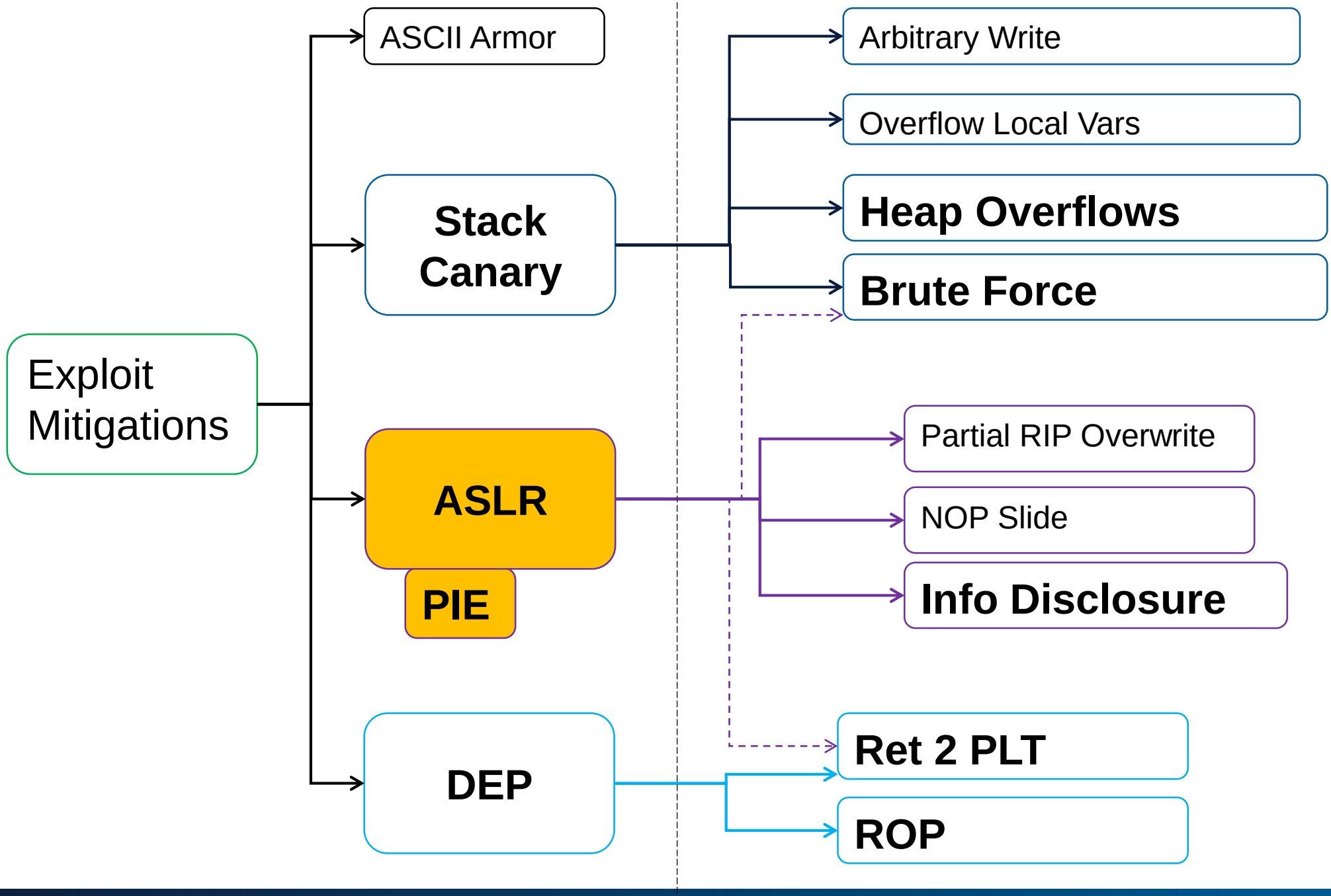
Exploit Mitigation - PIE


Exploit Mitigations
ASCII Armor
Stack Canary
ASLR
PIE
DEP
Arbitrary Write
Overflow Local Vars
Heap Overflows
Brute Force
Partial RIP Overwrite
NOP Slide
Info Disclosure
Ret 2 PLT
ROP

Recap! Exploit Mitigation Exploits

All three exploit mitigations can be defeated by black magic

Easily

Is there a solution?

