
# Reverse Engineering with Ghidra

Dr. Alexandros Kapravelos
akaprav@ncsu.edu

Aleksandr Napetyan
anahape@ncsu.edu

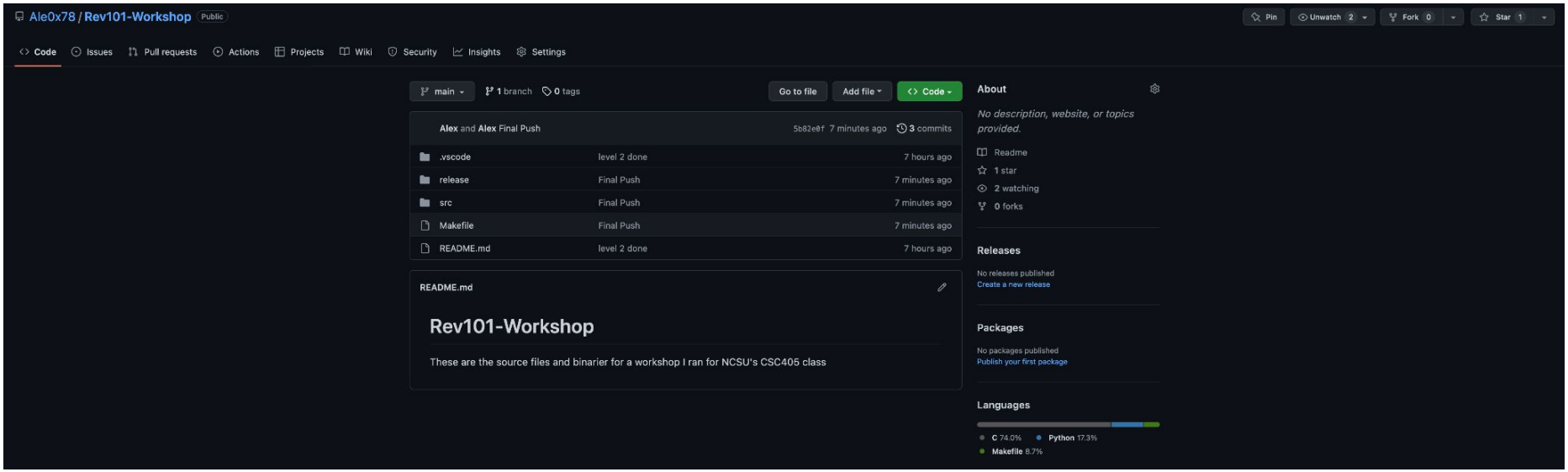
Slides adapted from John (Jack) Allison

# Reverse Engineering 10: Takeaways

- Ghidra and other tools
- Sample CTF Challenges
- Practice -> Write Up -> repeat
- Where do you go from here?

# Follow along!

go.ncsu.edu/rev101




Flags are in the format workshop{(.+)}

# “What’s a Ghidra?”

“A software reverse engineering (SRE) suite of tools developed by NSA's Research Directorate in support of the Cybersecurity mission”

*(It’s a tool for reverse engineering stuff)*

*(declassified after it’s existence was leaked by WikiLeaks ;) )*

*https://ghidra-sre.org/CheatSheet.html*

# What can it do?

- x86 **decompilation** (and ARM… and MIPS… and a lot more…)
  - Pretty much any architecture that's commonly used

- Debugging binaries under Windows and Linux (WinDbg and GDB)

- Scripting with Python (not covered here) to extend the feature set or automate tasks

# Extending Ghidra

- https://github.com/AllsafeCyberSecurity/awesome-ghidra
- https://github.com/topics/ghidra-scripts
- https://github.com/federicodotta/ghidra-scripts

# It's free and open source

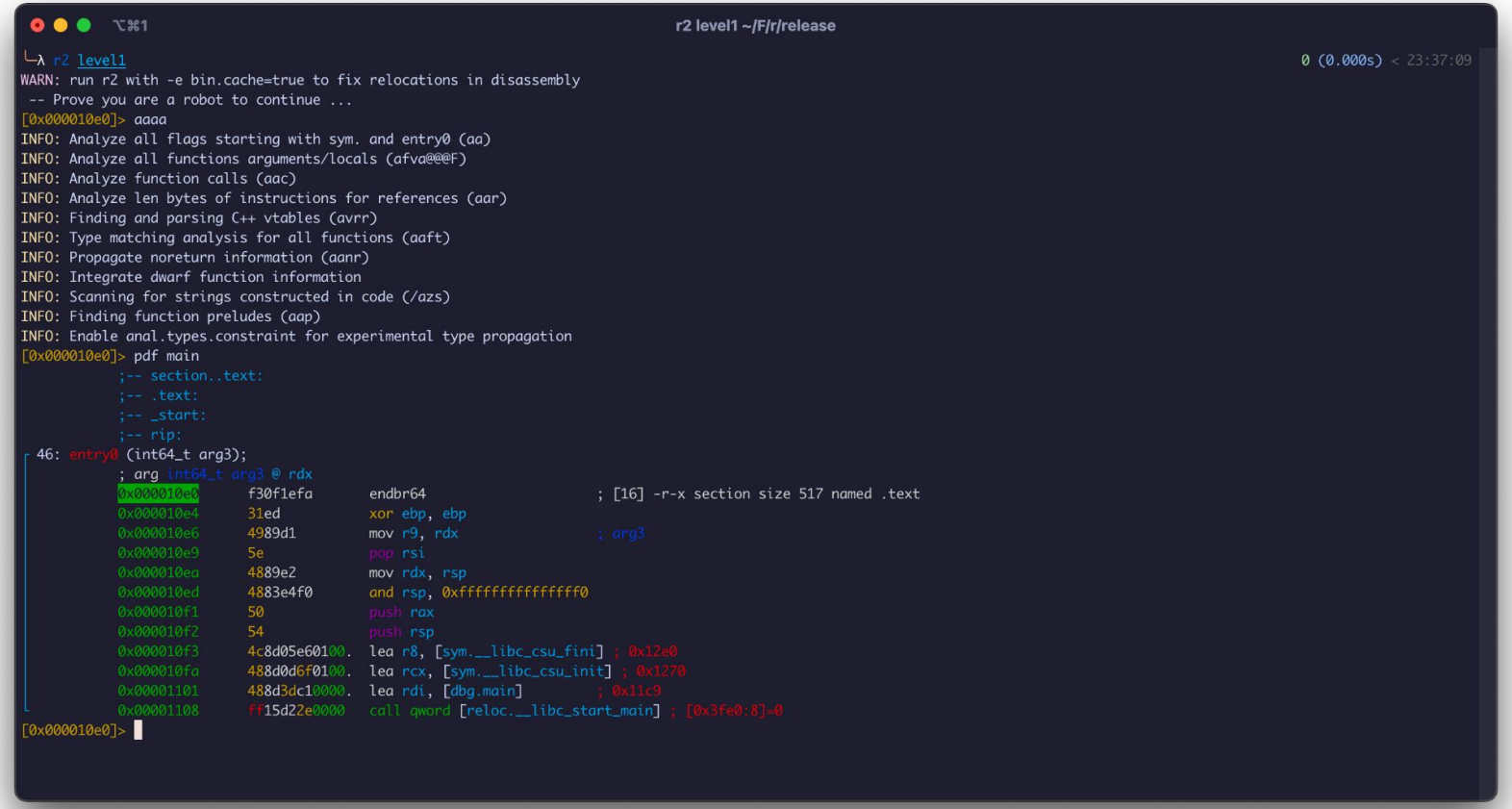
- Possibly limited feature set compared to…
  - IDA Pro (multiple thousands), state of the art
  - Binary Ninja ($300 personal license)
- Way easier than
  - Radare2
  - Cutter

- But, it's **free**

- And now has a debugger, which was a missing feature till recently

# Some assembly required

# Radare2


r2 level1 ~/F/r/release
r2 level1
0 (0.000s) < 23:37:09
WARN: run r2 with -e bin.cache=true to fix relocations in disassembly
-- Prove you are a robot to continue ...
[0x000010e0]> aaaa
INFO: Analyze all flags starting with sym. and entry0 (aa)
INFO: Analyze all functions arguments/locals (afva@@@F)
INFO: Analyze function calls (aac)
INFO: Analyze len bytes of instructions for references (aar)
INFO: Finding and parsing C++ vtables (avrr)
INFO: Type matching analysis for all functions (aaft)
INFO: Propagate noreturn information (aanr)
INFO: Integrate dwarf function information
INFO: Scanning for strings constructed in code (/azs)
INFO: Finding function preludes (aap)
INFO: Enable anal.types.constraint for experimental type propagation
[0x000010e0]> pdf main
;-- section..text:
;-- .text:
;-- _start:
;-- rip:
46: entry0 (int64_t arg3);
; arg int64_t arg3 @ rdx
0x000010e0 f30f1efa endbr64 ; [16] -r-x section size 517 named .text
0x000010e4 31ed xor ebp, ebp
0x000010e6 4989d1 mov r9, rdx ; arg3
0x000010e9 5e pop rsi
0x000010ea 4889e2 mov rdx, rsp
0x000010ed 4883e4f0 and rsp, 0xfffffffffffffff0
0x000010f1 50 push rax
0x000010f2 54 push rsp
0x000010f3 4c8d05e60100. lea r8, [sym.__libc_csu_fini] ; 0x12e0
0x000010fa 488d0d6f0100. lea rcx, [sym.__libc_csu_init] ; 0x1270
0x00001101 488d3dc10000. lea rdi, [dbg.main] ; 0x11c9
0x00001108 ff15d22e0000 call qword [reloc.__libc_start_main] ; [0x3fe0:8]=0
[0x000010e0]>

# What it can be used for

- Malware analysis
- CTF Challenges 👀
- Learning how your favorite program works, under the hood
- Want to run Linux on a new M1 Mac?

