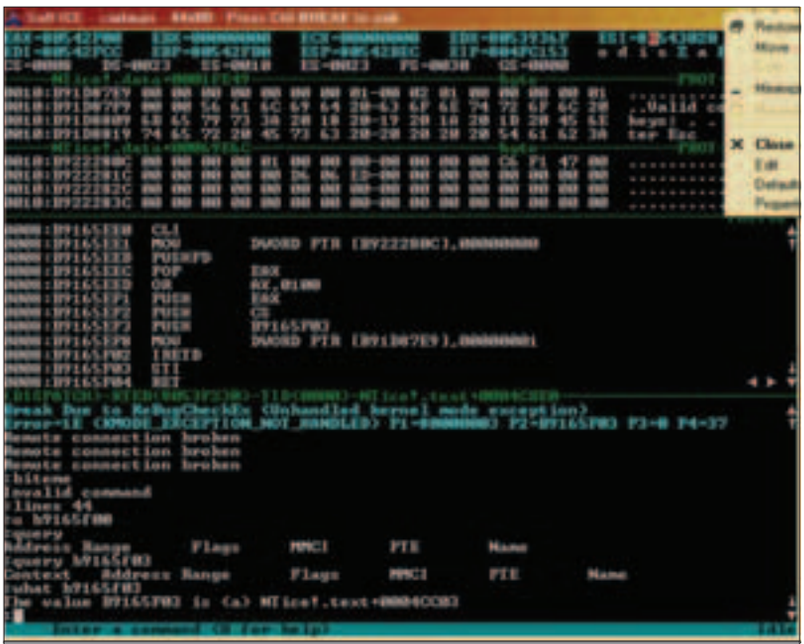
```
// Открываем устройство "NTICE", если это удалось,
                                //то отладчик SoftIce
                                // установлен в системе
if ( open("\\\\.\\NTICE"...) >= 0 )
{
printf("\n Unload SoftIce ! Program aborted !");
ExitProcess(...);
}
```

Такой код, будучи размещенным в начале программы, прервет ее выполнение еще до ввода ключа, и реверсер не сможет выполнить анализ кода. Но действительно ли не сможет? Такую попытку защиты легко отследить еще перед запуском исследуемой программы, установив в отладчике слежение за открытием файлов:

```
bpx CreateFileA if (*(esp+4)==0x5C2E5C5C).
```

Выполнив эту команду, реверсер установил в отладчике SoftIce слежение (всплытие отладчика по настроенному событию) за открытием файлов, имена которых начинаются с "\\.\". Скорее всего, первый и единственный вызов с таким именем файла со стороны защиты и будет происходить в коде, который "проверяет" наличие отладчика в системе.

Словом, между разработчиками и реверсерами издревле существует своего рода гонка вооружений. Несмотря на переменные успехи то одной, то другой стороны, ясно одно: при примерно одинаковом интеллектуальном потенциале и терпении у защиты и реверсера устанавливаются приблизительно равные шансы, но при неограниченном времени у последнего возникает практически 100% вероятность вскрытия кода! А это уже заставляет серьезно задуматься. К тому же разработчик никогда не может быть уверен в том, что он не допустил досадного промаха при проектировании/реализации защиты, а самое главное: его код, написанный раз и навсегда, в любой момент может быть атакован совершенно новыми техниками реверсинга/аппаратными возможностями процессора или запуском в той же виртуальной машине.



Обмануть легендарный SoftIce не так-то легко

Российская компания Aladdin – один из крупнейших разработчиков аппаратных ключей

Вынося же часть алгоритма в "черный ящик" аппаратного ключа, он может быть уверен хоть в чем-то, потому что исследовать начинку dongle'а намного сложнее, чем обычный код. Раз так, подробнее поговорим об аппаратных ключах защиты.

МАРКЕТИНГ РЕШАЕТ ВСЕ

Прежде чем приступить к рассказу о внутренностях ключей защиты, разрешим вопрос о том, что именно подталкивает разработчиков защит к их выбору? Основная причина очевидна – простота. У разработчика появляется возможность не писать защиту самостоятельно, не будучи профессионалом в этой области, а просто купить "коробку", которая вроде бы несложным образом подключится к его программе и защитит это детище. За ведущими разработчиками ключей стоит специализация в области защиты и отличная репутация как результат хорошей маркетинговой политики. Именно по этим причинам рынок аппаратных ключей защиты расширяется не по дням, а по часам. Разработчикам ключей относительно несложно привести "весомые" доказательства неуязвимости своих товаров, так как полный анализ всего комплекса защиты, как правило, занимает довольно значительное время и вряд ли будет выполнен в момент презентаций. Декларируемые разработчиком свойства ключа (функциональность) даже в плане ознакомления требуют от программиста значительного времени, что уж говорить о серьезном криптоанализе функций ключа. Производители ключей предлагают не просто "железку" - они предлагают полный комплект разработчика защиты, включающий в себя удобный интегратор защиты с защищаемой программой, хорошо документированное API защиты, техподдержку и т.п. При таких условиях сделать выбор в их пользу не так уж и трудно. Тем не менее, и здесь не все так просто.

Многие разработчики софта также "ведутся" на "коробочные" решения

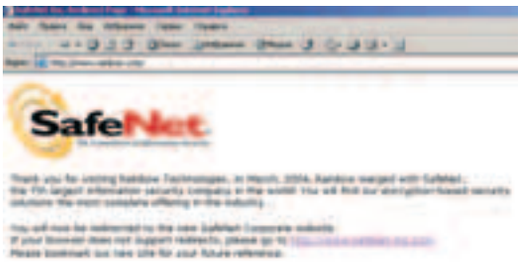
Эволюция аппаратных ключей eToken

Весьма популярный ключ Sentinel от Rainbow Technologies

$30 – И ВСЯ ЗАЩИТА!

Перейдем к построению общей схемы алгоритмов и компонент защищенного программного продукта. Начнем с защищаемого кода. Пусть имеется "нетронутый" защитой build release разработчика. Назовем его prog.exe. Его необходимо защитить аппаратным ключом таким образом, чтобы было очень сложно (а лучше невозможно) использовать функциональность prog.exe в отсутствие клю- »

По мнению разработчиков dongle'ов, USB-ключ должен присутствовать в связке с ключами от квартиры, машины...



Около года назад крупный разработчик ключей, компания Rainbow Technologies, вошла в состав SafeNet

ча. Простые подпрограммы проверки наличия ключа сведут на нет все достоинства аппаратной защиты: при их обнаружении и удалении они оставят реверсеру prog.exe в первозданном незащищенном виде, поэтому интеграция проверок с телом prog.exe должна быть максимальной. Например, самое простое - число проверок должно стремиться к максимально возможному, насколько это позволяет быстродействие обмена данными с ключом из prog.exe, но разработчики обычно не знают меры в этом и, по всей вероятности, используют простую формулу "Время торможения программы из-за присутствия аппаратной защиты = 2 x максимальное время терпения пользователя" ;-).

Идем дальше. Максимальным количеством проверок дело обычно не ограничивается, хотя история взломов программ, защищенных dongle'ами, знает множество примеров того, как разработчик выполнил... Две-три тривиальные проверки наличия ключа, которые по сложности взлома в мириады раз уступают качественно спроектированной защите на основе обычного кода активации. Здесь используются техники, присущие всяким программам, в которых необходимо скрывать код/алгоритмы - от троянов и вирусов до хитрых математических библиотек. Такие техники изначально совершенствовались вирусописателями, например в полиморфных вирусах. Но в наше время разработчики защит вполне уверенно используют их, нисколько не заботясь о чужом (пусть и незарегистрированном) копирайте. Итак, в код prog.exe максимально плотно внедряются алгоритмы проверки наличия и валидности ключа. Необходимо отметить криптографические методы привязки ключа к prog.exe. Допустим, ключ может аппаратно реализовывать функцию:

```
typedef struct {
...
} tSecrectKey;

tSecrectKey Get_SecretChiperKeyFromDongle( void ) {
...
}
```

Получив в программе по запросу к ключу SecrectKey, программист может выполнить шифровку/дешифровку собственного кода. Например у prog.exe есть еще secret.dll, в которой находятся важные функции. Разработчик выполняет шифрование тела secret.dll неким хорошим криптографическим алгоритмом:

```
Chiper.DLL = Encrypt(secret.dll, SecretKey)
```

Далее в своей программе prog.exe он предусматривает, что при ее запуске произойдет расшифровка критичного кода:

```
secret.dll = Decrypt(Chiper.DLL, SecretKey)
```

Разумеется, SecretKey не содержится внутри prog.exe - она может взять его только из аппаратного ключа, если выбранный алгоритм Encrypt/Decrypt стойкий. А его возможно выполнить из широкого спектра известных и проверенных алгоритмов - DES, AES, BlowFish... или даже рискнуть написать свой, особенно если жалко денег на лицензионный и криптографически вроде бы неуязвимый.

Это хороший, очень хороший ход разработчика защиты, но он становится уязвимым при одной плохой вещи: любой хакер, решившийся на покупку одного ключа (стоимость порядка $30), получит из него SecretKey и сможет расшифровывать любые программные продукты, защищенные этим ключом. Поэтому разработчики идут на различные хитрости в этом направлении: используют случайные запросы (данные для расшифровки выбираются случайным образом из достаточно большого списка, но при этом реверсер может исследовать код программы и найти весь список); запросы, сдвинутые во времени (допустим, в каждом следующем месяце программа обновляет данные для расшифровки и вроде бы рабочий крэк перестает работать); шифрование вводимых данных (допустим, шифрование записей базы данных или передача шифровального ключа SQL-серверу, поддерживающему защиту данных) и т.п. Словом, здесь используются все те же техники, которые можно встретить в "обычных" защитах или же вирусах. Однако следует отметить тот факт, что разработчики защит на основе аппаратных ключей отличаются особой паранойей в этой области: фактически их код стремится включить в себя максимальное число таких "фишек", благо user-код (код приложения win32, например) не ограничен по времени выполнения, в отличие от критичного кода драйвера, и, пока защита расшифрует свое тело раз пять-десять, пользователь может спокойно посидеть в интернете и т.п. По некоторым данным, аналогичные разработки "параноидальных" защитников для *nix-систем содержат ... существенно меньше таких приемов, и некоторые реверсеры вполне успешно сокращают время анализа защиты просто просматривая примеры реализации для этих систем.

Продукты Guardant также хорошо знакомы российскому пользователю

Известная компания StarForce имеет свой взгляд на аппаратную защиту

HASP-КЛЮЧ

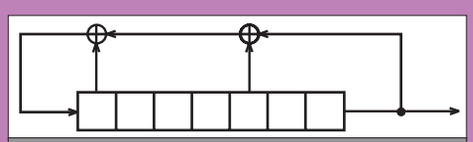
Перейдем к следующему критичному участку защиты - обмену данными между prog.exe и драйвером защиты driver.sys. Почему обмен данными не может быть выполнен напрямую с ключом из prog.exe? По многим причинам, например по такой: времена DOS давно прошли, и выполнять машинные команды для общения с устройствами (ins/outs) из кода уровня приложения просто невозможно. К тому же в более-менее серьезной операционной системе весь обмен с внешними устройствами реализуется только в драйверах (уровни абстракции), и она не только запрещает обращение к устройствам из user-кода (NT-системы), но и часто предоставляет весьма удобный механизм перехвата таких обращений из user-кода (9x-системы), что может существенно облегчить работу реверсера.

Обмен данными между prog.exe и driver.sys также важно защищать от перехвата и исследования. Почему? Потому что если, например, число проверок наличия ключа в prog.exe 1000 штук, но все они реализуют обмен с ключом одинаковым способом, например через стандартный механизм из Win API DeviceIoControl, то гораздо легче перехватить такой единственный канал обмена вместо выре-

ПРИМЕР СЕКРЕТНОЙ ФУНКЦИИ

В большинстве случаев в первую очередь следует обращать внимание на простейшие криптографические генераторы псевдослучайных последовательностей. Чаще всего применяются конгруэнтные генераторы (видимо, они уже ушли в прошлое) и LFSR:

```
; Конгруэнтный генератор имеет вид:
Gi+1=(a*Gi+c) mod m

; LFSR может иметь следующий вид:
UINT LFSR, NextHighBit;

NextHighBit= ((LFSR>>t1)^(LFSR>>t2)^(LFSR>>t3)...)&0x01;
// Шаг LFSR
LFSR= (LFSR>>1)|(NextHighBitgt<<N);
; LFSR может также иметь такой вид:
UINT LFSR, TAPS;

// Шаг LFSR
LFSR^= 1;
LFSR= (LFSR>>1)|(LFSR<<N);
if (LFSR&(0x01<<N)) LFSR^= TAPS;
```



Схематичное изображение работы 7-разрядного LFSR с отводами с битов 2 и 6

LFSR - один из самых распространенных элементов как в криптографии, так и в криптодевайсах. Он легко реализуется аппаратно, поэтому, видимо, часто используется в сравнительно недорогих устройствах (ключи стоят десятки долларов, а сами чипы - всего несколько зеленых). N - разрядность регистра сдвига (IfSR) в битах минус 1 (допустим, для восьмиразрядного регистра сдвига N=7), t1..tk или TAPS для второго типа LFSR - "тапы" или отводы битов из LFSR в некотором состоянии. Допустим, для первого случая t1=0 и t2=2, что означает, что на каждом шаге берут биты 0 и 2 из LFSR. Их результат ксорится и заносится в старший бит LFSR, сам же он предварительно сдвигается вправо (непосредственно бит 0 теряется). При правильном выборе TAPS или t1..tk LFSR с произвольным начальным заполнением проходит порядка 2^(N+1)-1 состояний (для N=7 это 2^8-1=255). Его состояния - псевдослучайные числа, которые можно использовать в криптографических алгоритмах. Не хочу сказать, что всякий ключ состоит из одного такого элемента: как правило, это каскад из LFSR'ов.

Даже если алгоритм "секретной функции" ключа известен, праздновать победу еще рано. Существует также подзадача определения секретных параметров алгоритма (фактически - ключей шифрования), которые не могут быть считаны из ключа иначе как изучением прошивки или же криптоанализом запросов-ответов секретной функции. Эта задача менее сложная, чем определение секретной функции ключа, но только до того момента, пока разработчик ключа не внесет внутрь ключа какой-нибудь надежный криптоалгоритм, например DES. В этом случае он даже может декларировать свою "секретную функцию" публично: ему незачем скрывать проверенный алгоритм! Однако ключ шифрования для DES ему лучше скрыть.

зания 1000 разбросанных по prog.exe подпрограмм проверки. Здесь защита наиболее уязвима (от перехвата обмена данными с ключом), но здесь же реверсер начинает отступать от "идеального взлома", заключающегося в полной очистке кода prog.exe от всякого присутствия защиты в нем.

Разработчики защиты прекрасно осведомлены о свойствах канала DeviceIoControl, но продолжают использовать его по ряду причин: техника перехвата API все еще никак не переедет из учебника хакера за второй класс в букварь, и нормально построенное шифрование на основе данных ключа некритично к прослушиванию (советую почитать про криптографические протоколы, стойкие к "атаке посередине"). Такой обмен может на некоторое время отвлечь реверсера от неких дополнительных скрытых каналов обмена. Какие же дополнительные каналы обмена может открыть защита для общения со своим драйвером?

Рассмотрим один из самых распространенных аппаратных ключей - HASP (Hardware Against Software Piracy). Защитный комплекс (его обычно называют "конверт") включает в себя, помимо защищенного prog.exe и нескольких драйверов защиты driver1,2..sys и ключа HASP, еще и дополнительный канал обмена через исключение UD #6 - Undefined Operation. Это означает, что в коде prog.exe защита размещает особым образом оформленные команды, которые вызывают в процессоре исключение недопустимой операции (процессор, работающий в защищенном режиме, способен реализовывать так называемые уровни защиты и при выполнении некорректных инструкций в менее привилегированном коде (User-level) немедленно передавать управление в более привилегированный (ядро OS). Таким образом, в определенный момент управление в prog.exe будет сознательно передано на такие команды, процессор передаст управление в ядро системы, но не просто системному менеджеру (в этом случае не произойдет ничего интересного - программа просто будет принудительно завершена системой с сообщением "Программа выполнила..."), а специально подстроившемуся под это драйверу защиты: в момент своей инсталляции он заранее документированными или не очень способами подготовит систему так, что он будет первым, то есть до менеджера системы, обрабатывать такое исключение. Он (драйвер защиты) может позволить себе это, поскольку в NT-системах драйверы имеют все привилегии, в том числе и такую вот возможность обрабатывать все проблемные ситуации в системе. Фактически драйвер защиты равен в правах с ядром системы и может использовать в своих целях всю функциональность процессора. Нужно заметить, что инструкция UD использовалась таким вот "нехорошим" способом не только этой защитой, но и ... собственно разработчиками из MS. Такой канал используется для прямого вызова ядра в NT, а в 98-ой он приведет к обычному сообщению "Программа выполнила...".

На таком канале строится что-то вроде собственного API-ключа. Это достигается тем, что команда UD специальным образом оформляется для »

Уже появилось пятое поколение HASP-ключей

того, чтобы обработчик исключения внутри драйвера мог отличить "свой" код от обычных сбоев, возможно, в чужих программах. Это может быть сделано, например, следующим способом (почти оригинальная цитата):

```
mov eax,40000007h ; Номер функции API заносится в
                  ; регистр eax
mov edi,offset DATA_FOR_KEY ; Адрес данных
                  ; для передачи в драйвер
...
UD ; Инструкция, вызывающая исключение -
                  ; см. документацию opcodes Intel
db 'DEADBEEFBABY'
Next:
```

Обработчик исключения в драйвере проверит наличие тестовой строки после команды UD и значения регистров. Если он признает исключение "своим", он обработает его соответственно: например, запишет данные из ключа по указателю в edi и передаст управление на метку Next.

Таким образом, мы рассмотрели один из возможных каналов обмена между защищенным приложением и драйвером защиты, работающим на уровне ядра. Их реализация может быть и иной, но, по крайней мере, при анализе следует проверять "свободные" исключения, которые обычно мало используются системой: это устаревшая команда bound (int 5) и некоторые другие. Следует немного ознакомиться с процессором Intel в защищенном режиме и в целом представлять, какие возможности могут быть у защиты для открытия каналов общения с драйвером. Напоследок брошу еще один камешек в огород разработчиков такой защиты. Совсем не удивительно, что процессор тратит длительное время на обработку исключений, а данная защита использует... тысячи вызовов такого типа. Поэтому обращаюсь к разработчикам и говорю: "Не думайте, что Ваши программы просто плохо написаны или даже что они очень умные - просто дайте защите немного потренироваться". Однако стоит отметить, что для следующей версии ключа - HASP4 - разработчики отказались от этой "замечательной" техники и остановились на "обычном" DeviceIoControl.

ДРАЙВЕР НЕ СПРЯЧЕШЬ

■ Перейдем на следующий уровень комплекса защиты и рассмотрим, что происходит в драйвере защиты. Для начала необходимо определить их наличие. Наверное, из стыда защитникам хотелось бы скрыть от пользователя наличие драйвера, но система им этого особо не позволяет, поэтому меню "Установка драйвера защиты" действительно означает то, что в нем указано. Список драйверов можно посмотреть в стандартных настройках Win, а детально рассмотреть его, например, при помощи команды "driver <имя_драйвера>" в SoftIce.

Драйвер выполняет не только функции передачи данных (запросов) из приложения и ответов ключа приложению (транспорт), но и часто содержит в себе часть функциональности, предоставляемой ключом. Так называемые "API ключа" часто разбиваются на две части: обертка реализована внутри драйвера, а нижние функции выполняет сам ключ. Для примера опишу следующую ситуацию. Пусть ключ способен выполнять шифрование четырехбайтных слов:

```
void Key_EncryptData(DWORD *EntryData)
{
...
*EntryData= ...
}
```

Следовательно, ключ может шифровать четырехбайтные слова неким своим секретным алгоритмом, детали которого или же ключи шифрования которого скрыты внутри его микросхем. Однако драйвер защиты и описание API для разработчика содержит обобщенную функцию:

```
void Driver_EncryptData(void
*EntryDataAboveOrEqualThan4Bytes, DWORD DataLen)
{
...
memcpy(EntryDataAboveOrEqualThan4Bytes, ... ,
DataLen);
}
```

Видно, что функция Driver_EncryptData может выполнять шифрование массива байт, длина которого может превышать четыре байта. Это может быть сделано по нескольким соображениям: разработчику не хватает шифрования двойных слов; функция ключа Key_EncryptData недостаточно криптоустойчива, и драйвер пытается усилить мощность защиты; проектировщик защиты решил укрепить бастионы на пути к оригинальной функциональности ключа и некоторым другим.

Также драйвер обычно пытается всеми доступными ему способами скрыть все свои алгоритмы, а в особенности - нижние процедуры типа Driver_EncryptData, содержащие внутренние криптографические алгоритмы и сам обмен с ключом. Обмен с ключом может быть реализован непосредственно в драйвере (например LPT-реализация ключа) или же обычными системными драйверами (в случае с USB-вариантом работать с ключом напрямую через порты ввода-вывода и управлять DMA было бы крайне нерационально: пришлось бы размещать внутри драйвера защиты функциональность, достаточную для совместимости с различными спецификациями USB-шины и т.п.). Собственно, обмен напрямую с LPT возможен только лишь потому, что он уже полностью стабилен и прост в реализации, но "не на бумаге" все оказалось сложнее: ключ был спроектирован как "прозрачный" для обмена с принтером, тем не менее были случаи сбоев в работе принтера, разделяющего этот ресурс.

Защита драйвера также может носить "параноидальный" характер, но при этом обладающий своими особенностями: акцент в ней сделан на защиту не от отладки, а от дизассемблирования (те вытаскивания критичных алгоритмов), и, в случае реализации обмена с ключом непосредственно в драйвере, защищается обмен: это скрытие от перехвата команд in/out.

Рассмотрим подробнее эти два аспекта защиты. В защите от дизассемблирования может применяться так называемая техника "обфускации". Исходный код драйвера, написанный, допустим, на C, транслируется в ассемблерный код:

```
void Driver_EncryptData(void
*EntryDataAboveOrEqualThan4Bytes, DWORD DataLen)
{
...
KeyStruct->Field7= MAG_CONST_FOR_FIELD7;
}
// После трансляции в ассемблер:
; KeyStruct->Field7= MAG_CONST_FOR_FIELD7;
mov dword ptr [ebx].Field7, 0xABCDEF77;
```

Далее ассемблерный код подается на вход некоей утилите, которая выполняет многократное (с точки зрения машинной реализации) усложнение кода на ассемблере:

```
// После "обфускатора":
;; KeyStruct->Field7= MAG_CONST_FOR_FIELD7; // СИ
; mov dword ptr [ebx].Field7, 0xABCDEF77; // ASM first
variant
push const1
push eax
moveax,const2
jmp@Label1
...
@Label1:
xor[esp+8],eax
push ebx
movebx,const3
jmp@Label2
...
@Label2:
sub[esp+0Ch],ebx
popebx
popeax
jmp@Label3
...
@Label3:
popdword ptr [ebx].Field7
```

В результате в поле KeyStruct->Field7 попадает значение (const1^const2)-const3, причем все константы подобраны таким образом, что в результате (const1^const2)-const3 == MAG_CONST_FOR_FIELD7. Понятно, что такой код читать намного сложнее, чем одну ассемблерную команду. Данная техника крайне многообразна: например, критичные поля могут разбиваться на части, за ними

Пример "обфусканного" кода (модуль hardlock.vxd)

следует множество команд типа pushfd/popfd или xchg ebp,ebp, ничего полезного не делающих, однако выполнение алгоритма от этого не изменяется.

