Reverse Engineering Fundamentals

uwin
Azure Assassin Alliance

Azure Assassin Alliance

Why Your Code Can be Run by Computer?

First Thing First

Azure Assassin Alliance

You should know that all programming languages will be eventually transformed to binary instructions and executed by CPU (or GPU).

Compiled vs Interpreted

Programming languages can be categorized into two main types based on how the code is executed: compiled languages and interpreted languages.

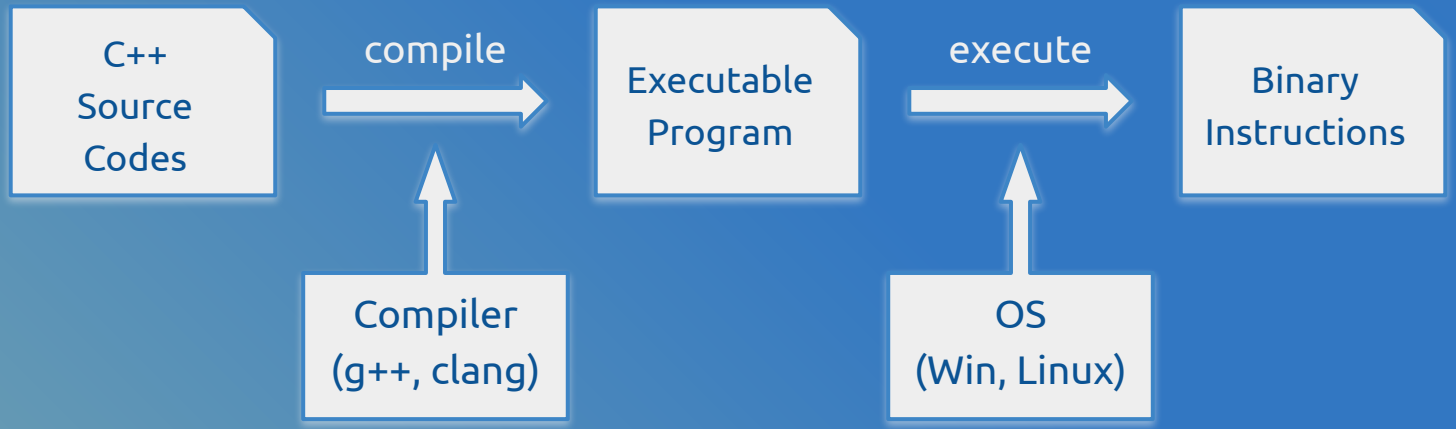
Compiled languages are typically associated with specific components called compiler, whereas interpreted languages are associated with interpreter.

- Popular compiled languages: C, C++, Go, Rust
- Popular interpreted languages: Python, Javascript, Ruby, Lua.