# Asahi Linux

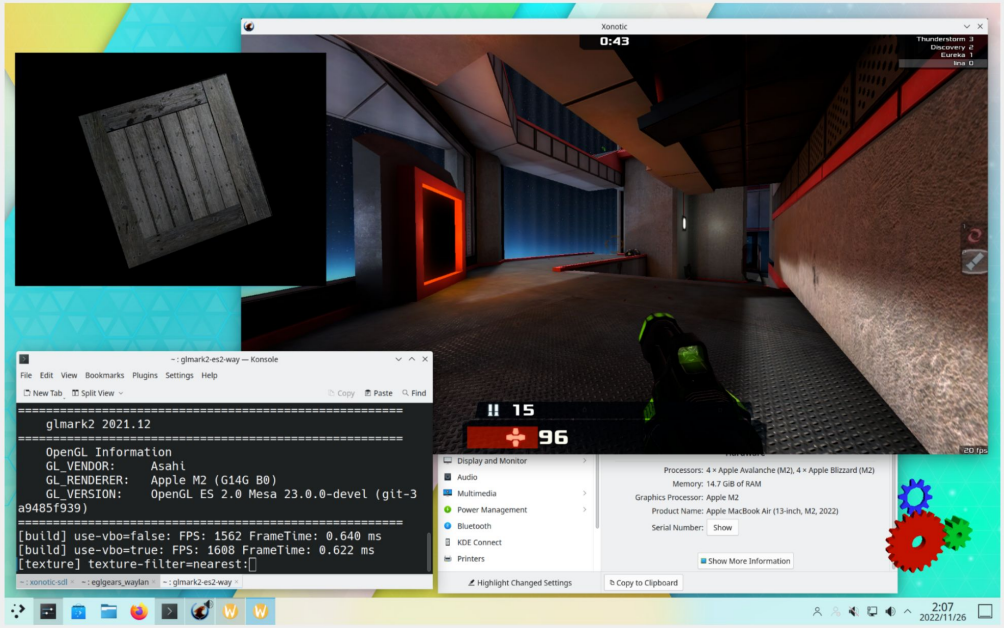


## Tales of the M1 GPU

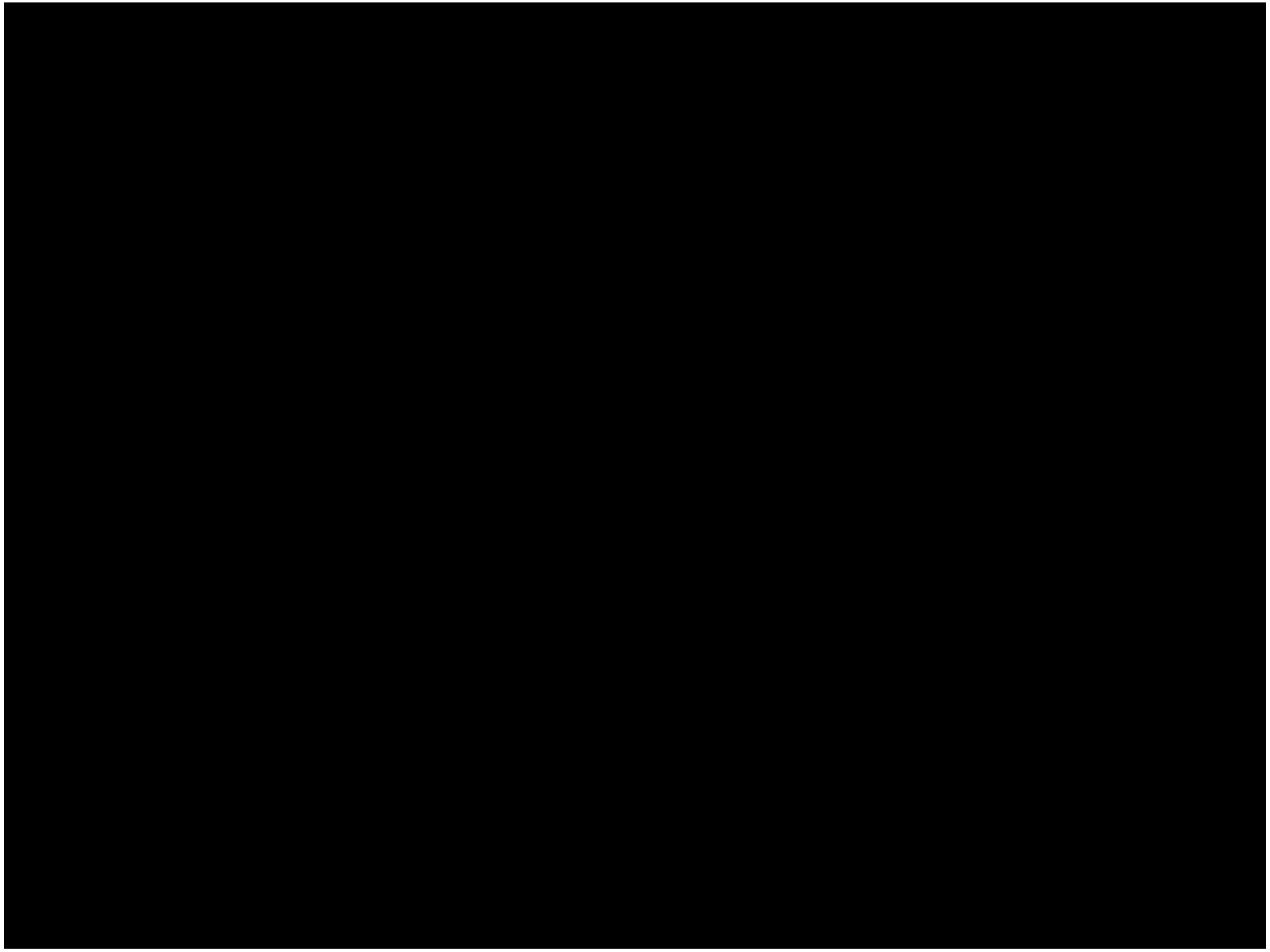
Previous Next

Hello everyone, Asahi Lina here!✨

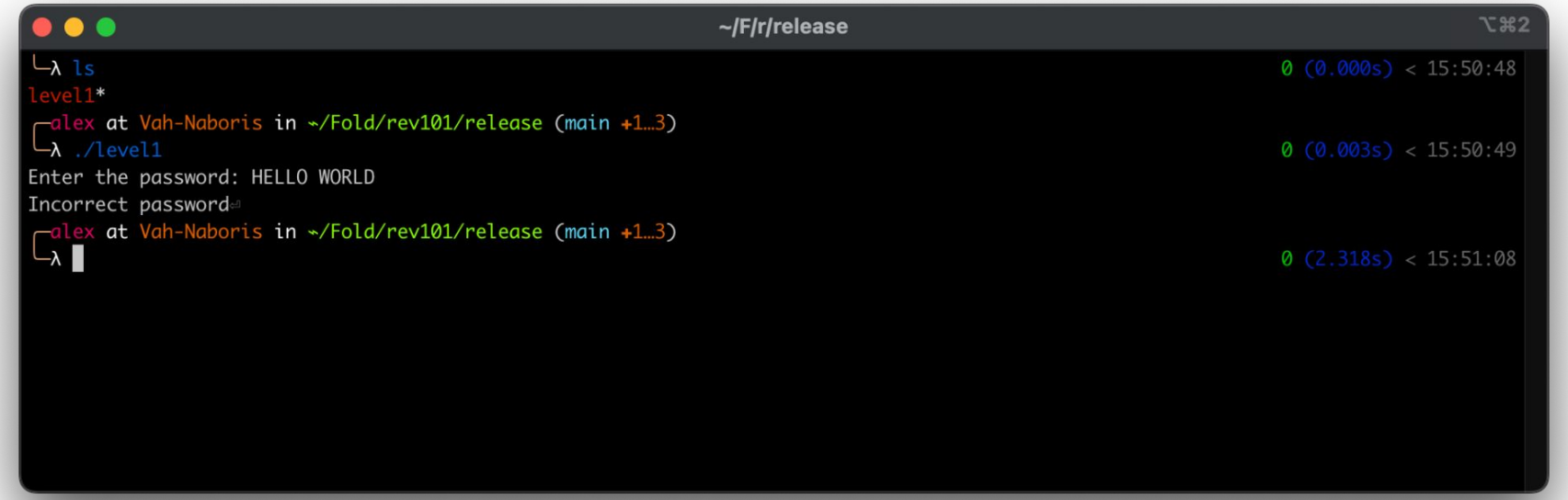
marcan asked me to write an article about the M1 GPU, so here we are~! It's been a long road over the past few months and there's a lot to cover, so I hope you enjoy it!

