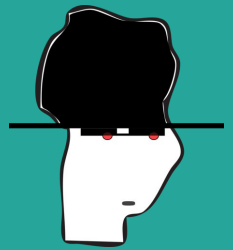


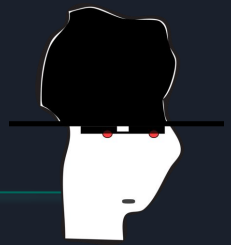
Protecciones contra Explotación2

—

Seguridad Ofensiva

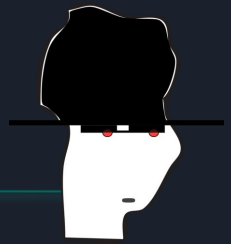
by Joshep Cortez S.
OLAPIC - FAMAF - IUA

x86: Buffer Overflow (ej1)



```
int main() {
   int cookie;
   char buf[80];

   gets(buf); //Lee hasta el primer ...
   if (cookie == 0x41424344)
      printf("Ganaste!\n");
}
```

x86: Protecciones

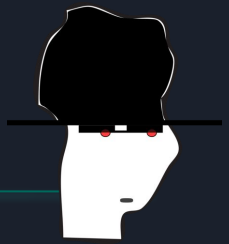


Los sistemas operativos (kernel) y compiladores tienen distintos mecanismos para minimizar el impacto y reducir las posibles vulnerabilidades de corrupción de memoria:

- ~~DEP / NX~~
- ASLR
- Stack Canaries

```
joe@zoidberg:~/Seg/Rev/bof$ ~/Soft/checksec.sh/checksec --file=01-challenge
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols       FORTIFY Fortified
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   65) Symbols     No    0
```

x86: ASLR

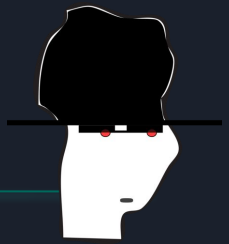


ADDRESS SPACE LAYOUT RANDOMIZATION

Protección para asegurar que los rangos de direcciones de memoria importantes son random en cada ejecución.

Objetivo: Mitigar ataques de/a memoria que dependen de direcciones fijas en el stack, .text, heap, libc, etc.

x86: ASLR

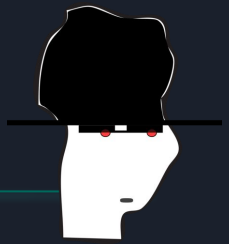


ADDRESS SPACE LAYOUT RANDOMIZATION

```
joe@zoidberg:~/Seg/Rev/bof$ cat /proc/sys/kernel/randomize_va_space
2
```

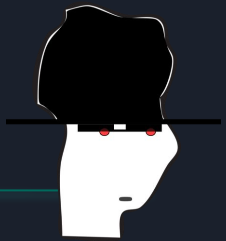
0: NO ASLR
1: Conservative Randomization
2: Full Randomization

x86: ASLR



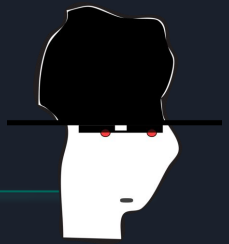
```
joe@zoidberg:~/Seg/Rev/bof$ cat /proc/sys/kernel/randomize_va_space
2
joe@zoidberg:~/Seg/Rev/bof$ ldd ./01-challenge
        linux-gate.so.1 (0×f7fb3000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0×f7d9c000)
        /lib/ld-linux.so.2 (0×f7fb4000)
joe@zoidberg:~/Seg/Rev/bof$ ldd ./01-challenge
        linux-gate.so.1 (0×f7fcf000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0×f7db8000)
        /lib/ld-linux.so.2 (0×f7fd0000)
```

x86: ASLR



```
joe@zoidberg:~/Seg/Rev/bof$ echo 0 | sudo tee /proc/sys/kernel/randomize_va_space
0
joe@zoidberg:~/Seg/Rev/bof$ ldd 01-challenge
        linux-gate.so.1 (0xf7fd3000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0xf7dbc000)
        /lib/ld-linux.so.2 (0xf7fd4000)
joe@zoidberg:~/Seg/Rev/bof$ ldd 01-challenge
        linux-gate.so.1 (0xf7fd3000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0xf7dbc000)
        /lib/ld-linux.so.2 (0xf7fd4000)
joe@zoidberg:~/Seg/Rev/bof$ ldd 01-challenge
        linux-gate.so.1 (0xf7fd3000)
        libc.so.6 ⇒ /lib/i386-linux-gnu/libc.so.6 (0xf7dbc000)
        /lib/ld-linux.so.2 (0xf7fd4000)
```

x86: ASLR

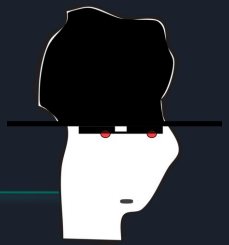


```
joe@zoidberg:~/Seg/Rev/bof$ cat /proc/self/maps
557b08357000-557b08359000 r--p 00000000 fe:02 2753679                    /usr/bin/cat
557b08359000-557b0835e000 r-xp 00002000 fe:02 2753679                    /usr/bin/cat
557b0835e000-557b08360000 r--p 00007000 fe:02 2753679                    /usr/bin/cat
557b08361000-557b08362000 r--p 00009000 fe:02 2753679                    /usr/bin/cat
557b08362000-557b08363000 rw-p 0000a000 fe:02 2753679                    /usr/bin/cat
557b08aa3000-557b08ac4000 rw-p 00000000 00:00 0                          [heap]
7f75c18bc000-7f75c18de000 rw-p 00000000 00:00 0
```

