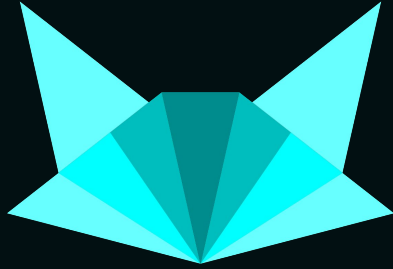


Intro to Reverse Engineering

Anirudh Oppiliappan

Cyware 2019

Self-promotion



- icyphox
- 6-ish years of messing with computers
- Open source <3
 - The Nim programming language
 - Packages and tools
- Offsec, RE and forensics

What even?

The process of taking
something apart to
better understand how it
works internally.

RE in security

- Reversing binaries
 - Static analysis
 - Dynamic analysis
- Cracking software
- Malware analysis
- Binary exploitation
- Firmware analysis
- DFIR

That's all cool, but what
are we doing today?

Agenda

- Tools of the trade
- Basic x86 assembly
- An overview of ELF
- Solving a couple of “crackmes”
- Q & A

“All that a good reverse engineer needs is a disassembler and a debugger.”

– Anonymous *(Anirudh probably)*

Tools!

Disassemblers

- IDA Pro \$\$\$
- Binary Ninja \$\$
- radare2 FOSS
- objdump(1) FOSS

```
x
r2 bin.elf

0x000007b2 bf0a000000 mov edi, 0xa ; size_t size
0x000007b7 e8c4feffff call sym.imp.malloc ; void *malloc(size_t size)
0x000007bc 488945f0 mov qword [s2], rax
0x000007c0 488b45f0 mov rax, qword [s2]
0x000007c4 c60061 mov byte [rax], 0x61 ; 'a' ; [0x61:1]=1
0x000007c7 c745ec010000. mov dword [local_14h], 1
,=< 0x000007ce eb2b jmp 0x7fb
--> 0x000007d0 8b45ec mov eax, dword [local_14h]
:| 0x000007d3 4898 cdqe
:| 0x000007d5 488d50ff lea rdx, [rax - 1]
:| 0x000007d9 488b45f0 mov rax, qword [s2]
:| 0x000007dd 4801d0 add rax, rdx ; '('
:| 0x000007e0 0fb600 movzx eax, byte [rax]
:| 0x000007e3 8d4801 lea ecx, [rax + 1]
:| 0x000007e6 8b45ec mov eax, dword [local_14h]
:| 0x000007e9 4863d0 movsxd rdx, eax
:| 0x000007ec 488b45f0 mov rax, qword [s2]
:| 0x000007f0 4801d0 add rax, rdx ; '('
:| 0x000007f3 89ca mov edx, ecx
:| 0x000007f5 8810 mov byte [rax], dl
:| 0x000007f7 8345ec01 add dword [local_14h], 1
:| ; CODE XREF from main (0x7ce)
--> 0x000007fb 837dec08 cmp dword [local_14h], 8
==< 0x000007ff 7ecf jle 0x7d0
0x00000801 488b45f0 mov rax, qword [s2]
0x00000805 4883c009 add rax, 9
0x00000809 c60000 mov byte [rax], 0
0x0000080c bf0a000000 mov edi, 0xa ; size_t size
0x00000811 e86afeffff call sym.imp.malloc ; void *malloc(size_t size)
0x00000816 488945f8 mov qword [s1], rax
```

Debuggers

- gdb(1)
- OllyDbg
- x64dbg
- Other gdb frontends

```
x      gdb bin.elf
Dump of assembler code for function main:
0x0000000000007aa <+0>:      push    rbp
0x0000000000007ab <+1>:      mov     rbp, rsp
0x0000000000007ae <+4>:      sub     rsp, 0x20
0x0000000000007b2 <+8>:      mov     edi, 0xa
0x0000000000007b7 <+13>:     call   0x680 <malloc@plt>
0x0000000000007bc <+18>:     mov     QWORD PTR [rbp-0x10], rax
0x0000000000007c0 <+22>:     mov     rax, QWORD PTR [rbp-0x10]
0x0000000000007c4 <+26>:     mov     BYTE PTR [rax], 0x61
0x0000000000007c7 <+29>:     mov     DWORD PTR [rbp-0x14], 0x1
0x0000000000007ce <+36>:     jmp     0x7fb <main+81>
0x0000000000007d0 <+38>:     mov     eax, DWORD PTR [rbp-0x14]
0x0000000000007d3 <+41>:     cdq     eax
0x0000000000007d5 <+43>:     lea     rdx, [rax-0x1]
0x0000000000007d9 <+47>:     mov     rax, QWORD PTR [rbp-0x10]
0x0000000000007dd <+51>:     add     rax, rdx
0x0000000000007e0 <+54>:     movzx   eax, BYTE PTR [rax]
0x0000000000007e3 <+57>:     lea     ecx, [rax+0x1]
0x0000000000007e6 <+60>:     mov     eax, DWORD PTR [rbp-0x14]
0x0000000000007e9 <+63>:     movsxd  rdx, eax
0x0000000000007ec <+66>:     mov     rax, QWORD PTR [rbp-0x10]
0x0000000000007f0 <+70>:     add     rax, rdx
0x0000000000007f3 <+73>:     mov     edx, ecx
---Type <return> to continue, or q <return> to quit---
```

