On the effectiveness of Full-ASLR on 64-bit Linux

Hector Marco-Gisbert, Ismael Ripoll
Universitàt Politècnica de València (Spain)

In-Depth Security Conference (DeepSec)

November 18-21, 2014

Table of contents

What have we done ?

We have deeply analyzed the effectiveness of the GNU/Linux ASLR and:

- Found a **weakness** on the current **GNU/Linux ASLR** implementation, named **offset2lib**.
- Built an attack which bypasses the NX, SSP and ASLR on a 64 bit system in < 1 sec.
- **Sent a small patch "ASLRv3"** (randomize_va_space = 3) to Linux developers, but **no response**.
- Some mitigation techniques against the offset2lib attack are presented.

ASLR Background

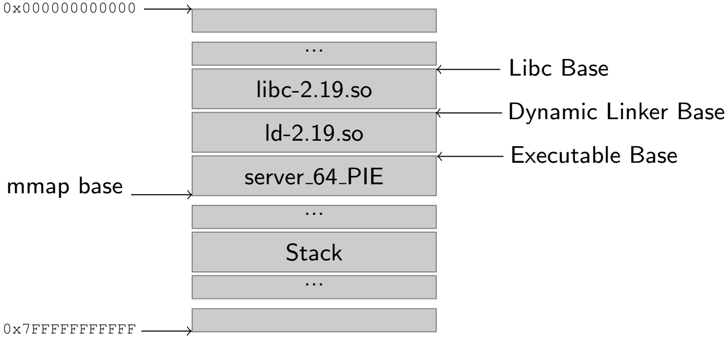
- ASLR does not remove vulnerabilities but make more difficult to exploit them.
- ASLR deters exploits which relays on knowing the memory map.
- ASLR is effective when all memory areas are randomise. Otherwise, the attacker can use these non-random areas.
- Full ASLR is achieved when:
  - Applications are compiled with PIE (-fpie -pie).
  - The kernel is configured with randomize_va_space = 2

    (stack, VDSO, shared memory, data segment)

Loading shared objects

The problem appears when the application is compiled with PIE because the GNU/Linux algorithm for loading shared objects works as follows:

- The **first** shared object is loaded at a **random position**.
- The next object is located right below (lower addresses) the last object.



All libraries are located "side by side" at a **single random place**.

Offset2lib

$ cat /proc/<pid>/server_64_PIE

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

Offset2lib

$ cat /proc/<pid>/server_64_PIE

0x5eb000

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

Offset2lib

$ cat /proc/<pid>/server_64_PIE

0x5eb000

0x225000

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

Offset2lib

offset2lib

offset2lib

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

We named this invariant distance **offset2lib** which:

- It is a **constant distance** between two shared objects even in different executions of the application.

Offset2lib

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

offset2lib

offset2lib

We named this invariant distance **offset2lib** which:

- It is a **constant distance** between two shared objects even in different executions of the application.

Any address of the app. → de-randomize all mmapped areas !!!

Why the Offset2lib is dangerous ?

Offset2lib scope:

- **Realistic**; applications are more prone than libraries to errors.
- Makes some vulnerabilities **faster**, **easier** and **more reliable** to exploit them.
- It is not a self-exploitable vulnerability but an ASLR-design weakness exploitable.
- It opens new (and old) attack vectors.

Why the Offset2lib is dangerous ?

Offset2lib scope:

- **Realistic**; applications are more prone than libraries to errors.
- Makes some vulnerabilities **faster**, **easier** and **more reliable** to exploit them.
- It is not a self-exploitable vulnerability but an ASLR-design weakness exploitable.
- It opens new (and old) attack vectors.

Next example:
Offset2lib on a standard stack buffer overflow.

