Bypassing ASLR on 64 bit PIE Linux

- Héctor Marco
- Ismael Ripoll

Bypass ASLR on 64 bit PIE Linux

- Server protections
- ASLR Weakness
- Offset2mem on a buffer overflow:
  - Defeating PIE
  - Stack Buffer Overflow
  - Guessing offsets
  - Building the ROP
- Offset2mem on web browsers
- Demo on Ubuntu 64 bits 12.10

Bypass ASLR on 64 bit PIE Linux

- Application Protections:

Full ASLR	randomize_va_space = 2
SSP enabled	-fstack-protector-all
NX enabled	PAE or x64
App. PIE	-fpie -pie
Full RELRO	-wl, -z, -relro, -z, now
64 bit compiled	-m64

ASLR Weakness

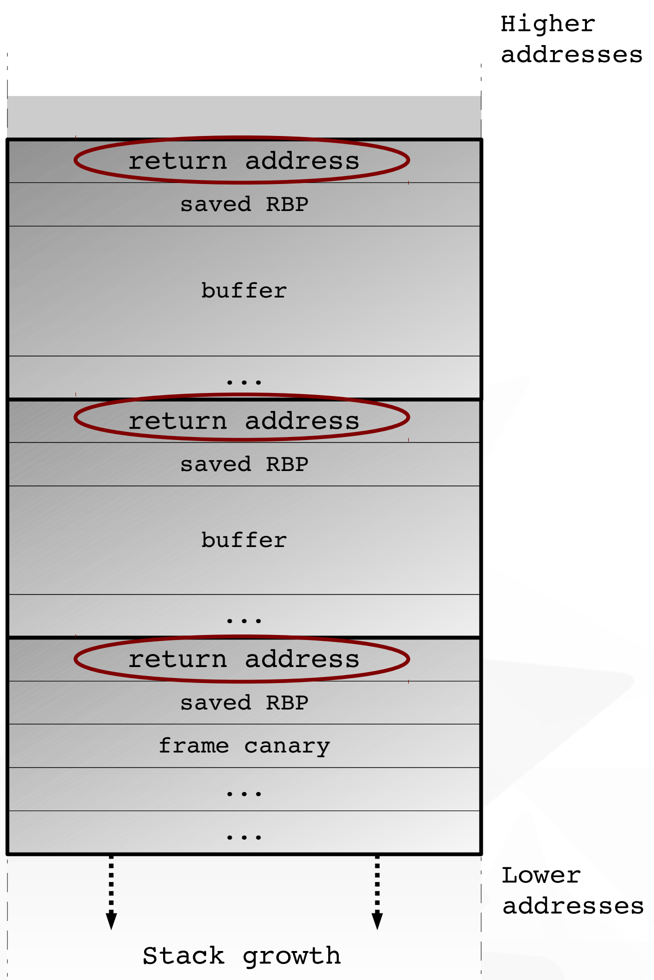
```
box@server:~$ cat /proc/`pidof server_64_PIE_SSP | cut -d" " -f1`/maps
7f36c6a07000-7f36c6bbc000 r-xp 00000000 08:01 1068155                    /lib/x86_64-linux-gnu/libc-2.15.so
7f36c6bbc000-7f36c6dbb000 ---p 001b5000 08:01 1068155                    /lib/x86_64-linux-gnu/libc-2.15.so
7f36c6dbb000-7f36c6dbf000 r--p 001b4000 08:01 1068155                    /lib/x86_64-linux-gnu/libc-2.15.so
7f36c6dbf000-7f36c6dc1000 rw-p 001b8000 08:01 1068155                    /lib/x86_64-linux-gnu/libc-2.15.so
7f36c6dc1000-7f3[illegible] 00000000 00:00 0
7f36c6dc6000-7f3[illegible] 00000000 08:01 1064012                    /lib/x86_64-linux-gnu/ld-2.15.so
7f36c6fd0000-7f3[illegible] 00000000 00:00 0
7f36c6fe5000-7f36c6fe8000 rw-p 00000000 00:00 0
7f36c6fe8000-7f36c6fe9000 r--p 00022000 08:01 1064012                    /lib/x86_64-linux-gnu/ld-2.15.so
