A Framework for Volatile Memory Forensics

Ellick Chan, Winston Wan, Amey Chaugule and Roy Campbell
{emchan, wwan2, achaugu2,rhc}@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

Current techniques used by investigators during search and seizure operations typically involve pulling the plug on the suspect's machine and performing a post-mortem analysis of the contents of persistent storage. This process can potentially result in loss of volatile state such as the contents of memory which may contain vital information such as active network connections, encrypted disk sessions and running processes.

We have designed a framework, ForenScope, for volatile state analysis which allows the investigator to control and explore a suspect's machine through an interactive *bash* shell even if it is protected by logon or screen saver protection. Our tool relies on memory remanence properties to patch and alter the memory image of an operating system across a forced reboot to defeat authentication, neutralize anti-forensics software and scan for evidence of interest. To preserve the forensic integrity of the investigation, our tool is specially crafted to fit in unused memory. ForenScope then preserves a snapshot of memory to a USB stick and installs a software write blocker to prevent alterations to the disk without violating the semantics of running programs.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security, Forensics

General Terms

Security

Keywords

Security, Forensics, Memory Remanence

1. INTRODUCTION

Current techniques used by forensic investigators during search and seizure operations generally involve *pulling the power* on suspect machines and performing traditional *dead box* post-mortem analysis on the persistent storage medium. However, with the growing focus away from standalone applications on individual machines, more and more interesting forensic information resides beyond the realm of the hard disk and into the domain of the networked and online domains. Information collected from web browser sessions, VPN connections, IM and e-mail can provide information relevant to an investigation. Several trends such as the growing size of memory, anti-forensic techniques and the use of drive encryption present further challenges which limit the usefulness of traditional forensic techniques to gather crucial evidence.

The objective of this work is to investigate methodologies to capture and analyze volatile state – information that would be lost if an investigator cuts power to a suspect's machine. ForenScope, based on BootJacker[2] allows an investigator to gain access to a machine in the form of an interactive shell and custom analysis modules through a forced reboot. The key insight that enables this technique is that the contents of memory on many machines are preserved across a warm boot. Upon reboot, ForenScope patches the contents of residual memory to kill screen savers, neutralize anti-forensics software and bypass other authentication mechanisms.

The key steps:

1. Physical access to the suspect machine is obtained.
2. The suspect machine is forced to immediately restart.
3. A ForenScope bootable device is connected to the suspect machine.
4. ForenScope is booted instead of the host operating system.
5. ForenScope revives the host software environment and allows the investigator to access the system and run custom analysis tools.
6. Optionally, the machine can be confiscated for further analysis.

To preserve forensic integrity during the investigation process, ForenScope is designed to minimize *distortive alterations* to the system, which can be caused by running forensic tools on a host machine. These actions can taint a computer's state and cause *forensic blurriness* by overwriting deallocated memory or disk blocks which may hold key information. ForenScope avoids this tainting problem by running directly from unused conventional memory and blocking disk writes during the investigation phase.

Major contributions of this work are:

1. A method to explore interactively the volatile state of a system through a standard *bash* shell.
2. A method to gather, query and snapshot system state from memory.
3. An implementation of a software write-blocker.

2. RELATED WORK

Recovering forensic data, such as the process list of a running computer has been explored in the context of analyzing memory dumps using data structure organization derived from reverse-engineered sources[3, 4]. FATKit is a related framework that attempts to reconstruct data from memory[8]. Lee[6] presents a comprehensive description of forensic methods and data evaluation. Anti-forensics is the art of deceiving forensic analysis and malware such as the FU rootkit shows how this can be accomplished through a technique called direct kernel object manipulation (DKOM) which can be used to corrupt a process list. Malware researchers have demonstrated techniques to escape memory analysis through the use of a low-level rootkit[9] and that certain virtualization-based rootkits may be very hard to detect[5]. To produce sound memory dumps which can resist some of these attacks, hardware-based approaches using a PCI card to access memory have been developed[1]. Since the process of memory acquisition has been attacked in numerous ways, a unified service provided by the operating system [7] has been recommended to provide consistent memory dumps.

