Review

Stuart Nevans Locke

Overview

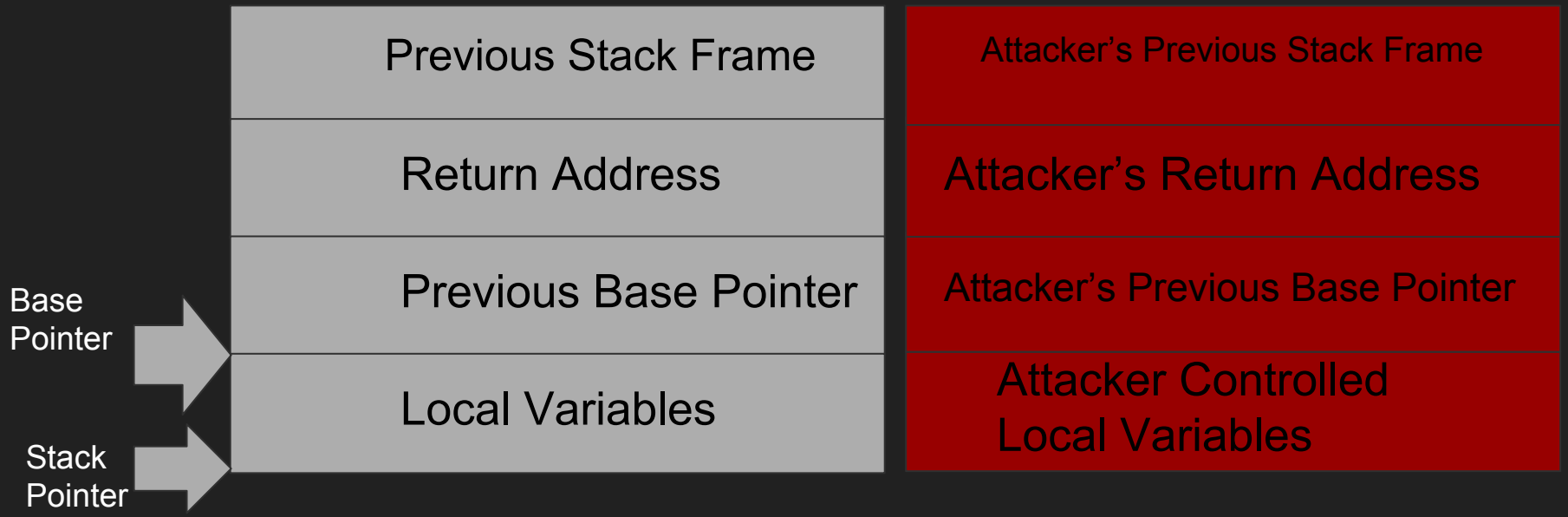
- Stack Frames
- Stack Overflows
- ROP
- Mitigations
  - NX
  - ASLR
  - PIE
  - Stack Canaries
- Elf Structure
  - GOT+PLT
- Mitigation
  - RELRO

Stack Frames

- Contains information about function
  - Return address
  - Previous Base Pointer
  - Local Variables
    - In some cases parameters

Stack Overflows

- We read too much data onto the stack
  - Overwrite everything!