Building the attack

The steps to build the attack are:

1. Extracting static information
2. Brute force part of saved-IP
3. Calculate base app. address
4. Calculate library offsets
5. Obtain mmapped areas

1) Extracting static information

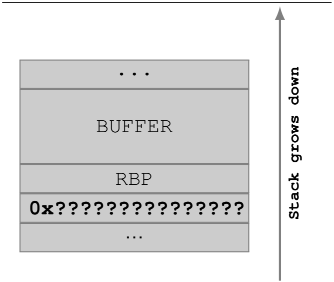
Our goal is to obtain an address belonging to the application.

We are going to obtain the **saved-IP** of vulnerable function caller.

Offset2lib with **saved-IP** $\Rightarrow$ **all** mmapped areas.

```
0000000000001063 <attend_client>:
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
..... .....          .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea  -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....          .....
```

STACK



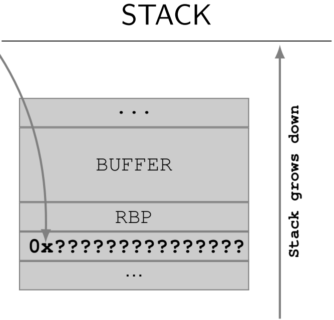
1) Extracting static information

Our goal is to obtain an address belonging to the application.

We are going to obtain the **saved-IP** of vulnerable function caller.

Offset2lib with **saved-IP** ⇒ **all** mmapped areas.

```
0000000000001063 <attend_client>:
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
..... .....          .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....          .....
```



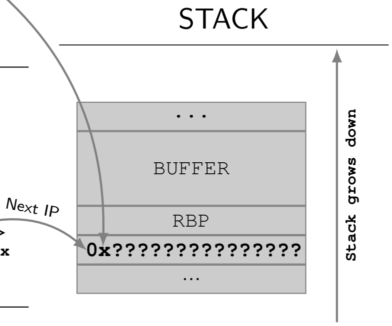
1) Extracting static information

Our goal is to obtain an address belonging to the application.

We are going to obtain the **saved-IP** of vulnerable function caller.

Offset2lib with **saved-IP** ⇒ **all** mmapped areas.

```
0000000000001063 <attend_client>:
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
..... .....         .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....         .....
```



1) Extracting static information

Our goal is to obtain an address belonging to the application.

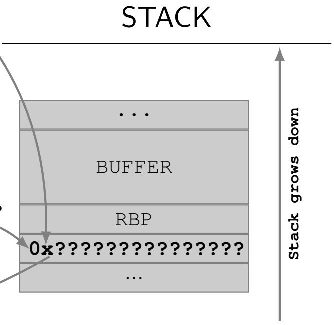
We are going to obtain the **saved-IP** of vulnerable function caller.

Offset2lib with **saved-IP** ⇒ **all** mmapped areas.

```
0000000000001063 <attend_client>:
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
..... .....         .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea  -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....         .....
```

Next IP

Address point to

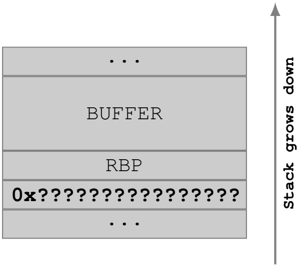


1) Extracting static information

Memory map

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

STACK



This value (`0x00007F`) can be obtained:

1. Running the application and showing the memory map.
2. Checking the source code if set any limit to stack.

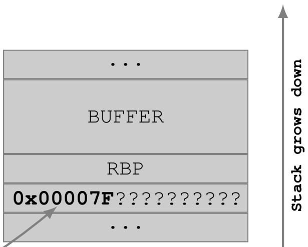
1) Extracting static information

Memory map

```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

Highest 24 bits

STACK



This value (0x00007F) can be obtained:

1. Running the application and showing the memory map.
2. Checking the source code if set any limit to stack.

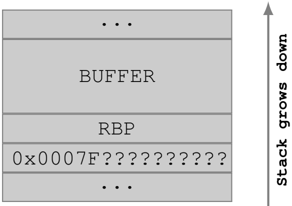
1) Extracting static information

Since the executable has to be `PAGE_SIZE` aligned, the 12 lower bits will not change when the executable is randomly loaded.

ASM Code

