CSC 405
Reverse Engineering, Ghidra

Follow Along: go.ncsu.edu/rev101

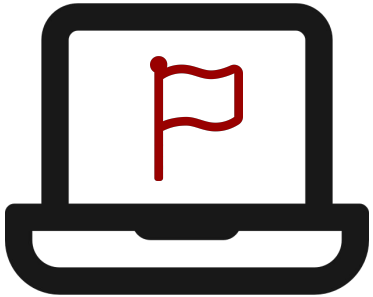
Adam Gaweda
agaweda@ncsu.edu

Alexandros Kapravelos
akaprav@ncsu.edu

Alex Napetyan
anahape@ncsu.edu

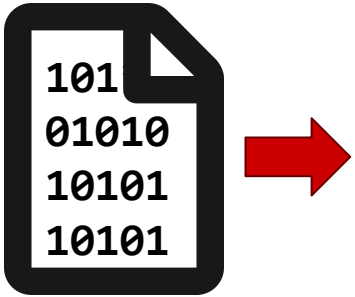
Thinking like a CTF

Designed around finding a **secret** in a binary/website that can be discovered through **exploiting** a **vulnerability**



Thinking like a CTF

Designed around finding a **secret** in a binary/website that can be discovered through **exploiting** a **vulnerability**



"flag{YouSolvedTheChallenge!}"

"Typically" in the form of a string format
`term{challenge_passcode}`

```
$ objdump -zd example
0000000000401275 <main>:
401275:  f3 0f 1e fa     endbr64
401279:  55              push   %rbp
40127a:  48 89 e5        mov    %rsp,%rbp
40127d:  48 83 ec 20     sub    $0x20,%rsp
401281:  89 7d ec        mov    %edi,-0x14(%rbp)
401284:  48 89 75 e0     mov    %rsi,-0x20(%rbp)
401288:  48 8b 45 e0     mov    -0x20(%rbp),%rax
40128c:  48 83 c0 08     add    $0x8,%rax
401290:  48 8b 00        mov    (%rax),%rax
401293:  48 89 c7        mov    %rax,%rdi
401296:  e8 e5 fd ff ff  call   401080 <atoi@plt>
40129b:  89 45 fc        mov    %eax,-0x4(%rbp)
40129e:  8b 45 fc        mov    -0x4(%rbp),%eax
4012a1:  89 c7           mov    %eax,%edi
4012a3:  e8 ce fe ff ff  call   401176 <function>
4012a8:  b8 00 00 00 00  mov    $0x0,%eax
4012ad:  c9              leave
4012ae:  c3              ret
```

Problem is reading hexadecimal and machine code is **incredibly difficult**!

Desig et in a binar vered thro ility

- Released in March 2019
- Developed by the NSA
  - Declassified after leak on WikiLeaks
- Open Source
  - https://github.com/NationalSecurityAgency/ghidra
- In development for ~20 years
  - History of Ghidra
- Scripting in Java and Python
- Headless Analyzer
- Ghidra Cheat Sheet
- Walkthrough of Solving a Simple Reverse Engineering Challenge

https://ghidra-sre.org/

Installing Ghidra

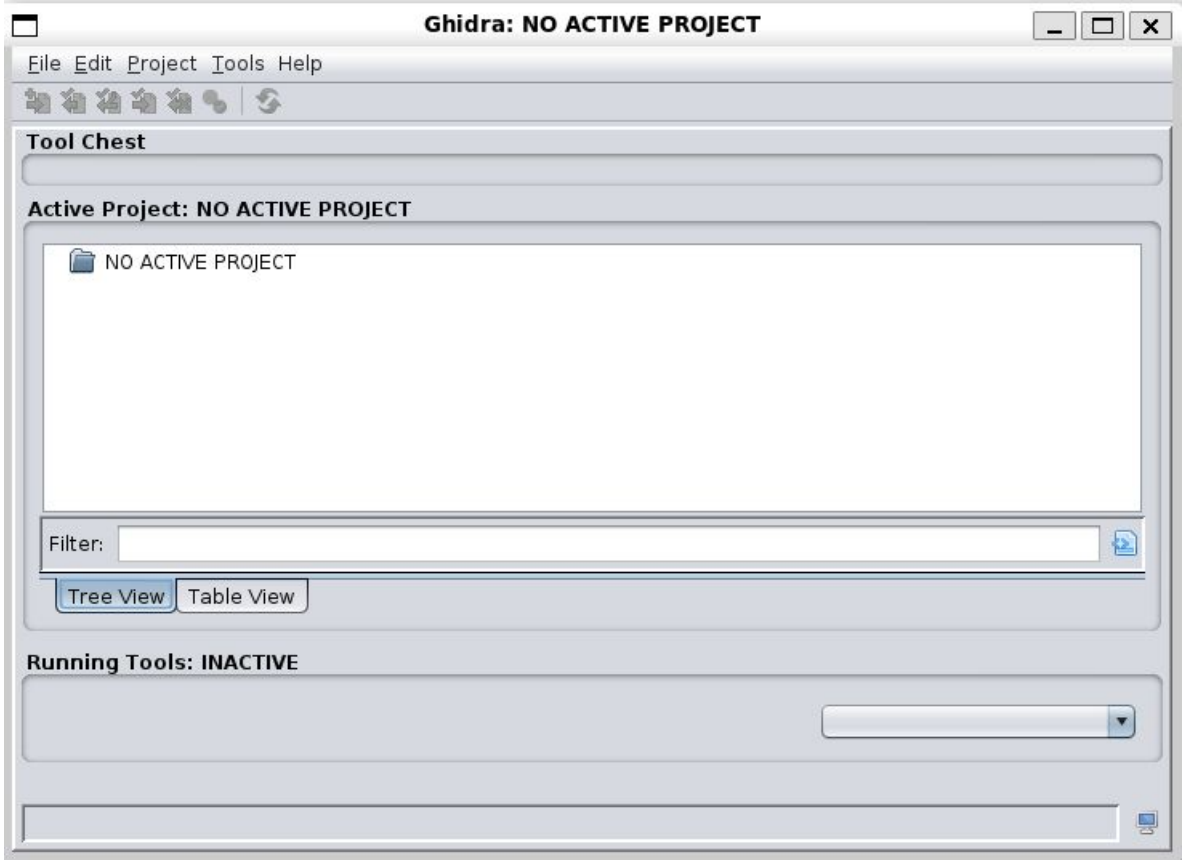
1. Grab the latest version of Ghidra

2. Install Ghidra via Gradle
   a. `gradle -I gradle/support/fetchDependencies.gradle init`
   b. If you need to install Java 17 and Gradle:
      i. `sudo apt install openjdk-17-jdk`
      ii. `sudo apt install gradle`