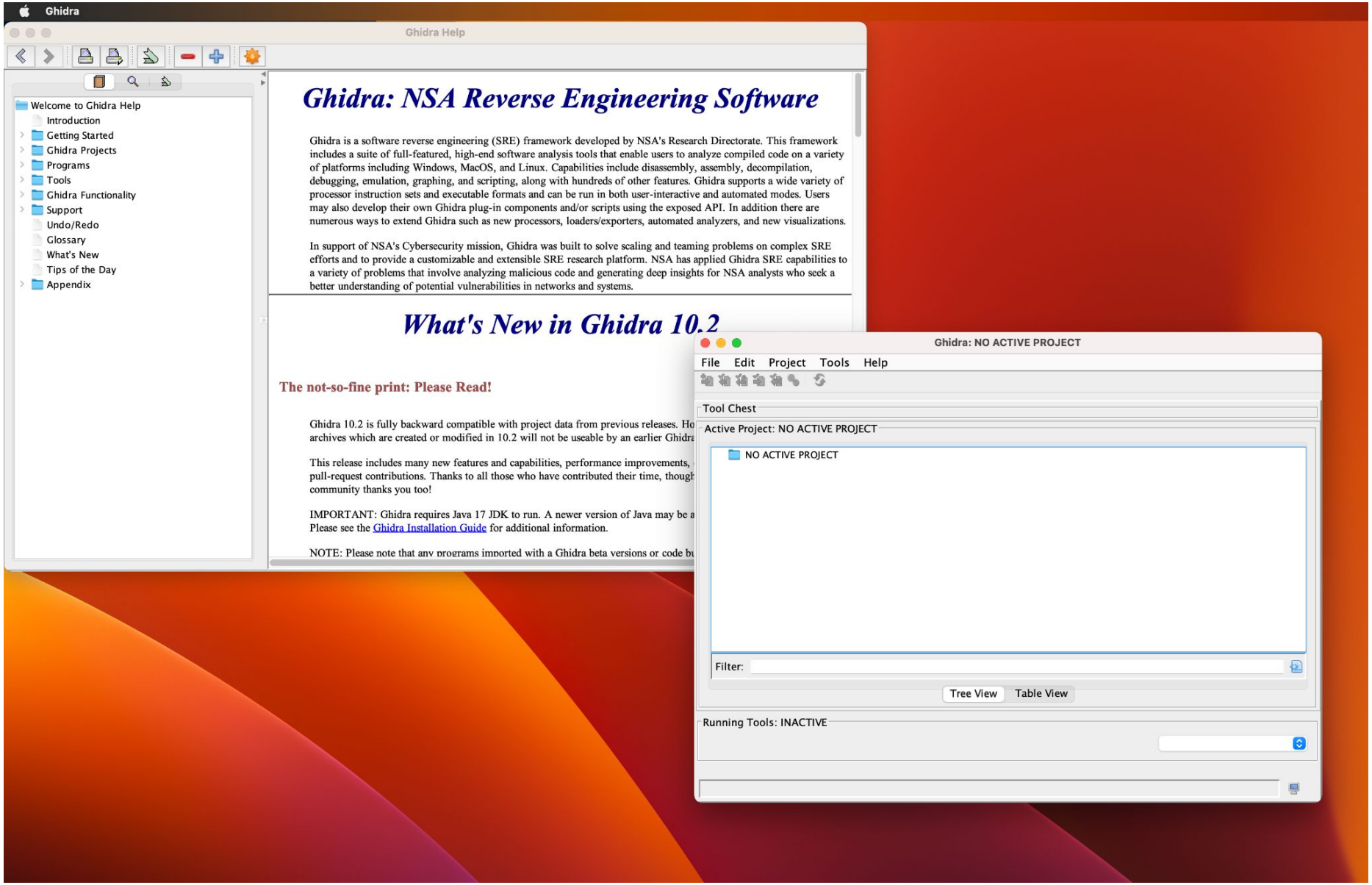


# Let’s do a quick demonstration.

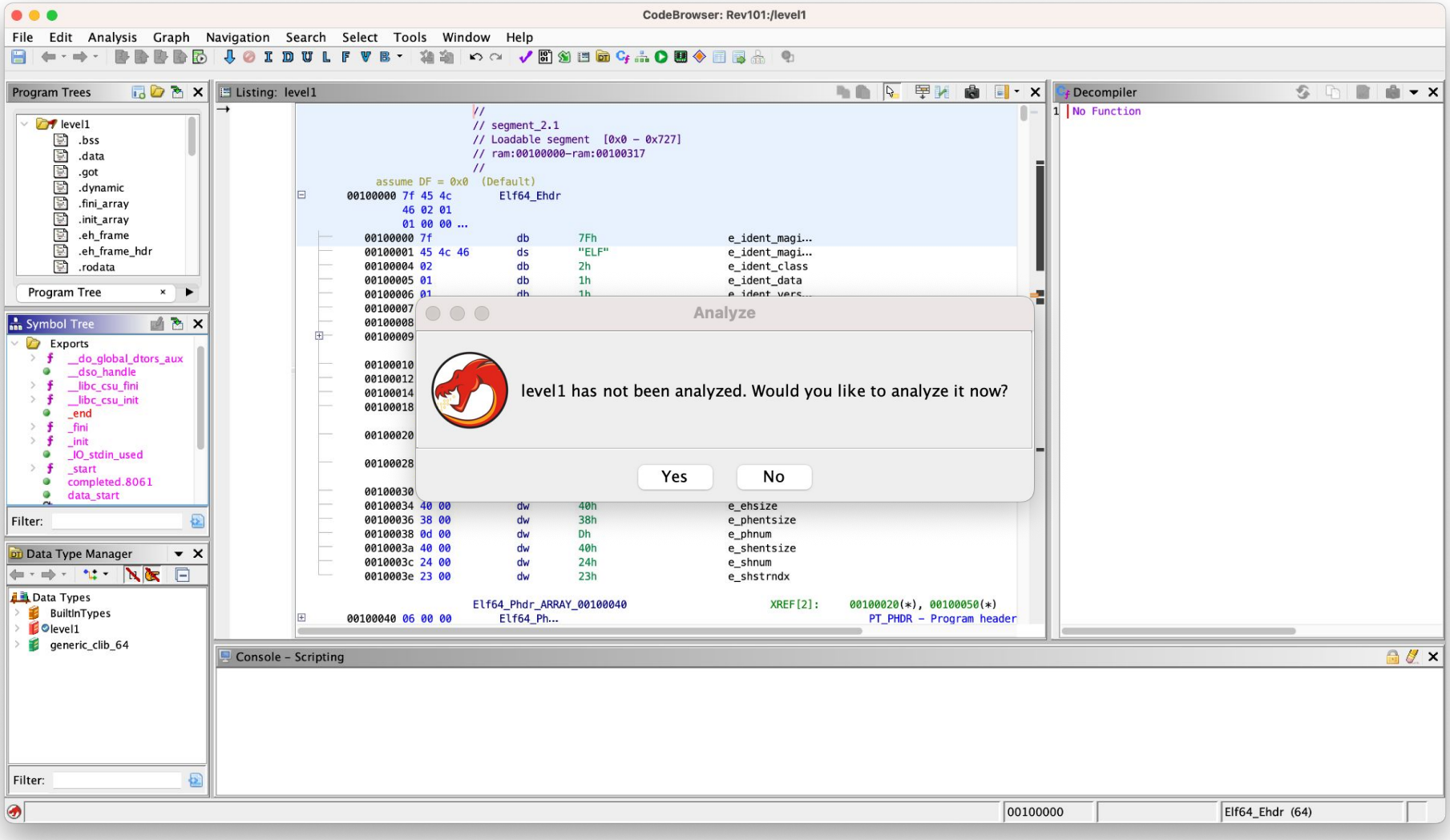


# Level 1


~/F/r/release
⌥⌘2
└λ ls
0 (0.000s) < 15:50:48
level1*
┌─alex at Vah-Naboris in ~/Fold/rev101/release (main +1…3)
└λ ./level1
0 (0.003s) < 15:50:49
Enter the password: HELLO WORLD
Incorrect password
┌─alex at Vah-Naboris in ~/Fold/rev101/release (main +1…3)
└λ
0 (2.318s) < 15:51:08


Ghidra
Ghidra Help
Welcome to Ghidra Help
Introduction
Getting Started
Ghidra Projects
Programs
Tools
Ghidra Functionality
Support
Undo/Redo
Glossary
What's New
Tips of the Day
Appendix
Ghidra: NSA Reverse Engineering Software
Ghidra is a software reverse engineering (SRE) framework developed by NSA's Research Directorate. This framework includes a suite of full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms including Windows, MacOS, and Linux. Capabilities include disassembly, assembly, decompilation, debugging, emulation, graphing, and scripting, along with hundreds of other features. Ghidra supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes. Users may also develop their own Ghidra plug-in components and/or scripts using the exposed API. In addition there are numerous ways to extend Ghidra such as new processors, loaders/exporters, automated analyzers, and new visualizations.
In support of NSA's Cybersecurity mission, Ghidra was built to solve scaling and teaming problems on complex SRE efforts and to provide a customizable and extensible SRE research platform. NSA has applied Ghidra SRE capabilities to a variety of problems that involve analyzing malicious code and generating deep insights for NSA analysts who seek a better understanding of potential vulnerabilities in networks and systems.
What's New in Ghidra 10.2
The not-so-fine print: Please Read!
Ghidra 10.2 is fully backward compatible with project data from previous releases. Ho
archives which are created or modified in 10.2 will not be useable by an earlier Ghidra
This release includes many new features and capabilities, performance improvements,
pull-request contributions. Thanks to all those who have contributed their time, though
community thanks you too!
IMPORTANT: Ghidra requires Java 17 JDK to run. A newer version of Java may be a
Please see the Ghidra Installation Guide for additional information.
NOTE: Please note that any programs imported with a Ghidra beta versions or code bu
Ghidra: NO ACTIVE PROJECT
File
Edit
Project
Tools
Help
Tool Chest
Active Project: NO ACTIVE PROJECT
NO ACTIVE PROJECT
Filter:
Tree View
Table View
Running Tools: INACTIVE

