ELF Structure + Defeating ASLR

Stuart Nevans Locke

Overview

- Review
  - Stack Overflow + ROP
    - ASLR
- ELF Structure
  - Overview
  - Read+Execute (.text, .plt)
  - Read+Write (.got, .bss, .data)
- Tool - Checksec
- Bypassing ASLR
  - got leaks
    - Demos
  - Partial Overwrite
  - Ret2plt
    - Demos

Quick Note

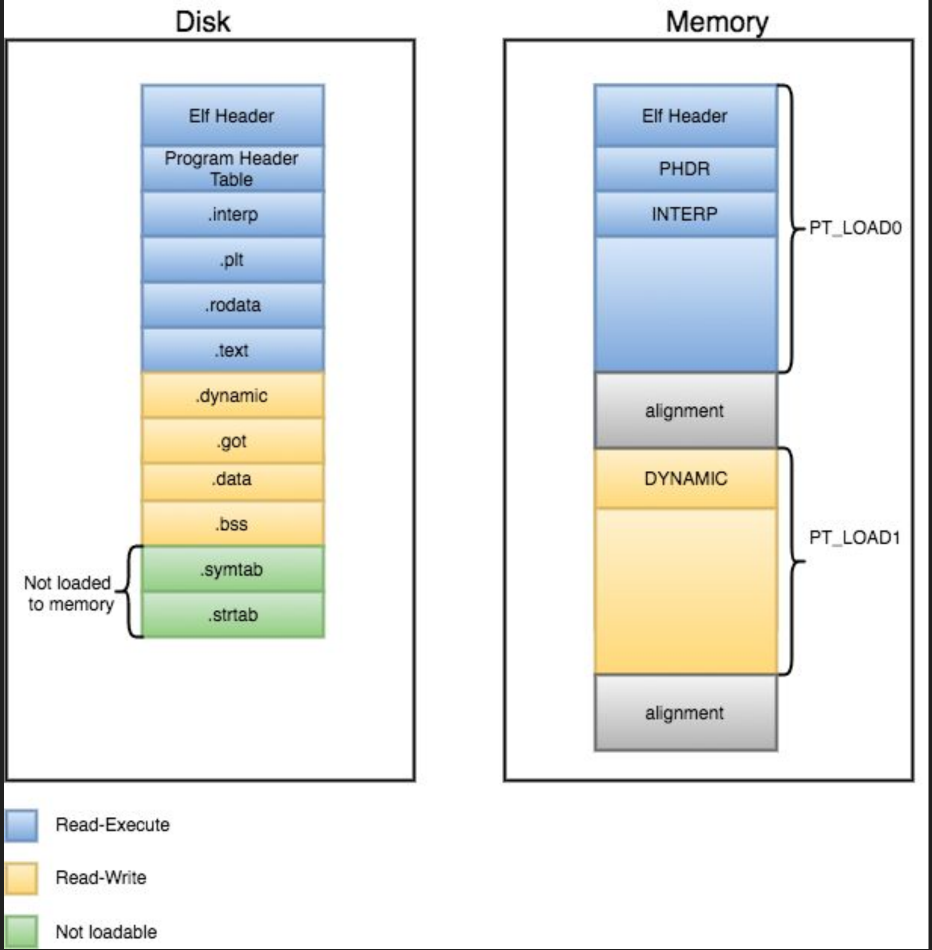
- If I use a term you haven’t heard of, interrupt me
    - I probably just forgot to explain it

Stack Overflow / ROP

- Stack Overflow
  - Allows us to call any ROP gadgets in the target binary
  - This lets us bypass DEP (Data Execution Prevention)
- Problem:
  - There probably aren’t always going to be great ROP gadgets
- ASLR
  - Address Space Layout Randomization
  - We don’t know where libc is loaded, meaning we don’t know where system is in memory

ELF Structure

- It will greatly help to know what an ELF file looks like
- Lots of sections
  - VERY IMPORTANT
    - .text
    - .plt
    - .got
  - Interesting too
    - .bss
    - .data
- DEP
  - W^X (Nothing is both Writeable and Executable)