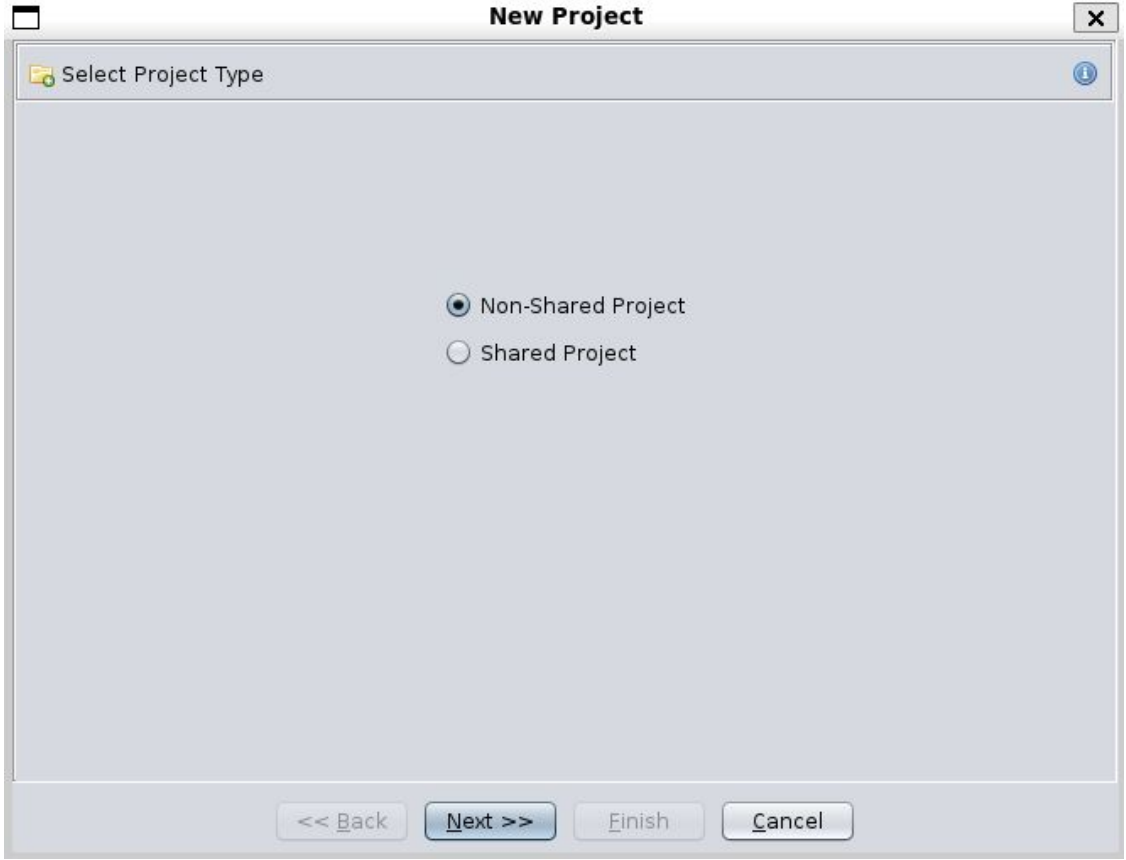
2.1 Install Ghidra on macOS via brew
   a. `brew install --cask temurin`
   b. `brew install --cask ghidra`

3. Run Ghidra via `./ghidraRun` or `ghidraRun.bat`
   a. Don't run `./ghidraRun` through WSL, has weird interactions; use the `.bat` version instead

