Reverse Engineering by Crayon: Game Changing Hypervisor and Visualization Analysis

Fine-grained covert debugging using hypervisors and analysis via visualization

Daniel A. Quist
Lorie M. Liebrock
Offensive Computing, LLC
New Mexico Tech

Defcon 17
Las Vegas, NV
Introduction

- Reverse Engineering is Hard!
- Hypervisor based executable monitoring
- Modifications for improved performance
- Visualization tool for rapid analysis
- Modifying the reverse engineering process
Difficulties of RE

- Time consuming process
- Difficult set of skills to acquire
- Tools are advanced, but still don’t provide adequate views.
- Focused on static analysis
- Software armoring makes process even more difficult
Process for Reverse Engineering

- Setup an isolated run-time environment
- Execution and initial analysis
- Deobfuscate compressed or packed code
- Disassembly / Code-level Analysis
- Identify and analyze relevant and interesting portions of the program
Isolated Analysis Environment

- Setup an Isolated Runtime Environment
  - Virtual machines: VMWare, Xen, KVM, …
  - Need to protect yourself from malicious code
  - Create a known-good baseline environment
  - Quickly allows backtracking if something bad happens
Execution and Initial Analysis

- **Goal**: Quickly figure out what the program is doing without looking at assembly

- Look for:
  - Changes to the file system
  - Changes to the behavior of the system
    - Network traffic
    - Overall performance
    - Ads or changed browser settings
Remove Software Armoring

- Program protections to prevent reverse engineering
- Done via packers – Small encoder/decoder
- Self-modifying code
- Lots of research about this
  - OllyBonE, Saffron, Polyunpack, Renovo, Ether, Azure
  - My research uses Ether
Packing and Encryption

- Self-modifying code
  - Small decoder stub
  - Decompress the main executable
  - Restore imports

- Play “tricks” with the executable
  - OS Loader is inherently lazy (efficient)
  - Hide the imports
  - Obscure relocations
  - Use bogus values for various unimportant fields
Software Armoring

- Compressed, obfuscated, hidden code
- Virtual machine detection
- Debugger detection
- Shifting decode frames
Normal PE File

```
push    ebp
mov     ebp, esp
sub     esp, 1Ch    ; lpMsg
call    ds:_imp__GetCommandLineW@0
push    [ebp+nCmdShow]  ; nCmdShow
push    eax           ; int
push    [ebp+hPrevInstance] ; int
push    [ebp+hInstance] ; hInstance
_call   _FSolInit@16    ; FSolInit(x,
test    eax, eax
jz      short locret_1001F13
push    esi
mov     esi, ds:_imp__GetMessageW@16
push    edi
mov     [ebp+Msg.wParam], 1
xor     edi, edi
jmp     short loc_1001EFE
```
Troublesome Protections

- Virtual Machine Detection
  - Redpill, ocvmdetect, Paul Ferrie’s paper

- Debugger Detection
  - IsDebuggerPresent()
  - EFLAGS bitmask

- Timing Attacks
  - Analyze value of RDTSC before and after
  - Really effective
<table>
<thead>
<tr>
<th>Thwarting Protections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two methods for circumvention</td>
</tr>
<tr>
<td>1. Know about all the protections before hand and disable them</td>
</tr>
<tr>
<td>2. Make yourself invisible</td>
</tr>
</tbody>
</table>
Virtual Machine Monitoring

- Soft VM Based systems
  - Renovo
  - Polyunpack
  - Zynamics Bochs unpacker

- Problems
  - Detection of virtual machines is easy
  - Intel CPU never traditionally designed for virtualization
  - Do not emulate x86 bug-for-bug
OS Integrated Monitoring

- Saffron, OllyBonE
  - Page-fault handler based debugger
  - Abuses the supervisor bit on memory pages
  - High-level executions per page

- Problems
  - Destabilizes the system
  - Need dedicated hardware
  - Fine-grain monitoring not possible
Fully Hardware Virtualizations

- Ether: A. Dinaburg, P. Royal
  - Xen based hypervisor system
  - Base functions for monitoring
    - System calls
    - Instruction traces
    - Memory Writes
  - All interactions done by memory page mapping

