Visualizing Compiled Executables for Malware Analysis

Daniel Quist
Lorie Liebrock
New Mexico Tech
Los Alamos National Laboratory
Overview

Explanation of Problem
Overview of Reverse Engineering Process
Related Work
Visualization for Reverse Engineering
VERA Architecture
Case Study: Mebroot
User Study
Contributions
Explanation of Problem

• Reverse engineering is a difficult and esoteric skill to learn

• Most new reversers struggle with understanding overall structure

• Knowing where to start is the most difficult task
Reverse Engineering Process

1. Setup an Isolated Environment
   - VMWare, Xen, Virtual PC
   - Dedicated Hardware

2. Initial Analysis and Execution
   - Sysinternals, CWSandbox
   - Look for OS State Changes
     - Files, registry, network

3. Deobfuscation / Software Dearmoring
   - Unpacking
   - Debuggers, Saffron, Ether

4. Disassembly / Code-level analysis
   - IDA Pro
   - OllyDbg

5. Identify Relevant and Interesting Features
   - Experience based
   - Newbies have trouble with this
Addressing the Situation

- Setup an Isolated Environment
- Initial Analysis and Execution
- Deobfuscation / Software Dearnoring
- Disassembly / Code-level analysis
- Identify Relevant and Interesting Features
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
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
```
Packed PE File

```
public start
start proc near
pusha
mov   esi, offset loc_406000
lea   edi, [esi-5000h]
push  edi
or    ebp, 0FFFFFFFh
jmp   short loc_406882
```
Related Work
• Illustrates Relationship of Function Calls
• Magenta represents imported API calls
• Black represents module subroutines
IDA Pro – Visualization Problems

Firefox Initialization

- Some graphs are useless
- Some graphs are too complex
- No indication of heavily executed portions
- Obfuscated code is gibberish
Alex Dragulescu – MyDoom Visualization

http://www.sq.ro/malwarez.php
Visualization for Reverse Engineering

• Identify major program functional areas
  – Initialization
  – Main loops
  – Communications / organizational structure

• Deobfuscation / dearmoring
  – Identify packing loops
  – Find self-modifying code

• Take “intuition” out of the reversing process
Enabling Technology: Ether

- Patches to the Xen Hypervisor
- Instruments a Windows system
- Base modules available
  - Instruction tracing
  - API tracing
  - Unpacking
- “Ether: Malware Analysis via Hardware Virtualization Extensions”
  Dinaburg, Royal, Sharif, Lee

ACM CCS 2008
Visualizing Executables for Reversing and Analysis

- OpenGL rendering of dynamic program execution
- Vertices represent addresses
- Edges represent execution from one address to another
- Thicker edges represent multiple executions
- Colors to help identify type of code
Graph Preview
VERA Architecture

Open Graph Display Framework
- Handles all layout and arrangement of the graphs
- Similar to Graphviz
- Works with large datasets
Vertices (Addresses)

- Basic blocks
  - Fundamental small grouping of code
  - Reduces data size
  - Useful for large commercial programs

- Instructions
  - Useful for small programs
  - Greater aesthetic value
  - Larger datasets can produce useless graphs
Edges (Transition)

• Transitions between addresses

• Thicker lines represent more executions
  – Easy identification of loops
  – Find heavy concentration of execution

• Multiple edges from a node represent decision point
Colors

- **Yellow** – Normal uncompressed low-entropy section data
- **Dark Green** – Section not present in the packed version
- **Light Purple** – SizeOfRawData = 0
- **Dark Red** – High Entropy
- **Light Red** – Instructions not in the packed exe
- **Lime Green** – Operands don’t match
Netbull Virus (Not Packed)
Netbull Zoomed View
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
### Color Key:
- **Normal**
- **No section present**
- **Section SizeOfRawData = 0**
- **High Entropy (Packed or Compressed)**
- **Instruction not present in packed executable**
- **Operands don’t match**
Case Study: Mebroot

• Took latest Mebroot sample from Offensive Computing collection

• Analyzed inside of VERA

• Seemed to be idling for long periods of time

• Actually executed based on network traffic

• Hybrid user mode / kernel malware
Mebroot – Initial Busy Loop

- Initial analysis shows decoder for driver
- Sits for 30 minutes waiting for us to get bored
- Moves on to the rest of the program
Mebroot – After Busy Loop
Mebroot – Entire View

30 Minute Busy Loop

Initialization

Main Unpacking Loop

Kernel Code Insertion
User Study

• Students had just completed week long reverse engineering course
• Analyzed two packed samples of the Netbull Virus with UPX and MEW
• Asked to perform a series of tasks based on the typical reverse engineering process
• Asked about efficacy of visualization tool
User Study: Tasks Performed

• Find the original entry point (OEP) of the packed samples
• Execute the program to look for any identifying output
• Identify portions of the executable:
  – Packer code
  – Initialization
  – Main loops
Overall Evaluation

0=No;
1=Yes

Likely to Use Again
Will Recommend

User 1 | User 2 | User 3 | User 4 | User 5 | User 6
--- | --- | --- | --- | --- | ---

Will Recommend
Likely to Use Again
Results of User Study

The bar chart above shows the results of the user study for different categories. The categories are:

- OEP S1
- OEP S2
- Initialization S1
- Initialization S2
- Main Loops S1
- Main Loops S2
- Likely to Use Again
- Will Recommend

The bars represent the scores given by users for each category, with 1 being the lowest and 6 being the highest.
Selected Comments

• “Wonderful way to visualize analysis and to better focus on areas of interest”

• “Fantastic tool. This has the potential to significantly reduce analysis time.”

• “It rocks. Release ASAP.”
Recommendations for improvement

• Need better way to identify beginning and end of loops

• Many loops overlap and become convoluted

• Be able to enter memory address and see basic blocks that match
Future Work

• General GUI / bug fixes
• Highlight temporal nature of execution
• Memory access visualization
• System call integration
• Function boundaries
• Interactivity with unpacking process
Conclusion

• Overall process for analyzing and reverse engineering malware is shortened

• Program phases readily identified

• Integration with existing tools

• Preliminary user study shows tool holds promise for speeding up reverse engineering
Questions?

• Source, tools, and latest slides can be found at:
  http://www.offensivecomputing.net

• If you use the tool, please give feedback

• Contact info: dquist@nmt.edu