```
0000000000001063 <attend_client>:
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
..... .....  .....
12d7: 48 89 c7              mov  %rax,%rdi
12da: e8 1c fc ff ff        callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff  lea  -0x440(%rbp),%rax
12e6: 48 89 c7              mov  %rax,%rdi
.....  .....  [From the ELF]     .....
```

STACK



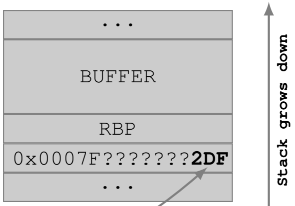
1) Extracting static information

Since the executable has to be `PAGE_SIZE` aligned, the 12 lower bits will not change when the executable is randomly loaded.

ASM Code

```
0000000000001063 <attend_client>:
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
..... .....  .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea  -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... ....  [From the ELF]    .....
```

STACK



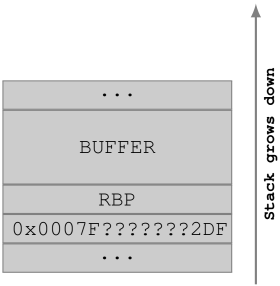
Lower 12 bits

2) Brute forcing Saved-IP address

```
void vuln_func(char *str, int lstr){
  char buff[48];
  int i = 0;
  ...
  for (i = 0; i < lstr; i++) {
    if (str[i] != '\n')
      buff[lbuff++] = str[i];
  ...
}
```

- The unknown 28 random bits: “byte-for-byte” attack.
- The first byte is “special”, we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... 0xC2, 0xD2, 0xE2, 0xF2}

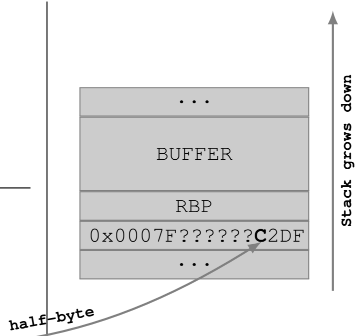
STACK



2) Brute forcing Saved-IP address

```
void vuln_func(char *str, int lstr){
  char buff[48];
  int i = 0;
  ...
  for (i = 0; i < lstr; i++) {
    if (str[i] != '\n')
      buff[lbuff++] = str[i];
  ...
}
```

STACK

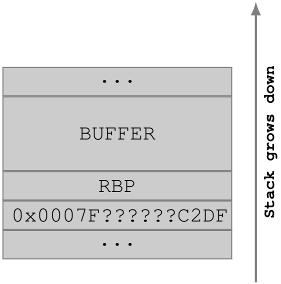


- The unknown 28 random bits: “byte-for-byte” attack.
- The first byte is “special”, we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... **0xC2**, 0xD2, 0xE2, 0xF2}

2) Brute forcing Saved-IP address

```
void vuln_func(char *str, int lstr){
  char buff[48];
  int i = 0;
  ...
  for (i = 0; i < lstr; i++) {
    if (str[i] != '\n')
      buff[lbuff++] = str[i];
  ...
}
```

STACK

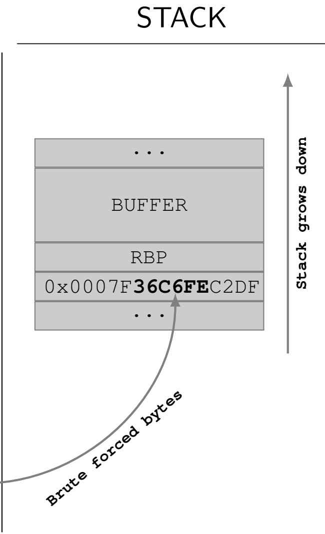


- The unknown 28 random bits: "byte-for-byte" attack.
- The first byte is "special", we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... **0xC2**, 0xD2, 0xE2, 0xF2}
- The remaining 3 bytes → standard "byte-for-byte" attack
  - $3x2^8 = 768$ attempts.
- After execute the byte-for-byte we obtained `0x36C6FE`

2) Brute forcing Saved-IP address