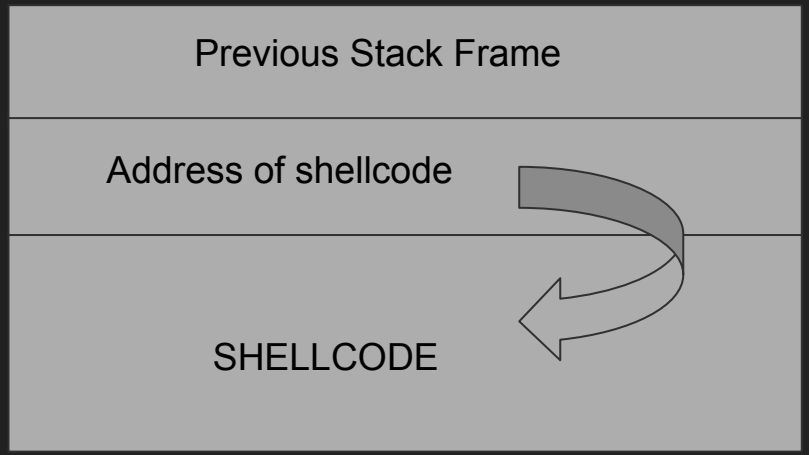


ROP

- Be intelligent in our overwriting
  - *ret == pop rip*
  - If we overwrite the return address to point to a ROP gadget, we can execute multiple pieces of code
    - ROP gadget: something ending with *ret*

Stack Overflows - Shellcode

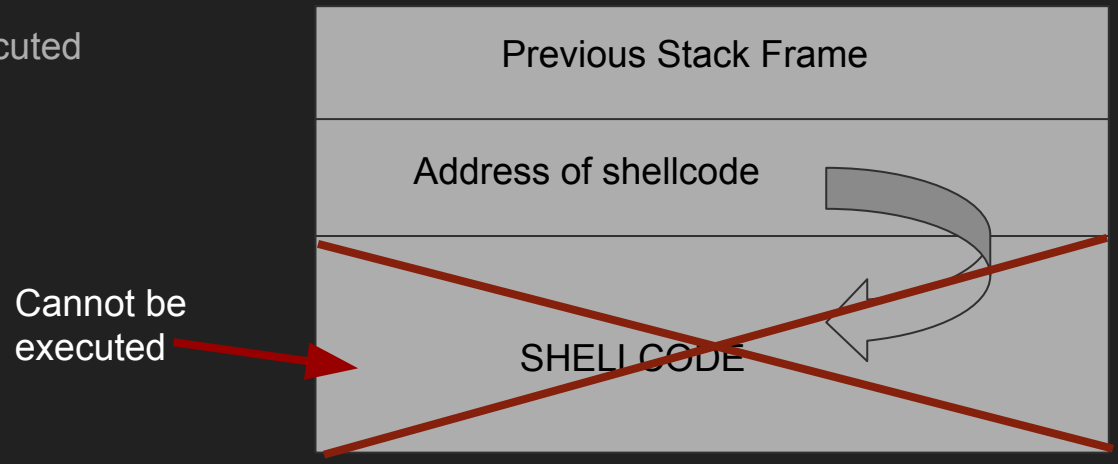
- Older way of exploiting overflows
- Point the return address into a buffer we control
  - Contains “shellcode”
    - Code that gives us a shell when run



Shellcode itself can sometimes be used, but this method is outdated

Mitigations - NX

- NX / DEP / W^X
  - Non Executable
  - Data Execution Prevention
  - Writable XOR Executable
- What does this mean for us?
  - The stack cannot be executed
  - No shellcode
    - D:
- How to bypass?
  - ROP



