Security for Hackers and Developers: Reverse Engineering

USING IDA PRO TO REVERSE CODE

Dr. Jared DeMott
CTO AND FOUNDER

@jareddemott www.vdalabs.com

Overview

Process

Tools

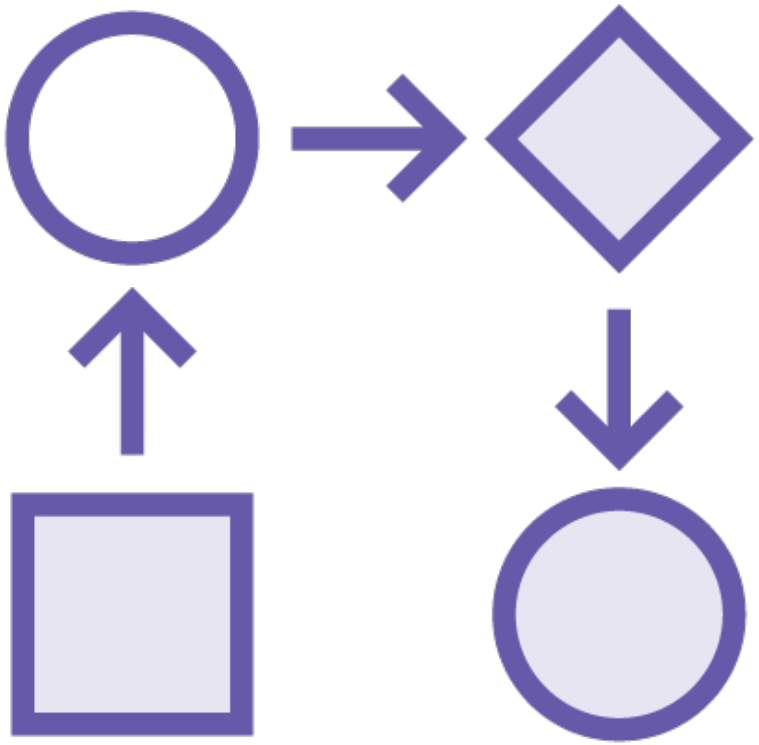
Getting started

Why reverse code?

- Develop against a closed source interfaces
- Understand and defend against threats like malware
- Find vulnerabilities
- Intellectual property theft

Some prerequisite knowledge may be required

- Electronics
  - Devices
- Systems
  - Architecture
- Protocols
  - TCP/IP
  - USB
- Language
  - Compiler



RE Process

- Dive in! Security is hard.
- Target → knowledge→ Tool → skills → objective
  - Practice on code you build and pull apart
- Mix dynamic and static
- Understand key data and the code

Choose the right tool for the right job

- Native code
- Managed code
  - Often less work compared to native
- Network traffic
- Electrical signals