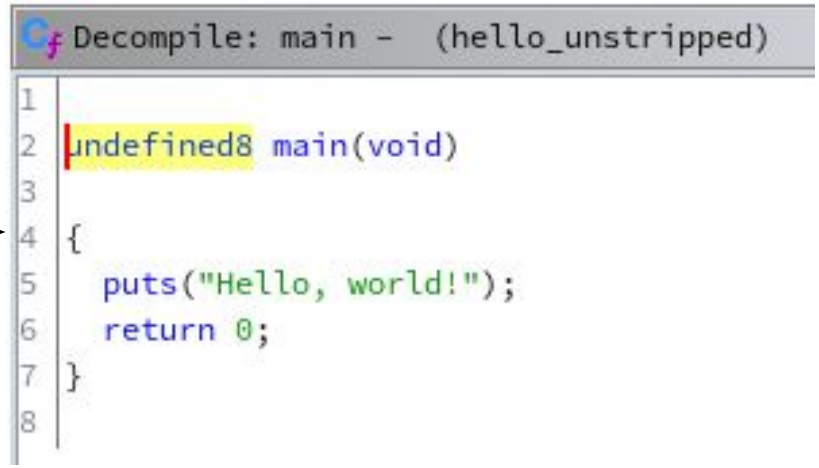
# Let Ghidra do the magic



# Decompiled code

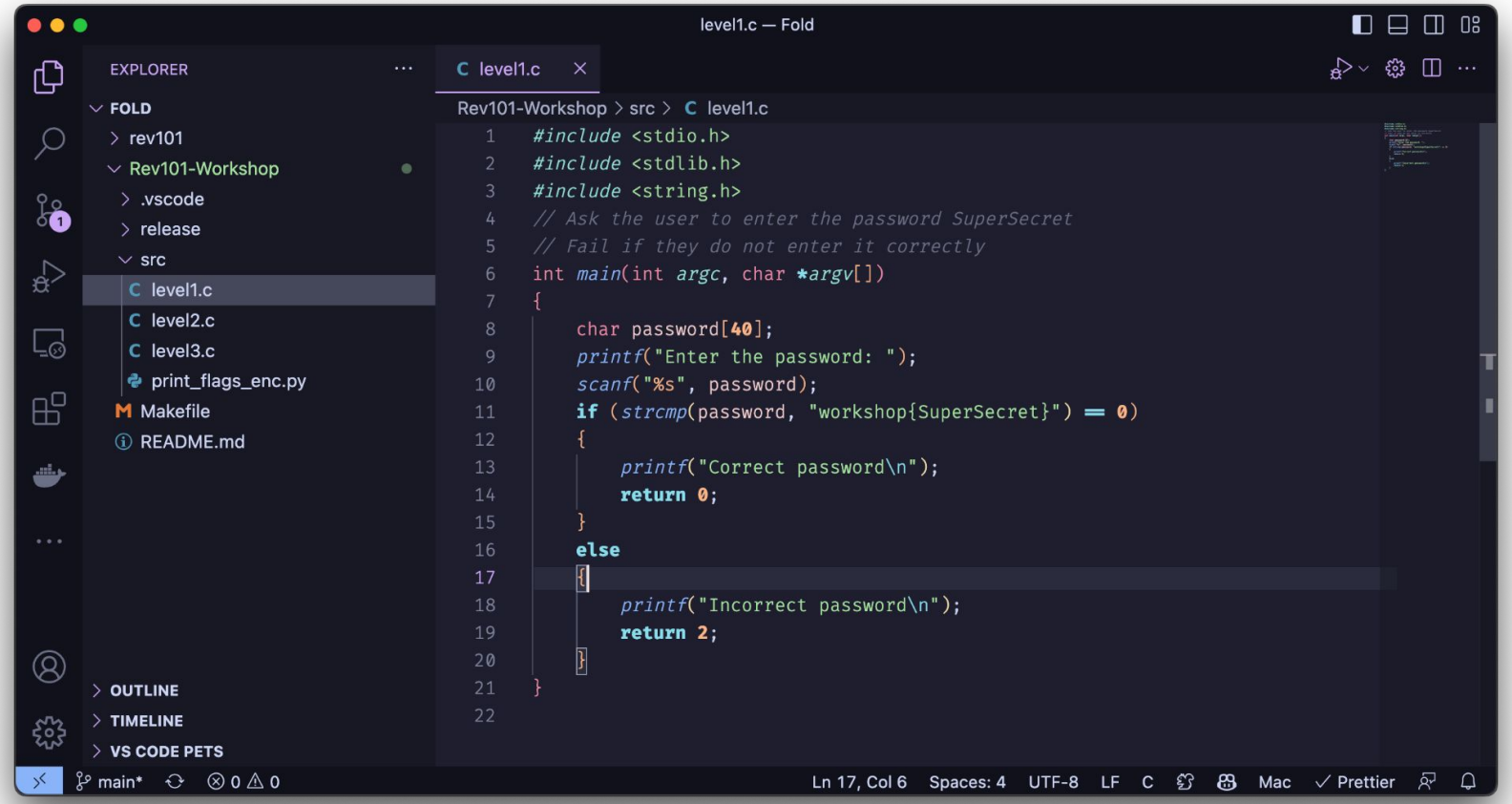
```c
#include <stdio.h>

int main(void)
{
    printf("Hello, world!\n");
    return 0;
}
```

