Reverse Engineering 101

...in **20** minutes

Who even am I?

Morgan Whitlow

- Currently:
  - Embedded firmware reverse engineer
  - Hardware enthusiast
  - Sarcastic smartass
- Education:
  - Master of Science in Applied Computer Science
  - B.S. Biology, B.A. Psychology
- Formerly:
  - SOC analyst
  - Lockpicking instructor
  - Nanomaterials researcher
  - Various other stuff

Overview

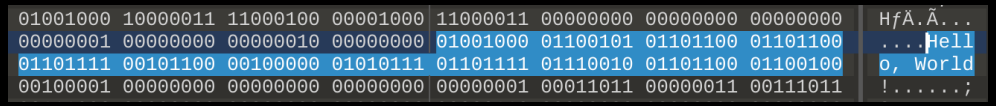
- Human vs machine code
- Layers of Abstraction
- What is reverse engineering?
- Static & Dynamic
- IDA vs Ghidra
- Niches & Specialties

Who speaks what?

- Binary
  - Mother tongue of machinery. Number system consisting literally of 1’s and 0’s
- Hexadecimal (‘hex’)
  - Base 16 number system (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F). Easier to read than binary, useful for quickly assessing specific characters or ‘op(eration) codes’
- Assembly language
  - Hex translated into mnemonics. Lowest level ‘human readable’ programming language. Says what specific instructions the processor is doing step-by-step. Diff kind of processor? Might be a diff assembly language.

Language Levels

- Machine Code
  - Binary



  - Represented as hex:

- Low Level Languages
  - Assembly

X86 (Computers)

```
.LC0:
        .string "Hello, World!"
main:
        push    rbp
        mov     rbp, rsp
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        mov     eax, 0
        pop     rbp
        ret
```

ARM (Embedded)

```
.LC0:
        .ascii  "Hello, World!\000"
main:
        push    {fp, lr}
        add     fp, sp, #4
        ldr     r0, .L3
        bl      printf
        mov     r3, #0
        mov     r0, r3
        pop     {fp, pc}
.L3:
        .word   .LC0
```

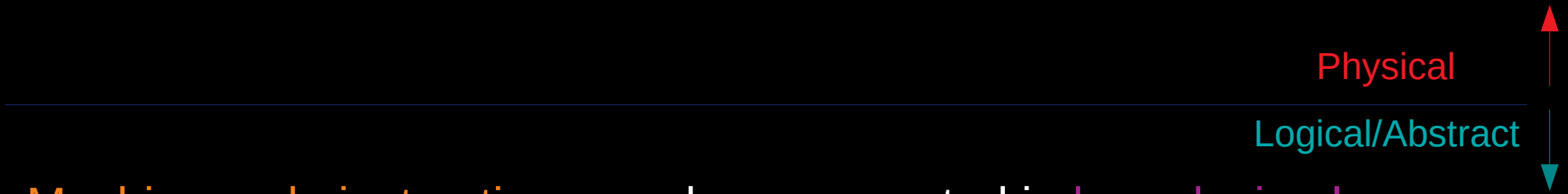
- High Level Languages
  - C, C++

Source code in C

```c
#include <stdio.h>
int main() {
  printf("Hello, World!");
  return 0;
}
```

Physical to Abstract

- Memory (RAM) matrices of + or – charged ‘cells’ ← 1’s and 0’s (binary)
- Processor reads the + and – cells like a map or instruction booklet ← 1’s and 0’s as data or instructions



- Machine code instructions can be represented in hexadecimal
- Hexadecimal can be disassembled(translated) to assembly language
- Assembly is the compiled result of human readable code, ie higher level languages like C and C++

Birth of a Program

- Human writes something in high level programming language (usually) like C or C++
- Compiler translates human readable code (C, C++) to machine readable (binary op codes) and spits out the end result (.exe, .bin, .dll, etc.)
- Computer runs the assembled instructions in the compiled program

Human Code → Compiler assembles → Machine Code

So what is RE?

