Forensix: A Robust, High-Performance Reconstruction System

Ashvin Goel* Mike Shea Sourabh Ahuja Wu-chang Feng
Wu-chi Feng David Maier Jonathan Walpole
Department of Computer Science and Engineering, OGI @ OHSU

When intrusions occur and systems are compromised, one of the most time-consuming, error-prone, and expensive operations is determining what happened. Unfortunately, current forensic systems and techniques fail to provide the ability to quickly and accurately reconstruct system activity. Often full-time investigators are employed to either do a post-mortem on the compromised machine or to instrument machines to monitor subsequent activity. In addition, because of the lack of forensic information on the initial intrusion, a large amount of manpower and time is lost attempting to piece together enough evidence to track down the responsible parties. For example, in the well-publicized cases of Mitnick, the Melissa virus, and Mafiaboy, it took several full-time investigators months of time and effort to finally nail their suspects.

With the precipitous drop in computing, networking, and storage costs, the ability to do large-scale auditing and analysis of system activity is now technically and economically feasible. In this project, we are constructing an experimental computer forensic system called *Forensix* that will allow system administrators, law enforcement officials, and security experts to quickly and easily track down sources of security incidents after they have happened.

The Forensix system itself consists of three main components: a system call auditing module that is installed on a target system such as a honeypot or a sensitive server, a backend system that inserts system call audit records into an optimized database, and a set of tools for reconstructing system state via queries on the database backend. These components are described below:

Target system: This component consists of a loadable kernel module for capturing and transmitting system calls, their parameters, and their return values across a private network interface to a secure, append-only, backend system. Monitoring at the system-call level within the kernel provides a high-resolution, application-independent view of all system activity, while transmitting the audit data in real-time over a secure channel, to a separate, hardened, logging machine makes the system resilient to a wide variety of attacks.

Backend system: This component receives the detailed audit trail from the target system and periodically inserts it into a database for subsequent querying. Given the amount of records being generated, it must support fast bulk loading with index creation and efficient post-mortem querying.

Reconstruction tools: In order to recreate system activity, a suite of tools for querying the database must be written. We have implemented powerful tools that perform session replays, reconstruct connection logs, and generate process traces to support forensic analysis. As an example, these tools can be used to quickly provide answers for queries such as

Query 1: Show all user sessions that executed `/bin/sh` from daemon processes other than `sshd`, `telnetd`, or `login` and group sessions by user.

Query 2: Generate a system activity log for all sessions that were generated from *Query 1*.

Query 3: Show all activity for a particular user session *S*, where *S* is denoted with a source IP address and port, a user ID, and a connection timestamp.

Perhaps the closest system to Forensix is a combination of BackTracker [2] and ReVirt [1]. BackTracker uses a timing-based approach to generate a dependency between processes, files and filenames and uses the dependency graph to detect intrusions. This approach is space-efficient but does not provide precise details about all the system activities. For example, it can show the steps that led to the modification of a sensitive password file but does not show the precise changes made to the file. For the latter information, BackTracker must be used in combination with ReVirt, which places the system within a virtual machine and logs the VM-to-host instruction stream. The clear advantage of ReVirt is that it removes non-determinism by serializing all system activity at the logging point and hence allows complete system replay. However, unlike Forensix, ReVirt cannot support arbitrary queries without forcing the user to replay the entire instruction stream. On a heavily loaded system, such replay requires time that is proportional to the length of time the system has been running since the last checkpoint. Since forensic analysis is often an iterative process, such an approach defeats the initial goal of our work in reducing the time and human overhead required to perform forensic analysis.

Our initial Forensix implementation has been released. We hope to demonstrate the tool at the conference.

References

[1] G. W. Dunlap, S. T. King, S. Cinar, M. Basrai, and P. M. Chen. Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, December 2002.

[2] S. T. King and P. M. Chen. Backtracking intrusions. In *Proceedings of the Symposium on Operating Systems Principles*, October 2003.

*Presently at the Department of Electrical and Computer Engineering, University of Toronto.