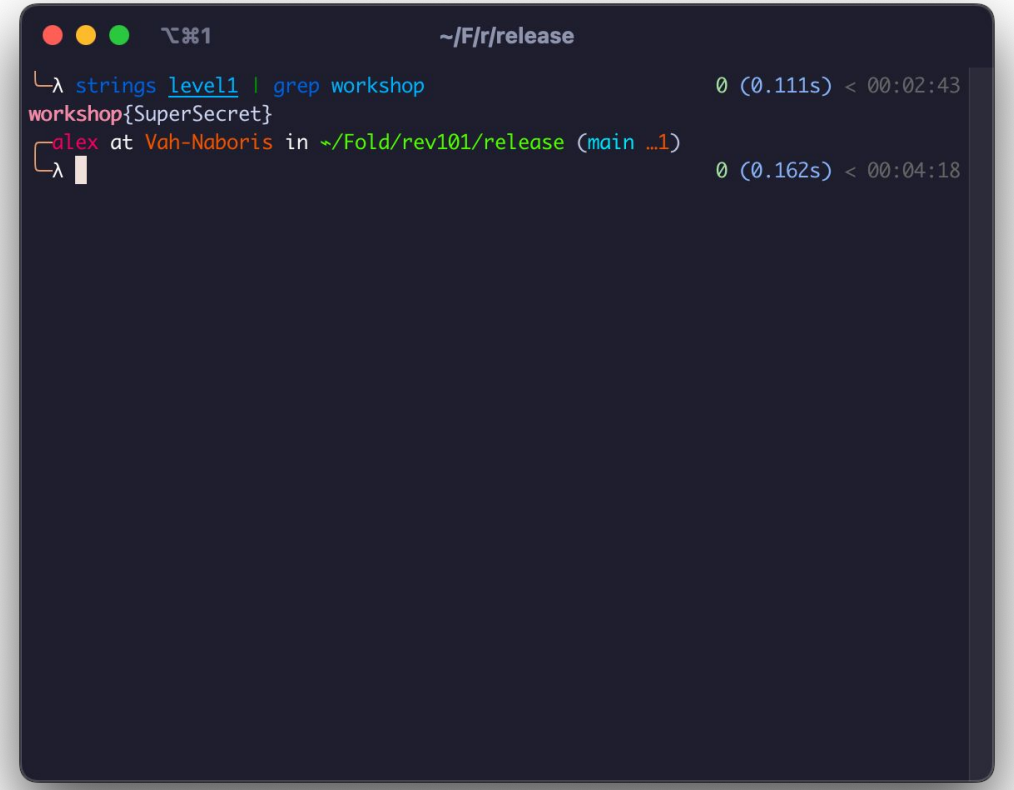



- Best effort attempt
- NOT 1:1
- Still, very helpful

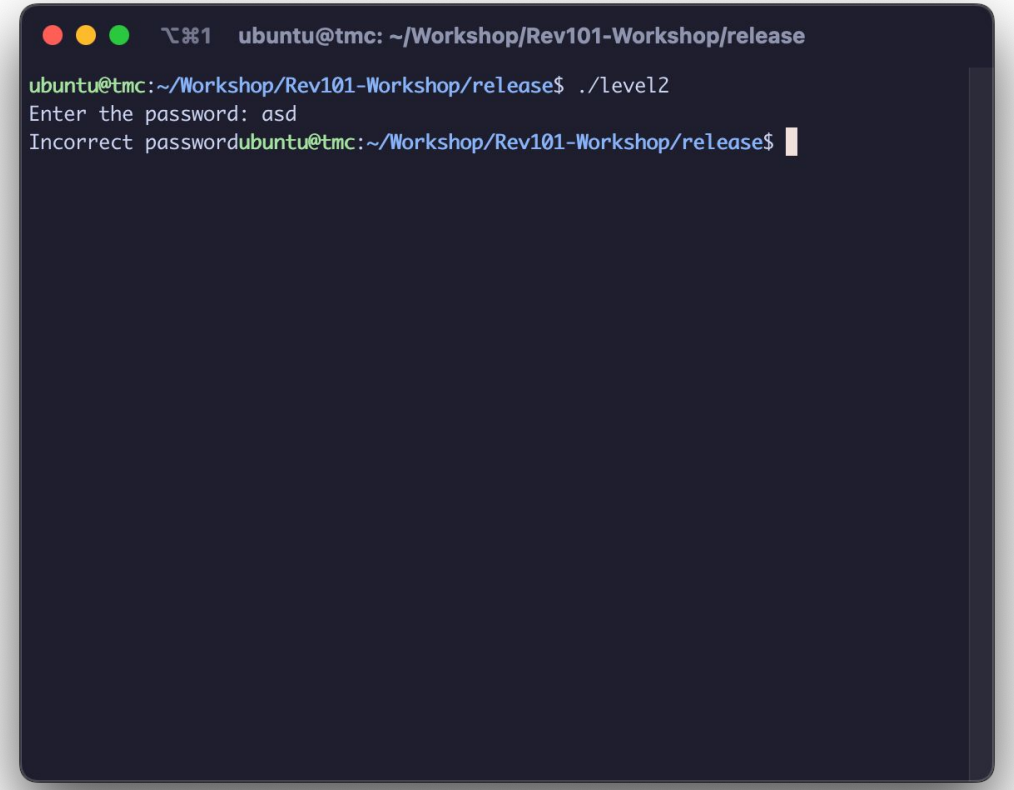
# Recall: C and Software Tools



# Enter: `strings`



# Level 2

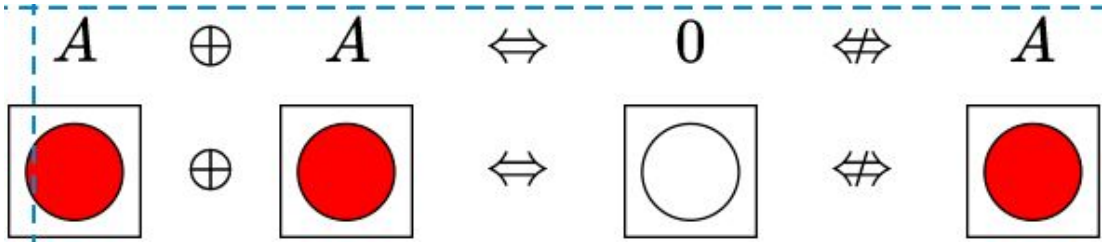

⌥⌘1 ubuntu@tmc: ~/Workshop/Rev101-Workshop/release
ubuntu@tmc:~/Workshop/Rev101-Workshop/release$ ./level2
Enter the password: asd
Incorrect passwordubuntu@tmc:~/Workshop/Rev101-Workshop/release$

# Recall: XOR

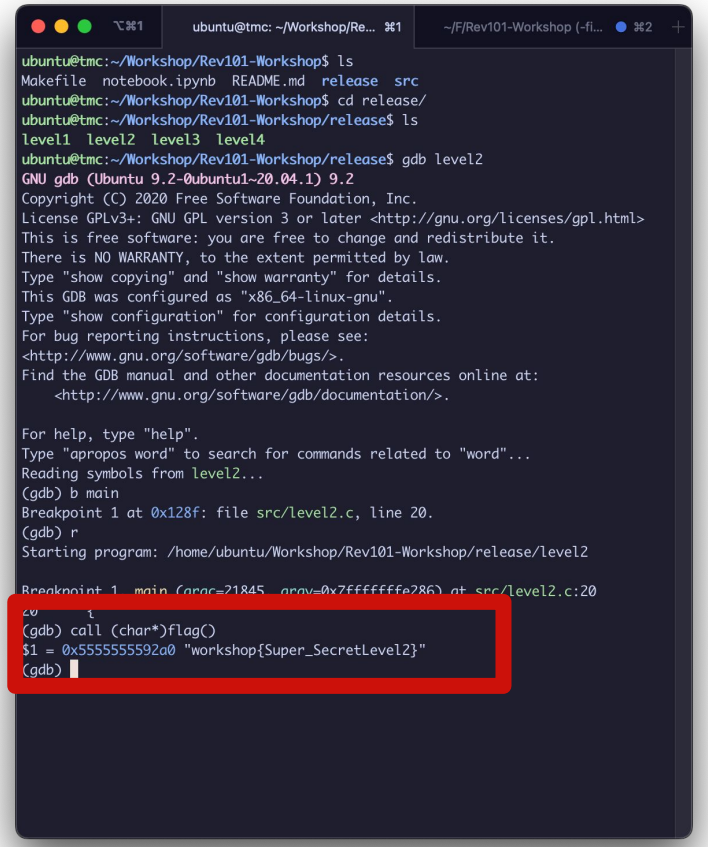
## Truth table [ edit ]

The truth table of A XOR B shows that it outputs true whenever the inputs differ:

| $A$ | $B$ | $A \oplus B$ |
|---|---|---|
| False | False | False |
| False | True | True |
| True | False | True |
| True | True | False |