. . .

```
7f75c1d9d000-7f75c1dbb000 r-xp 00001000 fe:02 2752715                    /usr/lib/x86_64-linux-gnu/ld-2.30.so
7f75c1dbb000-7f75c1dc3000 r--p 0001f000 fe:02 2752715                    /usr/lib/x86_64-linux-gnu/ld-2.30.so
7f75c1dc3000-7f75c1dc4000 r--p 00000000 fe:02 3584017                    /usr/lib/locale/en_US.utf8/LC_IDENTIFI
7f75c1dc4000-7f75c1dc5000 r--p 00027000 fe:02 2752715                    /usr/lib/x86_64-linux-gnu/ld-2.30.so
7f75c1dc5000-7f75c1dc6000 rw-p 00028000 fe:02 2752715                    /usr/lib/x86_64-linux-gnu/ld-2.30.so
7f75c1dc6000-7f75c1dc7000 rw-p 00000000 00:00 0
7fffc9bbd000-7fffc9bde000 rw-p 00000000 00:00 0                          [stack]
7fffc9bf9000-7fffc9bfc000 r--p 00000000 00:00 0                          [vvar]
7fffc9bfc000-7fffc9bfd000 r-xp 00000000 00:00 0                          [vdso]
```

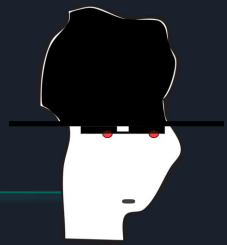
x86: ASLR



```
joe@zoidberg:~$ ps -waux | grep 01-chall
joe       134710  0.0  0.0   2416   612 pts/5    S+   12:10   0:00 ./01-challenge
joe       134731  0.0  0.0   6148   960 pts/1    R+   12:11   0:00 grep 01-chall
```

```
joe@zoidberg:~$ cat /proc/134710/maps
08048000-0804b000 r-xp 00000000 fe:03 22283659                           /home/joe/Seg/Rev/bof/01-challenge
0804b000-0804c000 r-xp 00002000 fe:03 22283659                           /home/joe/Seg/Rev/bof/01-challenge
0804c000-0804d000 rwxp 00003000 fe:03 22283659                           /home/joe/Seg/Rev/bof/01-challenge
08c5d000-08c7f000 rwxp 00000000 00:00 0                                  [heap]
f7d36000-f7f14000 r-xp 00000000 fe:02 2755830                            /usr/lib/i386-linux-gnu/libc-2.30.so
f7f14000-f7f16000 r-xp 001dd000 fe:02 2755830                            /usr/lib/i386-linux-gnu/libc-2.30.so
f7f16000-f7f18000 rwxp 001df000 fe:02 2755830                            /usr/lib/i386-linux-gnu/libc-2.30.so
f7f18000-f7f1a000 rwxp 00000000 00:00 0
f7f48000-f7f4a000 rwxp 00000000 00:00 0
f7f4a000-f7f4d000 r--p 00000000 00:00 0                                  [vvar]
f7f4d000-f7f4e000 r-xp 00000000 00:00 0                                  [vdso]
f7f4e000-f7f76000 r-xp 00000000 fe:02 2755805                            /usr/lib/i386-linux-gnu/ld-2.30.so
f7f76000-f7f77000 r-xp 00027000 fe:02 2755805                            /usr/lib/i386-linux-gnu/ld-2.30.so
f7f77000-f7f78000 rwxp 00028000 fe:02 2755805                            /usr/lib/i386-linux-gnu/ld-2.30.so
ffaf6000-ffb17000 rwxp 00000000 00:00 0                                  [stack]
```

x86:

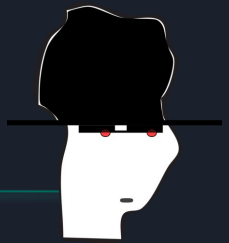


```
joe@zoidberg:~/Seg/Rev/hellow$ readelf -S ~/Seg/Rev/bof/01-challenge
There are 29 section headers, starting at offset 0x3760:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048194 000194 000013 00   A  0   0  1
  [ 2] .note.gnu.build-i NOTE            080481a8 0001a8 000024 00   A  0   0  4
  [ 3] .note.ABI-tag     NOTE            080481cc 0001cc 000020 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        080481ec 0001ec 000020 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          0804820c 00020c 000060 10   A  6   1  4
  [ 6] .dynstr           STRTAB          0804826c 00026c 00004f 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          080482bc 0002bc 00000c 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         080482c8 0002c8 000020 00   A  6   1  4
  [ 9] .rel.dyn          REL             080482e8 0002e8 000008 08   A  5   0  4
  [10] .rel.plt          REL             080482f0 0002f0 000018 08  RAI  5  22  4
  [11] .init             PROGBITS        08049000 001000 000020 00  AX  0   0  4
  [12] .plt              PROGBITS        08049020 001020 000040 04  AX  0   0 16
  [13] .text             PROGBITS        08049060 001060 0001c5 00  AX  0   0 16
  [14] .fini             PROGBITS        08049228 001228 000014 00  AX  0   0  4
  [15] .rodata           PROGBITS        0804a000 002000 000011 00   A  0   0  4
  [16] .eh_frame_hdr     PROGBITS        0804a014 002014 000044 00   A  0   0  4
  [17] .eh_frame         PROGBITS        0804a058 002058 000118 00   A  0   0  4
  [18] .init_array       INIT_ARRAY      0804bf0c 002f0c 000004 04  WA  0   0  4
  [19] .fini_array       FINI_ARRAY      0804bf10 002f10 000004 04  WA  0   0  4
  [20] .dynamic          DYNAMIC         0804bf14 002f14 0000e8 08  WA  6   0  4
  [21] .got              PROGBITS        0804bffc 002ffc 000004 04  WA  0   0  4
  [22] .got.plt          PROGBITS        0804c000 003000 000018 04  WA  0   0  4
  [23] .data             PROGBITS        0804c018 003018 000008 00  WA  0   0  4
  [24] .bss              NOBITS          0804c020 003020 000004 00  WA  0   0  1
  [25] .comment          PROGBITS        00000000 003020 00001d 01  MS  0   0  1
  [26] .symtab           SYMTAB          00000000 003040 000410 10     27  43  4
  [27] .strtab           STRTAB          00000000 003450 00020f 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 00365f 000101 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
```

READONLY

WRITABLE

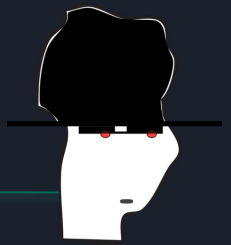
x86: ASLR



ADDRESS SPACE LAYOUT RANDOMIZATION

- 2004 (mayo) - OpenBSD 3.5 (mmap)
- 2005 (junio) - Linux Kernel 2.6.12 (stack, mmap)
- 2007 (enero) - Windows Vista (full)
- 2007 (octubre) - Mac OSX 10.5 Leopard (sys libraries)
- 2010 (octubre) - Windows Phone 7 (full)
- 2011 (marzo) - iPhone iOS 4.3 (full)
- 2011 (julio) - Mac OSX 10.7 Lion (full)

x86: ASLR

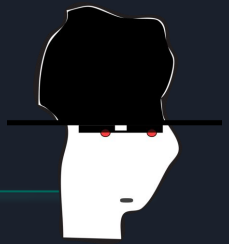


FUNDAMENTOS:

- Los segmentos de memoria no tienen que ser direcciones estáticas, sino que deben ser únicos para cada ejecución.
- Un buffer overflow podría servir para controlar EIP, lo cual no resulta un problema si un atacante no sabe a donde saltar para lograr.

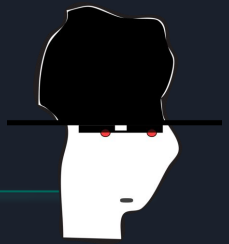
Se debe trabajar sin asumir dónde hay algo en la memoria.

x86: ASLR on (corrida 1)