Exploit Mitigation - PIE

The solution

The solution to all problems… PIE

Exploit Mitigation++

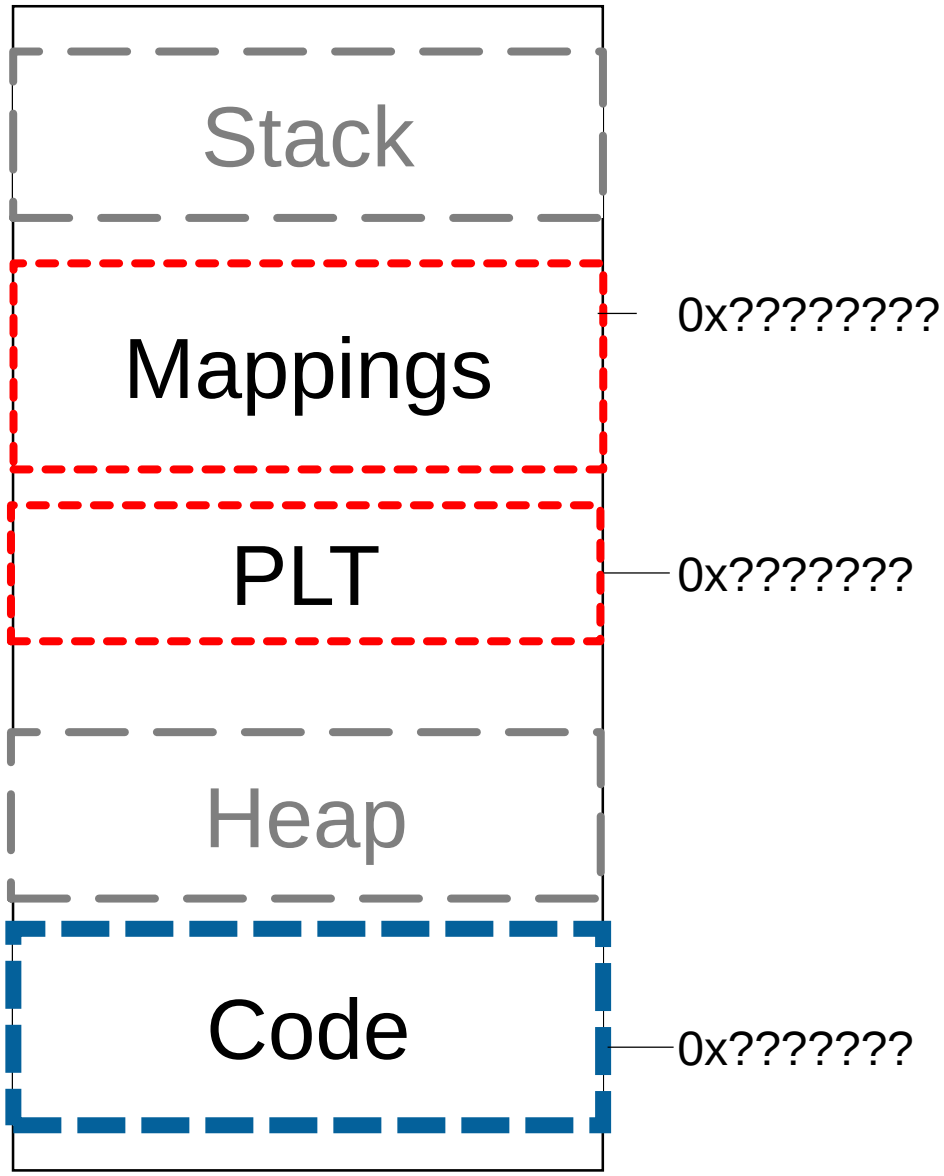
Fix:

- Compile as PIE
- PIE: Position Independent Executable
- Will randomize Code and PLT, too

Note:

- Shared libraries are PIC
  - (Position Independent Code)
- Because they don’t know where they are being loaded
- Always randomized, even without PIE

Exploiting: ASLR for code: PIE



PIE Executable

```
$ cat test.c
#include <stdio.h>

void func() {
        printf("\n");
}
void main(void) {
        printf("%p\n", &func);
}
$ gcc -fpic -pie test.c
$ ./a.out
0x557d9dee57c5
$ ./a.out
0x5581df9d67c5
```

