Code Vulnerabilities & Attacks: Modern Buffer Overflow Exploitations and Mitigations

Marcus Botacin

UFPR

2021

Topics

Topics

1. Recap
   - Last Class
2. Injections
   - Return to libc
   - Return Oriented Programming (ROP)
3. Mitigations
   - Safer Code
   - Compiler+OS Protections
   - Control Flow Protections
4. Conclusion
   - Demo
   - Exercises
   - Closing Remarks

Back to O.S (again!!)

Libs and permissions

```
marcus@malware-lab:~/Documentos/aula-ROP$ cat /proc/self/maps
00400000-0040c000 r-xp 00000000 08:05 11022740                           /bin/cat
0060b000-0060c000 r--p 0000b000 08:05 11022740                           /bin/cat
0060c000-0060d000 rw-p 0000c000 08:05 11022740                           /bin/cat
00d56000-00d77000 rw-p 00000000 00:00 0                                  [heap]
7f3411e47000-7f3412121000 r--p 00000000 08:05 9052298                    /usr/lib/locale/locale-archive
7f3412121000-7f34122e1000 r-xp 00000000 08:05 20189578                   /lib/x86_64-linux-gnu/libc-2.23.so
7f34122e1000-7f34124e1000 ---p 001c0000 08:05 20189578                   /lib/x86_64-linux-gnu/libc-2.23.so
7f34124e1000-7f34124e5000 r--p 001c0000 08:05 20189578                   /lib/x86_64-linux-gnu/libc-2.23.so
7f34124e5000-7f34124e7000 rw-p 001c4000 08:05 20189578                   /lib/x86_64-linux-gnu/libc-2.23.so
7f34124e7000-7f34124eb000 rw-p 00000000 00:00 0
7f34124eb000-7f3412511000 r-xp 00000000 08:05 20189562                   /lib/x86_64-linux-gnu/ld-2.23.so
7f34126ca000-7f34126ef000 rw-p 00000000 00:00 0
7f3412710000-7f3412711000 r--p 00025000 08:05 20189562                   /lib/x86_64-linux-gnu/ld-2.23.so
7f3412711000-7f3412712000 rw-p 00026000 08:05 20189562                   /lib/x86_64-linux-gnu/ld-2.23.so
7f3412712000-7f3412713000 rw-p 00000000 00:00 0
7ffea5db9000-7ffea5dda000 rw-p 00000000 00:00 0                          [stack]
```

Figure: Memory mapping and protection

Still overflowing

```
int main(int argc, char *argv[])
{
  int a = 10;
  char str[4];
  scanf("%s",str):
  printf("%d\n",a);
  return 0;
}
```

Code 1: variable overflow

Topics

The attack

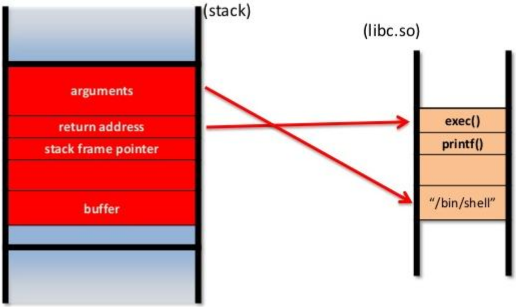


Figure: Return to libc

[0] https://tinyurl.com/yd8r36q4

Payload building

Finding strings

objdump -s ret2libc
(gdb) find 0x400000, 0x401000, "/bin"

Finding gadgets

ropper –file ret2libc –search "% ?di"

Ret2LibC Concepts

Definition

"Ataque de *Control Flow Hijacking* através da exploração de um *buffer overflow* tendo o endereço de funções e argumentos da `libc` como *payload*"

Topics

1. Recap
   - Last Class
2. Injections
   - Return to libc
   - Return Oriented Programming (ROP)
3. Mitigations
   - Safer Code
   - Compiler+OS Protections
   - Control Flow Protections
4. Conclusion
   - Demo
   - Exercises
   - Closing Remarks