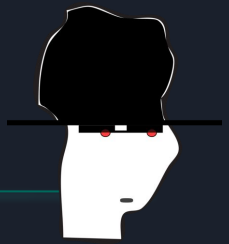
https://github.com/RPISEC/MBE

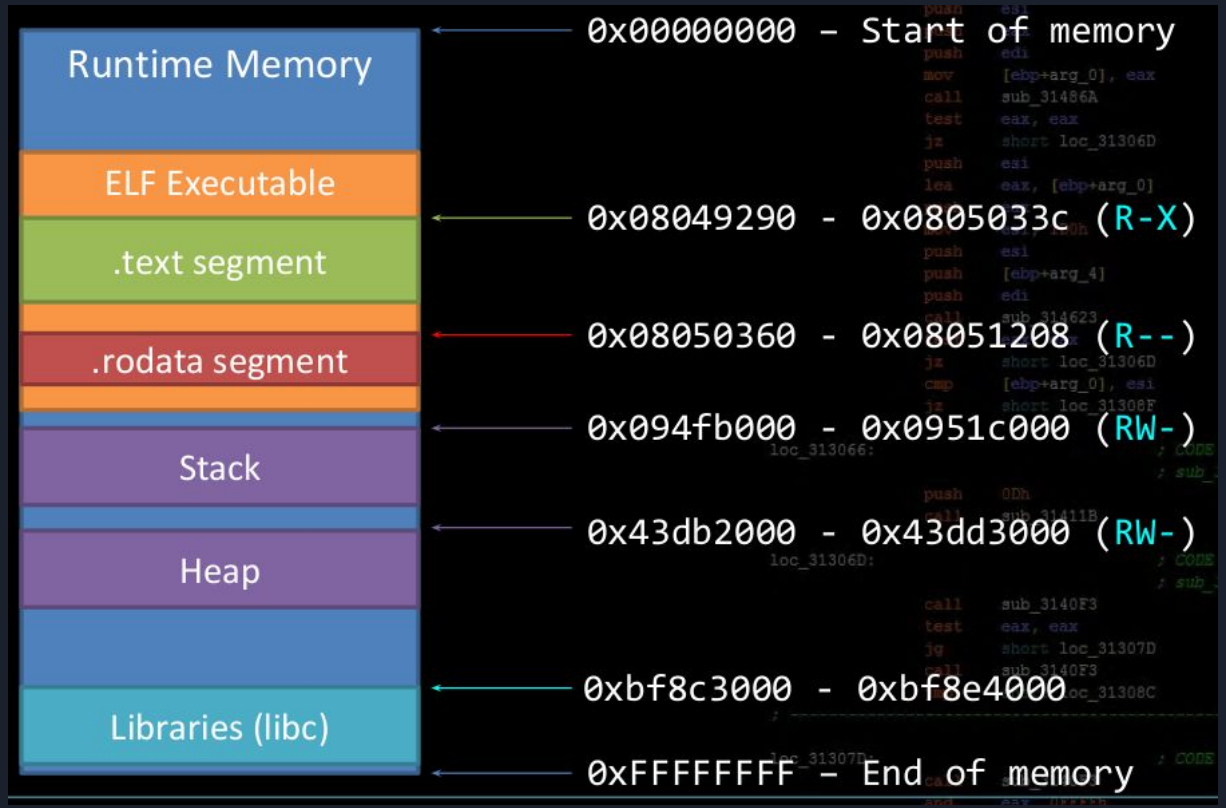
x86: ASLR on (corrida 2)



https://github.com/RPISEC/MBE

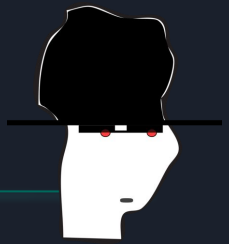
x86: ASLR on (corrida 3)





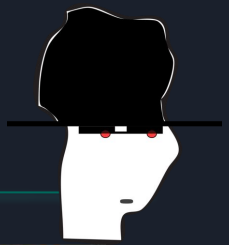
https://github.com/RPISEC/MBE

x86: ASLR on



```
joe@zoidberg:~/Seg/Rev/bof$ echo 2 | sudo tee /proc/sys/kernel/randomize_va_space
2
joe@zoidberg:~/Seg/Rev/bof$ cat get_stack.c
int
main(int argc, char*argv[], char*envp[]){
    int local;
    printf ("STACK@ %p\n", (void*) &local + 4*2);
}
joe@zoidberg:~/Seg/Rev/bof$ ./get_stack
STACK@ 0×ffd41b0c
joe@zoidberg:~/Seg/Rev/bof$ ./get_stack
STACK@ 0×ffc3173c
```

x86: ASLR on



```
joe@zoidberg:~/Seg/Rev/bof$ objdump -Mintel -d 01-challenge |grep -A16 '<main>:'
08049172 <main>:
 8049172:       55                      push   ebp
 8049173:       89 e5                   mov    ebp,esp
 8049175:       83 ec 54                sub    esp,0×54
 8049178:       c7 45 fc 00 00 00 00    mov    DWORD PTR [ebp-0×4],0×0
 804917f:       8d 45 ac                lea    eax,[ebp-0×54]
 8049182:       50                      push   eax
 8049183:       e8 a8 fe ff ff          call   8049030 <gets@plt>
 8049188:       83 c4 04                add    esp,0×4
 804918b:       81 7d fc 44 43 42 41    cmp    DWORD PTR [ebp-0×4],0×41424344
 8049192:       75 0d                   jne    80491a1 <main+0×2f>
 8049194:       68 08 a0 04 08          push   0×804a008
 8049199:       e8 a2 fe ff ff          call   8049040 <puts@plt>
 804919e:       83 c4 04                add    esp,0×4
 80491a1:       b8 00 00 00 00          mov    eax,0×0
 80491a6:       c9                      leave
 80491a7:       c3                      ret
```

x86: ASLR on => no todo rand

```
joe@zoidberg:~/Seg/Rev/bof$ python -c "print 'A'*80+'
BBBB'+'CCCC'+\"\x94\x91\x04\x08\" " | ./01-challenge
YOU WIN!
Segmentation fault
joe@zoidberg:~/Seg/Rev/bof$ python -c "print 'A'*80+'
BBBB'+'CCCC'+\"\x94\x91\x04\x08\" " | ./01-challenge
YOU WIN!
Segmentation fault
joe@zoidberg:~/Seg/Rev/bof$ python -c "print 'A'*80+'
BBBB'+'CCCC'+\"\x94\x91\x04\x08\" " | ./01-challenge
YOU WIN!
Segmentation fault
```