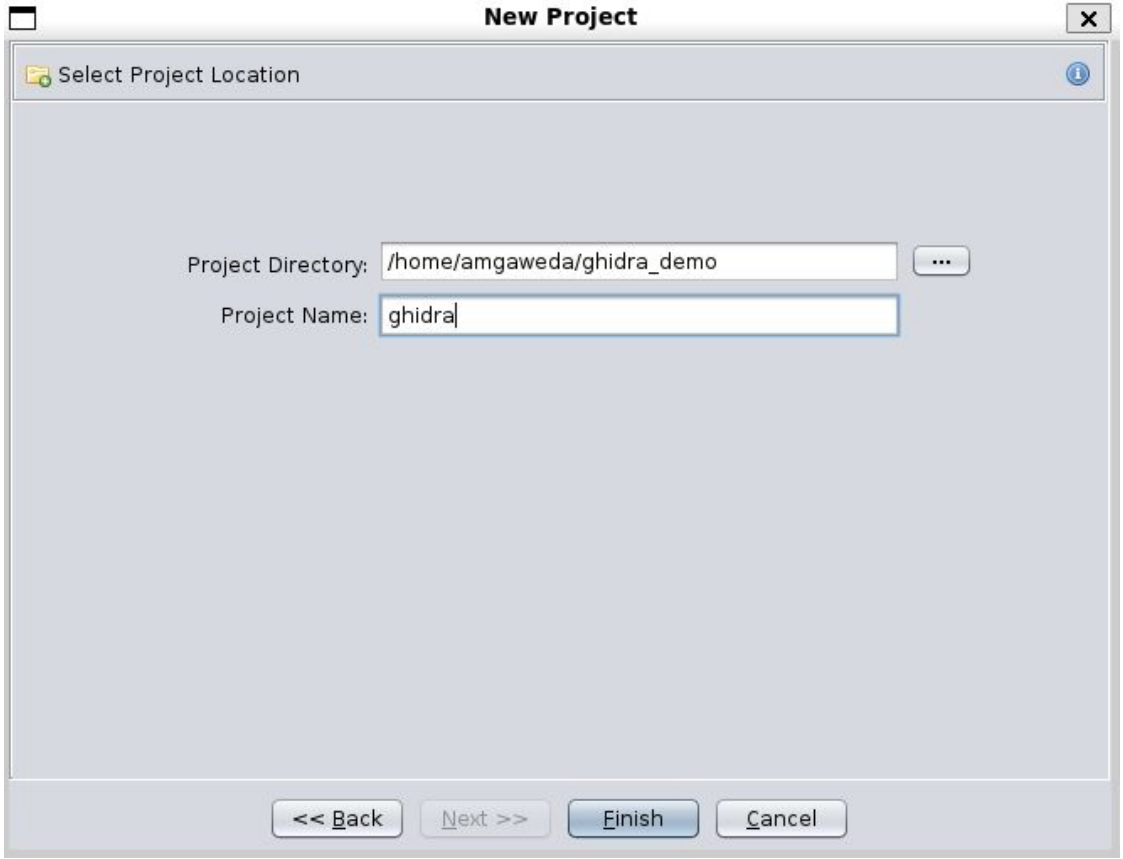
Running Ghidra - Starting a Project



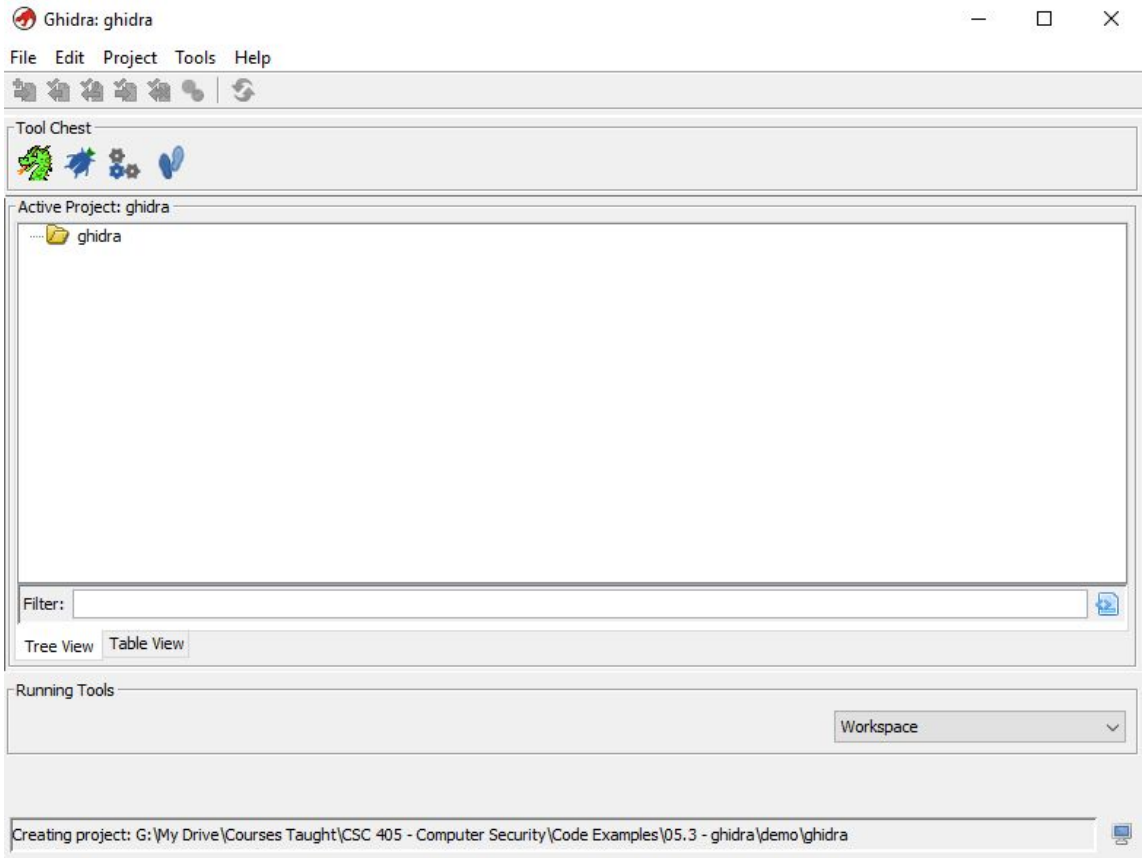
Running Ghidra - Starting a Project



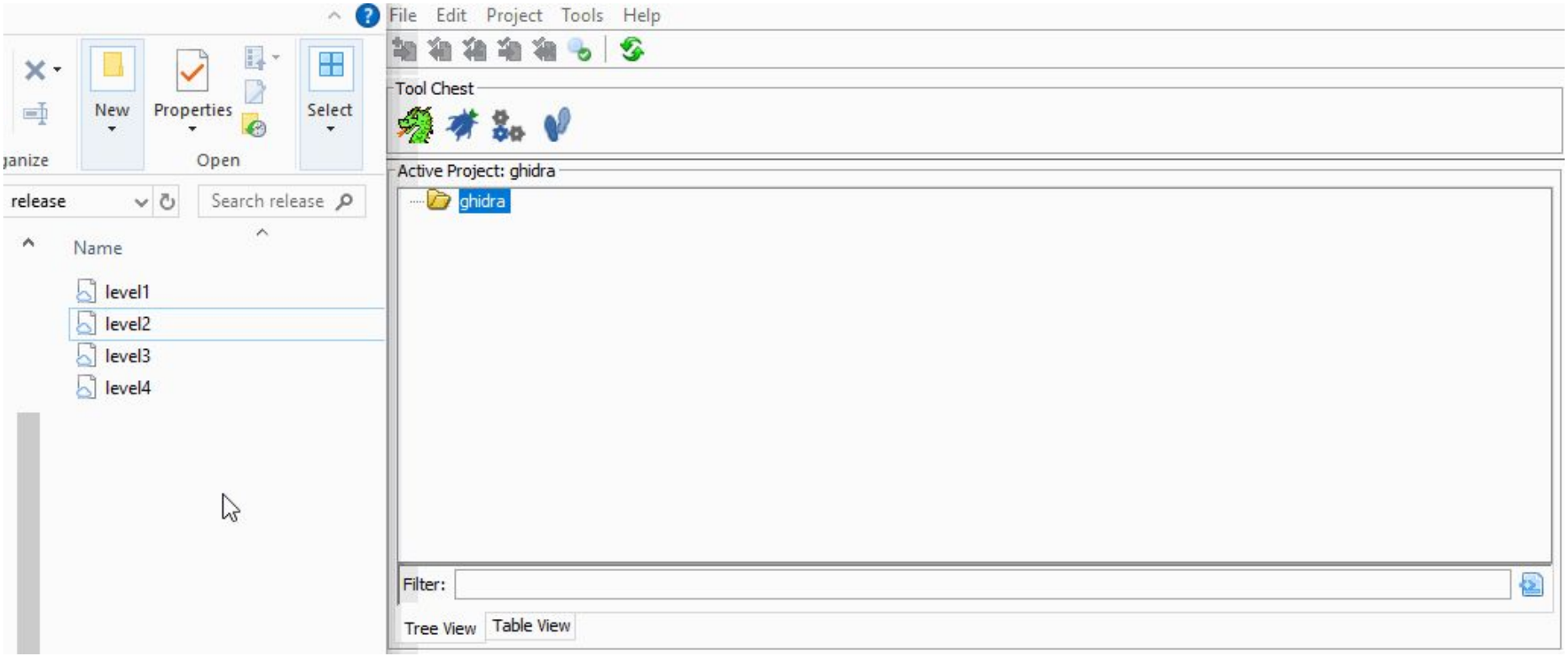
Running Ghidra - Starting a Project



Running Ghidra - Loading Binaries



Running Ghidra - Loading Binaries



Running Ghidra - Loading Binaries

Batch Import

Import Sources

/G:/My Drive/Courses Taught/CSC 405 - Computer Security/Code Examples/05.3 - ghidra/release/level2 [0 files/1 apps/0 containers/0 levels]
/G:/My Drive/Courses Taught/CSC 405 - Computer Security/Code Examples/05.3 - ghidra/release/level3 [0 files/1 apps/0 containers/0 levels]
/G:/My Drive/Courses Taught/CSC 405 - Computer Security/Code Examples/05.3 - ghidra/release/level4 [0 files/1 apps/0 containers/0 levels]
/G:/My Drive/Courses Taught/CSC 405 - Computer Security/Code Examples/05.3 - ghidra/release/level1 [0 files/1 apps/0 containers/0 levels]

Add
Remove

Depth limit: 2 Rescan

Files to Import

File Type	Loader	Language	Files
☑	Executable and Linking Format (ELF)	x86:LE:64:default:gcc*	4 files...

