Exploiting Linux and PaX ASLR's weaknesses on 32- and 64-bit systems

Hector Marco-Gisbert, Ismael Ripoll
Universitat Politècnica de València (Spain)
http://cybersecurity.upv.es

Black Hat Asia

March 29 - April 1, 2016

Table of contents

What have we done ?

We have deeply analyzed the ASLR of Linux and PaX and:

- Found some **weaknesses** and **limitations** on the current implementations:
  1. Too low entropy
  2. Non-uniform distribution
  3. Correlation between objects
  4. Inheritance
- Built attacks which exploit these weaknesses:
  - **Offset2lib**: bypasses the NX, SSP and ASLR in in < 1 sec.
  - Also, other attack vectors (exploiting other weaknesses)
- We have contributed to Linux kernel by:
  - Fixing the **Offset2lib** weakness.
  - Sketches a working in progress version of the ASLR $\rightarrow$ ASLR-NG.
  - Also some mitigation techniques will be presented *(RenewSSP)*
- We present ASLRA, a suit tool to analyze the entropy of Linux ASLR implementations.

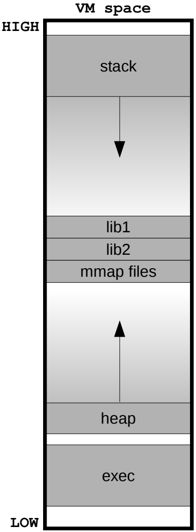
ASLR Background

- 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)

What is ASLR ?

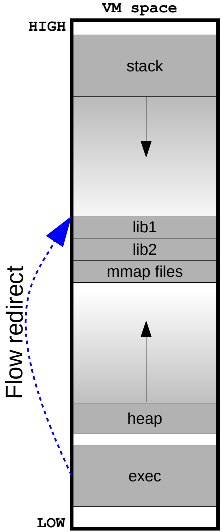
ASLR is a protection provided by the kernel to applications which:

- It loads the stack, executable, libraries and heap at random locations.
- It tries to deter attacks that rely on knowing the location of the target data or code.
- It makes vulnerabilities more difficult to exploit.



How the ASLR works: A simple example

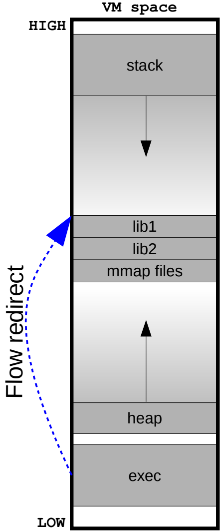
The attacker redirects the execution to the `exec()` library function (`lib1`):



How the ASLR works: A simple example

The attacker redirects the execution to the `exec()` library function (`lib1`):

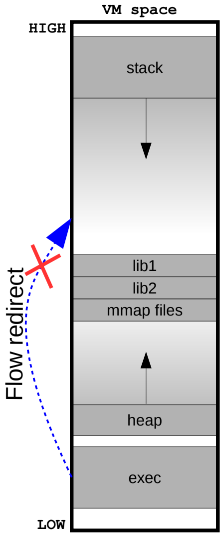
- Trivial if ASLR is off.



How the ASLR works: A simple example

The attacker redirects the execution to the `exec()` library function (`lib1`):

- Trivial if ASLR is off.
- But it fails when the ASLR is on.



How the ASLR works: A simple example

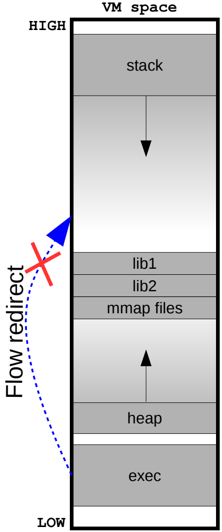
The attacker redirects the execution to the `exec()` library function (`lib1`):

- Trivial if ASLR is off.
- But it fails when the ASLR is on.

Not only the libraries but all other memory areas are randomized:

→ How difficult is to predict the target ?
- Depends on the entropy.

→ Are there other attack vectors ?
- Yes, We have found new weaknesses.



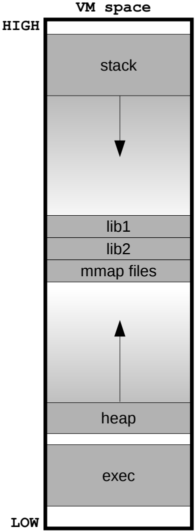
Linux and PaX ASLR weaknesses

Current ASLR was designed considering that some zones are growable but:

- Cannot be used safely because collisions with other allocations cannot be avoided.
- Currently, only used in the Stack and the Heap.

Growable objects impose strong limitations on ASLR design:

- Linux places each object as separately as possible (Stack and Heap)
- Unfortunately, this introduce weaknesses.