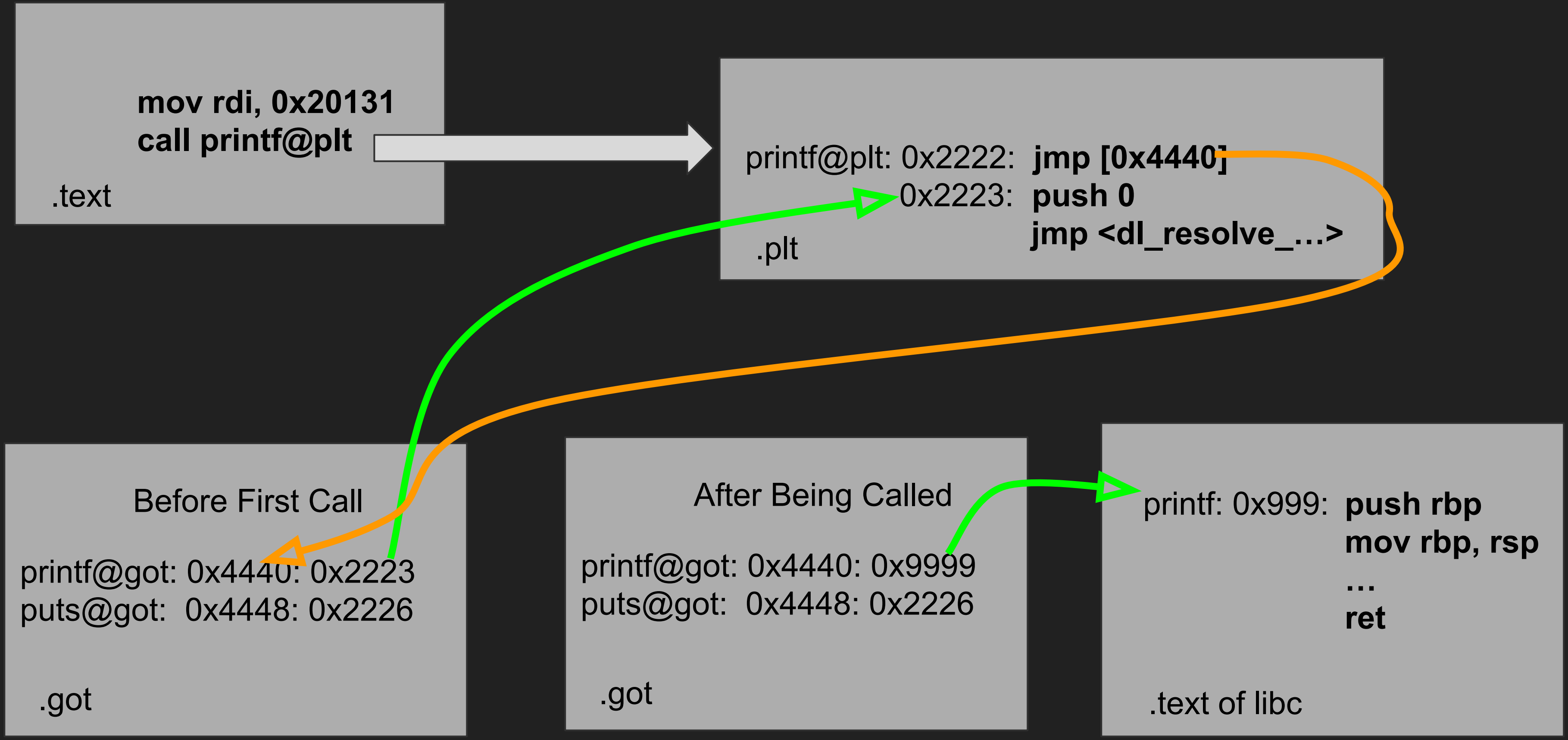


ELF Structure (Read+Execute)

- Code is readable and executable
- .text
    - This contains all the code for a binary (all the code you write goes here)
- .plt
    - Procedure Linkage Table
    - Used to handle calls to external functions
    - For example, let’s say you call printf() in some function
        - The code for printf isn’t compiled into your executable
        - Instead, it’s dynamically linked
            - This means the address is resolved at runtime
            - The first time the printf@plt is called, the address is resolved and stored to the GOT
        - plt is basically a crutch, calls to printf become printf@plt

ELF Structure (Read-Write) (.got)

