Lecture 08
Code Reuse (part 1)
Computer and Network Security
November 18, 2019
Computer Science and Engineering Department

Lecture 08
Code Reuse (part 1)

Computer and Network Security
November 18, 2019

Computer Science and Engineering Department

CNSO Defense Mechanisms
- static & dynamic analysis
- ASCII armored address space
- stack guard, canary value
- DEP: Data Execution Prevention
- ASLR: Address Space Layout Randomization

CNSO CTF crunch

Defense Mechanisms

- static & dynamic analysis
- ASCII armored address space
- stack guard, canary value
- DEP: Data Execution Prevention
- ASLR: Address Space Layout Randomization

CNSO Data Execution Prevention
- writable code may not be executed
- stack, heap, data, bsss
- you cannot (easily) inject code (and run it)

Data Execution Prevention

- writable code may not be executed
- stack, heap, data, bsss
- you cannot (easily) inject code (and run it)

CNSO Code Reuse

- use existing code
- text and library text
- interesting to use/call functions
- smaller pieces may also be used
- no need to inject code

CNSO CTF crunch

Code Reuse

- use existing code
- text and library text
- interesting to use/call functions
- smaller pieces may also be used
- no need to inject code

CNSO return-to-libc

- call existing library functions
- canonical exploit is calling system("/bin/sh") in libc
- code pointer is overwriten with address of library function

CNSO CTF crunch

return-to-libc

- call existing library functions
- canonical exploit is calling `system("/bin/sh")` in `libc`
- code pointer is overwriten with address of library function

CNSO Return-Oriented Programming
- use smaller pieces of code
- pieces are called *ROP gadgets*, ending in `ret` instruction
- payload consists of data on stack and pointers to ROP gadgets

CNSO CTF crunch

Return-Oriented Programming

- use smaller pieces of code
- pieces are called *ROP gadgets*, ending in `ret` instruction
- payload consists of data on stack and pointers to ROP gadgets

CNSO CTF crunch Outline
return-to-libc
Return-Oriented Programming

CNSO CTF crunch

Outline

CNSO High-level View
- call functions in libraries (libc)
- call system("/bin/sh") for a shell
- call puts() for information leak

CNSO CTF crunch

High-level View

- call functions in libraries (`libc`)
- call `system("/bin/sh")` for a shell
- call `puts()` for information leak

CNSO
Steps in a ret-to-libc attack
- buffer overflow required
- identify function addresses and addresses of arguments (such as the "/bin/sh" string)
- ovewrite code pointer with function address
- place arguments on stack

CNSO CTF crunch

Steps in a ret-to-libc attack

- buffer overflow required
- identify function addresses and addresses of arguments (such as the *"/bin/sh"* string)
- ovewrite code pointer with function address
- place arguments on stack

CNSO CTF crunch Sample Payloads for ret-to-libc attack
- padding + overwrite_lib_address + irrelevant_ret_address + arg1 + arg2 + ...
- offset*"A" + p32(system_address) + 4*"B" + p32(bin_sh_address)
- offset*"A" + p32(write) + 4*"B" + p32(1) + p32(buf_address) + p32(buf_len)

CNSO CTF crunch

Sample Payloads for ret-to-libc attack

- `padding + overwrite_lib_address + irrelevant_ret_address + arg1 + arg2 + ...`
- `offset*"A" + p32(system_address) + 4*"B" + p32(bin_sh_address)`
- `offset*"A" + p32(write) + 4*"B" + p32(1) + p32(buf_address) + p32(buf_len)`

CNSO Protecting Against ret-to-libc Attacks
- use ASLR to randomize function addresses in library
- use stack canary
- need information leak to bypass protection mechanisms

CNSO CTF crunch

Protecting Against ret-to-libc Attacks

- use ASLR to randomize function addresses in library
- use stack canary
- need information leak to bypass protection mechanisms

CNSO ret-to-libc Shortcomings
- cannot chain multiple calls due to stack limitations
- function calls may be too coarse; you may need smaller chunks

ret-to-libc Shortcomings

- cannot chain multiple calls due to stack limitations
- function calls may be too coarse; you may need smaller chunks