The attack

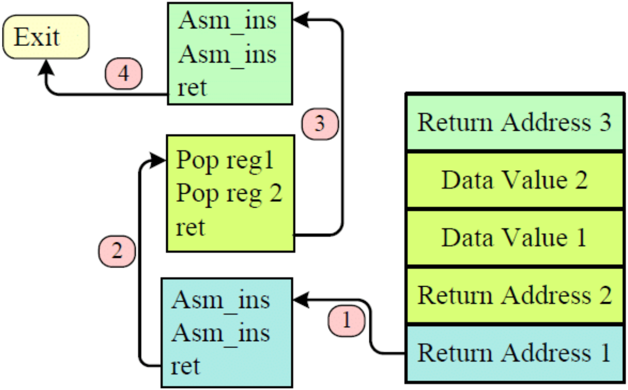


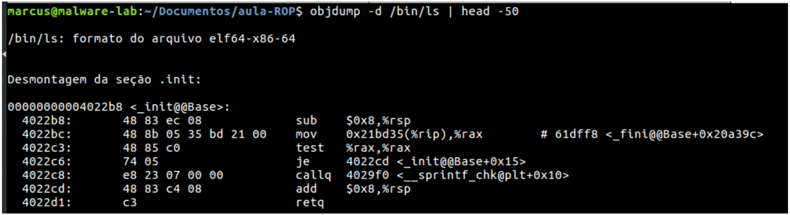
Figure: ROP gadget chaining

[0] https://tinyurl.com/y7p3hrkk

Questions

- What is the relation between ROP attacks and the x64 calling convention ?
- How to find gadgets ?

More background



```
marcus@malware-lab:~/Documentos/aula-ROP$ objdump -d /bin/ls | head -50

/bin/ls: formato do arquivo elf64-x86-64


Desmontagem da seção .init:

00000000004022b8 <_init@@Base>:
  4022b8:       48 83 ec 08             sub    $0x8,%rsp
  4022bc:       48 8b 05 35 bd 21 00    mov    0x21bd35(%rip),%rax        # 61dff8 <_fini@@Base+0x20a39c>
  4022c3:       48 85 c0                test   %rax,%rax
  4022c6:       74 05                   je     4022cd <_init@@Base+0x15>
  4022c8:       e8 23 07 00 00          callq  4029f0 <__sprintf_chk@plt+0x10>
  4022cd:       48 83 c4 08             add    $0x8,%rsp
  4022d1:       c3                      retq
```

Figure: Binary disassembly

Unaligned Instructions

```
c08: f2 0f 58 c3       addsd  %xmm3,%xmm0
c0c: 66 0f 13 44 24 04 movlpd %xmm0,0x4(%esp)
```

Code 2: Static disassembly of the `MSVCR71.dll` library.

```
c0a: 58 pop rax
c0b: c3 ret
```

Code 3: ROP Gadget.

ROP Concepts

Gadgets

"Sequência independente de instruções terminadas por `RET`"

Gadget Chain

"Encadeamento de *gadgets* com o objetivo de realizar uma computação."

ROP Attack

"Ataque de *control flow hijacking* através da exploração de um *buffer overlow* tendo um *gadget chain* como parte do *payload*."

Topics

Safe functions/languages

```
char* strcpy(char* dst, char* src);
```

Code 4: Popular strcpy prototype

```
char* strncpy(char* det, char* src, size_t num);
```

Code 5: Not so popular strncpy prototype

Safe functions/languages

```
typedef struct str{
  char str[MAX];
  uint size;
  uint max=MAX;
}string;
```

Code 6: String definition example

```
concat(str1,str2)
  if str1->size + str2->size < str1->max
```

Code 7: Concat implementation example

Safe String Functions

Definition

"Função que verifica os tamanhos dos *buffers* a fim de evitar um *overflow*."

Topics

1. Recap
   - Last Class
2. Injections
   - Return to libc
   - Return Oriented Programming (ROP)
3. Mitigations
   - Safer Code
   - Compiler+OS Protections
   - Control Flow Protections
4. Conclusion
   - Demo
   - Exercises
   - Closing Remarks

Stack Canaries

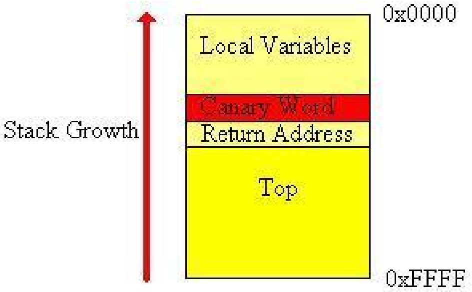


Figure: Stack canary

0 https://tinyurl.com/yd9947ol

0 Disable with: `gcc -fno-stack-protector`

Stack Canaries

Definition

"Marcador da continência de um *buffer*."