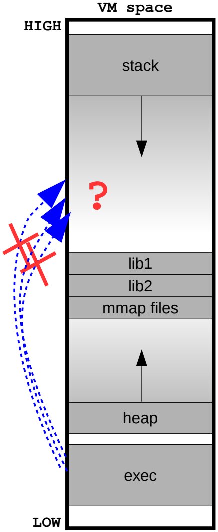
Weakness 1) Too low entropy

Brute force attacks to bypass ASLR:

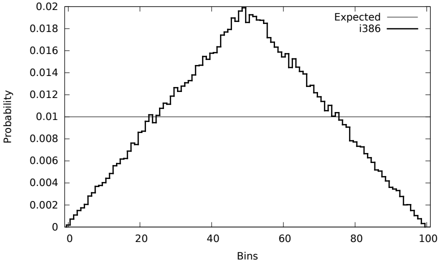
	x86	Entropy	100%	Mean
Linux	32-bit	2^8 (256)	$<$ 1 sec	$<$ 1 sec
	64-bit	2^{28} (260M.)	74 Hours	37 Hours

ASLR entropy and cost time (1000 trials/sec).

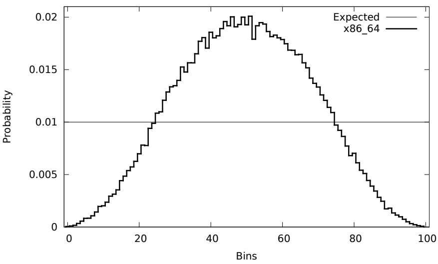
→ ASLR in 32-bit is almost useless (very low entropy).
→ In 64-bit the attack is feasible in some scenarios.
→ The weakness is present since the first Linux ASLR.



Weakness 2) Non-uniform distribution



PaX Libraries distribution in i386



PaX Libraries distribution in x86_64

Libraries are not uniformly distributed:

- Faster attacks by focusing on the most frequent (likely) addresses.
- Other objects are also affected.

Weakness 3) Correlation between objects

Instead of de-randomizing the target object, first de-randomize an intermediate object and then use it to de-randomize the target

$\rightarrow$ We made the first demonstration [Offset2lib]

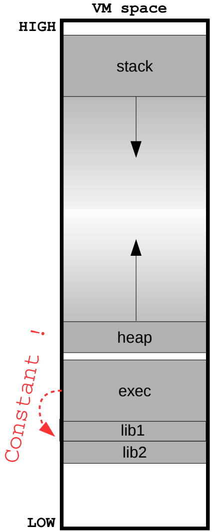
The **Offset2lib** attack vector:

- It bypass the full Linux ASLR on any architecture in < 1 sec.
- It does not use the *GOT* or *PLT*.
- It works even with NX and SSP enabled.
- It exploits a stack buffer overflow.

Weakness 3) Correlation between objects

We have developed a PoC to exploit the **Offset2lib** weakness:

1. De-randomize the executable exploiting a stack buffer overflow.
2. Calculate (offline) the constant distance to the target libraries.
3. The libraries are now de-randomized.



Weakness 3) Correlation between objects

0x5e4000

```
7f36c6a07000-7f36c6bbc000 r-xp   .../libc-2.15.so
7f36c6bbc000-7f36c6dbb000 ---p   .../libc-2.15.so
7f36c6dbb000-7f36c6dbf000 r--p   .../libc-2.15.so
7f36c6dbf000-7f36c6dc1000 rw-p   .../libc-2.15.so
7f36c6dc1000-7f36c6dc6000 rw-p
7f36c6dc6000-7f36c6de8000 r-xp   .../ld-2.15.so
7f36c6fd0000-7f36c6fd3000 rw-p
7f36c6fe5000-7f36c6fe8000 rw-p
7f36c6fe8000-7f36c6fe9000 r--p   .../ld-2.15.so
7f36c6fe9000-7f36c6feb000 rw-p   .../ld-2.15.so
7f36c6feb000-7f36c6fed000 r-xp   /tmp/app-PIE
7f36c71ec000-7f36c71ed000 r--p   /tmp/app-PIE
7f36c71ed000-7f36c71ee000 rw-p   /tmp/app-PIE
7fffe4018000-7fffe4039000 rw-p   [stack]
7fffe41b7000-7fffe41b8000 r-xp   [vdso]
```

Memory map of an application *PIE* compiled.

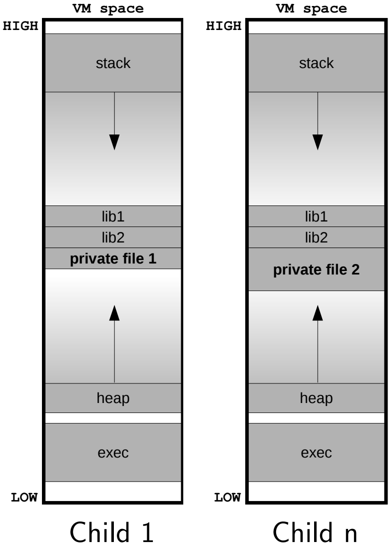
The `ld` is loaded consecutively to the app.: **`0x7f36c6feb000`**

Weakness 4) Inheritance

Again, all child processes share the same memory layout !