- .got
  - Global Offset Table
  - Holds the pointer to a specific symbol
  - For example, the got would contain the pointers to system, printf, puts, ...
- RELRO
  - Defines if got is filled lazily or at load time
    - First time printf is called, or when the binary is loaded into memory
  - Partial
    - got is writeable
    - Lazy got filling
  - Full
    - got is not at all writeable
    - got filled at load time



```
    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
```

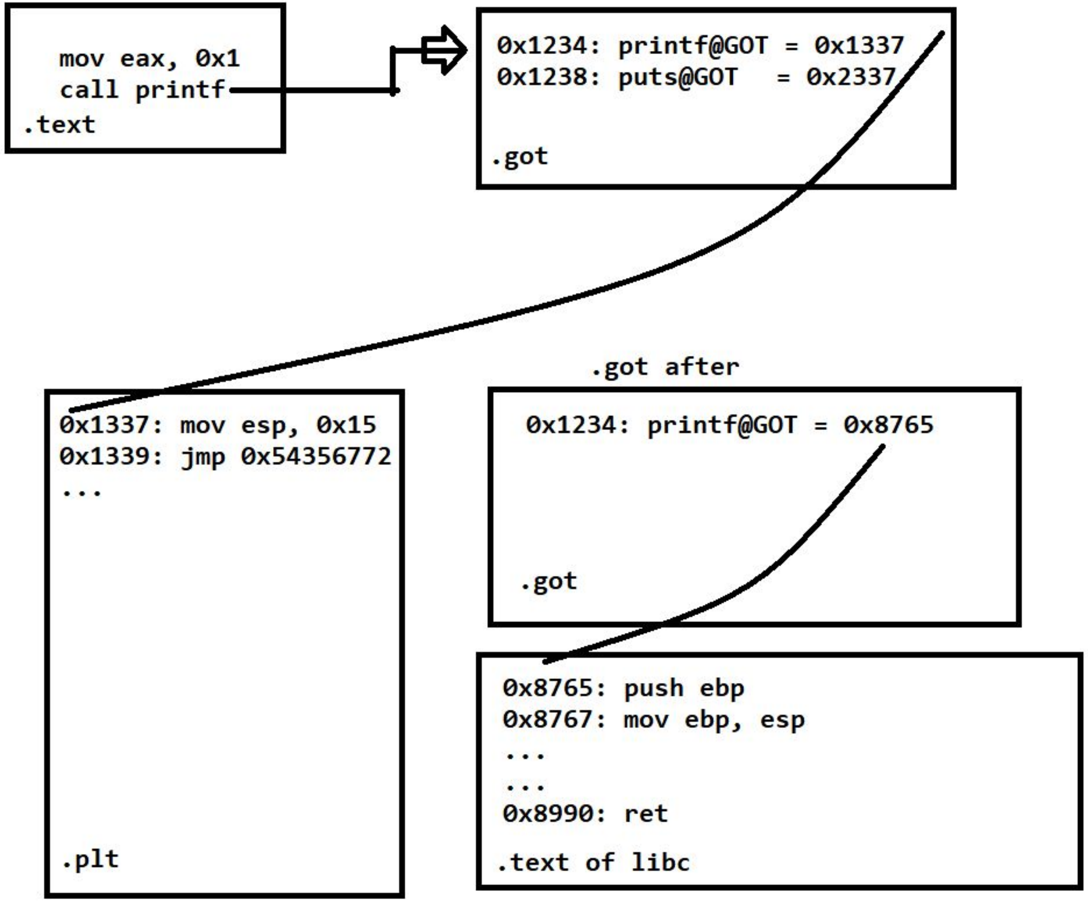
```
      After Being Called

printf@got: 0x4440: 0x9999
puts@got:   0x4448: 0x2226

.got
```

```
printf: 0x999:  push rbp
                mov rbp, rsp
                …
                ret

.text of libc
```



```
mov eax, 0x1
call printf
```

```
0x1234: printf@GOT = 0x1337
0x1238: puts@GOT   = 0x2337
```

```
0x1337: mov esp, 0x15
0x1339: jmp 0x54356772
...
```

```
0x1234: printf@GOT = 0x8765
```