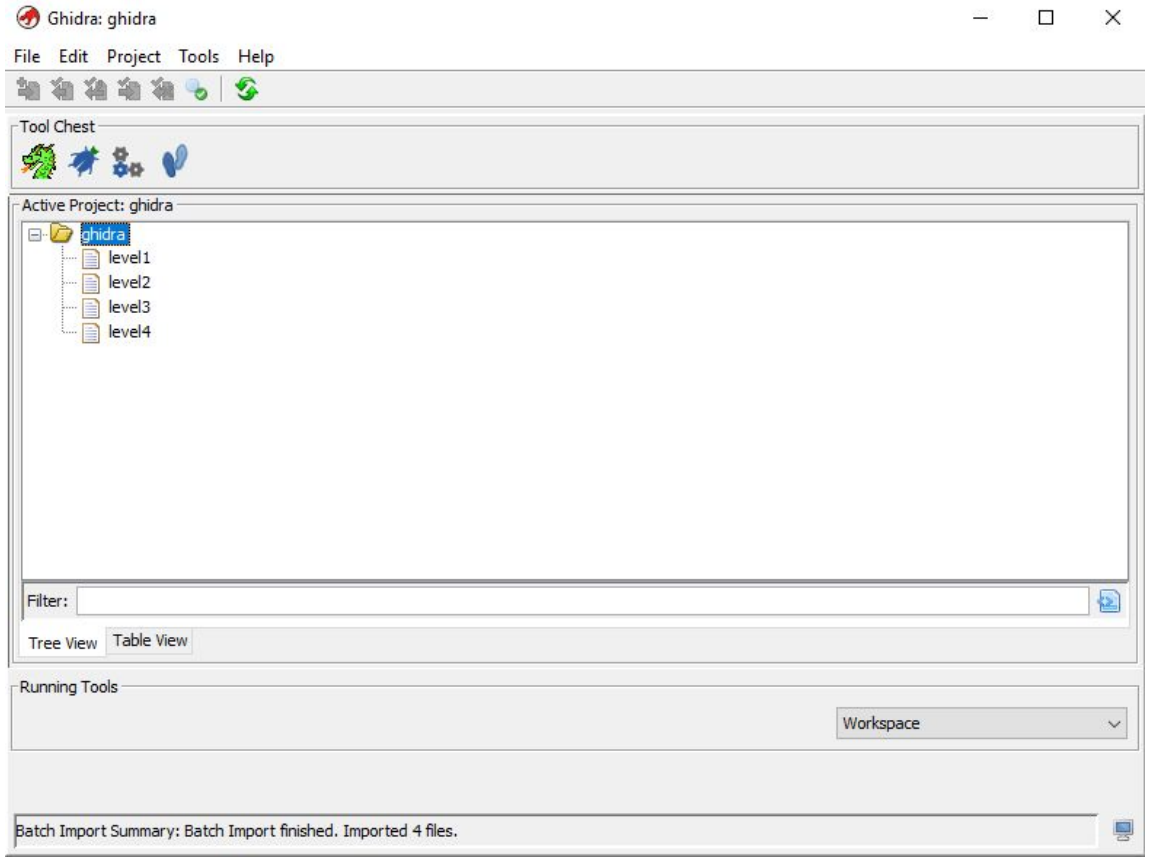
Import Options

☑ Strip leading path ☐ Strip container paths

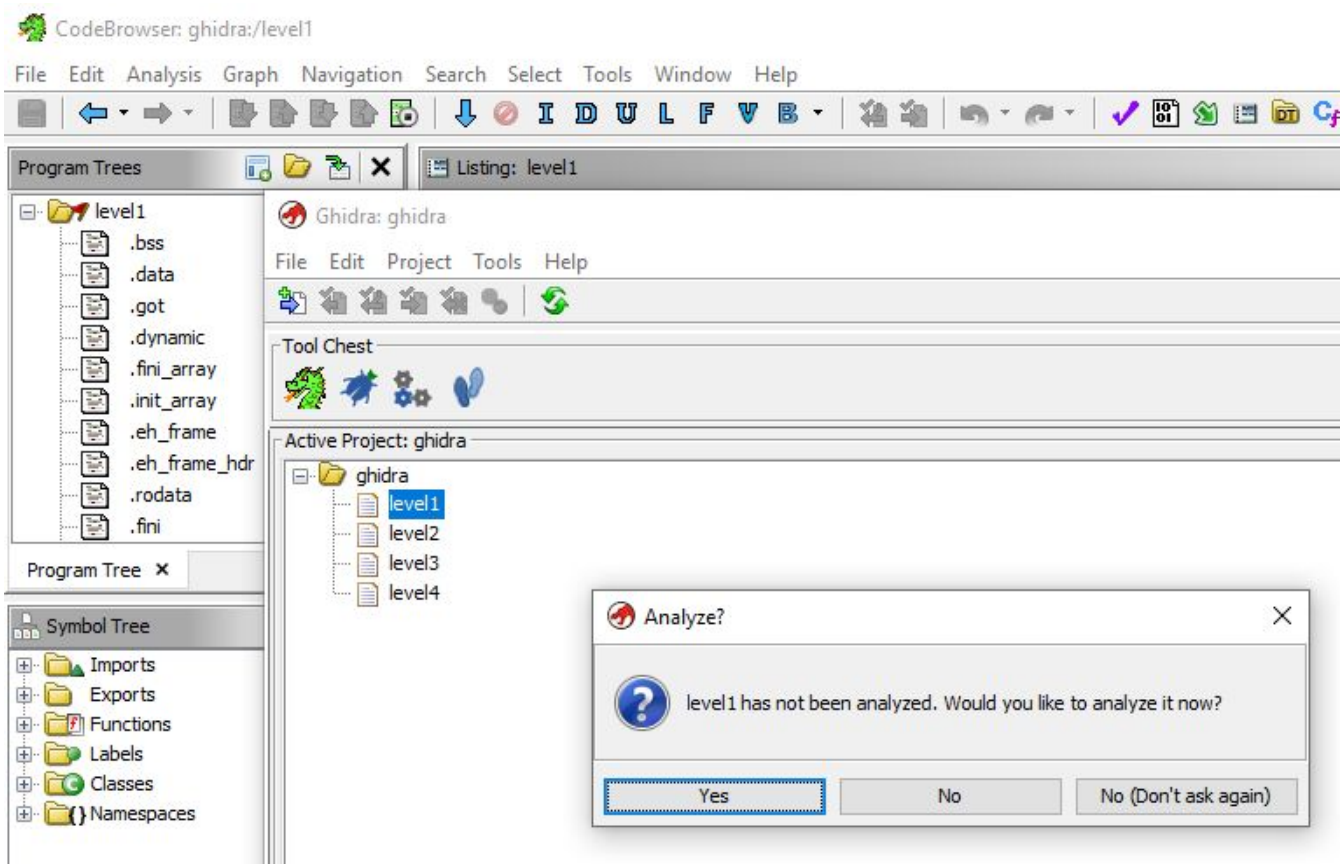
Destination Folder ghidra:/ ...

OK Cancel

Running Ghidra - Loading Binaries



Running Ghidra - Analyzing Level 1



Running Ghidra - Analyzing Level 1

CodeBrowser: ghidra:/level1

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

level1
- .bss
- .data
- .got

Program Tree

Symbol Tree

Exports

Functions
- __do_global_dtc
- __libc_csu_fini
- __libc_csu_init
- _fini
- _init
- _start
- deregister_tm_c
- frame_dummy
- FUN_00101020
- FUN_00101080
- main
- register_tm_clor

Filter:

Data Type Man...