New allocations belonging only to a child can be predicted by its parent and siblings !

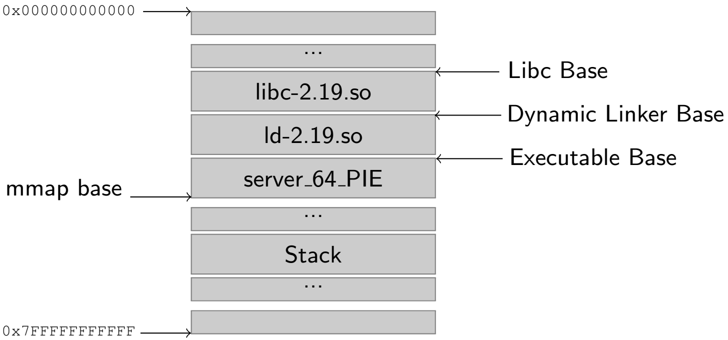
Example: Child 1 can easily guess where the **`private file 2`** has been mapped.



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

```
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]
```

0x5eb000

0x225000

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

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.

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:

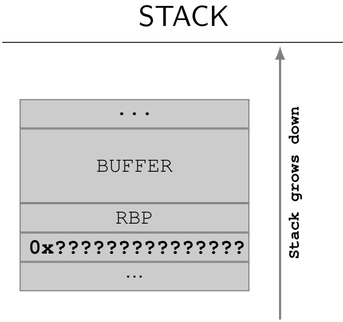
1. Extracting static information
2. Brute force part of saved-IP
3. Calculate base app. address
4. Calculate library offsets
5. Obtain mmapped areas

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** ⇒ **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
..... .....            .....
```

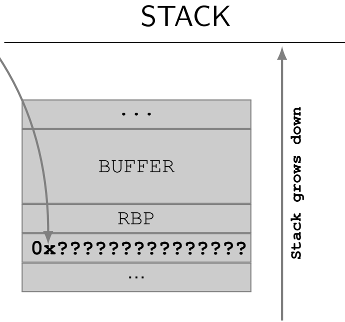


1) Extracting static information

→ 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
..... .....           .....
```



