Attacking Obfuscated Code with IDA Pro

Chris Eagle
cseagle@nps.edu
Outline

• Introduction
• Operation
• Demos
• Summary
First Order Of Business

- MOVE UP AND IN!
  - There is plenty of room up front
  - I can't increase the font size in IdaPro
Background

• IDA Pro
  – Interactive Disassembler Professional

• Premier disassembly tool for reverse engineers
  – Handles many families of assembly language

• Runs on Windows
  – Linux console version newly available.
What?

- **ida-x86emu** is a plugin for IDA Pro that allows for emulated execution of x86 instruction set
- **Written in C++**
  - Currently packaged as VC++ 6.0 project
- **Available here:**
  - [http://sourceforge.net/projects/ida-x86emu](http://sourceforge.net/projects/ida-x86emu)
Why?

- Hand tracing assembly language is difficult to do in large or complex programs
- Anti-reverse engineering techniques attempt to obfuscate code paths
- Allows automated unpacking/decrypting of "protected" binaries
  - UPX, burneye, shiva, tElock, A$P$ack, …
Primary Motivation

- Getting at protected executables
  - Most viruses/worms are protected in some way
  - Often UPX, tElock, ASPack
- Challenge for static reverse engineering is getting past the protection
  - ida-x86emu allows you to "run" through the decryption routine within IDA Pro
Outline

• Introduction
• Operation
• Demos
• Summary
IDA Pro

- Load the binary of interest
- IDA builds a database to characterize each byte of the binary
- Performs detailed analysis of code
  - Recognizes function boundaries and library calls
  - Recognizes data types for known library calls
Obfuscated Code

- Challenging for IDA
- Usually only get sensible output for entry point function
- Protected program appears as data rather than code because it is obfuscated/encrypted
- Jumps into middle of instructions confuse flow analysis
The Plugin

- Two pieces
  - User interface
    - Windows-specific gui code
    - Handles dialog boxes
  - x86 emulator
    - Platform independent (mostly!)
    - Executes a single instruction at a time
      - Reads from IDA database or user-supplied memory block
Console

[Image of x86 Emulator interface showing registers and stack information]
Using It

- Alt-F8 launches control console
- eip initialized to cursor location
  - (1st time only)
- Step and go
  - The plugin tells IDA to reorganize its code display based on ACTUAL code paths
  - Defeats jump into the middle of an instruction type obfuscation
Features

• Run to Cursor
  – No breakpoints yet

• Plugin supplies its own stack
  – Stack push places arguments on the stack
  – Useful if you want to setup a function call

• Plugin supplies its own heap
  – Redirect library functions to plugin provided equivalents
Limitations

• Slow
  – Because of emulated execution and IDA interactions

• Can't follow calls into dynamically linked functions

• Can't follow system calls in statically linked functions
Emulator Memory

- Code and static data must be fetched from IDA database
- Other references must be directed to either stack or heap
  - Every memory reference checked
  - Could easily add Valgrind type memory analysis
Memory Layout

- Emulation options allow you to specify memory layout
Emulated Stack

- Used by all stack operations in the program
  - Stack contents displayed in main emulation window
  - Auto scrolls to most recent reference
- Allows pushing data onto stack outside of program control
  - Useful to setup and run individual functions
Emulated Stack

Pushed right to left per C calling conventions
Emulated Heap

- Simple linked list memory allocator
- Does not emulate any specific allocation algorithm
  - Specifically, no in-band control info
- Won't mimic heap overflow problems
- Can detect access outside allocated blocks
Function Hooking

• Two methods
  – Manual invocation of emulator equivalent function
    • Result in eax, actual call statement in code must be "skipped"
  – Automatic, hooked invocation of emulator equivalent function
    • call statement redirected to emulated library function
Manual Function Hooking

• Required parameters, if any, taken from stack
• Result into eax
• No change to eip
Automatic Function Hooking

- Step through hooked call statement causes emulator equivalent to be executed instead.
Automatic Function Hooking

• Functions available for hooking
  – Windows
    • VirtualAlloc, VirtualFree
    • GetProcAddress
    • GetModuleHandle, LoadLibrary
  – Standard library
    • malloc, calloc, realloc, free
Windows Structured Exception Handling (SEH)

- Work in progress
- tElock for example uses SEH as an anti-RE technique
- Point FS register at dummy Thread Environment Block (TEB)
- Only a few recognized exceptions
  - Divide by zero, INT3, single step, Debug registers
SEH (continued)

