Proceedings of the 2010 CAE Workshop on Insider Threat
November 2010

Performing Live Forensics on Insider Attacks

Ellick Chan, Amey Chaugule, Kevin Larson and Roy Campbell
{emchan, achaugu2, klarson5, rhc}@illinois.edu

Abstract- An insider with trusted access, procedural and architectural knowledge of the system and its security blind spots has much greater potential to cause significant damage than an external attack. Insiders, armed with knowledge of the internals of the security and policies of a system are able to more easily slip past the security mechanisms that protect the system. To perform a meaningful forensic trace back, it is useful to preserve as much volatile evidence as possible. This includes the entire state of the running system with its open network sockets, encrypted file systems and processes. Preserving these resources can be helpful in identifying live ongoing attacks, dormant sleepers, and identifying the perpetrator.

Although specialized forensic tools can be used, they often induce side effects and may cause significant disruption during the evidence gathering process. This may affect the quality and quantity of artifacts gathered by tainting in the evidence. To allay these concerns, we present Forenscope, a lightweight tool which provides real-time forensic analysis of a system without any of the mentioned problems. We demonstrate that Forenscope can run in under 15 seconds and detect tools such as rootkits which may hide malicious software planted by an insider.

Index terms- forensics, security, introspection, memory remanence

I. Introduction

In order to mount a compelling and comprehensive investigation of insider attacks, we need to collect information with the highest fidelity. This must be done without disturbing critical business processes, and any tools must be small and unobtrusive. Our processes must fit to the guidelines of the Department of Justice (DOJ), the SANS Institute, and other accepted practices. In recent years, the forensic community has become more receptive to collecting the contents of volatile memory, which can be used as a source of evidence to help combat cybercrime [3,4]. To support this trend, the SANS Institute recently published a report on volatile memory analysis [6].

Contemporary tools mostly perform disk based analysis and thus are unable to unable to preserve the hardware and software state of a running system in its entirety. These post-mortem analysis tools involve shutting a computer down to inspect the disks. However, this process breaks network connections and unmounts encrypted disks, resulting in a loss of information and disabling critical processes. Live forensic tools can partially overcome these problems by inspecting active systems. However, the operation of these tools can *taint* forensic information in the memory or on disk. Qualitatively *blurriness* can be described as uncertainty in captured image. Techniques to record the state of the machine are known to result in forensic *blurriness* [19], wherein inconsistencies in the memory snapshot are introduced due to the operation of the system. As a result of these shortcomings, which introduce uncertainties in the evidence, courts have been hesitant to accept forensic data [2].

To help address the limitations of existing volatile memory analysis tools we present Forenscope [9], a tool for live forensics. Forenscope can capture, analyze and explore the state of a computer without disrupting the system or tainting important evidence.

The contributions of this work include:
1. A systematic methodology and tool for doing live forensics on a system compromised by an insider.
2. Efficient techniques to gather, snapshot, and explore a system without performing actions which may alert the insider or raise suspicion.
3. Implementation and evaluation on several machines and against several anti-forensics rootkits.

This paper is organized as follows: in Section II we introduce forensics techniques for analysis of insider attacks. Section III covers the design of Forenscope and Section IV evaluates its effectiveness. In Section V we cover related work and in Section VI, we draw our conclusions.

II. BACKGROUND

To ensure the fidelity of evidence collected, it is pertinent to follow procedural guidelines while investigating a system suspected of an insider breach. Failure to do so may cast doubts upon the validity of the gathered evidence.

Investigations typically follow the workflow detailed in the CERT guide on FBI investigation [10]. For your reference, the following steps are recommended:
1. Preserve the state of the system by backing up a copy of the logs and files left by the attacker.
2. If the incident is in progress then log the activity using real-time tools.
3. Document losses incurred by the organization as a result of this incident.
4. Contact law enforcement.

These guidelines are aimed at analyzing standard computer attacks. In contrast, analyzing an insider attack demands comprehensive real-time forensics to track the motives and actions performed by the insider. To uncover and investigate these attacks, live forensics tools are preferred, given their ability to provide access to the state of volatile resources such as active SSH and VPN sessions, file transfers etc.