7f36c6fe9000-7f36c6feb000 rw-p 00023000 08:01 1064012                    /lib/x86_64-linux-gnu/ld-2.15.so
7f36c6feb000-7f36c6fed000 r-xp 00000000 08:01 939105                     /home/box/server_64_PIE_SSP
7f36c71ec000-7f36c71ed000 r--p 00001000 08:01 939105                     /home/box/server_64_PIE_SSP
7f36c71ed000-7f36c71ee000 rw-p 00002000 08:01 939105                     /home/box/server_64_PIE_SSP
7fffe4018000-7fffe4039000 rw-p 00000000 00:00 0                          [stack]
7fffe41b7000-7fffe41b8000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

Constant !!!

- Linux ASLR randomizes only the **first** mapped area (i.e. library).
  - Subsequent maps are put side by side.
- A PIE executable is mapped as a normal a shared library.
- As a result the mapping distance between the application and any other memory region is always the same. We will call this “**Offset2mem**” technique.
- Once we know one single address, we can calculate any other → ASLR is defeated.

Defeating PIE

- Stuff on the stack:
  - When a function is called, the **instruction pointer** is pushed onto the stack to allow the program to return to the site of the call later.
- Depending on the bug, it is possible do bruteforce to the return address.



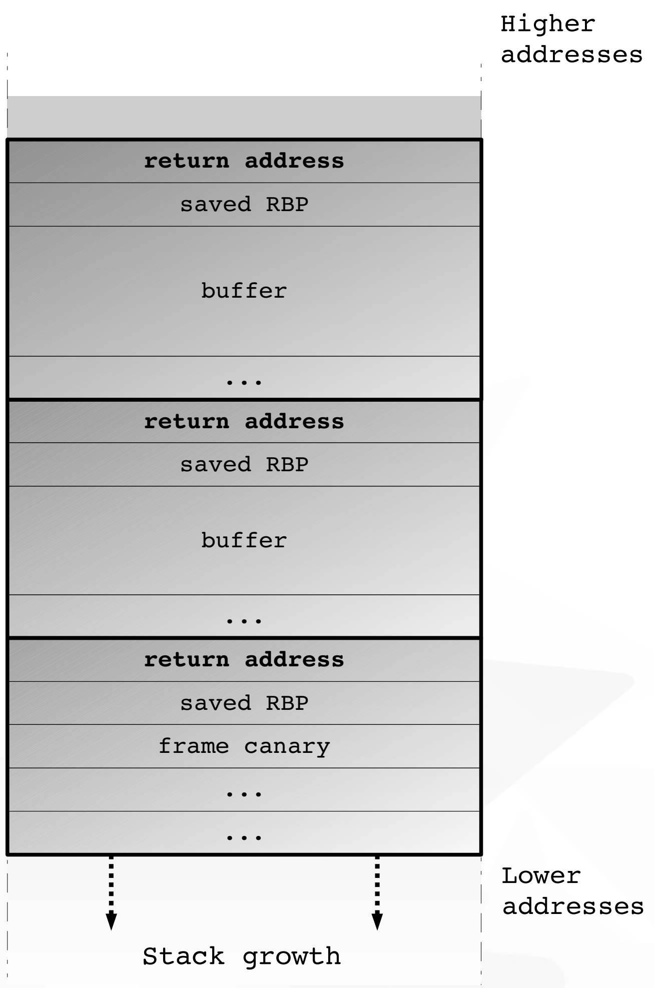
Defeating PIE

- Stack buffer overflow

```
/*
 * This is a dummy function which contain a simple buffer overflow
 */
void vulnerable_function(char *srcbuff, int lsrcbuff, int sock_c){
   char buff[48];

   memcpy(buff, srcbuff, lsrcbuff);