PIE Executable

```
Type           Offset             VirtAddr           PhysAddr
               FileSiz            MemSiz              Flags  Align
  PHDR         0x0000000000000040 0x0000000000000040 0x0000000000000040
               0x00000000000001f8 0x00000000000001f8  R E    8
  INTERP       0x0000000000000238 0x0000000000000238 0x0000000000000238
               0x000000000000001c 0x000000000000001c  R      1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD         0x0000000000000000 0x0000000000000000 0x0000000000000000
               0x00000000000009dc 0x00000000000009dc  R E    200000
[…]
 Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gn
u.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata
```

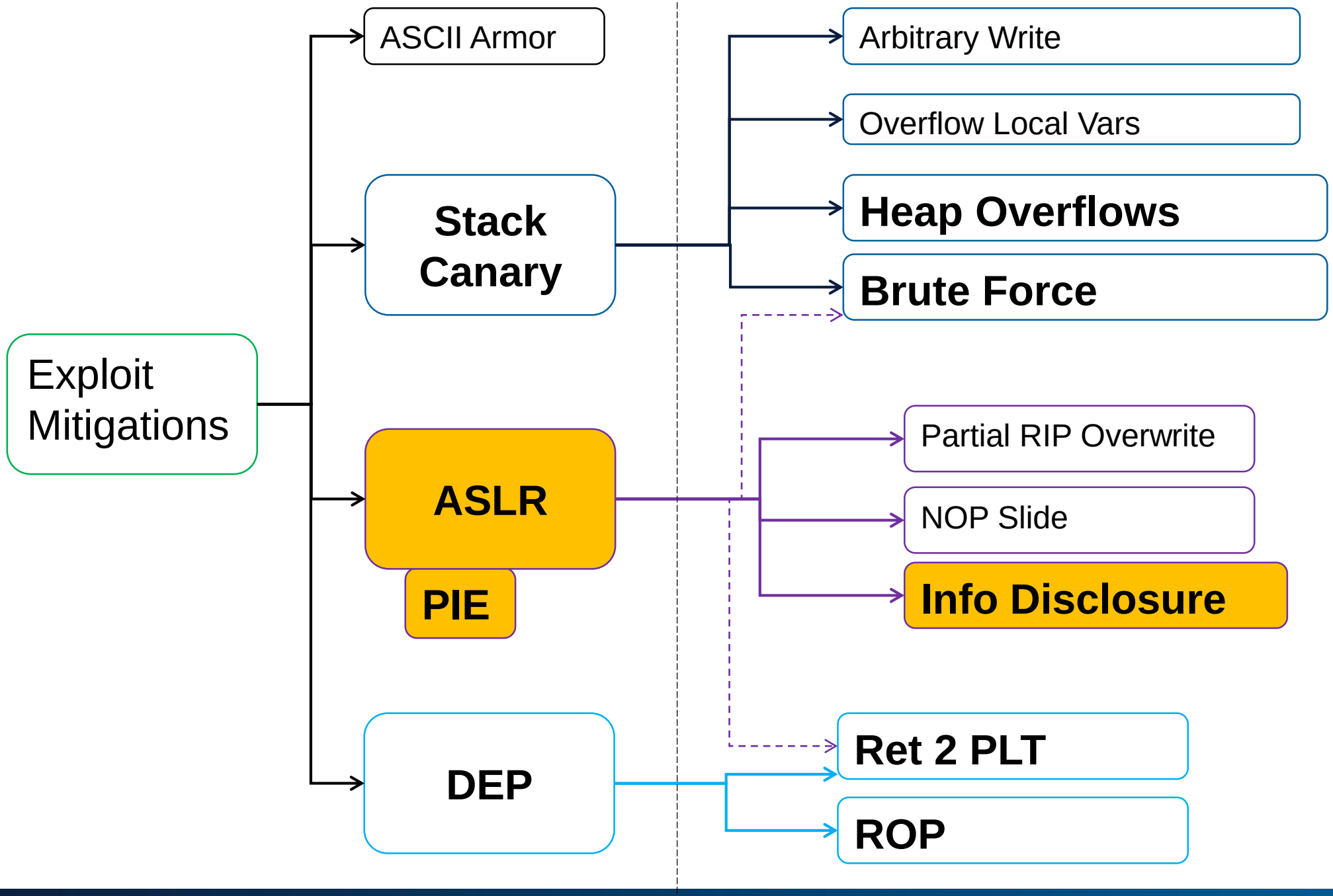
Exploiting: ASLR for code: PIE

PIE randomizes Code segment base address

PIE randomizes GOT/PLT base address too

No more static locations!