- Moving backwards:

  Machine Code

  → Disassemble

  → Recreated human code (“decompilation”)

- Figuring out how something works in human terms, by looking at the machine code instructions packaged in the binary

Dynamic Analysis

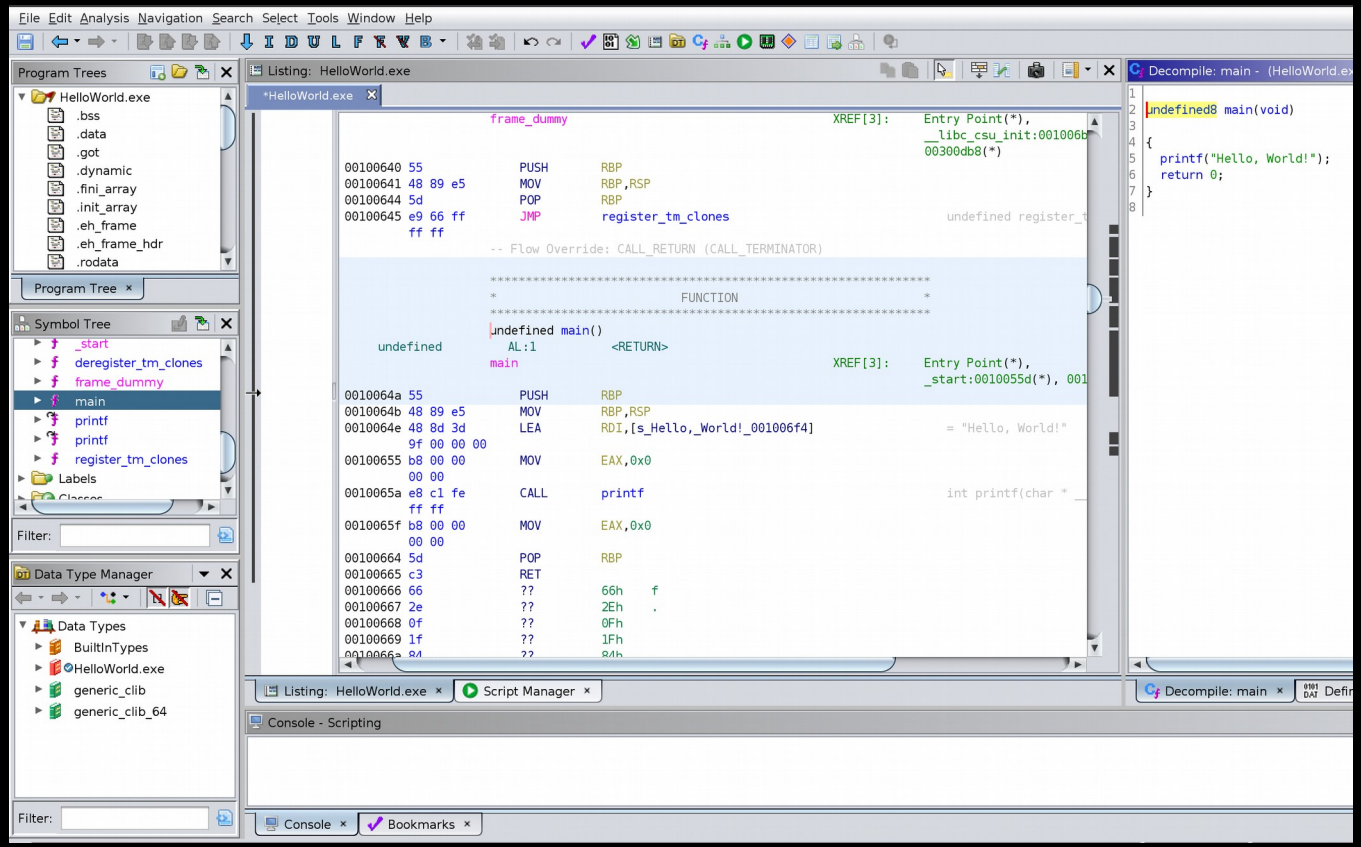
- Take file, run file, poke it with a stick and see what happens. “Vivisection”
- Detonate in a Sandbox → Take file, put in isolated digital space (virtual machine), run file, observe
- Debugger → Take file, run file, attach to it’s process and play with it. Can be risky if you’re not careful.
- Common Tool Options:
  - WinDBG, Immunity, OllyDBG