.Net

- https://github.com/0xd4d/dnSpy
- https://www.jetbrains.com/decompiler/

Java

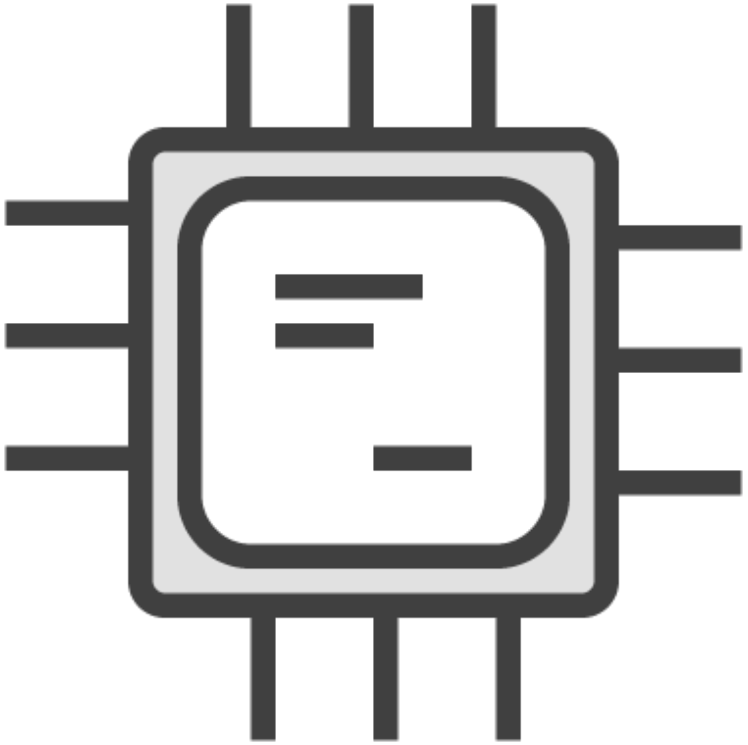
- http://bytecodeviewer.com/

JavaScript

- http://www.relentless-coding.com/projects/jsdetox

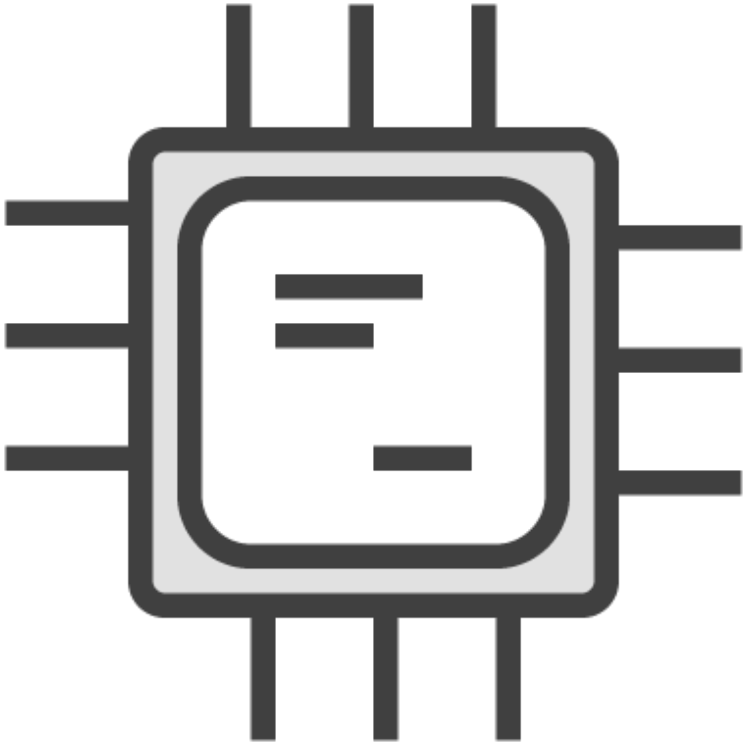
Flash

- https://www.free-decompiler.com/flash/download/



Native code

- Firmware
- OS
- Native C/C++ apps



IDA

- https://www.hex-rays.com/products/ida/index.shtml

BN

- https://binary.ninja/

Differences

- http://www.irongeek.com/i.php?page=videos/grrcon2016/216-binary-ninja-jared-demott

IDA Pro

Interactive

Stronger disassembly heuristics

- Old: objdump, dumpbin, nm, ldd, …
- IDA provides a recursive descent disassembly
  - For each branch/jmp add address to listing to further disassemble

Extensible

Many platforms

Load the file

Different GUI methods

- Just drag the file to the IDA pro icon
- Will detect the file type
- The defaults on the load page are sufficient for most normally formatted files

IDA will create a database (4 files) which it compresses on exit

- The original binary is no longer required unless debugging is desired

Demo

Determine the secret code by examining the binary file

- Download the IDA demo to play along
- “Crack me”
- Try static and dynamic analysis

As you work

No undo

Multiple Windows - hex, strings, stack, functions, and primary disassembly window

- Note the use of the “Intel syntax” for disassembly

Graphing, renaming, commenting, scripting, and plug-in creation

IDA auto names

- Sub_location, var_loation, byte_location, ‘aHelloWorld’

Navigation

Names and addresses can be double-clicked and followed

- The ‘esc’ key is very handy for backing up, almost like a web browser

Note: in the ‘names’ window based on imports, exports, and some analysis

- F is a function
- L is a library function
- C is code/instruction
- A is a string
- D is defined data
- I is an imported function that’s been Dynamically linked

Strings

Options → open subviews → strings

The default is C style strings

- Ascii and greater than 5 bytes long
  - Null terminated
  - Add UNICODE strings if needed

Be sure to change how these are searched if you suspect anything different

- Options → general → strings (tab)
- Or even better: R-click strings window, ‘set up’

Main window

- Stack at top
- Section:addr to left
- Xrefs to right
- Branch lines
- Assembly in the center
  - The hex-rays decompiler is similar to this but tries to create C style code instead
    - works about 70% of the time for x86 32bit C/C++?
- Comments to right
  - ; is a repeatable comment
    - shows up in xrefs
  - : is a non-repeatable comment

Batch Mode

Ida can be run from the command line in a batch mode

- for example:
  - idag.exe –B calc.exe –S bugs.py

Automatically generates a database for a named file

Can specify scripts or plugins to execute on the generated database

Vulnerability Discovery

Fuzzing covered in last class

- Static analysis is a good bug hunting technique as well
- Especially for hackers working on closed source software
- Halvar's BugScam is an old IDC tool
  - Iterates through unsafe C functions, and analyzes the arguments to each call, for possible unsafe use
- Useful for exploit development as well

Stack Analysis

Accurate stack display

- Required for determining proper placement in return address for exploit buffer
- Clear picture of what variables may get clobbered during an overflow

Is there a buffer in this stack frame?

- What is the exact distance from the buffer start to overwrite the saved instruction pointer?
- What variables lie between the buffer and EIP?

Virtual Address Layout

Ida acts like a loader when it analyzes a binary for the first time

- Maps the binary to virtual addresses just as actual loaders do

Easy to determine useful address when "write-anywhere" vulnerabilities are discovered

- This is assuming platforms like old FreeBSD
  - Just modify the static location of printf in the .GOT and the next time printf gets called, Bingo!
- ASLR/DEP on new OS significantly complicate exploitation

Summary

Basics

- RE process
- Tools
- Demo
  - IDA basics and uses

Next

- Learning x86 and Calling Conventions