

GHIDRA

REVERSE ENGINEERING 101

A very brief intro to using Ghidra

ABOUT THIS TALK

- ▶ Very VERY basic intro to reverse engineering
- ▶ Overview of Ghidra and some helpful tips
- ▶ Live Demo of extracting info from a simple program
- ▶ Please ask questions!

WHAT IS REVERSE ENGINEERING (RE)

- ▶ Trying to derive some level of understanding of a binary
- ▶ This can be done by trying to disassemble the binary's machine code to generate assembly and then convert that into a high level language

TOOLS

- ▶ New Hotness: Ghidra
 - ▶ Free, from the NSA ('Murica!)
 - ▶ <https://ghidra-sre.org>
- ▶ Also awesome: radare2 & optional gui "Cutter"
 - ▶ Free, cross platform, CLI only unless using Cutter (GUI)
 - ▶ <https://rada.re>
 - ▶ <https://github.com/radareorg/cutter/releases>
- ▶ Old 'n Busted (not really, IDA is still the most popular RE tool): IDA / IDA Pro
 - ▶ <https://www.hex-rays.com/products/ida/>

ABOUT GHIDRA (GEE-DRUH)

キングギドラ KINGU GIDORA



- ▶ From [Ghidorah, the Three-Headed Monster](#) (1964)
- ▶ Developed by NSA, released at RSA San Francisco 2019
- ▶ Java Based
- ▶ Supported Architectures:
 - ▶ [16, 32 and 64 bit x86](#)
 - ▶ [ARM and AARCH64](#)
 - ▶ [PowerPC 32/64 and PowerPC VLE](#)
 - ▶ [MIPS 16/32/64](#)
 - ▶ [MicroMIPS](#)
 - ▶ [68xxx](#)
 - ▶ [Java](#)
 - ▶ [DEX bytecode](#)
 - ▶ [PA-RISC](#)
 - ▶ [PIC 12/16/17/18/24](#)
 - ▶ [SPARC 32/64](#)
 - ▶ [CR16C](#)
 - ▶ [Z80](#)
 - ▶ [6502](#)
 - ▶ [8051](#)
 - ▶ [MSP430](#)
 - ▶ [AVR8](#)
 - ▶ [AVR32](#)

BASIC WORKFLOW

- ▶ First determine your goals
 - ▶ Learn more about something?
 - ▶ Search for exploits / vulns?
 - ▶ Set a target for yourself - don't get lost in the rabbit hole
- ▶ Load binary into tool
- ▶ Analyze it
- ▶ Inspect the decompiled code. Try to gain insight into what it does
 - ▶ Start at entry point
 - ▶ Look at variables defined
 - ▶ Rename variables to make things clearer
 - ▶ Look at strings and other hard coded information

IMPORT SETTINGS

- ▶ Create Project
- ▶ Add files
 - ▶ Check "Import External Libraries"
- ▶ Analyze file
 - ▶ Enable "Decompile Parameter IDs"

TIPS

Enable Decompiler Parameter ID

The screenshot shows the 'Analysis Options' dialog box in a decompiler. The 'Analyzers' tab is active, displaying a list of analysis options. The 'Decompiler Parameter ID' option is highlighted in orange and has a checkmark in the 'Enabled' column. An orange callout bubble with an exclamation mark and the word 'Enable' points to this option. The 'Description' pane on the right explains that this option creates parameter and local variables for a function using the decompiler, with a warning that it can be significant. The 'Options' pane shows settings for 'Analysis Clear Level' (ANALYSIS), 'Analysis Decompiler Timeout (sec)' (60), 'Commit Data Types' (checked), and 'Commit Void Return Values' (unchecked). The 'Analyze' button is highlighted in blue.

Analysis Options

Analyzers

Enabled	Analyzer Name
<input type="checkbox"/>	Aggressive Instruction Finder (Prototype)
<input checked="" type="checkbox"/>	Apply Data Archives
<input checked="" type="checkbox"/>	ASCII Strings
<input checked="" type="checkbox"/>	Call Convention Identification
<input checked="" type="checkbox"/>	Call-Fixup Installer
<input type="checkbox"/>	Condense Filler Bytes (Prototype)
<input checked="" type="checkbox"/>	Create Address Tables
<input checked="" type="checkbox"/>	Data Reference
<input checked="" type="checkbox"/>	Decompiler Parameter ID
<input checked="" type="checkbox"/>	Decompiler Switch Analysis

Select All Deselect All Restore Defaults

Description

Creates parameter and local variables for a Function using Decompiler.
WARNING: This can take a SIGNIFICANT

Options

Analysis Clear Level: ANALYSIS

Analysis Decompiler Timeout (sec): 60

Commit Data Types: ☒

Commit Void Return Values: ☐

Analyze Cancel

TIPS

Enable Binary Segment Overview

CodeBrowser: Examples:/password.out

on Search Select Tools Window Help

Listing: password.out

*password.out

100000eae 00 ?? 00n
100000eaf 00 ?? 00h

//
// __text
// __TEXT
// ram: 100000eb0-100000f21
//

* FUNCTION

int __stdcall main(void)

EAX:4 <RETURN>

Stack[-0xc]:4 local_c
Stack[-0x10]:4 local_10

Stack[-0x14]:4 local_14

Stack[-0x18]:4 local_18
Stack[-0x1c]:4 local_1c
Stack[-0x20]:4 local_20
Stack[-0x24]:4 local_24