Basics of x86 assembly

000000000000007aa <main>:

7aa:	55	push	rbp
7ab:	48 89 e5	mov	rbp, rsp
7ae:	48 83 ec 20	sub	rsp, 0x20
7b2:	bf 0a 00 00 00	mov	edi, 0xa
7b7:	e8 c4 fe ff ff	call	680 <malloc@plt>
7bc:	48 89 45 f0	mov	QWORD PTR [rbp-0x10], rax
7c0:	48 8b 45 f0	mov	rax, QWORD PTR [rbp-0x10]
7c4:	c6 00 61	mov	BYTE PTR [rax], 0x61
7c7:	c7 45 ec 01 00 00 00	mov	DWORD PTR [rbp-0x14], 0x1
7ce:	eb 2b	jmp	7fb <main+0x51>
7d0:	8b 45 ec	mov	eax, DWORD PTR [rbp-0x14]
7d3:	48 98	cdqe	
7d5:	48 8d 50 ff	lea	rdx, [rax-0x1]
7d9:	48 8b 45 f0	mov	rax, QWORD PTR [rbp-0x10]
7dd:	48 01 d0	add	rax, rdx
7e0:	0f b6 00	movzx	eax, BYTE PTR [rax]
7e3:	8d 48 01	lea	ecx, [rax+0x1]
7e6:	8b 45 ec	mov	eax, DWORD PTR [rbp-0x14]
7e9:	48 63 d0	movsxd	rdx, eax
7ec:	48 8b 45 f0	mov	rax, QWORD PTR [rbp-0x10]
7f0:	48 01 d0	add	rax, rdx
7f3:	89 ca	mov	edx, ecx
7f5:	88 10	mov	BYTE PTR [rax], dl
7f7:	83 45 ec 01	add	DWORD PTR [rbp-0x14], 0x1
7fb:	83 7d ec 08	cmp	DWORD PTR [rbp-0x14], 0x8
7ff:	7e cf	jle	7d0 <main+0x26>
801:	48 8b 45 f0	mov	rax, QWORD PTR [rbp-0x10]

Registers

- A special, high-speed storage area in the processor
- General Purpose:
 - 16-bit ax, bx, cx, dx
 - 32-bit eax, ebx, ecx, edx
 - 64-bit rax, rbx, rcx, rdx
- Pointer Registers:
 - *SP, *BP, *IP
- Index Registers:
 - *SI, *DI
- Flags

Instructions

AT&T (src, dst)

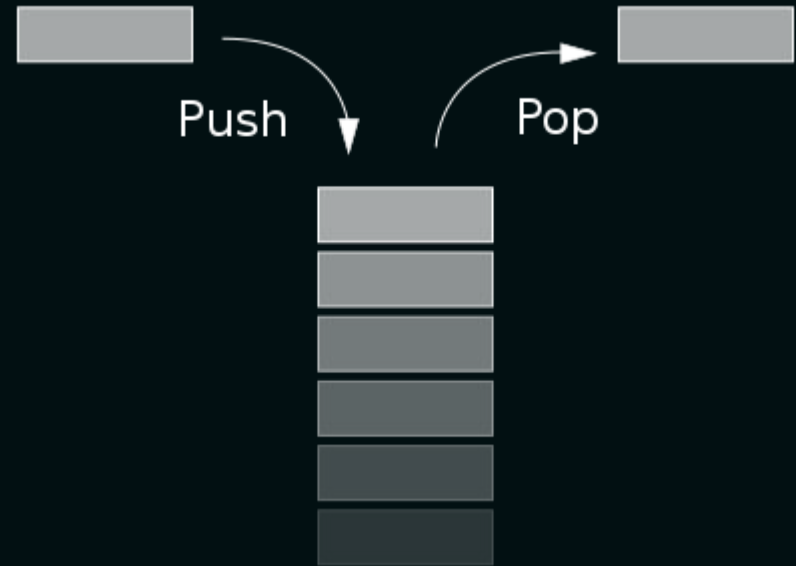
- push/pop %rbp
- mov \$0xa, %edi
- add/sub \$0xa, %rax
- call 0x680
- jle/jne/je/jmp 0x7fa
- lea 0xe3(%rip), %rdi
- ret