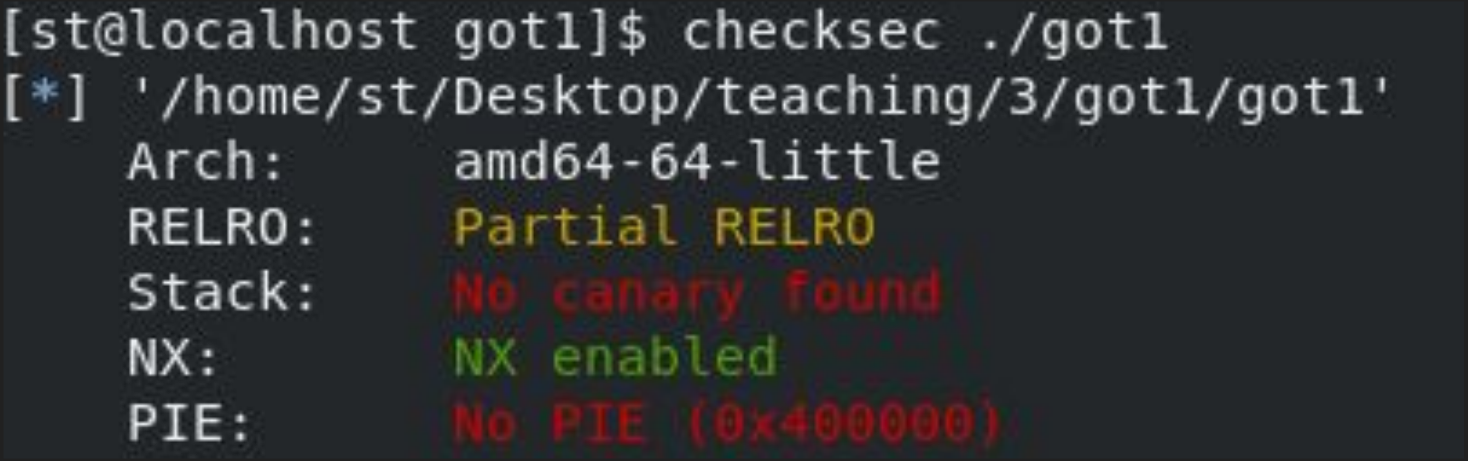
```
0x8765: push ebp
0x8767: mov ebp, esp
...
...
0x8990: ret
```

ELF Structure (Data Sections)

- .bss
  - Uninitialized data
  - char buffer[1024];
- .data
  - Initialized data
  - char buffer[1024]=”I am a buffer”;
- .rodata
  - Read Only Data (Constant)
  - const int x = 2;

Tool - Checksec

- Tool to output information about security property of ELF files
- Stack Canaries
- RELRO (got writeable)
- NX (Non Executable)
- PIE (Position Independent Executable)



```
[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)
```

Bypassing ASLR (GOT leaks)

- Our goal is to find the address of libc
  - This allows us to find the address of system()
    - With the address of system, we can ROP directly to the system() function
  - No more relying on callme functions
- Let’s assume we have some method of reading the data at any address
  - With this, we can bypass ASLR (assuming code is not position independent)
  - We leak the data in the GOT (Global Offset Table)
  - In the GOT, we have pointers to libc and any other imported things

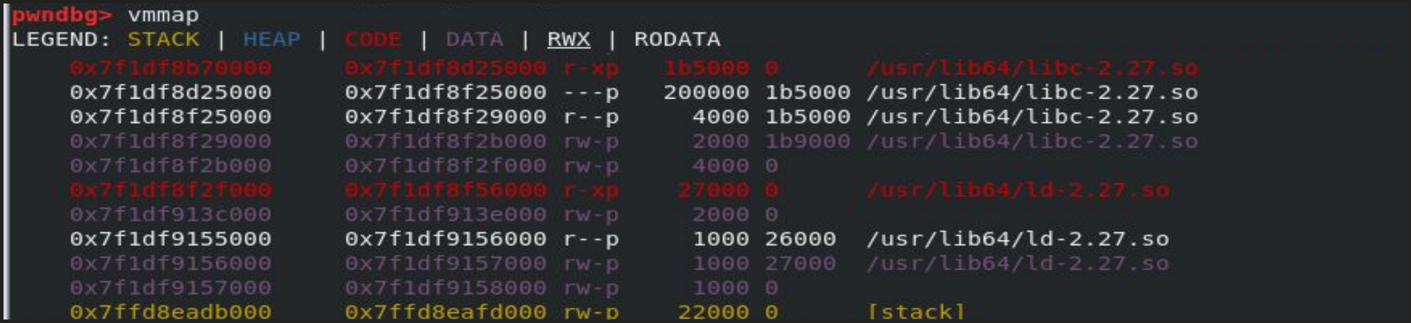
Demos

stnevans.me/3/got1/

stnevans.me/3/got2/

Bypassing ASLR (Partial Overwrite)

