Chapter 23
Effective Malware Analysis Using Stealth Breakpoints

Amit Vasudevan
CyLab, Carnegie Mellon University, USA

ABSTRACT

Fine-grained malware analysis requires various powerful analysis tools. Chief among them is a debugger that enables runtime binary analysis at the instruction level. One of the important services provided by a debugger is the ability to stop execution of code at arbitrary points during runtime, using breakpoints. Software breakpoints change the code being analyzed so that it can be interrupted during runtime. Most, if not all malware are very sensitive to code modification with self-modifying and/or self-checking capabilities, rendering the use of software breakpoints limited in their scope. Hardware breakpoints on the other hand, use a subset of the CPU registers and exception mechanisms to provide breakpoints that do not entail code modification. However, hardware breakpoints support limited breakpoint ability (typically only 2-4 locations) and are susceptible to various anti-debugging techniques employed by malware. This chapter describes a novel breakpoint technique (called stealth breakpoints) that provides unlimited number of breakpoints which are robust to detection and countering mechanisms. Further, stealth breakpoints retain all the features (code, data and I/O breakpoint abilities) of existing hardware and software breakpoint schemes and enables easy integration with existing debuggers.

INTRODUCTION

Malware -- a term that refers to viruses, trojans, worms, spyware or any form of malicious code -- is widespread today. Given the devastating effects that malware have on the computing world, detecting and countering malware is an important goal. Malware analysis is a challenging multi-step process providing insight into malware structure and functionality, facilitating the development of an antidote. To successfully detect and counter malware, malware analysts must be able to analyze them in binary, in both coarse- (behavioral) and fine-grained (structural) fashions.

Microscopic malware analysis is a fine-grained analysis process that provides insight into malware structure and inner functioning. This helps in gleaning important information about a malware
and facilitates the development of an antidote. With malware writers employing more complex and hard to analyze techniques, there is need to perform microscopic analysis of malicious code to counter them effectively.

Fine-grained analysis requires the aid of various powerful tools, chief among them being a debugger that enables runtime binary analysis at the instruction level. One of the important services provided by a debugger is the ability to stop execution of code being debugged at an arbitrary point during runtime. This is achieved using breakpoints, which can be of two types: Hardware and Software.

Hardware breakpoints are provided by the CPU and support precise breakpoints on code, data and I/O. They are deployed by programming specific CPU registers to specify the breakpoint locations and type. Software breakpoints on the other hand are implemented by changing the code being debugged to trigger certain exceptions upon execution (usually a breakpoint exception). However, both hardware and software breakpoints are severely limited in the context of malware.

Current CPUs however, contain support for only 2–4 hardware breakpoints. This imposes a serious restriction on the analysis process in the context of malware, most if not all of which carry complex polymorphic and metamorphic code streams (Szor, 2005). Further, many malware employ efficient anti-debugging primitives to prevent hardware breakpoints from being used altogether. For example, the W32/HIV virus uses CPU debug registers and the breakpoint exception for its internal computations while the W32/Ratos employs the breakpoint exception to handle its decryption in kernel-mode.

Current software breakpoint techniques either rely on the availability of program sources or entail modification to the executing code, both of which are unsuitable in the context of malware. Most malware exist in binary form and possess self-modifying and/or self-checking (SM-SC) capabilities. Further, software breakpoints do not support I/O breakpoints. Also, there are speculations regarding the correctness of certain software breakpoint implementations. The combination of trap and single-stepping may result in missed breakpoints in a multithreaded program if not correctly implemented by the debugger (Ramsey, 1994).

In contrast, stealth breakpoints provide unlimited number of code, data and I/O breakpoints, which are robust to detection and countermeasures. While various ideas using virtual memory for breakpoint purposes have been explored in many debuggers (Beander, 1983; Loukides et al., 1996; Compuware Corp., 1999), they allow only data read and/or write breakpoints. Also none of them are specifically tailored for malware analysis and their breakpoint implementation can be easily detected and thwarted. In contrast, stealth breakpoints leverage virtual memory, single-stepping and CPU task state segments (TSS; used to store all the information the CPU needs in order to manage a task. This includes the processor registers, stack and the task’s virtual memory mappings) to provide a stealth and portable breakpoint framework highly conducive for malware analysis. Further, the framework can be easily integrated into existing debuggers and its performance is within limits to suit interactive debugging.

This chapter starts with a brief background on breakpoints in general and discusses related work. This is followed by a detailed description of the design and implementation of stealth breakpoints. Following that is a discussion on analysis experiences with several real-world malware and the framework performance. The chapter concludes with a summary of stealth breakpoints and directions for future research in this space.

**BACKGROUND AND RELATED WORK**

Breakpoints — debugging aids that provide the ability to stop execution of code at an arbitrary