Question

Compile-time solutions

- Advantages ?
- Disadvantages ?

ASLR

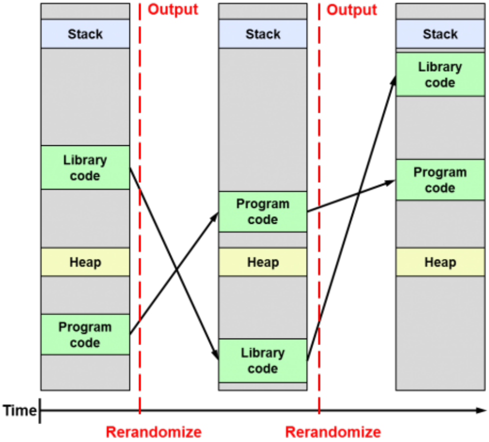


Figure: Address Space Layout Randomization (ASLR)

ASLR

Table: ASLR - library placement after two consecutive reboots.

library	ntdll	kernel32	kernelbase
address 1	0xbaf80000	0xb9610000	0xb8190000
address 2	0x987b0000	0x98670000	0x958c0000

ASLR

Definition

"Aleatorização do posicionamento de conteúdos de memória a fim de evitar ataques de *offset* fixo."

ASLR

Position Independent Code

- GCC `-fpic`

Jump Tables

```
00000000004022e0 <__ctype_toupper_loc@plt-0x10>:
  4022e0:       ff 35 22 bd 21 00       pushq  0x21bd22(%rip)        # 61e008 <_fini@@Base+0x20a3ac>
  4022e6:       ff 25 24 bd 21 00       jmpq   *0x21bd24(%rip)        # 61e010 <_fini@@Base+0x20a3b4>
  4022ec:       0f 1f 40 00             nopl   0x0(%rax)

00000000004022f0 <__ctype_toupper_loc@plt>:
  4022f0:       ff 25 22 bd 21 00       jmpq   *0x21bd22(%rip)        # 61e018 <_fini@@Base+0x20a3bc>
  4022f6:       68 00 00 00 00          pushq  $0x0
  4022fb:       e9 e0 ff ff ff          jmpq   4022e0 <_init@@Base+0x28>

0000000000402300 <__uflow@plt>:
  402300:       ff 25 1a bd 21 00       jmpq   *0x21bd1a(%rip)        # 61e020 <_fini@@Base+0x20a3c4>
  402306:       68 01 00 00 00          pushq  $0x1
  40230b:       e9 d0 ff ff ff          jmpq   4022e0 <_init@@Base+0x28>

0000000000402310 <getenv@plt>:
  402310:       ff 25 12 bd 21 00       jmpq   *0x21bd12(%rip)        # 61e028 <_fini@@Base+0x20a3cc>
  402316:       68 02 00 00 00          pushq  $0x2
  40231b:       e9 c0 ff ff ff          jmpq   4022e0 <_init@@Base+0x28>
```

Figure: Jump Table

ASLR - Really a solution?

Breaking Kernel Address Space Layout Randomization with Intel TSX

Yeongjin Jang, Sangho Lee, and Taesoo Kim
Georgia Institute of Technology

Figure: **Source:** `https://dl.acm.org/doi/10.1145/2976749.2978321`

CAIN: Silently Breaking ASLR in the Cloud

Antonio Barresi, *ETH Zurich*
Kaveh Razavi, *VU University Amsterdam*
Mathias Payer, *Purdue University*
Thomas R. Gross, *ETH Zurich*

Figure: **Source:** `https://www.usenix.org/conference/woot15/workshop-program/presentation/barresi`

Topics

Write XOR Execute

Write XOR Execute Policy

Write Xor Execute

- How to implement ?

```
cat /proc/cpuinfo
flags: fpu vme de pse tsc msr pae mce apic nx
```

Code 8: /proc/cpuinfo

Write XOR Execute Policy

Definition

"Política que assegura que páginas de memória executáveis não podem ser escritas."

Control Flow Integrity (CFI)

Example

```c
int main(int argc, char *argv[])
{
    printf("Hello\n");
    vulnerable_function(argv);
    return 0;
}
```

Code 9: Vulnerable function.