Defeat Exploit Mitigation: PIE


Exploit Mitigations
ASCII Armor
Stack Canary
ASLR
PIE
DEP
Arbitrary Write
Overflow Local Vars
Heap Overflows
Brute Force
Partial RIP Overwrite
NOP Slide
Info Disclosure
Ret 2 PLT
ROP


[the cake is a lie]

ASLR vs Information Leak

ASLR assumes attacker can’t get information

What if they can?

Meet: Memory Leak

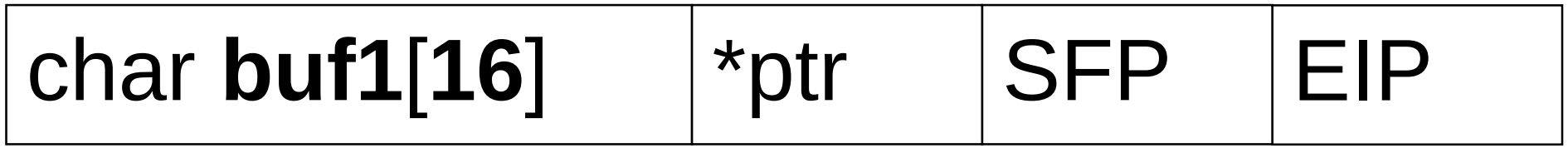
Memory Leak / Information Disclosure

Memory Leak

Memory leak or information disclosure:

- Return more data to the attacker than the intended object size
- The data usually includes meta-data, like:
  - Stack pointers
  - Return addresses
  - Heap-management data
  - Etc.

ASLR vs Memory Leak



Server:

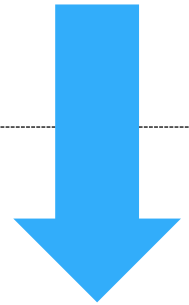
```
send(socket, buf1, sizeof(int) * 16, NULL);
```

- Oups, attacker got 64 bytes back
  - Pointer to stack, code, heap
  - Can deduce base address

ASLR vs Memory Leak

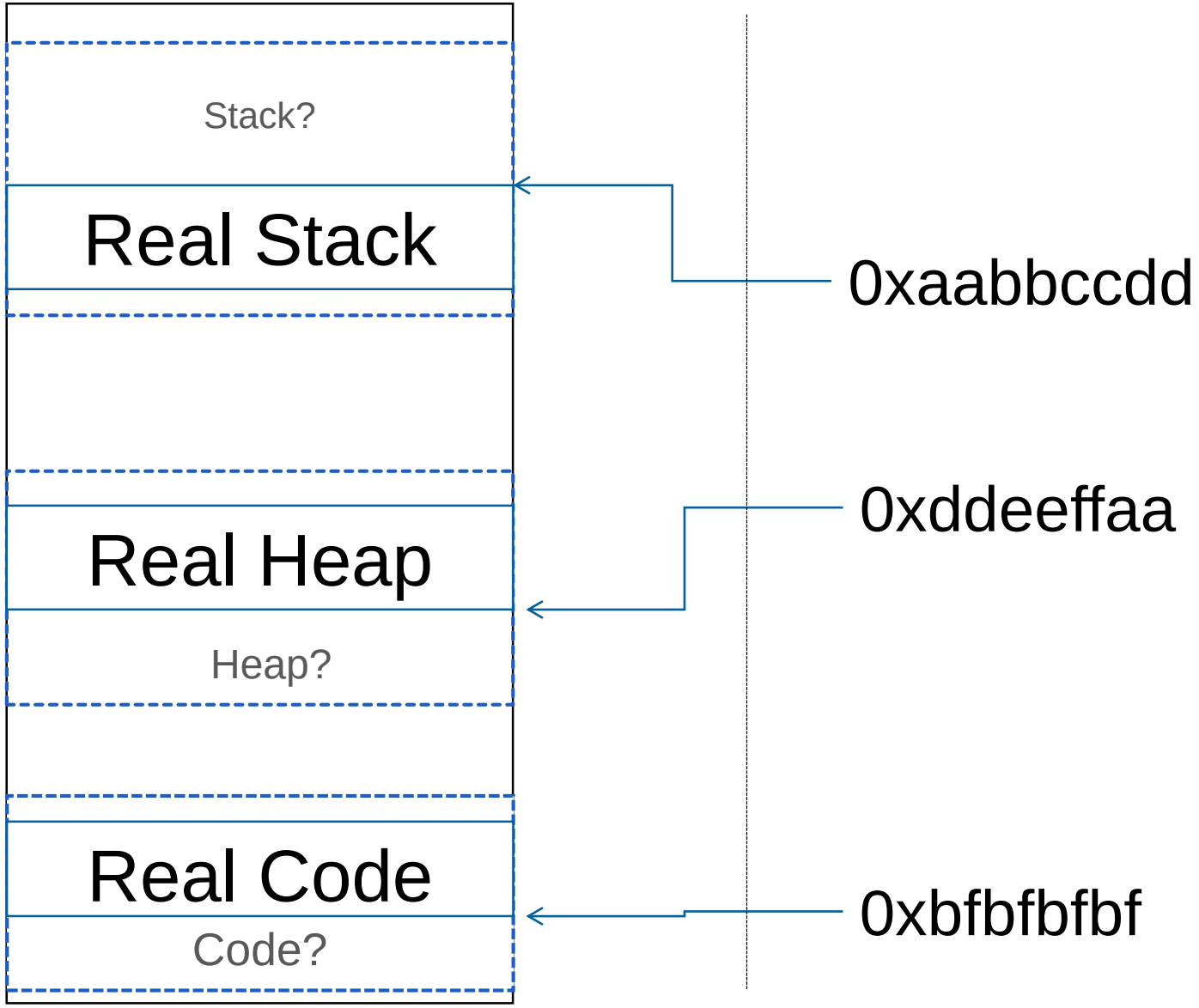
char **buf1**[16]	*ptr	SFP	EIP

```
send(socket, buf1, sizeof(int) * 16, NULL);
```

