Introduction  ary Analysis

Based on Chapter 1 of Andriesse's "Practical Binary Analysis"

Slides prepared by H. Gunadi

Outline

- Loading and execution of binary, virtual memory layout.
- Compilation process.
- Introduction to the x86 assembly.
- Basic disassembly and reverse-engineering tools.

Learning Outcomes

- Understand the relationship between binary, loaded process, and source.
- Familiarity with some basic tools for analysing binary and processes.
- Understand a high-level structure of binary.

Why binary analysis

- All programs need to be compiled down to machine code.
- Correctness/security gap: how do we know the compilation is correct, preserves security?
- What you think your (high-level) program does vs what it actually does (on the binary level).

Scope

- Focus on Linux binary (ELF) only, but similar principles apply to Windows binary (PE).

- Focus on x86 instruction set architecture only.

Binary Analysis

- Static analysis
  - Analysing a bi[illegible]out running it.
  - Can analyse the binary in one go; platform independent – since you don't need to run it.
  - Less precise – no knowledge of runtime state. Generally undecidable.

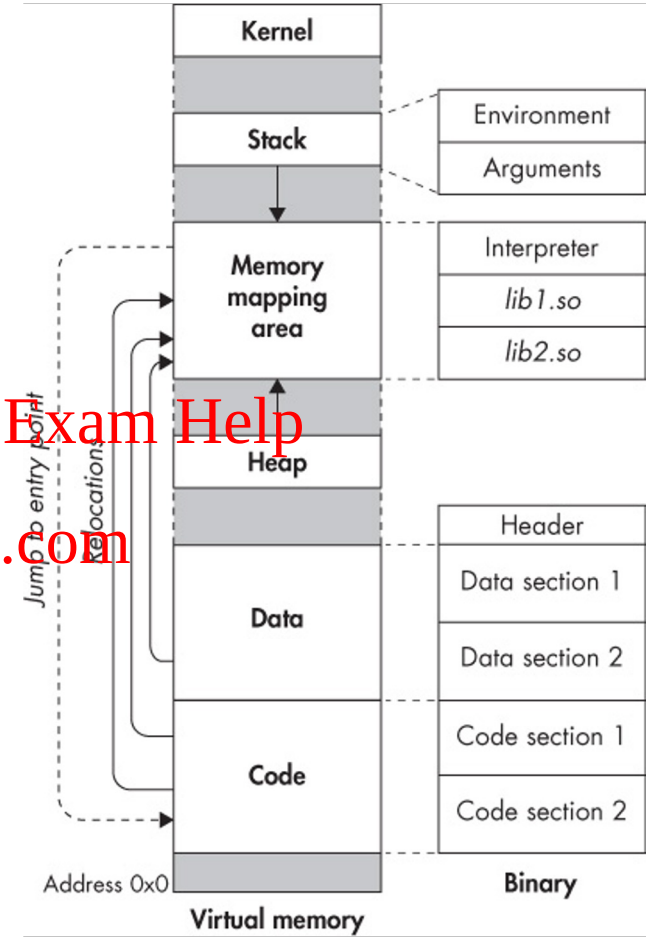
- Dynamic analysis
  - Runs the binary as it executes.
  - Access to entire system states – easier to analyse, more precise.
  - But may miss some parts of the code – you only see some particular runs, not all possible runs of the binary.

Challenges in binary analysis

- No symbolic info
- No type information
- No high-level abstraction
- Mixed code and data
- Location dependent data and code

Loading and Executing Binary

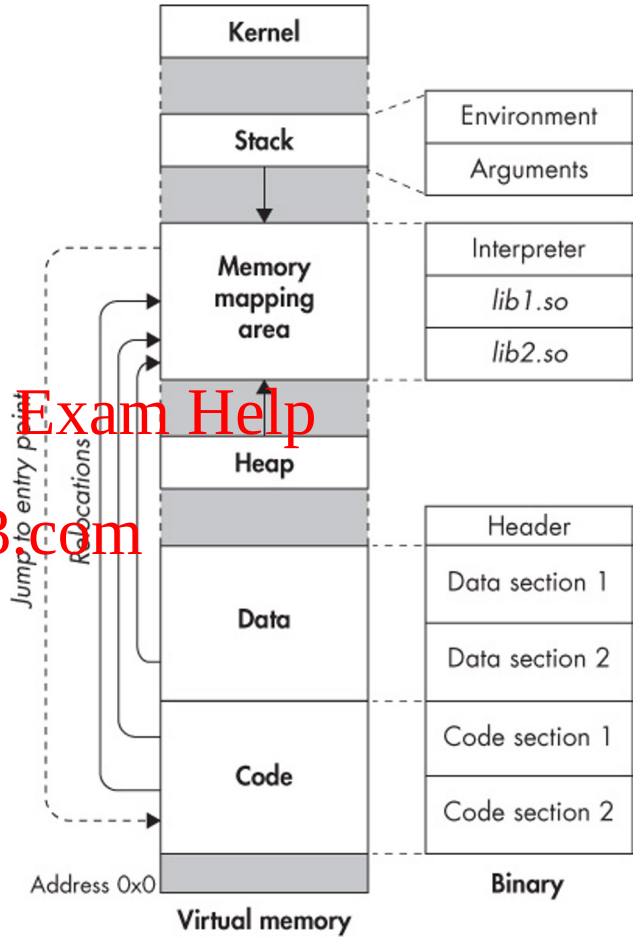
- Binary represen[illegible]nemory may differ from d[illegible] representation.
- OS starts the binary by setting up a new process, including a virtual address space.
- OS maps an interpreter into the process's virtual memory.
- OS transfers control to the interpreter.
- In linux, the interpreter is typically a shared library called *ld-linux.so*.



