Reverse Engineering by Crayon

Game Changing Hypervisor Based Malware Analysis and Visualization

Danny Quist
CEO, Offensive Computing, LLC
New Mexico Tech Department of Computer Science

Lorie Liebrock, Ph.D.
Department Chair, New Mexico Tech Department of Computer Science

Blackhat USA 2009

Abstract

Recent advances in hypervisor based application profilers have changed the game of reverse engineering. These powerful tools have made it orders of magnitude easier to reverse engineer and enabled the next generation of analysis techniques. We will also present and release our tool VERA, which is an advanced code visualization and profiling tool that integrates with the Ether Xen extensions. VERA allows for high-level program monitoring, as well as low-level code analysis. Using VERA, we'll show how easy the process of unpacking armored code is, as well as identifying relevant and interesting portions of executables. VERA integrates with IDA Pro easily and helps you to annotate the executable before looking at a single assembly instruction. Initial testing with inexperienced reversers has shown that this tool provides an order of magnitude speedup compared to traditional techniques.

Introduction

Modern compiled executables create a large level of complexity for the reverse engineer. In this situation the source code for the program consists entirely of assembly and creates an increased level of work for the analyst. There are several tools that are available for this task, however there still exists a dearth of information to sort through during the initial steps.

The primary task of a reverse engineer is to determine the functionality of a program. Although determining the intent of the code would be valuable, it is even more difficult to discern than the functionality is. Reverse engineers use a variety of tools to assist with the process. These include static disassemblers, debuggers, system call trackers, and scripting tools. Tracking the execution of a program is a task that is difficult as malware actively tries to avoid detection. It is imperative that modern tools measure and analyze executables with no detectable impact to the executable.

Binary only reverse engineering creates several specialized problems. First information is lost during the compilation process that could greatly aid the analysis of the program. Specifically information such as the local and global variables and function names are often omitted from execution. The compiler also optimizes much of the original code so often times portions of the underlying program flow can be lost as well.

Two main classes of tools exist: static and dynamic tools. Static analysis tools include disassemblers (such as IDA Pro), string searching tools, and signature matching systems (including many anti-virus programs). Dynamic analysis tools include system call and operating system state tracking, such as the Sysinternals' Processmon tool, and debuggers, such as OllyDbg, SoftICE, and GDB. The run-time of an executable is closely controlled with these tools.

Tools have been made that greatly reduce the workload of the reverse engineer. The most popular tool is the Interactive DisAssembler (IDA) Pro. Some visualization techniques have been employed that greatly speed the analysis of a program. Graph charting modules in IDA provide an overview of the program in a static method. The disadvantage to this is that the dynamic runtime information is lost. When using any reverse engineering tool, the underlying question of “Where do I start?” inevitably comes up. Traditionally reverse engineers rely on intuition and experience to guide their focus, however this is a fundamentally imprecise training process.

VERA Architecture

Our tool consists of three main parts. First we have modified the hypervisor-based monitoring framework, Ether, to monitor and track program execution including

memory reads and writes. Using the output of this data we construct a directed graph of all the basic blocks of an executable (represented as graph nodes). We use a weighted graph system from the Open Graph Display Framework (OGDF) to layout the graphs. The weight associated with each node is the number of times that block of code was executed in the analysis run. Similarly, edge weight is the number of times the particular control path was executed. The data is then displayed through the visualization interface. VERA provides a navigable interface to explore the code. It also links and connects to the IDA Pro reverse engineering tool to aid more detailed analysis.

The visualization front-end provides the interface to the user. This take sthe graphs generated by the ether monitoring system and presents them in a two dimensional view. We constructed the application using the OpenGL framework as an MFC application. This application then takes the candidate dumps from Ether and uses them to provide an integrated view with IDA Pro.

Each node has a corresponding color developed from the analysis of the packed executable. The colors and their characteristics are as follows:

- Yellow – Normal unpacked code
- Red – Execution in a section of code with high entropy. This calculation is performed with the same algorithm from the Python Pefile module.
- Green – Code executed in a section not present in the original (presumably packed) executable
- Light Purple – Execution inside of a section with no data in the original file
- Neon Green – Instructions that differ in memory from the original executable.

Conclusion

The VERA framework we have presented provides an enhanced method to speed reverse engineering. This tool has been used in a variety of commercial and academic settings to better understand the flow and composition of a compiled executable. The user study shows that the tool enhances analysis and lowers the total amount of time necessary to reverse engineer an executable.

We have modified the Ether analysis framework to better enable traditional analysis techniques, including our own visualization tool. These have been added to the mainline Ether system. There are several areas of future work that will be explored. First, better highlighting of the loops inside the executable will be implemented. This was a common request from users for the tool. Second, a kernel-based method of program analysis will be developed. This is in response to the transition of modern malware to the Windows kernel architecture. Finally a 3D visualization environment will be explored to provide further insight into program analysis.