Lecture 4
Exploiting. Shellcodes
Computer and Network Security
October 21, 2019
Computer Science and Engineering Department

Lecture 4
Exploiting. Shellcodes

Computer and Network Security
October 21, 2019

Computer Science and Engineering Department

CNSO Outline
Vulnerabilities and Exploits
Runtime Exploiting
Shellcode
Trigerring and Placing Shellcodes
Conclusion

CNSO CTF crunch

Outline

CNSO
Bugs and Vulnerabilities
- bugs: misbehaving sofware
- vulnerability: misbehaviour that can benefit an attacker
- exploiting: turning a vulnerability into an advantage for the attacker
- auditing: analyzing an application to determine its vulnerabilities

CNSO CTF crunch

Bugs and Vulnerabilities

- bugs: misbehaving sofware
- vulnerability: misbehaviour that can benefit an attacker
- exploiting: turning a vulnerability into an advantage for the attacker
- auditing: analyzing an application to determine its vulnerabilities

CNSO Why Bugs and Vulnerabilities?
- developer carelessness or ignorance
- poor development process
- poor design
- platform (hardware, OS, libraries) issues
- lack of resources

Why Bugs and Vulnerabilities?

- developer carelessness or ignorance
- poor development process
- poor design
- platform (hardware, OS, libraries) issues
- lack of resources

CNSO Preventing Vulnerabilities
- development process: defensive programming, code review, code audit
- design with security in mind
- audit systems, penetration testing
- security-centered training
- invest resources

CNSO CTF crunch

Preventing Vulnerabilities

- development process: defensive programming, code review, code audit
- design with security in mind
- audit systems, penetration testing
- security-centered training
- invest resources

CNSO Security Attacks
- eavesdropping, impersonating
- password breaking
- denial of service
- exploiting

CNSO CTF crunch

Security Attacks

- eavesdropping, impersonating
- password breaking
- denial of service
- exploiting

CNSO Exploiting
- exploiting vulnerabilities
- focus is controlling the system (root account)
- an intermediary step is gaining shell access to user
- privilege escalation

CNSO CTF crunch

Exploiting

- exploiting vulnerabilities
- focus is controlling the system (root account)
- an intermediary step is gaining shell access to user
- privilege escalation

CNSO Why Exploiting?
- money
- fame
- challenge
- fun
- political, ideological
- find security holes and fix them (ethical hacking)

CNSO CTF crunch

Why Exploiting?

- money
- fame
- challenge
- fun
- political, ideological
- find security holes and fix them (ethical hacking)

CNSO CTF crunch Detect/Prevent Exploiting
- monitoring
- update software
- stay connected
- in-depth security
- honeypots
- state of mind: "it will happen"

Detect/Prevent Exploiting

- monitoring
- update software
- stay connected
- in-depth security
- honeypots
- state of mind: "it will happen"

CNSO Types of Exploits
- local exploit
- remote exploit
- user space exploit
- kernel space exploit

CNSO CTF crunch

Types of Exploits

- local exploit
- remote exploit
- user space exploit
- kernel space exploit

CNSO Outline
Vulnerabilities and Exploits
Runtime Exploiting
Shellcode
Trigerring and Placing Shellcodes
Conclusion

CNSO CTF crunch

Outline

CNSO
Runtime Application Exploiting
- find vulnerability in process runtime: memory, use of resources
- alter normal execution pattern
- aim for: getting a shell, getting access to resources, information leak, crash application, denial of service
- usually tamper with process memory and bad ways of memory management
- special focus on string management functions, input/output, pointers

CNSO CTF crunch

Runtime Application Exploiting

- find vulnerability in process runtime: memory, use of resources
- alter normal execution pattern
- aim for: getting a shell, getting access to resources, information leak, crash application, denial of service
- usually tamper with process memory and bad ways of memory management
- special focus on string management functions, input/output, pointers

CNSO Runtime Exploit Components
- preparatory phase
- shellcode
- triggering phase

CNSO CTF crunch

Runtime Exploit Components

- preparatory phase
- shellcode
- triggering phase

CNSO Types of Runtime Application Vulnerabilities
- buffer overflow (on stack or heap)
- integer overflow
- race conditions
- string formatting

CNSO CTF crunch

Types of Runtime Application Vulnerabilities

- buffer overflow (on stack or heap)
- integer overflow
- race conditions
- string formatting

CNSO Buffer Overflow
- write beyond buffer limits
- stack-based overflow: overwrite variable, return address or function pointer
- heap overflow: corrupt dynamically allocated memory

CNSO CTF crunch

Buffer Overflow

- write beyond buffer limits
- stack-based overflow: overwrite variable, return address or function pointer
- heap overflow: corrupt dynamically allocated memory

CNSO CTF crunch Outline
Vulnerabilities and Exploits
Runtime Exploiting
Shellcode
Trigerring and Placing Shellcodes
Conclusion