Mitigations - ASLR

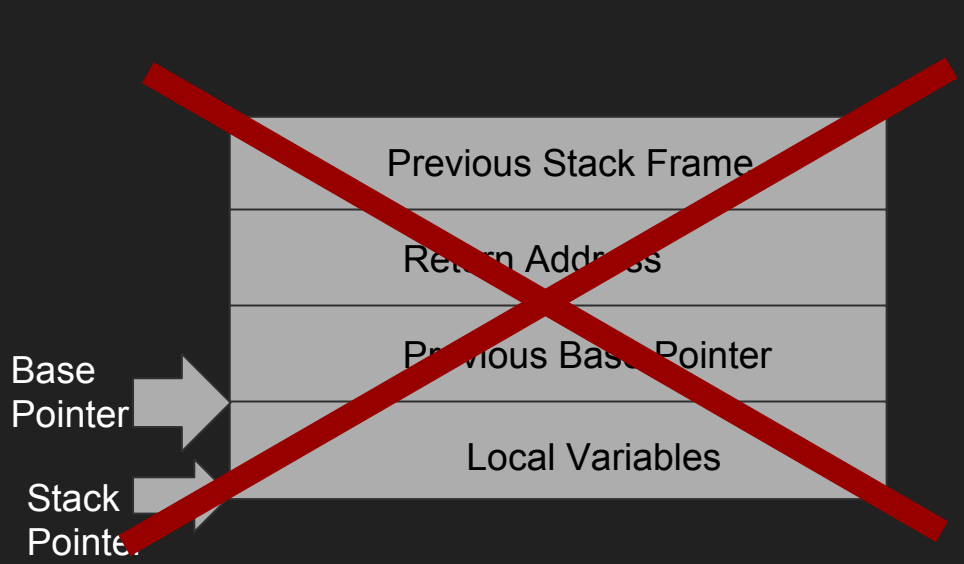
- Address Space Layout Randomization
- You can’t return to what you can’t find!
  - Randomizes libc
  - Randomizes Stack
  - Randomizes Heap
- Notably, our executable does NOT get randomized
- How to bypass
  - Leak Pointer
  - Partial Overwrite #we haven’t talked about
  - Ret2plt #we haven’t talked about
  - ROP in target executable

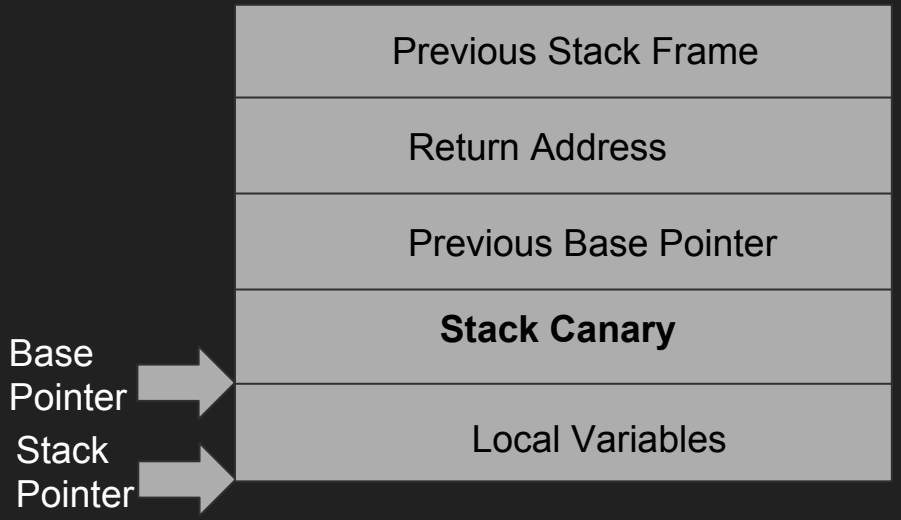
Mitigations - PIE

- Position Independent Execution
- ASLR -- But actually fully applied
- Randomizes the executable location
- Technically within the subset of ASLR
- How to bypass:
  - Leak Pointer
  - Partial Overwrite #we haven’t talked about

Mitigations - Stack Canaries

- Make stack overflows impossible to exploit
- Put random value on stack
- Check that it hasn't changed before returning