```
void vuln_func(char *str, int lstr){
  char buff[48];
  int i = 0;
  ...
  for (i = 0; i < lstr; i++) {
    if (str[i] != '\n')
      buff[lbuff++] = str[i];
  ...
}
```

- The unknown 28 random bits: "byte-for-byte" attack.
- The first byte is "special", we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... **0xC2**, 0xD2, 0xE2, 0xF2}
- The remaining 3 bytes → standard "byte-for-byte" attack
  - $3x2^8 = 768$ attempts.
- After execute the byte-for-byte we obtained **0x36C6FE**

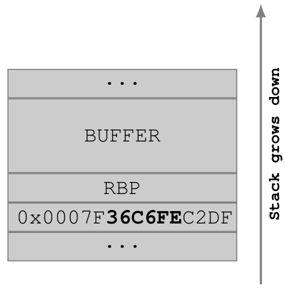


2) Brute forcing Saved-IP address

```
void vuln_func(char *str, int lstr){
  char buff[48];
  int i = 0;
  ...
  for (i = 0; i < lstr; i++) {
    if (str[i] != '\n')
      buff[lbuff++] = str[i];
  ...
}
```

- The unknown 28 random bits: “byte-for-byte” attack.
- The first byte is “special”, we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - `{0x02, 0x12, 0x22 ... 0xC2, 0xD2, 0xE2, 0xF2}`
- The remaining 3 bytes $\rightarrow$ standard “byte-for-byte” attack
  - $3x2^8 = 768$ attempts.
- After execute the byte-for-byte we obtained **0x36C6FE**
- We need to perform $\frac{2^4+3*2^8}{2} = 392$ attempts on average.

STACK

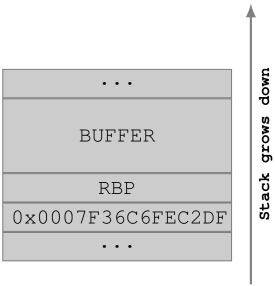


3) Calculating base application address

```
0000000000001063 <attend_client>:
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
..... .....   .....
12d7: 48 89 c7              mov  %rax,%rdi
12da: e8 1c fc ff ff        callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff  lea  -0x440(%rbp),%rax
12e6: 48 89 c7              mov  %rax,%rdi
..... .....        .....
```

App_base=(savedIP & 0xFFF)-(CALLER_PAGE_OFFSET << 12)

STACK

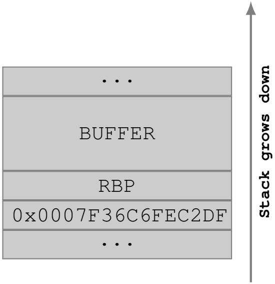


3) Calculating base application address

```
0000000000001063 <attend_client>:
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
..... .....  .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea  -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....         .....
```

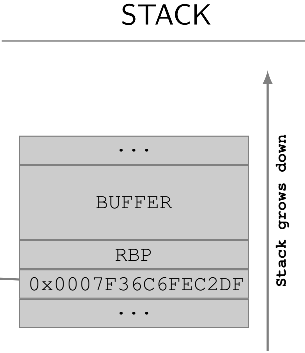
```
App_base=(savedIP & 0xFFF)-(CALLER_PAGE_OFFSET << 12)
```

STACK



3) Calculating base application address