3. FORENSCOPE

In a forensics sense, ForenScope can support an unobtrusive method for analyzing a machine in a semi-quiescent static state. Immediately after the reboot, code may be run with full access to the system resources, while the original operating system and existing processes are suspended. Tainting of memory is limited because the framework resides only in conventional memory. Using ForenScope as an entry point into a system affords forensic investigators a level of information trust because any user-level software that may have been running is rendered harmless. Kernel-level malware, such as those which patch the kernel process list, may still affect results. While in the ForenScope context, forensic analysis modules may iterate through various interesting kernel data structures such as *task_structs* that identify processes, and *file_structs* that identify open files, network connections, and inter-process sockets. These pluggable modules help facilitate the forensic investigation process.

RootShell
RootShell is a superuser shell spawned by ForenScope. Investigators can use this shell to explore the system interactively or run customized analysis tools, including loading other kernel modules or other forensic modules. At the point that RootShell runs, Terminator has already cleared the system of security and anti-forensic software.

When the RootShell module is invoked by ForenScope, it starts a new thread in the host kernel which in turn invokes the *kernel_execve()* function to load and run the shell program on the host computer with superuser privileges. The shell is spawned on one of the unused text mode virtual consoles (accessed by special keyboard sequences) of the Linux system. Like ForenScope, this module is non-persistent and does not leave any trace in the system; however, some operations that the investigator performs using the shell may leave distortive alterations if BitBlocker (below) is not used.

Scribe
Scribe snapshots basic information regarding the investigation such as the time, date, list of PCI hardware devices, processor serial number and other features identifying the source of a snapshot. These details are then stored along with the evidence collected by other modules to identify the source of a snapshot. After Scribe collects basic hardware identification information, it records other useful system state such as the contents of the kernel *dmesg* buffer, kernel version, running processes, open files and open network sockets.

Cloner
Cloner is a memory dump forensic tool that is able to capture a full image of volatile memory contents without worry of being tainted by rootkits or booby traps. Using Cloner, an analyst can capture a pristine copy of system memory to a trusted device.

ForenScope works around forensic blurriness issues and rootkit cloaking by running before the original host OS is re-executed and directly accessing memory and disks without relying on any part of the incumbent operating system or page tables. Instead, it relies directly on its own memory system by using Intel's *protected mode* memory access with its own set of page tables that cannot be tainted by corrupted settings in the incumbent operating system. To dump the contents of memory to disk, Cloner relies on BIOS disk accesses rather than using a direct disk driver.

Terminator
Terminator assists a wide variety of analyses by sending kill signals to security, logging and anti-forensic software on the system. It kills the system logger daemon, antivirus software, intrusion detection tools and other security software. Terminator accomplishes this task by scanning the process list in the kernel and sending appropriate termination signals to the victim processes.

Terminator eliminates the presence of certain security processes and threads such as the screensaver, keyboard lock and potential malware or anti-forensic packages. Terminator sends a SIGKILL signal to all targeted processes rather than a SIGTERM signal so that there is no opportunity to ignore the signal. Customized signals can be sent to each target process. For some system services that respawn such as certain security applications, changing the state of the process to a zombie (Z) or uninterruptible disk sleep (D) is desired over just killing the application and waiting for it to respawn. An alternative would be to send SIGSEGV to certain applications to mimic a crash.

BitBlocker
In the field of forensics, minimizing distortive alterations caused by the analysis process is highly desired. Since actions performed as part of the exploration process through RootShell can inadvertently leave undesired tracks, we have developed a module called BitBlocker which creates a vir-

tual read-only disk by hooking the disk writing functions in the kernel. Simply re-mounting a disk as read-only to avoid writes may cause some applications to fail because they need to use temporary files or they expect open files to remain writable. To preserve expected file semantics and prevent these failures, BitBlocker relies on modifying the kernel's buffer cache layer. When an application creates or writes files, these operations are generally not immediately flushed to disk. The buffer cache manages intermediate disk operations, and also handles subsequent reads from files with pending writes to the disk. Linux implements some RAM disks by creating a file system on top of the buffer cache without a disk as a backing store.