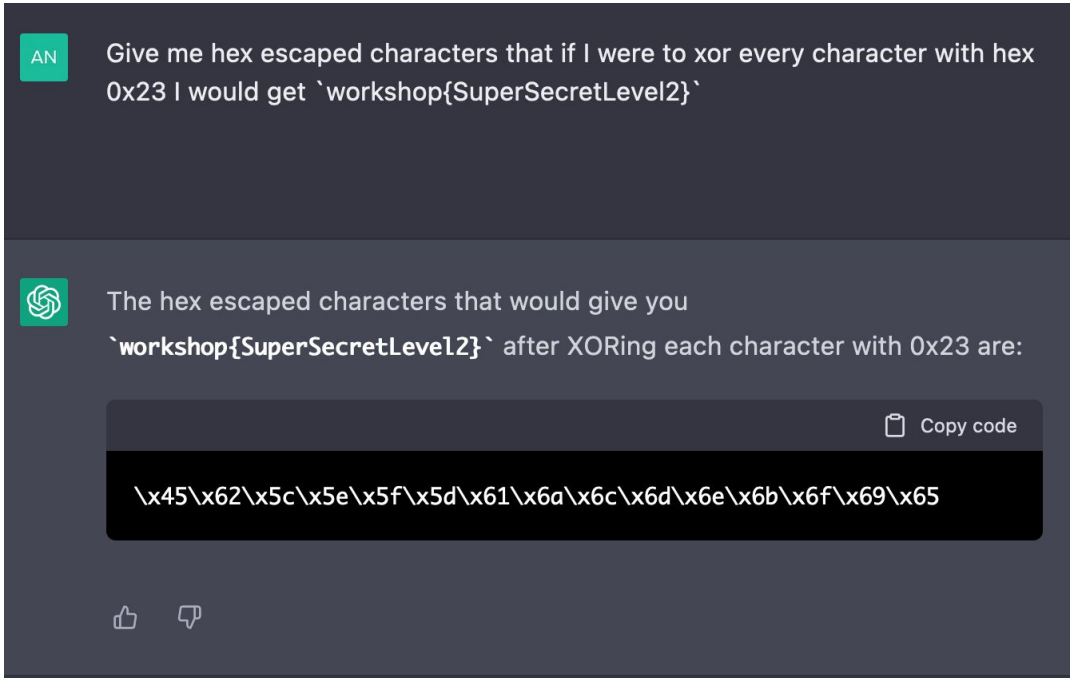


# Enter GDB

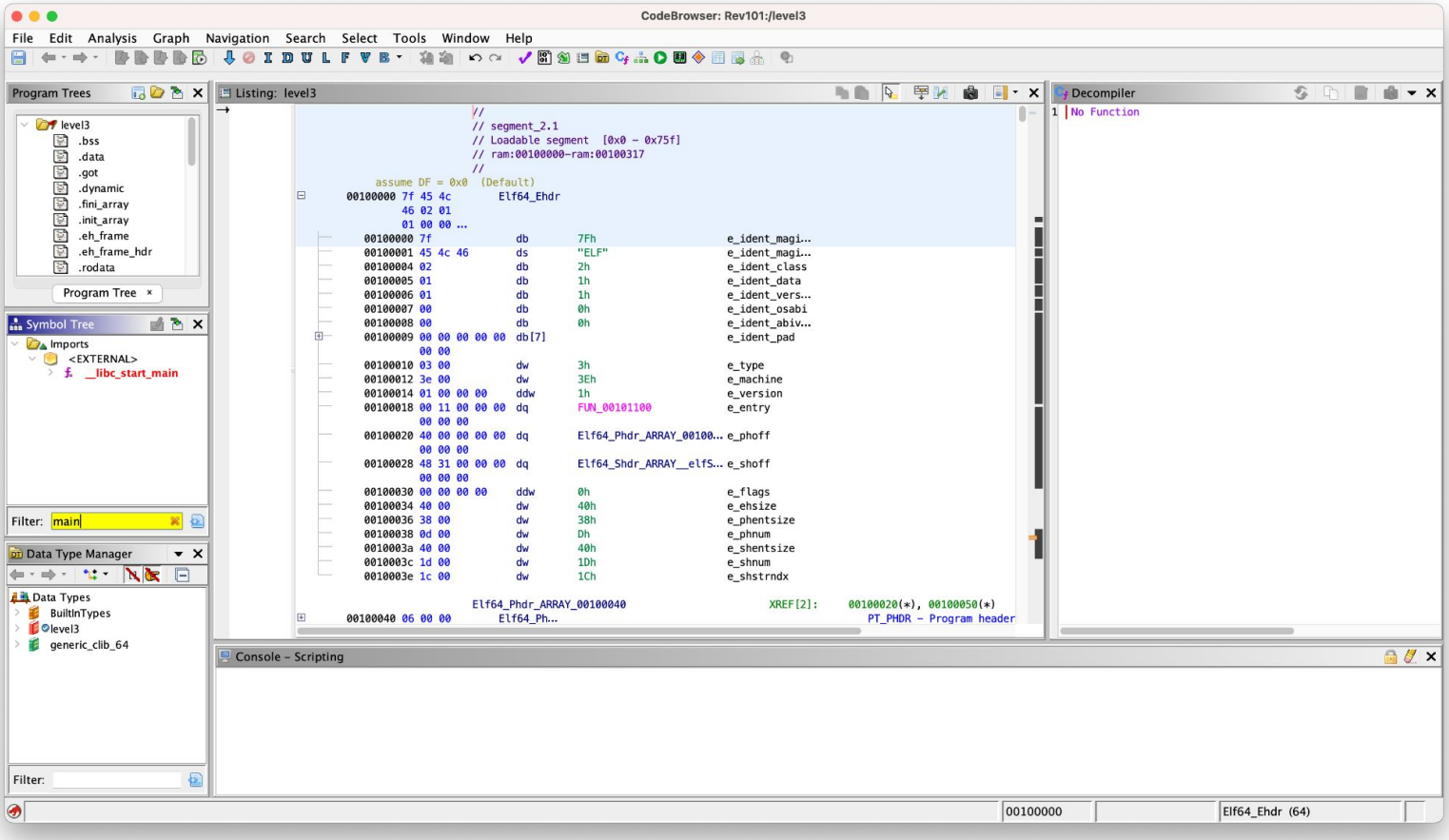

ubuntu@tmc: ~/Workshop/Re... ⌘1
~/F/Rev101-Workshop (-fi... ⌘2
ubuntu@tmc:~/Workshop/Rev101-Workshop$ ls
Makefile notebook.ipynb README.md release src
ubuntu@tmc:~/Workshop/Rev101-Workshop$ cd release/
ubuntu@tmc:~/Workshop/Rev101-Workshop/release$ ls
level1 level2 level3 level4
ubuntu@tmc:~/Workshop/Rev101-Workshop/release$ gdb level2
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from level2...
(gdb) b main
Breakpoint 1 at 0x128f: file src/level2.c, line 20.
(gdb) r
Starting program: /home/ubuntu/Workshop/Rev101-Workshop/release/level2
(gdb) call (char*)flag()
$1 = 0x5555555592a0 "workshop{Super_SecretLevel2}"
(gdb)

# Sidenote: Level 2

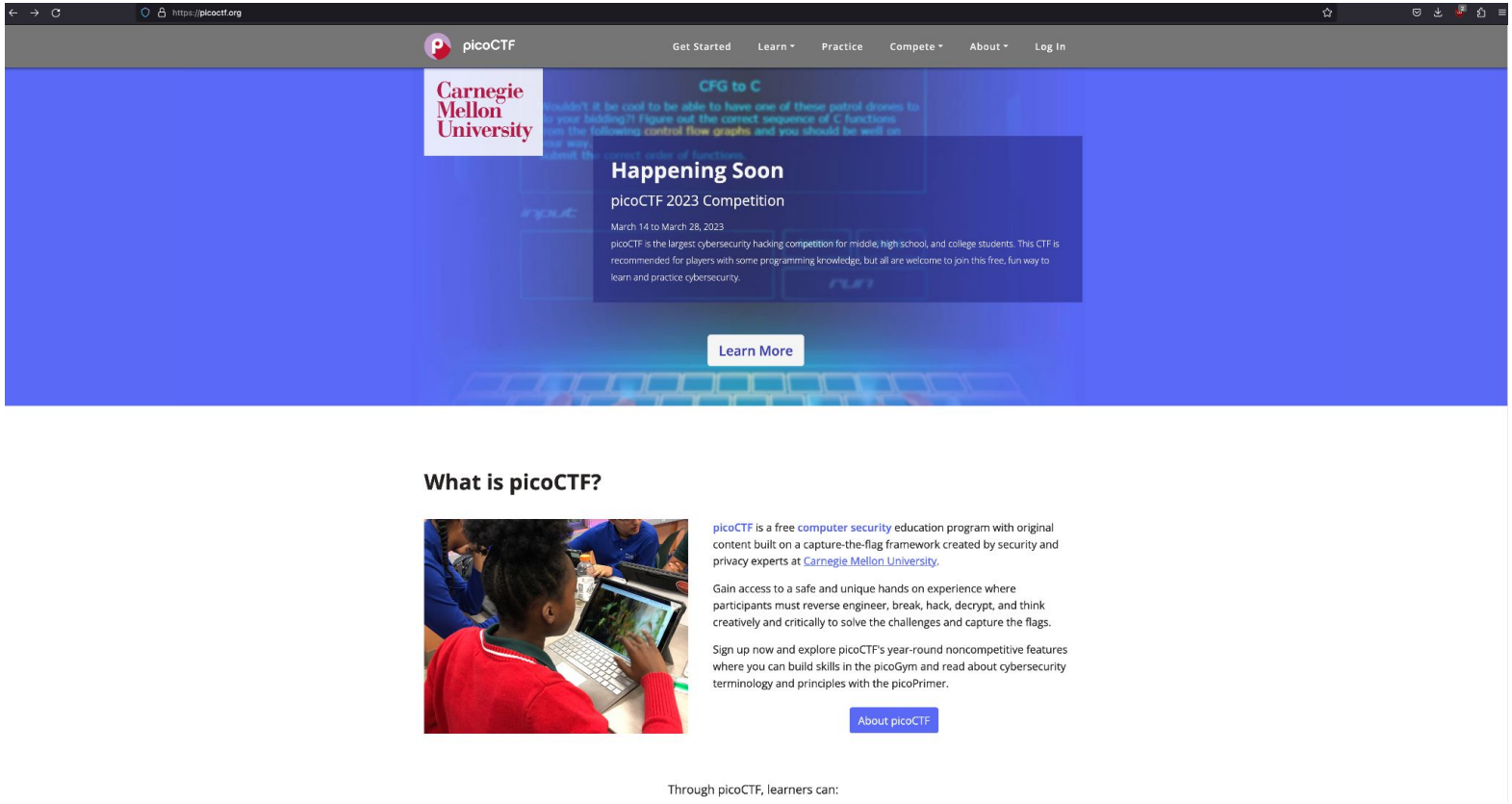
```
>>> for c in a:
...     print(chr(c ^ 0x23), end="")
...
fA }|~BIONMHLJF>>>
```

