Anti-Rev Mitigations

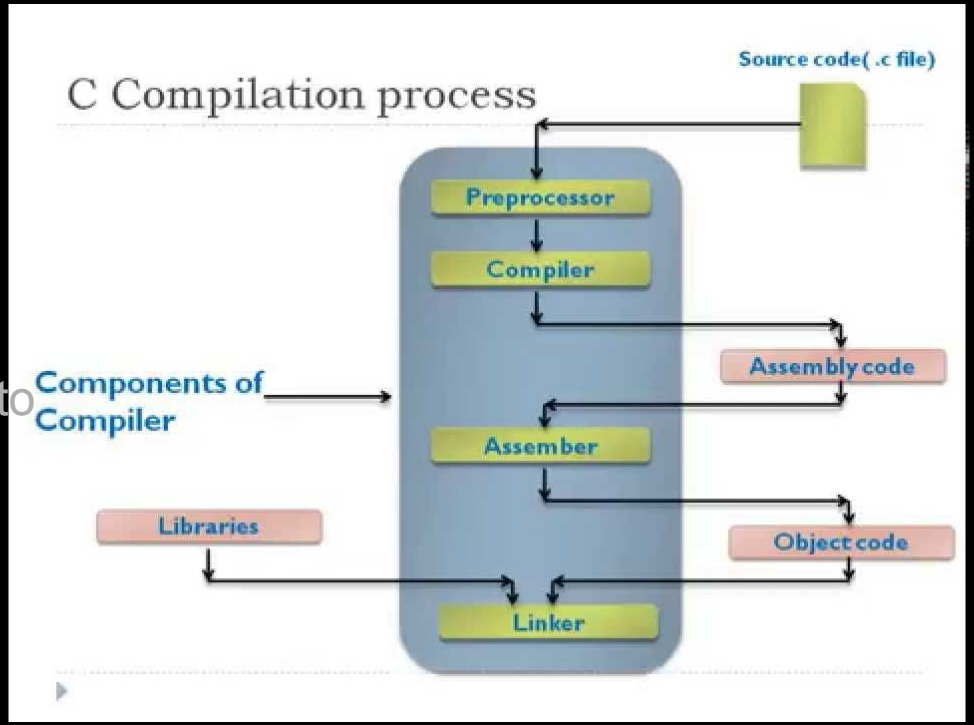
Troy Lopez & DAMSEC, 2026

[Outline]

1. REV Review
2. Rev Mitigations + Protections
3. A Note on crypto
4. Challenge lore
5. Teams + stuff

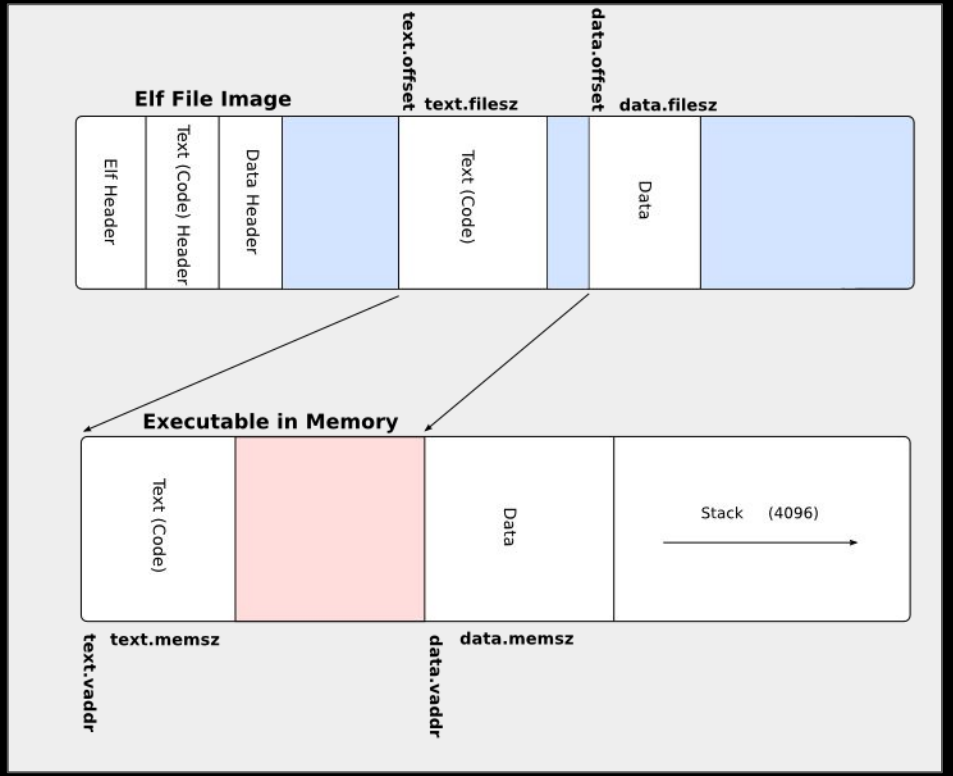
[C -> Executable]