Intel (dst, src)

- push/pop rbp
- mov edi, 0xa
- add/sub rax, 0xa
- call 0x680
- jle/jne/je/jmp 0x7fa
- lea rdi, [rip+0xe3]
- ret

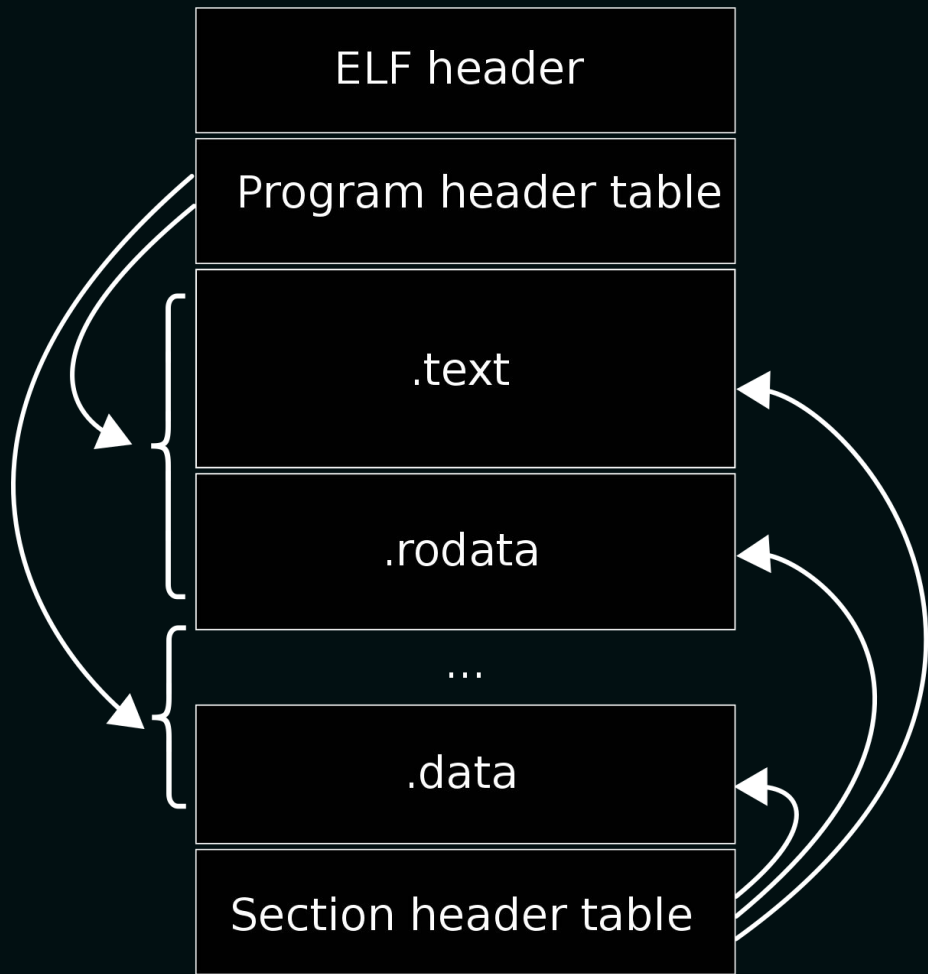
The Stack

- An area in memory that's used during program execution
- Follows the "stack" data structure
- Last In First Out (LIFO)



An overview of ELF

- Executable and Linkable Format
- Memory mapped
- Different sections:
 - .text - program instructions
 - .data - initialized data
 - .bss - uninitialized data
 - .rodata - read-only data
 - .got
 - .plt



Setup

- <https://github.com/icyphox/cyware19>
- ``sudo apt install build-essential``
- Check if you have ``gcc`` and ``gdb`` installed
- ``cd`` to the cloned directory

```
$ ./make.sh
```

Exploring ELF

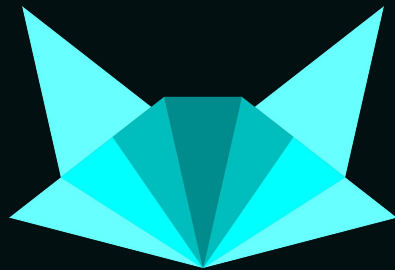
crackme01 – Static Analysis

crackme02 – Dynamic Analysis

Q & A

My links

- GitHub: <https://github.com/icyphox>
- Twitter: <https://twitter.com/icyphox>
- Email: icyph0x@pm.me
- Website: <https://icyphox.sh>



Extra Time – crackme03