_main
main

100000eb0 55 PUSH RBP
100000eb1 48 89 e5 MOV RBP,RSP
100000eb4 48 83 ec 20 SUB RSP,0x20
100000eb8 c7 45 fc MOV dword ptr [RBP + loc_100000f21],00000000
100000ebf c7 45 f8 MOV dword ptr [RBP + loc_100000f25],9a020000
100000ec6 48 8d 3d LEA RDI,[s_Please_enter_password]
100000ecd b0 00 MOV AL,0x0
100000ecf e8 4e 00 CALL stubs::printf

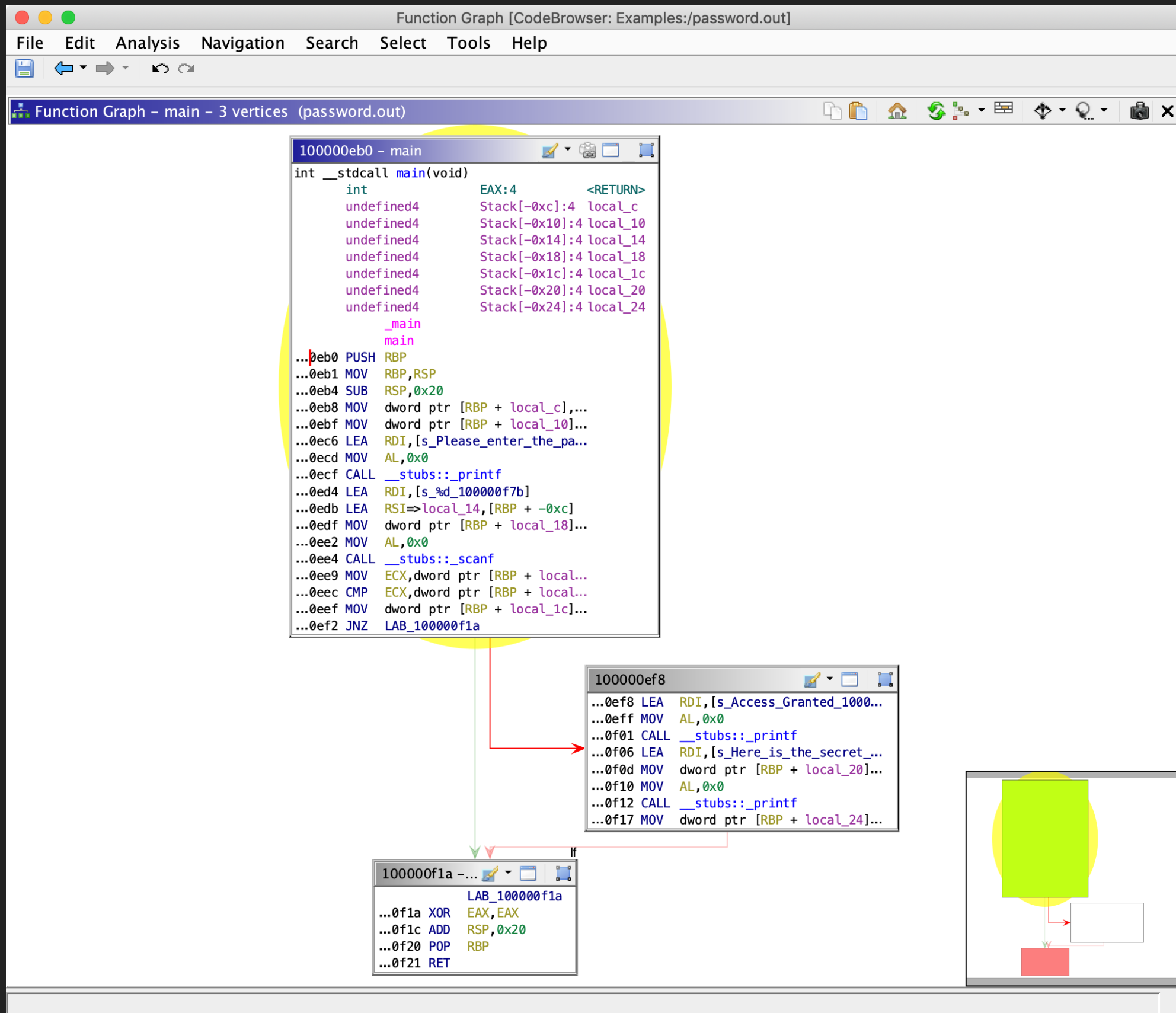
Decompile: main - (password.out)

```
3  
4 {  
5     int local_14;  
6     int local_10;  
7     undefined4 local_c;  
8  
9     local_c = 0;  
10    local_10 = 0x29a;  
11    _printf("Please enter the password to continue\n");  
12    _scanf("%d",&local_14);  
13    if (local_14 == local_10) {  
14        _printf("Access Granted\n");  
15        _printf("Here is the secret code: 123456");  
16    }  
17    return 0;  
18 }  
19
```

Show Entropy
✓ Show Overview

TIPS

Graph view provides a visual overview of program flow



HOTKEYS

- ▶ L - Rename
- ▶ G - Goto
- ▶ ; - Add comment

DEM0000000

RESOURCES

- ▶ Microsoft PE Format overview - <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format>
- ▶ Creating a simple C reverse shell - <https://rastating.github.io/creating-a-reverse-tcp-shellcode/>
- ▶ Ghidra Quick Tutorial - <https://www.youtube.com/watch?v=fTGTnrgjuGA>
- ▶ CrackMe's - Programs to test your RE skills - <https://crackmes.one>
- ▶ Ghidra Cheat Sheet - <https://ghidra-sre.org/CheatSheet.html>
- ▶ Amazon Corretto JDK 11 - <https://docs.aws.amazon.com/corretto/latest/corretto-11-ug/downloads-list.html>