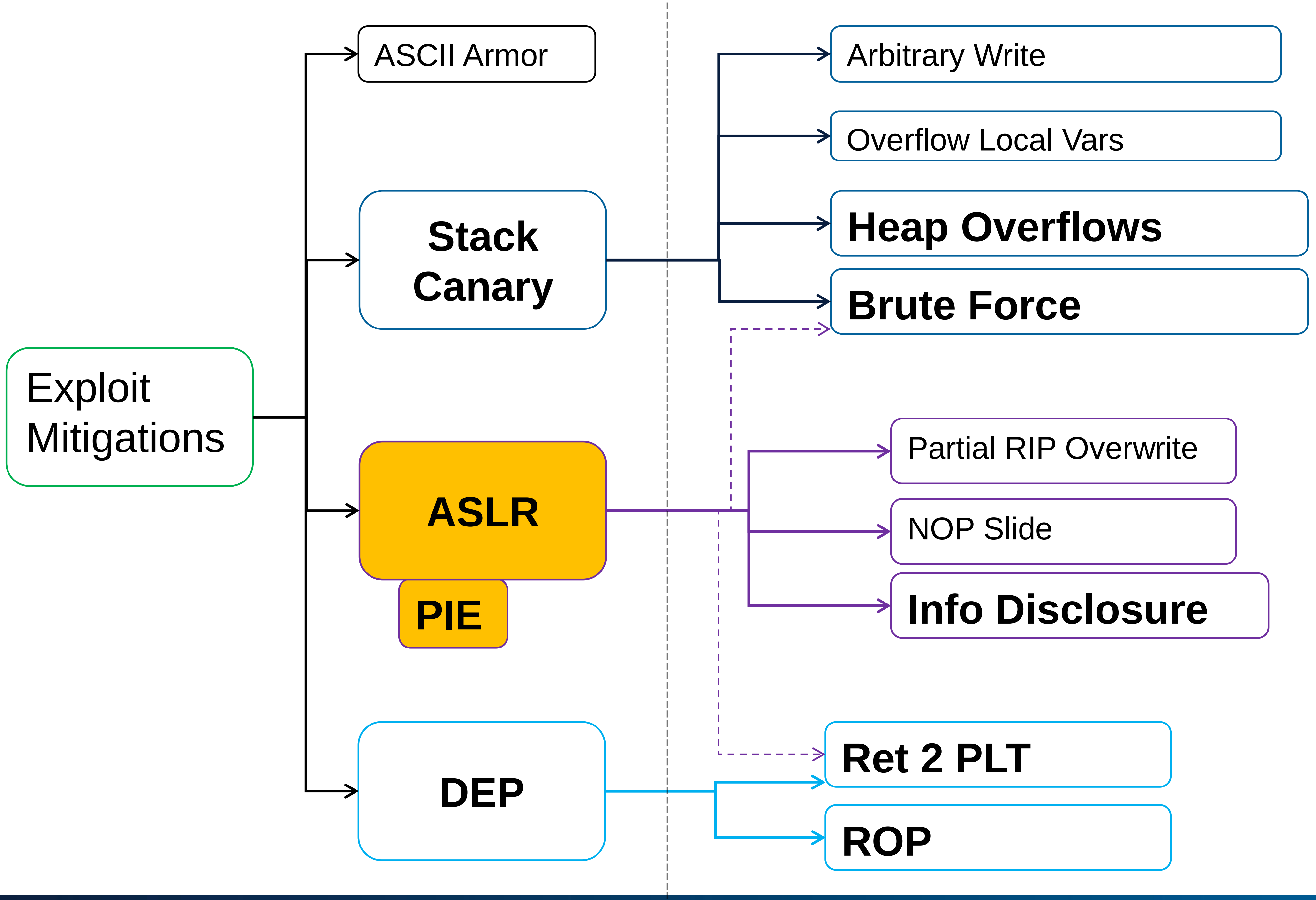
Exploit Mitigation - PIE


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++

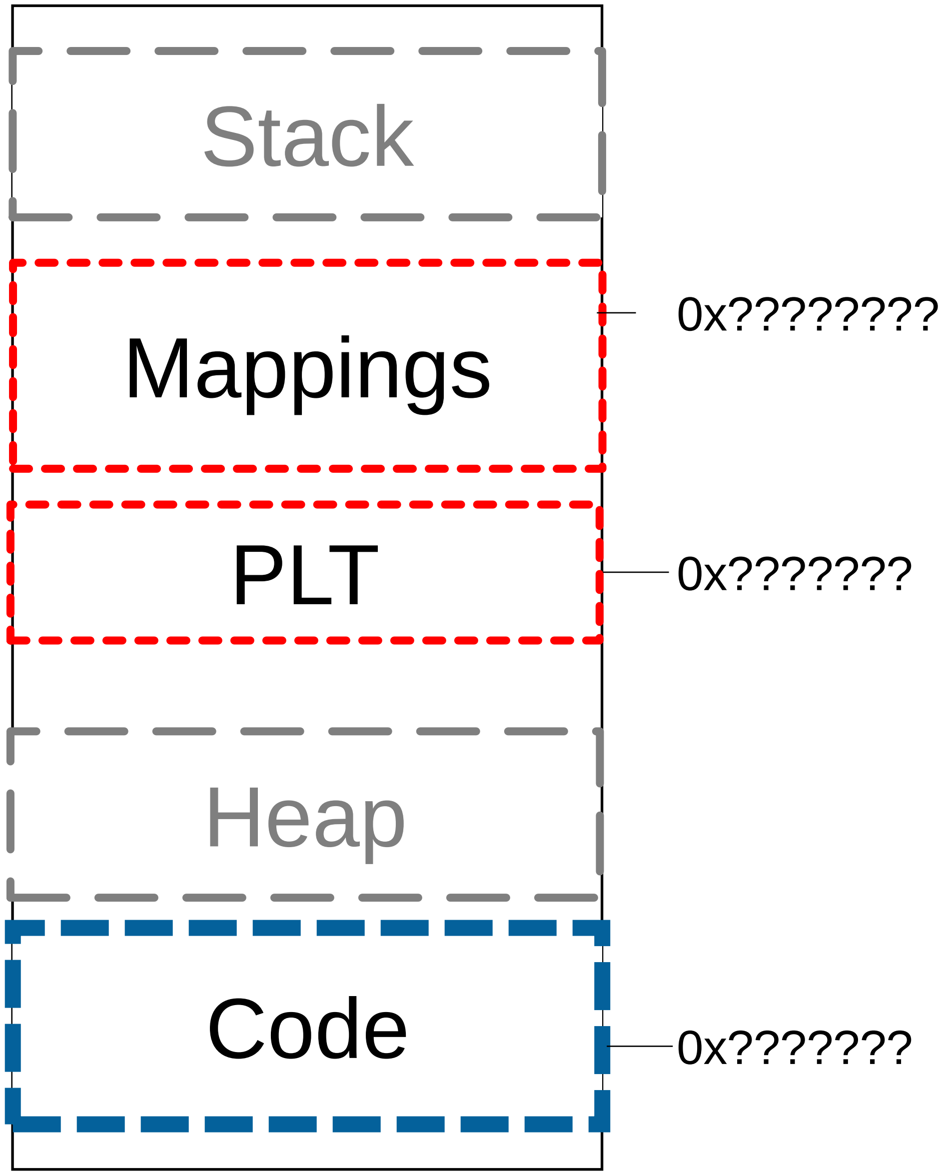
Fix:

- 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 .gnu.version .gnu.version_r .rela.dyn .rela.plt .init .plt .text .fini .rodata
```

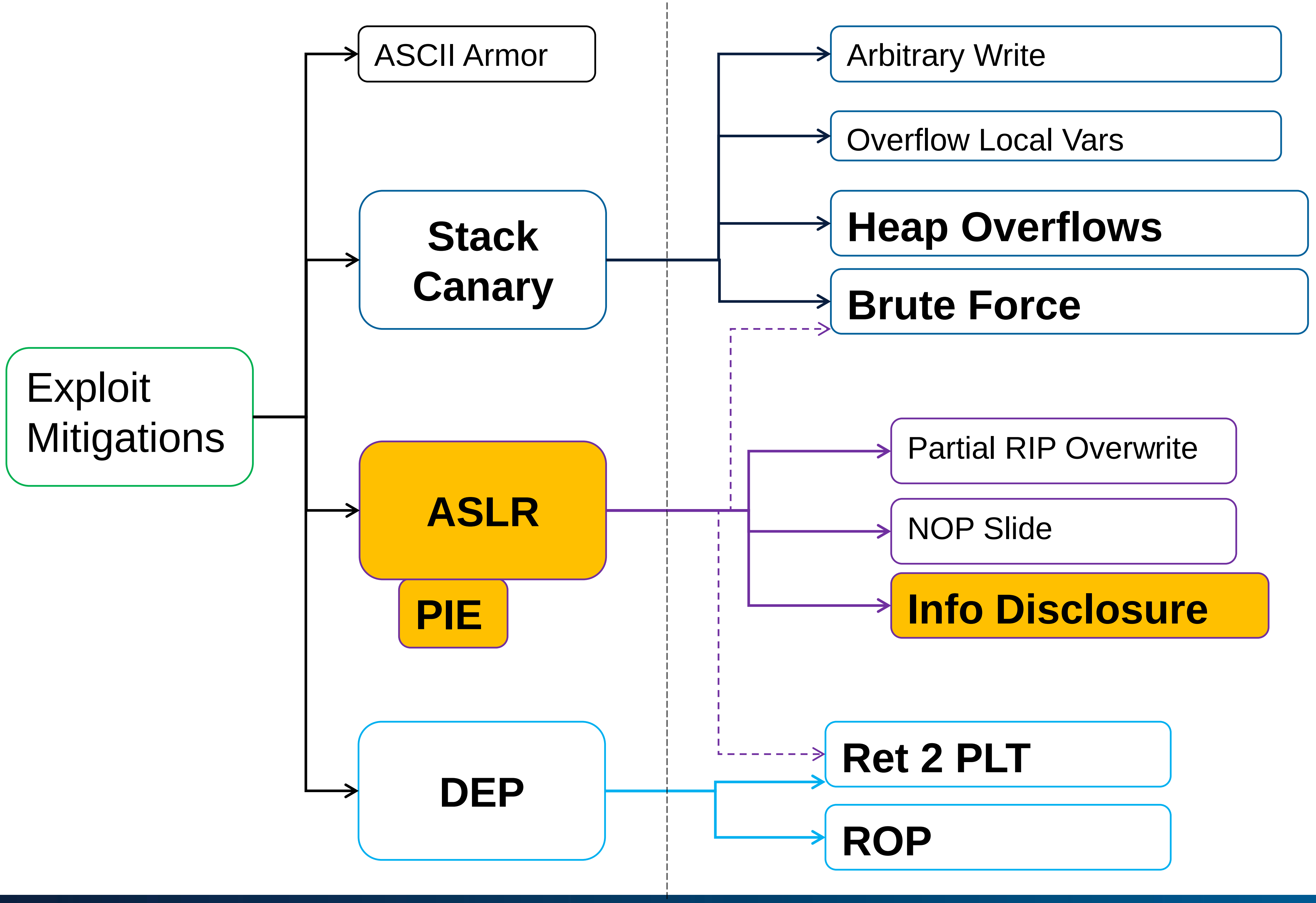
Exploiting: ASLR for code: PIE

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

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

Server:

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

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

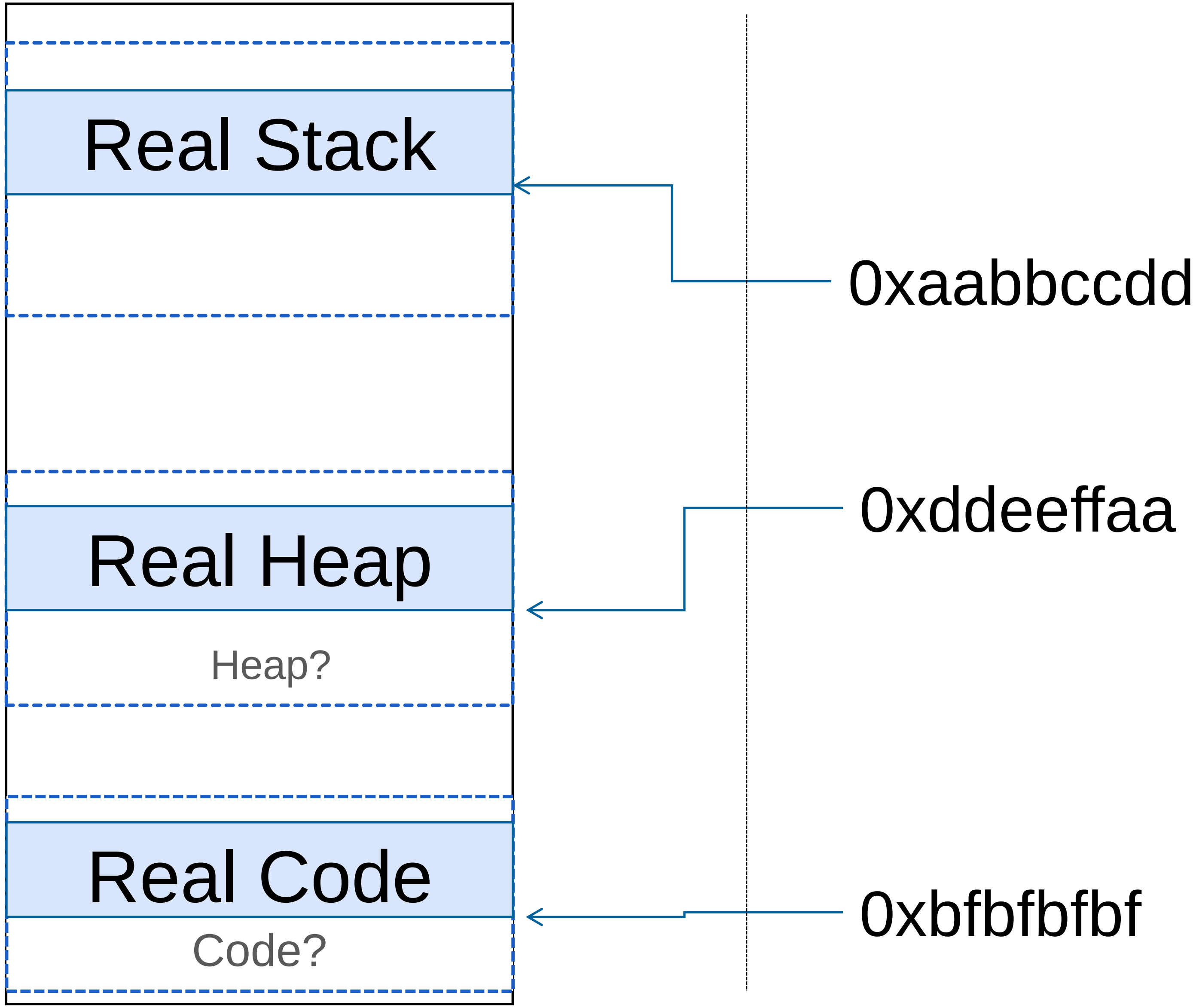
ASLR vs Memory Leak

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

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

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

‹ Prev 2 of 4 Next ›

Back to list

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:

Linux Ubuntu Hardening

Source package	8.04 LTS	9.04	9.10	10.04 LTS	10.10	11.04	11.10
openssh (native)	yes	yes	yes	yes	yes	yes	yes
apache2	--	yes	yes	yes	yes	yes	yes
bind9	--	yes	yes	yes	yes	yes	yes
openldap	--	yes	yes	yes	yes	yes	yes
postfix	--	yes	yes	yes	yes	yes	yes
cups	--	yes	yes	yes	yes	yes	yes
postgresql-8.3	--	yes	yes	yes	yes	yes	yes
samba (native)	--	yes	yes	yes	yes	yes	yes
dovecot	--	yes	yes	yes	yes	yes	yes
dhcp3	--	yes	yes	yes	yes	yes	yes
ntp	--	--	yes	yes	yes	yes	yes
amavisd-new	--	--	yes	yes	yes	yes	yes
squid	--	--	yes	yes	yes	yes	yes
cyrus-sasl2	--	--	yes	yes	yes	yes	yes