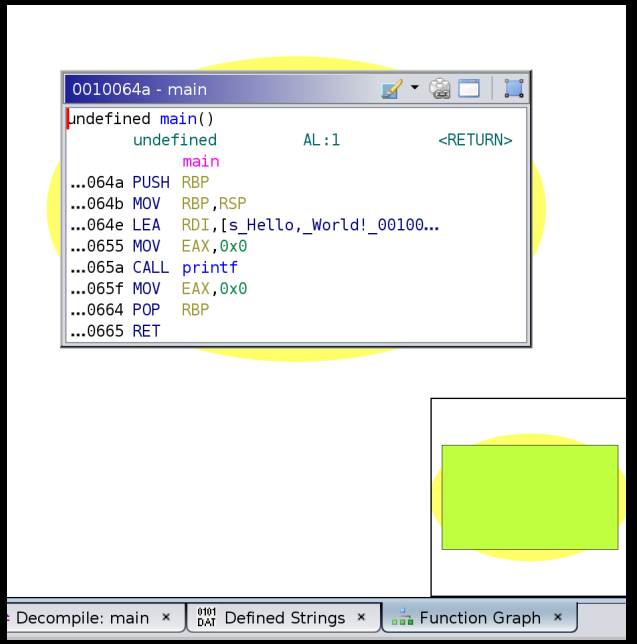
Static Analysis

Take file, take file apart. “Dissection”

- Disassembler → Binary in, assembly out
- Decompiler → Binary in, higher level code out (usually C)
- Most common tool choices:
  - IDA Pro, Ghidra, Binary Ninja, Radare2

Ghidra


File Edit Analysis Navigation Search Select Tools Window Help
Program Trees
HelloWorld.exe
.bss
.data
.got
.dynamic
.fini_array
.init_array
.eh_frame
.eh_frame_hdr
.rodata
Program Tree
Symbol Tree
_start
deregister_tm_clones
frame_dummy
main
printf
printf
register_tm_clones
Labels
Filter:
Data Type Manager
Data Types
BuiltInTypes
HelloWorld.exe
generic_clib
generic_clib_64
Filter:
Listing: HelloWorld.exe
*HelloWorld.exe
frame_dummy XREF[3]: Entry Point(*), __libc_csu_init:001006b 00300db8(*)
00100640 55 PUSH RBP
00100641 48 89 e5 MOV RBP,RSP
00100644 5d POP RBP
00100645 e9 66 ff ff JMP register_tm_clones undefined register_t
-- Flow Override: CALL_RETURN (CALL_TERMINATOR)
FUNCTION
undefined main()
undefined AL:1 <RETURN>
main XREF[3]: Entry Point(*), _start:0010055d(*), 001
0010064a 55 PUSH RBP
0010064b 48 89 e5 MOV RBP,RSP
0010064e 48 8d 3d 9f 00 00 00 LEA RDI,[s_Hello,_World!_001006f4] = "Hello, World!"
00100655 b8 00 00 00 00 MOV EAX,0x0
0010065a e8 c1 fe ff ff CALL printf int printf(char * __
0010065f b8 00 00 00 00 MOV EAX,0x0
00100664 5d POP RBP
00100665 c3 RET
00100666 66 ?? 66h f
00100667 2e ?? 2Eh .
00100668 0f ?? 0Fh
00100669 1f ?? 1Fh
Listing: HelloWorld.exe
Script Manager
Decompile: main - (HelloWorld.ex
undefined8 main(void)
{
printf("Hello, World!");
return 0;
}
Decompile: main
Console - Scripting
Console
Bookmarks


0010064a - main
undefined main()
undefined AL:1 <RETURN>
main
...064a PUSH RBP
...064b MOV RBP,RSP
...064e LEA RDI,[s_Hello,_World!_00100...
...0655 MOV EAX,0x0
...065a CALL printf
...065f MOV EAX,0x0
...0664 POP RBP
...0665 RET
Decompile: main
Defined Strings
Function Graph

IDA

File Edit Jump Search View Debugger Options Windows Help

Local Linux debugger

Library function | Regular function | Instruction | Data | Unexplored | External symbol

Functions window

Function name

- _init_proc
- sub_510
- **_printf**
- __cxa_finalize
- _start
- deregister_tm_clones
- register_tm_clones
- __do_global_dtors_aux
- frame_dummy
- **main**
- __libc_csu_init
- __libc_csu_fini
- _term_proc
- **printf**
- **__libc_start_main**
- __imp___cxa_finalize
- __gmon_start__