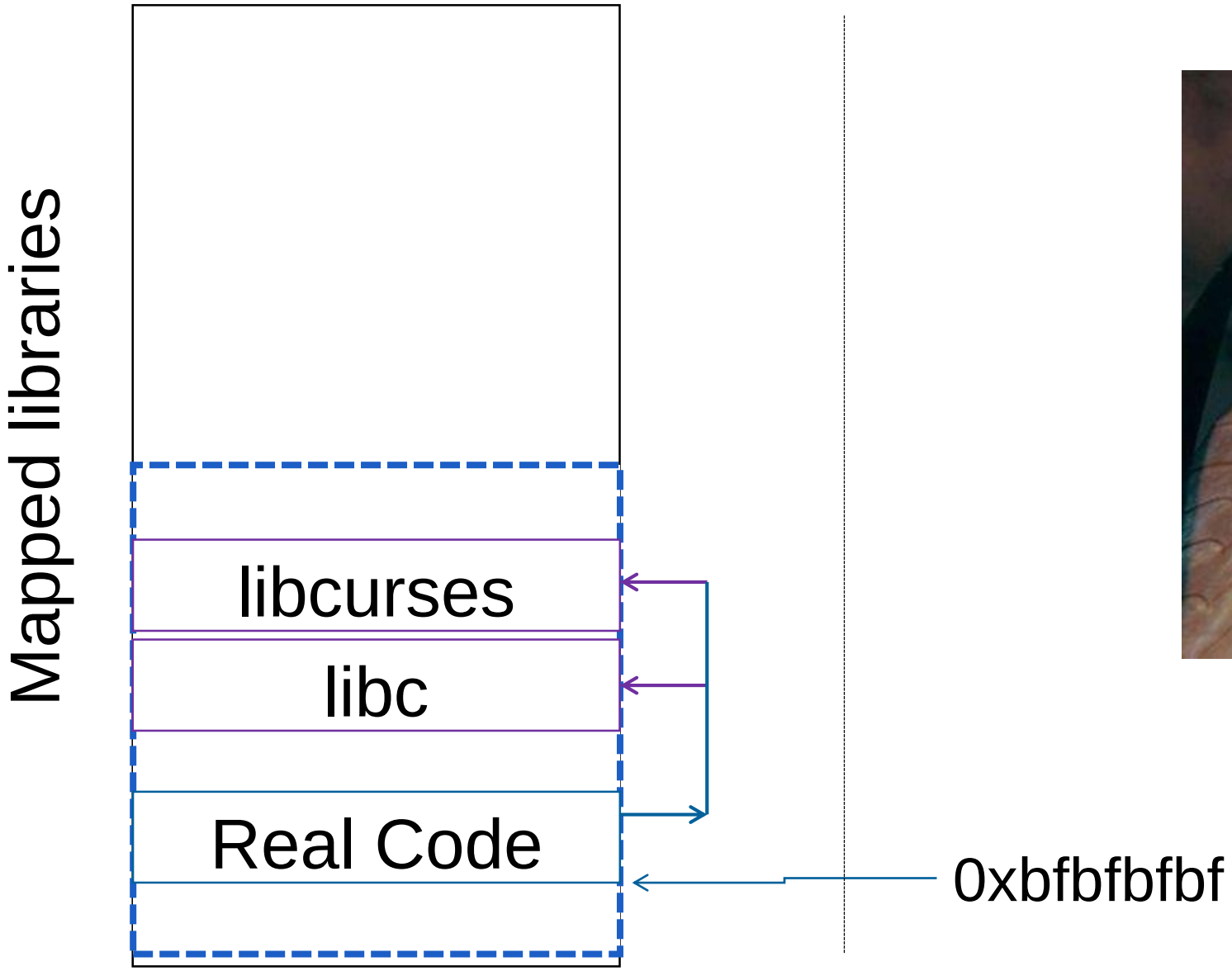


char **buf1**[16]	*ptr	SFP	EIP

Exploiting: ASLR for code: PIE



Exploiting: ASLR for code: PIE



Exploiting: ASLR for code: PIE

Attacker:

- Information disclosure / memory leak
- Gains a pointer (Address of memory location)
- From pointer: Deduct base address of segment
- From base address: Can deduct all other addresses

~~A note on code -> libraries:~~

- ~~Distance between code segment and mapped libraries is usually constant~~
- ~~Got SIP? Can use LIBC gadgets…~~

Exploiting: ASLR for code: PIE

Example: Windows memory disclosure (unpatched, 21.2.17, CVE-2017-0038)

```
As a consequence, the 16x16/24bpp bitmap is now described by just 4 bytes, which
is good for only a single pixel. The remaining 255 pixels are drawn based on
junk heap data, which may include sensitive information, such as private user
data or information about the virtual address space.
```

Windows gdi32.dll heap-based out-of-bounds reads / memory disclosure in EMR_SETDIBITSTODEVICE and possibly other records

‹ Prev 2 of 4 Next ›

Back to list

Project Member Reported by mjurczyk@google.com, Nov 16

In ~~issue #757~~, I described multiple bugs related to the handling of DIBs (Device Independent Bitmaps) embedded in EMF records, as implemented in the user-mode Windows GDI library (gdi32.dll). As a quick reminder, the