ТОНКОСТИ КУХНИ

Теперь о защите от перехвата обмена с ключом на самом нижнем уровне. Если в драйвере используются команды in/out, они охраняются от перехвата всеми возможными способами. Почему? Если реверсер может эмулировать ответы ключа или просто запомнить их на уровне in/out-команд, то, перехватив работу защиты с портами, он сможет удобным для себя способом разместить свой эмулирующий код сразу после драйвера защиты перед отсутствующим ключом. Перехват in/out в NT-системах на уровне ядра, то есть при котором перехватывающий и перехватываемый код находятся в равном положении, возможен только (это, скорее, тема отдельного разговора) через регистры отладки DRx. Если бы драйвер защиты находился на меньшем уровне привилегий или OS Windows использовала бы механизм вложенных задач (Nested Task), то было бы возможно также использовать битовую карту ввода-вывода - permission i/o access map. Защитить код от мониторинга портов ввода-вывода через регистры DRx в этом случае возможно только сбросом бита DE в регистре CR4 процессора, что может с успехом применить защита. Сброс же регистров отладки DRx, нередко используемый в защитах, имеет фатальный недостаток: доступ к этим регистрам также может быть перехвачен через те же регистры DRx, и код перехватчика полностью прозрачен для защиты. А недостаток этого способа эмуляции (он же - преимущество для защиты) состоит в необходимости для такого эмулятора разделять общие ресурсы с отладчиком - прерывание int1, которое может сильно затруднить отладку эмулятора или его использование.

ОСОБЕННОСТИ РЕАЛИЗАЦИИ КЛЮЧЕЙ

Рассмотрим детальнее сами ключи и функциональность, которую они могут предоставить разработчику защиты. Скорее всего, все они имеют пароли доступа, необходимые для того, чтобы реверсер, не зная их, не мог получить секретные поля ключа или изменить их (пока используется редко). В других случаях пароли доступа носят чисто факультативный характер для идентификации кода защиты в ключе и могут даже служить дополнительной информацией для... получения секретных данных ключа (те секретные данные ключа = f(password)). Последнее, как правило, объясняется тем, что устройство ключей не может быть очень сложным (пока, в дальнейшем их функциональность может быть существенно доработана), с другой стороны, ключи одной серии должны поддерживать различные программные продукты. Как правило, ключ может содержать память (~100-2000 байт), доступную как для чтения, так и для записи. Чтение такой памяти может дать защите возможность получения неких уникальных идентификаторов для шифрования или их простой проверки. Запись в память и последующая проверка правильности записи может расширить возможности защиты: например, при выключении питания компьютера эмулятор должен уметь сохранять такие данные на диске, иначе эмуляция будет неполной. Однако такие функции сравнительно легко эмулируются: реверсер аккуратно выясняет спецификацию функций чтения-записи, получает все возможные значения памяти ключа и запоминает их в эмуляторе для совершения корректных табличных ответов на запросы защиты.

Существуют также такие специфические функции ключа, как проверка статуса (~IsKeyHere) и таймер ключа. Обычно они идут первыми в снятых логах. Защита проверяет работу таймера ключа, которую сложно повторить в эмуляторе и т.п. Однако стержнем многих ключей является некая "секретная функция", каких всего одна-две из всего набора функций ключа (для мощных ключей порядка десятков).

Общий вид секретной функции может быть записан так:

```
OutData Key_TransData(? *EntryData)
{
...
}
```

Функция Key_TransData выполняет секретное преобразование некоторого количества EntryData->OutData. Что означает "секретное"? Реверсер может добраться до самого нижнего уровня обмена с ключом, но увидит только работу этой функции. Вся ее логика зашита в ключе, алгоритм считается абсолютно не известным с точностью до анализа прошивки. Пищей для криптоанализа является неограниченное количество запросов-ответов (EntryData, OutData), его методы - все методы криптоанализа и поиски "случайно оставленных" элементов функции Key_TransData в... коде защиты. Да, это совершенно реальный случай. Один из ключей был "сломан" именно так: внутри одного из драйверов защиты находился особо не прикрытый код Key_TransData...

ПОЧЕМУ КЛЮЧИ ЛОМАЮТ, И ПОЧЕМУ ОНИ ЛОМАЮТСЯ?

Все, что идет после этого заголовка, посвящено скорее социальным, чем техническим сторонам вопроса. Почему как у нас, так и в других странах одни из самых мощных защитных комплексов подвергаются постоянным и даже успешным атакам реверсеров? Не знаю, что движет товарищами Митника, кроме денег, конечно, но такая мощная защита используется более дорогими и функциональными программами. А у нас после развала СССР осталось довольно много технически грамотных специалистов, причем даже неважно, каким образованием они обладали: как раз кибернетика (это только мое мнение!) у нас была развита слабо, но пищи для ума нашему забугорному брату часто не хватает, даже несмотря на то, что многие действительно мощные реверсеры легко зарабатывают себе ~$1000-3000 в "обычной" конторе. Но, как правило, наши работодатели настолько stupid, что не в состоянии использовать потенциал гигантов кибернетической мысли, и для некоторых из них реверсинг - это просто хобби.

Почему же взлом ключей так успешен? Здесь, прежде всего, нужно упомянуть "плавный" переход от ломания простых ключей с элементарными алгоритмами еще в 90-х к более сложным. Разработчики ключей как бы учились вместе с реверсерами и заодно породили хорошую "школу" анализа различных аспектов программных защит. Также стоит отметить соотношение цены ключа к цене взлома алгоритма: если ключ продается по $20, то защита программы стоимостью в $500 с его помощью вряд ли гарантирует успех: стоимость программы настолько велика, что вызывает массовый приток усилий исследователей к этому ключу, но ключ, в который вложили $20, скорее всего, не может обеспечить стойкость стоимостью $500.

Content:

SPECial delivery

Симонов Илья aka Shturmovik (nazi@ghOsts.org)

CRACK-РЕСУРСЫ

ОБЗОР САЙТОВ ПО ВЗЛОМУ И ЗАЩИТЕ ПРОГРАММ

У меешь ли ты держать в руках дизассемблер или же не собираешься делать этого вовсе, неважно. Если ты пользуешься платными программами, то наверняка пользовался и таблетками к ним. А о том, где находятся архивы этих лекарств, где можно узнать рецепты и биографии докторов, пойдет речь в данной статье.

Ресурсов действительно очень много, поэтому чур не обижаться, если твой любимый или собственный ресурс не попал на эти страницы. Поскольку crack-ресурсы - это все сайты, содержание которых имеет окрас взлома программ, их можно разделить на сайты архивов крэков, инструментов крэкера, архивов статей и журналов по крекингу и реверсу, а также сайты всевозможных крэкерских группировок. Теперь все по порядку. Сейчас будет парад, пожалуй, самых посещаемых сайтов Сети. В последнее время людям нравится объединяться в группы. Что ж? Похвально, главное - чтобы оно того стоило.

АРХИВ ЛЕКАРСТВ

Множество людей пишут крэки, которые разными путями доставляются в пункты сбора, представляющие собой удобное хранилище лекарств от платных программ. Иногда пугает только то, что в таких местах вместе с крэками (а иногда вначале вместо них) подсовывают то, на что привык визжать, как резаный поросенок, антивирус Касперского. Совет один: искать крэки на официальных сайтах людей, выпустивших их. В первую очередь, из этих архивов хотелось бы отметить поисковик крэков, просто знаменитейший во всем интернете - http://astalavista.box.sk. Его глаза уже давно направлены на выдающиеся архивы таблеток.

Из самих же архивов нельзя пройти мимо www.cracks.am. Помимо pop-up'ов и вирусов, на сайте несколько десятков тысяч наименований с удобной сортировкой и возможностью самостоятельно добавлять новое средство. Далее на пути мы видим сайт наших заокеанских друзей www.keygen.us. Вирусов тут ничуть не меньше, однако, в отличие от простого упорядоченного списка, есть возможность скачать как все сразу, так и удобные базы данных по времени. И, наконец, третий ключевой сайт (остальные, кстати, ты найдешь в разделах Links и friends) - это crackdb.com, на котором есть хорошее разделение игровых и программных таблеток. Ну, наконец-то ты решил перестать бездельничать и пользоваться чужими средствами. И начать писать свое, для чего понадобятся кое-какие инструменты.

АРХИВ ИНСТРУМЕНТОВ

Конечно, ты можешь найти все необходимые средства на сайтах произ-

astalavista.box.sk – знаменитый поисковик крэков

На www.keygen.us есть возможность скачать удобные базы данных, отсортированные по времени

Армянские крэкеры не отстают: на cracks.am ты можешь сам добавить новое средство

На crackdb.com отличное разделение игровых и программных таблеток

водителей, а также я уверю, что любой крэк-портал, как и личный/командный сайт, содержит в себе с десяток инструментов, но по сравнению с www.exetools.net все остальное - цветочки.

Как ясно по его названию, сайт содержит в себе набор утилит для работы с EXE-файлами. Туда входят как упаковщики с протекторами, так и распаковщики, отладчики, дизассемблеры и многое другое. Самое замечательное в сайте - это его FTP-сервер, где находятся последние новинки, приватные программы и просто то, что очень непросто найти в Сети. Единственная трудность - это пароль от FTP, который сейчас выдается за сообщения на форуме. Но не все так просто: чтобы зарегистрироваться на форуме, нужно приглашение модератора. Вот так вот. Ну что делать? Втирайся в доверие. Большие архивы нового софта для взломщика содержат также различные сайты из домена cjb.net, и, как это ни странно, отличная подборка программ для разработчика (читай – для крэкера :)) находится на www.msdn.microsoft.com. Яркий тому пример - портал www.wasm.ru, где находятся не только инструменты, но и статьи, да и вообще там очень много чего находится. Советую посетить всем. Инструменты получили - дело за инструкцией.

CRACK-ПОРТАЛЫ

Пройти мимо www.crack-lab.ru невозможно. Это самый большой портал, посвященный исследованию защит программ в рунете, да и, пожалуй, не на последнем месте в мире. В буквальном смысле слова здесь есть все! Более сотни авторских статей, посвященных методам взлома и защиты программ, сухая теория и реальная практика. Огромная подборка крэкерского софта, в довесок ко всему на сайте можно заказать Crackl@b DVD крэкера, на котором (я тебя уверяю!) найдешь то, о чем мечтал всю жизнь (или еще помечтаешь). Cracklab.ru - это еще и огромный форум, где собираются наряду с новичками выдающиеся личности крэк-сцены, интервью с которыми можно также прочесть на сайте. Множество линков на другие ресурсы Сети, которые необходимо посетить в первую очередь после посещения Cracklab.ru. Также на сайте есть FAQ для новичков, книги по ассемблеру, справочник по WinAPI и многое другое. Другими словами, для начинающего крэкера этот сайт должен стать стартовой страницей в браузере. Подобных сайтов действительно много по всему интернету, но такой только один, и легко убедиться, что это действительно крэкерская лаборатория рунета.

Однако не будем забывать и о www.wasm.ru, на котором есть целый раздел статей, посвященных взлому, и не забудем прочесть "Теоретические основы crack'инга".

И конечно же, сайт живого дизассемблера наших дней Криса Касперски - www.kpnc.opennet.ru, где он выкладывает свои книги, выхода которых многие ждут с нетерпением.

Еще советую посетить англоязычный сайт http://bio-cyborg.cjb.net - аналог нашего Cracklab.

CRACKTEAMS

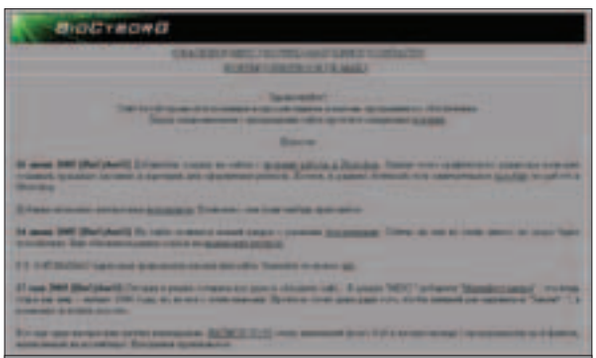
И переходим к заключительной части обзора. Итак, посмотрим на существующие в рунете сайты крэкерских команд. Думаю, это подавляющее большинство всех ресурсов в Сети. Почему их нужно было выдвинуть под отдельный заголовок? Наверное, потому что именно в этом типе ресурсов можно найти конкретные релизы каждой команды и многое другое. »

Самое лучшее в exetools.net – это его FTP-сервер

MSDN от Microsoft – отличная подборка для крэкера

На Wasm'е ты найдешь не только инструменты, но и статьи

Пройти мимо Cracklab.ru просто невозможно



На biocyborg.cjb.net есть очень неплохой форум

Бегом за основными ресурсами.

Сначала определим команды, которые занимаются только выпуском крэков и на сайте которых нет ничего другого. Почему же не воспользоваться поисковиком типа Astalavista или http://crackspider.net, а может, просто зайти на cracks.am? Потому что каждая команда упорно стремится поддерживать свой авторитет на сцене и не станет заниматься массовым распространением вирусов через свои крэки. И вообще, крэк из первых рук как-то спокойства придает.

Известные всему миру таблетки можно найти по настоящему адресу http://tsrh.crackz.ws. Об этой команде пишут многие, она известна своей огромной производительностью, каждый день в мир приходит несколько новых релизов от TSRh. К такой же команде, выпускающей новые крэки пачками, можно отнести www.revenge.crackdb.com, которая помимо простых таблеток каждый месяц выпускает удобно представленную БД нескольких сотен серийных номеров.

www.xtin.km.ru – сайт, посвященный исследованию протекторов и не только. Тут много интересного - от статей элиты русской крэк-сцены (да простят меня они, но я назову их: GL#OM, HEX, (c)Dragon, MozgC и остальных) до различных Crackmes - программ для самопроверки во взломе.

Если рассматривать сайты дедов реверса, нельзя пройти мимо www.Uinc.Ru. На самом сайте довольно много статей, посвященных информационной безопасности в целом. Тут можно найти статьи и программы NEOx'a, автора широко известной утилиты PETools, а также почитать статьи такой исторической личности, как Dr.Golova.

Если брать современные команды, которые кроме простых релизов также выпускают многочисленные утилиты, плагины к утилитам и т.д., то не забывай сайты www.int3.net, www.DotFix.net от GPcH - автора множества статей и тулз, посвященных защите и взлому программ, написанных на Visual Basic.

И напоследок можно заглянуть на AHTeam.org - сайт команды Alien Hack, где есть небольшое, но интересное собрание их программ: начинай от NFOmaker'ов и заканчивай различного рода анпакерами. И о статьях по взлому тоже нельзя забыть.

ПОСЛЕСЛОВИЕ

Сайтов, посвященных взлому, действительно очень много. Нам бы не хватило журналов, чтобы просто перечислить все ресурсы Сети, где хоть каким-нибудь краешком затрагивается тема защиты и взлома программ. Я постарался привести список наиболее значимых и выдающихся ресурсов этой тематики. Какой сайт станет твоим вторым домом, решать тебе. Может быть, решишь создать свой проект, а я буду только рад. Так или иначе, со временем, познавая крэкинг, ты будешь заходить на такие ресурсы только с целью приятного времяпрепровождения или чтобы добавить свою статью/релиз/утилиту. Так оно и будет. Идеальной защиты не существует: все, что создано человеком, можно сломать. Не забывай посещать www.xakep.ru, на котором, кстати говоря, тоже можно найти некоторые достаточно добротные вещи по сегодняшней теме.

На AHTeam.org много оригинальных статей

Каждый день выходит несколько новых релизов от TSRh

Посети http://crackteam.ws/pages/TSRh/1.shtml, где не только сможешь загрузить чистые лекарства от известных команд, но и посмотреть рейтинг группировок на мировой сцене

НЕ ОГРАНИЧИВАЙ СЕБЯ

Играй просто!
GamePost

ПОЛУЧИ МАКСИМУМ УДОВОЛЬСТВИЯ

ИСПОЛЬЗУЯ ДОПОЛНИТЕЛЬНЫЕ АКСЕСУАРЫ

Колонки/ M-Audio Studiophile LX4 5.1 Eander

$199.99

Наушники/ AKG K66

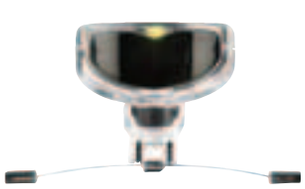
$32.99

Корпус/Shuttle SB95P

$489.99

Pinnacle Systems Studio 9 Plus RUS

$99.99



Трекер/NaturalPoint: TrackIR3 Pro

$175.99

Шлем/ i-O Display Systems i-glasses PC

$789.99

* В нашем агазине вас ждет более 1000 игр на ваш выбор
* Постоянно обновляемый ассортимент
* Постоянно обновляемый ассортимент

Тел.: (095) 780-8825
Факс. (095) 780-8824
www.gamepost.ru

GAMEPOST

Андрей Каролик (andrusha@real.xakep.ru)

КРЭКЕРЫ VS АВТОРЫ ЗАЩИТ

МНЕНИЯ СПЕЦИАЛИСТОВ

Можно сказать, что это вечное противостояние. Кто-то пишет защиту для программы, а кто-то ломает ее. Но разница между крэкерами и антикрэкерами (если их можно так назвать) очень мала, потому что, по сути, они обладают одними и теми же знаниями. Что они думают друг о друге и об этой проблеме, мы решили узнать непосредственно у них.

XS: Как вы относитесь друг к другу? Твое личное отношение к коллегам по цеху?

Крис Касперски aka мыщъх aka nezumi: Отношение сложное. Многие разработчики защит очень нервно реагируют на слово "хакер" или "крэкер" и ведут себя достаточно агрессивно, отказываясь общаться с нами. И это притом, что добрая половина "защитников" имеет весьма буйное хакерское прошлое.

Харон, талантливый человек, имеющий феноменальную работоспособность и гигантский опыт работы с железом и софтом: Строго говоря, хакеры взломом не занимаются :). Эта "расплывчатость терминологии" возникла, по-моему, по двум причинам: исторический "поиск термина" при переводах и то, что для взлома более или менее серьезной защиты действительно нужно иметь квалификацию хакера. Но называть крэкеров хакерами - это примерно то же, что называть хирургов палачами ;).

XS: Кого ты мог бы выделить на сцене и за какие достижения?

мыщъх: Есть такая поговорка: "Если приятели называют тебя хакером, знай, что ты ламер". Ибо настоящий хакер сидит в тени, притаившись тише травы, ниже радаров. Поскольку "хакеров в законе" пока что не наблюдается, приходится придерживаться строгой конспирации и действовать из подполья, поэтому никаких имен здесь не будет.

Харон: Из общеизвестных продуктов выделять можно только в отрицательном смысле. Например, всем (интересующимся этой областью) известный flexlm очень сильно облегчил жизнь взломщиков :). Нет, он, в принципе, позволяет строить почти терпимые защиты, но это лишь одна из его возможностей, причем крайне неудобная для enduser'ов. А поскольку эти самые юзвери не в состоянии понять, что все остальные его применения дают лишь видимость защиты... Ведь взлом любой защиты вида 'chk->jmp' сводится к поиску данной последовательности. А когда она унифицирована, то... А если еще учесть, что у пользователей flexlm хватает ума прикладывать pdb'шки от lm*.dll :). Таких примеров много, но принцип один: на сегодня в ширпотребовских продуктах защитой занимаются, как правило, люди, мало что в ней понимающие.

XS: Что же такое "крутая" защита?

мыщъх: Защита-головоломка, над которой можно просидеть целую неделю, но так ничего и не понять. Большинство трудноломаемых защит под эту категорию, кстати говоря, не подпадают. Разработчику ничего не стоит нашпиговаться кучей проверок, идущих из разных мест, или обернуть критический код сотней-другой упаковщиков. И хотя взлом займет море времени, удовольствия он доставит немного. Это будет нудная, однообразная работа без тени творчества вроде мытья полов или перекопки огорода.

Крис Касперски aka мыщъх aka nezumi

Харон: Крутые бывают только яйца :).

XS: Существует ли вообще защита, которую невозможно взломать? Возможно ли создать ее?

мыщъх: Взломать можно все, поскольку в крайнем случае программу можно переписать заново, что называется from scratch (с чистого листа), однако смысла в этом будет немного. Если речь идет о демонстрационной версии, то надежнее всего просто ограничить функциональные возможности. Например, выбросить модуль, ответственный за печать. Теоретически хакер может реализовать его и самостоятельно, но практически ни у кого мазы нет.

Харон: Нет. Есть старая дефиниция: "Все, что один человек сделал, другой завсегда сломать сможет" (с). Но, безусловно, существуют защиты, ломать которые нерентабельно. Трудозатраты на взлом могут существенно превышать стоимость (копии) продукта (и требовать большего времени), и заниматься в этом случае взломом будут только "пионеры". А сделать защиту, не взламываемую на этом уровне, вполне реально. Однако следует помнить, что стоимость взлома прямо пропорциональна стоимости самой защиты. Если мне не изменяет склероз, теоретический максимум отношения этих стоимостей - около десяти. Причем это число справедливо только для специализированных защит: любая внешняя защи-

та гарантированно имеет отношение намного хуже. Если рассматривать аппаратные защиты, то там это соотношение можно увеличить, но... не качественно.

XS: Какие отладчики и другие программы ты используешь в работе? Что ты используешь и для чего, то есть поподробнее.

мыщъх: Предпочитаю классику. Под Windows: дизассемблер IDA Pro, отладчик SoftIce (иногда OllyDbg), hex-редактор QVIEW. Под Linux: дизассемблер опять-таки IDA Pro, отладчик gdb и lince (неофициальный порт SoftIce), hex-редактор HT-edit. Все это хозяйство крутится под VMWare, как правило, одновременно работает три SoftIce'а и один linice в разных окнах. Красота! И никаких проблем с перезагрузками основной операционной системы.

Харон: А это вопрос абсолютно непринципиальный. Кто к чему привык, тот тем и пользуется. Я, например, отладчиком вообще очень редко пользуюсь: мне идой (ida) найти проще. А если пользуюсь, то только консольными, но это, повторюсь, вопрос вкуса. Я люблю консоль просто потому, что глаза меньше устают.

XS: Имеет ли какое-либо значение то, на каком языке написана программа? Или ломают и защищают только на общих принципах независимо от языка?

мыщъх: Разница есть. Проще всего ломаются программы, написанные на классическом C и работающие через Win32 API. С приплюснутым C, из-за обилия косвенных вызовов виртуальных функций, справиться уже сложнее, и для эффективного взлома отладчик и дизассемблер приходится запрягать в одну упряжку. Программы, написанные на Delphi, Builder или Visual C++ с MFC легко ломаются только тогда, когда библиотечные функции распознаются ИДОЙ (а происходит это далеко не всегда), еще выручает декомпилятор DeDe. Сложнее всего ломаются программы, скомпилированные в p-код или шитый код: Visual Basic, Forth и т.д. Особенно если программист не ограничился посимвольной сверкой пароля (она выполняется библиотечными функциями, которые легко перехватить), а реализовал целый алгоритм.