Loading and Executing Binary

The interpreter:

- Loads binary into its virtual address space.
- Maps required dynamic libraries into virtual address space
- Performs required relocations.
  - Usually with *lazy-binding*.
- Transfers control to the entry point of the program (e.g. the 'main' function).

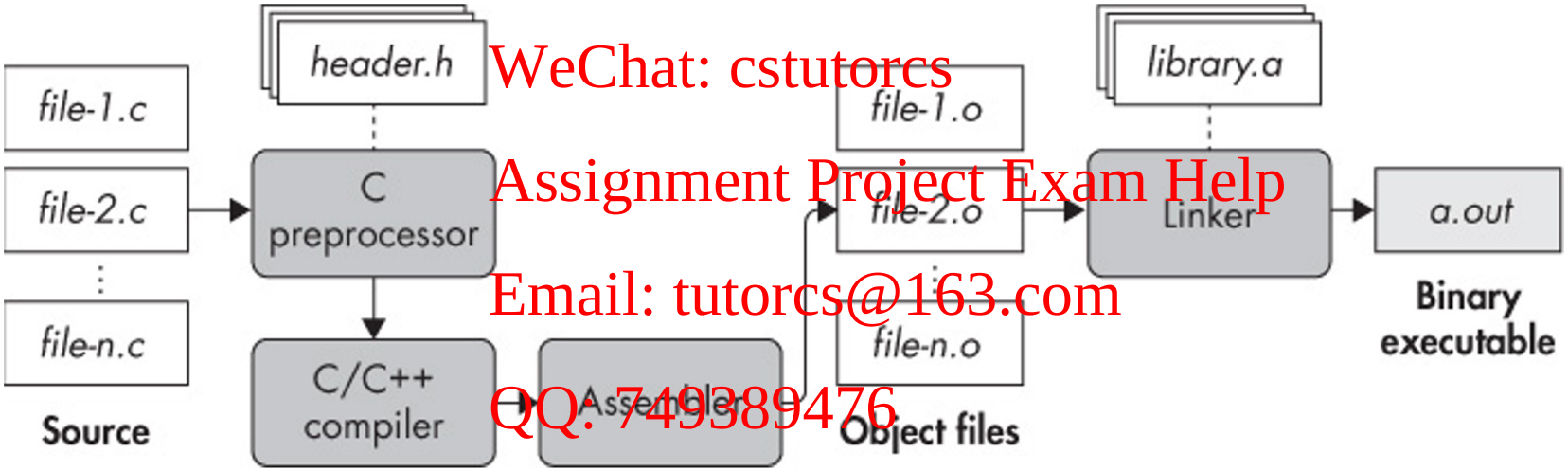


Memory Layout of a Process

Compilation

- Human-readable code → machine code.
- Involving several steps:
  - preprocessing,
  - compilation,
  - assembly, and
  - linking.
- By default, gcc does all the steps when invoked, but specific options can be used to stop at any of the steps.

Compilation (Picture)



Compilation – Preprocessing

- Input: source file
- Output: pre-processed source files.
- Expand macros and directives, such as #include.
- To tell gcc to stop after the pre-processing phase, use option -E.

