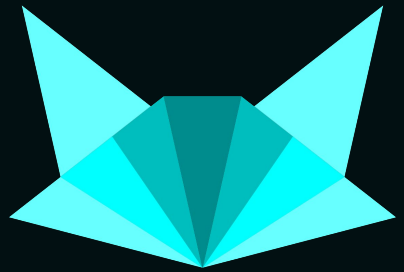
Intro to Reverse Engineering

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

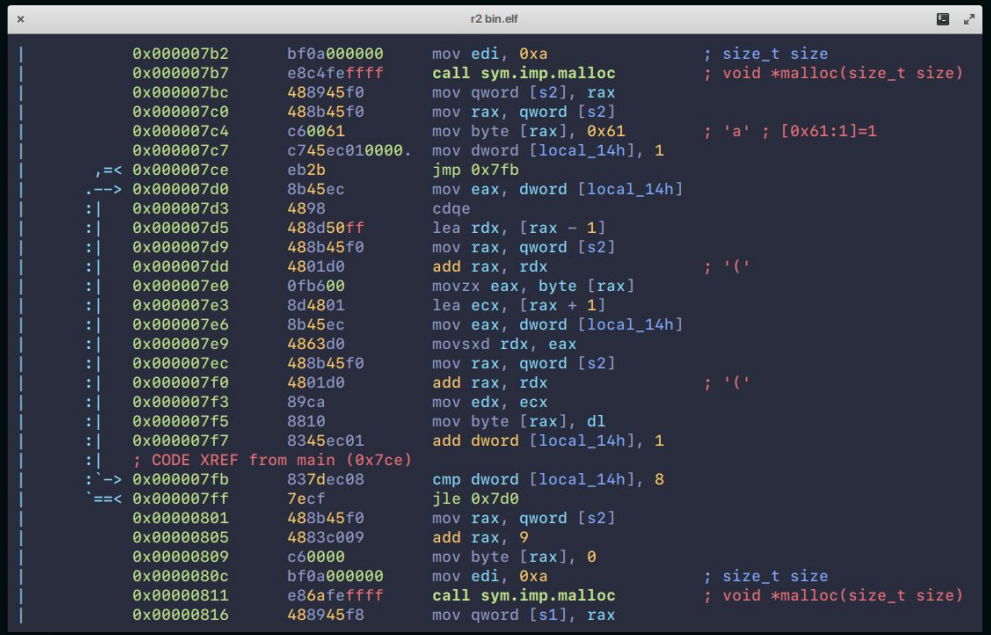
“All that a good reverse engineer needs is a disassembler and a debugger.”

— Anonymous *(Anirudh probably)*

Tools!

