Memory Corruption Part 2

Mina Zhou, John Cunniff
OSIRIS Lab Hack Night

Homework Review

Modern Mitigations

```
RELRO:    Full RELRO
Stack:    Canary found
NX:       NX enabled
PIE:      PIE enabled
```

RELRO:	ReLocation Read Only
Stack Canary:	Random fix-sized buffer between locals and return address
DEP/NX:	Data Execution Prevention/No Execution
ASLR/PIE:	Address space layout randomization/Position Independent Executable

Stack Canary

```
00401409  mov      rax, qword [fs:0x28]
00401412  mov      qword [rbp-0x8 {var_10}], rax
```

- To mitigate stack smashing process
- Random 8 bytes stored in fs:[0x28]
  - Always starts with a null byte. Why?
- Added before saved RBP
- It prevents simple buffer overflows

```
*** stack smashing detected ***: <unknown> terminated
Aborted
```

Stack Canary Bypass

If we can read the data in the stack canary, we can send it back to the program later because the canary stays the same throughout execution

- Data Leak
  - User-controlled Format String → printf(input);
  - User-controlled length of an output → “send me 1000000 bytes plz”’
  - Out of bounds array access → array[1000];

Stack Canary Bypass

- Brute forcing a Stack Canary

Buffer (N bytes)	Canary	RBP	RIP
“A” * N	00 ?? ?? ?? ?? ?? ?? ?? 00 51 ?? ?? ?? ?? ?? ?? 00 51 FE ?? ?? ?? ?? ?? 00 51 FE 01 ?? ?? ?? ??	“A” * 8	yayyy

NX / DEP

No eXecution/Data Execution Prevention

- Marks pages as not executable
  - stack
  - .bss
  - .data

NX / DEP Bypass

Two common techniques:

GOT Overwrite

ROP

GOT / PLT

- If we use a shared library, how do we know where functions are in memory?
  - Relocation
- The Global Offset Table (GOT) stores the location of external symbols
  - Locations are not known at compile time (b/c it changes every time you run)
    - Before functions are used, the GOT stores a temporary value (stub)
  - When a function is called for the first time, the dynamic loader will find the address and patch the GOT entry

GOT / PLT

.got (Global Offset Table)	offsets of external symbols
.plt (Procedure Linkage Table)	stubs that jump to right address or trigger linker to look up address
.got.plt	resolved addresses or address back to .plt if not resolved
.plt.got	code to jump to first entry in .got

GOT / PLT

Got overwrite?

.got.plt is really just array of function pointers that we can overwrite

```
000601000 ; ===========================================================================
000601000
000601000 ; Segment type: Pure data
000601000 ; Segment permissions: Read/Write
000601000 ; Segment alignment 'qword' can not be represented in assembly
000601000 _got_plt        segment para public 'DATA' use64
000601000                 assume cs:_got_plt
000601000                 ;org 601000h
000601000 _GLOBAL_OFFSET_TABLE_ dq offset _DYNAMIC
000601008 qword_601008    dq 0                  ; DATA XREF: sub_4005A0↑r
000601010 qword_601010    dq 0                  ; DATA XREF: sub_4005A0+6↑r
000601018 off_601018      dq offset puts        ; DATA XREF: _puts↑r
000601020 off_601020      dq offset __stack_chk_fail
000601020                                       ; DATA XREF: ___stack_chk_fail↑r
000601028 off_601028      dq offset system      ; DATA XREF: _system↑r
000601030 off_601030      dq offset printf      ; DATA XREF: _printf↑r
000601038 off_601038      dq offset __libc_start_main
000601038                                       ; DATA XREF: ___libc_start_main↑r
000601040 off_601040      dq offset fgets       ; DATA XREF: _fgets↑r
000601048 off_601048      dq offset setvbuf     ; DATA XREF: _setvbuf↑r
000601048 _got_plt        ends
000601048
```

GOT / PLT Demo

ROP

What does ret do again?

Return-Oriented Programing

Situation:

- NX on
- No give_shell function or system

What do we do?