In the normal case each time a disk write is executed, barring a *sync* operation, the operating system's disk buffer subsystem will hold the write in the buffer until a certain threshold or timeout is reached. In Linux, a system thread called *pdflush* handles the disk flushing process. By interfering with the operation of this daemon and the filesystem write layer, we are able to prevent flushes to disk. Further hooking at the disk layer to block writes ensures that BitBlocker catches all paths which may alter the disk. BitBlocker alters the write threshold of the disk to avoid disk buffer flushing and hooks the *sync, sync_file_range, fsync, bdflush* and *umount* system calls.

4. RESULTS AND EVALUATION

ForenScope and the tools that we developed were tested on the 2.6 Linux Kernel to show validity. Although this project was limited to the scope of Linux systems, further work is being pursued to expand this work to a range of operating systems and system configurations.

The ForenScope-based tools for retrieving the running process list, open file descriptors, and open network connections were successful in reflecting the same data as the output of the program ps on an uninfected system. Attempts to run all the forensic tools on real hardware were as effective as tests on QEMU. Although we have yet to run tests on malware-infected systems, the methodology behind the ForenScope process listing circumvents any application-level attacks on displaying system information. Resuscitation of the targeted machine after running these tools was verified to still work as in the original BootJacker. The major limitation of ForenScope is the shortage of memory space that it can use. Any use of memory exceeding the free space in conventional memory could cause memory corruption or tainting.

The acquisition of physical memory images was tested to be successful under various compression levels by comparing MD5 hashes as well as by using the program *diff*. The time taken to produce the memory dumps on physical hardware was found to be significantly longer than on simulated hardware. Table 1 shows the time elapsed and the compression levels for various test runs of the memory dump. The compressed version results in much few disk writes, and therefore runs faster than the uncompressed dump. However, the emulated hardware calls in QEMU do not accurately depict the time overhead need to repeatedly call BIOS functions in real mode.

Table 1: Memory Dump Speed and Compression

Test Conditions	Time Elapsed	Compression
No compress, emulated	21 s	0%
Compress, emulated	16 s	88%
Compress, hardware	15 s/MB	88% (theoretical)

The disk preservation mechanisms were shown to be effective against manual calls of sync, copious amounts of data copying through commands such as *cp* and *dd*, and unmounting of the disk. In the normal case, each of these actions should result in the buffer caches to be pushed to disk. However, after installation of the aforementioned mechanisms, no disk modifications were saved.

5. CONCLUSIONS

We have designed and implemented a tool for the exploration of volatile forensic state that may be lost by using traditional post-mortem analysis. As a result, investigators now have a powerful tool to study and capture the state of live machines. Our framework provides investigators and incident responders with the ability to clearly and confidently analyze the state of business and critical infrastructure machines without shutting them down.

6. REFERENCES

[1] B. Carrier and J. Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1(1):50–60, 2004.

[2] E. M. Chan, J. C. Carlyle, F. M. David, R. Farivar, and R. H. Campbell. Bootjacker: compromising computers using forced restarts. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 555–564, New York, NY, USA, 2008. ACM.

[3] B. Dolan-Gavitt. Searching for processes and threads in microsoft windows memory dumps. *Digital Investigation*, 3:10–16, 2006.

[4] B. Dolan-Gavitt. The vad tree: A process-eye view of physical memory. *Digital Investigation*, 4:62–64, 2007.

[5] S. King and P. Chen. Subvirt: Implementing malware with virtual machines. In *2006 IEEE Symposium on Security and Privacy*, pages 274–287, 2006.

[6] S. Lee, H. Kim, S. Lee, and J. Lim. Digital evidence collection process in integrity and memory information gathering. In *Systematic Approaches to Digital Forensic Engineering, 2005. First International Workshop on*, pages 236–247, 2005.

[7] E. Libster and J. Kornblum. A proposal for an integrated memory acquisition mechanism. *ACM SIGOPS Operating Systems Review*, 42(3):14–20, April 2008.

[8] N. Petroni, A. Walters, T. Fraser, and W. Arbaugh. FATKit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, 3(4):197–210, 2006.

[9] S. Sparks and J. Butler. Shadowwalker: Raising the bar for rootkit detection. In *Black Hat Japan*, pages 504–533, 2005.