```
0000000000001063 <attend_client>:
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
..... .....   .....
12d7: 48 89 c7              mov  %rax,%rdi
12da: e8 1c fc ff ff        callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff  lea  -0x440(%rbp),%rax
12e6: 48 89 c7              mov  %rax,%rdi
..... .....        .....
```

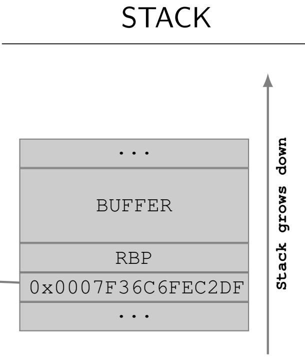


App_base=(savedIP & 0xFFF)−(CALLER_PAGE_OFFSET << 12)

0x7F36C6fEB000=(0x7f36C6FEC2DF & 0xFFF)−(0x1000)

3) Calculating base application address

```
0000000000001063 <attend_client>:
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
..... .....  .....
12d7: 48 89 c7                mov  %rax,%rdi
12da: e8 1c fc ff ff          callq efb <vuln_func>
12df: 48 8d 85 c0 fb ff ff    lea  -0x440(%rbp),%rax
12e6: 48 89 c7                mov  %rax,%rdi
..... .....        .....
```



App_base=(savedIP & 0xFFF)−(CALLER_PAGE_OFFSET << 12)

0x7F36C6fEB000=(0x7f36C6FEC2DF & 0xFFF)−(0x1000)

App. Base = 0x7F36C6fEB000

4) Calculating library offsets

offset2lib

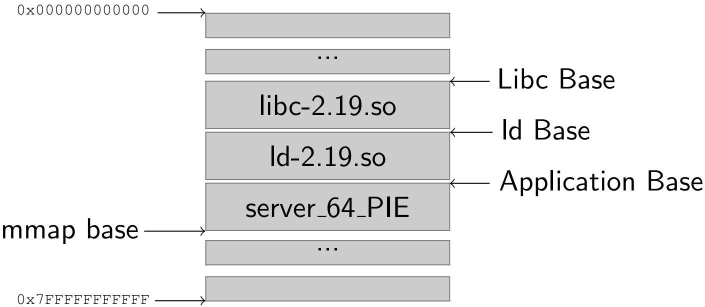
```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 rw-p
7fd1b4734000-7fd1b4737000 rw-p
7fd1b4737000-7fd1b4738000 r--p /lib/.../ld-2.19.so
7fd1b4738000-7fd1b4739000 rw-p /lib/.../ld-2.19.so
7fd1b4739000-7fd1b473a000 rw-p
7fd1b473a000-7fd1b473c000 r-xp /root/server_64_PIE
7fd1b493b000-7fd1b493c000 r--p /root/server_64_PIE
7fd1b493c000-7fd1b493d000 rw-p /root/server_64_PIE
7fff981fa000-7fff9821b000 rw-p [stack]
7fff983fe000-7fff98400000 r-xp [vdso]
```

Distribution	Libc version	Offset2lib (bytes)
CentOS 6.5	2.12	0x5b6000
Debian 7.1	2.13	0x5ac000
Ubuntu 12.04 LTS	2.15	0x5e4000
Ubuntu 12.10	2.15	0x5e4000
Ubuntu 13.10	2.17	0x5ed000
openSUSE 13.1	2.18	0x5d1000
Ubuntu 14.04.1 LTS	2.19	0x5eb000

5) Getting app. process mapping

Obtaining library base addresses:

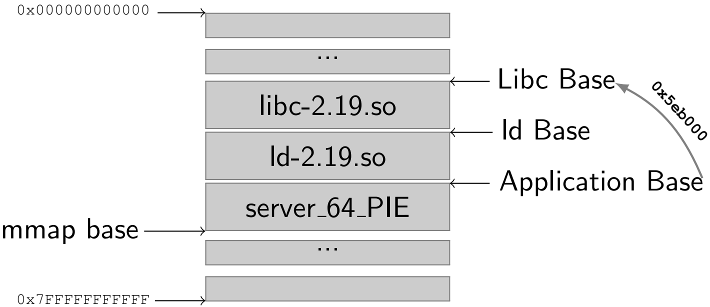
- Application Base = 0x7FD1B473A000
- Offset2lib (libc) = 0x5eb000
- Offset2lib (ld) = 0x225000



5) Getting app. process mapping

Obtaining library base addresses:

- Application Base = 0x7FD1B473A000
- Offset2lib (libc) = 0x5eb000
- Offset2lib (ld) = 0x225000

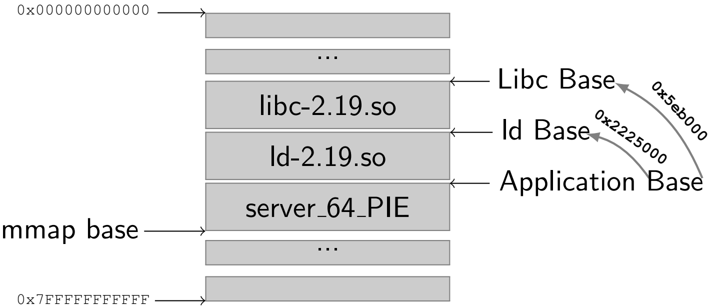


Libc Base = 0x7FD1B473A000 - 0x5eb000 = **0x7FD1B414F000**

5) Getting app. process mapping

Obtaining library base addresses:

- Application Base = 0x7FD1B473A000
- Offset2lib (libc) = 0x5eb000
- Offset2lib (ld) = 0x225000



Libc Base = 0x7FD1B473A000 - 0x5eb000 = **0x7FD1B414F000**

ld Base = 0x7FD1B473A000 - 0x225000 = **0x7fd1b4515000**

The vulnerable server

To show a more realistic PoC:

- Bypass NX, SSP, ASLR, FORTIFY or RELRO.
- We do not use GOT neither PLT.
- Valid for any application (Gadgets only from libraries)