CNSO CTF crunch

Outline

CNSO Shellcode
- sequence of machine level instructions
- stored in memory at a convenient address
- executed when requested by jumping at the start address

CNSO CTF crunch

Shellcode

- sequence of machine level instructions
- stored in memory at a convenient address
- executed when requested by jumping at the start address

CNSO
Shellcode Objectives
- typically the goal is to create a shell (if possible, with root privilege)
- may be any useful binary code execution, such as starting a client socket, or reading or writing a file, or sending a file over the network

CNSO CTF crunch

Shellcode Objectives

- typically the goal is to create a shell (if possible, with root privilege)
- may be any useful binary code execution, such as starting a client socket, or reading or writing a file, or sending a file over the network

CNSC CTF crunch
Shellcode Samples
- http://www.shell-storm.org/shellcode/
- hexadecimal form for exec-ing a shell process
- also dubbed payload

Shellcode Samples

- http://www.shell-storm.org/shellcode/
- hexadecimal form for exec-ing a shell process
- also dubbed payload

CNSO
Shellcode Types
- spawn shell using `execve` syscall
- use `setresuid` to restore root privileges (for setuid-enabled programs)
- port-binding shellcode: create listener socket, accept connections, duplicate file descriptors and spawn shell
- connect-back shellcode: create client socket and connect to remote listener socket (accesible and controled by attacker), duplicate file descriptors and spawn shell

CNSO CTF crunch

Shellcode Types

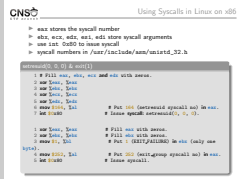
- spawn shell using `execve` syscall
- use `setresuid` to restore root privileges (for setuid-enabled programs)
- port-binding shellcode: create listener socket, accept connections, duplicate file descriptors and spawn shell
- connect-back shellcode: create client socket and connect to remote listener socket (accesible and controled by attacker), duplicate file descriptors and spawn shell

Creating a Shellcode

- may be done in C but it is recommended to do it in assembly
  - allows shorter shellcodes
  - complete control over the end result (binary machine code)
- need to use syscalls for `execve`, `setresuid`, `dup2` and others
- need to place the `/bin/sh` string in memory (or other strings) and pass it as argument to syscall

Creating a Shellcode

- may be done in C but it is recommended to do it in assembly
  - allows shorter shellcodes
  - complete control over the end result (binary machine code)
- need to use syscalls for `execve`, `setresuid`, `dup2` and others
- need to place the `/bin/sh` string in memory (or other strings) and pass it as argument to syscall



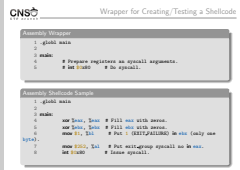
Using Syscalls in Linux on x86

- eax stores the syscall number
- ebx, ecx, edx, esi, edi store syscall arguments
- use int 0x80 to issue syscall
- syscall numbers in /usr/include/asm/unistd_32.h

setresuid(0, 0, 0) & exit(1)

```
    1 # Fill eax, ebx, ecx and edx with zeros.
    2 xor %eax, %eax
    3 xor %ebx, %ebx
    4 xor %ecx, %ecx
    5 xor %edx, %edx
    6 mov $164, %al           # Put 164 (setresuid syscall no) in eax.
    7 int $0x80               # Issue syscall: setresuid(0, 0, 0).
```

```
    1 xor %eax, %eax          # Fill eax with zeros.
    2 xor %ebx, %ebx          # Fill ebx with zeros.
    3 mov $1, %bl             # Put 1 (EXIT_FAILURE) in ebx (only one
byte).
    4 mov $252, %al           # Put 252 (exit_group syscall no) in eax.
    5 int $0x80               # Issue syscall.
```



CNSO CTF crunch

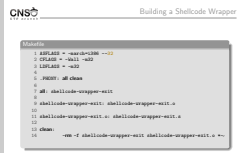
Wrapper for Creating/Testing a Shellcode

Assembly Wrapper

```
1 .globl main
2
3 main:
4         # Prepare registers an syscall arguments.
5         # int $0x80     # Do syscall.
```

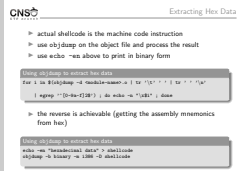
Assembly Shellcode Sample

```
1 .globl main
2
3 main:
4         xor %eax, %eax  # Fill eax with zeros.
5         xor %ebx, %ebx  # Fill ebx with zeros.
6         mov $1, %bl     # Put 1 (EXIT_FAILURE) in ebx (only one
byte).
7         mov $252, %al   # Put exit_group syscall no in eax.
8         int $0x80       # Issue syscall.
```



