A Survey on Forensic Event Reconstruction Systems

Abes Dabir*, AbdelRahman Abdou†, and Ashraf Matrawy‡
Carleton University, Ottawa, Canada
*Email: *abes.dabir@magorcorp.com, †abdou@sce.carleton.ca, ‡ashraf.matrawy@carleton.ca*

***Abstract*—Security related incidents such as unauthorized system access, data tampering and theft have been noticeably rising. Tools such as firewalls, intrusion detection systems and anti-virus software strive to prevent these incidents. Since these tools only prevent an attack, once an illegal intrusion occurs, they cease to provide useful information beyond this point. Consequently, system administrators are interested in identifying the vulnerability in order to (1) avoid future exploitation (2) recover corrupted data and (3) present the attacker to law enforcement where possible. As such, forensic event reconstruction systems are used to provide the administrators with possible information. We present a survey on the current approaches towards forensic event reconstruction systems proposed over the past few years. Technical details are discussed, as well as analysis to their effectiveness, advantages and limitations. The presented tools are compared and assessed based on the primary principles that a forensic technique is expected to follow.**

1. Introduction

A forensic event reconstruction systems system is a tool to help administrators and forensic analysts investigate a security breach that has occurred in a computer system. They achieve this by assisting the forensic analyst in reconstructing the sequence of events that led to the security breach, rather than having the analyst attempt to manually go through a plethora of evidences and logs in order to perform this step. Such systems can be used to help in the analysis of a variety of security breaches such as an intrusion, or a worm infection. The remainder of this paper will mainly use intrusions as a specific type of security breaches in the discussions, although the material is valid for other similar forms of attacks as well.

Some of the more recent forensic event reconstruction systems proposed can automatically reconstruct the sequence of events leading to the intrusion and present this information using graphical output. The actual detection of an intrusion however is done independently of these tools either manually, or with the aid of another tool such as an intrusion detection system (IDS). We present a detailed discussion on the importance and challenges of automatic event reconstruction systems, the current state of the art in this field and the various solutions proposed.

Once an intrusion has been detected, investigators are generally interested in finding out:

- The initial point of intrusion on the system
- How to prevent further intrusions using the same attack vector
- What information has been accessed by the intruder and the damage inflicted
- How to undo the damage and modifications made by the intruder

An event reconstruction system is intended to help investigators and administrators answer some or all of the concerns listed above, depending on which specific system is being discussed. These systems should be installed on the target machine before it is exposed to a hostile environment and subject to attacks. Once installed, such systems typically initiate extensive logging of system activity on the machine. This logging is done by the forensic reconstruction system itself and is independent of any logs kept by the operating system of the machine, or the logs generated by other applications running on it. The exact nature of the data logged depends on the proposed reconstruction system being used.

The more recent event reconstruction systems proposed typically take as input a specific intrusion detection point that has been identified by an administrator, such as a new file appearing in the system or a suspicious process. The tool would then analyze the extensive logs at its disposal to backtrack the events that have influenced the intrusion detection point and present the investigator with one or more possible sequences of events that could have lead to the intrusion. The sequences of events identified by the tool would contain the initial source of the attack, thus helping the investigator understand where the attacker first penetrated the system and also how to prevent future attacks using the same technique. There are issues of false positives to be taken into consideration as the tool tries to relate the different events in the logs and identify the attack sequence.

Once the source of an intrusion has been identified, some event reconstruction systems are capable of analyzing the logs onward from that point to determine all the modifications that have been made to the file-system by the attacker. The system may also support the ability to automatically

. Version created on: 16th August, 2016.
The final version of this paper will appear in the International Journal of Information and Computer Security (IJICS). This is the authors' copy for personal use. © 2016 Inderscience.

rollback the malicious modifications it has identified and restore the system to a state where only legitimate operations persist. In doing so, there are potential conflicts to sort out between legitimate and malicious operations that will be discussed.