ROP

Use the code that is already in the program!

x86 code is variable length, so we can execute in the middle of instructions

Use “gadgets” to set up function calls

Turing complete

Magic Gadget

Search your gadgets on your binaries to facilitate your ROP exploitation

ROPgadget

one_gadget

ROP Gadgets

Gadgets are any set of instructions that end in a ret, syscall, call, or jmp

Basic gadgets are of the form pop and then ret

```
0x0000000000400679 : nop ; pop rbp ; ret
```

A rop chain is a series of gadgets

ROP Gadgets

How do we set up a function call? Say system(“/bin/sh”)?

We need:

- rdi = pointer to “/bin/sh”
- ret to system

ROP Demo

Stack Pivot

What if we don’t have enough space on the stack for our rop chain?

Just write it somewhere else

Need:

- Big writable section of memory
- A pop rsp gadget

Stack Pivot

So where do we pivot into?

.bss section rw-

Uninitialized globals

```
.bss:0000000000601060 ; ===========================================================================
.bss:0000000000601060
.bss:0000000000601060 ; Segment type: Uninitialized
.bss:0000000000601060 ; Segment permissions: Read/Write
.bss:0000000000601060 ; Segment alignment '32byte' can not be represented in assembly
.bss:0000000000601060 _bss            segment para public 'BSS' use64
.bss:0000000000601060                 assume cs:_bss
.bss:0000000000601060                 ;org 601060h
.bss:0000000000601060                 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:
.bss:0000000000601060                 public stdout@@GLIBC_2_2_5
.bss:0000000000601060 ; FILE *stdout
.bss:0000000000601060 stdout@@GLIBC_2_2_5 dq ?                ; DATA XREF: LOAD:000000000040
.bss:0000000000601060                                         ; init+4↑r
.bss:0000000000601060                                         ; Alternative name is 'stdout'
.bss:0000000000601060                                         ; Copy of shared data
.bss:0000000000601068                 align 10h
.bss:0000000000601070                 public stdin@@GLIBC_2_2_5
.bss:0000000000601070 ; FILE *stdin
.bss:0000000000601070 stdin@@GLIBC_2_2_5 dq ?                 ; DATA XREF: LOAD:000000000040
.bss:0000000000601070                                         ; vuln+15↑r
.bss:0000000000601070                                         ; Alternative name is 'stdin'
.bss:0000000000601070                                         ; Copy of shared data
.bss:0000000000601078 completed_7621  db ?                    ; DATA XREF: __do_global_dtors_
.bss:0000000000601078                                         ; __do_global_dtors_aux+12↑w
.bss:0000000000601079                 align 20h
.bss:0000000000601080                 public name
.bss:0000000000601080 name            db    ? ;               ; DATA XREF: vuln+30↑o
.bss:0000000000601081                 db    ? ;
.bss:0000000000601082                 db    ? ;
.bss:0000000000601083                 db    ? ;
.bss:0000000000601084                 db    ? ;
.bss:0000000000601085                 db    ? ;
.bss:0000000000601086                 db    ? ;
.bss:0000000000601087                 db    ? ;
.bss:0000000000601088                 db    ? ;
.bss:0000000000601089                 db    ? ;
.bss:000000000060108A                 db    ? ;
.bss:000000000060108B                 db    ? ;
.bss:000000000060108C                 db    ? ;
.bss:000000000060108D                 db    ? ;
.bss:000000000060108E                 db    ? ;
.bss:000000000060108F                 db    ? ;
```

Stack Pivot

Caveats

- New stack needs to be setup before we pivot
- Need to have a big enough uninitialized global