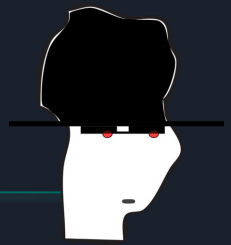
x86: ASLR no tan random linu

Según como haya sido compilado podría haber partes no randomizadas:

```
.text / .plt / .init / .fini   - (R-X)
.got / .got.plt / .data / .bss - (RW-)
.rodata                        (R--)
```

Entonces podrían encontrarse gadgets de todas maneras y aplicar ROP.

x86: PIE / PIC



POSITION INDEPENDENT EXECUTABLE / CODE

Feature de compilación que permite que el código se direccione relativamente.

Es decir cambie su base de direccionamiento en cada ejecución.

Toda biblioteca compartida moderna DEBERÍA ser compilada con esta opción.

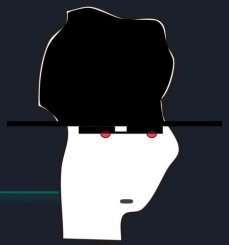
x86: PIE / PIC

```
joe@zoidberg:~/Seg/Rev/bof$ ~/Soft/checksec.sh/checksec --file=01-challenge
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH
Fortified       Fortifiable     FILE
Partial RELRO   No canary found   NX disabled   No PIE          No RPATH   No RUNPATH
```

```
joe@zoidberg:~/Seg/Rev/bof$ gcc -o 01-challenge 01-challenge.c
01-challenge.c: In function ‘main’:
01-challenge.c:11:5: warning: implicit declaration of function ‘gets’;
on-declaration]
   11 |     gets(buf);
      |     ^~~~
      |     fgets
```

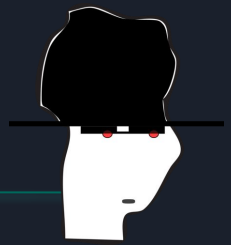
```
joe@zoidberg:~/Seg/Rev/bof$ ~/Soft/checksec.sh/checksec --file=01-challenge
RELRO           STACK CANARY      NX            PIE             RPATH      R
Fortified       Fortifiable     FILE
Partial RELRO   No canary found   NX enabled    PIE enabled     No RPATH   N
```

x86: PIE / PIC



```
000011a9 <main>:
    11a9:       8d 4c 24 04             lea    ecx,[esp+0×4]
    11ad:       83 e4 f0                and    esp,0×fffffff0
    11b0:       ff 71 fc                push   DWORD PTR [ecx-0×4]
    11b3:       55                      push   ebp
    11b4:       89 e5                   mov    ebp,esp
    11b6:       51                      push   ecx
    11b7:       83 ec 64                sub    esp,0×64
    11ba:       c7 45 f4 00 00 00 00    mov    DWORD PTR [ebp-0×c],0×0
    11c1:       83 ec 0c                sub    esp,0×c
    11c4:       8d 45 a4                lea    eax,[ebp-0×5c]
    11c7:       50                      push   eax
    11c8:       e8 fc ff ff ff          call   11c9 <main+0×20>
    11cd:       83 c4 10                add    esp,0×10
    11d0:       81 7d f4 44 43 42 41    cmp    DWORD PTR [ebp-0×c],0×41424344
```

x86: Bypass ASLR?



Toda biblioteca compartida moderna DEBERÍA ser compilada con esta opción.

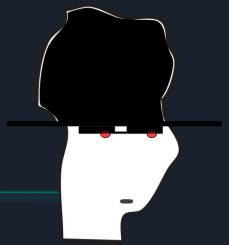
Asumiendo el control de EIP:

Desconocemos las direcciones de "todo".

Opciones?

- **Information disclosure (leak)**
- **Partial Overwrite + crash state**
- **Partial Overwrite + bruteforce**