- We use a fully updated Linux.

Parameter	Comment	Configuration
App. relocatable	Yes	`-fpie -pie`
Lib. relocatable	Yes	`-Fpic`
ASLR config.	Enabled	`randomize_va_space = 2`
SSP	Enabled	`-fstack-protector-all`
Arch.	64 bits	x86_64 GNU/Linux
NX	Enabled	PAE or x64
RELRO	Full	`-wl,-z,-relro,-z,now`
FORTIFY	Yes	`-D_FORTIFY_SOURCE=2`
Optimisation	Yes	`-O2`

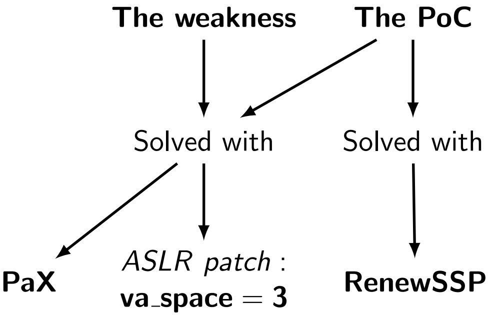
Bypassing NX, SSP and ASLR on 64-bit Linux

Demo: Bypass NX, SSP and ASLR in $<$ 1 sec.

How to prevent exploitation

- There are many vectors to exploit this weakness: Imagination is the limit. Basically, an attacker needs:
    1. The knowledge (information leak).
    2. A way to use it.
- There are many solutions to address this weakness:
    - Avoid information leaks at once:
        - Don't design weak applications/protocols.
        - Don't write code with errors.
        - . . .
    - Make the leaked information useless:
        - **PaX** patch
        - **randomize_va_space=3**
        - **RenewSSP**: Improve stack-smashing-protector.

Solutions overview



What is wrong with the current ASLR design?

At a very abstract level the answer is:

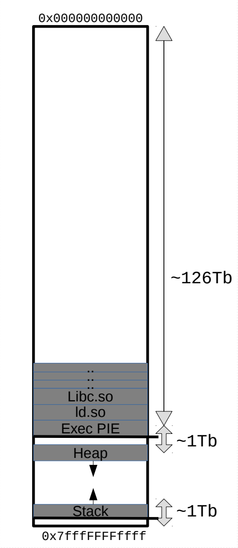
It does not honour MILS concepts

- MILS: "*Multiple Independent Levels of Security/Safety*".
- The whole system is as weak as the weakest part.
- Library vs. application code:
  - Library code is written by more experienced programmers.
  - Library code is intensively and extensively tested: many users used/abused it in many ways.
- Application code is more prone to programming bugs than libraries.

PaX Patch

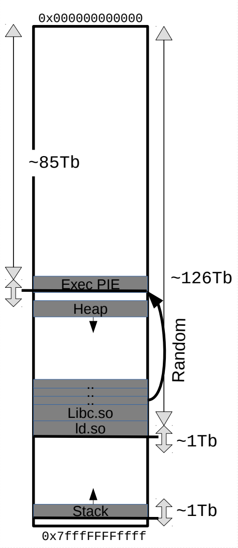
- PaX defines three areas:
  - delta_exec: code, data, bss, brk.
  - delta_mmap: libraries, mapped files, thread stack, shared memory, ...
  - delta_stack: user stack.
- **PaX ASLR does not have this weakness.**
- PaX is very robust and complete.
- It is able to randomise even non-PIE applications.
- Unfortunately, some people think that it is a complex patch with, may be, too many features.
- It is not in the Linux mainstream.

randomize_va_space=3



A very simple idea to workaround this weakness:
Place the executable and the libraries at different addresses

randomize_va_space=3



A very simple idea to workaround this weakness:

Place the executable and the libraries at different addresses

- If there is no relation between application executable and library addresses, then it is useless
- An executable memory leak can not be used to build a library ROPs
- It has been implemented as a small Linux kernel patch

This is basically the same solution that the one used by PaX, but smaller.

With randomize_va_space=2