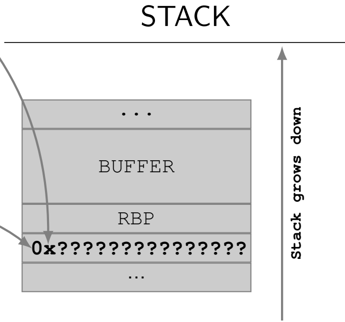
1) Extracting static information

→ 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
..... .....            .....
```

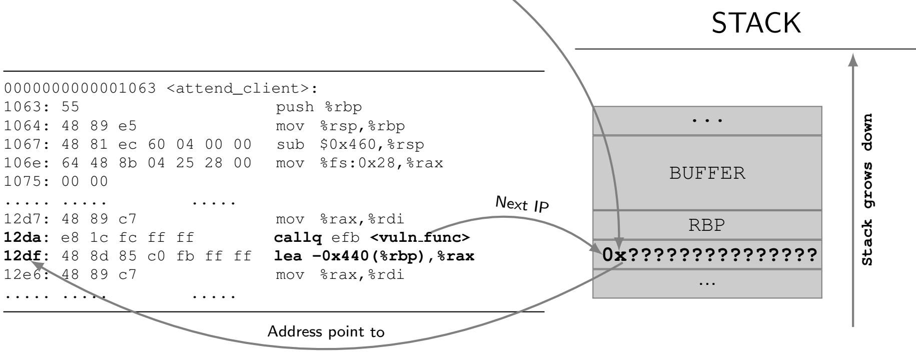
Next IP



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.

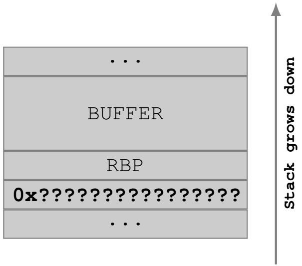


1) Extracting static information

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.

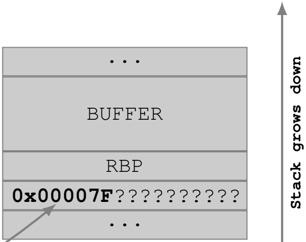
1) Extracting static information

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.

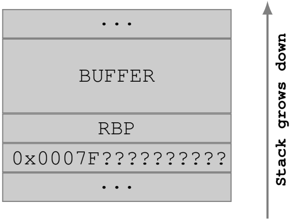
1) Extracting static information

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



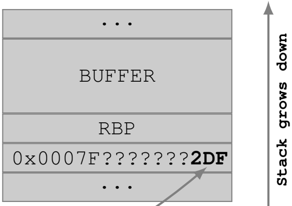
1) Extracting static information

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



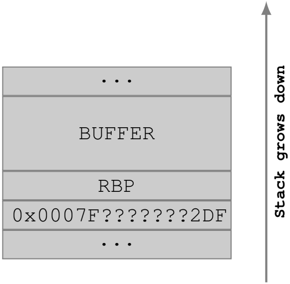
Lower 12 bits

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}

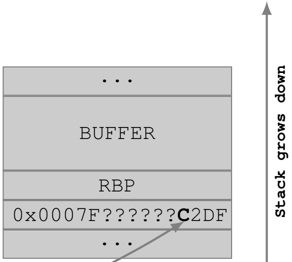
STACK



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

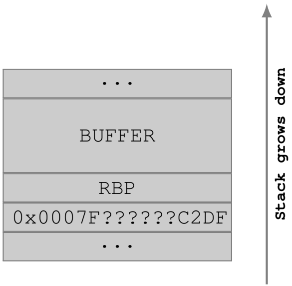


- The unknown 28 random bits: "byte-for-byte" attack.
- The first byte is "special", we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... **0xC2**, 0xD2, 0xE2, 0xF2}

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

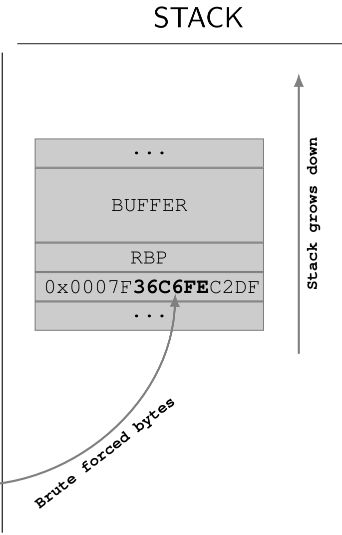


- The unknown 28 random bits: "byte-for-byte" attack.
- The first byte is "special", we know the lowest 4 bits:
  - $0x?2_{16} \rightarrow ??10_2 \rightarrow 2^4 = 16$ attempts
  - {0x02, 0x12, 0x22 ... **0xC2**, 0xD2, 0xE2, 0xF2}
- The remaining 3 bytes → standard "byte-for-byte" attack
  - $3x2^8 =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

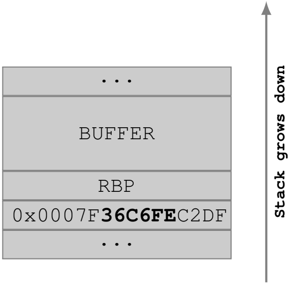


2) Brute forcing Saved-IP address

```
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

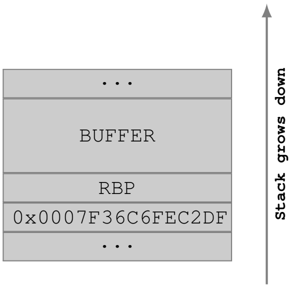


3) Calculating base application address

```
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

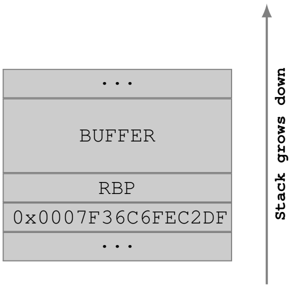


3) Calculating base application address

```
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
..... .....        .....
```

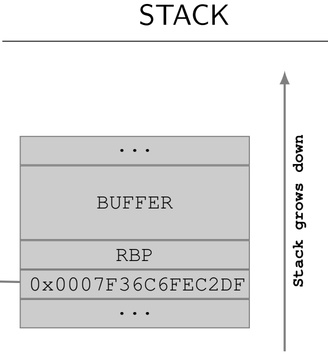
App_base=(savedIP & 0xFFF)-(CALLER_PAGE_OFFSET << 12)

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
..... .....         .....
```



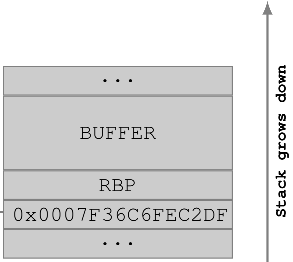
App_base=(savedIP & 0xFFF)-(CALLER_PAGE_OFFSET << 12)

