Introduction to Reverse Engineering for Penetration Testers

Stephen Sims
SANS Fellow
@Steph3nSims
...but I’m a pentester, why should I care about coding, reversing, and exploit dev??

• A quick rant about the past and present...
  • Back when I first started exploit dev there weren’t many jobs in the field
  • The number of people who knew how to write exploits was relatively low
  • Compared to today, exploit-writing was easy!

• The same applies to reversing malware

• Exploit dev is in the GXPN and OSCP...
...but I’m a pentester, why should I care about coding, reversing, and exploit dev??
• A quick rant about the past and present...
  • Back when I first started exploit dev there weren’t many jobs in the field
  • The number of people who knew how to write exploits was relatively low
  • Compared to today, exploit-writing was easy!
• The same applies to reversing malware
• Exploit dev is in the GXPN and OSCP...
• You don’t want to be this

Most overused meme in security, but still funny...
Windows XP shouldn’t be a thing anymore...

Windows XP

This is why we liked XP as attackers...
The attack surface has changed...

Windows XP

Windows 10
Low Level vs. High Level Languages

- There is no specific classification, but languages can be divided up as such:
  - Low level languages include machine code and assembly
  - Medium level languages included C and C++
  - High level languages include C# and Java
  - Even higher level languages include PowerShell and Python
- Lower level languages offer more power as they sit closer to the physical hardware
  - Unmanaged (low level) languages can pose security concerns over that of managed languages (high level)
  - There are various tools available to aid in the reverse engineering of each language, such as disassemblers and decompilers
Disassembly
• The process of taking machine code as input and converting it back to assembly, as originally assembled by the compiler from source code

Example x86 instruction set input:
There are two flavors of disassembly syntax...

• Intel and ATT Intel
  • Neither changes the code, only the way it is displayed
  • Source and destination are switched
  • ATT uses “$” & “%” for immediate and indirect operands, and () for pointers
  • Intel uses [] for pointers and spells out size and such (DWORD PTR)

• Example
  • Intel: 89 04 24 mov DWORD PTR [esp], eax
  • ATT: 89 04 24 movl %eax, (%esp)
Recommended Resources

• The IDA Pro Book
  • The Unofficial Guide to the World’s Most Popular Disassembler by Chris Eagle

• The Hex-Rays Forum - [http://www.hex-rays.com/forum/](http://www.hex-rays.com/forum/)
  • A great resource for research, questions, and answers
  • Must be a registered user (must have an IDA license)

• IDA Plugins

• IDA 7 is finally out
  • It’s now a 64-bit application with a completely redesigned API
What is IDA

• **Recursive Descent Disassembler and Debugger**
  – Supports multiple debuggers and techniques, including WinDbg, GDB, Bochs emulator, etc.
  – Disassembles many processor architectures including ARM, x86, AMD, Motorola, etc.
  – Provides many different graphical and structural views of disassembled code
  – Reads symbol libraries and cross-references function calls
  – Identifies jump tables, lists functions, exported and imported functions, conditional branches, etc.
  – At tool that visually makes you look and feel smarter! 😊
How I first saw IDA many years ago...
Disassembly Types

• **Linear Sweep Disassembly - gdb, WinDbg, objdump**
  • Easiest and most straightforward approach
  • Begin at Code Segment (CS) entry point & disassemble one instruction at a time linearly until the end of the CS
  • Does not accommodate control flow such as branches

• **Recursive Descent Disassembly - IDA**
  • Much more complex and effective approach
  • Can tell instructions from data
  • Handles branches such as jumps and calls
  • Defers branch target instructions based on a condition
Conditional Jump Example

• Jump on Zero (JZ) and similar instructions
• Checks Zero Flag
Primary Dashboard (1)

Overview Navigator

Function names

Graphical view of disassembled function
Primary Dashboard (2)
Import and Export Address Tables

• By clicking on the “Imports” or “Exports” pane you will get a listing of the IAT/EAT or PLT/GOT for the file examined

• There are other panes and views that will be discussed when appropriate
Debugging Symbols Resolved

Failed to load symbols
IDA Alternatives

- It is often asked as to what alternatives there are to IDA
  - Radare2: http://www.radare.org
    - A free reverse engineering framework
    - Installed on Kali Linux by default
    - Disassembler, debugger, diffing, extensible, etc.
  - Hopper: http://www.hopperapp.com/
    - Reverse engineering tool for Linux and OS X
    - $89 Personal License & $169 Computer License
    - Disassembler, decompiler, extensible, debugger, etc.
Remote Debugging with IDA

• IDA supports remote debugging, which allows you to use IDA’s graphical front-end to various debuggers remotely
  • Mac OS X 32-bit & 64-bit
  • Windows 32-bit & 64-bit
  • Linux 32-bit & 64-bit
  • Windows CE
  • ARM application debugging
  • Android application debugging
  • Remote GDB

64-bit application debugging only supported with IDA Professional, formerly IDA Advanced.
IDA SDK and Automation Overview

• Overview of features:
  • The IDA SDK allows you to write your own plugins, primarily in C & C++
  • Allows developers and users of IDA to expand IDA’s capabilities, automate analysis, etc.
  • IDA offers scripting support to interact with the IDA API and practically all contents of the IDA database
  • The IDA API allows for interaction via a C/C++ like language called the IDC scripting language
  • Since IDA 5.4, Python scripting is supported through the use of IDAPython
  • IDA 7 has a new API, so old plugins don’t work without porting over
IDAPython

• Plugin to IDA allowing Python scripting

• IDA Python is led by Gergely Erdelyi and available at http://code.google.com/p/idapython/

• More powerful than IDC with access to SDK

• We will focus more heavily on IDAPython due to ease of use, power, and community support

• Using the “ctypes” module in Python can help get around some SDK limitations

• Replaces the interactive box at the bottom of IDA
Common Exploit Technique: Pivoting the Stack Pointer?

• The stack pointer advances with pop’s and ret’s
• If we exchange it with another reg, we can yield its power
FLIRT and FLAIR

• Fast Library Identification and Recognition Technology (FLIRT)
  • Technology to look for patterns in common library functions
  • Helps reduce time spent reversing statically compiled library functions
  • https://hex-rays.com/products/ida/tech/flirt/in_depth.shtml

• Fast Library Acquisition for Identification and Recognition (FLAIR)
  • A tool set that allows you to write your own FLIRT signatures

• Used commonly by CTF teams and malware analysts as library code is often statically compiled into the programs

Quick Demo...
Summary

• Reversing may be intimidating at first, but you pick it up quickly and there are a lot of free resources online
• Practice makes perfect
• There’s just not enough time to get everything done without automating as much of your work as possible
Thanks!

Questions?

Stephen Sims
@Steph3nSims