}
```

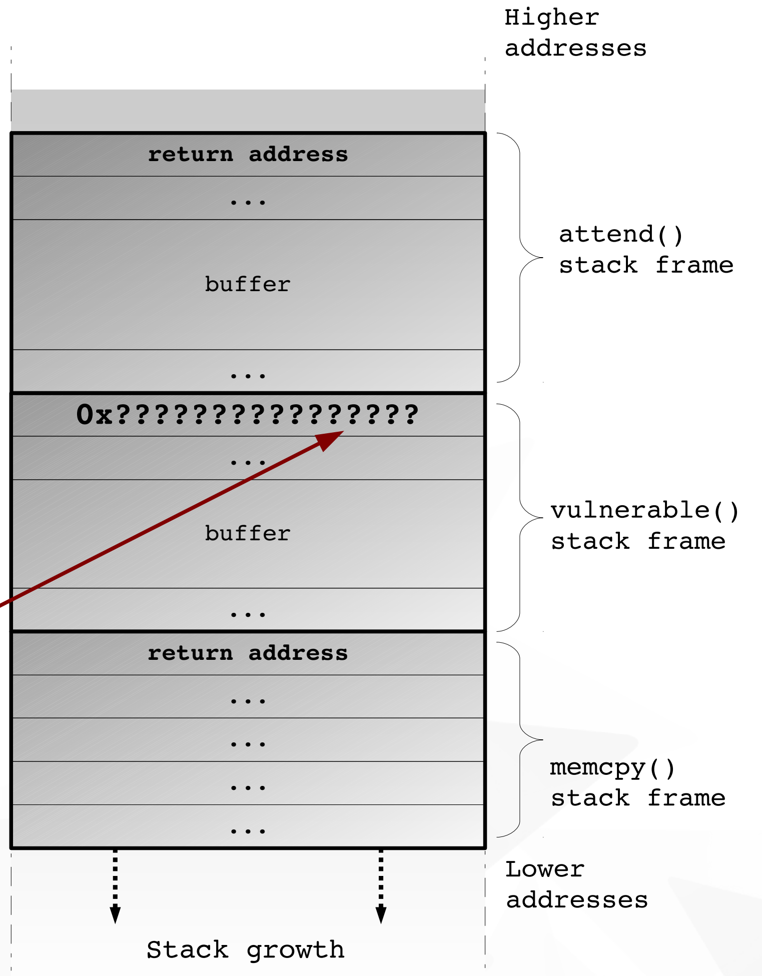


- This kind of bugs allow to brute force the return address
- Knowing the return address PIE is defeated.

Defeating PIE

```
0000000000000efb <vulnerable_function>:
    efb:        55                          push   %rbp
    efc:        48 89 e5                    mov    %rsp,%rbp
   .....        .....                       .....
   .....        .....                       .....
    f30:        e8 0b fd ff ff              callq  c40 <memcpy@plt>
    f35:        48 8b 45 f8                 mov    -0x8(%rbp),%rax
    f39:        64 48 33 04 25 28 00        xor    %fs:0x28,%rax
    f40:        00 00
    f42:        74 05                       je     f49 <vulnerable_function+0x4e>
    f44:        e8 77 fc ff ff              callq  bc0 <__stack_chk_fail@plt>
    f49:        c9                          leaveq
    f4a:        c3                          retq


0000000000001063 <attend_non_return>:
   1063:        55                          push   %rbp
   1064:        48 89 e5                    mov    %rsp,%rbp
   1067:        48 81 ec 60 04 00 00        sub    $0x460,%rsp
   106e:        64 48 8b 04 25 28 00        mov    %fs:0x28,%rax
   1075:        00 00
   1077:        48 89 45 f8                 mov    %rax,-0x8(%rbp)
   107b:        31 c0                       xor    %eax,%eax
   .....        .....                       .....
   .....        .....                       .....
   12d5:        89 ce                       mov    %ecx,%esi
   12d7:        48 89 c7                    mov    %rax,%rdi
   12da:        e8 1c fc ff ff              callq  efb <vulnerable_function>
   12df:        48 8d 85 c0 fb ff ff        lea    -0x440(%rbp),%rax
   12e6:        48 89 c7                    mov    %rax,%rdi
   .....        .....                       .....
   .....        .....                       .....
```



Defeating PIE

Hardcoded Unknown

Saved RIP **0x00007F?????????2DF**