Data Types
- BuiltInTypes
- level1
- generic_clib_64

Filter:

Listing: level1

```
001011bf 00          ??          00h

                     **************************************************
                     *                    THUNK FUNCTION
                     **************************************************
                     thunk undefined frame_dummy()
                       Thunked-Function: register_tm_clones
     undefined         AL:1          <RETURN>
                     frame_dummy


001011c0 f3 0f 1e fa     ENDBR64
001011c4 e9 77 ff        JMP         register_tm_clones
         ff ff
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     **************************************************
                     *                      FUNCTION
                     **************************************************
                     int __stdcall main(int argc, char * * argv)
     int               EAX:4         <RETURN>
     int               EDI:4         argc
     char * *          RSI:8         argv
     undefined8        Stack[-0x10]:8 local_10

     char[40]          Stack[-0x38]... password
```

Decompile: main - (level1)

```
int main(int argc,char **argv)

{
  long lVar1;
  int iVar2;
  long in_FS_OFFSET;
  char **argv-local;
  int argc-local;
  char password [40];

  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
  printf("Enter the password: ");
  __isoc99_scanf(&DAT_00102019,password);
  iVar2 = strcmp(password,"workshop{SuperSecret}");
  if (iVar2 == 0) {
    puts("Correct password");
    iVar2 = 0;
  }
  else {
    puts("Incorrect password");
    iVar2 = 2;
  }
  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not r
    __stack_chk_fail();
```

Console - Scripting

001011c9 | main | ENDBR64

Running Ghidra - Analyzing Level 1

CodeBrowser: ghidra:/level1

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

level1
- .bss
- .data
- .got

Program Tree

Symbol Tree

Exports

Functions
- __do_global_dtc
- __libc_csu_fini
- __libc_csu_init
- _fini
- _init
- _start
- deregister_tm_c
- frame_dummy
- FUN_00101020
- FUN_00101080
- main
- register_tm_clor

Filter:

Data Type Man...

Data Types
- BuiltInTypes
- level1
- generic_clib_64

Filter:

Listing: level1

```
001011bf 00          ??          00h

                     **********************************************
                     *                THUNK FUNCTION
                     **********************************************
                     thunk undefined frame_dummy()
                       Thunked-Function: register_tm_clones
     undefined         AL:1          <RETURN>
                     frame_dummy


001011c0 f3 0f 1e fa     ENDBR64
001011c4 e9 77 ff        JMP        register_tm_clones
         ff ff
                     -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                     **********************************************
                     *                FUNCTION
                     **********************************************
                     int __stdcall main(int argc, char * * argv)
     int               EAX:4         <RETURN>
     int               EDI:4         argc
     char * *          RSI:8         argv
     undefined8        Stack[-0x10]:8 local_10

     char[40]          Stack[-0x38]... password
```

Decompile: main - (level1)

```
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
iVar2 = strcmp(password,"workshop{SuperSecret}");
if (iVar2 == 0) {
  puts("Correct password");
```

```
12  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
13  printf("Enter the password: ");
14  __isoc99_scanf(&DAT_00102019,password);
15  iVar2 = strcmp(password,"workshop{SuperSecret}");
16  if (iVar2 == 0) {
17    puts("Correct password");
18    iVar2 = 0;
19  }
20  else {
21    puts("Incorrect password");
22    iVar2 = 2;
23  }
24  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
25                    /* WARNING: Subroutine does not r
26    __stack_chk_fail();
```

Console - Scripting

001011c9 | main | ENDBR64

Running Ghidra - Analyzing Level 1

```
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
iVar2 = strcmp(password,"workshop{SuperSecret}");
if (iVar2 == 0) {
  puts("Correct password");
```

Okay, that one was super simple, even `strings` could have found it...

Running Ghidra - Analyzing Level 2

```
lVar1 = *(long *)(in_FS_OFFSET + 0x28);
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
__s2 = flag();
iVar2 = strcmp(password,__s2);
if (iVar2 == 0) {
  printf("Correct password");
  iVar2 = 0;
}
else {
  printf("Incorrect password");
  iVar2 = 3;
}
```

Same program, but now the flag is obfuscated

Running Ghidra - Analyzing Level 2

```
lVar1 = *(long *)(in_FS_OFFSET + 0x28);
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
__s2 = flag();
iVar2 = strcmp(password,__s2);
if (iVar2 == 0) {
  printf("Correct password");
  iVar2 = 0;
}
else {
  printf("Incorrect password");
  iVar2 = 3;
}
```

```
char * flag(void)

{
  undefined8 *__s;
  size_t sVar1;
  int i;
  char *flag;

  __s = (undefined8 *)malloc(0x28);
  *__s = 0x534c4b5048514c54;
  __s[1] = 0x707c514653567058;
  __s[2] = 0x55466f5746514046;
  *(undefined4 *)(__s + 3) = 0x5e114f46;
  *(undefined *)((long)__s + 0x1c) = 0;
  i = 0;
  while( true ) {
    sVar1 = strlen((char *)__s);
    if (sVar1 <= (ulong)(long)i) break;
    *(byte *)((long)__s + (long)i) = *(byte *)((long)__s + (long)i) ^ 0x23;
    i = i + 1;
  }
  return (char *)__s;
```

But if I dig a little deeper, we can observe what this `flag` function is really doing

Running Ghidra - Analyzing Level 2

```
lVar1 = *(long *)(in_FS_OFFSET + 0x28);
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
__s2 = flag();
iVar2 = strcmp(password,__s2);
if (iVar2 == 0) {
  printf("Correct password");
  iVar2 = 0;
}
else {
  printf("Incorrect password");
  iVar2 = 3;
}
```

```
char * flag(void)

{
  undefined8 *__s;
  size_t sVar1;
  int i;
  char *flag;

  __s = (undefined8 *)malloc(0x28);
  *__s = 0x534c4b5048514c54;
  __s[1] = 0x707c514653567058;
  __s[2] = 0x55466f5746514046;
  *(undefined4 *)(__s + 3) = 0x5e114f46;
  *(undefined *)((long)__s + 0x1c) = 0;
  i = 0;
  while( true ) {
    sVar1 = strlen((char *)__s);
    if (sVar1 <= (ulong)(long)i) break;
    *(byte *)((long)__s + (long)i) = *(byte *)((long)__s + (long)i) ^ 0x23;
    i = i + 1;
  }
  return (char *)__s;
```

Ghidra IS having some trouble understanding this part, which is why it's labeled as "undefined"

Running Ghidra - Analyzing Level 2

```
lVar1 = *(long *)(in_FS_OFFSET + 0x28);
printf("Enter the password: ");
__isoc99_scanf(&DAT_00102019,password);
__s2 = flag();
iVar2 = strcmp(password,__s2);
if (iVar2 == 0) {
  printf("Correct password");
  iVar2 = 0;
}
else {
  printf("Incorrect password");
  iVar2 = 3;
}
```

```
char * flag(void)

{
  undefined8 *__s;
  size_t sVar1;
  int i;
  char *flag;

  __s = (undefined8 *)malloc(0x28);
  *__s = 0x534c4b5048514c54;
  __s[1] = 0x707c514653567058;
  __s[2] = 0x55466f5746514046;
  *(undefined4 *)(__s + 3) = 0x5e114f46;
  *(undefined *)((long)__s + 0x1c) = 0;
  i = 0;
  while( true ) {
    sVar1 = strlen((char *)__s);
    if (sVar1 <= (ulong)(long)i) break;
    *(byte *)((long)__s + (long)i) = *(byte *)((long)__s + (long)i) ^ 0x23;
    i = i + 1;
  }
  return (char *)__s;
```