In order to reach a comprehensive conclusion, evidence must be of the highest quality. Taint and blurriness are loss or damage to data caused by the use of forensic tools. With traditional live forensic tools, the contents of memory changes from the start to the end of the capture, and the final snapshot is different from the state of the memory at any point.

There are two types of techniques for obtaining a snapshot of a machine’s state, passive techniques and active techniques. Passive techniques are non-intrusive and capture the machines state externally, through the use of techniques like VM introspection. Unfortunately these tools frequently capture blurry snapshots due to their inability to freeze the state of the machine. Active techniques run into different issues. They capture images through the use of an agent, but as a result, the memory footprint taints the snapshot.

In contrast, the techniques employed by post-mortem forensics tools involve the capture of a pristine image of volatile memory. Most conventional live-forensic tools however are rather ineffectual in preserving the fidelity of the image and often induce taint in this process. Moreover, current forensics tools require a complete shutdown of a system for introspection which is impractical in case of critical monitoring systems.

To better perform forensics on insider attacks, we must first characterize them. Insider attacks span a large number of areas, the most common of which are listed below [18]:

1. Unauthorized extraction, duplication, or ex-filtration of data
2. Tampering with data (unauthorized changes of data or records)
3. Destruction and deletion of critical assets
4. Downloading from unauthorized sources or use of pirated software which might contain backdoors or malicious code
5. Eavesdropping and packet sniffing
6. Spoofing and impersonating other users
7. Social engineering attacks
8. Misuse of resources for non-business related or unauthorized activities
9. Purposefully installing malicious software

To specifically address these threats, we have implemented several modules that can check for the presence of malware, detect open network sockets, and locate evidence in memory such as rootkit modifications to help the investigator identify suspicious activity.

The techniques presented in this paper demonstrate that threats 2,4,5 and 9 can be detected using live forensic approaches such as Forenscope. Section IV describes this in more detail. On the other hand, threats 6 and 7 require organizational changes and are beyond the scope of this paper.

III. DESIGN

Forenscope performs its analysis by first entering a safe state. To use Forenscope, the investigator needs to issue a hardware reboot on the target system and boot off a USB stick or a CD. After the system reboots, Forenscope can be selected from the boot loader.

Algorithm

Forenscope has been designed to preserve memory to the fullest extent and only induces taint to the bottom 640 KB of computer memory. This memory, which is known as conventional memory, is unused by modern operating systems and applications.

Once Forenscope boots, it restores the pre-boot state of the system by utilizing the property of memory remanence of DRAM chips (Section VII.A). Studies have shown that data persists in DRAM chips for up to 5 minutes after removing the source of power [15]. This preservation property enables us to access the exact state of the system at the time of reboot.

Forenscope is also able to restore the state of the system, including network connections and the state of other peripherals. Unlike traditional forensics tools, which are unable to capture live resources for introspection, Forenscope's design allows it to provide for forensic analysis on a live running system.

Many device drivers are able to handle errant conditions of the hardware and Forenscope relies on this facility to restore the state of the hardware. For instance, 71% (86 out of 121) of Linux network drivers implement this instance.

Forenscope monitors the state of the system using introspection which allows it to operate without risks of introducing taint or blurriness. We term this as the *golden state*. In this state, the system is inactive and can be analyzed and in-memory data structures and operating system services can be examined and queried [9].

Forenscope uses its own resources below the operating system. The observation capabilities afforded by Forenscope offer additional visibility in these scenarios. We show how Forenscope can fit into accepted workflows to enhance the evidence gathering process. For more details of the full algorithm and technical details, please see appendix Section VII.B.

Modules

In order to address the threats described in Section II, we have created a number of modules to aid the inspection process. While each module in this section provides a specific functionality to support the forensic process, a complete analysis generally requires the use of two or more modules in tandem to collect sufficient evidence. To illustrate the use of these modules, we provide an example use of how an investigator would use Forenscope. According to the guidelines referenced in Section II, semantic artifacts such as word documents, passwords, and web history are of particular interest especially when analyzing insider attacks.