Харон: Защищают-то почти независимо, но разница, как ни странно, есть. Наиболее сложны для взлома программы, написанные на ммм... "некомпилируемых" языках. Скажем, на васике или жабе. Проблема, естественно, не в том, чтобы найти, что ломать (тут особой разницы нет), а в том, что сделать патч к класс-файлу намного более трудоемкое занятие, чем к какому-нибудь PE'шнику. Но и тут есть соображение "в утешение крэкерам": серьезных продуктов на подобных языках не пишут :).

XS: Опиши стандартные шаги защиты/взлома программы.

мыщъх: Взлом - дело творческое, и общепринятой методики здесь нет. Одну и ту же защиту можно взломать десятком независимых способов, и невозможно предугадать, какой путь будет быстрее. Вот только одна из возможных схем. Смотришь, упакована ли программа. Если упакована, пытаешься найти готовый распаковщик. Если же его нет, снимаешь дамп утилитами типа ProcDUMP/LordPE или пишешь распаковщик самостоятельно. Распакованный файл загружаешь в дизассемблер и просишь его сгенерировать MAP-файл, содержащий всю символьную информацию, что существенно упрощает отладку. Дальше IDA Pro и SoftIce будут действовать в одной связке. Дизассемблер ищет перекрестные ссылки на "ругательные строки", отладчик - перехватывает обращения к API-функциям, подозрительным ячейкам памяти, портам ввода-вывода и т.д. Все вместе они помогают понять алгоритм работы защиты и взломать ее. Затем либо пишется генератор ключей/серийных номеров, либо правятся несколько байтиков в программе. Незапакованные программы правятся прямо в QVIEW, для запакованных пишется специальный онлайновый патчер, меняющий их на лету.

Харон: На этот вопрос я могу ответить одним советом (совет тем, кто защищает). Если есть стандартные шаги, то защитой лучше не заниматься: все равно она продержится до первого пожелавшего ее взломать :). Защита должна быть не просто интегрирована в продукт - она должна разрабатываться под конкретный продукт. И использовать для нее надо все "специфики" продукта. Скажем, если приложение общается с внешним миром, то самое милое дело - выкинуть кусочек (хоть несколько байтов) защиты тоже вовне. Ну, и хорошо бы до того, как начинать делать защиту, подумать, а что, собственно, ты собираешься защищать? В смысле, чего хочется избежать за счет этой самой защиты. Защита ведь - это далеко не всегда "защита от копирования".

XS: На каких ошибках ты обжигался и что можешь посоветовать на основе печального опыта?

мыщъх: Сообщать разработчикам об ошибках ;). Серьезно. Многие из них впадают в сумеречное состояние души, граничащее с полной прострацией, и поднимают вселенский вопль, благодаря которому мы узнаем много новых ругательных слов. Также ни в коем случае, ни при каких обстоятельствах никому не следует угрожать ни в явном, ни в предполагаемом виде. Добром это не кончится! Никогда нельзя доверять ничему написанному, пока все не проверишь сам! Ошибаются даже авторитеты (например, в моих книгах ошибок просто толпа).

Харон: При защите: устанавливать на продукт цену, которая провоцирует взлом :). При взломе: начинать ломать, не заглянув в интернет :).

XS: Какая защита больше всего впечатлила за последнее время?

мыщъх: Ulink от Юрия Харона ошеломил своей оригинальностью (ему посвящено целых две главы в "Записках мыщъха"). StarForce 3 поразил своей навороченностью, которая все равно не спасла его от взлома :). Другие защиты как-то не отложились в памяти.

XS: Что нужно обычному человеку, чтобы стать крэкером или кодером защит?

мыщъх: Хакер, как и программист (в смысле "инженер") - это состояние души. Оно приходит либо с рождением, либо вспыхивает и подбрасывает тебя, как гроза. Кодер и крэкер - это ремесло, которому легко научиться. Достаточно выучить английский, поставить осла и вытянуть пару гигабайтов электронных книг. Ах да! Еще необходимо найти время, чтобы их прочитать!

Харон: Учиться, учиться и еще раз учиться (с). А если детальней, то один общий совет действительно можно дать: не стоит пытаться разрабатывать защиты, если нет опыта их взлома.

Проще всего ломаются программы, написанные на классическом C и работающие через Win32 API.

XS: Имеет ли вообще смысл защищать программы, если их все равно взломают хакеры?

Автор проекта InsidePro (www.InsidePro.com)**:** На мой взгляд, смысла нет. Мне, как программисту, интересней сделать свою программу чуть быстрее, чуть более функциональной, надежней и удобней для пользователя, чем тратить время на создание защиты, которую все равно взломают. А если и не взломают, то в момент, когда в интернет просочится хоть один серийный номер или регистрационный ключ, программу смело можно называть взломанной, так как ее теперь сможет использовать кто угодно. Лично мне в подобный момент будет ужасно жаль времени, потраченного на создание защиты :). Я понимаю, что можно навесить на программу уже готовую защиту типа ASProtect, Armadillo и прочих, но здесь есть ряд моментов. В Сети уже немало готовых руководств по взлому подобных защит, и есть спецы, знающие нутро таких защит как "Отче наш", так что кому нужно будет взломать - все равно взломают. И еще: когда пишешь программу размером в несколько десятков килобайт, оптимизируешь код, пишешь куски критичного кода на асме, а потом "навешиваешь" на нее протектор типа Armadillo и видишь, что твой файл стал "весить" более полумегабайта, то становится грустно :). Если бы существовали способы автоматического создания стойких защит и увеличивающих размер программы на 5-15 Кб, я, может быть, и воспользовался бы.

XS: Защищаешь ли ты свои программы хотя бы минимально - от дураков?

Автор проекта InsidePro: От дураков - да: я упаковываю PECompact'ом :). Конечно, можно найти распаковщик, распаковать EXE-файл, затем начать дизассемблировать. Но человека, который начнет этим заниматься, я бы к дуракам уже не причислил.

XS: Придумывают ли сейчас что-то новое в области защиты или продолжают использовать давно известные наработки?

Автор проекта InsidePro: Конечно, придумывают. Но и взломщики на месте не сидят: уровень их знаний также повышается, инструменты для взлома совершенствуются и т.д. Я думаю, взломщики растут синхронно с авторами защит. Короче говоря, симбиоз.

XS: Какие способы защиты программ сейчас наиболее стойкие к взлому?

Автор проекта InsidePro: Самые стойкие методы защиты - аппаратные. Их взлом - задача, непосильная для подавляющего большинства взломщиков, так как в этом случае взломщик защиты должен обладать знаниями не только в области взлома софта, но и отлично разбираться в радиоэлектронике, знать схемотехнику, иметь опыт программирования микроконтроллеров, должен уметь настроить осциллограф, уметь работать с паяльником (паяльной станцией) и т.п. (По-моему, большинство аппаратных защит ломается программно, надо просто знать как. - прим. Горлума.) А это уже не спец, а профи. Но таких мало :). Что касается программных методов защиты софта, для меня есть только один критерий: если временные (и материальные) средства, затрачиваемые на взлом, больше стоимости этой программы, то данный способ защиты является стойким.

XS: Какой уровень программирования нужен разработчику защиты или хакеру? На чем удобнее программировать и почему?

Автор проекта InsidePro: Несомненно, уровень должен быть высоким. Для того чтобы самому написать стойкую защиту или же самому суметь взломать подобную защиту, нужно быть спецом. Лучшие защиты пишутся на ассемблере или на асме и C. Взломщик же просто обязан знать ассемблер!

> Самые стойкие методы защиты - аппаратные. Их взлом - задача, непосильная для большинства взломщиков.

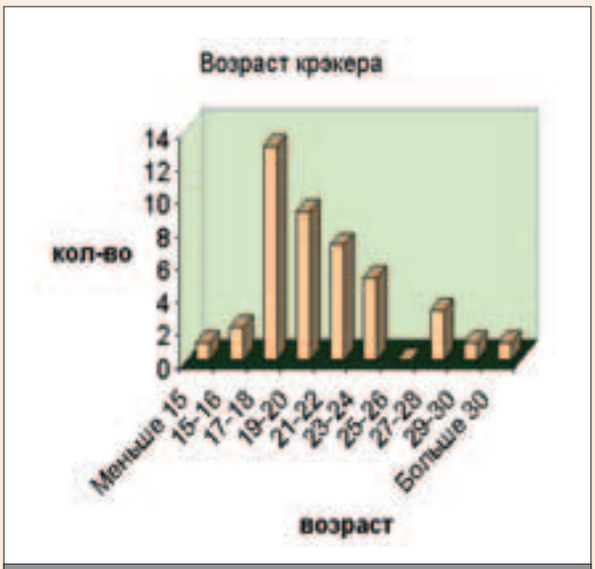
XS: Чему и кому посвящен твой проект? Хакерам или тем, кто пытается защититься от них?

Создатель портала исследования защиты программ www.cracklab.ru, ***Bad_guy:*** Мой проект посвящен исследованию защиты компьютерных программ, или, попросту, крэкингу. Для меня существует огромная разница между хакером и крэкером, мой проект именно для крэкеров - тем он и отличается от многочисленных хакерских порталов рунета. Проект для тех, кто ломает, или для тех, кто защищает? Мне было бы приятно, если бы сайтом интересовались и те, и другие. Но на данный момент разработчиков защит на сайте и на форуме не так уж много: крэкеров намного больше, хотя это и закономерно, так как статьи на сайте лежат в основном по снятию защиты, а не по принципам ее построения.

XS: К кому ты относишь себя? Ты активно занимаешься хаком? Как появилась идея создать подобный проект? Что он дает тебе?

Bad_guy: Крэкерством я занимаюсь более четырех лет. Все это началось, когда мне попалась на одном из обычных дисков с документами статейка "Как ломать программы Windows" от ED!SON [UCF], переведенная на русский (кстати, есть она вот тут: http://cracklab.ru/art/page21.php). Вот она меня и увлекла. Пару лет я занимался крэкерством очень активно и с большим азартом, тогда у меня не было интернета, и научился я всему сам по старым статьям от Фокса Малдера с Пилорамы. Через полтора года я устроился на работу в некую фирму, где мне поручили сделать им сайт, тогда же я и познакомился с интернетом. Делал я им сайт на хостинге narod.ru, параллельно зарегистрировал cracklab.narod.ru, где планировал разместить свои статьи по исследованию защиты программ - уж очень хотелось тоже писать. Я написал несколько материалов, потом со временем познакомился с крэкерами Vallkor и Fess, предложил поместить их статьи на мой сайт. Они согласились, после чего cracklab.narod.ru превратился из домашней странички в что-то более крупное. Потом я стал искать другие статьи крэкеров, и, в итоге, теперь на сайте уже более 300 статей. В какой-то момент родилась идея открыть форум: на сегодняшний день он является самым популярным разделом сайта. Позже хостинг на narod.ru перестал устраивать меня, тогда я решился на покупку домена www.CRACKLAB.ru и хостинга для сайта. Финансово мне с этим помогли участники форума, за что им огромное спасибо. В январе этого года проект CRACKL@B отметил свой трехлетний юбилей. Сайт мне приносит моральное удовлетворение.

XS: В чем разница между крэкерами и хакерами? Или, по сути, одинаковые методы, одинаковые подходы - только разные цели?



Возраст крэкера

Bad_guy: Принципиальной разницы действительно нет. Однако крэкер спокойно может заниматься своей деятельностью не имея совершенно никакого доступа к компьютерной сети. Все, что ему нужно - это отладчик, дизассемблер и диск со свежим софтом. К тому же крэкер, который не публикует свои крэки, никому не мешает своей деятельностью, никого не хочет обидеть, нанести какой-либо ущерб, что, в общем, более гуманно, чем взлом в привычном для нас понимании.

XS: Крэкинг - твой стиль жизни, хобби, возможность обуздать свои амбиции или способ заработать? Кто вообще идет в крэкинг?

Bad_guy: Дело в том, что я имею специальность "ЭВМ-комплексы системы и сети". Вот когда я учился, слушал лекции по микропроцессорам и ассемблеру, меня заинтересовал крэкинг, я погрузился в него с интересом, чтобы более глубоко внедриться в глубины работы компьютерной системы. А именно взлом программ - это интересно, ведь вид зеленой надписи "Registered" - это как золотая медаль чемпиону, и, думаю, многие крэкеры с этим согласятся. Насчет заработка: действительно, если участвовать в крэкерской команде и выполнять заказы, то можно немного подзаработать, но это только на карманные расходы студенту. Деньги, на самом деле, небольшие. В крэкинг же идут в основном студенты. Мы на форуме CRACKL@B однажды проводили опрос по возрасту крэкеров, и получается, что в основном крэкингом занимаются люди в возрасте от 17-ти до 25-ти лет.

XS: Есть ли способ противостоять крэкерам? От эпидемии есть прививки, от саранчи - химикаты, а как быть с неуемными крэкерами?

Bad_guy: Принципиальных способов противодействия два: техническое и юридическое. Техническое противодействие - это когда авторы программ усложняют защиту своих программ, применяют разные ухищрения, навешивают на исполняемый файл разнообразные протекторы, усложняющие взлом. Некоторые умельцы, которых, наверное, по пальцам перечесть, пытаются создавать самодельные эффективные алгоритмы против взлома своей программы. Естественно, все эти ухищрения в некоторой степени помогают защитить программу от многих неопытных крэкеров, но сложная защита, наоборот, только раззадоривает опытных. На данный момент сильнейшей технической защитой является StarForce, которой стало модно защищать компьютерные игры, чтобы их нельзя было запустить без диска.

Юридический подход – это противодействие скорее распространению крэков, чем взлому программы. Во-первых, я не раз слышал про происки компании 1С, которая привлекала людей, распространяющих взломанные версии их программ. Обычно такие дела доходили до суда, и нарушителям назначали условный срок и крупный штраф. Во-вторых, была такая история: существовала крэкерская команда (меня просили не называть никаких имен), мемберы которой размещали крэки прямо на сайте команды.

Один из авторов программ, к которой был опубликован крэк на сайте команды, обратился в органы: итогом стал арест сервера, на котором находился сайт команды, причем владелец сервера не имел никакого отношения к команде, он просто сдавал хостинг в аренду. В результате сервер, находящийся в Америке (!), был арестован прямо там по заявлению русского (!) программиста. Арендатор хостинга и участники команды не пострадали, однако сайт их так больше и не появился в Сети. И таких примеров масса.

XS: Как крэкеры выбирают объекты для атаки? Как не стать их жертвой?

Bad_guy: Единственный гарантированный способ не стать жертвой крэкеров - не писать программ. А вообще крэкеры занимаются в основном популярными программами. Поэтому чем более популярна программа, тем быстрее появится для нее крэк, а если программа выпущена только на родном языке (не английском) или слабо раскручена, то, возможно, крэкеры даже и не доберутся до нее никогда. Лично я всегда ломал только те программы, которые мне самому были нужны и на которые не было под рукой готового крэка, однако я никогда не выпускал свои крэки в Сеть и от моей деятельности разработчики явно не пострадали. Хотя как посмотреть... Учитывая создание мной CRACKL@B...

XS: Что нужно, чтобы стать крэкером? Как ты сам постигал крэкинг? Сколько потребовалось времени и когда ты понял, что знаешь достаточно? Кстати, что ты считаешь достаточным уровнем знаний?

Bad_guy: Чтобы стать крэкером, нужно, в первую очередь, желание. Но этого оказывается мало: нужно знать основы ассемблера, желательно знать архитектуру процессора, например 486-го. Мне эти знания очень помогли освоиться в крэкинге легко и быстро. Обязательно стоит прочитать несколько статей от MozgC, Bitfry. Когда я начинал, то обходился одним лишь отладчиком SoftI, который теперь идет в пакете DriverStudio, мне его вполне хватало. Потом уже я стал разбираться в дизассемблировании. Позже, когда пошла мода на EXE-упаковщики и протекторы, познакомился и с PE-инструментами, и форматом PE. Когда я два года позанимался крэкерством, у меня появилась мысль о том, что я знаю в крэкинге достаточно. А "достаточный уровень знаний"... Хм, просто возникло такое внутреннее убеждение, что теперь я при желании смогу разобраться в защите абсолютно любой программы. Кстати, о том, как стать крэкером и с чего начать, меня спрашивают достаточно часто, и поэтому я на сайте кратко все расписал (http://cracklab.ru/kid.php).

Единственный гарантированный способ не стать жертвой крэкеров - не писать программ.

Спец Хакер

АНОНС

Читай в следующем номере Спеца

БЕЗОПАСНОСТЬ "КЛИЕНТОВ" И ПРОТОКОЛОВ

+ Весь софт на CD

- Windows на страже порядка
- Шпионаж за системными событиями
- RingO
- Проникновение в protect storage
- Ошибки клиентских приложений
- Поиск дыр в бинарном коде
- Контент-фильтрация
- Переполнение буфера
- Обход защиты клиентских приложений
- Проход через брандмауэр
- Криптопротоколы
- Уязвимости протоколов ICQ, FTP, SMTP

А также:

- TOL - ЗАБЫТЫЙ ПРОТОКОЛ ОТ MIRABILIS И ЕЩЕ МНОЖЕСТВО НЕОЧЕВИДНЫХ УЯЗВИМОСТЕЙ В ОЧЕВИДНЫХ ВЕЩАХ!

СКОРО В СПЕЦЕ:

● МОБИЛЬНЫЕ УСТРОЙТВА И ИХ БЕЗОПАСНОСТЬ

Взлом с помощью мобильных устройств. Bluejacking, bluesnarfing и взлом Wi-Fi-сетей. Снифферы Wi-Fi/Bluetooth. Все о wardriving. Мобильные вирусы и трояны. Security-софт под мобильные платформы. Фрикинг, безопасность в телекоммуникациях. Спам.

● ИНТЕРНЕТ-ДЕНЬГИ

Обменники валюты, казино и другие web-сервисы, связанные с интернет-валютой. Различные платежные системы: WebMoney, e-gold, GoldMoney, PayPal и др. Заработок/процессинг: что и как реализовать. Как сделать свою пирамиду/банк, как кидают в е-бизнесе.

● ШПИОНСКИЕ ХИТРОСТИ

"Большой брат следит за тобой". Все о жучках. Компьютерный шпионаж. Тайна PGP: есть ли в нем "троян". Реализация слежки и противостояние ей.

А Н К Е Т А

Если ты хочешь помочь нам делать журнал, вступай в фокус-группу Спеца! Участники фокус-группы смогут первыми оценить предстоящие нововведения, высказывать свое мнение о каждом номере напрямую редакции. От тебя требуется немного: быть в онлайне, периодически отвечать на вопросы редакции и, самое главное, желание. Чтобы попасть в фокус-группу, нужно всего лишь заполнить эту анкету и прислать ее нам. **Если ты не хочешь быть в тест-группе, все равно пришли анкету - нам это очень важно!**

ЗАПОЛНИ АНКЕТУ - ПОЛУЧИ ПРИЗ!
Среди всех читателей, заполнивших эту анкету и приславших нам ее до 31 августа 2005 года, будет разыгран USB flash drive Kingston DataTraveller Elite на 512 Мб!

Давно ли ты читаешь "Хакер Спец"?
- ○ С первых номеров
- ○ Около года
- ○ Несколько последних номеров
- ○ Первый раз

Как ты считаешь, изменился ли "Хакер Спец" за последнее время?
- ○ Да, улучшился
- ○ Да, ухудшился
- ○ Нет, по-моему, не изменился

Какой из последних номеров тебе понравился больше всего?
- ○ 05.05(54) - Цифровое видео
- ○ 06.05(55) - Компьютеры будущего
- ○ 07.05(56) - Мобильные деньги
- ○ 08.05(57) - (anti)cracking

Хотелось бы тебе новых рубрик в ОФФТОПИКе?
- ○ Да
- ○ Нет

Достаточно ли объемна ТЕМА НОМЕРА?
- ○ Вполне
- ○ Ее надо увеличить
- ○ Слишком большая

Было бы тебе интересно читать новости в Спеце?
- ○ Да
- ○ Нет

Интересна ли тебе СТОРИ?
- ○ Да
- ○ Нет
- ○ А что это?

Какие компьютерные журналы ты еще читаешь?
- ☐ Хакер
- ☐ CHIP
- ☐ CHIP Special
- ☐ Компьютерра
- ☐ Upgrade
- ☐ Мик ПК
- ☐ Upgrade Special
- ☐ Другой _____

Какой оптический привод в твоем компьютере?
- ○ CD-ROM/CD-RW
- ○ Combi CD-RW/DVD-ROM
- ○ DVD-ROM/DVD-RW

Предложи тему для очередного номера:

_____

_____

О себе

ФИО

_____

_____

Где ты живешь?

_____

_____

_____

_____

E-mail

_____

_____

Сколько тебе лет?
- ○ Меньше 17
- ○ 18-20
- ○ 21-23
- ○ 24-27
- ○ 28-30
- ○ 30-33
- ○ Больше 33

Твое семейное положение?
- ○ Холост
- ○ Женат

В каком вузе ты учишься?
- ○ Техническом
- ○ Гуманитарном
- ○ Я не учусь в вузе

Связана ли твоя работа с информационными технологиями?
- ○ Да
- ○ Да - планирую работать в ИТ
- ○ Нет
- ○ Я не работаю

Твой средний месячный доход?
- ○ Меньше $100
- ○ $100-300
- ○ $300-700
- ○ Больше $700

Сможешь ли ты сам собрать компьютер?
- ○ С закрытыми глазами
- ○ По книжке
- ○ Сомневаюсь

Какой у тебя канал в интернет?
- ○ Выделенка
- ○ Dial-up
- ○ Нет интернета

Чем ты пользуешься для общения в Сети?
- ☐ E-mail
- ☐ Чаты
- ☐ ICQ и другие мессенджеры
- ☐ Другое_____

На каком языке ты пишешь?
- ☐ Assembler
- ☐ C/C++
- ☐ Pascal/Delphi
- ☐ Basic/VB
- ☐ Perl
- ☐ Другое_____
- ☐ Я не программер

С какими платформами у тебя есть опыт работы?
- ☐ PC (Windows)
- ☐ *nix (Unix, Linux, BSD)
- ☐ Macintosh
- ☐ Palm OS
- ☐ Pocket PC (Windows CE)
- ☐ EPOC/Symbian
- ☐ Другое_____

Какие из перечисленных вещей у тебя есть?
- ☐ DVD-плеер
- ☐ DVD-ROM
- ☐ MP3-плеер
- ☐ Ноутбук
- ☐ Домашний кинотеатр
- ☐ Мобильный телефон
- ☐ КПК (коммуникатор)
- ☐ Цифровой фотоаппарат
- ☐ Цифровая видеокамера
- ☐ GPS-навигатор

☐ **Да, я хочу в фокус-группу!**

Заполненную анкету присылай по адресу: 101000, Москва, Главпочтамт, а/я 654, Хакер Спец с пометкой «Анкета» или на vote@real.xakep.ru.

