Lecture 3
Data Execution Prevention

MICS - 2019

Dr. Alexandre Bartel
www.abartel.net

Previously...

Lecture 1

- Software Development
- Software Security
- Software Vulnerability

Lecture 1: Software Development

- Tukey, 1958
- Functions, compilers, documentation (vs. hardware)
- Life-cycle
  - idea, requirements, design, implementation, deployment
- Approaches: Waterfall, Agile
- Goals: less risk, better quality

Lecture 1: Software Security

- Security policy
- Software system tries to maintain the following attributes in accordance with the security policy:
  - C…
  - I…
  - A…
- How? With security mechanisms
  - Access control
  - Sandbox

Lecture 1: Software Vulnerability

- Life cycle:
  - Birth, discovery, disclosure, correction, publicity, scripting, death
- Non-disclosure, full disclosure, responsible disclosure
- CVE number, MITRE

Lecture 2: Buffer Overflow on the Stack

- Buffer: consecutive bytes in memory
- Local buffer: stored on the stack
- Function f1 calls f2 at instruction i: return @ of instruction i + 1 on the stack
- Buffer overflow overwrites return @
- Attacker puts shellcode in buffer jumps to it

“Introduction to Software Security”_Course Plan

2. Memory Attacks and Defenses
   - ➔ Buffer overflow
   - ➔ Heap overflow
   - ➔ Integer overflow
   - ➔ String format vulnerabilities
   - ➔ Type confusion
   - ➔ Use After Free

Preventing
Buffer Overflow Attacks
On the Stack

Attack Prevention: First Idea

- Gently ask developers to check bounds!
- Does not work:
  - Ex: Intel [1], AMD [2]
- Problem 1: programmers might do it… or NOT
- Problem 2: does not protect legacy software

[1] Ermolov, Mark, and Maxim Goryachy. "How to Hack a Turned-Off Computer, or Running Unsigned Code in Intel Management Engine." Black Hat Europe (2017).
[2] https://www.bleepingcomputer.com/news/security/security-flaw-in-amds-secure-chip-on-chip-processor-disclosed-online/ (6 January 2018)

Attack Prevention: Second Idea

- Mark the stack as NON-executable
  - Called Data Execution Prevention (DEP)
  - AKA Non-eXecute bit (NX bit)
- The attacker can still put the shellcode in the buffer
- But jumping to it triggers a segmentation fault

Problem Solved?

Problem Solved?

- Attacker can still “jump” anywhere he/she likes
- Attacker could put the shellcode at some executable page in memory (ex: JIT) and jump to it.
  - Problem: where is the address of the page?

Problem Solved?

- Attacker can still “jump” anywhere he/she likes
- Attacker could execute small code snippets ending in “ret”
- Introducing “gadgets”
- Introducing “ROP” (Return Oriented Programming)