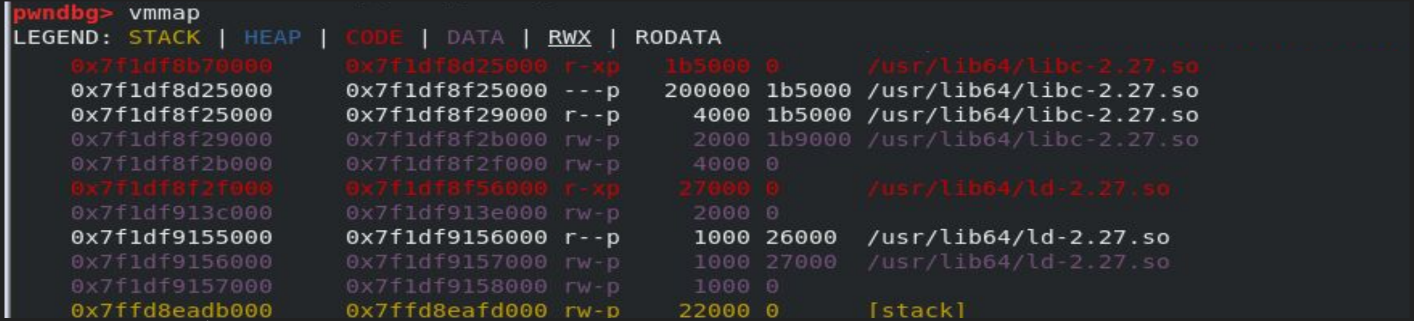
- ASLR only randomizes the higher bytes
  - Page aligned
  - Bottom byte is totally independent of ASLR
- If we have a valid pointer and we only change the bottom byte, it stays valid
  - If we change the second byte, it might not be valid
    - Potentially brute-forceable



```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x7f1df8b70000     0x7f1df8d25000 r-xp   1b5000 0      /usr/lib64/libc-2.27.so
    0x7f1df8d25000     0x7f1df8f25000 ---p   200000 1b5000 /usr/lib64/libc-2.27.so
    0x7f1df8f25000     0x7f1df8f29000 r--p     4000 1b5000 /usr/lib64/libc-2.27.so
    0x7f1df8f29000     0x7f1df8f2b000 rw-p     2000 1b9000 /usr/lib64/libc-2.27.so
    0x7f1df8f2b000     0x7f1df8f2f000 rw-p     4000 0
    0x7f1df8f2f000     0x7f1df8f56000 r-xp    27000 0      /usr/lib64/ld-2.27.so
    0x7f1df913c000     0x7f1df913e000 rw-p     2000 0
    0x7f1df9155000     0x7f1df9156000 r--p     1000 26000  /usr/lib64/ld-2.27.so
    0x7f1df9156000     0x7f1df9157000 rw-p     1000 27000  /usr/lib64/ld-2.27.so
    0x7f1df9157000     0x7f1df9158000 rw-p     1000 0
    0x7ffd8eadb000     0x7ffd8eafd000 rw-p    22000 0      [stack]
```