```
# echo 2 > /proc/sys/kernel/randomize_va_space
# hello_world_dynamic_pie
7f621ffbb000-7f6220176000 r-xp 00000000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f6220176000-7f6220376000 ---p 001bb000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f6220376000-7f622037a000 r--p 001bb000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f622037a000-7f622037c000 rw-p 001bf000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f622037c000-7f6220381000 rw-p 00000000 00:00 0
7f6220381000-7f62203a4000 r-xp 00000000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f622059c000-7f622059d000 rw-p 00000000 00:00 0
7f622059d000-7f622059e000 r-xp 00000000 00:00 0
7f622059e000-7f62205a3000 rw-p 00000000 00:00 0
7f62205a3000-7f62205a4000 r--p 00022000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f62205a4000-7f62205a5000 rw-p 00023000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f62205a5000-7f62205a6000 rw-p 00000000 00:00 0
7f62205a6000-7f62205a7000 r-xp 00000000 00:02 4896 /bin/hello_world_dynamic_pie
7f62207a6000-7f62207a7000 r--p 00000000 00:02 4896 /bin/hello_world_dynamic_pie
7f62207a7000-7f62207a8000 rw-p 00001000 00:02 4896 /bin/hello_world_dynamic_pie
7fff47e15000-7fff47e36000 rw-p 00000000 00:00 0 [stack]
7fff47e63000-7fff47e65000 r--p 00000000 00:00 0 [vvar]
7fff47e65000-7fff47e67000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