x86: Leak

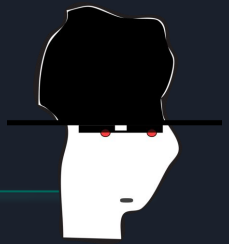


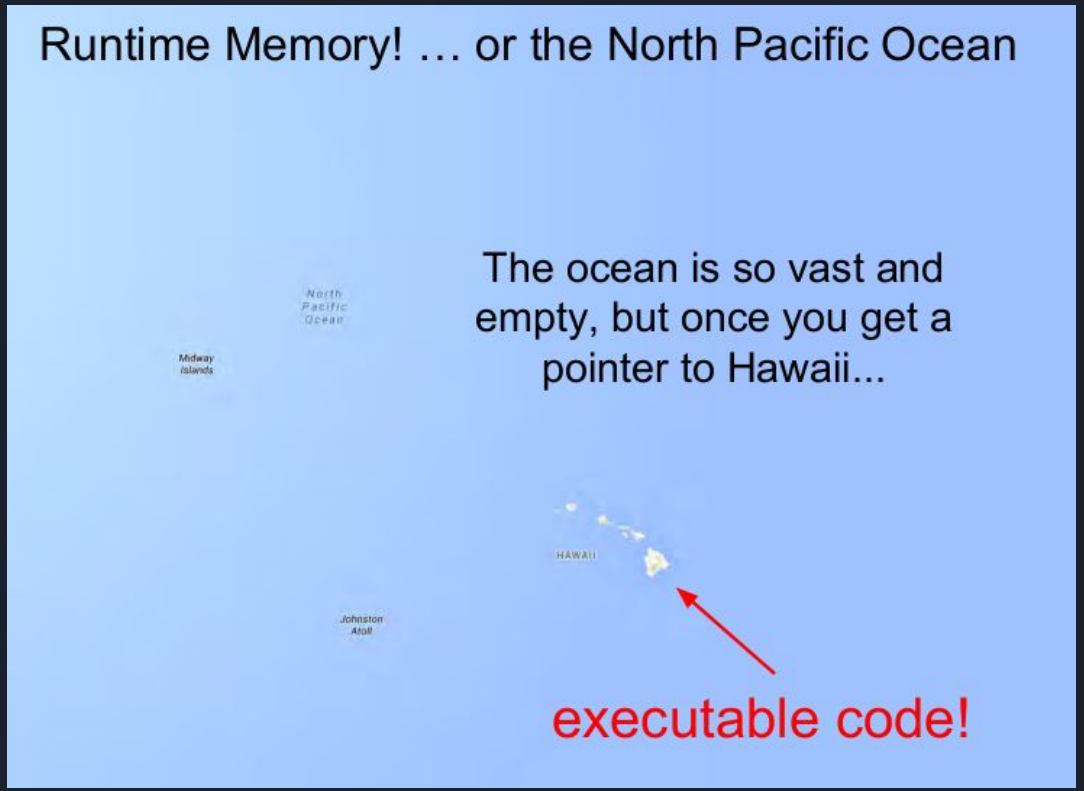
Toda biblioteca compartida moderna DEBERÍA ser compilada con esta opción.

Se trata de extraer información significativa del programa protegido, como por ejemplo una dirección de memoria de interes.

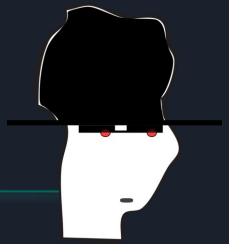
Si se puede filtrar cualquier tipo de puntero al código durante el ataque, es probable que se pueda derrotar a ASLR

x86: 1 puntero?

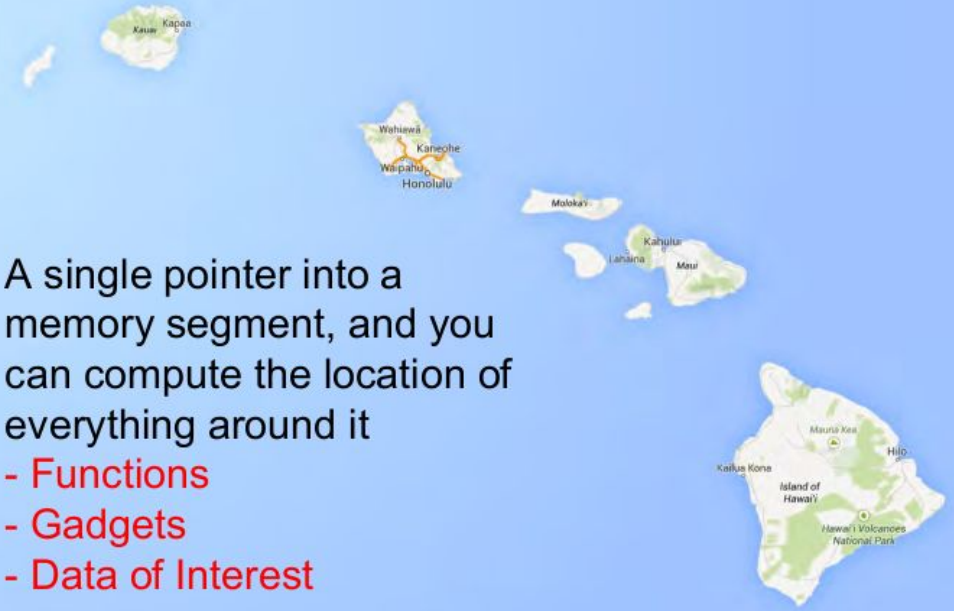




x86: 1 puntero?



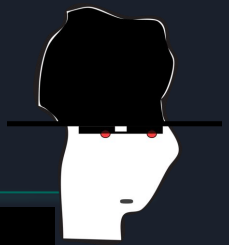
Everything becomes relative



A single pointer into a memory segment, and you can compute the location of everything around it

- Functions
- Gadgets
- Data of Interest

x86: 1 puntero?