IDA View-A | Hex View-1 | Structures | Enums

```
; void _libc_csu_init(void)
public __libc_csu_init
__libc_csu_init proc near
push    r15
push    r14
mov     r15, rdx
push    r13
push    r12
lea     r12, __frame_dummy_init_array_entry
push    rbp
lea     rbp, __do_global_dtors_aux_fini_array_entry
push    rbx
mov     r13d, edi
mov     r14, rsi
sub     rbp, r12
sub     rsp, 8
sar     rbp, 3
call    _init_proc
test    rbp, rbp
jz      short loc_6C6
```

```
xor     ebx, ebx
nop     dword ptr [rax+rax+00000000h]
```

```
loc_6B0:
mov     rdx, r15
mov     rsi, r14
mov     edi, r13d
call    qword ptr [r12+rbx*8]
add     rbx, 1
cmp     rbp, rbx
jnz     short loc_6B0
```

```
loc_6C6:
add     rsp, 8
pop     rbx
pop     rbp
pop     r12
pop     r13
pop     r14
pop     r15
retn
__libc_csu_init endp
```

Graph overview

80.00% (-680,-34) (459,976) 00000674 0000000000000674: __libc_csu_init+4 (Synchronized with Hex View-1)

Output window

```
File '/home/synrw/PancakeCon/ReverseEngineering_101/HelloWorld.exe' has been successfully loaded into the database.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```


View Debugger Options Windows Help

Local Linux debugger

ular function | Instruction | Data | Unexplored | External symbol

IDA View-A | Hex View-1 | Structures | Enums


```
000000064A
000000064A ; =============== S U B R O U T I N E =======================================
000000064A
000000064A ; Attributes: bp-based frame
000000064A
000000064A ; int __cdecl main(int argc, const char **argv, const char **envp)
000000064A                 public main
000000064A main            proc near               ; DATA XREF: _start+1D↑o
000000064A                 push    rbp
000000064B                 mov     rbp, rsp
000000064E                 lea     rdi, format     ; "Hello, World!"
0000000655                 mov     eax, 0
000000065A                 call    _printf
000000065F                 mov     eax, 0
0000000664                 pop     rbp
0000000665                 retn
0000000665 main            endp
0000000665
0000000665 ; ---------------------------------------------------------------------------
0000000666                 align 10h
0000000670
0000000670 ; =============== S U B R O U T I N E =======================================
0000000670
0000000670
0000000670 ; void _libc_csu_init(void)
0000000670                 public __libc_csu_init
0000000670 __libc_csu_init proc near               ; DATA XREF: _start+16↑o
0000000670                 push    r15
0000000672                 push    r14
0000000674                 mov     r15, rdx
0000000677                 push    r13
0000000679                 push    r12
000000067B                 lea     r12, __frame_dummy_init_array_entry
0000000682                 push    rbp
0000000683                 lea     rbp, __do_global_dtors_aux_fini_array_entry
000000068A                 push    rbx
000000068B                 mov     r13d, edi
000000068E                 mov     r14, rsi
0000000691                 sub     rbp, r12
0000000694                 sub     rsp, 8
0000000698                 sar     rbp, 3
000000069C                 call    _init_proc
00000006A1                 test    rbp, rbp
```

0000064E 000000000000064E: main+4 (Synchronized with Hex View-1)

Output window

File '/home/synrw/PancakeCon/ReverseEngineering_101/HelloWorld.exe' has been successfully loaded into the database.
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.

IDC

AU: idle Down Disk: 1496GB

RE Subfields by Layer

- Software <-> Firmware <-> Hardware spectrum
- General leanings, not hard definitions:
  - Software: Web, Apps, Programs
  - Firmware: Drivers, bootloaders
  - Hardware: Embedded, PCBs
- Super fuzzy overlapping boundaries and definitions that no one agrees on

RE Subfields by Target

- Traditional “Computers”
  - PC, Servers
- Embedded devices
  - Phones, vehicles, medical devices, ICS
- Malware
- Signals & Protocols
  - RF (these people are insane), networks

Questions, comments, or just geeking out?

Contact info:

- Twitter: @SynapticRewrite
- SynapticRewrite@Gmail.com

Stay Safe, Stay Curious!