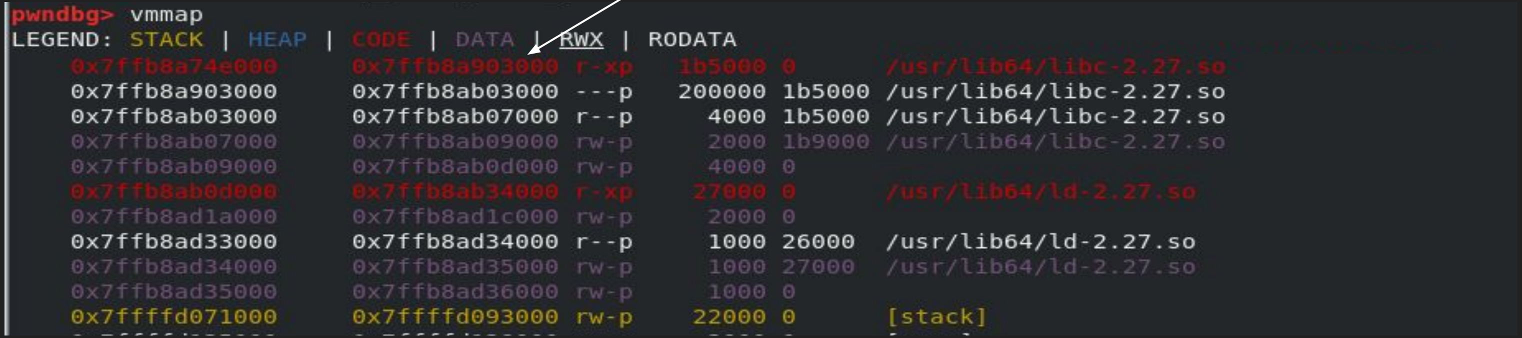
Page Aligned
Addresses

Bypassing ASLR(Partial Overwrite)



```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x7f1df8b70000     0x7f1df8d25000 r-xp   1b5000 0      /usr/lib64/libc-2.27.so
    0x7f1df8d25000     0x7f1df8f25000 ---p   200000 1b5000 /usr/lib64/libc-2.27.so
    0x7f1df8f25000     0x7f1df8f29000 r--p     4000 1b5000 /usr/lib64/libc-2.27.so
    0x7f1df8f29000     0x7f1df8f2b000 rw-p     2000 1b9000 /usr/lib64/libc-2.27.so
    0x7f1df8f2b000     0x7f1df8f2f000 rw-p     4000 0
    0x7f1df8f2f000     0x7f1df8f56000 r-xp    27000 0      /usr/lib64/ld-2.27.so
    0x7f1df913c000     0x7f1df913e000 rw-p     2000 0
    0x7f1df9155000     0x7f1df9156000 r--p     1000 26000  /usr/lib64/ld-2.27.so
    0x7f1df9156000     0x7f1df9157000 rw-p     1000 27000  /usr/lib64/ld-2.27.so
    0x7f1df9157000     0x7f1df9158000 rw-p     1000 0
    0x7ffd8eadb000     0x7ffd8eafd000 rw-p    22000 0      [stack]
```

Page Aligned (Two different runs)



```
pwndbg> vmmap
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
    0x7ffb8a74e000     0x7ffb8a903000 r-xp   1b5000 0      /usr/lib64/libc-2.27.so
    0x7ffb8a903000     0x7ffb8ab03000 ---p   200000 1b5000 /usr/lib64/libc-2.27.so
    0x7ffb8ab03000     0x7ffb8ab07000 r--p     4000 1b5000 /usr/lib64/libc-2.27.so
    0x7ffb8ab07000     0x7ffb8ab09000 rw-p     2000 1b9000 /usr/lib64/libc-2.27.so
    0x7ffb8ab09000     0x7ffb8ab0d000 rw-p     4000 0
    0x7ffb8ab0d000     0x7ffb8ab34000 r-xp    27000 0      /usr/lib64/ld-2.27.so
    0x7ffb8ad1a000     0x7ffb8ad1c000 rw-p     2000 0
    0x7ffb8ad33000     0x7ffb8ad34000 r--p     1000 26000  /usr/lib64/ld-2.27.so
    0x7ffb8ad34000     0x7ffb8ad35000 rw-p     1000 27000  /usr/lib64/ld-2.27.so
    0x7ffb8ad35000     0x7ffb8ad36000 rw-p     1000 0
    0x7ffffd071000     0x7ffffd093000 rw-p    22000 0      [stack]
```

Bypassing ASLR (re2plt)

- As mentioned before, the plt is used to resolve dynamically linked functions
  - If we call the plt stub to a function, we don’t have to worry about ASLR
    - plt automatically looks up the address in the got and locates the function
- We don’t need leaks if we can return to the plt

Demos

stnevans.me/binex/3/aslr1

stnevans.me/binex/3/hard

If you can do the hard one, you have a pretty solid handle on the elf structure and ROP

Thanks to Duc again