By Example:

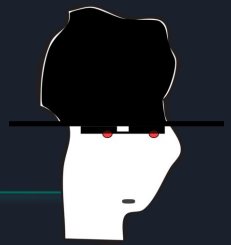
-You have a copy of the libc binary, ASLR is on

-You've leaked a pointer off the stack to printf()
  printf() is @ 0xb7e72280

-Look at the libc binary, how far away is system() from printf()?
  system() is -0xD0F0 bytes away from printf()

therefore system() is at @ 0xb7e65190
(0xb7e65190-0xD0F0)

x86: Partial Overwrite



Si no se tiene un leak de memoria, pero se puede sobreescribir parcialmente una direccion:

0xb756b132 — 100% exploit reliability

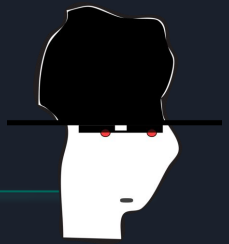
0xb758e132

0xb75e5132 — 6.25% exploit reliability

0xb754d132

0xb75cf132 — 0.024% exploit reliability

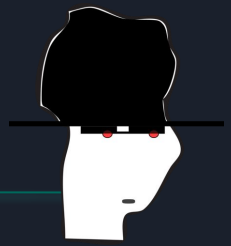
x86: Resumiendo



- Buscar leaks de información interesante.
- Analizar el contexto:
  - que hay en los registros ?
  - que hay en la pila ?
- Profundizar el análisis.

Hacer y leer research:
https://cybersecurity.upv.es/attacks/offset2lib/offset2lib-paper.pdf

x86: Resumiendo2

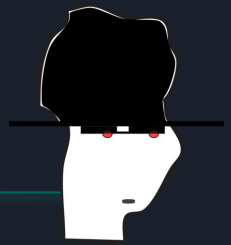


- **Al igual que otras tecnologías de mitigación, ASLR es una solución "táctica" que solo dificulta las cosas**

- **Las vulnerabilidades y exploits se vuelven más complejas y precisas**

- **DEP y ASLR son los dos pilares principales de las tecnologías modernas de mitigación de exploits y normalmente necesitan más de 1 bug.**

x86: Stack protections



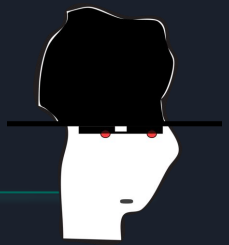
- **Canaries | Stack guards | Cookies**

Son una protección que requiere opciones de compilación y por parte del sistema operativo un chequeo de datos.

Durante la ejecución se incorporan valores aleatorios (canaries) en cada “stack frame”

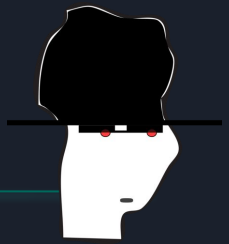
Se verifica la integridad antes de ejecutar RET