exim4	--	--	yes	yes	yes	yes	yes
nagios3	--	--	yes	yes	yes	yes	yes
nagios-plugins	--	--	yes	yes	yes	yes	yes
xinetd	--	--	yes	yes	yes	yes	yes
ipsec-tools	--	--	yes	yes	yes	yes	yes
mysql-dfsg-5.1	--	--	yes	yes	yes	yes	yes
evince	--	--	--	yes	yes	yes	yes
firefox	--	--	--	yes	yes	yes	yes
gnome-control-center	--	--	--	--	--	yes	yes
tiff						yes	yes

init	1235	Full RELRO	Canary found	NX enabled	PIE enabled
dbus-launch	1436	Partial RELRO	Canary found	NX enabled	No PIE
dbus-daemon	1453	Partial RELRO	Canary found	NX enabled	No PIE
dbus-daemon	1454	Partial RELRO	Canary found	NX enabled	No PIE
upstart-event-b	1465	Full RELRO	No canary found	NX enabled	PIE enabled
window-stack-br	1471	Partial RELRO	No canary found	NX enabled	No PIE
upstart-dbus-br	1486	Full RELRO	No canary found	NX enabled	PIE enabled
upstart-dbus-br	1488	Full RELRO	No canary found	NX enabled	PIE enabled
upstart-file-br	1497	Full RELRO	Canary found	NX enabled	PIE enabled
ibus-daemon	1503	Partial RELRO	Canary found	NX enabled	No PIE
unity-settings-	1517	Partial RELRO	No canary found	NX enabled	No PIE
bamfdaemon	1519	Partial RELRO	Canary found	NX enabled	No PIE
at-spi-bus-laun	1523	Full RELRO	Canary found	NX enabled	PIE enabled
gnome-session	1524	Partial RELRO	Canary found	NX enabled	No PIE
dbus-daemon	1529	Partial RELRO	Canary found	NX enabled	No PIE
gvfsd	1533	Partial RELRO	No canary found	NX enabled	No PIE
ibus-dconf	1538	Partial RELRO	No canary found	NX enabled	No PIE
ibus-ui-gtk3	1539	Partial RELRO	No canary found	NX enabled	No PIE