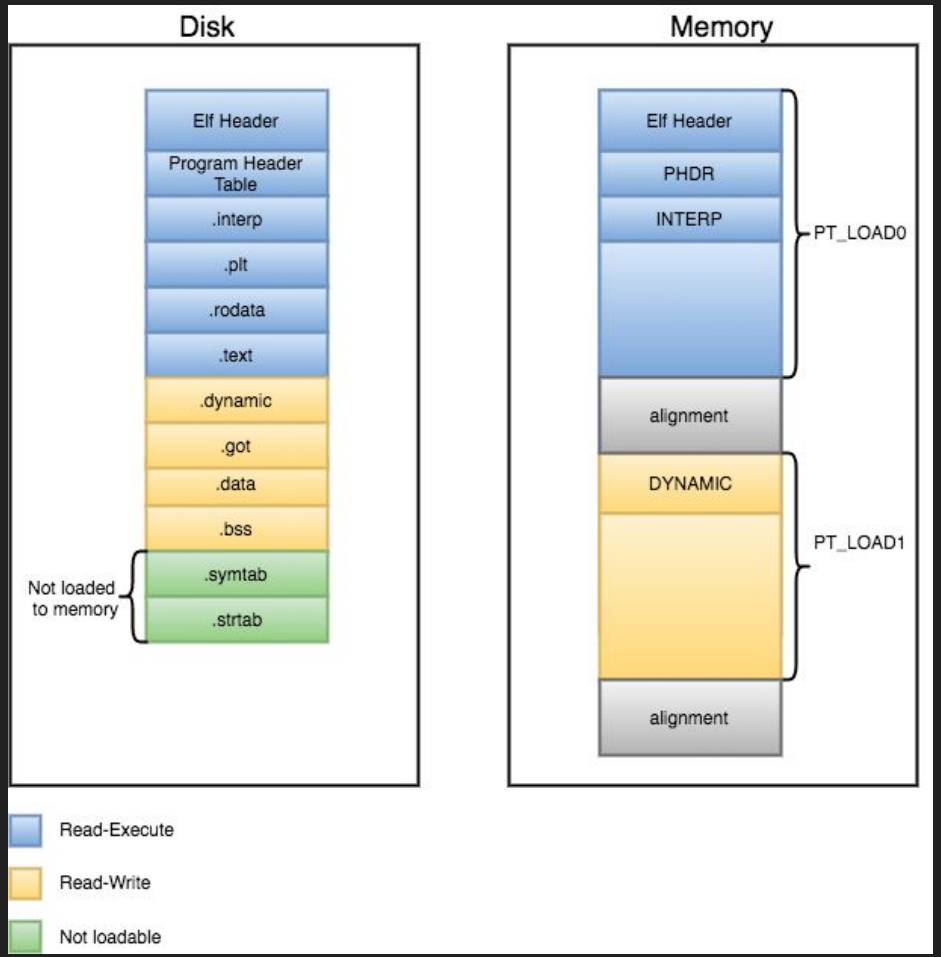
Tool - Checksec

- How to tell which ones are enabled
  - checksec /path/to/file
  - Comes with pwntools
- ASLR is actually enabled system wide, it’s by default on

```
[st@localhost got1]$ checksec ./got1
[*] '/home/st/Desktop/teaching/3/got1/got1'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

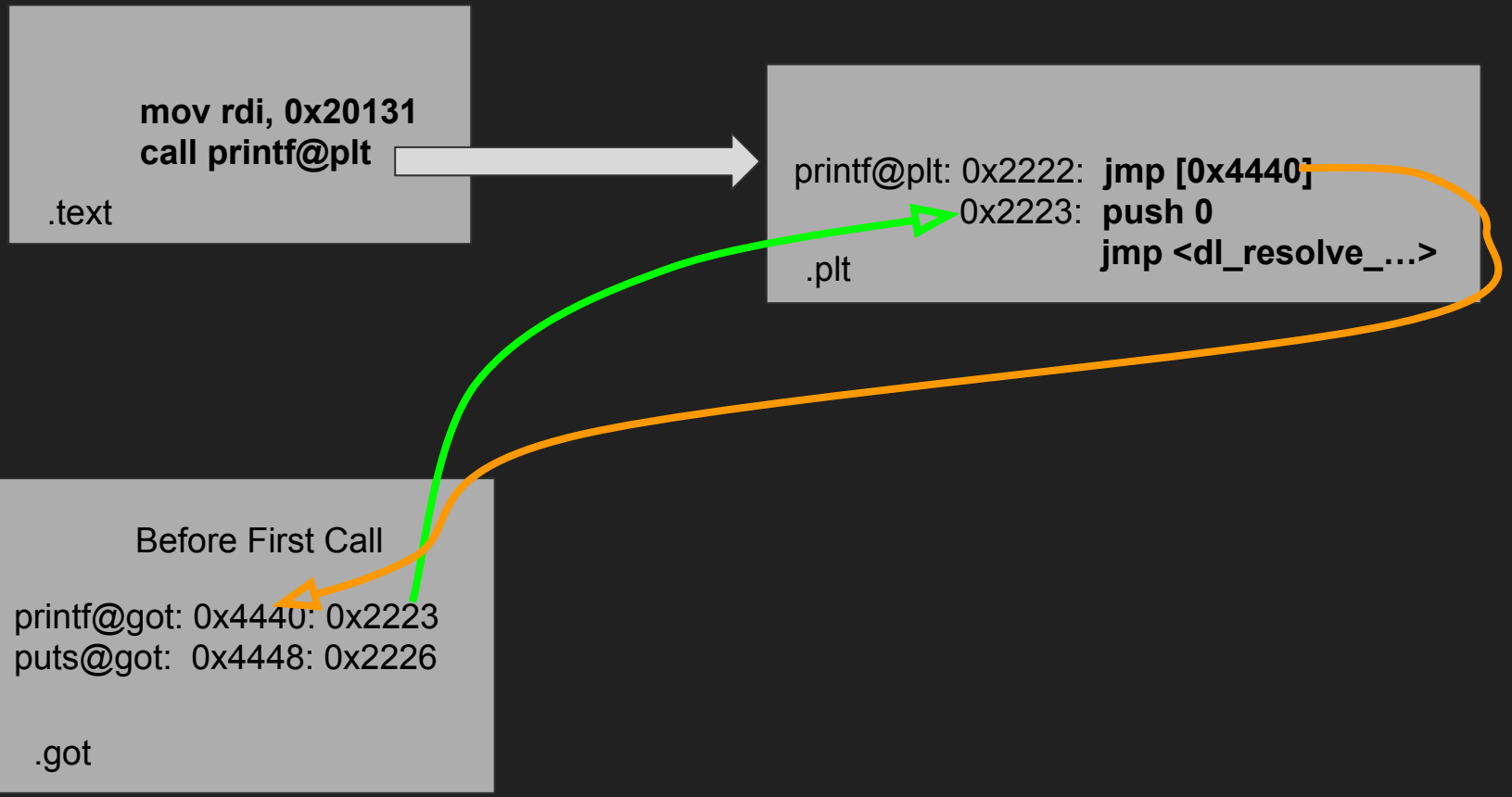
Elf Structure

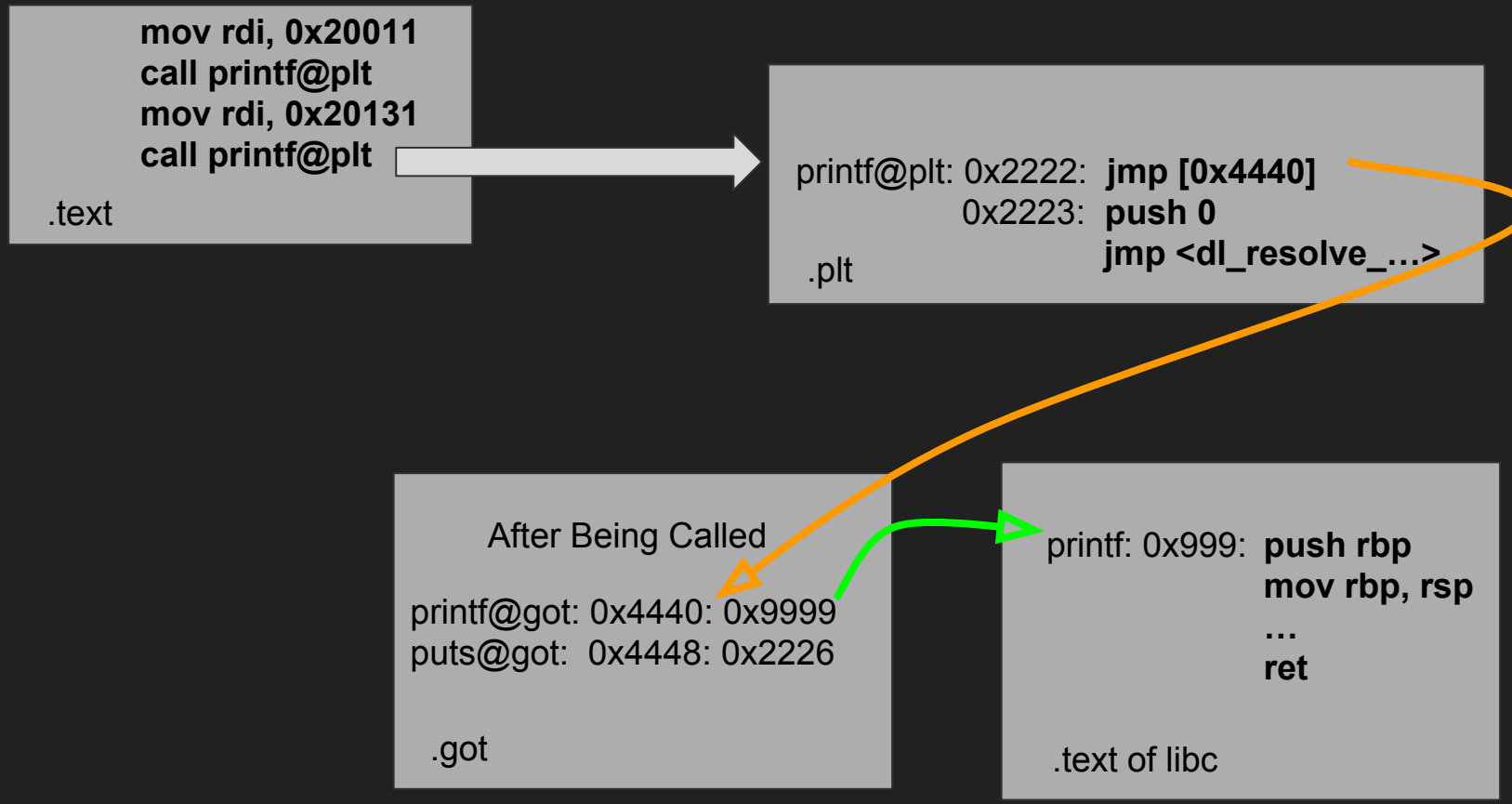
- Lots of sections
- Interesting Ones:
  - .text
  - .plt
  - .got
- No Write+Execute
  - That's DEP at work!



