UCL CYBER SECURITY SOCIETY

UNDERSTANDING BINARIES WITH GHIDRA

27 November 2024

OVERVIEW

1. Binary Analysis
2. Binary Files
3. Assembly
4. Ghidra
5. Example

BINARY ANALYSIS

BINARY ANALYSIS

- Process of examining the properties of binary files
    - Instructions
    - Data
- Learn more about program's purpose
- **Static analysis**:
  examine the executable binary without running it
- **Dynamic analysis**:
  observe the program as it executes
- Static analysis is usually just the first step

WHY?

- Examine behavior of executable files without source code
  → third-party libraries, drivers, and other system components
  → source code unavailable
- Discover bugs and security vulnerabilities
- Game cheat/exploit development
- Malware analysis
  → detection
  → characteristics
  → authorship attribution
- Digital forensics
  → looking for information, not exploits

HOW?

- file
- binwalk
- readelf and ldd
- xdd
- strings
- strace and ltrace
- Ghidra, IDA Pro and radare2

STEPS

- Environment setup
  → VMs or containers for safety
  → install essential tools
- Inspect binary (file and readelf)
- Disassemble (Ghidra or IDA Pro)
- Trace execution (gdb)
- Identify vulnerabilities (e.g. strcpy or sprint)
- Test inputs and observe results

EVADING TECHNIQUES

- Obfuscation
  → hide explicit values
  → conceal logic
- Anti-debugging
  → checks if debugger is attached to process
  → locks down program
- Stripping symbols

CHALLENGES

- No symbolic information
  → no relevance at binary level
  → often stripped of symbols
  → hard to understand
- No type information
  → variable types are never explicitly stated
- No high-level constructs (e.g. classes)
  → huge blobs of code and data rather than well-structured programs
- Minor modification could break binary
- Cannot fully recover source code

REVERSE ENGINEERING

- Binary analysis:
  understanding the “what”
- Reverse engineering:
  understanding the “how” and “why”
- Binary analysis is a subset of skills used in RE
- RE often starts with binary analysis

CTF VS REAL-WORLD

Zero-trust: everything is self-reliant, no third-party software

	CTF RE	Real-World RE
Purpose	Solve well-defined challenges	Understand undocumented, custom systems
Complexity	Simple environments	Complex enterprise systems
Security	Known protocols	Proprietary security mechanisms
Documentation	Challenges may have hints or references	Black-boxed systems with no public documentation
Tools	Standard RE tools	Standard + advanced and custom tools

Binary Files

BINARY FILES

- Compiler: source code → machine code
- Stores data as sequence of bytes
- Not human readable
- Meant to be processed by computer’s processor
- File types: executable, library, database, …

EXECUTABLE FORMATS

- ELF (Executable and Linkable Format): Linux
  → no file extension
- PE (Portable Executable): Windows
  → file extension .exe
- Mach-O: Mac
  → no file extension

FILE STRUCTURE

- Header: metadata, e.g. architecture, entry point, type
- Text section: code
- Data section: initialised data
- BSS section: uninitialised data
- Segments: memory-mapped parts used during execution
- Symbol table: maps function names and variables to addresses
  → in unstripped binaries
- Tools: readelf, objdump, strings

ASSEMBLY

BASICS

- Low-level programming language specific to particular computer architecture
- Converted to machine code using assembler
- You should be familiar with
  - Registers
  - Data sizes (word, double word, …)
  - Binary and hexadecimal number system
  - Addressing data in memory

CPU ARCHITECTURES

- Assembly instructions vary by architecture
  → e.g. x86: mov, ARM: ldr
- Binaries are compiled for specific CPUs
  → must match CPU's instruction set to run correctly
- Common architectures
  - x86/x64: common on desktops
  - ARM: common on mobile and embedded devices
  - MIPS: common in IoT and some hardware
  - RISC-V

GHIDRA

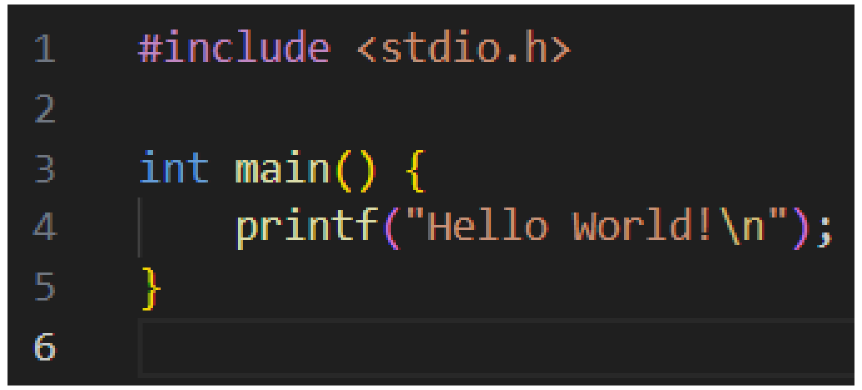
WHAT IS GHIDRA

- Reverse engineering tool
- Developed by the NSA
- Free
- Open-source
- Used to analyse compiled binaries
- Decompilation: produce approximate source code
- Disassembly: construct assembly from machine code