Compilation – Preprocessing


```
#include <stdio.h>

#define FORMAT_STRING    "%s"
#define MESSAGE          "Hello, world!\n"

int
main(int argc, char *argv[]) {
  printf(FORMAT_STRING, MESSAGE);
  return 0;
}
```


$ gcc -E -P compilation_example.c

```
typedef long unsigned int size_t;
typedef unsigned char __u_char;
typedef unsigned short int __u_short;
/* ... */
extern void flockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));
extern int ftrylockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__)) ;
extern void funlockfile (FILE *__stream) __attribute__
((__nothrow__ , __leaf__));

int
main(int argc, char *argv[]) {
  printf("%s", "Hello, world!\n");
  return 0;
}
```

Compilation - Compiler

- Input: Preprocessed source files
- Output: Assembly files
- Preprocessed code → assembly language.
- Performs heavy optimization, typically configurable as an optimization level, e.g., -O3
- Supply -S flag to stop after compiler phase
- Default is AT&T syntax, use -masm=intel to change to Intel syntax (recommended)

Compilation - Compiler


```
$ gcc -S -masm=intel compilation_e
$ cat compilation_example.s

        .file   "compilation_example.c
        .intel_syntax noprefix
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:

.LFE0:
        .size   main, .-main
        .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0
20160609"
        .section .note.GNU-stack,"",@progbits
```



```
main:
.LFB0:
        .cfi_startproc
        push    rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        mov     rbp, rsp
        .cfi_def_cfa_register 6
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], edi
        mov     QWORD PTR [rbp-16], rsi
        mov     edi, OFFSET FLAT:.LC0
        call    puts
        mov     eax, 0
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
```

Compilation - Assembler

- Input: Assembly
- Output: Object f[illegible]nine code) / modules
- Use -c to generate the object files.
- Indicated as relocatable – does not rely on being placed at any particular address in memory.
- Also known as Position Independent Code (PIC).


```
$ gcc -c compilation_example.c
$ file compilation_example.o
compilation_example.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

Compilation - Linker

- Input: Object
- Output: Single binary executable.
- Sometimes there is additional optimization pass, Link-Time Optimization (LTO).

Compilation - Linker

- Relocation symbols specify how functions and variable references should eventually be resolved.
- Symbolic references: references that rely on a relocation symbol.
- Absolute address for own functions or variables are also symbolic.

Static and Dynamic Libraries

- Static:
  - Merged into binary executable
  - References are resolved entirely
- Dynamic:
  - Shared in memory with all programs in a system and loaded into memory only once.
  - Symbolic references to these libraries are kept in the final executable, resolved when loaded to memory.

Libc

- Shared library that provides core functionalities to GNU systems.
- Header files, e.g., stdio.h, string.h
- Some of the important APIs: open, read, write, malloc, printf, getaddrinfo, dlopen.
- Most of the time, in Linux, C programs will be linked with /lib/libc.so.6.

Symbols

- Keep track of sym es and record which binary code and data correspond t nbol.
- E.g., providing a full mapping between source lines and binary-level instructions.
- Extremely useful if exists, but it can be stripped.


```
Symbol table '.symtab' contains 67 entries:
  Num:    Value          Size Type    Bind   Vis      Ndx Name
    ...
   58: 0000000000400550   101 FUNC    GLOBAL DEFAULT   14 __libc_csu_init
    ...
```

Disassembly – Object files vs. Binary

- Disassembly: Tra[illegible]achine instructions back into assembly.
- Compared to object files disassembly, an executable disassembly:
  - Has a lot more code
  - Contains sections (more to come)
  - Resolves incomplete code and data references

Tools – GDB (GNU Debugger)

- See what is happening inside a program during execution.
- Can be attached to an already running process, or start a new program running with debugging.
- Program needs to be compiled without optimization (-O0) and with debug symbols (-g).
- For debugging, avoid stripping the symbols from the executable binary.

Tools - GEF

- Plugin for GDB int[illegible]r exploit development and reverse-engineering.
- Self-contained and really easy to install.
- Has lots of useful functionalities, e.g., displaying context of code, memory dereferences, virtual memory map, etc.

Tools - Ghidra

- Software Reverse Engineering (SRE) framework by National Security Agency (NSA).
- Lots of software analysis tools and features.
- Scripting languages for customising binary analysis.
- Cross platforms, runs in Windows, macOS, Linux.
- Graphical user interface for most functionalities.

Resources

- Practical Binary Analysis. Dennis Andriesse. Chapter 1
- Assembly:
  - https://azeria-labs.com/arm-data-types-and-registers-part-2/
  - https://software.intel.com/en-us/articles/introduction-to-x64-assembly
  - https://www.microcontrollertips.com/risc-vs-cisc-architectures-one-better/
- GEF
  - https://blahcat.github.io/static/bhusa_2017/BH-USA-17-Alladoum-GDB-Enhanced-Features.pdf
  - https://gef.readthedocs.io/en/master/
- Ghidra:
  - https://ghidra-sre.org
- Libc:
  - https://www.gnu.org/software/libc/