We present a survey of the state of the art “event reconstruction systems" proposed in the literature or commercially available. An event reconstruction tool is a special class of forensic systems. It aims to reconstruct the series of events that occurred during an attack incident. Such reconstruction provides the forensic analyst with better understanding to the system state, the damage that occurred, the exploited set of vulnerabilities and how to avoid similar future attacks. These systems have special requirements that we highlight in this paper.

The rest of this paper is organized as follows. Section 2 introduces basic principles that a forensic event reconstruction system should follow. Section 3 presents the currently existing tools that carryout forensic investigations. Counter forensic tools are also discussed. Next, in Section 4, the general theory of operation of forensic logging systems is described and the two main approaches towards forensic event reconstruction are overviewed; virtual machine and instrumented kernel. Sections 5 and 6 show the current research progress in forensic tools that are categorized as virtual machine based and instrumented kernel based respectively. The proposed solutions are assessed in Section 7. Finally, a conclusion is provided in Section 8.

2. Principles to Follow in Forensic Analysis Tools

[1] devised a set of high-level principles and qualities for forensic analysis in general. These principles are applicable to forensic event reconstruction systems, and are as follows:

1) Take the entire system into consideration rather than just certain subsystems or components.
2) Log information irrespective of any assumptions made about the threats, policies, or trust in users.
3) The logging process should not be limited to the actions driving the events. Rather, it should include the effects of these events as well.
4) Context, such as the state of the objects involved or the parameters being passed, is helpful in understanding and interpreting an event, therefore it should be considered.
5) Data should “be processed and presented in a way that can be analyzed and understood by a human forensic analyst." [1, p. 86].

Following all of these principles in a forensic event reconstruction system is impractical however, as they require the recording of a tremendous amount of data. Instead, a practical trade off has to be made between accuracy and the amount of data recorded [2]. How to achieve the perfect balance between these two factors is still an open research question.

[2] tried to advance their work on these principles by advocating a set of qualities for a good forensic model to have. They also proposed a forensic model that has the qualities they identified. Their proposed model requires identifying the ultimate goals an attacker may have on the system, as well as the intermediate goals and the starting goals. The attacker has to achieve a starting goal to begin the attack. Starting goals lead to numerous potential intermediate goals that an attacker has to achieve to get to the ultimate goal. It is likely that there are simply too many intermediate goals to take into consideration, therefore the model supports the use of metrics such as the severity of the resulting attack to filter out some of the intermediate goals. This reduces the amount of data that needs to be captured. In order to identify what data needs to be logged, the model works backwards from the ultimate goal to the starting goal, and identifies the pre-conditions and post-conditions of each goal. The data that needs to be logged is that which corresponds to the pre-conditions and post-conditions identified.

None of the forensic event reconstruction systems discussed in this paper fully adhere to the principles of the forensic model mentioned, perhaps for practical reasons. However, they represent advancements and could prove useful in the design of future forensic event reconstruction systems.

3. Current Approaches Towards Forensic Analysis

The current approaches (such as EnCase, SMART, Sleuth kit, Live View, and SIFT which are discussed later in this section) towards forensic analysis in practice are mainly oriented towards post incident analysis of a compromised host. On the other hand, the forensic event reconstruction systems being proposed and discussed in this paper aim to perform extensive logging of events on the host prior to the occurrence of a security breach. These logs would then be used after the attack to analyze and understand the intrusion.

With the current approaches, the first goal is to preserve both the non-volatile persistent data on the host’s hard drive, as well as the volatile data in the RAM. The proper way to analyze the hard disk contents is to make a duplicate image of the disk first and perform analysis on this copy. Traditionally, the focus of forensics has been on capturing the hard disk image, while analysis of the RAM contents has only been gaining momentum over the past few years [3] as the tools and techniques in this area mature.