Спец Хакер

Content:

Окунев Дмитрий, Шамаев Дмитрий, test_lab (test_lab@gameland.ru)

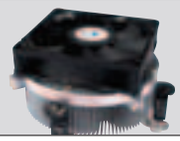
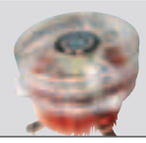
С ХОЛОДНЫМ LGA

ТЕСТИРОВАНИЕ КУЛЕРОВ ДЛЯ ПЛАТФОРМЫ LGA775

Лето ведет себя по отношению к пользователям компьютеров весьма благородно. Погода позволяет нам проводить меньше времени за изрядно надоевшей за рабочий год "тачкой" и, соответственно, на время отбросить навязчивые мысли об апгрейде. Но у этого времени года есть и оборотная сторона: все та же жаркая погода, так благоприятно влияющая на нас, к нашим компьютерам совсем не дружелюбна. Даже работающие на номинальной частоте процессоры подчас пасуют перед высокой температурой и начинают нещадно глючить, что уж говорить об их разогнанных до упора собратьях! Но выход из досадной ситуации все же есть! Привить "камню" иммунитет против жары способна достойная система охлаждения, благо рынок сейчас просто завален решениями на любой вкус и кошелек. Чтобы помочь тебе сориентироваться во всем этом многообразии, мы посвятили этот материал исследованию нескольких новых и не очень моделей кулеров.

test_lab выражает благодарность за оборудование, предоставленное на тестирование, компаниям "БЮРОКРАТ" (тел. (095) 745-5511, www.buro.ru), NEVADA (тел. (095) 101-2819, www.nevada.ru), ICEHAMMER Electronics Russia (www.icehammer.ru), "ПИРИТ" (тел. (095) 785-55-54, www.pirit.ru), "Аэртон" (www.airton.com).

СПИСОК УСТРОЙСТВ

	GlacialTech Igloo 5050 PWM
	GlacialTech Igloo 5100 PWM
	GlacialTech Igloo 5600 Light
	Intel Box Cooler
	IceHammer IH3875WV
	CoolerMaster Vortex TX
	CoolerMaster Hyper 6+

ТЕХНОЛОГИИ

За всю историю систем охлаждения их технология в общем изменилась мало и до сих пор представляет собой сочетание радиатора с хорошей теплопроводностью и вентилятора. Радиатор тесно контактирует с процессором и поглощает исходящее от него тепло, в то время как вентилятор либо обдувает его, либо, наоборот, отводит от него горячий воздух. Материал, из которого изготавливается радиатор - это, как правило, алюминий или медь, причем последняя обладает более высокой теплопроводностью, в то время как алюминий позволяет значительно снизить вес устройства. Описанная технология за время своего существования неоднократно подтвердила свою эффективность и, разумеется, отказываться от нее никто из производителей не собирается. Но улучшить ее путем добавления различных полезных "фишек" вполне возможно. Так, очень популярными в последнее время стали кулеры с измененной формой радиатора. Здесь все ограничивается только фантазией и возможностями разработчика. К примеру, особую популярность в этой области снискала продукция компании Zalman, вся линейка которой отличается оригинальностью конструкции. Кроме того, современные модели систем охлаждения часто снабжают датчиками температуры и реобасами для регулировки скорости вращения вентилятора.

Если изначально конструкция кулера подразумевала горизонтальное расположение

вентилятора, то сейчас на рынке нередко можно встретить модели с его вертикальным расположением. Такие системы охлаждения работают по принципу турбины самолета: воздух с силой выдувается не вверх, а вбок от охлаждаемой поверхности. Это не дает какой-либо пользы для охлаждения процессора, зато очень хорошо сказывается на близлежащем железе, например на видеокарте, да и чисто эстетически смотрится очень приятно.

Еще одно новшество, нашедшее применение в новейших моделях систем охлаждения - тепловые трубки. Внутри такой трубки находится полость с жидкостью, которая при нагреве закипает и испаряется. Соответственно, в противоположном (более холодном) конце трубки пар конденсируется, отдавая тем самым накопленное тепло. Этот цикл повторяется бесконечно и значительно повышает эффективность работы кулера.

НА ЧТО ОБРАТИТЬ ВНИМАНИЕ

Кулер - девайс недорогой, но возлагающий на себя огромную ответственность, поэтому и выбирать его нужно очень внимательно, руководствуясь сразу несколькими его характеристиками, описанными нами выше.

Во-первых, обрати внимание на материал, из которого изготовлена основа кулера - радиатор. Если ты увлекаешься разгоном процессора или же твой "камешек" сам по себе работает на высокой частоте, то лучше если это будет медь. Алюминий в твоем случае обеспечит едва достаточный уровень охлаждения, которого в определенных условиях может и не хватить. В крайнем случае можно обратить внимание на модель, использующую оба материала. Как правило, подобные радиаторы либо содержат металлы в одинаковой пропорции, либо же медь используется только для сердечника, а ребра сделаны из алюминия.

Во-вторых, до покупки ознакомься с характеристиками вентилятора. Здесь все зависит от твоих потребностей. Высокая скорость вращения не только обеспечит лучший уровень охлаждения, но и создаст ощутимый фоновый шум. В то же время модель с низкоскоростным вентилятором будет функционировать практически незаметно, правда, и справляться со своей основной функцией она будет хуже. В подобной ситуации неплохо, если к кулеру прилагается регулятор скорости вращения вентилятора. Тогда ты сам сможешь выбирать нужное соотношение эффективности и шума. Такие устройства выпускаются как во внутреннем варианте, так и в виде планок, предназначенных для установки на заднюю панель системного блока. Но самый удобный тип - это реобасы, предназначенные для установки в пятидюймовый отсек корпуса. Эти девайсы значительно облегчают управление кулером и нередко содержат на себе дополнительные "полезности" вроде ЖК-экрана с показаниями температуры или USB-хаба. Возможен также вариант, при котором материнская плата умеет динамически изменять скорость вращения вентилятора в зависимости от показаний температурного датчика. При этом с тебя окончательно снимаются все лишние заботы - система позаботится о себе сама.

Наконец, немаловажная составляющая системы охлаждения - это крепление. Конечно, большинство пользователей устанавливают и снимают кулер нечасто, но при неудачной конструкции эти редкие ситуации способны создать немало проблем. Если кулер крепится к сокету при помощи проходящей через него скобы, обрати внимание на ее качество. Она не должна быть очень податливой (иначе плохой контакт радиатора с процессором обеспечен), но и слишком тугой тоже быть не должна, то есть придется запастись немалыми силами для установки. Неплохо, если кулер крепится непосредственно к самой материнской плате. Это довольно надежно, правда, у этого метода есть и минус: плату придется демонтировать.

К счастью, в последних моделях процессоров решена одна из наиболее злостных проблем предыдущих поколений - возможность скола ядра при установке кулера. Если ты обладатель Intel Pentium4 или процессора AMD на платформе Socket 754/939, то твой "камень" надежно защищен стальной пластиной и тебе не придется, скрипя зубами, устанавливать строптивый кулер, попутно опасаясь за сохранность ядра.

МЕТОДИКА ТЕСТИРОВАНИЯ

Чтобы уровнять условия, каждый кулер использовался не со своим термоинтерфейсом, а совместно с термопастой "АлСил-3". Эффективность кулеров измерялась путем прогрева процессора программой S&M и замера температуры до и после теста. Для снятия показаний датчика использовалась программа Asus PC Probe.

GLACIALTECH IGLOO 5050 PWM

Эта модель во многом повторяет строение боксового кулера Intel, но лишь на первый взгляд. Отличия проявляются в конструкции радиатора: он полностью алюминиевый, в отличие от "оригинала". Также здесь используется собственная система крепления: на обратной стороне материнской платы необходимо установить специальную крестообразную металлическую пластину. Уже к ней крепится само устройство, что позволяет избежать прогиба "мамки" и сделать конструкцию более устойчивой плюс значительно продлевает время установки. Вентилятор на девайсе установлен достаточно мощный. Его скорость составляет до 3800 об/мин и при этом может свободно регулироваться (минимальное значение - 800 об/мин). Подошва радиатора уже содержит на себе термопасту, так что возиться с ней дополнительно не придется (или же наоборот, если ты исключительный фанат старого доброго АлСил-3 :)).

Технические характеристики:
Поддерживаемые разъемы: LGA775
Материал радиатора: алюминий
Скорость, об/мин: 800 +- 300 - 3800 +- 10%
Уровень шума, дБ: 18 - 40 +- 10%
Вес, г: 395
Размеры радиатора, мм: 90x90x36
Размеры вентилятора, мм: 80x80x27
Скорость потока, CFM: 13,3-63,1 +- 10%
Разъем для подключения, контактов: 4
Максимальная температура при нагрузке, ° C: 76
Минимальная температура без нагрузки, ° C: 52

Устройство показало далеко не потрясающий, но и не самый плохой результат: процессор нагрелся до 74-х градусов. Видимо, здесь дал знать о себе алюминиевый радиатор - "медные" собратья все же показали себя значительно лучше...

GLACIALTECH IGLOO 5100 PWM

Модель, опять же, сконструирована под впечатлением от боксового варианта Intel. В самом деле, зачем сильно видоизменять то, что уже не раз проверено "в боях" и обеспечивает сносную эффективность? Радиатор этой модели имеет круглую форму и лишь слегка больше в диаметре, чем детище Intel. Его основа- медный цилиндр, являющийся заодно и основанием купера, от которого, как лучи, ответвляются тонкие алюминиевые ребра. Модель устанавливается, опять же, стандартно, что должно прийтись по вкусу большинству пользователей: ножки просто защелкиваются в сокете без особых усилий со стороны устанавливающего. Правда, как показал тест, такое крепление не оправдывает себя полностью: процессор набрал целых 76 градусов! Девайсу не помог и неслабый вентилятор на 3600 об/мин (кстати, довольно шумный, что неудивительно), хотя, казалось бы, при таких характеристиках устройство просто обязано проявить себя с лучшей стороны! В целом данный кулер имеет только один очевидный плюс - удобство установки. Можем порекомендовать его только обладателям не очень мощных "камней".

Технические характеристики:
Поддерживаемые разъемы: LGA775
Материал радиатора: медный сердечник и алюминиевый радиатор
Скорость, об/мин: 600 +- 300 - 3600 +- 10%
Уровень шума, дБ: 15 - 36,9 +- 10%
Вес, г: 530
Размеры радиатора, мм: 90x90x36
Размеры вентилятора, мм: 80x80x18
Скорость потока, CFM: 8,76-51,65 +- 10%
Разъем для подключения, контактов: 4
Максимальная температура при нагрузке, ° C: 76
Минимальная температура без нагрузки, ° C: 52

GLACIALTECH IGLOO 5600 LIGHT

Конструкция этой модели очень оригинальна. Алюминиевая пластина, служащая подошвой купера, тесно соединена с алюминиевым же радиатором. Но это только "нижний этаж" системы. От него, в свою очередь, вверх отходят медные тепловые трубки, на которые надето множество алюминиевых пластин, также образующих радиатор - это "верхний этаж". В верхней части вертикально закреплен вентилятор, выдувающий воздух вбок, что позволяет обдувать еще и железо, находящееся в стороне от процессора. Так как название этой модели имеет приставку "Light" ("тихий" вариант), то и вентилятор здесь соответствующий - со скоростью вращения всего 2400 об/мин. В общем, устройство довольно внушительное, но кроткое :).
Установка этой модели, как и Igloo 5050 PWM, требует внимания материнской платы и прикручивания к ней специальной пластины, но усилия и потраченное время полностью оправдаются. Кулер не только довольно качественно стартовал (с температурой 49 градусов), но и позволил процессору разогреться всего до 69 градусов!

Технические характеристики:
Поддерживаемые разъемы: LGA775
Материал радиатора: алюминиевый радиатор с медными тепловыми трубками
Скорость, об/мин: 2400 +- 10%
Уровень шума, дБ: 26 +- 10%
Вес, г: 470
Размеры, мм: 108x96,5x90
Скорость потока, CFM: 35,04 +- 10%
Разъем для подключения, контактов: 3
Максимальная температура при нагрузке, ° C: 69
Минимальная температура без нагрузки, ° C: 49

INTEL BOX COOLER

Боксовый кулер - стандартная модель, поставляемая с процессорами Intel Pentium4 для платформы LGA775. Представляет собой невысокий круглый алюминиевый радиатор с медной подошвой (именно подошвой, а не сердечником, как у GlacialTech Igloo 5100 PWM) и водруженным сверху вентилятором. Сразу отметим главный недостаток этого вентилятора: его лопасти практически не закрыты от внешнего воздействия. Если провода внутри корпуса плохо закреплены, то вполне смогут попасть в подвижную часть, из-за чего повредится как сам кулер, так и процессор! Девайс довольно легко устанавливается на материнскую плату простым защелкиванием в специальных пазах, при этом никаких дополнительных инструментов не требуется. Но, как показала практика, простота крепления оборачивается слабым контактом подошвы и ядра процессора. В нашем случае температура "камня" подскочила аж до 79 градусов! Это самый "слабый" результат в нашем тесте, так что делаем выводы: положительные черты девайса исчерпаны доступностью и простотой установки.

Технические характеристики:
Поддерживаемые разъемы: LGA775
Материал радиатора: медный сердечник и алюминиевый радиатор
Скорость, об/мин: N/A
Уровень шума, дБ: N/A
Вес, г: 380
Размеры радиатора, мм: 90x90x36
Размеры вентилятора, мм: 80x80x20
Скорость потока, CFM: N/A
Разъем для подключения, контактов: 4
Максимальная температура при нагрузке, ° C: 79
Минимальная температура без нагрузки, ° C: 55

ICEHAMMER IH3875WV

Компания IceHammer отличилась на рынке систем охлаждения в первую очередь тем, что не разрабатывала собственный дизайн или не использовала стандартные наработки, а полностью скопировала конструкцию известной модели Zalman - CNPS 7700-Cu. Насколько же удачным получился этот "клон"? Форма радиатора у этой модели достаточно оригинальна: он выполнен в виде огромной чаши, состоящей из спрессованных и "распушенных" на 360 градусов медных пластин. Подошва кулера - это как раз то место, где соединены пластины. Из отличий от "прародителя" можно назвать небольшие косметические изменения: уменьшилось количество ребер, а сами они были изогнуты в форме волны (это позволяет снизить потери тепла благодаря уменьшению площади теплообмена). Приятно, что есть возможность установки устройства даже на устаревшую платформу Socket A, а в комплекте присутствует регулятор скорости вращения вентилятора - это преумножает достоинства модели перед остальными участниками теста. Что до температуры, то результат был показан неплохой - со своими 73 градусами Цельсия кулер занял почетное третье место :).

Технические характеристики:
Поддерживаемые разъемы: LGA775, Socket A, 370, 478, 754, 939, 940
Материал радиатора: медь
Скорость, об/мин: 1200-2200 +- 10%
Уровень шума, дБ: 15-23 +- 10%
Вес, г: 733
Размеры, мм: 138x138x66
Скорость потока, CFM: 53,5-78,5
Разъем для подключения, контактов: 3
Максимальная температура при нагрузке, ° C: 73
Минимальная температура без нагрузки, ° C: 50

COOLERMASTER VORTEX TX

Этот кулер возможно установить, наверное, на все известные платформы. Внешне эта модель выглядит очень симпатично: сказывается радиатор, чем-то похожий на боксовый от Intel, но выполненный целиком и полностью из меди! Конструкцию довершает круглый прозрачный вентилятор с переменной скоростью вращения от 1800 до 3200 об/мин. Но первое впечатление быстро развеялось при более пристальном осмотре. Как оказалось, расстояния между разными ребрами радиатора намного отличаются. Это, конечно, не критично, но создается впечатление, что кулер уже использовали длительное время. Вполне может быть, что это была проблема экземпляра, который попал в руки к нам. Крепление системы - отдельный разговор: оно требует прикручивания специальных пластин на материнскую плату и на сам кулер, что довольно сложно при установке (нужно демонтировать материнскую плату).

Технические характеристики:
Поддерживаемые разъемы: LGA775, Socket A, 370, 478, 754, 939, 940
Материал радиатора: медь
Скорость, об/мин: 1800-3200
Уровень шума, дБ: 26-36
Вес, г: 440
Размеры радиатора, мм: 88x88x35
Размеры вентилятора, мм: 92x92x25
Скорость потока, CFM: 28,2-46,8
Разъем для подключения контактов: 4
Максимальная температура при нагрузке, ° C: 77
Минимальная температура без нагрузки, ° C: 54

COOLERMASTER HYPER 6+

Этот кулер - настоящий "красавчик" по сравнению с остальными моделями, принявшими участие в нашем тесте. Но его главное достоинство в собрании лучших наработок в области охлаждения: подошва девайса сделана из меди и соединена тепловыми трубками с верхней частью, образованной множеством алюминиевых ребер. Они, в свою очередь, заключены в кожух, добавляющий системе пару баллов в оценке стиля. Вся эта красота охлаждается мощным вентилятором с частотой вращения до 3600 об/мин, ощутимо громким, зато он снабжен синими светодиодами. Так внутри корпуса создастся очень приятная атмосфера, что наверняка понравится любителям моддинга.

Теперь о главном - о результатах тестирования. Кулер показал себя просто превосходно и легко занял в нашем тесте первое место, даже опередил "серебряного призера" на 10 градусов! Если бы не сложная и долгая процедура установки (с использованием все тех же прижимных пластин), девайс можно было бы назвать идеальным. С другой стороны, можно вполне пожертвовать удобством установки ради такого качества.

Технические характеристики:
Поддерживаемые разъемы: LGA775, Socket A, 370, 478, 754, 939, 940
Материал радиатора: радиатор алюминиевый с медным основанием и тепловыми трубками
Скорость, об/мин: 1800-3600
Уровень шума, дБ: 20,6-46,4
Вес, г: 800
Размеры радиатора, мм: 135x112x80
Размеры вентилятора, мм: 100x100x25
Скорость потока, CFM: 31,33-72,14
Разъем для подключения, контактов: 4
Максимальная температура при нагрузке, ° C: 59
Минимальная температура без нагрузки, ° C: 44

ВЫВОД

Как видишь, ни одни из протестированных нами кулеров нельзя назвать идеальным. Какие-то модели сдают свои позиции в плане удобства установки, какие-то обеспечивают слабое охлаждение, некоторые слишком громко шумят... Это стандартная ситуация, и руководствоваться в ней придется не только нашими фактами, но и своими предпочтениями, а также суммой, которую ты готов потратить на подобный девайс. За отличный дизайн и высокую эффективность награду "Наш Выбор" получает модель CoolerMaster Hyper 6+, недостатки которой ограничиваются излишне шумным вентилятором. Если же этот параметр тебя смутит, стоит обратить внимание на кулер GlacialTech Igloo 5600 Light, признанный нами "Лучшей Покупкой" за кроткий нрав, легкость монтажа, и, конечно же, тишину при работе.

Сергей Никитин, test_lab (test_lab@gameland.ru)

GOTVIEW PCI 7135

TV-ТЮНЕР УМНЕЕ ТЕЛЕВИЗОРА

Компьютер, в отличие от телевизора, предназначен для программирования, поэтому вполне разумно требовать от TV-тюнера максимума настроек и расширенной функциональности. Но почему-то не все производители уделяют этому достаточное внимание. Мы протестировали GoTView PCI 7135 и убедились, что из него как раз можно выжать массу полезных функций.

Начнем с того, что весь софт на русском языке и поддерживает программу телепередач, обновляемую через интернет. Благодаря этому всегда можно узнать, какие передачи идут по разным каналам и сколько времени осталось до конца текущей программы.

Если разные каналы работают с разной громкостью, как это нередко бывает, то можно для каждого из них настроить уровень звука отдельно, а также, при необходимости, корректировать под каждый канал настройки изображения.

Автоматический поиск каналов находит не только основные каналы, но и паразитные, так что после поиска придется несколько почистить табличку настройки от лишних каналов. Кстати, в этой же таблице хранятся настройки УКВ- и FM-радиостанций, так как тюнер позволяет слушать и радио. GoTView PCI 7135 поддерживает несколько звуковых эффектов эмуляции объемного звучания. Причем звук может выводиться как по внешним, так и по внутренним коннекторам, плюс может передаваться по шине PCI.

Некоторые каналы, как и на телевизоре, могут приниматься с помехами в виде полос на экране. Для улучшения качества картинки, например для ликвидации "эффекта расчески", можно использовать специальные фильтры, которые настраиваются в программном обеспечении. Правда, они обычно дают дополнительную нагрузку на систему. Однако с помощью гибких настроек легко добиться хорошего качества даже на слабой системе.

В комплекте имеется удобный пульт, все кнопки которого можно запрограммировать. IR-датчик подключается прямо к TV-тюнеру и не занимает USB- или COM-порт на компьютере.

Как любой нормальный тюнер, GoTView PCI 7135 позволяет захватывать видео, причем при записи будут доступны фильтры. Также можно захватывать отдельные кадры и записывать их в одном из графических форматов. Но самое интересное, что можно смотреть телевизор удаленно с другого компьютера, при этом видео будет передаваться по сети. Это оценят пользователи, имеющие дома несколько компьютеров.

Сейчас стали популярны дешевые миниатюрные видеокамеры, их можно купить на любом вещевом рынке по цене 1000-2000 рублей. Софт данного ТВ-тюнера может обнаружить движение и начать запись видео и вдобавок к этому выполнить какие-то другие команды при обнаружении движения.

В итоге мы получаем ТВ-тюнер с качественной электронной начинкой, позволяющей выводить картинку не хуже обычного телевизора, с массой дополнительных возможностей по настройке как традиционных функций, так и с экзотическими, но весьма полезными сетевыми и охранными свойствами. Не оставлена без внимания и функция отложенного просмотра – Time Shift. Плюс при наличии двух тюнеров в компьютере можно смотреть сразу два канала – режим PIP.

Технические характеристики:
Тип тюнера: внутренний
УКВ- и FM-радио: есть
Поддержка форматов: AVI, WMV, MPEG 1/2/4
Пульт ДУ: есть
ПО в комплекте: GOTVIEW PRO, WinDVD Creator 2
Кабели в комплекте: кабель с ИК-датчиком, audio, внешняя антенна
Основной чипсет: Philips SAA7135
Интерфейсы: TV antenna, S-video in, RCA in, mini-jack in/out, IR
Цена, $64

СПЕЦ ХАКЕР

СОДЕРЖАНИЕ CD

- Спец 06(55), Компьютеры будущего
- Хакер 06(78)
- Железо 06(16)
- Мобильные компьютеры 06(57)
- Обновления для Windows за месяц

Если ты не занимался крэком, то, готов поспорить, частенько задумывался о возможности этого. Что ж, теперь у тебя есть все для того, чтобы попробовать себя на этом нелегком, но чрезвычайно интересном поприще!

И ЕЩЕ:
весь софт из номера!