- Problems
  - Unpacking code primitive
  - Dumps mangled and not possible to disassemble
  - Old version of Xen hypervisor
Disassembly and Code Analysis

- Most nebulous portion of the process
- Largely depends on intuition
  - Example: When we reversed the MP3 Cutter and MIRC programs
  - Takes time and experience
- Looking at assembly is tedious
- Suffers from “not seeing the forest from the trees” syndrome
- Analyst fatigue – Level of attention required yields few results
Find Interesting and Relevant Portions of the Executable

- Like disassembly, this relies on a lot of intuition and experience

- Typical starting points:
  - Look for interesting strings
  - Look for API calls
  - Examine the interaction with the OS

- This portion is fundamentally imprecise, tedious, and often frustrating for beginners and experts
Contributions

- Modifications to Ether
  - Improve malware unpacking
  - Enable advanced tracing mechanisms
  - Automate much of the tedious portions

- Visualizing Execution for Reversing and Analysis (VERA)
  - Speed up disassembly and finding interesting portions of an executable
  - Faster identification of the Original Entry Point
Extensions to Ether

- Removed unpacking code from hypervisor into userspace
- Better user mode analysis
- PE Repair system – Allows for disassembly of executables
- Added enhanced monitoring system for executables
Results

- Close to a truly covert analysis system
  - Ether is nearly invisible
  - Still subject to bluepill detections
- Fine-grain resolution of program execution
- Application memory monitoring and full analysis capabilities
- Dumps from Ether can now be loaded in IDA Pro without modification
Open Problems

- Unpacking process produces lots of candidate dump files
- Need to figure out what the OEP is
- Import rebuilding is still an issue
- Now that there is a nice tool for tracing programs covertly, we need to do analysis
## Visualization of Trace Data

### Goals:
- Quickly visually subvert software armoring
- Identify modules of the program
  - Initialization
  - Main loops
  - End of unpacking code
- Figure out where the self-modifying code ends (OEP detection)
- Discover dynamic runtime program behavior
- Integrate with existing tools
Visualizing the OEP Problem

- Each block (vertex) represents a basic block executed in the user mode code
- Each line represents a transition
- The thicker the line, the more it was executed
- Colors represent areas of memory execution
## VERA

- Visualization of Executables for Reversing and Analysis
- Windows MFC Application
- Integrates with IDA Pro
- Fast, small memory footprint
VERA Architecture

Ether Analysis System

Linux Dom0 Management OS

Instrumented Windows XP VM

VM Disk Images Ether Mgmt Tools

Xen Hypervisor with Ether Extensions Ring -1

Intel x86 CPU w/ Hardware Virtualization
Visualizing Packers

- Memory regions marked for PE heuristics
Demo!
Netbull Virus (Not Packed)
Netbull Zoomed View
Visualizing Packers

- Memory regions marked for PE heuristics

Color Key:

- Normal
- No section present
- Section SizeOfRawData = 0
- High Entropy (Packed or Compressed)
- Instruction not present in packed executable
- Operands don't match
UPX

Color Key:
- Normal
- No section present
- Section SizeOfRawData = 0
- High Entropy (Packed or Compressed)
- Instruction not present in packed executable
- Operands don't match
UPX - OEP
ASPack

Color Key:
- Normal
- No section present
- Section SizeOfRawData = 0
- High entropy (Packed or Compressed)
- Instruction not present in packed executable
- Operands don’t match
TeLock

Color Key:
- Normal
- No section present
- Section Size Of Raw Data = 0
- High Entropy (Packed or Compressed)
- Instruction not present in packed executable
- Operands don’t match
Future Work

- General GUI / bug fixes
- Integration with IDA Pro
- Memory access visualization
- System call integration
- Function boundaries
- Interactivity with unpacking process
- Modify hypervisor to work with WinDBG, OllyDbg, IDA Debugger
Conclusions

- Visualizations make it easy to identify the OEP
- No statistical analysis of data needed
- Program phases readily identified
- Graphs are relatively simple
- Preliminary user study shows tool holds promise for speeding up reverse engineering
Questions?

These slides are out of date! Find the latest ones at:

http://www.offensivecomputing.net/