Control Flow Integrity (CFI)

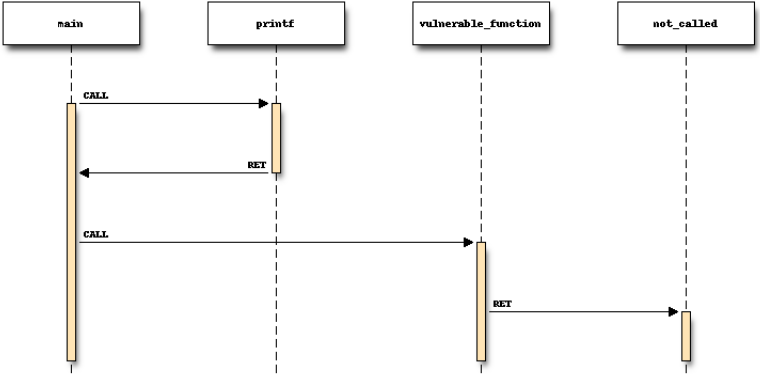


Figure: CALL-RET policy

Shadow Stacks

LBR Stack

	Branch	Target	
00	73802745	738028D7	
01	05F015CA	05F00E17	
02	7C348B06	7C34A028	G01
03	7C34A02A	7C34252C	G02
04	7C34252D	7C36C55A	G03
05	7C36C55B	7C345249	G04
06	7C34524A	7C3411C0	G05
07	7C3411C1	7C34B8D7	G06
08	7C34B8D8	7C366FA6	G07
09	7C366FA7	7C3762FB	G08
10	7C3762FC	7C378C81	G09
11	7C378C84	7C346C0B	G10
12	7C346C0B	7C3415A2	G11
13	7C3415A2	74F64347	
14	74F64908	752AD0A1	
15	752D6FC8	752AD0AD	

Figure: Branch Stack. Source: `http://www.cs.columbia.edu/~vpappas/papers/kbouncer.sec13.pdf`

Shadow Stacks Implementations

"A Technical Look at Intel's Control-flow Enforcement Technology"

https://software.intel.com/content/www/us/en/develop/articles/technical-look-control-flow-enforcement-technology.html

"Understanding Hardware-enforced Stack Protection"

https://techcommunity.microsoft.com/t5/windows-kernel-internals/understanding-hardware-enforced-stack-protection/ba-p/1247815

"Enabling Hardware-enforced Stack Protection in Chrome"

https://security.googleblog.com/2021/05/enabling-hardware-enforced-stack.html

CFI - Really a solution?

Control Jujutsu:
On the Weaknesses of Fine-Grained Control Flow Integrity*

Isaac Evans
MIT Lincoln Laboratory
ine@mit.edu

Fan Long
MIT CSAIL
fanl@csail.mit.edu

Ulziibayar Otgonbaatar
MIT CSAIL
ulziibay@csail.mit.edu

Howard Shrobe
MIT CSAIL
hes@csail.mit.edu

Martin Rinard
MIT CSAIL
rinard@csail.mit.edu

Hamed Okhravi
MIT Lincoln Laboratory
hamed.okhravi@ll.mit.edu

Stelios Sidiroglou-Douskos
MIT CSAIL
stelios@csail.mit.edu

Figure: **Source:** `https://dl.acm.org/doi/10.1145/2810103.2813646`

CFI

CFI Definition

“Imposição de que o fluxo de execução siga uma determinada política.”

CALL-RET Definition

“Política que exige que todas as instruções `RET` sejam precedidas por um `CALL`.”

What to do when you don't know what to do?

Heuristics

Questions

- What *gadgets'* features?
- Limitations?

Heuristic Example

- Block very short code portions (*gadgets*)

Gadget Size Heuristic - solution ?

Size Does Matter: Why Using Gadget-Chain Length to Prevent Code-Reuse Attacks is Hard

Enes Göktaş, *Vrije Universiteit Amsterdam;* Elias Athanasopoulos, *FORTH-ICS;*
Michalis Polychronakis, *Columbia University;* Herbert Bos, *Vrije Universiteit Amsterdam;*
Georgios Portokalidis, *Stevens Institute of Technology*

Figure: **Source:** `https://www.usenix.org/biblio/size-does-matter-why-using-gadget-chain-length-prevent-code-reu`

Topics

Demonstration

- `https://www.youtube.com/watch?v=svM_6TN7kSE`

Topics

Challenge

Can you understand the ROP payload?

Topics

Summary

- One day, it was easy.
- Now, many protections.
  - Modern exploitation is usually a chain of exploits.
- Harder, but still possible!

Conclusion

Questions?

mfbotacin@inf.ufpr.br