POCKETPC$HACK
- eMbedded Visual C++ 4.0
- eMbedded Visual C++ 4.0 SP4
- Microsoft Pocket PC 2003 SDK

ВЗЛОМ И РЕВЕРСИНГ
- DeDe 3.50.02.1619
- DJ Java Decompiler v.3.8.8.85
- EMS Source Rescuer
- Hex Workshop v.4.23
- ht 0.9.1
- icedump 6.026 & ntice-dump 1.14
- IceExt v0.67
- IDA 4.8
- OllyDbg 1.10
- PE Tools v1.5.400.2003 Xmas Edition
- PEiD 0.93 + plugins
- RACEVB6 3.4.0
- REC 2.0
- Semi VB Decompiler 0.03
- VB Decompiler Lite 0.1
- VB RezQ 2.6
- VBParser 7.1.0.41 + src
- VBReFormer 3.8
- Winspector

ЗАЩИТА
- Armadillo 4.20
- ASPack 2.12
- ASProtect 1.23
- Obsidium 1.2
- ORiEN
- SoftwarePassport v.2.2.1
- Stealth PE 2.1
- Hide PE 1.1
- UPX 1.25 (dos+linux+win32+src)
- VMProtect v.1.05
- FSG 2.0
- Mew 11 SE v.1.0
- Morphine v.2.7
- (Win)Upack v.0.29b

СОФТ ОТ NONAME
- BurnInTest v.4.0 Pro
- Blaze DVD Copy 3.5
- Deer Park Alpha 2
- Hidden Camer 2.11
- Microsoft Baseline Security
- Analyzer 2.0
- Morpheus 5.0 BETA
- NetPeeker 2.82
- Onlineeye Pro
- Opera 8.02
- Quintessential Media Player
- Build 103
- Total Uninstall 3.32
- URLBase 5 Pro
- Sarmsoft Web Camera v2.0
- WinXP Manager
- ZendStudio 4.0.2

Все это на МУЛЬТИЗАГРУЗОЧНОМ CD!

NeO1eX (NeO1eX@rambler.ru; www.alexhak.narod.ru)

ПАЯЛЬНИК

ШНУРКИ К ТЕЛУ

Ты хочешь стать фрикером? Не знаешь, с чего начать? Ты кто? Жестянщик или в столовую строем ходишь? С практики, естественно!

ПРЕДИСЛОВИЕ

В жизни каждого мужчины настает момент, когда он уже не чувствует в себе былой удовлетворенности от обыденной жизни. Когда обычные игры с паяльником уже не приводят в экстаз и душе хочется новых ощущений и переживаний. Писк очередного мультивибратора уже не приносит той удовлетворенности - некоторые в такие моменты спиваются, другие подсаживаются на наркоту, третьи вообще пускаются во все тяжкие, сгорая тем самым как личность бесповоротно. Когда у меня настал такой момент, я предпочел деградации смену профессиональной ориентации. Тогда-то меня и посетила впервые эта замечательная мысль - подсоединить шнурок к телу. Не то чтобы прежний шнурок меня не устраивал - у меня его просто не было. Для этого мне не понадобилось обращаться к хирургу: настоящий жестянщик сам себе хирург. Однако довольно лирики, лучше определимся, стоит ли читать статью "этого извращенца", то есть меня, дальше. Прежде чем сесть за написание этой статьи, я проштудировал архивы журналов "Хакер" и "Хакер Спец" за последние два года. Утверждать, что за это время ничего толкового по подключению телефонов к компьютеру не было, значит изначально навлечь на себя гнев постоянных читателей и обрести врагов в лице коллег по цеху. Поэтому я не буду критиковать статьи по этой тематике за прошедший период времени, а лишь упомяну те, которые меня задели за живое. Таких статей ровно одна штука - "Дело - труба" небезызвестного автора Skylord'а ("Хакер" №53, май/2003). Саму статью ты можешь найти в архиве на сайте, я лишь скажу, что именно она стала "отправной точкой" для опубликованного ниже сабжа. Даже больше: я настоятельно рекомендую ознакомиться со статьей Skylord'а, потому как некоторые аспекты моего поведения будут понятны только вкупе с ней. Именно в этой статье Skylord категорически правильно ответил на вопрос "зачем?", поэтому я постараюсь ответить лишь на вопрос "как?". А ты и только ты сможешь объективно оценить, насколько хорошо или плохо я это сделал.

ОБЪЕКТЫ ИЗДЕВАТЕЛЬСТВ

Конечно же, данная статья подразумевает наличие у тебя какого-нибудь тела и компьютера. Если с последним все более-менее понятно, то про объект нашего издевательства, телефон, думаю, стоит рассказать поподробнее. Первым сотовым телефоном, над которым я надругался, стал Nokia 3310. С учетом его невысокой цены в настоящее время и отличного сигнального тракта именно его я рекомендую в качестве подопытного кролика независимо от того, есть у тебя тело или нет. Однако ограничиться только моделью Nokia 3310 означает сильно сузить целевую аудиторию данной статьи. Ведь мы-то все на ixbt.com ходили, "МС" читаем и знаем, какие марки телефонов пользуются популярностью у народа... Поэтому я считаю, что не упомянуть про линейку телефонов Siemens будет слишком жестоко. Ну а если я и Siemens затронул, то без полусамсунгов-недосименсов Siemens STxx вообще не обойтись. А что же делать в таком случае читателям, скажем, с французскими Alcatel'ами или корейскими LG? Читать еще внимательней статью, не смотреть на фото и ознакомиться с приатаченой к диску дополнительной литературой, ибо по прочтению статьи ты сможешь подключить к компьютеру любое тело. Даже то, у которого нет отверстия ни спереди, ни сзади.

Телефон может иметь следующие интерфейсы связи с окружающим миром: радиоканал Bluetooth, канал инфракрасной связи IrDA и канал проводной связи по какому-нибудь последовательному порту. Что касается первых двух, то для жестянщика они не интересны в принципе. С Bluetooth ни вардрайвинг замутить, ни прошивку поменять, ибо и радиус маленький, и риск загубить тело какой-нибудь из ниоткуда появившейся помехой слишком велик. То же можно сказать про IrDA, а если учесть ее уверенную связь в пределах прямой видимости да на паре-тройке дециметров, то вообще можно сделать вполне логичный вывод, что кроме закачек порнокартинок из интернета этот канал больше ни на что и не годен. Вот и получается, что как ни крути, как ни извращайся с этим хай-теком, ничего практически полезного из него не выжать. А провода как "рулили" десять лет назад, так и рулят по сей день, ничуть не уступая своего главенствующего места в мобильнотелефонном мире. Вот потому-то до сих пор и оснащают производители сотовых телефонов любой, даже самый навороченный смартфон обыкновенным отверстием в надежде, что пользователь чего-нибудь в него "привсунет". И каждый производитель выдумывает для своей линейки телефонов свой собственный вариант распайки разъема последовательного порта, по его (производителя) мнению, удобный для глумления над телом. А уж до чего причудливы порой бывают материальные воплощения фантазий дизайнеров на этом поприще... И лишь только немногие производители придерживаются хоть каких-то стандартов. Однако справедливости ради стоит заметить, что в пределах одного производителя, как правило, существует хоть какая-то стандартизация и унификация распайки и форм-фактора разъема. Как правило, но не всегда. Скажем, горячим финским парням из Nokia вообще "по большому зеленому барабану" на этот форм-фактор, ибо как иначе объяснить столь мозолящее глаз различие в цоколевке даже в пределах двух соседних (по времени запуска в серийное производство) модельных рядов? И даже Siemens, столько времени хранившая верность своему разъему, опорочила свою репутацию, связавшись с Samsung'ом. Плод этого греха получил имя STxx, и на момент написания статьи он представлен двумя версиями: ST55 и ST60. Но чтобы у тебя не

Рис. 00: USB-шнурок

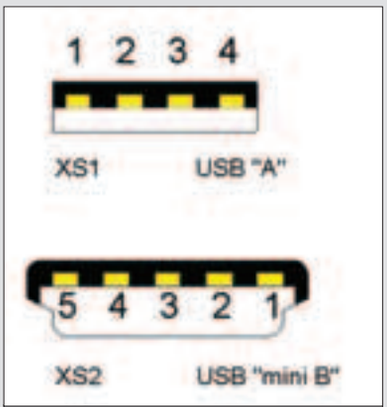


Рис. 01: Распайка USB-разъемов

Рис. 02: USB-преобразователь на Prolife PL-2303

возникло ни капли подозрения о моей предвзятости по отношению к этим двум моделям (ST55 и ST60), спешу заверить, что с ST60 я сожительствовал не один месяц, и, в целом, тело мне понравилось. Да и вообще цель этой статьи - не обзор достоинств и

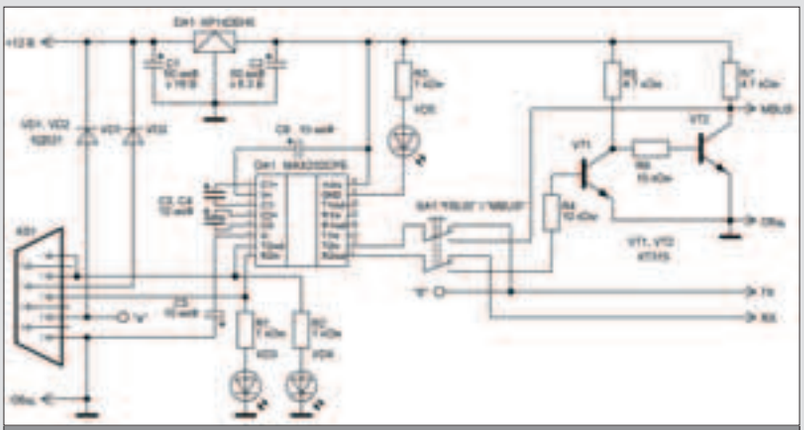


Рис. 03: Схема преобразователя RS-232<->ТТЛ

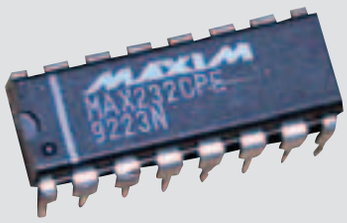


Рис. 04: MAX232CPE

недостатков той или иной модели, поэтому не будем больше отвлекаться и займемся непосредственно шнурками.

ПРО ШНУРКИ

Проще всего подключить к компьютеру телефоны фирмы Motorola. Они, как правило, придерживаются общемировых стандартов и снабжают свои детища розеткой USB. За примерами далеко ходить не надо: C250, C330, C450, C550, V150. Что не может не радовать. А посему обладатели таких моделей могут смело дальше не читать, ибо для изготовления сего шнура не нужно прикладывать никаких телодвижений: в любом компьютерном магазине тебе без проблем продадут то, что изображено на рис. 0. А когда дух жестянщика бунтует и противится, предлагаю без излишних прелюдий взглянуть на рис. 1.

Если уж мы заговорили о USB, то без упоминания о "пролайфиках" - однокристальных преобразователей USB-RS232 - здесь точно не обойтись. Мне не хотелось бы показаться "подстрекателем" и призвать народ засунуть в темное место паяльник и купить за $15-20 этот девайс, однако поддержка ядром Linux-сабжа заставляет, как минимум, задуматься об этом. Сам сабж представлен на рис. 2. Собственно, только линуксоидам я его и рекомендую, ибо мы и так Богом обижены :-). В среде ОС Windows преимуществ от него не ощущается (например тонкой настройки скорости общения с телефоном, которая больше или равна 115 кБит). А если еще учитывать совершенно непонятный для меня алгоритм общения с драйвером (который имеет склонность меняться от версии к версии драйвера), то целесообразность (и безопасность при перепрошивке) применения ставится под большим ракообразным вопросом. Тем не менее, я пользовался этим девайсом и проверил на себе его работоспособность. Однако в домашних условиях, приближенных к боевым, изготавливать крайне не рекомендую, ибо только стоимость микросхемы PL-2303 можно поставить под все тот же вопрос. Есть еще вариант на микросхеме CP1201, однако трудоемкость изготовления на ее базе также не оправдывает вложенных в устройство средств. Если уж и изготавливать шнурок самостоятельно, то он, как минимум, должен быть на не специализированных элементах, которые можно достать если не на помойке, то уж в самом отстойном магазине Урюпинска. Вот тот, что изображен на рис. 3, вполне подходит под это требование. Преобразователь уровней сигналов выполнен на микросхеме MAX232CPE (рис. 4) от буржуйской конторы MAXIM. Почему же "преобразователь уровней", если мы говорим об обычном порте RS-232? Да он только со стороны компьютера обычный с лог.1, равной -12 В, и лог.0, равным +12 В соответственно. А со стороны телефона эти уровни сигналов по меньшей мере неразумны. Представь, за сколько времени "садился" бы аккумулятор сотового телефона, имеющий такой сигнальный тракт. О перепрошивке или мобильном интернете с ноутбука тогда можно было бы вообще забыть или же носить с собой внешний источник питания в виде двух чемоданов аккумуляторов от КАМАЗ'а. А так и овцы сыты, и волки целы :). Собственно, сама микросхема MAX232CPE и есть преобразователь уровней. Полный datasheet на всю серию MAXIM'овских преобразователей ты найдешь на диске, а пока рекомендую взглянуть на рис. 5, где показано ее внутреннее устройство. Как показывает рис. 5, микросхема состоит из двух пар таких преобразователей. 2 ТТЛ в RS-232 и 2 RS-232 в ТТЛ. А так как нам для счастья много не нужно, то в схеме на рис. 3 и использовано ровно по одному. Переключателем с фиксацией SA1 мы выбираем, с каким телефонным интерфейсом будем работать. А! Я же тебе еще не расска-

зал, что их бывает несколько! Да, целых два - FBUS и MBUS. Собственно, все различие заключается в том, что в телефонном интерфейсе FBUS сигналы приема и передачи (RX и TX соответственно) разделены на две линии. Ну а в интерфейсе MBUS такого разделения нет, а потому сигналы RX и TX передаются по одному проводку. Как это осуществляется в телефоне, тебе, в принципе, простительно и не знать, а в преобразователе на рис. 3 это осуществляется каскадом на транзисторах VT1 и VT2. Светодиоды VD3 и VD4 указывают своим свечением, что происходит именно в данный момент (прием или передача данных), ну а VD5 просто информирует о том, что девайс включен и готов к труду и обороне. Вот на этом слове можно было закончить описание преобразователя и спокойно идти на заслуженный отдых, однако, мне кажется, что у тебя осталось несколько неразрешенных вопросов. Попытаюсь разъяснить их.

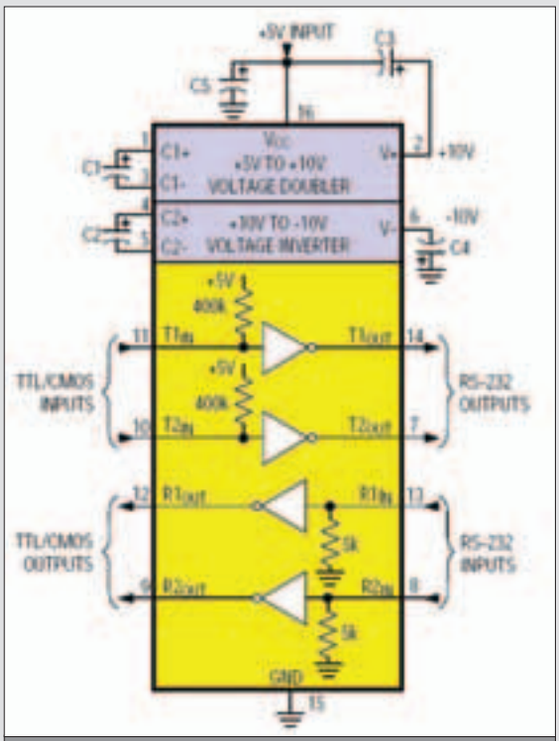


Рис. 05: MAX232CPE. Вид изнутри

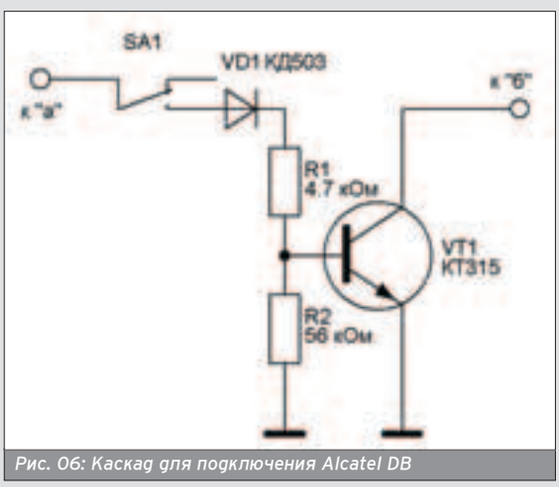


Рис. 06: Каскад для подключения Alcatel DB

Вопрос на сто рублей: что это за клеммы "а" и "б", подписанные автором, прикинувшимся Штирлицем. Чтобы не городить на девайс лишних деталей, которые, возможно, тебе никогда не пригодятся, я не стал включать в преобразователь узел согласования с французскими телефонами Alcatel DB. Тем более зверь это вымирающий, распространен в почти вырубленных лесах России мало. Однако возможность использования преобразователя уровней для связи такого телефона с компьютером оставил. Для этого лишь необходимо задействовать каскад, показанный на рис. 6.

А вообще-то изначально преобразователь интерфейса изготавливался под мою Nokia 3310 (рис. 7). Это не значит, что он не будет работать с другими телефонами - будет и еще как. Однако модель я упомянул не случайно. Дело в том, что данная модель телефона (а также некоторые другие) использует оба интерфейса: как MBUS, так и FBUS. Не одновременно конечно, но использует. Соответственно, и в некотором софте присутствует возможность выбора нужного интерфейса, а где такового не имеется, то тут уж тебе ручками придется выбрать правильное положение переключателя SA1 на схеме рис. 3. Однако если ты завел этого зверя недавно или только собираешься, рассмотрев перед сном витрину магазина, то можешь уличить меня во лжи. Можешь, не моргнув глазом, сказать: "Нет такого разъема у этого тела ни спереди, ни сзади". Да, действительно, это так: он у него... внутри и расположен под аккумулятором! Сие чудо представлено на рис. 8. Конечно, лучше воспользоваться "фирменным" разъемом, то есть тем, который разрабатывался исключительно под это тело. Его можно приобрести за пачку сигарет не только в сервисном центре у бородатого дядьки, но и на помойке рядом с этим сервисным центром. Какой способ лучше - решай сам. Я же в свое время вообще использовал кусок фольгированного текстолита и четыре медных сапожных гвоздя с резиновой направляющей. Фотография разъема промышленного образца дана на рис. 9.

Естественно, собирать сабж мы будем не на картоне, прикручивая гнилыми проводами без изоляции, а на печатной плате из фольгированного стеклотестолита толщиной 0,8-1,0 мм. Размеры платы - 65 мм x 50 мм. Шаблон для печатного монтажа с разрешением 600 dpi ты найдешь на диске, а пока предлагаю посмотреть на рис. 10. Первое, на что стоит обратить внимание - элементы красного цвета. Именно они изображены на рис. 3. Ну а элементы фиолетового цвета - соответственно для схемы на рис. 6. Эта часть нуждается только в одном комментарии: дабы не мудрить с еще одним переключателем, который еще неизвестно пригодится когда-нибудь или нет, я ввел обыкновенную контактную тройку под перемычку. Остальные компоненты в особом представлении не нуждаются. В качестве переключателя SA1 использован уже знакомый П2К с фиксацией положения контактной группы. Конденсаторы на 10 мкФ - любые танталовые бескорпусные ("сопли"), например фирмы TREC. Остальные - отечественные К50-35. Резисторы - любые, подходящие по габаритам и сопротивлению, мощностью не менее 0,125 Вт. Транзисторы - любые из серии КТ315, диоды КД521. В качестве VD5 лучше всего использовать светодиод красного цвета свечения, остальные - зеленого или желтого. Исполнение той части тракта RS-232, которая соединяется с разъемом XS1, остается на твое

Рис. 07: Nokia 3310

Рис. 08: Подкравшись внутрь, сзади

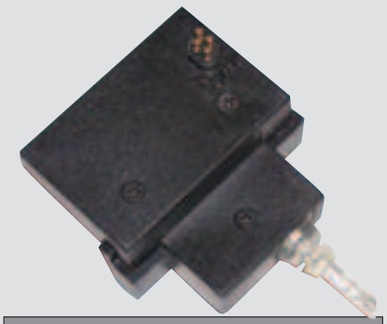


Рис. 09: Разъем для Nokia 3310

усмотрение: хочешь, напрямую провода припаяй, а хочешь - сделай правильно. Под "правильно" я подразумеваю использование стандартного шлейфа RS-232 на AT-плату, в котором DB-9M (папа) заменен на DB-9F (мама). Шестой вывод десятиконтактной розетки должен остаться свободным. Расположение проводников представлено на рис. 11. Распайка разъема для подключения к телефону (ТТЛ-часть преобразователя) дана на рис. 12, вид со стороны телефона.

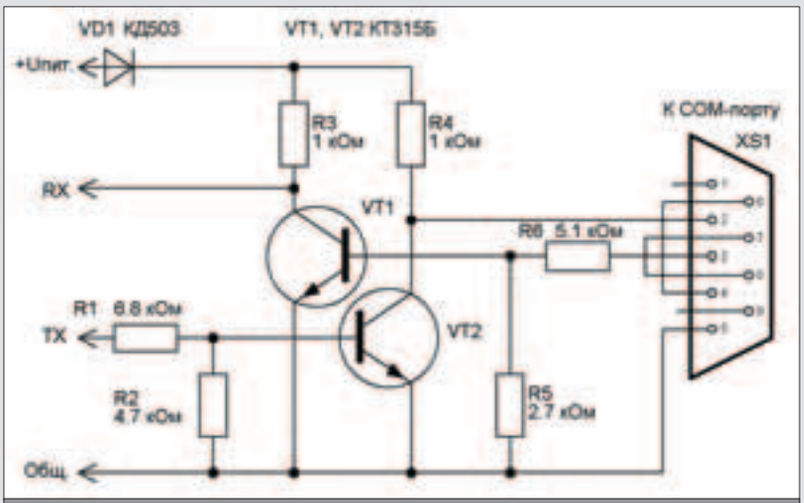


Рис. 13: Схема преобразователя с одним интерфейсом

ВМЕСТО СЛАДКОЙ ВАТЫ

Чего всегда не любил делать - катать напоследок вату о крутости и пользе сего девайса. Это тебе и дядька Google расскажет, ты только волшебное слово с моделью телефона скажи. Лучше расскажу, какое схемотехническое решение я использовал для подключения своего Siemens ST60. Дело в том, что не стоит городить такую схему, как на рис. 3, если твой телефон использует только одну FBUS. Это не обязательно должен быть ST60, работоспособность девайса проверена на C60 и S65, а следовательно, на всей серии, о которой упоминал Skylord. Можно, конечно, упростить ту, что на рис. 3, однако настоящий жестянщик пробует несколько схемотехнических решений, дабы выбрать лучшее. Это я тебе и предлагаю сделать, взглянув на рис. 13. Собственно, аналогичный каскад был использован в схеме картоприемника, поэтому затруднений в изготовлении этого девайса у тебя возникнуть не должно. Тут, как видишь, их два. Девайс питается от напряжения 5..12 В. Детали аналогичны предыдущему преобразователю (смотри рис. 14), поэтому предлагаю перейти непосредственно к конструкции сабжа. Он тоже изготовляется на плате из фольгированного текстолита толщиной 0,8-1,0 мм и размерами 40x20 мм. К сожалению, негативы (:-)) фотографии перед травлением засветились, что было замечено слишком поздно, а жаль, потому как посмотреть было на что. Дело в том, что плата рисовалась не как обычно - лаком, а красным маркером Digitex для нанесения надписей на CD. Конечно же, рисунок пришлось наносить в несколько слоев, однако я рекомендую этот способ начинающим жестянщикам. Впрочем, ты и сам уже знаешь, как лучше рисовать платы, а потому смотри на расположение компонентов на рис. 15 и на трассировку платы (рис. 16) и рисуй, рисуй, рисуй... В итоге у тебя должно получиться нечто, примерно похожее на то, что представлено на рис. 17. Кстати, размеры платы случайно (;-)) совпали с размерами корпуса под горелый "пролайфик". Думаю, этим нельзя не воспользоваться. И напоследок пара слов о разъеме. В этом случае для соединения с телефоном я использовал разъем от зарядного устройства. Проблема нехватки нужных контактов решается с помощью консервной банки и ножниц. Ты что это меня слушаешь, а паяльник еще не включил? Смотри, тогда я иду со своим... Нагретым... Ага, ориентацию поменяем... ;-).