But whatever it is, the binary seems to loop through those hex values and then **XOR** (^) them with another fixed hex value (**`0x23`**)

Solving Level 2 with GDB

Since we know the function names in the binary, we can use **gdb** to call that function directly

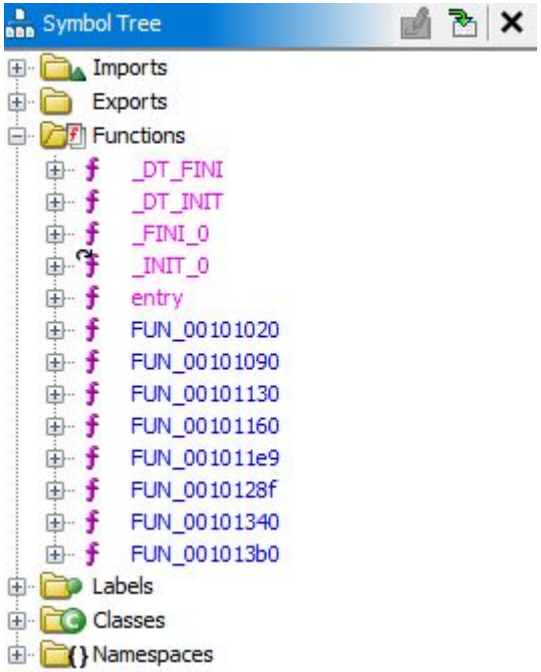
```
$ gdb level2
Reading symbols from level2...
(gdb) b main
Breakpoint 1 at 0x12a2: file src/level2.c, line 20.
(gdb) r
Starting program: /path/to/level2
[Thread debugging using libthread_db enabled]
Using host libthread_db library
"/lib/x86_64-linux-gnu/libthread_db.so.1".
Breakpoint 1, main (argc=1, argv=0x7fffffffda48) at src/level2.c:20
20      src/level2.c: No such file or directory.
(gdb) call (char*)flag()
$1 = 0x5555555592a0 "workshop{Super_SecretLevel2}"
```

Solving Level 2 with Python

Alternatively, we can take those raw hex values from Ghidra and write a short Python script to decrypt them

```python
from struct import pack, unpack

parts = [
  0x534c4b5048514c54,
  0x707c514653567058,
  0x55466f5746514046,
  0x5e114f46
]

# < means little endian byte order
# Q means unsigned long integer (8 bytes)
flag = b"".join([ pack("<Q", hex_chunk) for hex_chunk in parts])
# trim null bytes
flag = flag.replace(b"\x00", b"")

decrypted_flag = ""
for char in flag:
  # XOR each byte with 0x23 to get password
  decrypted_flag += chr(char ^ 0x23)

print(decrypted_flag)
```

Running Ghidra - Analyzing Level 3



NC STATE UNIVERSITY

Static Techniques

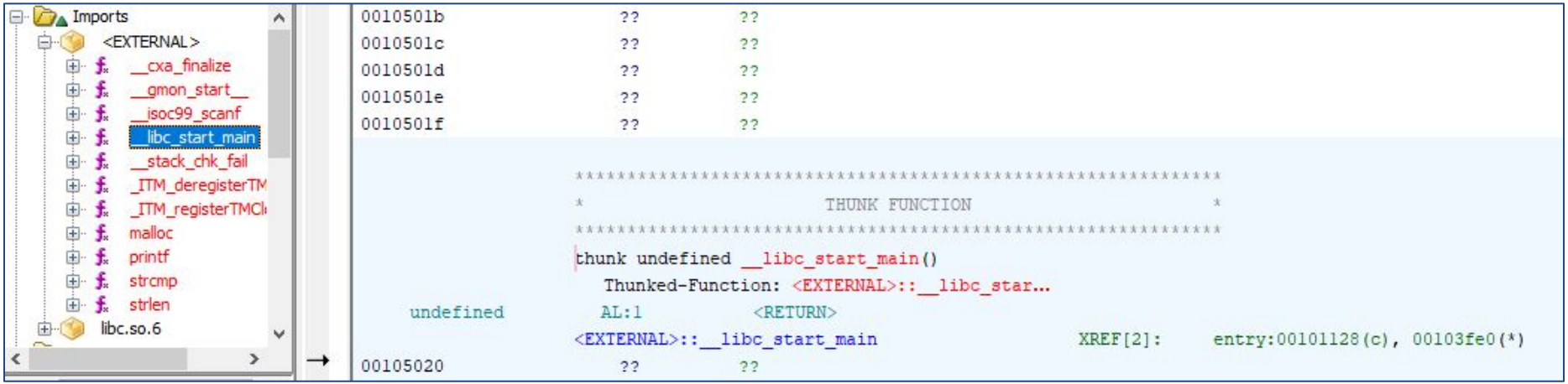
Displaying program symbols

("T": The symbol is in the text (code) section)

```
$ nm example | grep " T"
00000000004010c0 T _dl_relocate_static_pie
00000000004012b0 T _fini
0000000000401000 T _init
0000000000401090 T _start
0000000000401176 T function
0000000000401275 T main
$ strip example
$ nm example | grep " T"
nm: example: no symbols
```

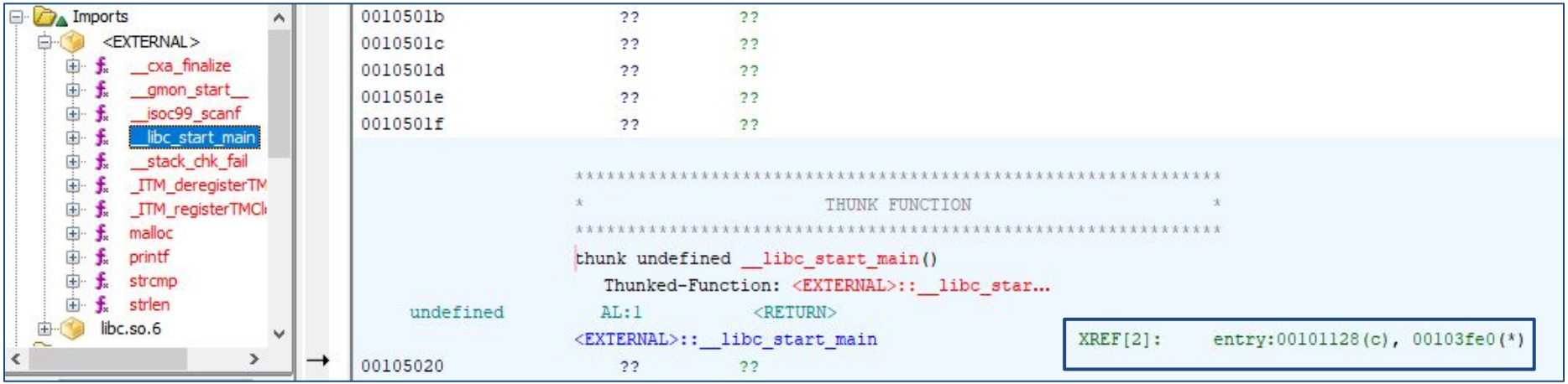
Oh no! Someone used **`strip`** to remove function identifiers!