ibus-x11	1542	Partial RELRO	Canary found	NX enabled	No PIE
gvfsd-fuse	1545	Partial RELRO	No canary found	NX enabled	No PIE
at-spi2-registr	1555	Full RELRO	Canary found	NX enabled	PIE enabled
pulseaudio	1645	Full RELRO	Canary found	NX enabled	No PIE
ibus-engine-sim	1692	Partial RELRO	No canary found	NX enabled	No PIE
metacity	1775	Partial RELRO	Canary found	NX enabled	No PIE
dconf-service	1781	Partial RELRO	Canary found	NX enabled	No PIE
gnome-panel	1819	Partial RELRO	Canary found	NX enabled	No PIE
indicator-appli	1835	Partial RELRO	No canary found	NX enabled	No PIE
unity-fallback-	1836	Partial RELRO	No canary found	NX enabled	No PIE
indicator-bluet	1837	Partial RELRO	No canary found	NX enabled	No PIE
vmtoolsd	1839	Partial RELRO	Canary found	NX enabled	No PIE
polkit-gnome-au	1841	Partial RELRO	No canary found	NX enabled	No PIE
nautilus	1848	Partial RELRO	Canary found	NX enabled	No PIE
nm-applet	1852	Partial RELRO	Canary found	NX enabled	No PIE
initctl	1853	Full RELRO	No canary found	NX enabled	PIE enabled
indicator-messa	1858	Partial RELRO	No canary found	NX enabled	No PIE
indicator-power	1863	Partial RELRO	No canary found	NX enabled	No PIE

Ubuntu 16.10: PIE everywhere ?!

Built as PIE

All programs built as Position Independent Executables (PIE) with "-fPIE -pie" can take advantage of the exec ASLR. This protects against "return-to-text" and generally frustrates memory corruption attacks. This requires centralized changes to the compiler options when building the entire archive. PIE has a large (5-10%) performance penalty on