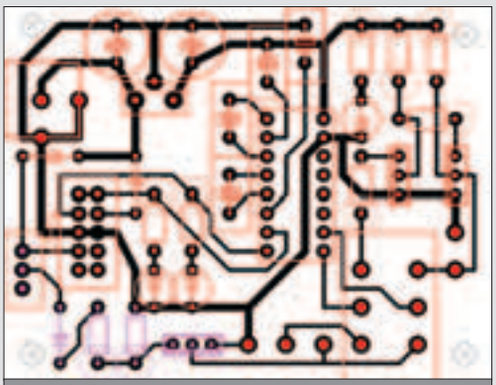


Рис. 10: Расположение элементов на плате

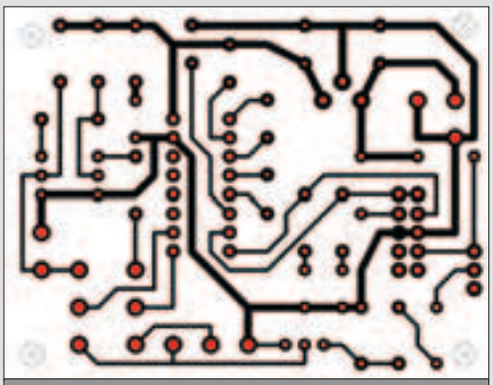


Рис. 11: Расстрассированные проводники

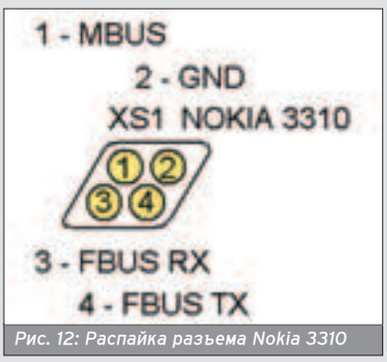


Рис. 12: Распайка разъема Nokia 3310

Рис. 14: Детали преобразователя с одной FBUS

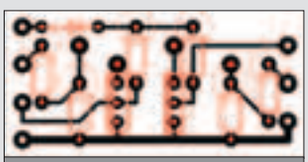


Рис. 15: Расположение компонентов

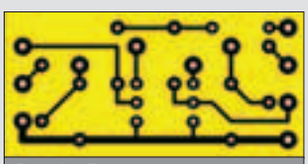


Рис. 16: Трассировка платы

Рис. 17: Результат

ЗАКАЗ ЖУРНАЛА В РЕДАКЦИИ

Бесплатный телефон по всем вопросам подписки для регионов:
8-800-200-3-999
(в том числе для абонентов МТС, Билайн, МегаФон), для Москвы:
935-70-34

ВЫГОДА
Цена подписки на 20% ниже, чем в розничной продаже
Бонусы, призы и подарки для подписчиков
Доставка за счет редакции

ГАРАНТИЯ
Ты гарантированно получишь все номера журнала
Единая цена по всей России

СЕРВИС
Заказ удобно оплатить через любое отделение банка
Доставка осуществляется заказной бандеролью или курьером

Стоимость заказа на Хакер Спец

115 р. на один месяц (экономия 40 рублей*)
690 р. на 6 месяцев (экономия 240 рублей*)
1242 р. на 12 месяцев (экономия 620 рублей*)

Стоимость заказа на комплект Хакер Спец и Хакер**

207 р. комплект на один месяц (экономия 85 рублей*)
1242 р. комплект на 6 месяцев (экономия 510 рублей*)
2236 р. комплект на 12 месяцев (экономия 1250 рублей*)

+

*от средней розничной цены по Москве
**Хакер с 2CD или Хакер с DVD

ЗАКАЖИ ЖУРНАЛ В РЕДАКЦИИ И СЭКОНОМЬ ДЕНЬГИ!

ПОДПИСНОЙ КУПОН

Прошу оформить подписку:

- ☐ на журнал Хакер Спец
- ☐ на комплект Хакер Спец и Хакер с DVD
- ☐ на комплект Хакер Спец и Хакер с 2CD

на ☐ месяцев

начиная с _____ 2005 г.

- ☐ Доставлять журнал по почте на домашний адрес
- ☐ Доставлять журнал курьером на адрес офиса (по г. Москве)

Подробнее о курьерской доставке читайте ниже*

(отметьте квадрат выбранного варианта подписки)

Ф.И.О. _____

дата рожд. ☐☐. ☐☐. ☐☐ г.
(день . месяц . год)

АДРЕС ДОСТАВКИ:

индекс _____

область/край _____

город _____

улица _____

дом _____ корпус _____

квартира/офис _____

телефон (код) _____

e-mail _____

сумма оплаты _____

* Курьерская доставка осуществляется только по Москве на адрес офиса. Для оформления доставки курьером укажите адрес и название фирмы в подписном купоне.

Извещение

ИНН 7729410015 ООО «Гейм Лэнд»

ЗАО Международный Московский Банк, г. Москва

р/с № 40702810700010298407

к/с № 30101810300000000545

БИК 044525545 КПП - 772901001

Плательщик _____

Адрес (с индексом) _____

Назначение платежа	Сумма
Оплата за «_____» с _____ (месяц) 2005 г.	

Ф.И.О. _____

Подпись плательщика _____

Кассир

Квитанция

ИНН 7729410015 ООО «Гейм Лэнд»

ЗАО Международный Московский Банк, г. Москва

р/с № 40702810700010298407

к/с № 30101810300000000545

БИК 044525545 КПП - 772901001

Плательщик _____

Адрес (с индексом) _____

Назначение платежа	Сумма
Оплата за «_____» с _____ (месяц) 2005 г.	

Ф.И.О. _____

Подпись плательщика _____

Кассир

Как оформить заказ

1. Заполнить купон и квитанцию
2. Перечислить стоимость подписки через Сбербанк
3. Обязательно прислать в редакцию копию оплаченной квитанции с четко заполненным купоном любым из перечисленных способов:

- по электронной почте: subscribe@glc.ru;
- по факсу: 924-96-94;
- по адресу: 107031, Москва, Дмитровский переулок, д. 4, строение 2, ООО «Гейм Лэнд», отдел подписки.

ВНИМАНИЕ!

Подписка оформляется в день обработки купона и квитанции.

- купоны, отправленные по факсу или электронной почте, обрабатываются в течение 5 рабочих дней.
- купоны, отправленные почтой на адрес редакции обрабатываются в течение 20 дней.

Рекомендуем использовать электронную почту или факс.

Подписка производится с номера, выходящего через один календарный месяц после оплаты. Например, если произвести оплату в сентябре, то подписку можено оформить с ноября.

По всем вопросам по подписке звони бесплатно по телефону 8-800-200-3-999 (для регионов, в том числе с мобильных телефонов МТС, Билайн, Мегафон) или 935-70-34 (из Москвы),
Вопросы по подписке можно задавать по e-mail: info@glc.ru

Подписка для юридических лиц

Москва: ООО «Интер-Почта», тел.: 500-00-60, e-mail: inter-post@sovintel.ru
Регионы: ООО «Корпоративная почта», тел.: 953-92-02, e-mail: kpp@sovintel.ru
Для получения счета на оплату подписки нужно прислать заявку с названием журнала, периодом подписки, банковскими реквизитами, юридическим и почтовым адресом, телефоном и фамилией ответственного лица за подписку.
www.interpochta.ru

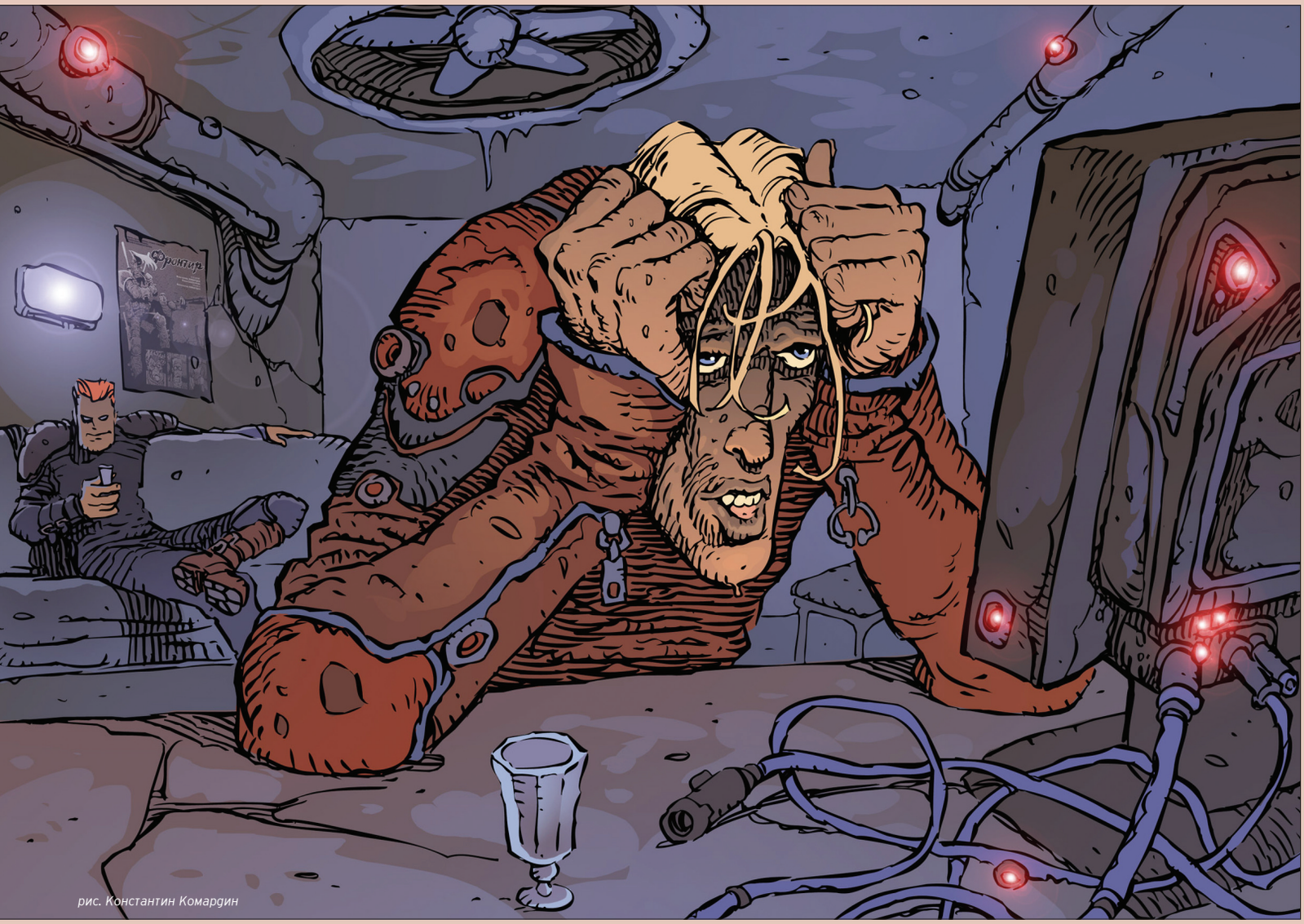


рис. Константин Комардин

Niro (niro@real.xakep.ru)

HAPPY BIRTHDAY TO YOU...

3

Человек за компьютером был весел: его пальцы постукивали по клавиатуре в такт звучащей из колонок музыке. Перед ним на мониторе разворачивалась история - глупая, смешная история. Временами глаза наблюдателя смотрели на индикатор записи: все, что происходило сейчас на экране, записывалось в файл.

- Будет чем похвастаться сегодня, - промурлыкал он себе под нос, когда события на экране стали развиваться особенно бурно. - Начальство будет довольно.

Он подрегулировал резкость и яркость, откинулся на спинку кресла и положил ноги на стол рядом с мышкой и чашкой с остатками чая. Руки сами собой сложились крестом на груди. Поза была до нельзя удобной. Человек прикрыл веки, но прежде чем задремать, протянул руку к колонке и приглушил звук. Содержание его в настоящий момент не интересовало: оно ему было просто ни к чему. Скоро он ровно задышал, сон смежил его веки. А на экране продолжалось действо...

* * * * *

Тарелка с салатом была уже на две трети пустой. Фомин тупо возил в ней ложкой, стараясь зацепить несколько горошин сразу, но это у него не получалось. Правда, степень его упорства давно перевалила за сто процентов, остановиться он уже не мог и был обречен умереть от голода возле этой тарелки. Ложка выделывала немыслимые круги, разрисовывая все вокруг майонезом, словно мороз на стекле окна. Дыхание было тяжелым, будто весу в ложке было не меньше ста килограммов. Фомин облизывал губы и придерживал предплечье свободной рукой, пытаясь прицелиться. Горошины, как назло, вываливались из ложки именно в тот момент, когда он был готов подхватить очередную, стараясь довести их число в ложке до десяти.

- Ух ты... - шептал он себе под нос. - А вот... Блин...

Задача была не из легких. Учитывая тот факт, что две трети тарелки съел не он, можно было предположить, что от голода он действительно скоро умрет. Казалось, что десять горошин в ложке - это некое абсолютное условие существования, без которого дальше жить и уж тем более есть, было просто невозможно.

Обстановка вокруг Фомина располагала к долгой ловле горошка. Начнем с того, что в комнате он был совершенно один. Никто не торопил его, не заставлял делать все быстро. Никто не толкал под локоть, не просил передать соль или плеснуть еще водки в рюмку.

Вокруг не было никого.

- Сейчас... Вот еще чуть-чуть...

Он высунул язык, словно ребенок, увлеченный сверхважным делом. Ложка выделывала немыслимые пируэты, словно приклеенная к горошинам в салате. И когда последняя из них уже была готова вот-вот сорваться в мельхиоровую западню, в дверь кто-то позвонил.

Фомин вздрогнул, растеряв половину того, что уже поймал к этому моменту, потерял из виду ту, за которой гонялся последние минут десять, грохнул кулаком по столу и пошел открывать. По пути он задел ногой пару стульев, расставленных вдоль стола, едва не уронил бокал, стоящий на углу, и выдал в воздух пару матерков в адрес того, кто пустил насмарку его занятие.

За дверью стоял Петя, или, как он любил называть себя еще со студенческих времен, Петр Иваныч. Почему он требовал ото всех обращения "по-взрослому", было тайной, но никто особо не перечил, ибо звучало это довольно прилично.

- Не ждал? - прищурившись, спросил Иваныч, опираясь одной рукой о косяк. - Еще осталось что-нибудь?

Фомин кивнул.

- Фома, ты чего? - наклонив голову, спросил Иваныч. - Ты при памяти? Я войду?

Фомин в очередной раз кивнул и отошел в сторону, пропуская гостя, после чего закрыл дверь и махнул рукой в сторону комнаты.

Петя разулся, не отрывая взгляда от хозяина квартиры, в которую вошел, и не переставая ухмыляться. Видно было, что Фомин изрядно пьян, но, по разумению Петра Иваныча, это сегодня было само собой разумеющееся состояние. Вошли они в комнату практически одновременно, едва пропихнувшись в двери. Петр осмотрел комнату и присвистнул.

- Никого?! Все-таки они сделали, как сказали? Ну, парни... Я думал, они шутят. Хотя такие практически никогда не шутят...

Фомин скривил губы и посмотрел в сторону тарелки с салатом.

- Нет, почему же, был тут один... Сожрал оливье... Почти все сожрал. По-моему, это был кто-то из группы Гифа... Чего-то не помню, он представился то ли именем, то ли ником, хрен их разберет.

- Драки не было? - поинтересовался Петр, уже по-другому рассматривая лицо Фомина и выискивая на нем следы битвы за правое дело.

- Нет, - отмахнулся Фома. - А надо бы... То есть можно было бы мне и по морде дать. Я бы не против.

- Могу дать, - улыбнулся Иваныч. - Но не считаю нужным.

Пройдя вдоль стены, Петр опустился на стул рядом с телевизором, взял с него пульт, ткнул в кнопки. Экран засветился, обрадовав Иваныча кровавыми кадрами "Дежурной части".

- Знаешь, здесь вот (Петр махнул пультом в сторону экрана) все гораздо круче, чем в "Крестном отце" Марио Пьюзо. Поэтому твоя проблема - это ничто.

Фомин скривил губы, тупо вглядываясь в происходящее на экране. Кто-то, очевидно, понадеявшись на авось, пытался сбыть партию фальшивых долларов. Получалось это из рук вон плохо: практически с самого начала (стоило ему только проснуться и подумать о том, как бы совершить преступление) ему впарили массу скрытых камер, следили за каждым его шагом, все записывали и фотографировали. Короче, очень грустный получался Голливуд. Кончилось все тем, что парню посреди города заломили руки, ткнули мордой в капот и радостно сообщили о том, что еще одной сволочью стало меньше.

Иваныч время от времени переводил глаза с телевизора на Фому и обратно. Хозяин квартиры был изрядно пьян, но чувствовалось, что он анализирует ситуацию еще очень даже четко. Когда сотрудник милиции, стоя спиной к камере, рассказывал об успехах своей опергруппы, Фомин тоскливо кивнул сам себе, встретился взглядом с Петром и пробрался к своему месту за столом.

- Здорово, ничего не скажешь, - сказал он в тарелку с салатом. - Правда, не думаю, что ты знал, что именно покажут по ящику, когда включал его. Совпадение?

Ну, парни... Я думал, они шутят. Хотя такие практически никогда не шутят...

- Сто процентов, - ответил Иваныч, разглядывая стол. Чувствовалось, что он очень даже не против перекусить. - Собственно, сам не понимаю, чего меня дернуло взяться за пульт.

- Судьба, - согласился Фомин. Протянув руку к бутылке водки, он тяжело поднял ее, плеснул в рюмку, что стояла рядом с ним, после чего вопросительно взглянул на Петра.

- Запросто, - ответил тот, быстро сориентировавшись и поставив еще одну рюмку под готовую рвануться из бутылки струю. Фома наливал будто из брандспойта - не заботясь о том, сколько разольет на скатерть. Иваныч особо не контролировал процесс: ну, мимо так мимо. Хозяин-барин.

Поставив рюмку возле себя, Петр накидал в пустую тарелку несколько кружков копченой колбасы, протянул руку за нарезанным сыром и вдруг углядел на дальнем конце стола салат из крабовых палочек с кукурузой.

- Фома, это выше моих сил, - демонстративно облизываясь, сказал он. - Если ты не против, я пересяду.

- Только телевизор выключи, - задумчиво кивнул Фома. - А то мало ли: вдруг покажут еще что-нибудь подобное, так я не выдержу, глупостей наделаю.

- Каких же, если не секрет? - спросил Иваныч, нажимая кнопку на пульте и примериваясь, как бы получше обойти стол. - Тарелкой в экран?

- Мелко, - отрицательно покачал головой Фомин. - Телевизор тут ни при чем. Виноват тот, кто его включил...

- Или тот, кто его купил, - согласился Петр, накладывая себе несколько полных ложек салата. Фома поднял на него усталый взгляд, нахмурил лоб, словно пытаясь понять, кто же на самом деле купил этот чертов телевизор. Спустя пару секунд он оставил все попытки, опрокинул в себя рюмку, сморщился, прижав рукав к лицу. Свободная рука сама потянулась к вилке, нацепила кружок колбасы.

- Ну, - закусив, Фомин посмотрел на Петра, - спрашивай.

- О чем? - не отрываясь от крабовых палочек и кукурузы, спросил Петр.

- Сам знаешь о чем.

- А тост? Не нужен? - поинтересовался Иваныч, контролируя одной рукой рюмку на столе. - Как-никак...

- Короче. Тост - и потом спрашивай.

Петр встал, откашлялся и, оглядев пустые стулья по периметру, сказал:

- Я, Фома, знаю тебя уже много лет... Достаточно много. Восемь. Хочу сказать, что... Понимаешь, мы уже перешли некую границу. А за ней все иначе... »

- Ты с кем сейчас разговариваешь? - внезапно спросил Фомин, наклонив голову в сторону. - Тост где? Я уже наливаю.

- Решил напиться? - укоризненно спросил Петр. - Они этого не любят.

- "Они"? Что ты про них знаешь? - Фомин вертел в руках бутылку, словно измеряя объем.

- Извини, Фома, ладно... Это я чего-то загнул. Пофилософствовал. Если можно, я продолжу. Хочу сказать в этот день, день твоего, Фома, тридцатилетия, хочу пожелать, хочу...

- Спасибо, - кивнул Фомин и выпил. - В такой обстановке лучшего тоста и не придумаешь...

Петр пожал плечами и последовал его примеру, после чего опустился за стол и принялся молча пережевывать колбасу. Интерес к крабовым палочкам он потерял.

В комнате стало тихо. По разные углы стола сидели два человека, которые очень хотели бы поговорить друг с другом, но одному не позволяло упрямство и алкоголь, а второму - смятение и грусть. Периодически они исподлобья кидали друг на друга взгляды, но ни разу не встретились глазами, так что повода заговорить пока не было. Фома откинулся на спинку стула и прикрыл глаза, его посетило нечто вроде нирваны, он расслабился и был готов задремать, но внезапно Петр не выдержал и спросил:

- Так все-таки - что толкнуло? Деньги?

Фомин открыл глаза и посмотрел куда-то прямо перед собой, словно пробуя на вкус каждое произнесенное Иванычем слово.

- Деньги - это, Петя, фигня, - спустя пару секунд ответил он. - Пожалуй, даже не так. Это я еще загнул. Никакая не фигня, а вообще ничто. Вещь, не имеющая в данной ситуации никакого веса. У моего поступка нет цены в денежном эквиваленте.

Петр недоумевая посмотрел на Фому и не нашелся, что сказать.

- Понимаю, Иваныч, твое недоумение, прекрасно понимаю. Сам чувствовал то же самое, когда принимал решение.

Иваныч развел руками. После упоминания о "жучках" ему вообще расхотелось говорить вслух.

- Неужели ты... По идеологическим соображениям? - произнес Петр. - Неужели сам?