To address these guidelines, Forenscope employs a number of modules to collect these artifacts from memory dump, copies of the disk, and running processes. The typical workflow of Forenscope is to run these modules in the three stages described below. These modules run in groups where stage 1 modules run in the golden state to collect pristine information while stage 2 modules rely on OS services to provide a shell and block disk writes. Finally, after all the modules have run, stage 3 resumes the original operating environment.

Stage 1

Scribe: A basic functionality of any forensic tool is to collect information to uniquely identify a machine. Scribe is the first module to run when Forenscope is started. It collects basic hardware information about the machine. This is stored along with the data collected by the other modules to identify the source of a snapshot.

Cloner: Upon the completion of Scribe, Cloner is run. It is a memory dump forensic tool that is able to capture a high-fidelity image of volatile memory contents to a trusted device.

A capture of memory is necessary because certain anti-forensic rootkits deceive system tools and the only reliable way to detect these rootkits is to analyze the memory image. Existing memory capture techniques rely on system resources and are vulnerable to such deception techniques. Cloner avoids rootkit cloaking by running in stage 1 before control is returned to the original host OS. In the golden state, the system uses protected mode to access memory directly through Forenscope's safe memory space. Using this technique, Cloner accesses memory directly without relying on the incumbent operating system. To dump the contents of memory, Cloner writes to disk using BIOS calls instead of using an OS disk driver and most BIOS firmware supports read/write access to USB flash drives and hard disks. Once Cloner captures a clean memory dump, the investigator can run other modules tools that may alter the contents of memory without worry of tainting the evidence.

Informant: The next module, Informant builds on the information gathered by Cloner. It checks for alterations in program code and memory that may indicate a malicious infection. Such alterations have the potential to hinder the investigation process and Informant helps to assess the integrity of a machine before further analysis is attempted. After Informant verifies the system, it then records other useful information such as the contents of the kernel log, kernel version, running processes, open files and open network sockets. This information can help expedite the investigation process by capturing a part of the system's dynamic state as a starting point for an investigation.

Neutralizer: The last module to be executed in stage 1 is Neutralizer. Malware installed by an attacker (threats 4, 9) may attempt to disable forensic software. With the help of Neutralizer, Forenscope is able to counteract this threat. Neutralizer disables anti-forensic software by detecting and repairing alterations in binary code and key system data structures such as the system call table. These structures can be repaired by restoring them with clean copies extracted from the original sources. Since many rootkits rely on alteration techniques, Neutralizer can recover from the effects of common forms of corruption. Presently, Neutralizer is unable to recover from corruption or alteration of dynamic data structures. Neutralizer also suppresses certain security services such as the screensaver, keyboard lock and potential malware or anti-forensic tools by terminating them. Neutralizer selects processes to kill based on the analysis mode. For incident response on server machines, a white list approach is used to terminate processes that do not belong to the set of core services. This policy prevents the execution of unauthorized applications that may cause harm to the system.

Stage 2

BitBlocker: During the forensic process, the act of examining and exploring the system can alter the contents of the system. A running program, performing any action may create a temporary file or write to the swap file. In order to prevent this, BitBlocker is the first thing to be run in Stage 2.

BitBlocker is a software-based write blocker that inhibits writing to storage devices to avoid tainting the contents of persistent media. Since actions performed by ForenShell during exploration can inadvertently leave undesired tracks, BitBlocker helps to provide a safe non-persistent analysis environment that emulates disk writes without physically altering the contents of the media. For additional details about BitBlocker, see appendix Section VII.C.