architectures with small numbers of general registers (e.g. x86), so it should only be used for a select number of security-critical packages (some upstreams natively support building with PIE, other require the use of "hardening-wrapper" to force on the correct compiler and linker flags). PIE on 64-bit architectures do not have the same penalties, and will eventually be made the default (as of 16.10, it is the default on amd64, ppc64el and s390x).

PIE in Ubuntu

Security Improvements

In Ubuntu 18.04 LTS, gcc is now set to default to compile applications as position independent executables (PIE) as well as with immediate binding, to make more effective use of Address Space Layout Randomization (ASLR). All packages in main have been rebuilt to take advantage of this, with a few exceptions.

```
* Core-Dumps access to all users: Not Restricted
```

COMMAND	PID	RELRO	STACK CANARY	Clang CFI	SafeStack	SECCOMP	NX/PaX	PIE	FORTIFY
systemd	1	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
sshd	125958	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
bash	125999	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
tmux: client	126020	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
login	1299	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
rsyslogd	129948	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
systemd-network	130214	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
systemd-resolve	130220	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
systemd-journal	130225	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
sshd	131778	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
sftp-server	131815	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
systemd	1339	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
(sd-pam)	1340	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
bash	1350	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
tmux: server	1446	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
bash	1447	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
accounts-daemon	149	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
systemd-logind	150	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
cron	153	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
networkd-dispat	159	Partial RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	No PIE	Yes
dbus-daemon	163	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
agetty	179	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
sshd	187	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
master	583	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
qmgr	591	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
pickup	94362	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes
bash	94483	Full RELRO	Canary found	No Clang CFI found	No SafeStack found	Seccomp-bpf	NX enabled	PIE enabled	Yes

```
root@ubuntu-1804:~/cfi/checksec.sh#
```

What is the fundamental difference between attack and defense?

You know when an attack does not work...


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