- Кто из нас пьян, не пойму, - ухмыльнулся Фома. - Ты сам-то понял, что сказал? "По идеологическим..." Это ж полная чушь! Какая тут может быть идеология? Я же их всех ненавижу!

Он встал, покачнувшись, подошел к окну и ткнул в него пальцем.

- Здесь все принадлежит им! Мы даже дышим до сих пор бесплатно, потому что они решили, что им пока хватает! А как захотят еще денег, так надо будет кислородные подушки в магазинах покупать втридорога!

Он с досады махнул рукой и хлопнул ей по оконной раме. С его губ сорвался тяжелый вздох.

Петр молчал, ничего не понимая. От его полной готовности поговорить с Фомой не осталось и следа. Он тоскливо осмотрел стол и понял, что его занимает сейчас один вопрос: как пополнить рюмку. Фома словно почувствовал это, направился к столу и по пути бросил взгляд на экран выключенного монитора, стоящего в углу комнаты.

И его словно взорвало.

- Да как ты мог подумать, что я добровольно пошел работать к ним?! Ты мог себе такое представить? Да мне бы это в страшном сне не приснилось! Надо же, а... Так вы, наверное, все так думаете? Да ну, скажи. Все?! Ведь почему-то же никто не пришел ко мне на день рожденья?!

- Нет, ну, я же пришел... - опешил от такой экспрессии Петр. - И ты говорил, что приходил еще кто-то...

- Знаешь, зачем он приходил? - почти вплотную приблизился к Иванычу Фома. - Чтобы сказать мне, что я самая последняя сволочь в этой жизни! САМАЯ! ПОСЛЕДНЯЯ! СВОЛОЧЬ! Твою мать... Да что вы понимаете в этой жизни, хакеры хреновы!

- Ничего, - мгновенно согласился Петр. Сказать ему пока было нечего. Судя по всему, тот парень, что приходил до него, был не особенно расторопен, если успел съесть за время беседы больше половины тарелки салата. Похоже, он исполнял тут некий обвинительный монолог, который ему поручили сказать ребята из его хак-группы.

- Тебе тоже поручили передать мне "черную метку"? - спросил Фома. Похоже, сегодня он был просто в ударе по угадыванию чужих мыслей. - Тебя послали сюда, как самого красноречивого? Или как самого бесстрашного? Или как-нибудь еще? Может, я чего-то не понимаю?!

- Фома, присядь, - только и сумел выдавить из себя Петр. - Сделай паузу, скушай "Твикс".

- Обхохочешься какой ты остроумный, - мотнул головой Фомин. Его тут же повело в сторону, он ухватился за спинку стула и едва не повалился на пол. Иваныч протянул руку и ухватил его за плечо. Фома, обретя равновесие, вырвался из цепких пальцев Петра, сделав это с презрительным выражением лица.

- Нечего меня тут поддерживать... Где вы раньше были, друзья липовые...

- Ты о чем, Фома? - спросил Петр. - Все друзья у тебя были настоящие...

- Вот именно "были". Вы все смелые каждый за своим ящиком. Кодерышмодеры. А когда до дела доходит, все в кусты. Да поглубже, поглубже, чтоб никто и никогда... "Это не я, меня там не было, я вообще ни при чем". Тьфу!

- О чем ты, Фома? - пожал плечами Петр. Он немного освоился с манерой хозяина излагать свои мысли, протянул руку за бутылкой, плеснул им обоим по полной рюмке, встал и подошел к Фомину, протягивая ему емкость. - Давай по маленькой, а потом ты чуть помедленней и чуть поподробней. Договорились?

Фомин кивнул, выпил и отказался от предложенного кусочка сыра. Петр все-таки настоял на своем, запихал его Фоме в рот, после чего вернулся на свое место. Фома же опустился в кресло возле компьютера. Закинув ногу на ногу, он застыл в какой-то нелепой позе, рассматривая дырку на носке.

- Ну как? - поинтересовался Петр. - Говорить можешь?

- Могу, - кивнул Фома. - Сейчас, только подумаю, с какого места начать... А кстати, компьютер не нужен? Хороший.

Он махнул рукой в сторону стоящего на столе монитора, после чего взглянул под стол, где стоял корпус со снятой крышкой. Лицо исказила какая-то жуткая гримаса злости и сожаления.

Иваныч отрицательно покачал головой и спросил:

- Не пользуешься?

- А как? - вопросом на вопрос ответил Фома. - Колпак - покруче бронированного. Я иногда думаю, что у меня дома камер и "жучков" понатыкано больше, чем в американском посольстве. Да не крути ты башкой... Страшно стало, что ли? Я думаю, что тебе и так давно уже все известно. Они, похоже, вообще всех моих друзей и знакомых прозондировали.

Петру явно стало не по себе. Он нервно отломил корочку хлеба и принялся разминать ее пальцами. Изредка он кидал взгляды в углы комнаты, но обнаружить хоть какие-то средства слежения ему не удалось.

- Не ищи. Я уже искал, и не так как ты - косыми взглядами. Я все конкретно делал - со сканером, с отверткой и плоскогубцами. Думал, найду, раздолбаю все к чертовой матери. Не нашел. Не судьба. А может, и нет тут ничего.

Иваныч посмотрел на Фому взглядом, в котором было все: жалость, сострадание, понимание, укор. Вся боль человечества - не иначе. И Фома, увидев его глаза, одними губами прошептал: "Я нашел. Жучок за монитором. Не стал трогать, пусть думают, что не знаю".

Иваныч, с губ которого был готов сорваться вопрос о дальнейшей жизни Фомы, замер. Фоме он верил: тот не стал бы ради шутки нагнетать обстановку. Он нашел один "жучок" - а где один, там может быть много. Поэтому в разговорах надо быть поосторожнее...

- Тебе по-прежнему интересно узнать, как все случилось? - сменив позу в кресле, спросил Фома. - Интересно узнать, как же низко пал ваш великий Джент, который два года руководил самой громкой и известной хак-командой? Хочешь окунуться в это дерьмо?

Иваныч развел руками. После упоминания о "жучках" ему вообще расхотелось говорить вслух.

- Зря молчишь, - сказал Фома. - Нет в мире радости больше, чем радость человеческого общения. Ладно, будем считать, что ты спросил.

Он встал с кресла, залез под стол - туда, где тихо пылился раскрытый и раскуроченный корпус. Нетвердыми пальцами он стал соединять там какие-то проводки, потом вытащил на свет винчестер, взглянул на расположение джамперов на нем, удовлетворенно покачал головой и засунул все обратно.

- Давай-давай, - ободрял он сам себя. - Время есть.

- Помочь? - спросил Петр, подойдя поближе.

- Не напрягайся. Лучше еще водки выпей, - раздалось из-под стола. - И мне налей, я тут недолго.

Петр подчинился, налил водки, взял рюмки и подошел к Фоме, встав у него за спиной. Тот постепенно перемещался куда-то совсем к стене, оставив снаружи только ноги. Иваныч смотрел на него сверху вниз, пытаясь понять, зачем Фоме все это понадобилось. Судя по всему, хозяин восстанавливал провода вдоль плинтуса, которые были кем-то оборваны.

Пару раз Фома попросил отвертку, потом изоленту. Петр отставил рюмки, понемногу втянулся в процесс, присел на корточки и стал разглядывать внутренности компа Фомы. Железки там были очень даже ничего, крутые. Фома вообще никогда не жалел денег на содержание машины - покупал все самое новое и фирменное. Начинки его компьютера (по цене) могло

хватить на оформление компьютерного класса в школе. Петр всегда поражался стремлению друга заполучать все самое лучшее.

- Знаешь, - донеслось откуда-то из-под стола, - я тоже думал о таких, каким стал сам: "Как они могут, блин? Дружба, идеалы, все такое... Как?" Ответ прост до безобразия: могут, да еще как.

Петр вертел в руках ненужную пока отвертку и понимающе кивал. Наконец, Фома выбрался обратно, отряхнул ладони, потом дополнительно вытер их о джинсы и удовлетворенно произнес:

- За это надо выпить.

Иваныч протянул ему рюмку. Фома покачал головой:

- Я бы и от бутербродика не отказался.

После чего встал и вернулся к столу. Стул скрипнул под ним, когда он тяжело опустился на прежнее место, сохраняя противостояние с Петром в три метра и следя за тем, чтобы масса тарелок, графин с соком и торт разделяли их, словно демаркационная линия.

- Я включу его чуть позже, - кивнул он в сторону компьютера. - А пока у нас есть время договорить.

Он пригладил волосы на голове и начал:

- Два года назад, если ты помнишь, я выдвинул бредовую идею о попытке заработать в виртуале живые деньги. Все, конечно же, были "за" и по общему согласию спихнули идею разработать план мне. Любили вы меня тогда, ценили и уважали...

Петр попытался вставить слово, но Фома жестом остановил его:

- Я знаю, Петя, что ты в моей группе никогда не был и о тех самых планах понятия не имел. Но ведь слишком велика тайна, Иваныч, чтобы ей делиться. Хакеры, Петя, как бабы. Секреты хранить могут, но не хотят. Поэтому по тем временам было все очень просто: чем меньше народу посвящено в то, чем мы занимаемся, тем лучше. Идея возникла да вроде бы и умерла тут же. Но в душу запала... А у меня, если помнишь, ничего так просто никуда не пропадало.

- Конечно, Фома, - согласился Петр, который все это время настороженно смотрел в сторону монитора, за которым находился найденный хозяином квартиры "жучок". - У тебя все было по полочкам разложено.

Фома напрягся, потер виски и продолжил:

- Тогда у меня был, наверное, пик формы. То, что мы делали, было просто здорово. Море информации, добытой для клиентов, удовольствие от происходящего - это заставляло думать дальше, искать новые формы. Нас было четверо...

Петр не выдержал и кивнул в сторону монитора.

- Чего? - не понял Фома, проследил взгляд гостя и отмахнулся. - Я же сказал "было". Их всех... Короче, уже можно. Так вот, нас было четверо, как мушкетеров. Я отвечал за планирование, программированием занимались два брата (мы их называли Чук и Гек), а аналитическую работу я отдал девчонке: была у нас тут одна, Жанна, бредила компами, но никак не могла коечему обучиться, не давали длинные ноги. Зато логики в ней было - на всех нас вместе взятых, она выводы делала быстрей, чем компьютер. И, знаешь, похожа была на Софию Ротару в молодости. Не поверишь - одно лицо. Чем она меня купила - этим, наверное... Но, Петр, скажу сразу, слабым звеном она не была, хоть и баба.

- Охотно верю, - согласился Петр, пытаясь отогнать всякие нехорошие мысли, которые появились, когда Фома сказал "Уже можно..." - Ты умеешь подбирать людей. Всегда умел и, думаю, не растерял это свое умение.

- Спасибо, - картинно поклонился Фома. Голова тяжело упала и с трудом вернулась в исходное положение. Чувствовалось, что Фомин изрядно пьян и только чудом сохраняет устойчивость за столом и ясность мысли (впрочем, довольно относительную: Петр видел это по затуманенному взгляду). - Приятно видеть рядом с собой понимающего человека...

Иваныч кивнул и немного поморщился. Пьяному любая лесть в радость. "Ладно, послушаем..."

- Продолжаю. Сконцентрируйся, Петя, - щелкнул несколько раз пальцами Фома. - Вот так. Пойми, цель у нас была высокая - заработать денег, сделать себя свободными материально. Не думай, что деньги портят человека - я всегда буду уверен в обратном, даже сейчас. Я просто устал от теперешней действительности и нашел еще трех таких же уставших, недовольных людей. И мы решили стать богатыми.

- Зачем? - спросил Петр. - То есть, я понял насчет великой цели... А что-то более приземленное было? Я имею в виду, на что хотелось потратить эти деньги?

- Конечно, было. Хотелось, как это ни банально, уехать. Причем, в первую очередь, уехать куда-нибудь посмотреть мир. То есть... Ну, ты понял. Я не хотел эмигрировать - я хотел стать вечным туристом.

- Сколько же денег вы хотели сделать? - приподнялся Петр, не в силах даже примерно прикинуть необходимую сумму. - Или все было наугад - и если бы не получилось, пришлось бы повторять свою работу до тех пор, пока не набралась бы нужная куча валюты?

- Иваныч, иди к черту! - выругался Фома. - Да кто ж их считает, когда они в руки сами плывут?! Вот если бы мне надо было под огонь автоматов идти, с вооруженной охраной какого-нибудь банка воевать, вот тогда бы я рассчитал и сумму, и риск и решил бы, стоит ли игра свеч. А тут-то чего париться? Сидишь себе в мягком кресле. Все, что работает - это мозги и пальцы. Да сколько бы их там ни было, всегда можно было взять еще!

- Что ж вы такого придумали? - спросил Петр. - А можно чайник поставить?

- Валяй, я тоже с удовольствием... Но чай не водка - много не выпьешь. Вот тут и торт кстати, не зря купил.

Петр пошел на кухню, унося в себе страх. Он не понимал, к чему Фома разоткровенничался. Сам он пришел к Фомину исключительно по старой памяти: они много лет были друзьями в реале, и поступок Фомы поставил Иваныча в тупик. Он решил понять своего друга, потому и пришел к нему тогда, когда все его бросили, проявив пренебрежение (в лучшем случае) и ненависть (в основном). Пока из крана в чайник наливалась вода, Петр думал о том, как будет продолжаться их разговор, но любопытство разгоралось в нем все больше и больше. Ткнув в кнопку на чайнике, он присел на кухне на табуретку и задумался...

* * * * *

Дремота временами уходила, как море в отлив. Глаза открывались наполовину, отмечая происходящее на экране. Вот один из них ушел куда-то из поля зрения. Рука сама щелкнула по кнопке, изображение на экране разделилось на две части. Та-ак. Второй ушел на кухню, включил чайник, присел. Судя по всему, будет ждать, когда же закипит. Беседа у них явно была какой-то натянутой: то ли они не понимали друг друг друга, то ли боялись.

Первый через несколько секунд встал со своего стула, взглянул в сторону двери, налил себе рюмку, вздохнул и выпил ее одним махом.

У наблюдателя непроизвольно рот наполнился слюной. Глаза стрельнули в сторону холодильника на изображении, в котором (он помнил это чет-

Хакеры, Петя, как бабы. Секреты хранить могут, но не хотят.

ко) стояли три бутылки пива. Начинать явно не стоило, однако то, с каким аппетитом сейчас закусывал огурчиком парень, очень и очень стимулировало.

- Нет, не надо поддаваться, - шепнул наблюдатель сам себе и прикрыл глаза. - Хрен с ними, с этими хакерами...

В общем, работа у него была - не бей лежачего.

* * * * *

Пока Петра не было, Фомин успел выпить две рюмки. Правда, нетвердость речи и покачивание пола под ногами заставили его, наконец, обратить внимание на еду. Во рту было жутко сухо. Он выпил полграфина апельсинового сока, даже не поморщившись, хотя полчаса назад в него не влез бы и стакан: настолько он терпеть не мог эту кислятину. Он и купил-то ее для гостей, вот только гостей он так и не дождался.

Его бросили все, с кем он был близок последнее время. Друзья, знакомые, фанаты. Слухи в их среде расползаются хоть и медленно, но въедаются в репутацию "виновника торжества" намертво. Изменить что-то в своей жизни он уже не мог, да и не хотел. Его сломали, сломали навсегда. Хорошо хоть оставили в живых...

Первое время Фомин жалел и об этом - уж слишком много было поставлено на карту, слишком велика была цена успеха или неудачи. Жить не хотелось совершенно объективно. Проигрыш он расценил как нечто страшное и непоправимое, смирился с этим и был готов умереть. Но его оставили жить. И он втянулся.

Тот, из группы Гифа, который пришел утром и внаглую слупил почти полную тарелку салата (кажется, его зовут Максим)... А впрочем, неважно, как кого зовут. Он был глашатаем. Он донес до Фомы весть о том, что друзей у него больше нет. Нет и вряд ли когда-нибудь появятся. Он жевал салат, запивал его соком и говорил, говорил... А Фома слушал, скрипя зубами, и ничего не мог поделать.

Он не мог даже просто попросить его выйти. Максим должен был сказать все, поставить все точки над i в этой гнусной истории. Вытирая рот салфеткой, Максим порой произносил слова невнятно, небрежно, но все было понятно и без слов. Он мог просто молча прийти, сожрать здесь все и выйти за дверь, даже не оглянувшись. Но даже в этом случае все было бы предельно просто и понятно. Уважения и сочувствия больше не было в его доме...

- Знали бы они... - сказал Фома, стоя у компьютера и разглядывая свое отражение в черном стекле монитора. - Скольких тогда застрелили? Двоих? »

Нет, троих: Маршал умер потом в больнице... Через пять недель... А сколько могло бы погибнуть еще? И ведь им невдомек...

На кухне раздался щелчок: чайник вскипел. Петр хозяйничал там, заваривая чай в двух кружках. Фома нашел взглядом нож, протянул к нему руку и принялся резать торт с числом 30 на нем. Лезвие вошло в бисквит, разделило пополам несколько розочек. Фома вдруг понял, что если не остановится сейчас, то сделает из торта кашу. Внезапно в комнату вошел Иваныч, неся чай. Фома откашлялся и швырнул нож на стол.

- Заждался я тебя, - кинул он в сторону Петра, не поднимая глаз. - Устал молчать.

- Сейчас и договорим, - согласился Иваныч, ставя кружки на стол. - Давай, присоединяйся.

Фома присел в кресло, переставил к себе чай и положил на блюдце кусок торта, едва не уронив его на пол. Руки у него заметно тряслись. Он все время облизывал сухие губы и нервно зевал. Петр, видя все это, решил: если так и дальше пойдет, лучше уйти, пусть Фома тут со своими тайнами как-нибудь сам, без него. В следующий раз разберемся, но по-трезвому.

- Вкусный чай, - произнес Фома, отхлебнув пару глотков. - Вот только горячий... Понимаешь, я тогда вдруг понял, что деньги сами к нам в руки плывут - только протяни и возьми. Тогда в городе появился банк, который очень щедро налево и направо дарил кредиты. Я когда вижу что-то подобное, то понимаю, что честно заработанные деньги так не раздают. То есть при удачном раскладе за эти деньги искать будут очень тихо и медленно, без эмоций и милиции. Я влез к ним в сетку, просканировал кое-что, прикинул... Ребята здорово помогали с проникновением, а Жанна просчитала вероятности нахождения крупных сумм в самом банке и на его счетах, потом пришла к выводу, что денег там не то что на четверых - на тысячу человек хватит. Я даже пару раз в банк заходил доллары менял. Посмотрел, как там внутри, и понял: временное все это, лишь бы деньги отмыть. Правда, очень большие деньги быстро не отмоешь, но они торопились, явно торопились... Ремонт у них был

Тихо заныл кулер. По экрану побежали строки БИОСа.

так себе, косметика одна, лишь бы дыры заделать. Вывеска, правда, яркая, но без этого никак. Реклама по всему городу - да ты должен сам помнить...

Петр кивнул - он действительно помнил рекламную кампанию, охватившую город в те времена. Население города жило с тех пор в кредит и собиралось это делать еще долгое время. Банк наделял всех деньгами за небольшой процент, делал все демократично и быстро, не требуя особых гарантий, справок и документов. Это подкупало людей. Они брали деньги, брали охотно: когда еще что-то подобное появится в их краю?

- ...Мы стали готовиться. Неудержимое желание пройтись по чужим счетам бульдозером. Тебе это должно быть знакомо... - Фома, казалось, трезвел на глазах. Речь стала более четкой, он смотрел прямо на Петра, а не блуждал взглядом по углам комнаты. - Я тоже хорошо помню это ощущение - волнующее, распирающее чувство силы и безнаказанности. Правда, я всегда себя контролировал. Боялся сорваться в эту самую безнаказанность и поверить в нее. Но жизнь всегда подталкивала меня к мысли о том, что я, как Раскольников, право имею. И в качестве той старушки, которую надо было убить, я выбрал этот самый банк.

Иваныч откинулся на спинку дивана и закинул ногу на ногу. Чувствовалось, что сегодня можно будет получить ответы на многие вопросы. Фомин всегда отличался либо максимальной скрытностью, либо предельной откровенностью - в зависимости от ситуации и расположения к людям, которые его окружали.

- То есть ты хочешь сказать, что к деньгам, которые ты хотел получить, примешивалось еще и что-то спортивное? То самое, что ты всегда пытался пресекать в нас? Неужели азарт?

- Нет, Иваныч, ты не понял, - разочарованно махнул рукой Фома. - Ну причем здесь азарт? Ты что, думаешь, Раскольников бабку из азарта завалил? В таком случае он на одной вряд ли бы остановился, ты ведь должен себе представлять это. Человек, который хоть раз испытал возбуждение при игре в казино, никогда его не забудет и всегда станет стремиться испытать его снова и снова. Я же был готов испытать ощущение победы ОДИН раз...

- Смысл? - в лоб спросил Петр, который все-таки чего-то не понимал. - Ведь выигрыш денег в случае удачи перекрывал все остальное, что ты пытался вложить в предприятие. Вот если бы я шел на эти миллионы, я бы не думал больше ни о чем, кроме них. Деньги - это же такой приз, что все остальное просто ни к чему. Выиграй его, а потом испытывай все доступные тебе чувства и эмоции в другом месте и в другое время...

- Знаешь, Петр, в чем наше с тобой основное отличие? - укоризненно склонил голову Фома. - Для тебя главное - чтобы не поймали. Для меня же главное - чтобы ловили...

Петр развел руками:

- Чего-то я тебя, Фома, вообще понять не могу... Куда-то ты завернул... В философию, блин...

- А что, у хакера не должно быть своей философии? Вот ты всегда стремился делать все тихо, не рисковать, взвешивать каждый шаг. Ну и где твои победы? Кто помнит о тебе? Какая хак-команда берет с тебя пример, изучает твои атаки, пытается рассуждать так, как ты, жаждет проникнуться ТВОЕЙ философией? Ты вечно второй. Я бы даже сказал, третий, а может, цифра больше...

Петр слушал закусив губу. Они как-то незаметно отклонились от темы беседы, нападение на банк ушло на задний план. Фома задел больные струны, сыграл на них издевательскую мелодию. Все, что говорил он сейчас, было истиной в последней инстанции. Иваныч вдруг понял, что у него мелко трясется нижняя губа: обида, еще не до конца понятая и проанализированная, уже пыталась овладеть им.

- А я? - вдруг спросил Фома. - Что ты думаешь обо мне? Ведь, согласись, хоть я и не стремился к показушности и громкой славе, эта слава всегда преследовала меня по пятам. Иногда мне казалось, что подобный подход сыграет со мной когда-нибудь злую шутку... Шутка, конечно, удалась. Вот только цена ее была крайне высока.

Он наклонился и нажал кнопку включения питания компьютера. Тихо заныл кулер. По экрану побежали строки БИОСа. Фома заботливо просмотрел их, кивнул и сказал:

- Все было хорошо, за исключением одного, и, к несчастью, я не понял этого сразу. Знаешь, Петр, они мне безгранично доверяли: я для них был этакий гардемарин за компьютером. Алешка Корсак со шпагой. Спаситель Отечества. Что бы ни случилось - я приду и спасу.