ForenShell: While these modules provide a variety of forensic capabilities, in some cases, more targeted applications are required. ForenShell is the last module to be run in Stage 2. It is a special superuser *bash* shell that allows interactive exploration of a system by using standard tools. When coupled with BitBlocker, ForenShell provides a safe environment to perform customized live analyses. In this mode, ForenShell becomes non-persistent and it does not taint the contents of storage devices. Once ForenShell is started, traditional tools such as Tripwire or Encase may be run directly for further analysis. To provide an audit log of the investigator's activities, ForenShell provides a built-in keylogger that writes directly to the evidence collection medium without tainting the disk. Forenscope launches the superuser shell on a virtual console. ForenShell runs as the last analysis module after Informant and Neutralizer have been executed. After these modules have been run, the system has been scanned for malware and anti-forensic software. If Neutralizer is unable to clean an infection, it displays a message informing the investigator that the output of ForenShell may be unreliable due to possible system corruption.

IV. Evaluation

To gauge the effectiveness of Forenscope against the insider attacks listed in Section II, we evaluated the bare boned implementation of Forenscope (without Cloner) against the four fundamental parameters of good real-time forensic tools: correctness, downtime, taint, and effectiveness against malware.

We have thoroughly tested and evaluated Forenscope for correctness on an SEL-1102, a power substation industrial computer, and an IBM desktop workstation. Systems such as SEL-1102 which oversee critical infrastructures are typically on isolated networks making remote attacks difficult leaving insider attacks as a better avenue for attackers to breach these systems. This coupled with the high value nature of the infrastructure makes studying insider attacks on these systems highly relevant.

Application	Results
Idle system	System is correctly recovered over 100 test cycles.
SSH	SSH session recovers, protocol handles lost packets.
PPTP VPN	VPN connection recovers, queued messages in tunnel are delivered.
AES pipe	File encryption continues.
Netcat	File transfers correctly without checksum errors.
DM-crypt	Mounted filesystem remains accessible.

Table 1: Correctness assessment

A. Correctness

To show that the implementation of Forenscope is correct and safe, we ran it against a barrage of test applications listed in Table 1. In each case, Forenscope was able to take control and run successfully without breaking the semantics of the application.

We chose a mix of applications to show that a wide range of hardware, software and network applications continue to work in a Forenscope environment. As a basic test, Forenscope was able to revive a running system with no load. SSH, PPTP and Netcat showed that the network connections persisted. DM-crypt and AES showed that the security programs continued to operate and all of these applications demonstrate that disk operations continued to work.

By demonstrating the correctness of Forenscope, we have shown that the system is returned to a state which appears unperturbed to the insider.

B. Downtime

To show that Forenscope minimally disrupts the operation of critical systems, we measured the amount of time required to activate the system. Without running Cloner, Forenscope executed in 15.1 s on the SEL- 1102 and in 9.8 s on the IBM Intellistation. Many network protocols and systems can handle a brief interruption gracefully without causing significant problems. We tested this functionality by verifying that VPN, SSH and web browser sessions continue to work without timing out despite the interruption. These protocols have a timeout tolerance that is sufficiently long to avoid disconnections while Forenscope is operating and TCP is designed to retransmit lost packets during this short interruption.

C. Taint

Taint is measured by the memory/disk footprint in bytes imprinted by the use of a forensic tool. To measure the taint created by Forenscope, we set up a system running Ubuntu 8.04 with 512M of memory in a virtual machine environment hosted in QEMU. We then compared the taint created by running Forenscope at regular time intervals that of various applications in a standard Linux environment.

From this evaluation we show that the taint caused by running Forenscope is less than 0.1 percent of the total memory. This is entirely contained within conventional memory and that this is much less than the taint caused by an idle Ubuntu system.

D. Effectiveness against anti-forensics tools

Many of the threats listed in Section II rely heavily on concealing the tracks of an insider attack. An

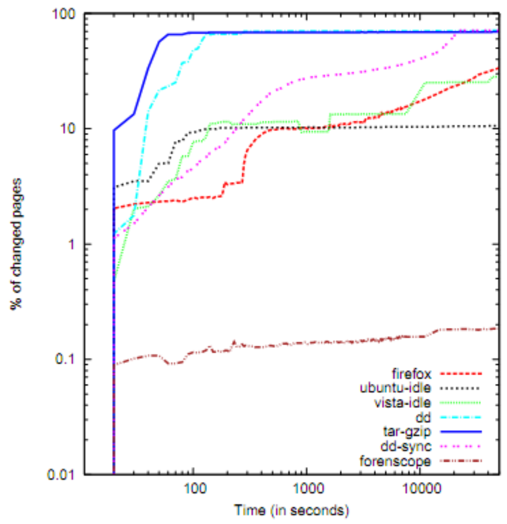

Figure 1. Comparison of Memory taint

insider's stealth not only is important for going undetected, but also allows for a greater depth and breadth of attacks.