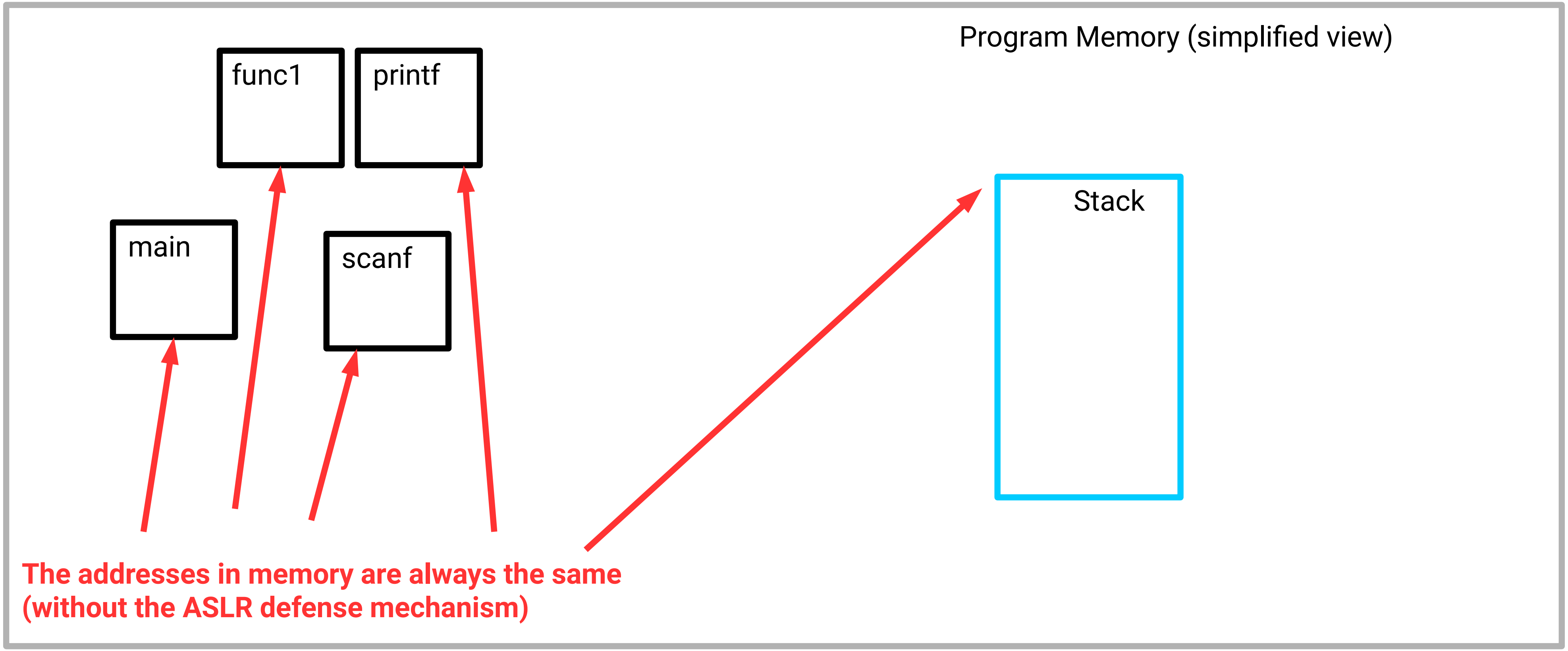
ROP Gadgets

- Attacker wants to execute program P
- P features N instructions
- Recipe (simplified):
  - For instruction “i” in program “P”:
    - Find ROP gadget “Gi” executing “i”
  - Chain all ROP gadgets together with data (this as called a ROP chain)
  - Execute program “P” through ROP gadgets