Logistical nightmare

ASLR / PIE

Address Space Layout Randomization / Position Independent Code

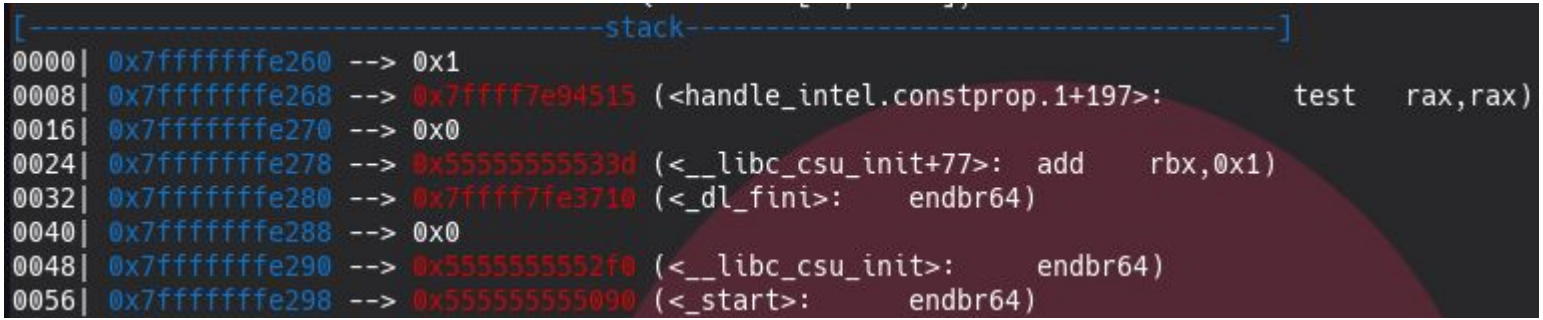
The location of these sections in memory are randomized :

- Program
- Heap
- Shared Libraries (libc)
- Stack

ASLR / PIE Bypass

How do we get around it?

- Leak an address



ASLR / PIE Bypass

General ways of leaking different regions

- libc : __libc_start_main
- stack : pushed rbp
- program : pushed returned pointer

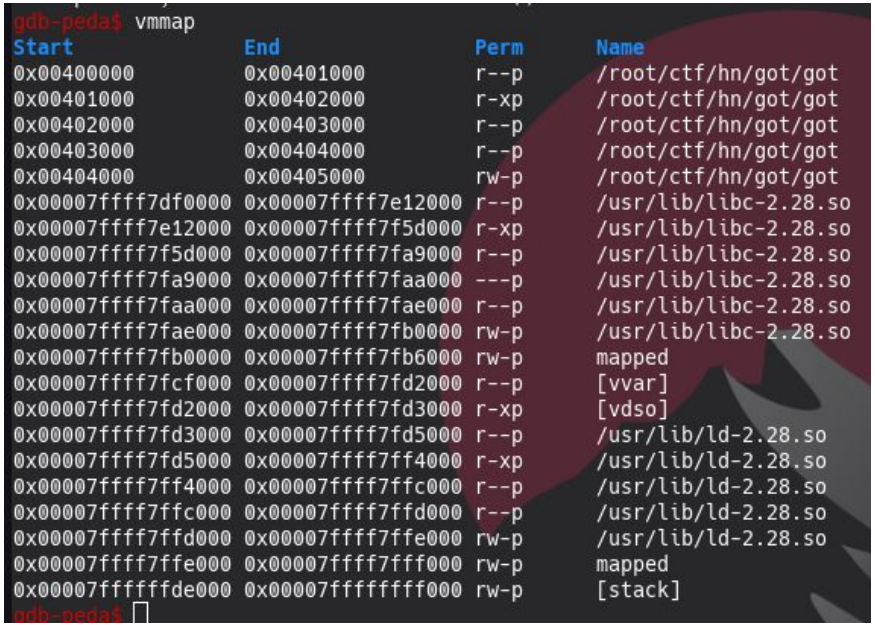
ASLR / PIE Demo

Partial Overwrite

PIE only changes the start of a data area
These areas are also page aligned

What can we do with this?

- Normal return address: 0x???123
- Give shell address: 0x???abc



RELRO

RELocation Read Only

- Partial RELRO
  - Places GOT before BSS, preventing buffer overflows from a global variable
  - Not very useful defence if we have arbitrary write
- Full RELRO
  - Makes entire GOT read only, preventing GOT overwrites
  - Still able to read GOT for address leaks

Questions?