Threats 1-5, 8, 9 (Section II) are all ideal applications of a rootkit. A rootkit is a collection of tools that allow intruders to conceal their activity on a computer so that they can covertly monitor and control the system for an extended period [7]. Many rootkits, especially the kernel-level rootkits, typically hook into the kernel to avoid detection by the admins. Once root access to the target machine is obtained, attackers can then install these rootkits on the system as loadable kernel modules. This allows insiders to maintain privileges to the system.

Although these anti-forensic tools take many shapes and forms, they usually employ rootkits to hide the malware control logic. In most cases, detecting the rootkit leads to detecting the malware. For instance, with the Stuxnet worm[21], which attacks industrial control systems, a rootkit was involved in the malware package. In many of these cases, detecting the rootkit will lead to the detection of the underlying malware planted by the insider.

These rootkits are not only hard to detect, but provide a backdoor enabling an insider to launch additional attacks. As they are kernel level programs rootkits can also hinder forensic analysis of an attack. Hence, rootkit detection and removal is integral to mitigating an insider attack. Forenscope was tested against a variety of rootkits, which are described below.

DR rootkit

The DR rootkit uses processor-level hardware debug facilities to intercept system calls rather than modifying the actual system call table itself. DR employs a hardware breakpoint which is reached every time a system call is made [13]. The breakpoint then intercepts the call and runs a malicious piece of code before returning control to the desired system call. Since Forenscope does not restore the state of debug registers, DR is rendered ineffective across the reboot, and hidden processes are revealed. Since DR was written to be a proof-of-concept, stealth wasn't the primary goal. As a result, Informant detects DR in several ways: DR is present in the module list, DR symbols are exported to the kernel and DR debug strings are present in memory. If an attacker modifies DR to make it stealthier by removing these obvious giveaways, we believe that it is still hard to deceive Forenscope, since the debug registers are cleared as part of the reboot process.

Phalanx B6

Phalanx hijacks the system call table by directly writing to memory via the /dev/mem memory device rather than using an easily-detected kernel module. It works by scanning the internal symbol table of the kernel and redirecting control flow to its own internal functions. Informant detects Phalanx by checking the system call table and common kernel pointers. Neutralizer restores the correct pointers to dispel the effects of Phalanx.

Adore

Adore is a classic rootkit which hijacks kernel pointers to deceive tools such as *ps* and *netstat* [1]. It operates by overwriting pointers in the /proc file system to redirect control flow to its own functions rather than directly modifying the *syscall* table. Informant detects that the pointers used by Adore do not belong to the original read-only program code of the kernel and Neutralizer restores the correct pointers. Restoration of the original pointers is simple and safe because the VFS function operations tables point to static functions such as proc readdir, whereas Adore has custom handlers located in kernel module address space.

Mood-NT
Mood-NT is a versatile multi-mode rootkit that can hook the system call table, use debug registers and modify kernel pointers. Because of its versatility, the attacker can customize it for each scenario. Like the rootkits described previously, Forenscope can detect Mood-NT in various modes. Our experiments indicate that Mood-NT hooks 44 system calls and Forenscope detects all these alterations. Furthermore, each hook points out of the kernel's read-only program code address space and into the space of the rootkit.