- Гардемарин... - словно пробуя на вкус, повторил это слово Иваныч. - Не знаю не знаю... Ты чего-то сегодня говоришь загадками. Так в чем там было дело? - подтолкнул он Фому к беседе о банке и деньгах. - Ты сумел пробиться на счета? Деньги, о которых мы говорили, ты их получил?

- Любопытство, Петя - страшная штука, - улыбнулся Фома. - Но я тебя прекрасно понимаю, сам я такой же. Отвечаю на твой вопрос. Я получил эти деньги. И вместе с ними получил такую проблему, от которой хотелось выть. Волком.

Иваныч напрягся.

- Ты все-таки сумел... - прошептал он. - Но, в таком случае, почему...

Он развел руками, пытаясь всем своим видом показать полное непонимание ситуации. Фома проследил его жест и криво улыбнулся.

- Не так быстро. Я расскажу...

Он посмотрел на экран монитора, задумался на пару секунд, а потом сказал:

- Расскажу, а потом кое-что покажу. Ну и вот: мы, как и все нормальные люди, хотели все и сразу...

Петр превратился в соляной столп, глядя на Фому во все глаза.

- Вариант с переводом большой суммы денег на подставные счета - это, конечно, хороший вариант. Но есть свои сложности. Отследить путь перемещения сумм проще, чем снять эти деньги. Ты понимаешь, что извлечь большую сумму из банковских кладовых, не привлекая к себе особого внимания, практически невозможно. Эти проклятые буржуйские структуры с огромной неохотой расстаются с вложениями, сделанными доверчивыми вкладчиками.

- Поменьше теории, - нетерпеливо сказал Петр, спохватился и машинально зажал ладонью рот. - Я хотел сказать, что вся эта лирика сейчас снова уведет нас в сторону...

- Эта лирика - часть моего плана, - недовольно произнес Фома. - Именно благодаря этим рассуждениям я решил, что надо заставить их отдать деньги самим. Много и сразу. И я стал прослеживать их контакты. Жанна помогла мне вычислить их основных партнеров по перемещениям и хранению денег. На первый взгляд, несложная задача, но этот банк был каким-то особенным. Их компьютеры общались с таким количеством сетей по России и ближнему зарубежью, что я порой не успевал воспринимать всю информацию, что сваливалась на меня. Помогли ребята из группы программирования: они накатали утилиту, которая анализировала все сама. Мне оставалось только оценить все происходящее, исходя из человеческой логики. Вот тут-то Жанне не было равных. Получая от нее заключения, я строил дальнейшие планы.

Фома повернулся к компьютеру, пощелкал мышкой, и на экране появилась фотография девушки, действительно похожей на Софию Ротару.

- Вот она, - не поворачиваясь к Петру, сказал Фома. - Единственная фотография, что у меня осталась. Она хотела сделать загранпаспорт - была на все сто процентов уверена в нашем успехе... Фотографии не пригодились.

- Она... Что с ней стало? - дрогнувшим голосом спросил Петр.

- Ее застрелили первой, - ответил Фома, и Петр поразился отсутствию эмоций в его голосе. - Она была ближе всех к двери, когда они вошли... Зря ты спросил. Да и я зря вспомнил.

Петр понял, что Фомин сейчас нальет себе рюмку. Так и вышло.

- Жанна помогла мне понять, откуда в банк приходят финансы, откуда берется наличка. Некая контора - не буду сейчас останавливаться на том, где она и как называется, это тоже "Рога и копыта" - отсылала в наш город деньги. Большими партиями. Эти самые деньги использовались банком для выдачи кредитов. Причем все это была валюта. Доллары. Я просматривал ее аналитические отчеты и никак не мог зацепиться ни за что. Не было ответа на вопрос, как прийти туда и взять деньги таким образом, чтобы тебе их дали и позволили с ними выйти. И вдруг в одном из отчетов я заметил, что был и обратный канал... Однажды - один раз за два месяца - деньги уехали назад. Восемьсот тысяч долларов. Ничего себе сумма, да?

Петр выпучил глаза, услышав эту цифру.

- Увидев это, я понял, что слабое звено где-то здесь. И стал ждать. Великое дело - терпение. Парни не понимали, чего я хочу. Я сам еще толком не понимал, но знал, что я решу эту задачку. И партия денег в обратном направлении спустя три недели всплыла снова...

- Не понимаю, - покачал головой Иваныч. - Я, наверное, так же, как и твоя группа, не понимаю логики. Чего ты ждал?

- Мне нужно было понять закономерности и условия перемещения денег из банка наружу. Я хотел просчитать ту самую логику, непонятную тебе, и в нужный момент встать между двумя конторами маленьким фильтром с крупной сеткой... Дело было за малым: надо было узнать, почему они возвращают деньги. Понимаешь, Петян, ведь если в банк может прийти человек, показать паспорт и взять в кредит тысячу долларов, значит, туда может приехать машина, показать накладную и вывезти миллион баксов... Только надо знать, когда.

Петр наконец-то стал понимать.

- Ты стал вычислять эту закономерность...

- И я никогда бы не понял, в чем смысл, если бы не случай.

Фома встал и прошел к окну. Закрыв глаза, он подставил лицо солнцу.

- Мне показалось, что я смогу понять все только внутри. И я пошел в банк. Придумал неплохую легенду, вошел внутрь и вышел через тридцать минут с кредитом в тысячу долларов на новый монитор. Вот на этот.

Он указал Петру на компьютерный стол. Иваныч кивнул и застыл в ожидании продолжения.

- Кредит я отдавать не собирался. Я вообще не собирался заморачиваться на деньги в этом проклятом городе. В принципе, я уже был одной ногой за границей. Взять деньги и рвануть в курортную зону! Кипр, Майами... "В мире столько мест, в которых я ни разу не был!"

- Трофим поет, - машинально отметил Петр, который тоже любил эту песню. - Хороший монитор. Правда, за тысячу долларов можно было купить два, а то и три монитора... Но ведь это не важно, да, Фома? Не ради монитора все это затевалось?

- С таким же успехом я мог купить себе новый диван. Но зачем диван человеку, который в своей квартире собирался спать всего лишь пару недель, максимум месяц в год? Не потащу же я его с собой в Швейцарию или Новую Зеландию! А монитор - тот нужен для дела. Приятно вершить свой самый большой в жизни подвиг, не ломая глаза.

Петр согласился одним кивком и спросил:

- А почему Новая Зеландия?

- А там "Властелина Колец" снимали. До сих пор в себя прийти не могу - какая природа!

Фома сделал несколько шагов по комнате, потом внезапно остановился и посмотрел на Петра:

- Чего ты мне зубы заговариваешь? Причем тут Зеландия? Тебе же интересно, чем все кончилось. Так и не перебивай меня своими дурацкими вопросами. Короче, продолжаю. Суть в том, что монитором дело не кончилось. Точнее сказать, именно с него все и началось. Понимаешь, он стоил почти восемь сотен, чуть больше. Осталась сотня баксов и еще немного "деревянных". Я пришел домой, распаковал монитор, установил, включил, полюбовался... А потом стал думать. И, пока я думал, вертел эту проклятую сотню в руках, разглядывал, будто впервые увидел. Что я хотел на ней найти тогда? Рецепт счастья? Даже и не знаю сейчас. Но факт остается фактом. Нашел.

- Что? - подался вперед Иваныч. Фома хитро посмотрел на него.

- Врать не буду, это я образно сказал. Нашел не я - нашла Жанна. Я был ей должен. Правда, должен я был больше, но это ерунда. Она взяла деньги, поблагодарила за то, что не наплевал на долг - в шутку, конечно... Не поверишь, Петя, о ней больше всех жалею. Человек она была хороший, и так все нелепо... Ведь не дай я ей тогда эту сотню, возможно, все были бы живы. Понимаешь, Иваныч, сотня оказалась фальшивой.

- Как? - удивленно поднял брови Петр. - А как же те десятки и сотни кредитов, которые брали люди? В конце концов, как же твой монитор? Ты его тоже за левые баксы взял? Но их же сейчас в каждом магазине проверяют! Как?! Как это получилось?

- Получилось, Петя, получилось. Я потом у Жанны спрашивал, почему она решила проверить купюру. Знаешь, что она сказала? Она пыталась понять, почему из банка увозят деньги. Вот же человеческий фактор! Ведь если из магазина увозят какой-то товар обратно на склад, что это означает?

- И что же?

- Что товар некачественный, неужели так трудно понять? - Фома даже рассердился на Иваныча. - Ведь все лежало на поверхности! Она предположила, что банк может пропускать через себя время от времени фальшивые деньги, отсюда и простота выдачи кредитов, и репутация банка, этакого временного образования на теле города. Ведь ни для кого не секрет, что банк выглядел так, будто собирался в любую секунду испариться. И точно - как в воду глядела! У нее были связи, она сумела сделать анализ бумаги, на которой печатались деньги. Бумага оказалась поддельной, несмотря на то, что все остальное было сделано настолько удачно, что придраться было не к чему. Купюра проходила проверку на любых стендах, имитировала любую защиту. Клише, на котором печатались деньги, было едва ли не лучше, чем оригинал в Федеральном казначействе.

- И никто за целый год не догадался? - спросил Петр. - Ведь тех, кто пользовался услугами банка, было несколько тысяч...

- Да кому оно надо! - возмутился Фома. - Никто и не станет копаться в этом, если деньги проходят проверку на детекторах в кассах магазинов! Всех устраивали эти красивые зеленые, а потом и розовые бумажки!

- А Жанна посчитала, что они уж слишком красивые? - вдруг сказал Петр, и Фомин кивнул, соглашаясь.

- Именно, Иваныч, именно... Так и сказала: "Уж очень они здорово выглядят, будто их сюда прямо из Америки возят". А возили их из соседней области.

> Дело было за малым: надо было узнать, почему они возвращают деньги.

Петр пригладил волосы, как делал всегда, когда волновался и с трудом переваривал поступающую информацию.

- Но ведь это... Это такой криминал! - только и сумел выдавить он из себя. - Это же сколько денег наводнили наш город и район! Черт возьми...

- Криминал, Петя. Да еще какой! Но из любого криминала можно извлечь выгоду, если ты обнаруживаешь его первым. Ситуация была такая, что распознать деньги никто не мог. Приходилось только восхищаться теми ребятами, что изготовили клише и сломали все степени защиты банкнот. И когда я понял, что их работа заслуживает похвалы, я принял решение. Надо было перехватить машину с деньгами, идущую назад.

- Зачем? Ты собирался напасть на них? - глаза у Петра полезли на лоб.

- Никогда, - хмыкнул Фома. - Все гораздо проще: они должны были сами мне их отдать.

- Как?

- Понимаешь, я решил, что решение о том, пускать деньги в оборот или не пускать, принимают именно в банке. Ведь если бы бракованную партию обнаружили прямо в цехе фальшивомонетчиков, то вряд ли бы она доехала сюда. Ее ликвидировали бы на месте. Значит, в банке сидит эксперт, который и решает, имеет ли партия денег право на существование. Уж не знаю, что он с ней делает, как проверяет, но именно он дает добро. Или делает так, что деньги уезжают обратно...

- А почему бы их не уничтожать на месте? Поняли, что не получились баксы - да и в печь их? В котельной какой-нибудь?

- А чего в чужой монастырь со своим уставом лезть? Откуда я знаю? Может, у них там такие жестокие отношения между собой, что не дай бог хоть один доллар утаить - приедут на танке и разнесут банк к чертовой матери. Хотя в твоих словах есть разумный момент: надо было держать в банке двух экспертов, с обеих сторон. Вместе присели, коньячку плеснули в рюмки и договорились, в печку или в сейф. Но, повторюсь, так было, и изменить это я сейчас не в состоянии.

Фома опустился за компьютер.

- Мои ребята стали плотно опекать банковские сети. Ничто не уходило и не приходило на их серверы незамеченным. Мы отслеживали весь их трафик, составили расписание перемещения денег, уточнили все детали, какие только можно было проверить, порой не задумываясь, нужны ли они будут в будущем. Спустя пару недель у нас сформировался четкий компьютерный »

портрет банка и той организации, что делала для них "товар". Осталось только ждать сообщения о том, что пришла некачественная партия - мы надеялись на то, что сумеем понять корпоративный шифр. К тому времени мы могли изготовить любой документ банка для всех надобностей: каждая бумажка прошла тщательный контроль и могла быть использована совершенно безбоязненно. Весь транспорт, что приходил и уходил из ворот банка, был сфотографирован нами. На частном предприятии - они с ГИБДД сотрудничали - были изготовлены фальшивые номерные знаки... Сложно даже сказать, какую работу мы провернули, не жалея последних своих сбережений...

- Но зачем? - спросил Петр. - Зачем вам нужны были бракованные деньги?

- Знаешь, Иваныч, я тоже поначалу сам себе задавал этот вопрос. Ответ был прост: судя по всему, брак был неочевиден. Эксперт просто перестраховывался и, как мне кажется, просто оправдывал свое существование. Ему нужно было временами показывать бурную деятельность, что у него неплохо получалось. В дальнейшем мы это подтвердили...

Спустя шестнадцать дней работы мы, наконец, получили результат. Мы знали день и час прибытия машины. Мы знали имена, фамилии и внешность всех банковских сотрудников, которые были задействованы в этой операции. Мы подготовили машину и необходимые документы.

Фома внезапно замолчал, словно не в силах был говорить дальше. Потом, переборов в себе что-то, не дававшее продолжать, вновь заговорил:

- И мы получили эти деньги. Каждый из нас до последнего сомневался в том, что у нас получится. Это я узнал уже потом, когда мы пришли сюда, в эту квартиру, после того, как надежно спрятали деньги. Все мы - и я в том числе - боялись. Боялись до дрожи в коленях, до стука зубов, до предательского пота на висках... Но мы сделали это. Мы взяли шестьсот пятьдесят тысяч долларов. Разделив это на четверых, мы пришли к выводу, что нам хватит на первое время.

Квартира к тому времени была отмыта от крови, трупы убраны.

- Шестьсот пятьдесят тысяч... С ума сойти... - прошептал Петр. - Да это целое состояние...

- На первый взгляд да. Это много денег. Но каждый меряет по-своему. Мы тоже вступили друг с другом в спор - хватит нам или нет. Жанна хотела уехать во Францию, двое программеров жаждали американской свободы, я же, как ты знаешь, был готов оставить эту страну ради Новой Зеландии. Мы рассуждали о том, как каждый из нас будет заниматься вопросами своего отъезда, как мы будем перемещать деньги, как будем их отмывать...

Перед нами был непочатый край работы. Намного больше было сделано для того, чтобы получить эти деньги. Но мы были готовы. Мы хотели жить... Мы были напуганы и счастливы одновременно. Мы пили шампанское, шутили, расслаблялись...

А потом кто-то позвонил в дверь, Жанна пошла открывать и получила пулю в сердце. Мы не слышали выстрела. Когда в комнату вошел мужчина с пистолетом, он задал всего один вопрос: кто из трех парней здесь Фома. Не сговариваясь, парни посмотрели в мою сторону и тут же были убиты. Вот прямо здесь, на этом диване. Там же, где ты сейчас сидишь...

Петр машинально отодвинулся от того места, которое занимал, в дальний угол дивана.

- Вот-вот, - покачал головой Фома. - Потом он подошел ко мне (а я пьяный да еще напуганный) и предложил пройти с ним. И я пошел. Это оказался киллер из той конторы, что делала фальшивые деньги. Я мгновенно протрезвел и подчинился...

Иваныч слушал, раскрыв рот. Пальцы его сжались в кулаки, сам он превратился в статую. Широко распахнутые глаза смотрели на Фому, как на Иоанна Крестителя, возвещающего о скором прибытии Христа.

- Со мной долго не разговаривали. Человек, к которому меня привезли, вежливо спросил, где деньги. Я сказал. Он отправил туда машину. Через полчаса ему сообщили, что товар возвращен. Тогда он пригласил в комнату, где меня держали, еще одного человека - ноут он за собой таскал. Тот стал расспрашивать меня о том, как мы это сделали. Я попытался было спихнуть основную часть проблемы на убитых ребят, но меня быстро привели в чувство шокером и угрозой подсадить на иглу. Пришлось рассказывать и показывать на компьютере... Знаешь, Петя, как легко все это делать под дулом пистолета, особенно когда помнишь простреленную Жанну, лежащую поперек коридора... Короче, я рассказал все. Потом первый человек вернулся. Они пошептались со вторым и пришли к выводу, что такие мозги, как у меня, нельзя просто "закатать в бетон" или "сбросить в реку". Их надо использовать по назначению. Мне сделали предложение - ну как, скажи, я мог от него отказаться? Я и стал работать на них.

Через пару дней они объявили о назначении меня новым экспертом по компьютерной безопасности. Квартира к тому времени была отмыта от крови, трупы убраны. Меня поселили сюда, как в золотую клетку. Я перемещаюсь под их присмотром, за мной следят без конца - и в постели, и в туалете. Мой компьютер контролируется кем-то, кого я никогда не видел, но он практически равен мне по силе. Мне разрешили отметить мой день рождения... Вот только никто не захотел прийти и поздравить того, кто, по их мнению, предал ребят и перешел на службу к тем, кто всегда был объектом нападения. Все посчитали меня предателем. Никто толком так и не знает, куда делась моя группа: люди просто исчезли из этой жизни... Но я всегда помню их. Мне так трудно жить, зная о том, что я работаю на тех, кто убил... Но я боюсь... Боюсь смерти, боюсь боли.

- Зачем ты мне все это рассказал? - спросил Петр.

- Мне в этой жизни осталось только одно, - чеканя каждое слово, ответил Фома. - ОТОМСТИТЬ.

- Кому? Этим людям? Тем, кто расстрелял твоих друзей?

- Нет, Петр, не им. Не им. Так уж получилось, что мои мозги, которые работают сейчас на моих врагов, ни на секунду не забывали все то, что я умел раньше... Тот человек, что курирует мою работу дома, силен. Очень силен. Но не настолько, насколько необходимо. Все эти камеры, жучки - чушь. Вот здесь (он ткнул пальцем себе в висок) есть все, чтобы заткнуть за пояс любого.

Он повернулся к компьютеру и положил пальцы на клавиатуру.

* * * * *

Человек проснулся оттого, что загудел тревожный сигнал. Он автоматически ткнул пальцем в клавиши, экран засветился.

- Сложно сказать, что он делает, но я отмечаю сетевую активность, - быстро проговорил он в гарнитуру, прижимая ее к щеке. - Я пытаюсь остановить, но у меня не получается.

Он быстро забарабанил по клавишам, время от времени сверяясь с показаниями мониторов сети. Губы шевелились, сыпались проклятия.

- Не могу, не могу, - бормотал он себе под нос. - Не могу, не могу... Так не бывает!!!

Он ударил кулаком по столу и откатился назад в кресле.

- Похоже, у меня мало времени, - сказал он сам себе, потом посмотрел на часы, вскочил и, накинув куртку на плечи, выбежал в дверь. Надо было поймать такси до аэропорта. Оставаться здесь было опасно...

* * * * *

Руки работали отдельно от Фомы. Он прикасался к клавишам, словно пианист, временами закрывая глаза.

- Что ты делаешь? - спросил Петр, подавшись вперед.

- Ты видел запись? - в свою очередь задал вопрос Фома. - Я знаю, у киллера была маленькая камера, приколотая к лацкану пиджака.

- Какую запись?

- Ты отодвинулся от того места, где лежали трупы. Не думаю, что это случайность. Ты знаешь, где они сидели, потому что видел запись.

- Что ты говоришь? Я не понимаю тебя! - Иваныч вскочил с дивана и отошел на шаг, ближе к коридору.

- Ведь это ты, Петя... Ты нас сдал. Я знаю, я прощупал всю их аппаратуру, у меня есть доказательства. Ты - единственный, кто завидовал мне в открытую. Ты ведь ко мне сегодня только с одной целью - посмотреть, каково мне сейчас, брошенному и заплеванному. Я не представляю, как ты живешь сейчас, но уверен - тебе недолго осталось.

Фома в последний раз ударил по клавишам и выключил комп.

- Ну? - улыбнулся он Петру. - В тот раз они пришли очень быстро...

И когда в дверь ворвались двое с пистолетами, он прикрыл глаза, вспоминая Жанну, похожую на Софию Ротару в молодости...

А в цехе, где шла печать очередной партии долларов, что-то случилось с программой, отвечающей за вывод. Внезапно повысилась температура в станке, бумагу стало корежить и выгибать дугой. Никто не мог остановить процесс, словно кнопки "Отмена" не существовало. И когда очередной лист перекосило и заклинило, внутри все пошло вразнос. Клише вырвало из своего гнезда и расплющило о выходной лоток...

А Петр с простреленной головой лежал в тарелке лицом в любимом салате из крабовых палочек и кукурузы. Именинник с пулей в сердце чуть покачивался в кресле напротив компьютера...

Праздник удался.

LG
Life's Good

FLATRON™
freedom of mind

FLATRON F700P

Абсолютно плоский экран
Размер точки 0,24 мм
Частота развертки 95 кГц
Экранное разрешение 1600x1200
USB-интерфейс

Dina Victoria
(095) 688-61-17, 688-27-65
WWW.DVCOMP.RU

Москва: АБ-групп (095) 745-5175; Акситек (095) 784-7224; Банкос (095) 128-9022; ДЕЛ (095) 250-5536; Дилайн (095) 969-2222; Инкотрейд (095) 176-2873; ИНЭЛ (095) 742-6436; Карин (095) 956-1158; Компьютерный салон SMS (095) 956-1225; Компания КИТ (095) 777-6655; Никс (095) 974-3333; ОЛДИ (095) 105-0700; Регард (095) 912-4224; Сетевая Лаборатория (095) 784-6490; СКИД (095) 232-3324; Тринити Электроникс (095) 737-8046; Формоза (095) 234-2164; Ф-Центр (095) 472-6104; ЭЛСТ (095) 728-4060; Flake (095) 236-992; Force Computers (095) 775-6655; ISM (095) 718-4020; Meijin (095) 727-1222; NT Computer (095) 970-1930; R-Style Trading (095) 514-1414; USN Computers (095) 755-8202; ULTRA Computers (095) 729-5255; ЭЛЕКТОН (095) 956-3819; ПортКом (095)777-0210; **Архангельск:** Северная Корона (8182) 653-525; **Волгоград:** Техком (8612) 699-850; **Воронеж:** Рет (0732) 779-339; РИАН (0732) 512-412; Сани (0732) 54-00-00; **Иркутск:** Билайн (3952) 240-024; Комтек (3952) 258-338; **Краснодар:** Игрек (8612) 699-850; **Лабытнанги:** КЦ ЯМАЛ (34992) 51777; **Липецк:** Регард-тур (0742) 485-285; **Новосибирск:** Квеста (38322) 332-407; **Нижний Новгород:** Бюро-К (8312) 422-367; **Пермь:** Гаском (8612) 699-850; **Ростов-на-Дону:** Зенит-Компьютер (8632) 950-300; **Тюмень:** ИНЭКС-Техника (3452) 390-036.

MOUNTAIN BIKE ACTION

ГЛАВНЫЙ ЖУРНАЛ РОССИИ О МАУНТИН БАЙКЕ

В ПРОДАЖЕ С 6-го ИЮЛЯ