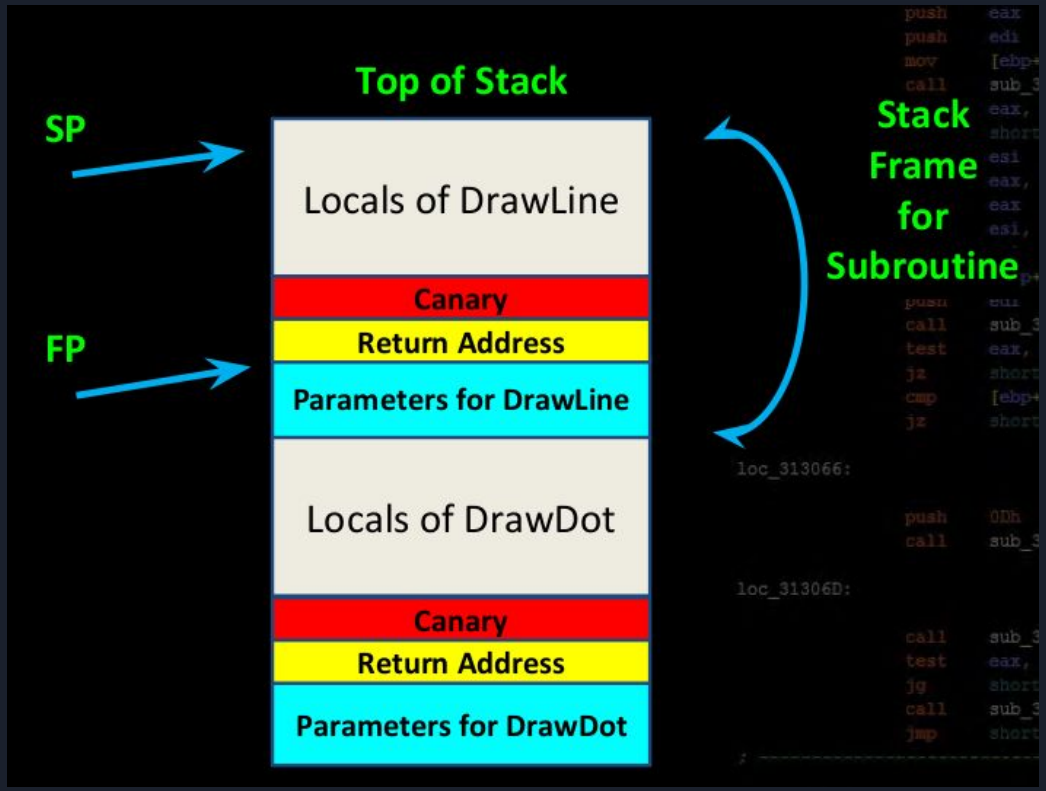
x86: Canary



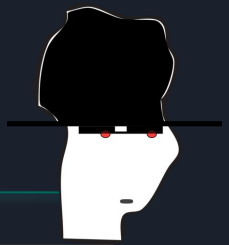
- **Número aleatorio**
- **Se pushea a la pila después de ciertos disparadores: ej. una llamada a funcion**
- **Se popea de la pila luego de que es verificado.**

x86: Canary




Top of Stack
SP
FP
Locals of DrawLine
Canary
Return Address
Parameters for DrawLine
Locals of DrawDot
Canary
Return Address
Parameters for DrawDot
Stack Frame for Subroutine

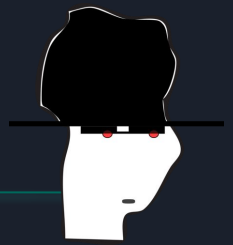
x86: Canary



```
11c3:       55                      push   ebp
11c4:       89 e5                   mov    ebp,esp
11c6:       51                      push   ecx
11c7:       83 ec 74                sub    esp,0×74
11ca:       89 c8                   mov    eax,ecx
11cc:       8b 40 04                mov    eax,DWORD PTR [eax+0×4]
11cf:       89 45 94                mov    DWORD PTR [ebp-0×6c],eax
11d2:       65 a1 14 00 00 00       mov    eax,gs:0×14
11d8:       89 45 f4                mov    DWORD PTR [ebp-0×c],eax
11db:       31 c0                   xor    eax,eax
```

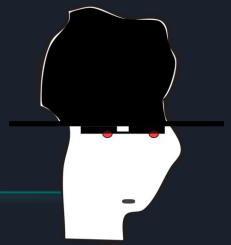
```
joe@zoidberg:~/Seg/Rev/bof$ ./01-challenge
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
*** stack smashing detected ***: <unknown> terminated
Aborted
```

x86: Inconvenientes



- **Sobrecarga**
- **Solo protegen contra Desbordes en la pila**
- **Podrían llegar a leakearse:**
  - **Format String**
  - **Otro**

x86: Tipos



- **Terminator Canaries (canary = CR, LF, 00, -1)**
- **Randomized Canaries (using /dev/random)**
- **Single XOR Canaries (xored return address)**
- **First Canary (1998)**
  Hardcoded 0xDEADBEEF

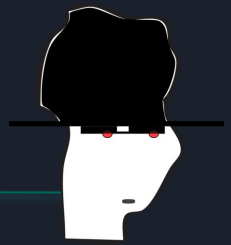
```
-fstack-protector-all
-fstack-protector

     +  char array of 8 bytes or
        more declared on the stack
     +  --param=ssp-buffer-size=N
```

x86: Bypass canaries (random)

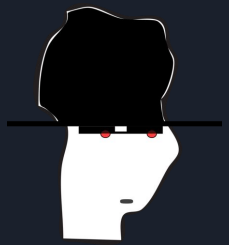
Sobreescribir:

- **usando bruteforce (asumiendo mismo canary)**
- **predecir (buscando debilidades en los PRG)**
- **sobreescribir datos no protegidos**
- **leyendo de la pila el valor correcto (format string)**

x86: Hoy en /usr/bin



```
Partial RELRO   No canary found   NX enabled    No PIE
16              /usr/bin/x86_64-linux-gnu-gcov-dump-8
RELRO           STACK CANARY      NX            PIE
Fortified       Fortifiable     FILE
No RELRO        No canary found   NX enabled    No PIE
11              /usr/bin/skipfish
RELRO           STACK CANARY      NX            PIE
Fortified       Fortifiable     FILE
Full RELRO      Canary found      NX enabled    PIE enabled
16              /usr/bin/nbackup
RELRO           STACK CANARY      NX            PIE
Fortified       Fortifiable     FILE
Partial RELRO   Canary found      NX enabled    PIE enabled
2               /usr/bin/kpsewhich
RELRO           STACK CANARY      NX            PIE
Fortified       Fortifiable     FILE
No RELRO        No canary found   NX enabled    No PIE
2               /usr/bin/rtpflood
RELRO           STACK CANARY      NX            PIE
```



“Preventing the introduction of malicious code is not enough to prevent the execution of malicious computations”

-Dino Dai Zovi