- Emulated program must setup an exception handler
  - Emulator does not offer a default exception handler
- Emulator creates SEH data structures, pushes them on the stack and jumps to user defined exception handler
Outline

• Introduction
• Operation
• **Demos**
• Summary
UPX Demo

• One of the most common obfuscators
• Reversible using UPX itself
• UPX corruptors exist that break UPX's reversing capability
• Simple unpacking loop, no tricks
• No problem for the plugin
• Doesn't rebuild import table yet
ASPack Demo

- ASPack requires
  - LoadLibrary, GetProcAddress
    - Used to retrieve VirtualAlloc and VirtualFree
- Currently emulator mimics VirtualAlloc and VirtualFree
- Hook LoadLibrary and GetProcAddress calls
ASPack Demo

- Hooked LoadLibrary reports to message window
- Hooked GetProcAddress returns unique id for each function lookup
- Automatic hooking by GetProcAddress will create hooks for VirtualAlloc and VirtualFree
  - Could use returned id to hook VirtualAlloc and VirtualFree calls
tElock Demo

• Sets up Windows exception handlers, then generates exceptions to jump into handlers
• Grab some memory for TEB and point FS register at it
  – Execute a malloc or manually push a bunch of data
• SEH only enabled when a Windows PE is loaded
Burneye Demo

- Early ELF protector by Team TESO
- Embeds the entire protected ELF binary within a protective unwrapper
  - Offers layers of obfuscation/encryption
- Once decrypted, the protected binary can be dumped out of the IDA database
  - Plugin provides a "dump block to file" capability
Shiva Demo

• Shiva is a binary protector
  – Similar goals to Burneye
• Multilevel encryption protects binary
• Polymorphic stage 1 decryptor
• Embedded key generation functions for last stage decryption
Shiva Key Recovery

- Shiva contains 5 different types of encrypted blocks
- Each block gets its own key
  - Blocks of same type share the same key
- In this case we need to generate 5 keys in order to decrypt all of the types of blocks
Key Obfuscation

- Shiva contains a key generation function for each type of crypt block
- Block decryption sequence
  - Identify block type (0-IV)
  - Call appropriate key generation function
  - Decrypt block
  - Clear the key
Key Generation

• Functions are obfuscated
  – Similar to layer 1 decrypt
  – Differ from one binary to the next
  – Resistant to script-based recovery

• But
  – They are easy to locate
  – A table points to the start of each function
Key Recovery

• The plugin can be used to run the functions and collect the keys!
• Setup desired parameters on the stack
  – Pointer parameters need to point to valid memory blocks
    • Grab memory on stack
    • Manually invoke malloc
• Point eip at the function and step
Using the Keys

• With 5 keys in hand it is possible to decrypt all of the crypt blocks
• The plugin can be used to invoke Shiva's decryption function
  – Setup the stack
    • Pointer to the block
    • Pointer to the key
  – Step through the decryption function
Outline

- Introduction
- Operation
- Demos
- Summary
To Do

- Breakpoints
- More library calls
- Better memory displays
- Memory use reporting
- Improved exception handling
Summary

- Acts as something of a "universal" decryption script for protected binaries
- Dramatically reduces time to reverse protected binaries
- Emulator code can be used independently of gui code to create automated unwrappers
  - Combine with ELF or PE parser
- Suggestions welcome
Questions?

- Thanks for coming
- Contact info:
  - Chris Eagle
  - cseagle@nps.edu
References

• Armouring the ELF: Binary encryption on the UNIX platform, grugq & scut,
  http://www.phrack.org/phrack/58/p58-0x05
• Shiva: Advances in ELF Runtime Binary Encryption, Clowes & Mehta, Black Hat USA 2003,
• Strike/Counter Strike: Reverse Engineering Shiva, Eagle, Black Hat Federal 2003,
References

• Shiva-0.96, Clowes & Mehta,
  http://www.blackhat.com/presentations/bh-usa-03/bh-usa-03-mehta/bh-us-03-shiva-0.96.tar

• Burneye-1.0.1, scut, http://teso.scene.at/releases/burneye-1.0.1-src.tar.bz2

• IDA Pro, Data Rescue,
  http://www.datarescue.com/idabase/

• The Ultimate Packer for eXecutables
  http://upx.sourceforge.net/