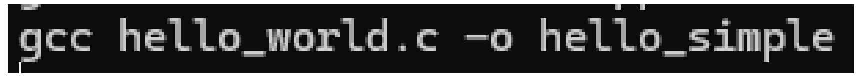
OTHER TOOLS

- IDA Pro
  - Better UI
  - More functionality
  - Not free
- Radare2

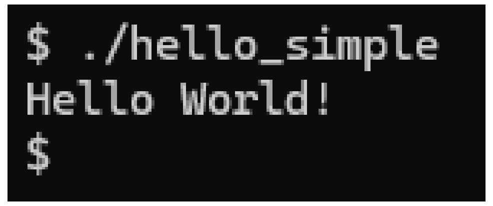
HELLO WORLD



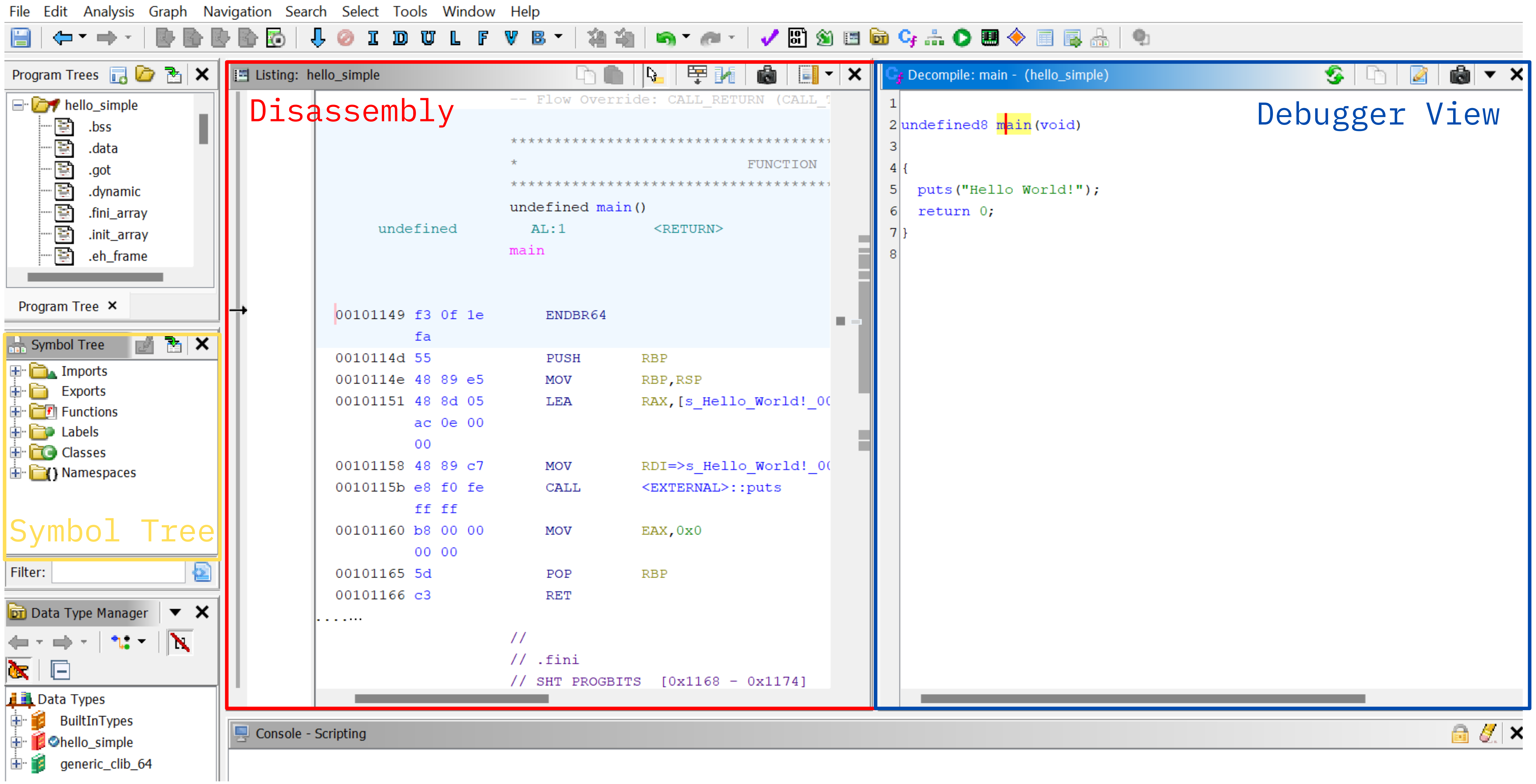
```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
}
```



```
gcc hello_world.c -o hello_simple
```



```
$ ./hello_simple
Hello World!
$
```



```
                          -- Flow Override: CALL_RETURN (CALL_
                          ************************************
                          *                           FUNCTION
                          ************************************
                          undefined main()
      undefined             AL:1           <RETURN>
                          main
00101149 f3 0f 1e      ENDBR64
         fa
0010114d 55            PUSH       RBP
0010114e 48 89 e5      MOV        RBP,RSP
00101151 48 8d 05      LEA        RAX,[s_Hello_World!_00
         ac 0e 00
         00
00101158 48 89 c7      MOV        RDI=>s_Hello_World!_00
0010115b e8 f0 fe      CALL       <EXTERNAL>::puts
         ff ff
00101160 b8 00 00      MOV        EAX,0x0
         00 00
00101165 5d            POP        RBP
00101166 c3            RET
.....
                          //
                          // .fini
                          // SHT_PROGBITS  [0x1168 - 0x1174]
```

```c
undefined8 main(void)

{
  puts("Hello World!");
  return 0;
}
```

EXAMPLE

EXAMPLE

- Import binary
- Identify entry point and main function
- Follow function calls and control flow
- Use decompiler to read higher level code
- Label functions and variables for clarity
- Identify strings and global variables
- Detect basic anti-analysis techniques
- Patching binary

THANK YOU FOR COMING!!