Running Ghidra - Analyzing Level 3



Luckily, all programs need a starting point, which we can figure out via **`__libc_start_main`**

Running Ghidra - Analyzing Level 3



This reference points us to the program's starting function

Running Ghidra - Analyzing Level 3

```
00101106 49 89 d1        MOV        R9,RDX
00101109 5e              POP        RSI
0010110a 48 89 e2        MOV        RDX,RSP
0010110d 48 83 e4 f0     AND        RSP,-0x10
00101111 50              PUSH       RAX
00101112 54              PUSH       RSP=>local_10
00101113 4c 8d 05        LEA        R8,[FUN_001013b0]
         96 02 00 00
0010111a 48 8d 0d        LEA        RCX,[FUN_00101340]
         1f 02 00 00
00101121 48 8d 3d        LEA        RDI,[FUN_0010128f]
         67 01 00 00
00101128 ff 15 b2        CALL       qword ptr [-><EXTERNAL>::__libc_start_main]      undefine
         2e 00 00                                                                    = 00105C
```

```
 2 void processEntry entry(undefined8 param_1,undefined8 param_2)
 3
 4 {
 5   undefined auStack_8 [8];
 6
 7   __libc_start_main(FUN_0010128f,param_2,&stack0x00000008,FUN_00101340,FUN
 8                     auStack_8);
 9   do {
10                     /* WARNING: Do nothing block with infinite loop */
11   } while( true );
12 }
13
```

Jumping to that memory address, we can see what it's doing

Running Ghidra - Analyzing Level 3

```
00101106 49 89 d1        MOV        R9,RDX
00101109 5e              POP        RSI
0010110a 48 89 e2        MOV        RDX,RSP
0010110d 48 83 e4 f0     AND        RSP,-0x10
00101111 50              PUSH       RAX
00101112 54              PUSH       RSP=>local_10
00101113 4c 8d 05        LEA        R8,[FUN_001013b0]
         96 02 00 00
0010111a 48 8d 0d        LEA        RCX,[FUN_00101340]
         1f 02 00 00
00101121 48 8d 3d        LEA        RDI,[FUN_0010128f]
         67 01 00 00
00101128 ff 15 b2        CALL       qword ptr [-><EXTERNAL>::__libc_start_main]      undefine
         2e 00 00                                                                    = 001050
```

```
 2 void processEntry entry(undefined8 param_1,undefined8 param_2)
 3
 4 {
 5   undefined auStack_8 [8];
 6
 7   __libc_start_main(FUN_0010128f,param_2,&stack0x00000008,FUN_00101340,FUN
 8                     auStack_8);
 9   do {
10                     /* WARNING: Do nothing block with infinite loop */
11   } while( true );
12 }
13
```

Specifically, we can see libc's main is being called on function pointer **`FUN_0010128f`**

Running Ghidra - Analyzing Level 3

```
undefined8 FUN_0010128f(void)

{
  int iVar1;
  char *__s2;
  undefined8 uVar2;
  long in_FS_OFFSET;
  char local_38 [40];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  printf("Enter the password: ");
  __isoc99_scanf(&DAT_00102019,local_38);
  __s2 = (char *)FUN_001011e9();
  iVar1 = strcmp(local_38,__s2);
  if (iVar1 == 0) {
    printf("Correct password");
    uVar2 = 0;
  }
  else {
    printf("Incorrect password");
    uVar2 = 3;
  }
```

```
undefined8 * FUN_001011e9(void)

{
  undefined8 *__s;
  size_t sVar1;
  int local_24;

  __s = (undefined8 *)malloc(0x28);
  *__s = 0x534c4b5048514c54;
  __s[1] = 0x707c514653567058;
  __s[2] = 0x55466f5746514046;
  *(undefined4 *)(__s + 3) = 0x5e114f46;
  *(undefined *)((long)__s + 0x1c) = 0;
  local_24 = 0;
  while( true ) {
    sVar1 = strlen((char *)__s);
    if (sVar1 <= (ulong)(long)local_24) br
    *(byte *)((long)__s + (long)local_24)
    local_24 = local_24 + 1;
  }
  return __s;
```

And we're back to our decryption function

Want More Practice on Reverse Engineering?

picoCTF

Get Started | Learn | Practice | Compete | About | Log In

Carnegie Mellon University

CFG to C

Wouldn't it be cool to be able to have one of these patrol drones to do your bidding?! Figure out the correct sequence of C functions from the following control flow graphs and you should be well on your way.

Submit the correct order of functions.