With randomize_va_space=3

```
# echo 3 > /proc/sys/kernel/randomize_va_space
# hello_world_dynamic_pie
54859ccd6000-54859ccd7000 r-xp 00000000 00:02 4896 /bin/hello_world_dynamic_pie
54859ced6000-54859ced7000 r--p 00000000 00:02 4896 /bin/hello_world_dynamic_pie
54859ced7000-54859ced8000 rw-p 00001000 00:02 4896 /bin/hello_world_dynamic_pie
7f75be764000-7f75be91f000 r-xp 00000000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f75be91f000-7f75beb1f000 ---p 001bb000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f75beb1f000-7f75beb23000 r--p 001bb000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f75beb23000-7f75beb25000 rw-p 001bf000 00:02 5192 /lib/x86_64-linux-gnu/libc.so.6
7f75beb25000-7f75beb2a000 rw-p 00000000 00:00 0
7f75beb2a000-7f75beb4d000 r-xp 00000000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f75bed45000-7f75bed46000 rw-p 00000000 00:00 0
7f75bed46000-7f75bed47000 r-xp 00000000 00:00 0
7f75bed47000-7f75bed4c000 rw-p 00000000 00:00 0
7f75bed4c000-7f75bed4d000 r--p 00022000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f75bed4d000-7f75bed4e000 rw-p 00023000 00:02 4917 /lib64/ld-linux-x86-64.so.2
7f75bed4e000-7f75bed4f000 rw-p 00000000 00:00 0
7fffb3741000-7fffb3762000 rw-p 00000000 00:00 0 [stack]
7fffb377b000-7fffb377d000 r--p 00000000 00:00 0 [vvar]
7fffb377d000-7fffb377f000 r-xp 00000000 00:00 0 [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0 [vsyscall]
```

Brute forcing the SSP

In this PoC, before bypassing the ASLR, we had to bypass the SSP. A better implementation of the SSP would have blocked the attack.

- SSP is very effective[1] to protect stack smashing.
- The canary (stack guard) is a fairly large random number on 32bits and a ginormous number on 64bits.
- Unfortunately, the byte-for-byte attack yields the SSP almost useless, as shown in the PoC.
- The problem is that all the children (on a forking server) inherit the same canary (the secret).

[1] But not always, see CVE-2014-5439

RenewSSP (I)

⇒ What if every child process has a different canary value?

⇒ It will be **impossible** to make a **brute force attack**

- RenewSSP[2] is an extension of the stack-protector technique which renews the reference-canary at key points during the execution of the application.
- One of such points is when a new process is created (forked).
- We showed (in a previous paper) that it is possible to renew the canary, set a new random value, on a child process and it can continue normally.

[2] http://renewssp.com

RenewSSP (II)

- On current systems with SSP (all existing ones):
  - ⇒ brute force is possible because the attacker can discard the guessed values until the correct one is found.
  - ⇒ It is a *"sampling without replacement"* statistical process.
  - ⇒ Known also as: brute-force.
- With RenewSSP:
  - ⇒ a guessed value can not be discarded, because it may appear again.
  - ⇒ It is a *"sampling with replacement"* statistical process.
  - ⇒ Known also as: trial-and-test.
- Regarding byte-for-byte: RenewSSP disables the possibility to split the attack into single bytes. Which renders the attack as a trial-and-test to the whole canary word.
- With RenewSSP, the PoC showed on this presentation is prevented.

Conclusions

- Using offset2lib, we have shown the fastest way to bypass the ASLR on Linux 64 bits by exploiting a stack buffer overflow.
- Incrementing the ASLR entropy bits does not thwart our attack.
- We consider that PIE linked application prone to byte-for-byte attacks are not secure.
- We have proposed to Linux kernel developers the ASLRv3 which removes the weakness by randomising the distance between the libraries and the executable.
- As far as we know, the RenewSSP is the only technique which prevents the attack vector used in the PoC.

Questions ?

* Hector Marco-Gisbert http://hmarco.org
* Ismael Ripoll Ripoll http://personales.upv.es/iripoll
* Cyber-security research group at http://cybersecurity.upv.es

In collaboration with Packet Storm Security.