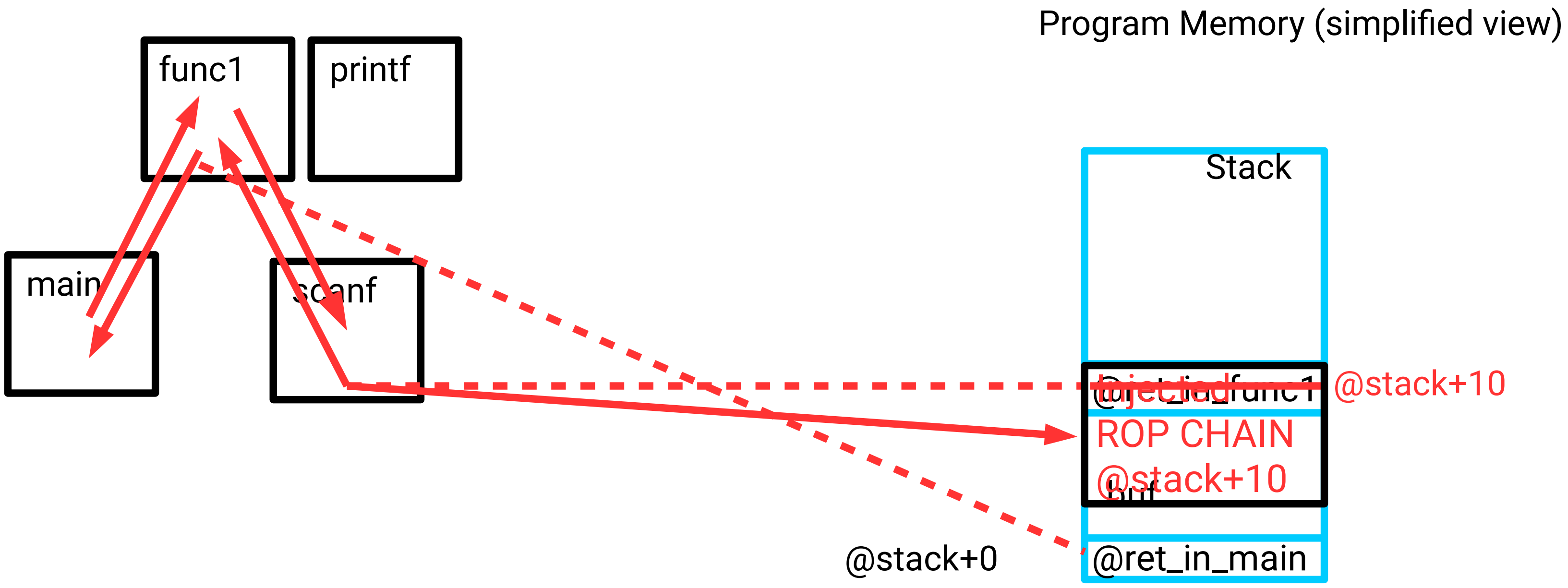
Why ROP works

- Code is always loaded at the same address
- Only data is pushed to the stack
- Code is “reused”
- No code is injected

Custom Code Execution



Custom Code Execution



The control flow of the program can be changed by exploiting the buffer overflow vulnerability to change the value of @ret_in_func1!

Where are the Gadgets?

- Gadgets are in the code segments (program itself or linked libraries) loaded in the virtual memory of the process

```
$ cat /proc/self/maps
558a5d145000-558a5d147000 r--p 00000000 fe:01 524608                     /bin/cat
558a5d147000-558a5d14c000 r-xp 00002000 fe:01 524608                     /bin/cat
558a5d14c000-558a5d14e000 r--p 00007000 fe:01 524608                     /bin/cat
558a5d14f000-558a5d150000 r--p 00009000 fe:01 524608                     /bin/cat
558a5d150000-558a5d151000 rw-p 0000a000 fe:01 524608                     /bin/cat
558a5ec90000-558a5ecb1000 rw-p 00000000 00:00 0                          [heap]
7f603660d000-7f60368f2000 r--p 00000000 fe:01 278299                     /usr/lib/locale/locale-archive
7f60368f2000-7f6036914000 r--p 00000000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036914000-7f6036a5c000 r-xp 00022000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036a5c000-7f6036aa8000 r--p 0016a000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036aa8000-7f6036aa9000 ---p 001b6000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036aa9000-7f6036aad000 r--p 001b6000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036aad000-7f6036aaf000 rw-p 001ba000 fe:01 403316                     /lib/x86_64-linux-gnu/libc-2.28.so
7f6036aaf000-7f6036ab3000 rw-p 00000000 00:00 0
7f6036ab3000-7f6036ab5000 rw-p 00000000 00:00 0
7f6036aed000-7f6036b0f000 rw-p 00000000 00:00 0
7f6036b0f000-7f6036b10000 r--p 00000000 fe:01 394494                     /lib/x86_64-linux-gnu/ld-2.28.so
7f6036b10000-7f6036b2e000 r-xp 00001000 fe:01 394494                     /lib/x86_64-linux-gnu/ld-2.28.so
7f6036b2e000-7f6036b36000 r--p 0001f000 fe:01 394494                     /lib/x86_64-linux-gnu/ld-2.28.so
7f6036b36000-7f6036b37000 r--p 00026000 fe:01 394494                     /lib/x86_64-linux-gnu/ld-2.28.so
7f6036b37000-7f6036b38000 rw-p 00027000 fe:01 394494                     /lib/x86_64-linux-gnu/ld-2.28.so
7f6036b38000-7f6036b39000 rw-p 00000000 00:00 0
7ffd20e58000-7ffd20e79000 rw-p 00000000 00:00 0                          [stack]
7ffd20f38000-7ffd20f3b000 r--p 00000000 00:00 0                          [vvar]
7ffd20f3b000-7ffd20f3d000 r-xp 00000000 00:00 0                          [vdso]
```

Custom Code Execution

If the analyst inputs the following data:

‘f’	‘o’	@g3	‘d2	@g2	d1	@g1	i0	i1	i2	...

The pop instruction get a value from the *stack*. Thus, after every pop instruction, the stack pointer, rsp, is moved to the next stack memory location.

