C23 Memory Forensics: Reliable In-Memory Code Identification Using Relocatable Pointers

Irfan Ahmed, PhD, University of New Orleans, 2000 Lakeshore Drive, New Orleans, LA 70148; Vassil Roussev, PhD, Computer Science, 2000 Lakeshore Drive, 308 Mathematics Bldg, New Orleans, LA 70148; and Aisha Ali-Gombe, MS, University of New Orleans, Computer Science, 308 Mathematics Bldg, 2000 Lakeshore Drive, New Orleans, LA 70148*

After attending this presentation, attendees will understand that memory forensics requires the exact knowledge of code that is present in a physical memory in order to apply code-specific data-structure knowledge for extraction of forensic-relevant artifacts. Attendees will learn about the concept of the relocation table and its ubiquitous use in Microsoft® (MS®) Windows® executable files and how relocations can be used to create distinct fingerprints of executable code for extracting their in-memory pages from a memory dump. Using a comprehensive test of more than 50,000 executables, a large-scale experimental validation and statistical analysis of the methodology is presented. It shows that relocation tables are closely tied to the specific version of a code executed in memory and can be used to strongly identify (a piece of) code found during the analysis.

This presentation will impact the forensic science community by discussing a code fingerprinting technique that is able to identify a specific version of code in a memory. Unlike previous techniques, which require parsing of memory contents, disassembly of code, and identification of the address space of kernel and processes, this presentation will show that the derived fingerprints from relocations works reliably on a memory capture without any prior knowledge about the content of the memory. Prior techniques also have a limited focus on identifying operating system code and its specific version. The presented technique works on any relocatable code and is effective in identifying any MS® Windows® executable code.

The relocations are pointers in the code that need to be adjusted depending on where the executable is loaded into the memory. A relocation table in an executable file contains the locations of the pointers in the code, which are the offsets to the pointers from the beginning of the file. The table is only required during the file's load time and is then discarded from the memory. The presented technique extracts the relocation table from the file. It then divides an executable file into memory-size pages and uses relocations to create separate signatures (or fingerprints) for the pages of the executable file. A signature consists of offsets to pointers from the beginning of a page and the pointer values. The signatures are then used to search physical memory page-by-page to find the executable file pages with no intrinsic knowledge about memory content.

The presentation will address a number of additional challenges including paging and Address Space Layout Randomization (ASLR). Most code is, by default, pageable which implies that parts of the executable may not be physically present in volatile memory. This study shows that a *majority-wins* approach is sufficient to overcome this obstacle. ASLR presents a different challenge in that, on every execution, it loads the code starting from a randomly chosen base address and adjusts the necessary pointers in the loaded code. As a result, the representation of the code on-disk and in-memory diverge; however, this study identifies reliable invariants in this process, which enable the correct one-to-one mapping as well as the computation the actual run-time base address.

Code Fingerprinting, Memory Forensics, Relocation Tables