- Human Code is compiled through assembly language to machine code
- Normally, function names are stripped out for efficiency
- When executed, code is loaded into memory. Data from various sections are loaded or referenced at runtime



[C -> Executable]

- Code is separated into different segments. Most importantly are **Text** and **Data**.
- These segments are supposed to split up executable code and referenced data. Additional sections can be made for different types of content stored in a program.
- Each segment has specific attributes, such as read-only, writable, etc.



Reverse Engineering

[Ghidra]

- Reverse engineering tool that decompiles and analyzes code
- Published by the Feds
- Somewhat scuffed, but free and the most popular option
- Installation is a bit weird, we can help troubleshoot during work time
- Alternatives include IDA, Binja, Cutter, etc.
    - Specific tools for specific frameworks/binaries

NationalSecurityAgency/
ghidra

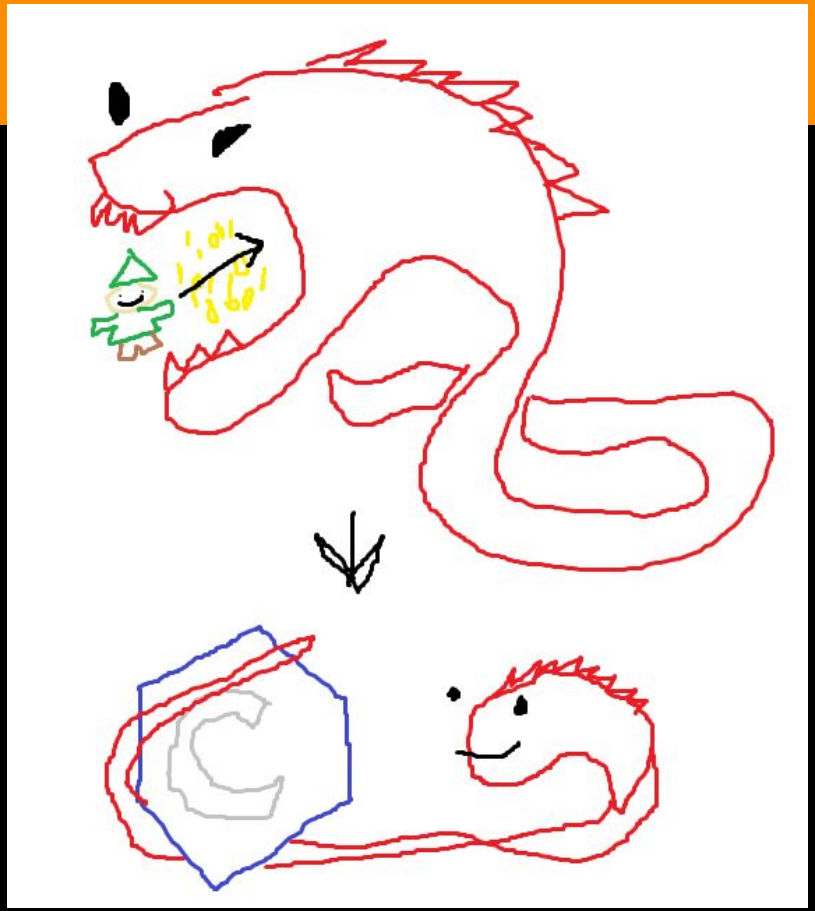
Ghidra is a software reverse engineering (SRE) framework

306 Contributors | 1 Used by | 772 Discussions | 54k Stars | 6k Forks

BINARYNINJA

[Obligatory Dragon Slide]

- Read ELF (binary)
- Disassemble to Assembly instructions
- *Decompile* to pseudo-C code
  - Ghidra does its best to recreate source code but is heuristic
- Analysis
  - Data flow
  - Symbols, functions, etc
- Interactive annotation
  - **This is the part you do!**