Compiled vs Interpreted

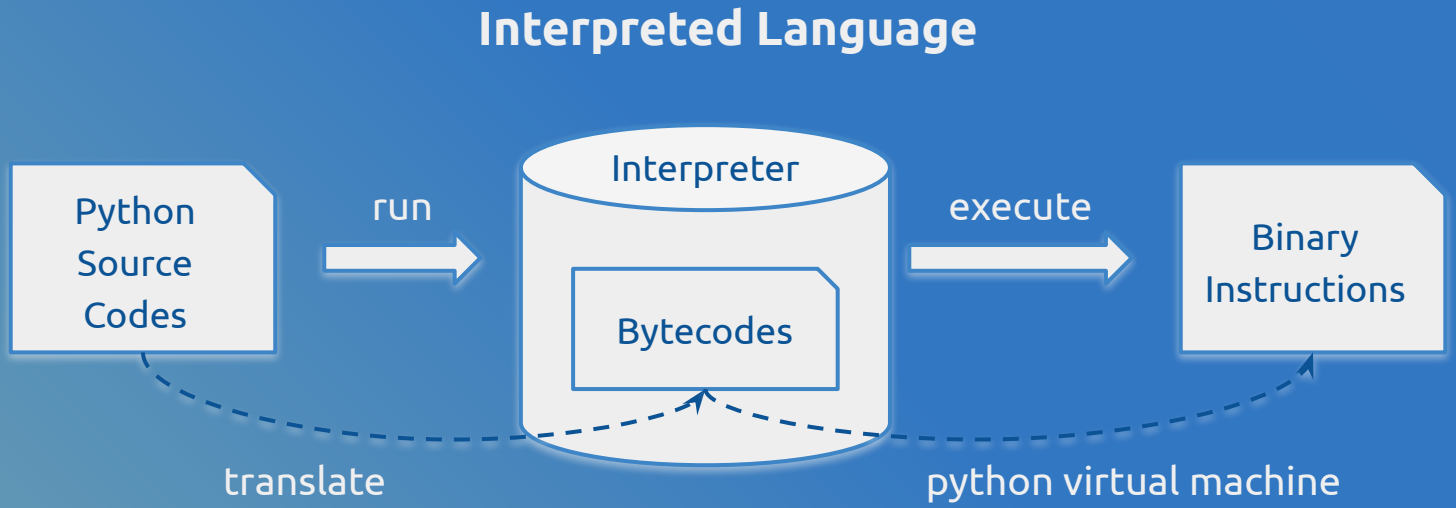
Azure Assassin Alliance

Compiled Language



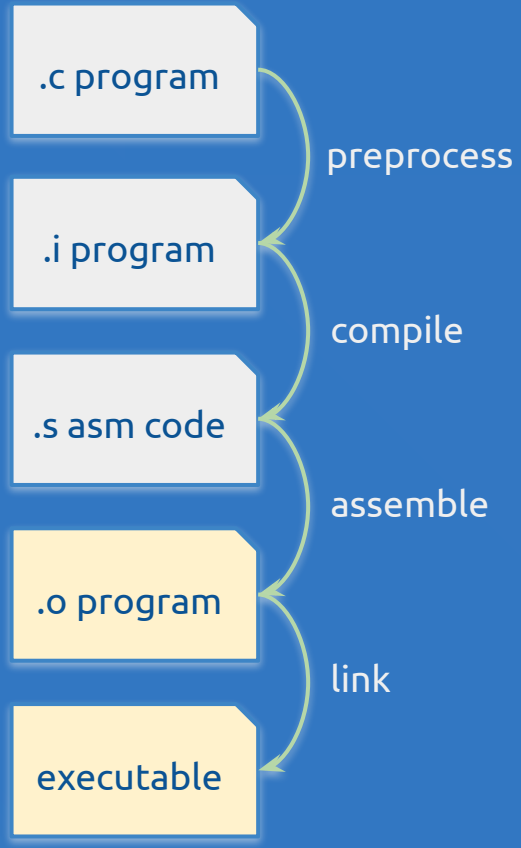
Compiled vs Interpreted

Azure Assassin Alliance



Run “Hello World!”

1. Think (how to design my C code?)
2. Implement (write C code with editor or IDE)
3. Compile (ask compiler to generate program)
    a. preprocess (e.g. gcc -E)
    b. compile (e.g. gcc -S)
    c. assemble (e.g. gcc -c)
    d. link (e.g. ld)
4. Run! (execute your program)



Run “Hello World!”

1. Think (how to design my C code?)
2. Implement (write C code with editor or IDE)
3. Compile (ask compiler to generate program)
    a. preprocess (e.g. gcc -E)
    b. compile (e.g. gcc -S)
    c. assemble (e.g. gcc -c)
    d. link (e.g. ld)
4. Run! (execute your program)

ELF Format

Azure Assassin Alliance

ELF (Executable and Linkable Format) is a common standard file format for “executables” in Linux (or Unix-like) systems.

An ELF file tells:

- what components does the program contain
- how the program should be loaded

ELF Layout

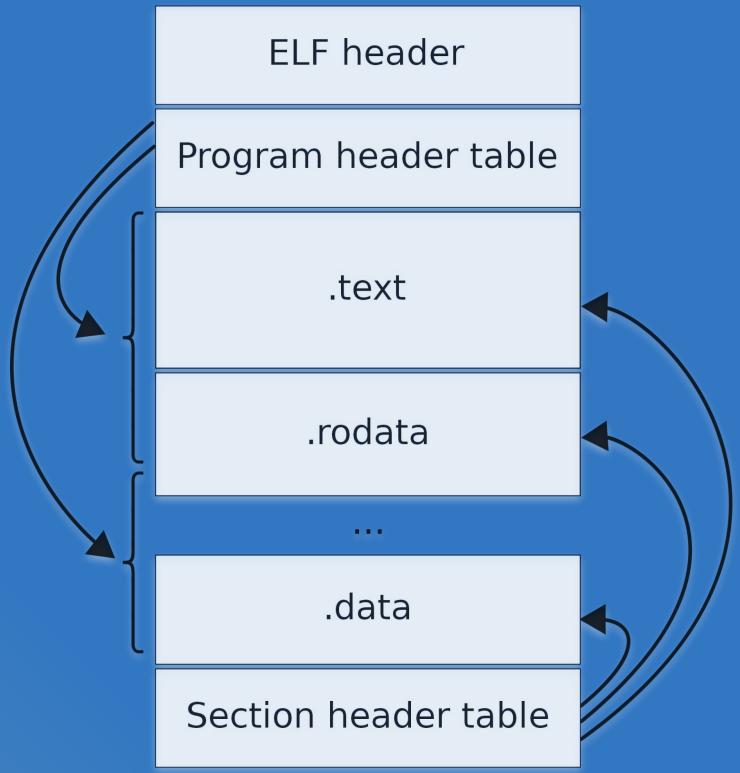
- ELF header:

  basic info, entry, where to find program/section headers and their size
- Sections:

  include codes, symbols, relocations and so on
- Program headers:

  each describing a segment or other info the OS needs to execute the program
- Section headers:

  where to find each section, what is each section for



Program to Process

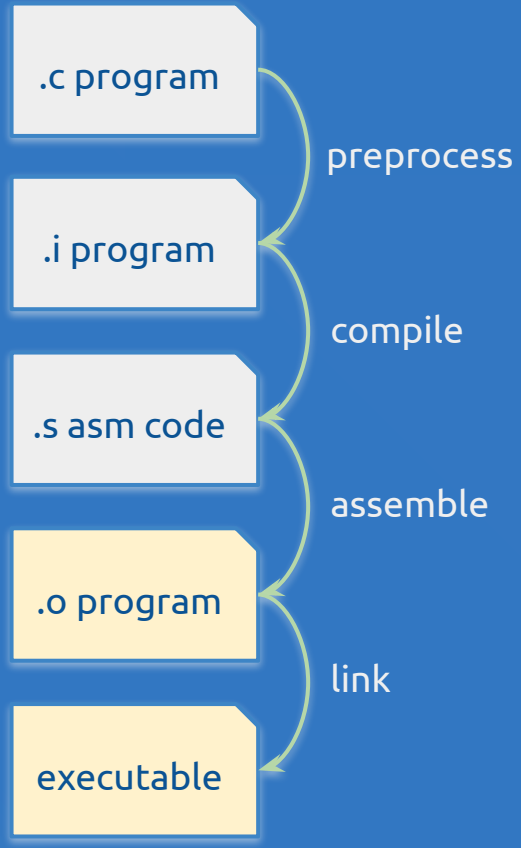
The Linux kernel's support for the ELF format is implemented in fs/binfmt_elf.c.

1. examine ELF header and program headers to check its format and prepare
2. load the program's segments and interpreter into virtual memory, set up the virtual memory
3. start the execution of the program (statically linked) or start the execution of the interpreter (dynamically linked)
4. if dynamically linked, interpreter will recursively load other dependencies
5. program launched and it will become process(es) in OS

So, What is RE?

Basically, RE (Reverse Engineering) is about interacting with given objects (e.g. ELF executables) and try to figure out what they are doing.

“RE is 30% guess work, 70% hard work.”



Interact with Your Target - Statically

Besides just running the binary, there are much more we can do.

GNU Binutils provides a collection of powerful tools that are used to create, modify and manipulate binary files. In addition, there are many other useful command line tools.

- file — determine file type
- strings — print the sequences of printable characters in files
- readelf — display information about ELF files
- nm — list symbols from object files
- strip — discard symbols and other data from object files
- objdump — display information from object files
- patchelf — modify ELF files

Interact with Your Target - Dynamically

- gdb (basic usage)
  - break, brea, bre, br, b -- Set breakpoint at specified location.
  - catch -- Set catchpoints to catch events.
  - delete, del, d -- Delete all or some breakpoints.
  - awatch, rwatch, watch -- Set a (read/access) watchpoint for EXPRESSION.
  - start, run, continue, finish, s, si, n, ni -- control debug flow.
- strace, ltrace

Reverse Like A Pro

Azure Assassin Alliance

A sharp blade quickens the work. There are several state-of-the-art tools.

- Commercial
  - IDA Pro (https://hex-rays.com/ida-pro/)
  - Binary Ninja (https://binary.ninja/purchase/)
- Free
  - IDA Free (https://hex-rays.com/ida-free/)
  - Binary Ninja Cloud (https://cloud.binary.ninja/)
- Open Source
  - Ghidra (https://github.com/NationalSecurityAgency/ghidra)
  - Cutter (https://github.com/rizinorg/cutter)
  - angr (https://github.com/angr/angr)

Reverse Like A Pro

Also, targets that are generated from other programming languages can be analyzed.

- For C# (.NET): ILSpy (https://github.com/icsharpcode/ILSpy), dnSpy (https://github.com/dnSpy/dnSpy)
- For Java (Android): JEB (https://www.pnfsoftware.com/jeb/), jadx (https://github.com/skylot/jadx)
- For VB: VB decompiler (https://www.vb-decompiler.org/)

IDA - How to Use

Azure Assassin Alliance

The basic usage of IDA.

- get familiar with each window
- get familiar with each subview
- take a look at CFG (Control Flow Graph)
- the amazing button -- F5 (decompile)
- deal with ugly names and unclear types
- who use/call this? -- cross references (xrefs)

IDA - How to Use

Azure Assassin Alliance

The advanced usage of IDA.

- you need more types? -- create structures
- obfuscated or anti-debug? -- patch the program
- function undefined? -- manually create function
- to lazy to reverse? -- automatic reversing engineering

Practice!

There is a simple challenge for you.

Try it and answer the following questions. (15 mins?)

1. there is an algorithm which do the encryption, find the related function.
2. once you find the function, can you tell the “key”?
3. the program simply encapsulates the “string” type, can you recover its structure?
4. give me your flag :)

Takeaways

- Be more patient, don't be hasty. Try to figure out how the target is generated (what framework? How is it compiled?) and it will be very helpful for subsequent work.
- In your reversing process, combining static analysis and dynamic analysis will get twice the result with half the effort.
- Try to understand in the shoes of the developer.