input

run

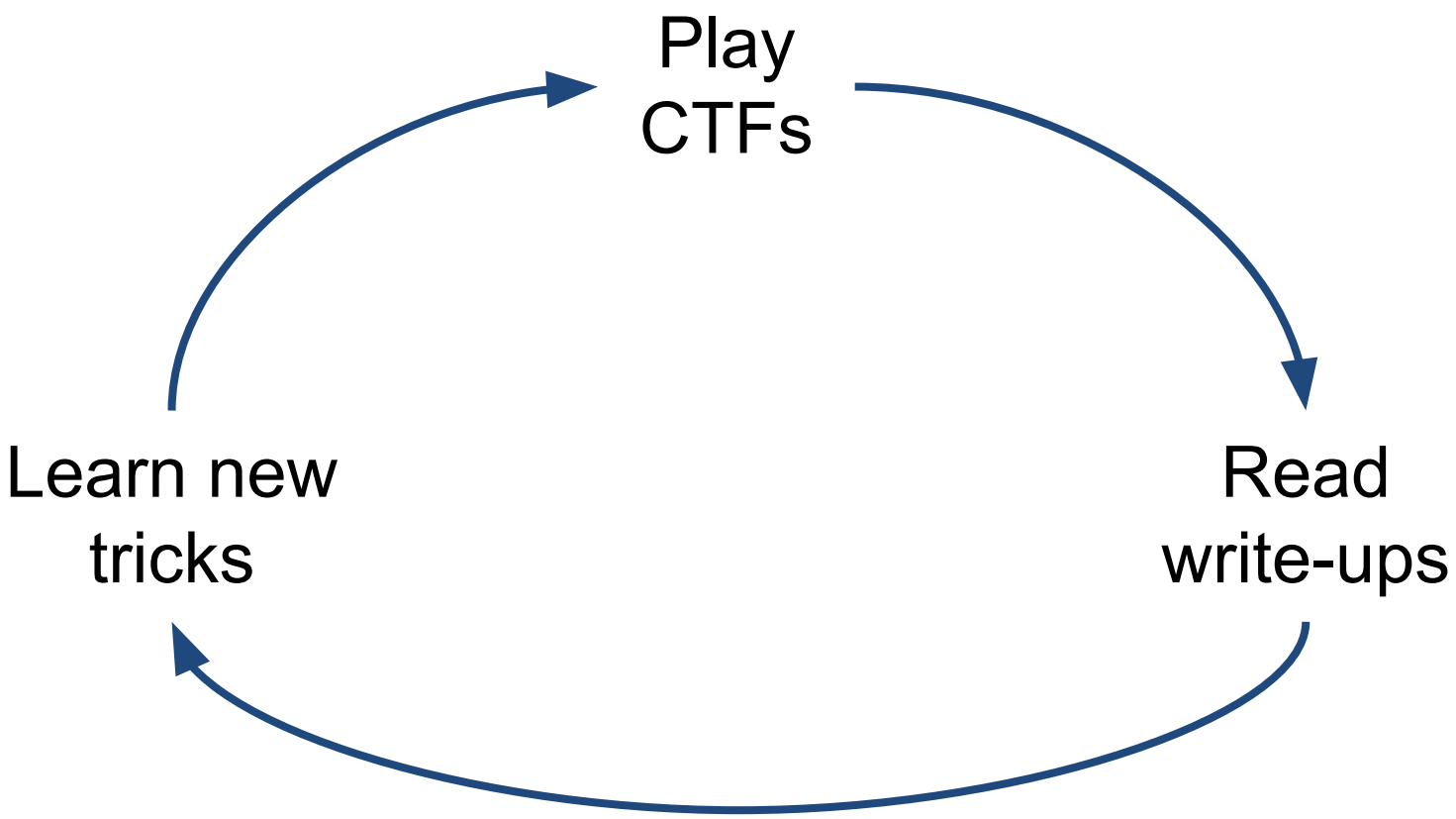
picoCTF 2024

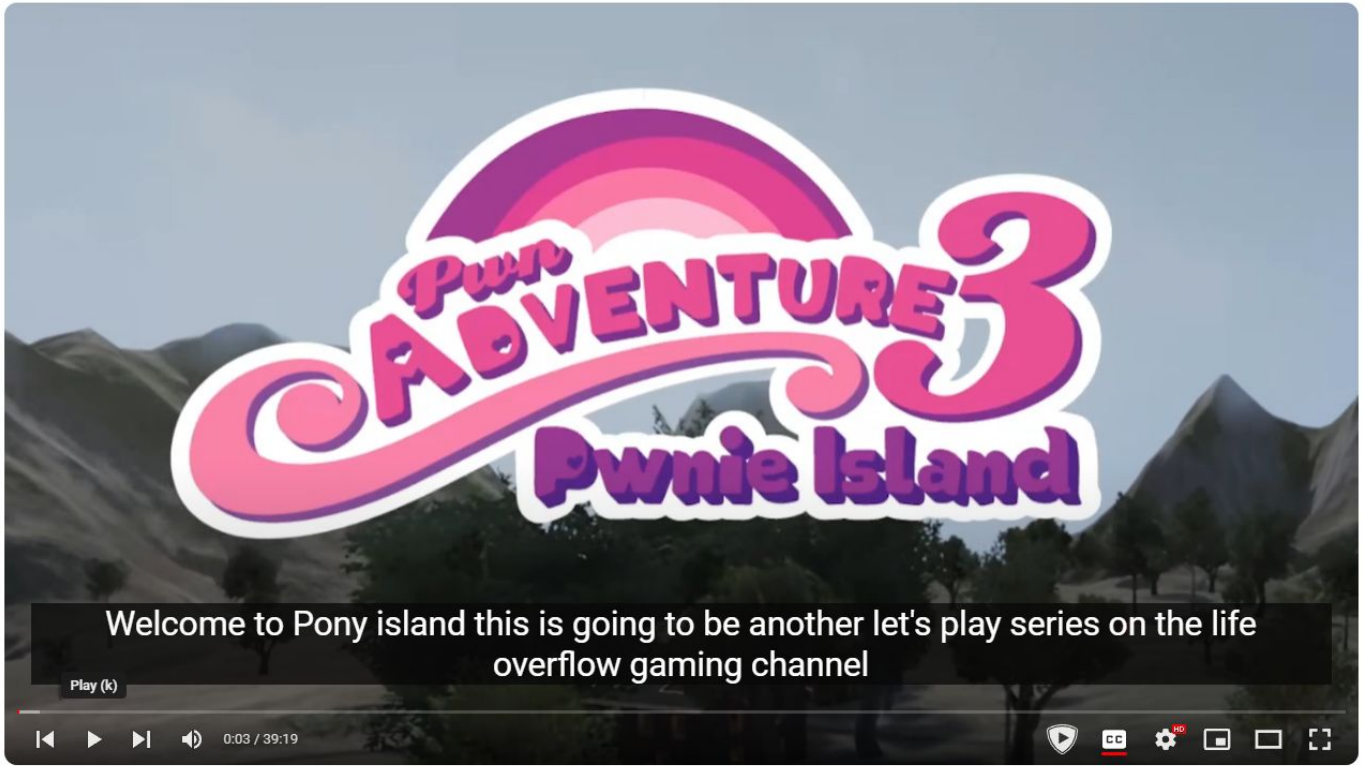
March 12 to March 26, 2024

picoCTF 2024 is a two-week competitive CTF open to anyone, with prizes available to eligible teams.

Register Now | Learn More

https://picoctf.org/


Play CTFs
Read write-ups
Learn new tricks



Playlist Link

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

Flareon 6 2019 Writeups

This is a writeup of the 12 Challenges from the Flareon 6 CTF 2019.

CTF | WINDOWS | LINUX | ANDROID | NES

Anti-Analysis Techniques

This is a survey of anti-analysis techniques. Will include some tips to bypass.

WINDOWS | ANTI-REVERSING | ANTI-DEBUGGING | ANTI-AUTOMATION

MalwareUnicorn.org

Google CTF: Beginner Quest

John Hammond

10 videos 46,736 views Last updated on Jun 29, 2018

Play all Shuffle

1 

Google CTF: Beginner Quest: FLOPPY (File Carving with Binwalk)

John Hammond • 33K views • 5 years ago

2 

Google CTF: Beginner Quest: MOAR

John Hammond • 18K views • 5 years ago

3 

Google CTF: Beginner Quest: OCR IS COOL! (Simple Cryptography)

John Hammond • 43K views • 5 years ago

4 

Google CTF: Beginner Quest: ADMIN UI (Local File Inclusion)

John Hammond • 22K views • 5 years ago

5 

Google CTF: Beginner Quest: SECURITY BY OBSCURITY (ZIP Archive Compression)

John Hammond • 31K views • 5 years ago

YouTube Playlist

Security Zen - At least Taylor Swift's BF Won

Rest of Class - Level 4 Practice

```
void main(void)

{
  long lVar1;
  int iVar2;
  long in_FS_OFFSET;
  char input [40];

  lVar1 = *(long *)(in_FS_OFFSET + 0x28);
  memset(input,0,0x28);
  printf("Enter the password: ");
  __isoc99_scanf(&DAT_00102034,input);
  iVar2 = check_flag(input);
  if (iVar2 == 0) {
    printf("Correct password");
  }
  else {
    printf("Incorrect password");
  }
  if (lVar1 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```