Abstract

Analyzing unknown programs and determining their semantic and implementation details is a critical task for computer security. Especially in the context of malicious software it is an indispensable requirement for protecting systems and mitigating the effects of attacks and security breaches. The insights gained with the help of reverse engineering are necessary for developing protection mechanisms and security software, to clean up infected systems, and to forensically reconstruct the details of occurred attacks.

Such analysis can be performed statically or dynamically, and both approaches have their advantages and drawbacks. While static analysis allows the examination of every single instruction of an application, it is time consuming and complex. Current malicious software further complicates it by using protection methods such as encryption, obfuscation, or virtual machines. Dynamic methods speed up the analysis process by supplying runtime values of relevant memory locations and register values. As a downside, they only keep track of one single execution path and are unable to deliver complete results with respect to code coverage. Nevertheless, they can be automated and also be utilized for generating behavior reports of unknown software. Only by using such methods it is possible to cope with the ten thousands of new malware samples that emerge every day.

Often software emulators are used for dynamic analysis, since they provide full control and isolate the underlying system from the observed malware. Due to the high complexity of modern CISC architectures, emulation is never perfect, and hence can be detected and evaded. Therefore, our approach is to instrument real systems instead and we examine which hardware and software facilities can be used to that end. We start by utilizing the operating system's memory management, namely by intervening with the MMU and modifying the page fault handler. By further enforcing illegitimate code to always reside in non-executable memory, we are able to intercept all attempts to execute it. Then we make use of the branch tracing facility of contemporary CPUs, which enables us to reconstruct the complete execution path after a program's termination. We further realize API and system call hooking to obtain insight into the interaction between the observed malware and the operating system. Finally, we utilize hardware-virtualization and develop a novel form of intermodular transition monitoring to further improve our analysis.

Our requirements for effective and efficient program analysis are transparency, isolation, soundness and performance. While all of our proposed methods are fast and sound, our hypervisor-based approach in particular fulfills these requirements to a high degree. By combining virtualization with module transition monitoring, we realize a valuable trade-off between executing performance and result quality. Its capability to analyze powerful 64-bit kernel rootkits in particular makes it a powerful and meaningful completion of our work.