There are many forensics tools that are designed to help the analyst extract useful data from a hard disk and analyze it. Some of these include Guidance Software’s EnCase Forensic [4], ASR Data’s SMART [5], and The Sleuth Kit [6]. These tools may be able to recover deleted files, show files hidden by rootkits, access data stored stealthily on the disk in unconventional ways, and much more. Another interesting tool called Live View [7,], which is developed by CERT, can take a physical disk or a disk image and mount it in a VM in a read-only manner such that all modifications to the image are written to a separate file instead of the actual disk to preserve its original state, therefore eliminating the need to make duplicate copies of the disk prior to analysis. The value of the results obtained by these forensics tools is dependent on the sophistication of the attacker in terms of how much data useful to investigating the attack has been left on the hard disk in a retrievable way.

SIFT [8] is an opensource forensics toolkit with a wide range of capabilities. It works in a similar manner as Live View, where the analyst should extract a memory image and plug it into a VM for analysis. In addition to the capabilities of commercial forensic tools, SIFT can carry out a larger view of investigation through network forensics. It can also perform malware analysis as well as a timeline generation, which includes a vast amount of metadata. This metadata contains details about the contents of the examined file and the actions performed on it (for instance, the last time the file was modified and who modified it) [9].

There is a number of anti-forensics tools designed to counter forensics tools by eliminating useful forensics data from the disk, or just make the task of forensic analysis more difficult on the data that remains. Examples of these include Timestomp [10], which allows the attacker to modify all the NTFS timestamps of many files at once, and the Defiler's Toolkit [11] which can be used to delete records of file I/O activity. Forensic event reconstruction systems can help in this area by incrementally recording modifications to the disk and files as they occur, and storing these logs in an immutable fashion for later analysis.

The RAM could contain valuable state-related information, or other data such as cached unencrypted file contents that may be very helpful during the attack investigation. More modern sophisticated attacks tend to try and stay resident in the memory, as opposed to touching the hard disk. Therefore, it is important to capture the contents of the RAM since the hard disk may not contain any evidence of the attack. Reliably capturing the contents of the RAM is somewhat difficult though. Using software for this task has the potential problem of having malware running, at the same privilege level, feed it incorrect information, as well as the issue of causing modifications to the system through the act of running this software [12]. Using pre-installed hardware in the form of a PCI card is the preferred solution for acquiring the RAM contents since it neither requires any software to be run on the machine, nor does it involve the CPU. However, even this technique is not fully reliable and could be subject to attacks [12]. Forensic event reconstruction systems might be able to help in this area by potentially being able to log activity that loads malicious code into the RAM. By incrementally logging system activity during and after the attack, they lessen the need to have to obtain the RAM contents for forensic analysis.

The main issues with both the hard disk image and RAM contents available to the forensic analysts is that even though they might contain valuable information about the final state of the machine, they typically don't contain detailed information regarding what occurred in the system during the attack. This is what forensic event reconstruction systems try to address by capturing data that would tell the analyst exactly what happened during the attack.

It is also possible that network logs may be available to the forensic analyst. One of the tools in this area that supports capturing network traffic and provides powerful analysis features is CA Network Forensics [13]. Amongst its features, CA Network Forensics is able to reconstruct instant messaging streams, emails, file transfers, chat sessions and more based on the captured packets. Its other features include the ability to play back captured packets, correlate network activity with security events received from other systems such as an IDS or firewall, pattern analysis to identify unusual traffic patterns, packet content analysis, and powerful visualization capabilities. The challenge with logged network data pertaining to an attack is that they may contain encrypted data, or be obfuscated by the attacker to prevent easy analysis.

4. Logging Events in Forensic Reconstruction Systems

The logging components of current operating systems typically log an insufficient amount of information to answer all the questions outlined in the beginning of this paper. As such, logging is a major component of any forensic event reconstruction system. Such tools must be able to collect accurate and detailed information on a wide range of system activities so that these logs can be used during the investigation phase to understand the nature of a security incident and its consequences. It is helpful to consider the principles outlined in Section 2 when designing the logging component of an event reconstruction tool.

In order to be effective, the integrity of the logs must be preserved. Since these logs serve as evidence, their value is diminished should they be tampered with. Securing the logs involves securing the logging facility, logging messages, as well as the log storage location. This is another area where the logging components of current operating systems are somewhat lacking. They generally store logs on the local file-system and put a lot of faith in the security of the kernel. Should the kernel be compromised, the logs stored on the local file-system can be manipulated, as well the system can be prevented from collecting useful logs after the point of compromise.