Anti-Reverse Engineering Techniques

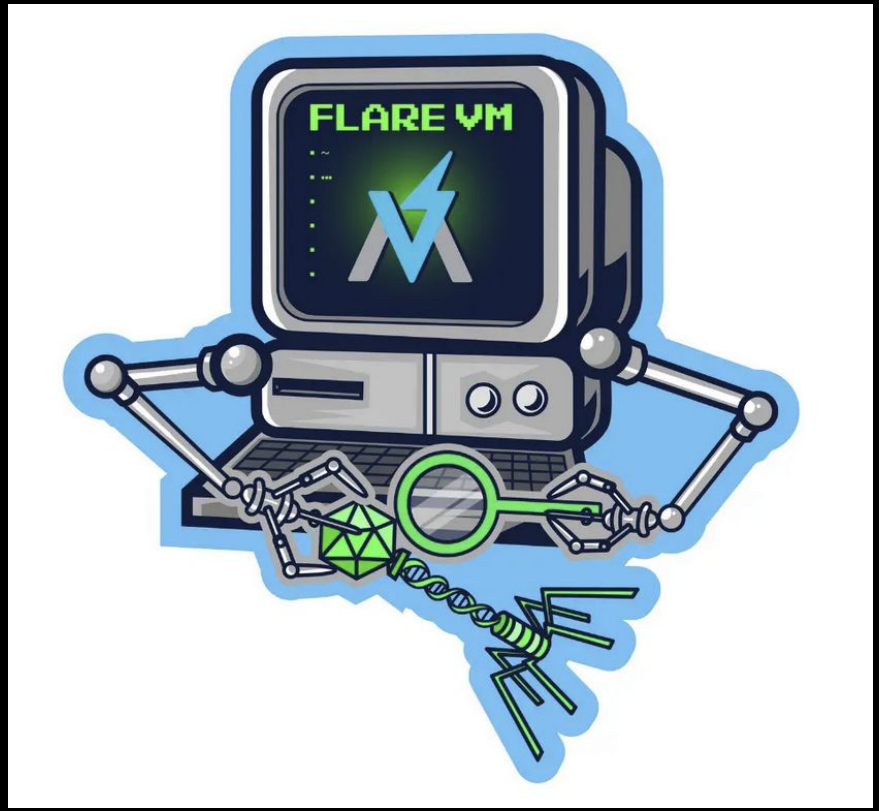
[Goals]

- Evade detection and analysis
  - Make it hard to identify what is happening
- Protect Source code
  - Don’t cheat in fortnite!
- Pretend to be benign software
- Apply client-side integrity checks

Solutions include VM Detection, anti-debugging checks,Obfuscation and Packing, DRM software, checksums, virtualization, reliance on external servers, etc.

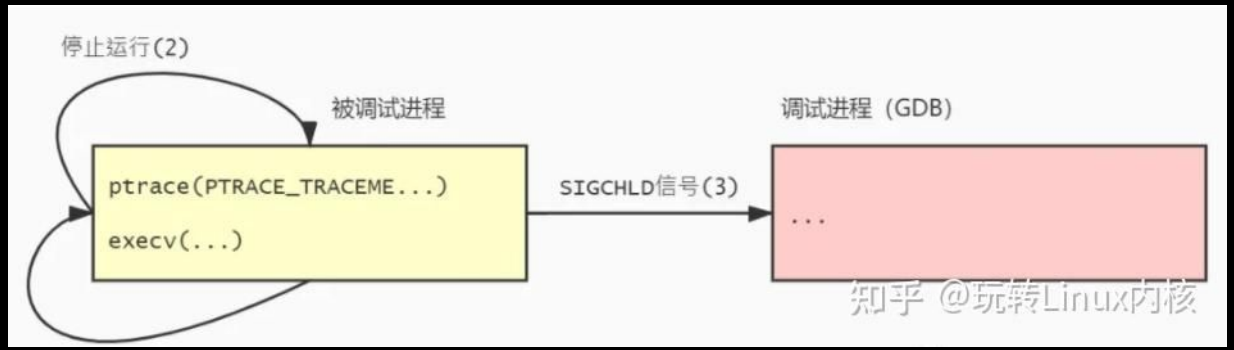
[VM Detection]

Software and malware may refuse to run in a VM, or behave differently in a VM. This would slow down analysis and complicate inspection. Dynamic analysis would be hindered, requiring analysts to either patch out detection code or rely on static analysis.