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
..... .....        .....
```

STACK



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

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

App. Base = 0x7F36C6fEB000

4) Calculating library offsets

offset2lib

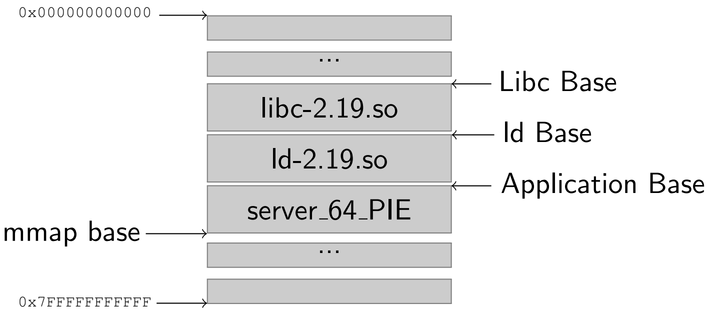
```
7fd1b414f000-7fd1b430a000 r-xp /lib/.../libc-2.19.so
7fd1b430a000-7fd1b450a000 ---p /lib/.../libc-2.19.so
7fd1b450a000-7fd1b450e000 r--p /lib/.../libc-2.19.so
7fd1b450e000-7fd1b4510000 rw-p /lib/.../libc-2.19.so
7fd1b4510000-7fd1b4515000 rw-p
7fd1b4515000-7fd1b4538000 r-xp /lib/.../ld-2.19.so
7fd1b4718000-7fd1b471b000 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:

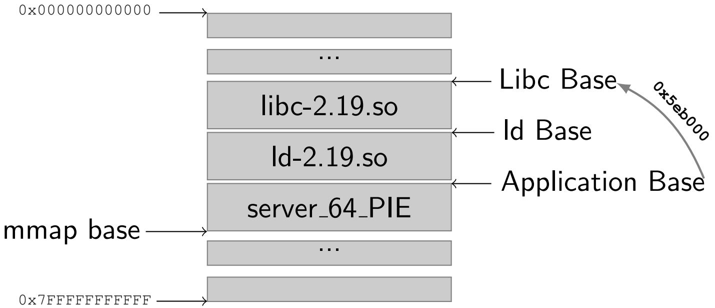
- Application Base = `0x7FD1B473A000`
- Offset2lib (libc) = `0x5eb000`
- Offset2lib (ld) = `0x225000`



5) Getting app. process mapping

Obtaining library base addresses:

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

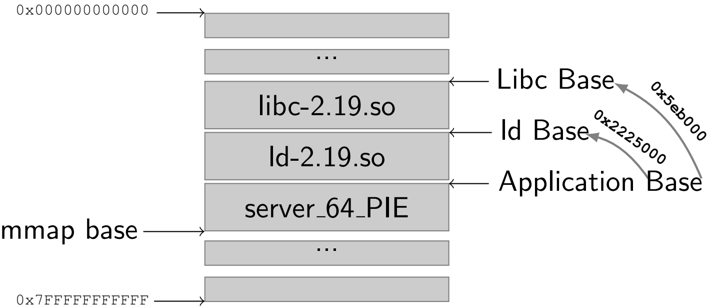


Libc Base = `0x7FD1B473A000` - `0x5eb000` = **`0x7FD1B414F000`**

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`

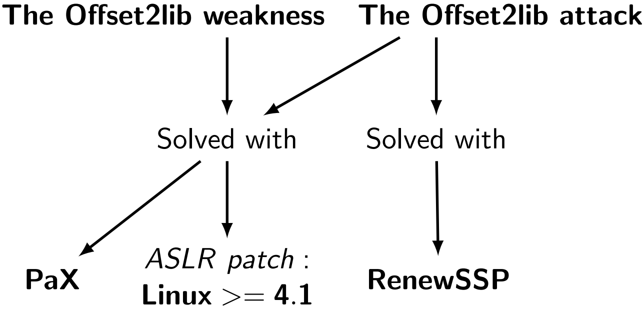
Bypassing NX, SSP and ASLR on 64-bit Linux

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
    - **Linux Kenrel** >= **4.1**
    - **RenewSSP**: Improve stack-smashing-protector.

Solutions overview



All weaknesses are only solved by the ASLR-NG

With Linux Kernel < 4.1

```
# 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 Linux Kernel >= 4.1