Makefile

```
1  ASFLAGS = -march=i386 --32
2  CFLAGS = -Wall -m32
3  LDFLAGS = -m32
4
5  .PHONY: all clean
6
7  all: shellcode-wrapper-exit
8
9  shellcode-wrapper-exit: shellcode-wrapper-exit.o
10
11 shellcode-wrapper-exit.o: shellcode-wrapper-exit.s
12
13 clean:
14         -rm -f shellcode-wrapper-exit shellcode-wrapper-exit.o *~
```



CNSO CTF crunch

Extracting Hex Data

- actual shellcode is the machine code instruction
- use `objdump` on the object file and process the result
- use `echo -en` above to print in binary form

Using `objdump` to extract hex data

```
for i in $(objdump -d <module-name>.o | tr '\t' ' ' | tr ' ' '\n'

    | egrep '^[0-9a-f]2$') ; do echo -n "\x$i" ; done
```

- the reverse is achievable (getting the assembly mnemonics from hex)

Using `objdump` to extract hex data

```
echo -en "hexadecimal data" > shellcode
objdump -b binary -m i386 -D shellcode
```

CNSO CTF crunch Shellcode Constraints
due to input data filtering
small code
null-free
position-independent
alphanumeric (not always)
more on the next lecture

Shellcode Constraints

- due to input data filtering
- small code
- null-free
- position-independent
- alphanumeric (not always)
- more on the next lecture

CNSO CTF crunch Null-free

- required when dealing with null-terminated strings
- BAD: mov $1, %eax
  - uses null bytes
  - \xb8\x01\x00\x00\x00
- GOOD: xor %eax, %eax + inc %eax
  - doesn't use null bytes
  - \x31\xc0\x40
- BAD: mov $100, %eax
  - uses null bytes
  - \xb8\x64\x00\x00\x00
- GOOD: xor %eax, %eax + mov $100, %al
  - doesn't use null bytes
  - \x31\xc0\xb0\x64

CNSO CTF crunch

Null-free

- required when dealing with null-terminated strings
- BAD: `mov $1, %eax`
  - uses null bytes
  - `\xb8\x01\x00\x00\x00`
- GOOD: `xor %eax, %eax` + `inc %eax`
  - doesn't use null bytes
  - `\x31\xc0\x40`
- BAD: `mov $100, %eax`
  - uses null bytes
  - `\xb8\x64\x00\x00\x00`
- GOOD: `xor %eax, %eax` + `mov $100, %al`
  - doesn't use null bytes
  - `\x31\xc0\xb0\x64`

CNSO Outline
Vulnerabilities and Exploits
Runtime Exploiting
Shellcode
Trigerring and Placing Shellcodes
Conclusion

CNSO CTF crunch

Outline

CNSO Using the Local Stack
- place shellcode in local buffer on stack
- rewrite return address to point to beginning of the buffer on the stack
- may need NOPs if exact address is not known
- unable to be done if stack is non-executable

CNSO CTF crunch

Using the Local Stack

- place shellcode in local buffer on stack
- rewrite return address to point to beginning of the buffer on the stack
- may need NOPs if exact address is not known
- unable to be done if stack is non-executable

CNSO
Using an Environment Variable
- 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
- more on the next lecture

CNSO CTF crunch

Using an Environment Variable

- 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
- more on the next lecture

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

CNSO CTF crunch

Using the Heap

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

CNSO Triggering Shellcodes
- stack buffer overflow
  - overwrite return address and point to address on stack or environment variable
  - overwrite local pointer and point to address on stack or environment variable
- heap buffer overflow
  - overwrites metadata pointers for heap allocated data

CNSO CTF crunch

Triggering Shellcodes

- stack buffer overflow
  - overwrite return address and point to address on stack or environment variable
  - overwrite local pointer and point to address on stack or environment variable
- heap buffer overflow
  - overwrites metadata pointers for heap allocated data

CNSO CTF crunch Outline
Vulnerabilities and Exploits
Runtime Exploiting
Shellcode
Trigerring and Placing Shellcodes
Conclusion

Outline

CNSO CTF crunch Keywords

- bugs
- vulnerabilities
- exploit
- shellcode
- shellcode construction
- shellcode triggering
- shellcode placing
- syscall
- null
- stack buffer overflow
- heap buffer overflow
- pwntools

Keywords

- bugs
- vulnerabilities
- exploit
- shellcode
- shellcode construction
- shellcode triggering
- shellcode placing
- syscall
- null
- stack buffer overflow
- heap buffer overflow
- pwntools

CNSO CTF crunch Useful Links
- http://www.blackhatlibrary.net/Category:Shellcode
- http://www.shell-storm.org/shellcode/
- http://www.metasploit.com/
- https://github.com/Gallopsled/pwntools
- https://docs.pwntools.com/en/stable/

CNSO CTF crunch

Useful Links

- http://www.blackhatlibrary.net/Category:Shellcode
- http://www.shell-storm.org/shellcode/
- http://www.metasploit.com/
- https://github.com/Gallopsled/pwntools
- https://docs.pwntools.com/en/stable/

CNSO CTF crunch References

- 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

CNSO CTF crunch

References

- 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