V. Related Work

Forenscope uses a wide variety of technologies to perform its forensic duties. Forenscope's introspection techniques are inspired by other introspection technologies, such as VMware's VMsafe[5]. The primary uses of VMsafe are to protect guest operating systems from malware. Projects such as Xenaccess [17] take the idea of introspection further and provide a way to list running processes, open files and other items of interest from a running virtual machine in a Xen environment. Although Xenaccess and Forenscope share a lot of functionality, Xenaccess, unlike Forenscope, requires that the system run on top of the Xen framework. Forenscope's techniques to recover operating system state from structures in memory dumps are influenced by Gavitt and Payne at al[12, 20]. Cozzie et al [11] used machine learning to detect data structures in memory. Forenscope builds upon works the virtual machine introspection community by allowing forensic analysis of machines not prepared for introspection. It provides a transparent and supports services such as BitBlocker. The recovery techniques used by Forenscope have been studied in the systems community. BootJacker [10] compromises a running operating system with a forced reboot.

VI. Concluding remarks

In this paper we have discussed insider attack patterns which can be detected using a real-time forensic platform such as Forenscope. We have shown that Forenscope is effective and practical to use and can perform live forensic analysis without necessitating a complete system shutdown.

While Forenscope is a powerful technical tool to assist the auditing process in the event of an insider attack, much work still remains to address the underlying social and organizational issues behind the attacks themselves.

We hope that the use of live forensic tools will lead to advancements in both the methodology and technology used in identifying and analyzing insider attacks.

Acknowledgements: We would like to thank the anonymous reviewers, Winston Wan, Mirko Montanari, and Shivaram Venkataraman for their valuable feedback. This research was supported by grants from DOE DE-OE0000097 under TCIPG (tcip.iti.illinois.edu) and a Siebel Fellowship. The opinions expressed in this paper are those of the authors alone.

VII. Appendix

A. Memory Remanence (summarized from [9])
Modern memory chips are composed of capacitors which store binary values using charge states. Over time, these capacitors leak charge and must be refreshed periodically. To save power, these chips are designed to retain their values as long as possible. This is especially important in mobile devices such as laptops and cell phones. Contrary to common belief, the act of rebooting or shutting down a computer often does not completely clear the contents of memory. Link and May [16] were the first to show that current memory technology exhibited remanence properties back in 1979. More recently, Gutmann [14] elaborated on the properties of DRAM memory remanence. Halderman et al. [15] recently showed that these chips can retain their contents for tens of seconds at room temperature and the contents can persist for several minutes when the RAM chips are cooled by spraying an inverted can of commonly available computer duster spray over them. The liquid coolants found in these duster sprays freeze the chips and slow the natural rate of bit decay. Forenscope utilizes memory remanence properties to preserve the full system state to allow recovery to a point where introspection can be performed. We refer the reader to [9,10,15] for a more detailed analysis of memory remanence.

B. Forenscope (summarized from [9])

Once the machine restarts, it enters the *golden state* monitor mode which suspends execution and provides a clean external view of the machine state. To explain how the monitor works, we first describe the operating states of the x86 architecture. When a traditional PC boots, the processor starts in *real mode* and executes the BIOS. The BIOS then loads the bootloader which in turn loads the operating system. During the boot sequence, the operating system first enables protected mode to access memory above the 1 MB mark and then sets up page tables to enable virtual memory. This is the OS bootstrapping stage. Once the OS is booted, it starts scheduling programs and user-level applications. Forenscope interposes on this boot sequence and first establishes a bootstrap environment residing in the lower 640 KB rung of legacy conventional memory and then it reconstructs the state of the running machine. Forenscope has full control of the machine and its view is untainted by any configuration settings from the incumbent operating system because it uses a trustworthy private set of page tables; thus rootkits and malware which have infected the machine cannot interfere with operations in this state. Next, Forenscope obtains forensically-accurate memory dumps of the system and runs various kinds of analyses described later in this section. To maintain integrity, Forenscope does not rely on any services from the underlying operating system. Instead, it makes direct calls to the system's BIOS to read and write to the disk. Therefore, Forenscope is resistant to malware that impedes the correct operation of hardware devices. The initial forensic analysis modules are executed in this state and then Forenscope restores the operation of the incumbent operating system. This entire process is executed in a matter of seconds and the system is restored fully without delay to minimize disruption.