```
# 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]
```

Addressing the ASLR weaknesses

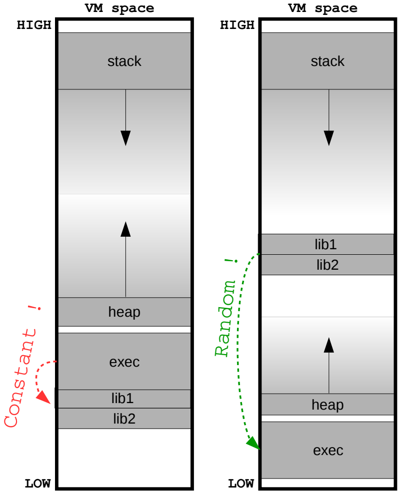
ASLR-NG addresses all these weaknesses but because of the urgency to fix the **Offset2lib** weakness, it was fixed in current Linux.

- It can be seen as a minor part of the **ASLR-NG.**
- It does not remove the correlation problem between all objects.

How we addressed the Offset2lib weakness ?

The particular Offset2lib fix

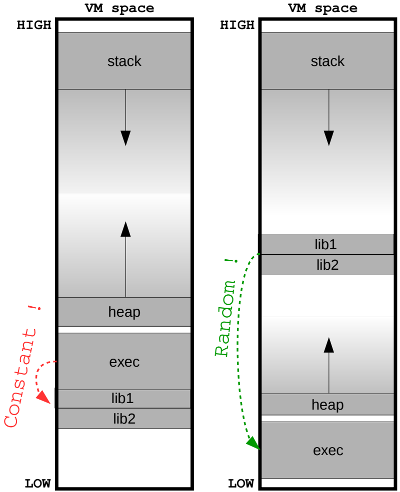
<u>Offset2lib fix:</u>
We have removed the correlation between the executable $\leftrightarrow$ libraries



The particular Offset2lib fix

Offset2lib fix:
We have removed the correlation between the executable ↔ libraries

Attack rewarded by Packet Storm Securit
Offset2lib was classified as 1-day vulnerability

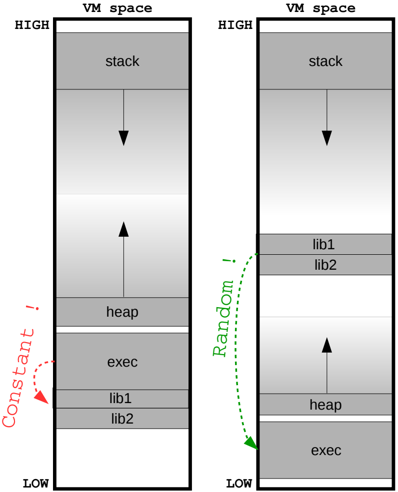


The particular Offset2lib fix

Offset2lib fix:
We have removed the correlation between the executable ↔ libraries

Attack rewarded by Packet Storm Securit
Offset2lib was classified as 1-day vulnerability

Linux Kernel 4.1 patch:
We have created and sent a patch to Linux, which was considered **urgent**.

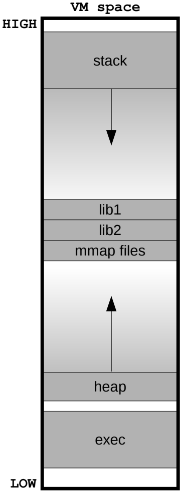


ASLR-NG: Address Space Layout Randomization Next Generation

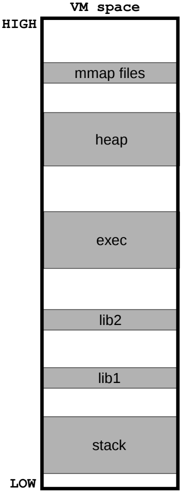
ASLR-NG: The core ideas

A deep analysis of growable objects shows that they (stack and heap) can be bounded.

This key idea allowed me to load objects:



Linux ASLR



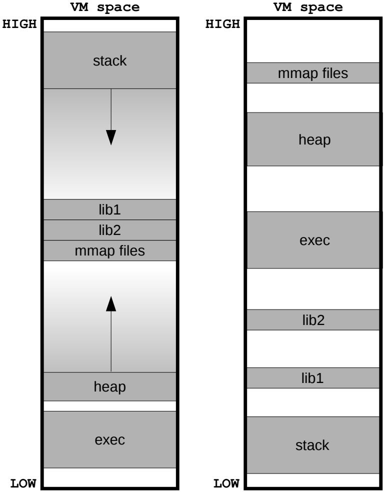
Linux ASLR-NG

ASLR-NG: The core ideas

A deep analysis of growable objects shows that they (stack and heap) can be bounded.

This key idea allowed me to load objects:

- Freely along the VM:
  → Huge increment of entropy.



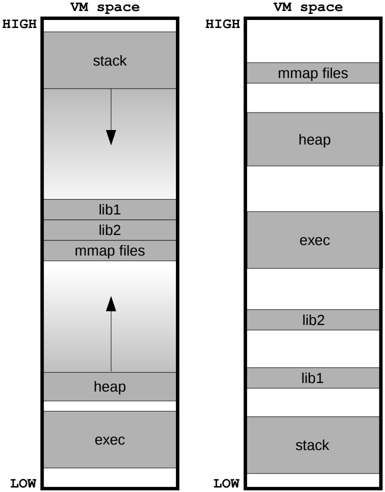
Linux ASLR Linux ASLR-NG

ASLR-NG: The core ideas

A deep analysis of growable objects shows that they (stack and heap) can be bounded.

This key idea allowed me to load objects:

- Freely along the VM:
  → Huge increment of entropy.
- Uniformly distributed:
  → No more likely addresses.



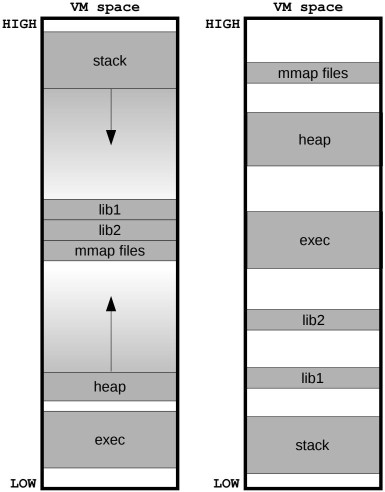
Linux ASLR Linux ASLR-NG

ASLR-NG: The core ideas

A deep analysis of growable objects shows that they (stack and heap) can be bounded.

This key idea allowed me to load objects:

- Freely along the VM:
  → Huge increment of entropy.
- Uniformly distributed:
  → No more likely addresses.
- Uncorrelated:
  → No more correlated attacks.



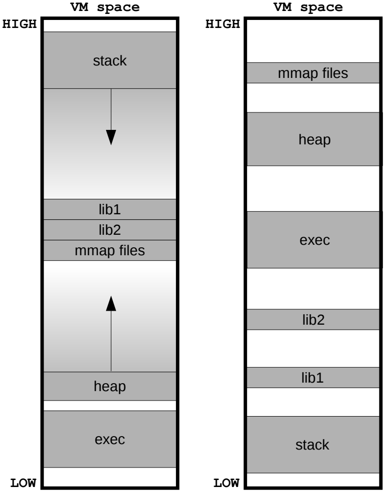