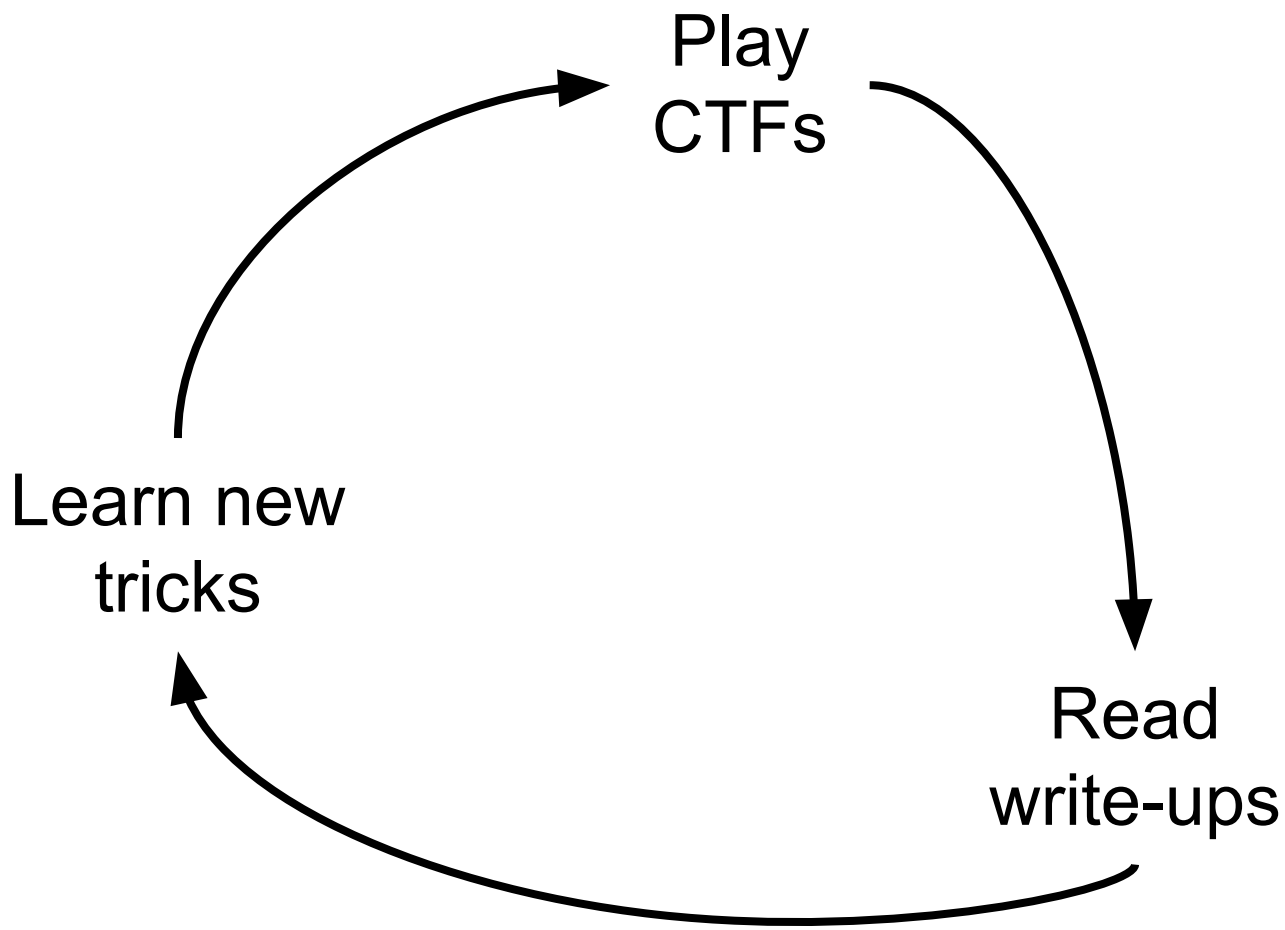


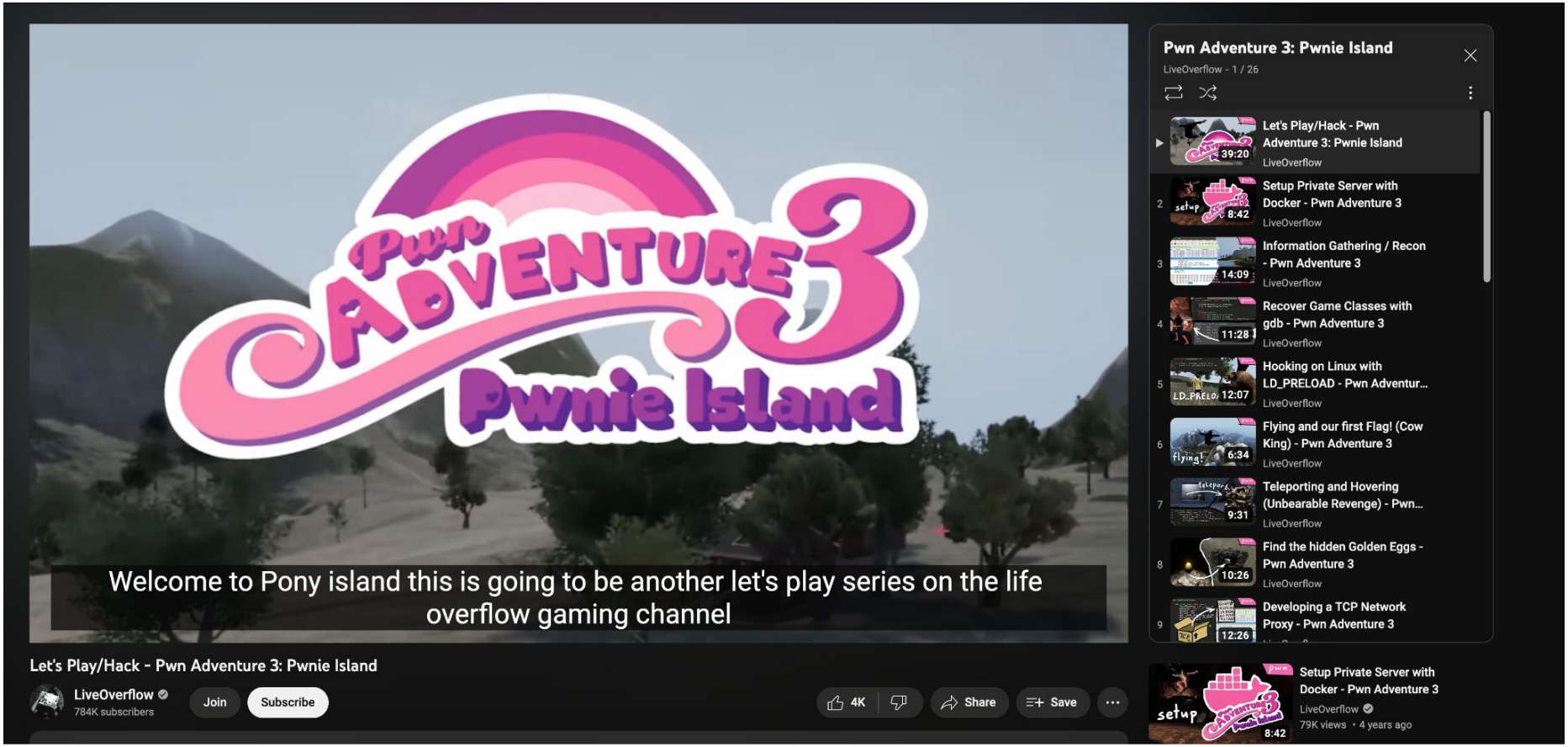
# Level 3

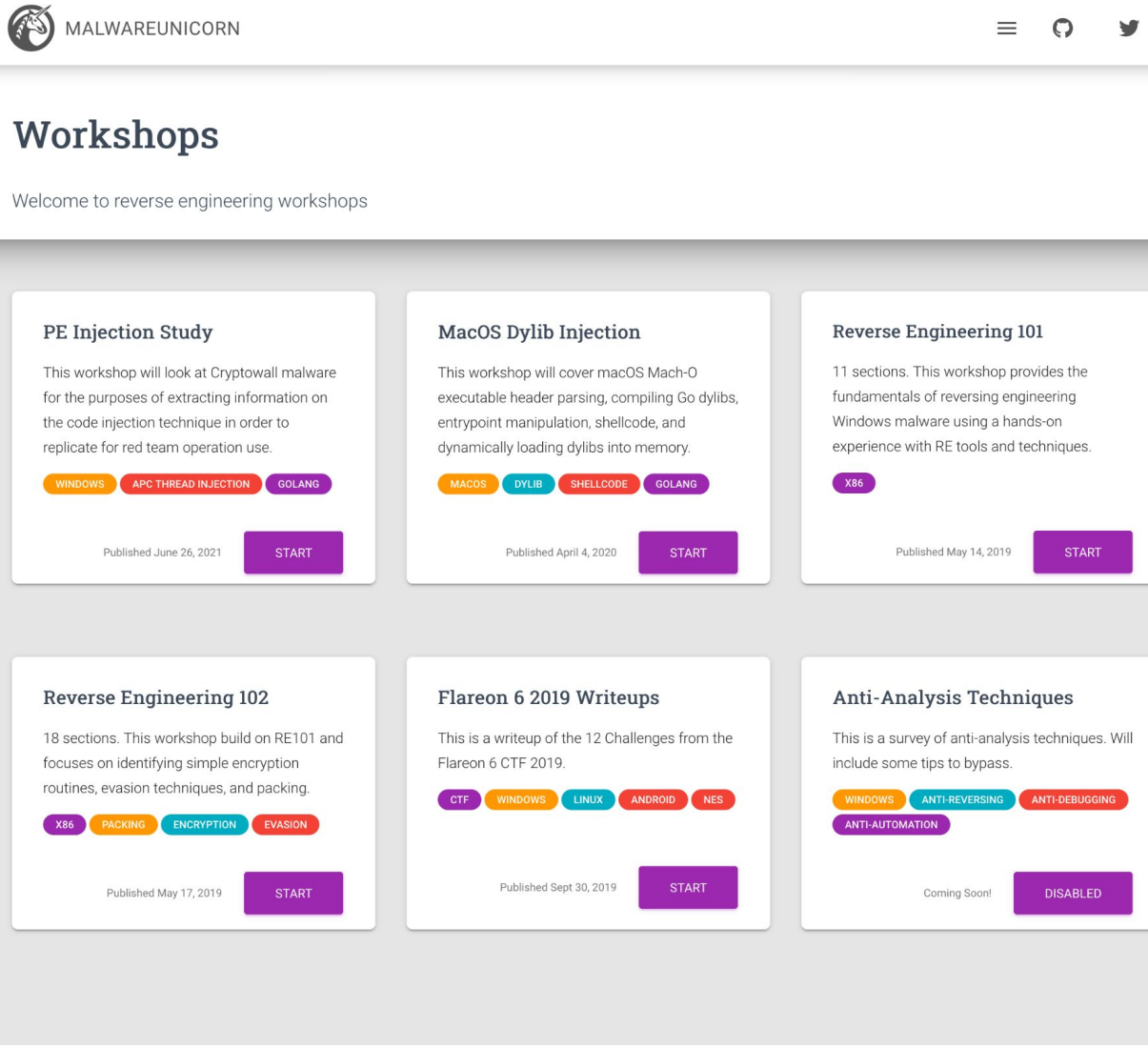

CodeBrowser: Rev101:/level3
File Edit Analysis Graph Navigation Search Select Tools Window Help
Program Trees
level3
.bss
.data
.got
.dynamic
.fini_array
.init_array
.eh_frame
.eh_frame_hdr
.rodata
Program Tree
Symbol Tree
Imports
<EXTERNAL>
__libc_start_main
Filter: main
Data Type Manager
Data Types
BuiltInTypes
level3
generic_clib_64
Filter:
Listing: level3
// segment_2.1
// Loadable segment [0x0 - 0x75f]
// ram:00100000-ram:00100317
assume DF = 0x0 (Default)
00100000 7f 45 4c Elf64_Ehdr
46 02 01
01 00 00 ...
00100000 7f db 7Fh e_ident_magi...
00100001 45 4c 46 ds "ELF" e_ident_magi...
00100004 02 db 2h e_ident_class
00100005 01 db 1h e_ident_data
00100006 01 db 1h e_ident_vers...
00100007 00 db 0h e_ident_osabi
00100008 00 db 0h e_ident_abiv...
00100009 00 00 00 00 00 db[7] e_ident_pad
00 00
00100010 03 00 dw 3h e_type
00100012 3e 00 dw 3Eh e_machine
00100014 01 00 00 00 ddw 1h e_version
00100018 00 11 00 00 00 dq FUN_00101100 e_entry
00 00 00
00100020 40 00 00 00 00 dq Elf64_Phdr_ARRAY_00100... e_phoff
00 00 00
00100028 48 31 00 00 00 dq Elf64_Shdr_ARRAY__elfS... e_shoff
00 00 00
00100030 00 00 00 00 ddw 0h e_flags
00100034 40 00 dw 40h e_ehsize
00100036 38 00 dw 38h e_phentsize
00100038 0d 00 dw Dh e_phnum
0010003a 40 00 dw 40h e_shentsize
0010003c 1d 00 dw 1Dh e_shnum
0010003e 1c 00 dw 1Ch e_shstrndx
Elf64_Phdr_ARRAY_00100040 XREF[2]: 00100020(*), 00100050(*)
00100040 06 00 00 Elf64_Ph... PT_PHDR - Program header
Decompiler
1 No Function
Console - Scripting
00100000
Elf64_Ehdr (64)