C. Bit Blocker (summarized from [9])

Blocking writes simply re-mounting a disk in read-only mode can prevent further writes to the disc, it may cause some applications to fail because they may need to create temporary files and expect open files to remain writable. When an application creates or writes files, the changes are not immediately flushed to disk and they are held in the disk's buffer cache until the system can flush the changes. The buffer cache manages intermediate disk operations and services subsequent read requests with pending writes from the disk buffer when possible. BitBlocker mimics the expected file semantics of the original system by reconfiguring the kernel's disk buffer cache layer to hold all writes instead of flushing them to disk. This approach works on any type of file system because it operates directly on the disk buffer which is one layer below the file system. Although BitBlocker inserts hooks into the operating system, it does not interfere with the operations of Informant and Neutralizer because those modules are run before BitBlocker and they operate on a clean copy of memory.

VIII. References

[1] The Adore Rootkit. http://stealth.openwall.net/.

[2] *Electronic Crime Scene Investigation*: A Guide for First Responders, pages 25–27. National Institute of Justice, 2008.

[3] *Prosecuting Computer Crimes*, pages 141–142. US Department of Justice, 2007.

[4] SANS Top 7 New IR/Forensic Trends In 2008. http://computer-forensics.sans.org/community/top7_forensic_trends.php.

[5] VMware VMsafe Security Technology. http://www.vmware.com/go/vmsafe.

[6] K. Amari. Techniques and Tools for Recovering and Analyzing Data from Volatile Memory, 2009.

[7] B. Blunden. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Wordware, 2009.

[8] C. C. Center. How the FBI Investigates Computer Crime. http://www.cert.org/tech_tips/FBI_investigates_crime.html, 2004.

[9] E. Chan et. al. *Forenscope: A framework for live forensics* to appear in ACSAC 2010.

[10] E. Chan, J. Carlyle, F. David, R. Farivar, and R. Campbell.BootJacker: Compromising Computers using Forced Restarts. *In Proceedings of the 15th ACM conference on Computer and Communications Security*, pages 555–564. ACM New York, NY, USA, 2008.

[11] A. Cozzie, F. Stratton, H. Xue, and S. King.Digging for Data Structures.In *Symposium on Operating Systems Design and Implementation* (OSDI), 2008.

[12] B. Dolan-Gavitt. The VAD tree: A Process-eye View of Physical Memory. *Digital Investigation*, 4:62–64, 2007.

[13] J. Edge. DR rootkit released under the GPL. http://lwn.net/Articles/297775/.

[14] P. Gutmann. Secure Deletion of Data from Magnetic and Solid-State Memory. In *Proceedings of the 6th USENIX Security Symposium*, pages 77–90, July 1996.

[15] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, and J. A. Calandrino. Lest We Remember: Cold Boot Attacks on Encryption Keys. In *Proceedings of the 17th USENIX Security Symposium*, San Jose, CA, July 2008.

[16] W. Link and H. May. Eigenshaften von MOS-Ein-TransistorspeicherzellenbeitieftenTemperaturen. In *Archiv fur Elektronik und Ubertragungstechnik*, pages 33–229–235, June 1979.

[17] B. Payne, M. de Carbone, and W. Lee. Secure and flexible monitoring of virtual machines. In *Computer Security Applications Conference, 2007.ACSAC 2007. Twenty-Third Annual*, pages 385–397, 2007.

[18] M. B. Salem, S. Hershkop, S. J. Stolfo, A Survey of Insider Attack Detection Research http://sneakers.cs.columbia.edu/ids/publications/Survey.pdf

[19] A. Savoldi and P. Gubian. Blurriness in Live Forensics: An Introduction. *In Advances in Information Security and Its Application: Third International Conference, ISA 2009, Seoul, Korea, June 25-27, 2009. Proceedings*, page 119. Springer, 2009.

[20] A. Schuster. Searching for Processes and Threads in Microsoft Windows Memory Dumps.The Proceedings of the 6th Annual Digital Forensics Research Workshop, 2006.

[21] CERT-In:Indian Computer Emergency Response Team: Stuxnet Rootkit. http://www.cert-in.org.in/virus/Stuxnet_Rootkit.htm, 2010.