Linux ASLR Linux ASLR-NG

ASLR-NG: The core ideas

A deep analysis of growable objects shows that they (stack and heap) can be bounded.

This key idea allowed me to load objects:

- Freely along the VM:
  → Huge increment of entropy.
- Uniformly distributed:
  → No more likely addresses.
- Uncorrelated:
  → No more correlated attacks.
- Have different VM layout:
  → Forking model more secure.



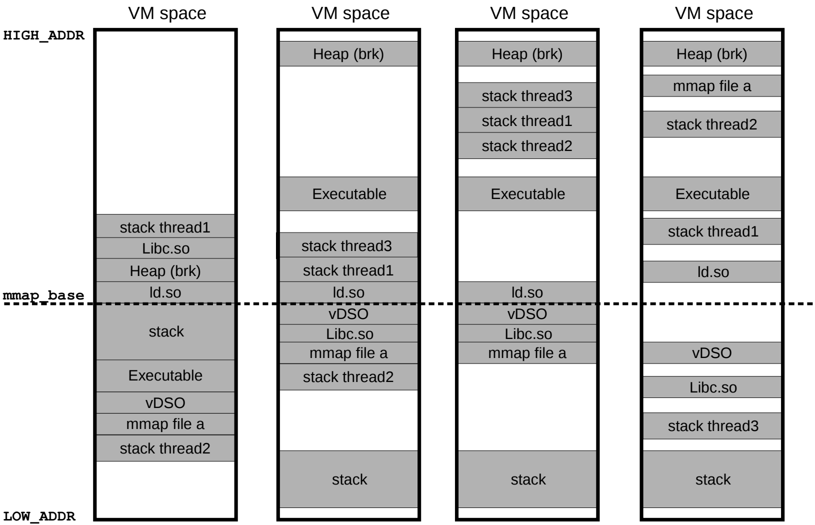
Linux ASLR   Linux ASLR-NG

ASLR-NG: New randomisation forms

We have categorized and designed new randomization forms:

	Feature	Description
When	Per-boot	Every time the system is booted.
	Per-exec	Every time a new image is executed.
	Per-fork	Every time a new process is spawned. **new!**
	Per-object	Every time a new object is created. **new!**
What	Stack	Stack of the main process.
	LD	Dynamic linker/loader.
	Executable	Loadable segments (text, data, bss, ...).
	Heap	Old-fashioned dynamic memory of the process: `brk()`. **improved !**
	vDSO/VVAR	Objects exported by the kernel to the user space.
	Mmaps/libs	Objects allocated calling `mmap()`. **improved !**
How	Partial VM	A sub-range of the VM space is used to map the object.
	Full VM	The full VM space is used to map the object. **new!**
	Isolated-object	The object is randomised independently from any other. **new!**
	Sub-page	Page offset bits are randomised. **new!**
	Bit-slicing	Different slices of the address are randomised at different times. **Google!**
	Direction	Topdown/downtop search side used on a first-fit allocation strategy. **new!**
	Specific-zone	A base address and a direction where objects are allocated together. **new!**

ASLR-NG: Profile modes



(a) Concentrated (b) Conservative (c) Extended (d) Paranoid

ASLR-NG: Evaluation

We have developed ASLRA, a test suit to analyze the entropy of objects. ASLRA is composed of three tools:

1. Simulator:
   → Simulates several ASLRs, including the proposed ASLR-NG.
2. Sampler:
   → An application which generate million of samples (address of mapped objects) and saves the raw data.
3. Analyzer:
   → Performs the statistical analysis.
   → Individual byte, Shannon entropy, flipping bits, etc.

ASLR-NG: Evaluation

We have developed ASLRA, a test suit to analyze the entropy of objects. ASLRA is composed of three tools:

1. Simulator:
   → Simulates several ASLRs, including the proposed ASLR-NG.
2. Sampler:
   → An application which generate million of samples (address of mapped objects) and saves the raw data.
3. Analyzer:
   → Performs the statistical analysis.
   → Individual byte, Shannon entropy, flipping bits, etc.

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log_2 p(x)$$

The most interesting!

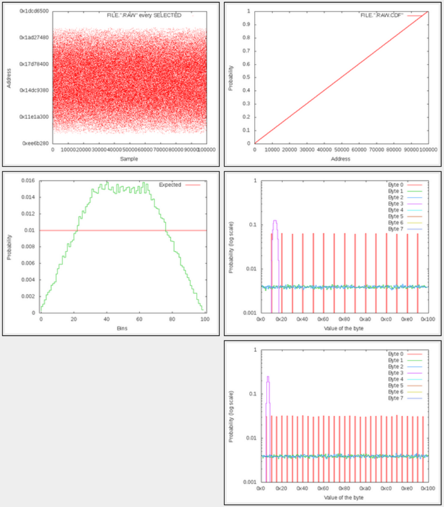
ASLR-NG: ASLR analyzer tool

Sbrk

Parameter	Value
Samples:	100000
Range:	0x000000000bf3cc61
Errors:	0
Min:	0x000000001006bbc0
Max:	0x000000001bfa8820
Mean measured:	0x000000001601a6e1
Mean theoric:	0x000000001600a1f0
Mean error:	0.02%
Median measured:	0x000000001600c670
Median theoric:	0x000000001600a1f0
Median error:	0.00%
StdDev measured:	0x2935654
StdDev theoric:	0x37349a6
StdDev error:	25.35%
And-ed:	0x0000000010000000
Or-ed:	0x000000001ffffff0
Bits that change:	0x000000000ffffff0
Bits stick to 0:	4
Flipping bits:	**24**
Shannon entropy:	23.36
Byte Shannon:	23.37
Frequent addresses:	histogram

Chi2 tests

bins [	low <	Chi2	< high]
32 [	12.20 <	24699.01	< 61.10]
64 [	33.91 <	24837.17	< 103.44]
100 [	61.14 <	24891.05	< 148.23]
256 [	190.87 <	25044.97	< 330.52]
1024 [	888.90 <	25848.37	< 1168.50]
2048 [	1854.96 <	26863.10	< 2250.44]
4096 [	3821.03 <	28760.39	< 4380.37]
8192 [	7801.17 <	32899.96	< 8592.23]
8263 [	7870.46 <	32861.88	< 8664.94]



ASLR analyser: Screenshot of a Heap (brk)

Linux vs PaX vs ASLR-NG

	32-bits			64-bits		
Object	**Linux**	**PaX**	**ASLR-NG**	**Linux**	**PaX**	**ASLR-NG**
ARGV	11	27	31.5	22	39	47
Main stack	19	23	27.5	30	35	43
Heap (brk)	13	23.3	27.5	28	35	43
Heap (mmap)	8	15.7	27.5	28	28.5	43
Thread stacks	8	15.7	27.5	28	28.5	43
Sub-page object	-	-	27.5	-	-	43
Regular mmaps	8	15.7	19.5	28	28.5	35
Libraries	8	15.7	19.5	28	28.5	35
vDSO	8	15.7	19.5	21.4	28.5	35
Executable	8	15	19.5	28	27	35
Huge pages	0	5.7	9.5	19	19.5	26

Comparative summary of bits of entropy.

ASLR-NG: Benefits

The main features of ASLR-NG are:

- Uses full memory space to randomise objects, which in turn provides maximum entropy.
- A novel solution for reducing fragmentation, without reducing entropy.
- Objects containing sensitive information are automatically isolated.
- Sequentially loaded libraries are randomised.
- It provides a strong protection against absolute and correlation attacks.
- Effectively removes the four weaknesses previously identified.

During the design of the ASLR-NG we have fixed **three** vulnerabilities in the Linux ASLR that were **rewarded** by **Google**.

The Offset2lib attack was rewarded by **Packet Storm Security** classified as a 1-day vulnerability.

Questions ?

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