Reverse Engineering and Malware Analysis Fundamentals

Exercises

Adam Podlosky
@apodlosky

Always Use a Virtualized OS

- Do NOT analyze executables on your host operating system
  - Setup and secure a virtual machine (see resources)
  - Also, **REMnux** (“Kali” for malware): https://remnux.org
- The provided binaries were written and compiled by myself
  - They are **NOT** malicious
  - Endpoint/antivirus software may display false-positive alerts

Tools Used In Demo

- PE-bear - PE file format viewer/editor (by @hasherezade)
- IDA - Industry standard disassembler ($$$, freeware version)
    - Alternatives: Cutter/Radare2, BinaryNinja ($), Hopper ($)
- PEiD - PE and packer identification
- ResourceHacker - View and edit PE resources
- SysinternalsSuite - Windows troubleshooting tools
- x32dbg/x64dbg - Great assembly-level debugger for Windows

empty.exe - Part 1, Example 1

- Source File: part1_intro/empty.c
- Objectives:
  - **View source**: Valid C program? Can it compile? Can it link?
  - **PE-bear**: entry point? sections?
  - **IDA**: entry function?
  - **x32dbg**: loaded/mapped modules?

hello.exe - Part 1, Example 2

- Source File: part1_intro/hello.c
- Objectives:
  - **PE-bear**: find entry point
  - **IDA**:
    - So many functions from a single-line program?
    - Find entry point and compare with written code

hello_msgbox.exe - Part 1, Example 3

- Source File: part1_intro/hello_msgbox.c
- Objectives:
  - **View source**: *WinMain()* instead of *main()*?
  - **PE-bear**: compare subsystem value with hello.exe
  - Review subsystems: Console, Windows, (older, POSIX)

hello_winapi_nocrt.exe - Part 1, Example 4

- Source File: part1_intro/hello_winapi_nocrt.c
- Objectives:
    - View source:
        - WinAPI functions instead of standard C functions?
        - *EntryPoint()* instead of *main()* function?
    - **PE-bear**: find entry point
    - **IDA**: a lot fewer functions without C runtime library

greeting.dll - Part 2, Example 1

- Source File: part2_dll/greeting.{c,h,def}
- Objectives:
    - View source: *DllMain()* instead of *main()* function?
    - **PE-bear**: exported functions

nullpad.exe - Part 2, Example 2

- Source File: part2_dll/nullpad.{c,h,rc}
- Objectives:
  - View source: dynamically resolves functions from greeting.dll using *LoadLibrary* and *GetProcAddress*
  - **PE-bear**: imported functions, resources
  - **ResourceHacker**: view resources
  - Vulnerable to DLL hijacking?

annoying.dll - Part 2, Example 3

- Source File: part2_dll/greeting.{c,h}
- Objectives:
  - View source: greeting.c compiled with *-DANNOYING*
  - Review *Dynamic-Link Library Search Order*
  - How could this DLL be used in a hijacking attempt against the Nullpad application

hello_getproc.exe - Part 3, Example 1

- Source File: part3_obfus/hello_getproc.c
- Objectives:
    - **PE-bear**: imports
        - *GetModuleHandle*, *LoadLibrary*, and *GetProcAddress*?
    - **IDA**: identify functions, cross-reference strings

hello_modenum.exe - Part 3, Example 2

- Source File: part3_obfus/hello_modenum.c
- Objectives:
  - **PE-bear**: imports
  - **IDA**: identify functions, cross-reference strings helps
  - **x32dbg**: set breakpoint on *GetProcAddress*
    - Debugging can greatly speed up reversing

hello_stealth.exe - Part 3, Example 3

- Source File: part3_obfus/{hello_stealth.c, nt_internal.h}
- Objectives:
  - **PE-bear**: entry point, imports, section names
  - **PEiD KANAL**: any signatures?
  - **IDA**: several functions, no imports nor strings for clues
  - **x32dbg**: debugging can save time, e.g. return values
  - Bonus: zero out TLS directory, still executes - how?

hello_zeros.exe - Part 3, Example 4

- Source File: part3_obfus/hello_stealth.c
- Objectives:
  - Identify entry point, imports, sections
  - Identify any strings or cryptographic signatures
  - Fully reverse engineer the executable in IDA
    - How are imported functions resolved?
    - Debugging can save reversing time
  - Bonus: zero out TLS directory, still executes - how?

infector.exe - Part 4

- Source File: part4_infect/infector.c
- Infection process:
  - Locate a code cavity in target executable
  - Write target’s OEP into stub code
    - After stub executes, returns to OEP
  - Write stub into code cavity, adjust section headers if needed
  - Set PE’s new entry point to the inserted stub

crackme1.exe - Take Home Project!

- Objective: reverse engineer the program and determine the algorithm required to generate the secret code
- Console-based crackme, run from command prompt
- Hints:
  - Code validation is base on the entered name
  - Locate validation function by debugging or following references to strings

crackme2.exe - Take Home Project!

- Objective: reverse engineer the program and determine the algorithm required to generate the secret code
- Also, it plays chiptunes while you’re busy reversing :-)
- Hints:
  - Identify cryptographic signatures (e.g. *findcrypt*, *PEiD KANAL*)
  - Code validation is based on strings from certain WinAPI calls
  - *User32!GetDlgItemTextA* retrieves text from an edit control