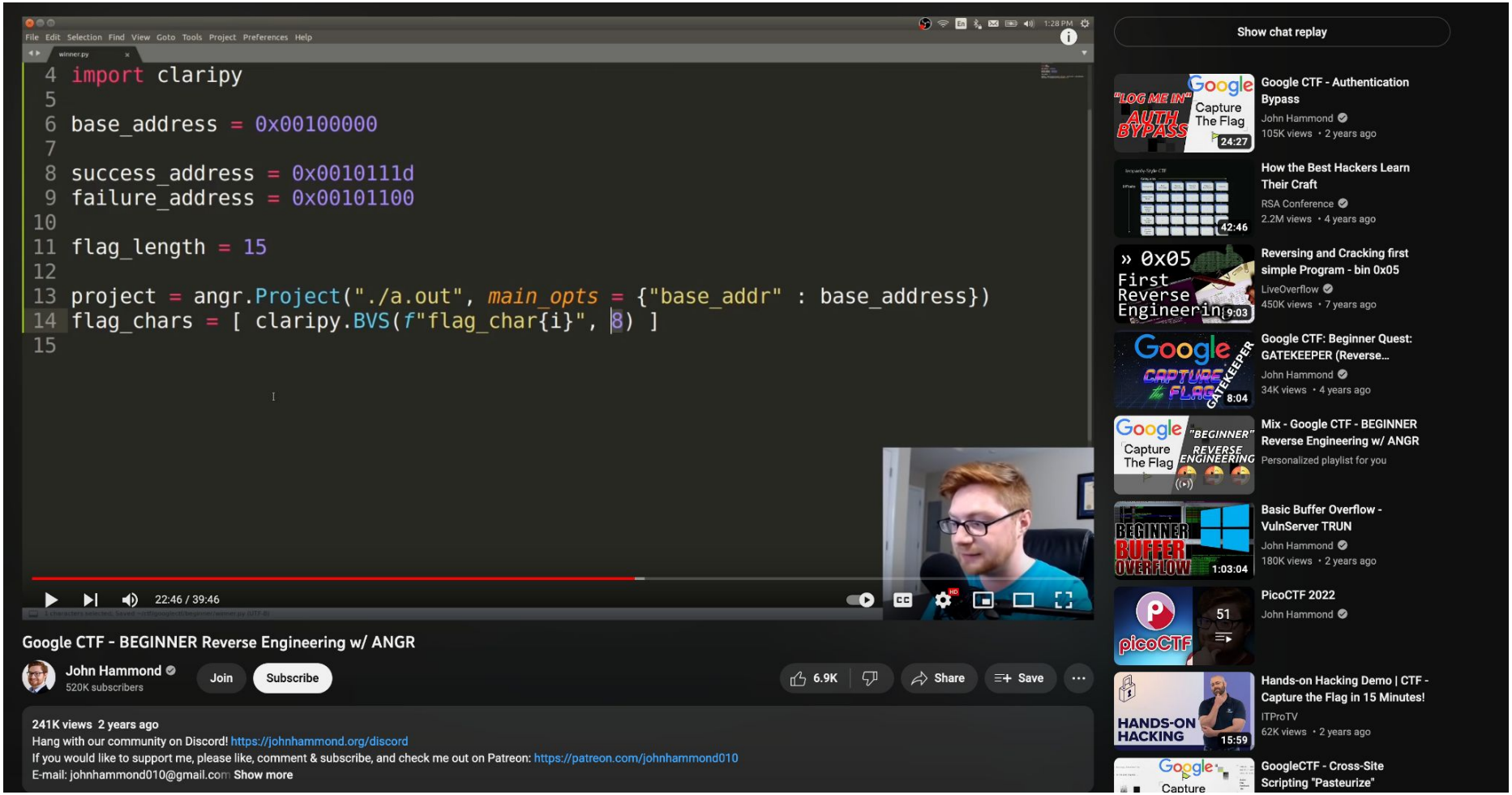
# Interested in Rev?




Play
CTFs
Read
write-ups
Learn new
tricks


Pwn ADVENTURE 3
Pwnie Island
Welcome to Pony island this is going to be another let's play series on the life overflow gaming channel
Let's Play/Hack - Pwn Adventure 3: Pwnie Island
LiveOverflow
784K subscribers
Join
Subscribe
4K
Share
Save
Pwn Adventure 3: Pwnie Island
LiveOverflow - 1 / 26
Let's Play/Hack - Pwn Adventure 3: Pwnie Island
39:20
LiveOverflow
2
Setup Private Server with Docker - Pwn Adventure 3
8:42
LiveOverflow
3
Information Gathering / Recon - Pwn Adventure 3
14:09
LiveOverflow
4
Recover Game Classes with gdb - Pwn Adventure 3
11:28
LiveOverflow
5
Hooking on Linux with LD_PRELOAD - Pwn Adventur...
12:07
LiveOverflow
6
Flying and our first Flag! (Cow King) - Pwn Adventure 3
6:34
LiveOverflow
7
Teleporting and Hovering (Unbearable Revenge) - Pwn...
9:31
LiveOverflow
8
Find the hidden Golden Eggs - Pwn Adventure 3
10:26
LiveOverflow
9
Developing a TCP Network Proxy - Pwn Adventure 3
12:26
Setup Private Server with Docker - Pwn Adventure 3
LiveOverflow
79K views • 4 years ago
8:42


MALWAREUNICORN
Workshops
Welcome to reverse engineering workshops
PE Injection Study
This workshop will look at Cryptowall malware for the purposes of extracting information on the code injection technique in order to replicate for red team operation use.
WINDOWS
APC THREAD INJECTION
GOLANG
Published June 26, 2021
START
MacOS Dylib Injection
This workshop will cover macOS Mach-O executable header parsing, compiling Go dylibs, entrypoint manipulation, shellcode, and dynamically loading dylibs into memory.
MACOS
DYLIB
SHELLCODE
GOLANG
Published April 4, 2020
START
Reverse Engineering 101
11 sections. This workshop provides the fundamentals of reversing engineering Windows malware using a hands-on experience with RE tools and techniques.
X86
Published May 14, 2019
START
Reverse Engineering 102
18 sections. This workshop build on RE101 and focuses on identifying simple encryption routines, evasion techniques, and packing.
X86
PACKING
ENCRYPTION
EVASION
Published May 17, 2019
START
Flareon 6 2019 Writeups
This is a writeup of the 12 Challenges from the Flareon 6 CTF 2019.
CTF
WINDOWS
LINUX
ANDROID
NES
Published Sept 30, 2019
START
Anti-Analysis Techniques
This is a survey of anti-analysis techniques. Will include some tips to bypass.
WINDOWS
ANTI-REVERSING
ANTI-DEBUGGING
ANTI-AUTOMATION
Coming Soon!
DISABLED


File Edit Selection Find View Goto Tools Project Preferences Help
winner.py
4 import claripy
5
6 base_address = 0x00100000
7
8 success_address = 0x0010111d
9 failure_address = 0x00101100
10
11 flag_length = 15
12
13 project = angr.Project("./a.out", main_opts = {"base_addr" : base_address})
14 flag_chars = [ claripy.BVS(f"flag_char{i}", 8) ]
15
22:46 / 39:46
Google CTF - BEGINNER Reverse Engineering w/ ANGR
John Hammond
520K subscribers
Join
Subscribe
6.9K
Share
Save
241K views 2 years ago
Hang with our community on Discord! https://johnhammond.org/discord
If you would like to support me, please like, comment & subscribe, and check me out on Patreon: https://patreon.com/johnhammond010
E-mail: johnhammond010@gmail.com Show more
Show chat replay
Google CTF - Authentication Bypass
John Hammond
105K views • 2 years ago
How the Best Hackers Learn Their Craft
RSA Conference
2.2M views • 4 years ago
Reversing and Cracking first simple Program - bin 0x05
LiveOverflow
450K views • 7 years ago
Google CTF: Beginner Quest: GATEKEEPER (Reverse...
John Hammond
34K views • 4 years ago
Mix - Google CTF - BEGINNER Reverse Engineering w/ ANGR
Personalized playlist for you
Basic Buffer Overflow - VulnServer TRUN
John Hammond
180K views • 2 years ago
PicoCTF 2022
John Hammond
Hands-on Hacking Demo | CTF - Capture the Flag in 15 Minutes!
ITProTV
62K views • 2 years ago
GoogleCTF - Cross-Site Scripting "Pasteurize"