@gadget1
Pop rax
ret

@gadget2
Pop rcx
ret

@gadget3
Syscall
...

This technique is Used to initialize Registers with Attacker controlled values

Memory @	Value
	...
	...
...	...
@stack + 0x08	...
@stack + 0x07	
@stack + 0x06	
@stack + 0x05	
@stack + 0x04	
@stack + 0x03	
@stack + 0x02	‘o’
@stack + 0x01	‘f’
@stack + 0x00	@code + 0x01

stack grows this way

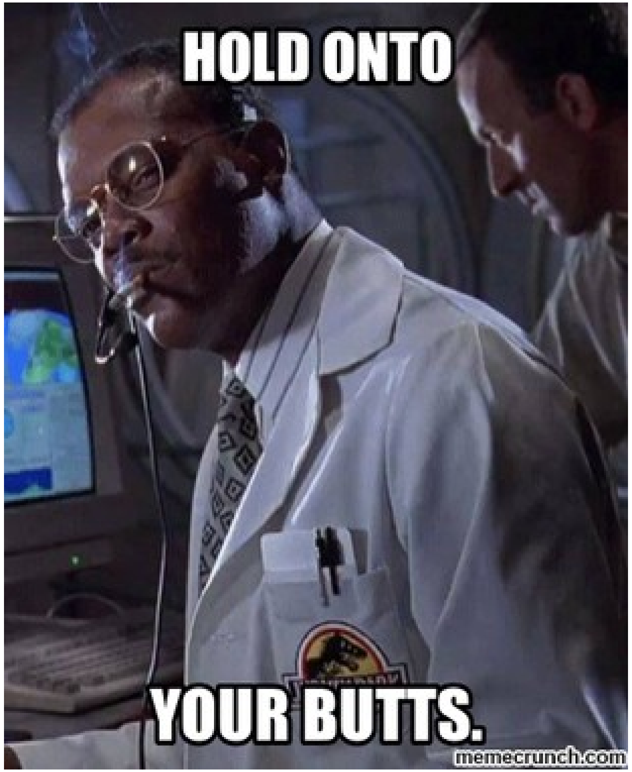
Finding Gadgets

‘58’ is the machine
Representation
Of “pop rax”.

```
00000000000e76a0 <__getauxval@@GLIBC_2.16>:
[...]
   e76f8: 48 8b 40 58             mov    0x58(%rax),%rax
   e76fc: c3                      retq
   e76fd: 0f 1f 00                nopl   (%rax)
   e7700: 48 8b 80 10 01 00 00    mov    0x110(%rax),%rax
   e7707: c3                      retq
   e7708: 0f 1f 84 00 00 00 00 00 nopl   0x0(%rax,%rax,1)
```

The gadget starts at address 0xe76fa, ends at address 0xe76fc and contains 2 instructions ‘58’ (pop rax) and ‘c3’ (ret). The code is extracted from libc (and yes, it is possible to jump in the middle of “normal” instructions to execute “new” instructions)

Example



Consequences

1. The analyst can still put data on the stack!
   - Data
   - Addresses to gadgets
2. The injected gadgets
   - Are called one after the other (they end in ‘ret’)
   - Can put data in registers (with ‘pop’ instructions)
   - Can call a syscall (with the ‘syscall’ instruction)
   - “Simulate” the execution of a shellcode

Defenses

- Two main approaches to limit exploitation of a buffer overflow
  - Stack canary (aka stack cookie)
  - Data Execution Prevention (DEP)
- Bypassing DEP
  - ROP attack with gadget chain
- Two main approaches to prevent ROP attacks bypassing DEP:
  - Address Space Layout Randomization (ASLR)
  - Control Flow Integrity (CFI)

ROP Attacks: A bit of History

- 1997: return into-libc [1]
- 2004: attack techniques [2]
- 2007: academic description of gadgets [3]

[1] Getting around non-executable stack (and fix) - Solar Designer, 1997
[2] Pincus, Jonathan, and Brandon Baker. "Beyond stack smashing: Recent advances in exploiting buffer overruns." IEEE Security & Privacy 2.4 (2004): 20-27.
[3] Shacham, Hovav. "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)." ACM conference on Computer and communications security. 2007.

Questions?

- Of course you have some questions