```
12d7:     48 89 c7                  mov    %rax,%rdi
12da:     e8 1c fc ff ff            callq  efb <vulnerable_function>
12df:     48 8d 85 c0 fb ff ff      lea    -0x440(%rbp),%rax
12e6:     48 89 c7                  mov    %rax,%rdi
```

- Page number where the call is made (page_no_of_the_call).
- We will use this number to obtain the base address of the application `.text`

Defeating PIE

- Bruteforcing unknown bytes (byte for byte strategy)
  - Only to 3 and a half bytes.
  - Max trials: 256*3 + 128 = 896
  - Very quick: less than 1 second.

`0x00007F????????2DF`

- Non-optimized code
  - Bruteforcing RBP
    - Help to derandomize the stack.
    - Most applications optimize code (no RBP).
    - It is not as good as saved RIP reference.
      - Since RIP is always available, it is better to bruteforce against the saved RIP.

Guessing offsets

- The offset from executable to libraries depends the Linux distribution (the size of each library and local mappings):
  - But it is always constant on the same system.

- Some libc offsets:
  - Ubuntu 12.10 = 0x5e4000
  - Ubuntu 12.04 L.T.S = 0x5e4000
  - Debian 7.1 = 0x5ac000

- Using offset2mem we can use as many mapped libraries as we need to create the ROP:
  - We are not limited to use only the libc library.

Building the attack

- Steps to bypass ASLR 64 bits for a PIE compiled application
  1. Extract low bits from the application.
  2. Make a brute force attack against Saved RIP
     - Set high bits (`0x00007Fxxxxxxxxxx`)
     - Set low bits previously extracted (`0x00007Fxxxxxxx2DF`)
     - Obtain the page number where the call is performed.
     - Obtain saved RIP by bruteforce (less than 1 second)
  3. Obtain the base address of the application .text
     - Text_Base = (Saved RIP) & ~0xFFF – (page_no_of_the_call << 12)
  4. Subtract Base to the OFFSET
     - `Text_Base – 0x5e4000` = libc mapping
     - `Text_Base – 0x???000` = other library mapping
  5. Build the ROP
     - Use as many libraries as needed to build the ROP attack.

Other applications of the Offset2mem

- Firefox

```
box@server:~$ cat /proc/`pidof firefox | cut -d" " -f1`/maps | grep rwx
7fa4e7c26000-7fa4e7c36000 rwxp 00000000 00:00 0
7fa4ed402000-7fa4ed452000 rwxp 00000000 00:00 0
7fa4fb703000-7fa4fb704000 rwxp 00000000 00:00 0
```

- Chrome

```
box@server:~$ cat /proc/`pidof /usr/lib/chromium-browser/chro | cut -d" " -f1`/maps | grep rwx
ca169406000-ca169407000 rwxp 00000000 00:00 0
ca169506000-ca169507000 rwxp 00000000 00:00 0
ca169606000-ca1696ff000 rwxp 00000000 00:00 0
ca16a206000-ca16a2ff000 rwxp 00000000 00:00 0
```

- We know the distance between the sprayed area and the application.
  - Add reliability to current exploitation techniques

- It can open the door to new exploitation techniques.

Conclusions

- For this kind of vulnerability (byte-for-byte overflow) the PIE is worse than NON-PIE !!
- Since all the areas are mapped side by side on a PIE application, it is possible to calculate the address of the libraries.
- Offset2mem technique is not limited to bruteforce Saved EIP. This technique only require know an application value (.text, stack, heap …) to obtain the full mapping of all libraries.