CNSO CTF crunch Outline
return-to-libc
Return-Oriented Programming

CNSO CTF crunch

Outline

return-to-libc

Return-Oriented Programming

CNSO Remember ret Instruction
- pop instruction/code pointer from stack
- code pointer was placed by call instruction ...
- ... or by exloit payload

Remember `ret` Instruction

- pop instruction/code pointer from stack
- code pointer was placed by `call` instruction ...
- ... or by exloit payload

CNSO
pop-ret
- a sequence: data + code pointer on the stack is used by a pop; ret sequence
- e.g.: pop eax; ret: place data in eax and pop instruction pointer from stack
- you may use pop2-ret or pop3-ret etc.

CNSO CTF crunch

pop-ret

- a sequence: data + code pointer on the stack is used by a `pop; ret` sequence
- e.g.: `pop eax; ret`: place data in `eax` and pop instruction pointer from stack
- you may use `pop2-ret` or `pop3-ret` etc.

CNSO Freeing the Stack
- chain together multiple functions
- after calling a function do a pop-ret or popX-ret to free function arguments

Freeing the Stack

- chain together multiple functions
- after calling a function do a `pop-ret` or `popX-ret` to free function arguments

CNSO ROP Payloads
- offset*"A" + p32(puts) + p32(pop_ret_gadget) + p32(puts_string_address)
- offset*"A" + p32(write) + p32(pop3_ret_gadget) + p32(1) + p32(buf) + p32(buf_len)

CNSO CTF crunch

ROP Payloads

- `offset*"A" + p32(puts) + p32(pop_ret_gadget) + p32(puts_string_address)`
- `offset*"A" + p32(write) + p32(pop3_ret_gadget) + p32(1) + p32(buf) + p32(buf_len)`

CNSO Return Oriented Programming
- using function calls + ret-based calls to chain together code reuse chunks
- makes use of *ROP gadgets*
- is a Turing-complete *language*

Return Oriented Programming

- using function calls + ret-based calls to chain together code reuse chunks
- makes use of *ROP gadgets*
- is a Turing-complete *language*

CNSO CTF crunch ROP Gadget
small sequences ending in ret
use ROPgadget tool (comes with pwntools)
use ropgadget or ropsearch or asmsearch in PEDA

CNSO CTF crunch

ROP Gadget

- small sequences ending in `ret`
- use `ROPgadget` tool (comes with `pwntools`)
- use `ropgadget` or `ropsearch` or `asmsearch` in PEDA

CNSO ROP Chain
- chain together function calls + ROP gadgets
- do information leak, rewrites, open sockets, run shells

CNSO CTF crunch

ROP Chain

- chain together function calls + ROP gadgets
- do information leak, rewrites, open sockets, run shells

CNSO Use Cases
- leak information: variables, addresses
- open shell
- call mprotect() to disable DEP and then inject shellcode

CNSO CTF crunch

Use Cases

- leak information: variables, addresses
- open shell
- call `mprotect()` to disable DEP and then inject shellcode

CNSO CTF crunch Keywords

- DEP
- code reuse
- return-to-libc
- ROP
- ROP gadget
- ROP chain
- ROPgadget

CNSO CTF crunch

Keywords

- DEP
- code reuse
- return-to-libc
- ROP
- ROP gadget
- ROP chain
- `ROPgadget`

CNSO References
- https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf
- https://trailofbits.files.wordpress.com/2010/04/practical-rop.pdf
- http://codearcana.com/posts/2013/05/28/introduction-to-return-oriented-programming-rop.html
- https://github.com/JonathanSalwan/ROPgadget

CNSO CTF crunch

References

- https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf
- https://trailofbits.files.wordpress.com/2010/04/practical-rop.pdf
- http://codearcana.com/posts/2013/05/28/introduction-to-return-oriented-programming-rop.html
- https://github.com/JonathanSalwan/ROPgadget