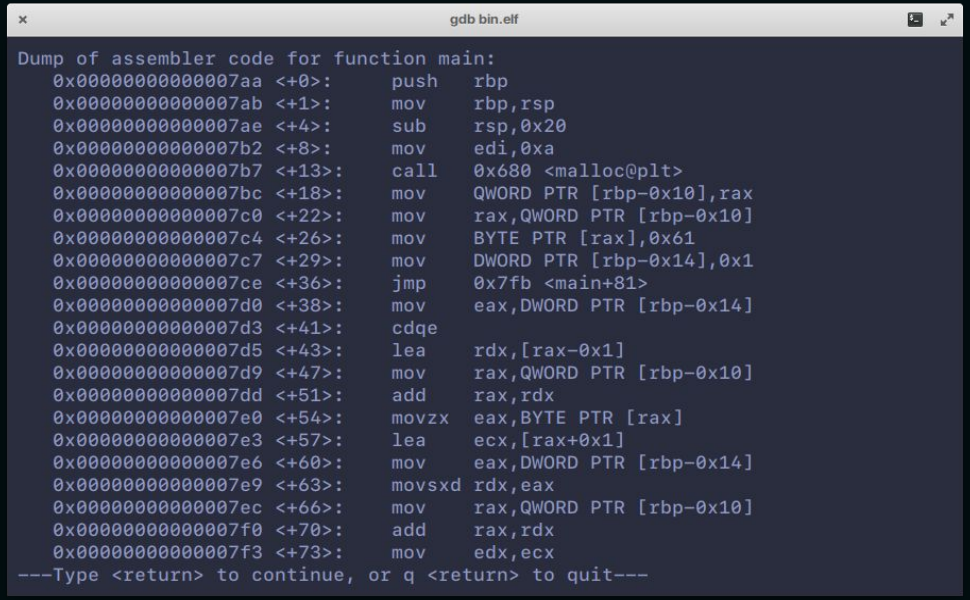
Disassemblers

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



Debuggers

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



Basics of x86 assembly

```
00000000000007aa <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)

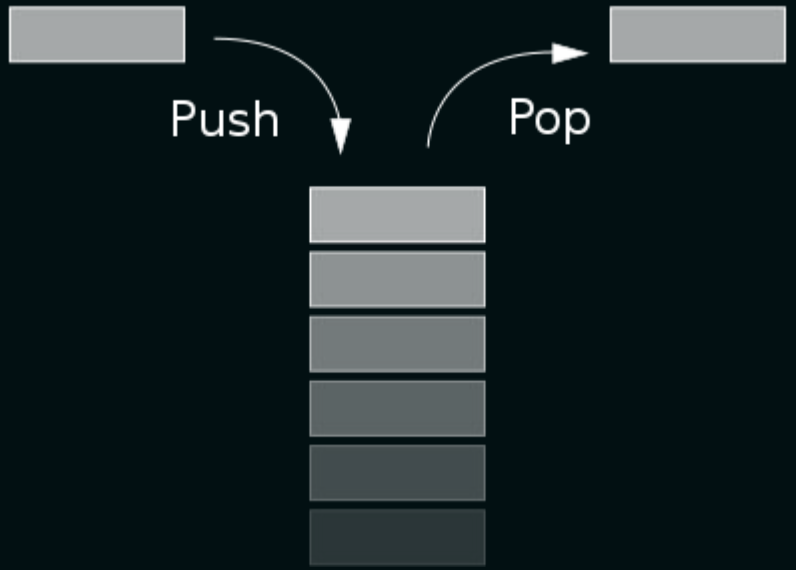
- push/pop %rbp
- mov $0xa, %edi
- add/sub $0xa, %rax
- call 0x680
- jle/jne/je/jmp 0x7fa
- lea 0xe3(%rip), %rdi
- ret

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

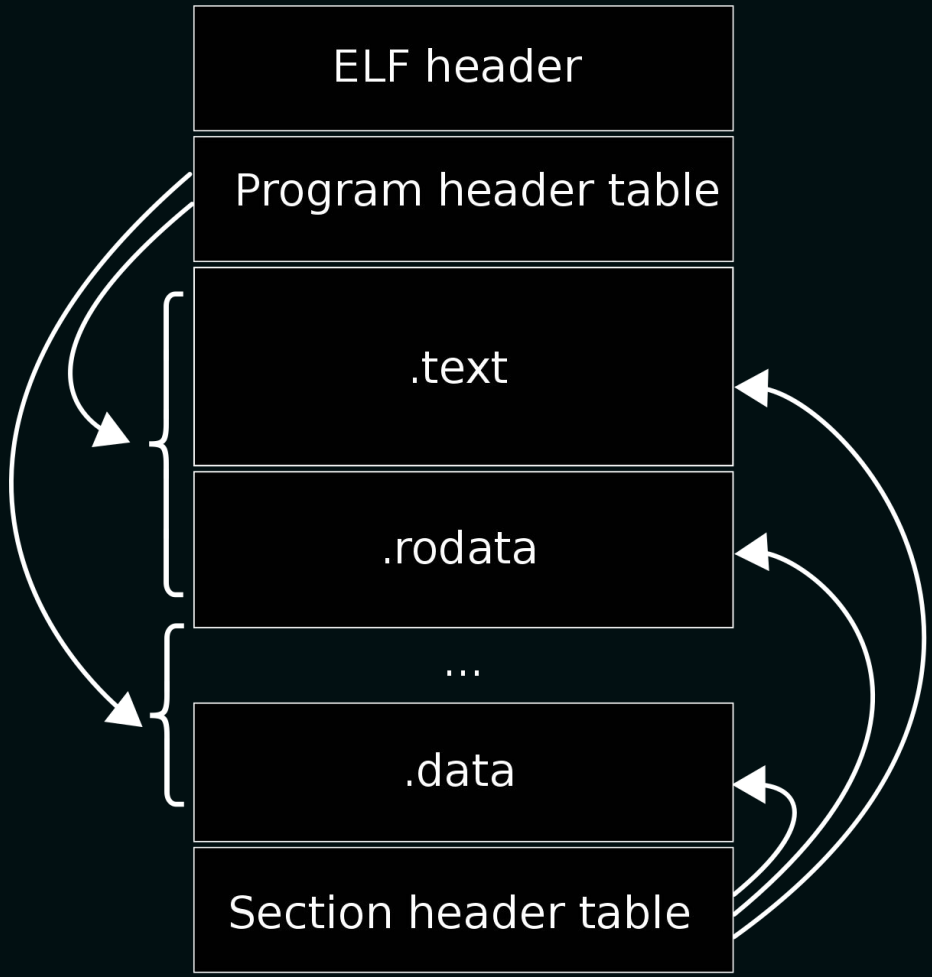
The Stack

- 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

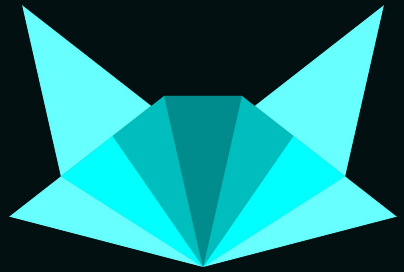
crackme01 — Static Analysis

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