DIB-embedding records follow a common scheme: they include four fields, denoting the offsets and lengths of the DIB header and DIB data (named offBmiSrc, cbBmiSrc, offBitsSrc, cbBitsSrc). A correct implementation should verify that:


GIT THE PRINCESS!
HOW TO SAVE THE PRINCESS USING 8 PROGRAMMING LANGUAGES
BY toggl Goon Squad
YOU HAVE JAVASCRIPT
YOU SPEND HOURS PICKING LIBRARIES, SETTING UP NODE & BUILDING A FRAMEWORK FOR THE CASTLE.
BY THE TIME YOU'RE FINISHED WITH THE FRAMEWORK, THE FORT HAS BEEN ABANDONED AND THE PRINCESS HAS MOVED TO ANOTHER CASTLE
YOU HAVE C
YOU HAVE A LIBRARY FOR A CASTLE & A LIBRARY FOR THE PRINCESS - CHARGE!
YOU RESCUE THE PRINCESS HER DOG, HER ENTIRE WARDROBE & EVERYTHING SHE HAS EVER EATEN...
FUCK - DID I FORGET A NULL-TERMINATOR?

Exploit Mitigation Conclusion

Defeat Exploit Mitigations: TL;DR

Enable ALL the mitigations (DEP, ASLR w/PIE, Stack Protector)

- Defeat ALL the mitigations:
  - ROP shellcode as stager to defeat DEP
  - Information leak to defeat ASLR
  - Non stack-based-stack-overflow vulnerability

Recap

Information disclosure can eliminate ASLR protection

Which enables ROP to eliminate DEP

References

References:

- ROP CFI RAP XNR CPI WTF? Navigating the Exploit Mitigation Jungle
  - https://bsidesljubljana.si/wp-content/uploads/2017/02/ropcfirapxnrcpiwtf-rodler-bsidesljubljana2017.pdf