Sections

- .text
  - Contains all the code we write
  - Just assembly
- .plt
  - Trampoline for external calls
  - Example: printf, puts, fgets, system, …
  - Handles dealing with ASLR to find true addresses of external functions
- .got
  - Table of addresses
  - Used by plt to store true addresses of external functions


mov rdi, 0x20131
call printf@plt
.text
printf@plt: 0x2222: jmp [0x4440]
0x2223: push 0
jmp <dl_resolve_...>
.plt
Before First Call
printf@got: 0x4440: 0x2223
puts@got: 0x4448: 0x2226
.got


mov rdi, 0x20011
call printf@plt
mov rdi, 0x20131
call printf@plt
.text
printf@plt: 0x2222: jmp [0x4440]
0x2223: push 0
jmp <dl_resolve_…>
.plt
After Being Called
printf@got: 0x4440: 0x9999
puts@got: 0x4448: 0x2226
.got
printf: 0x999: push rbp
mov rbp, rsp
…
ret
.text of libc

Mitigation - RELRO

- RELocations Read Only
- Because of how the GOT is lazily loaded, it is writeable
- Full RELRO:
  - GOT is filled at load time rather than runtime
  - GOT is not writeable
- Partial RELRO:
  - Default
  - Doesn’t mean anything to us

Questions or Comments

- Anything anyone is remotely hazy on