[Debugger Detection]

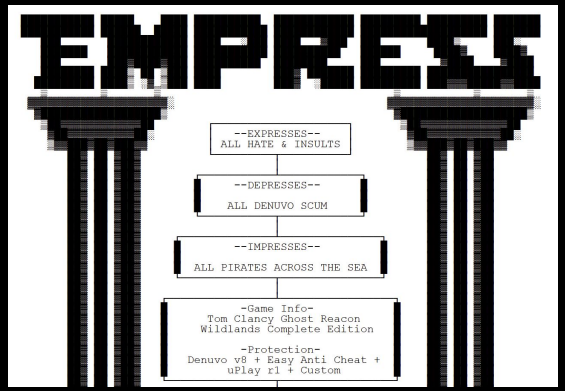
Software can easily detect if it’s being run under a debugger such as GDB by referencing the PTRACE_TRACEME status through a syscall. If an analyst identifies where the comparison check is made in the program through static analysis, they could potentially hex-edit the binary to allow debugging.



[DRM Software]

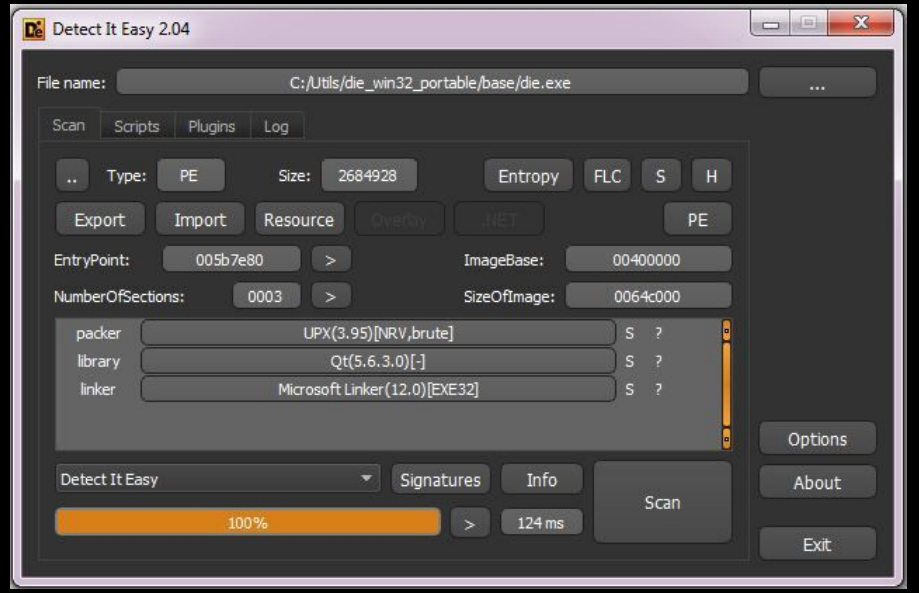
More common in Software than Malware, Digital Rights Management software is used to control access to content. For instance, DRM settings inhibit screenshotting Netflix, or playing EA games without wifi. DRM goes much deeper than the scope of this talk, but common “solutions” require connecting to a third-party server to verify a license (not ownership!) to a game, hashing your hardware details and refusing to execute if values change, and applying digital watermarks to prevent copying (by ethical software).

The DMCA of 1998 makes disabling DRM illegal! Disabling DRM is a major flex for pirates and game crackers.



[Packing and Obfuscation]

- Tools to compress and encrypt executable files into a wrapper/container
- Appears in disassemblers as hard to read, garbage data that decrypts or otherwise utilizes a large section/segment of the program
- Can be identified with tools like **DetectItEasy** or **PEStudio (for PE files)**.
- Can be undone with specialized tools or the packer itself



[Virtualization]

- Languages like Java rely on a virtual machine platform (the JVM) to run code.
- To obfuscate our code, we can create our own VM, and bundle it into our program!
- Custom instructions for behaviors can be hard to detect in RE software and often will need dynamic analysis to trace.
- Obfuscated or minimized interpreted languages can be a good stepping stone for learning VM RE.

Brief note on Cryptography

[Known-plaintext attacks]

- During WWII, the allied forces were able to crack Axis messages by identifying patterns with how weather was broadcast, by noticing how each message started the same way.
- If you know a fragment of the plaintext for a given ciphertext, you can use this to verify if a decrypted message is “correct”
- Modern encryption systems aim to prevent this by ensuring small changes to a PT create a drastically different CT. Evil Corp might not be this smart.

The Challenge

[Challenge]

- 2 files: An archive of messages we found on a compromised machine with a TimmyScan, and the software we believe to be responsible. These appear to be messages intended for an “ECorp operative”???
- At least 1 anti-forensic technique was used to create the binary
- 1 flag is present in a message file, and 1 in the binary itself

1. Reverse engineer the binary
2. Determine what to do with the messages
3. Find flags!