Virtual Machine Approach versus Instrumented Kernel Approach to Logging

The event reconstruction systems discussed in this paper mainly follow two different approaches to logging system activity. The first approach involves running the target operating system and applications within a virtual machine and logging the execution of that virtual machine at a very detailed level. The second approach tries to introduce a secure logging component into the kernel of the operating system to be monitored.

The next two sections will address logging integrity for each approach, where some of the more prominent event reconstruction systems proposed by the research community over the past several years are presented. Different approaches to logging in forensic event reconstruction systems are discussed as well. The two main solutions discussed are BackTracker [14] and Forensix [15], while the majority of the other solutions presented are variations or evolutions of these two.

5. Virtual Machine Approach

In general inspecting a virtual machine from the outside in order to analyze the software executing within it is called Virtual Machine Introspection (VMI) [16]. ReVirt [17], DACSA [18] and BackTracker [14] use this VMI approach to monitor a system and log its events. This approach involves running the operating system (guest operating system) and applications (guest applications) to be monitored inside of a virtual machine (VM) which is run on the host operating system and managed by a virtual machine monitor (VMM). The attacker's primary targets are the guest applications and guest OS. The host operating system is the one which runs natively on the actual physical hardware. The guest operating system does not need to be the same as the host operating system. Similarly, the emulated hardware does not have to be exactly the same as the actual physical hardware present. The VMM is primarily responsible for emulating the hardware that the virtual machine will see.

For the purposes of forensic event reconstruction, a new security service can be added to the VMM to monitor and log the execution of the VM. Alternatively the VMM code itself could be modified to perform the logging. The virtual machine based approach provides a good level of isolation between the logging system and the operating system being monitored along with its applications. Effectively, the logging software is placed at a layer beneath the guest OS. The log files could be placed on the file-system of the host OS which is not visible to the guest OS. This isolation ensures that if the guest OS gets compromised, the integrity of the logging component is not easily compromised. This is not to say the VMM itself cannot be subject to a security compromise. It is possible for a malicious program to recognize that it is running inside a VM, then proceed to exploit a bug in the VMM implementation in order to 'escape' that environment and try affect the host OS environment [19]. However, because VMMs generally have a small code size and a narrow interface that only provides critical functionality, it is much easier to validate their security than that of an operating system. The work done by [16] contains a number of suggestions on how to harden a kernel against attacks. Ideally, these should be followed to harden the guest OS kernel.

VMMs have three properties that make them particularly useful for the purpose of inspecting an application's execution [16]:

- Isolation: Software running in one VM cannot manipulate the software running in another VM, or the VMM.
- Inspection: The VMM has access to the entire state of a VM.
- Interposition: The VMM has the ability to interpose on certain events, or operations being carried out by the VM. For example, the ability to interpose on an interrupt in order to log it (this may require adding hooks in the VMM code).

One of the downsides of this approach is the potential overhead introduced by additional logging activities. ProTracer [20] however is a recently proposed system that can reduce logging overheads by refraining from blindly logging *all* system calls. Rather, it logs only when changes to the external environment or to the permanent storage are made. Another downside is that since the logging component is running at a lower layer than the guest OS, it generally doesn't have access to high-level information and data structures in the guest OS. Rather, it has access to low-level information such as the memory address space and IO operations. Such a gap between the host and the guest OS is called the "semantic gap". There are ways to overcome the limitations caused by semantic gaps. For instance, BackTracker's logging component is compiled with headers from the guest kernel and is able to read kernel data structures from the guest operating system's physical memory. The Plato project [21], discussed in Section 5.5, is specifically aimed at bridging this "semantic gap" [22] between the low-level view a VMM has of the guest OS activity, and the high-level state information present in the guest OS environment.

5.1. ReVirt

ReVirt [17] is one of the earlier works in the field of forensic event reconstruction systems. It was worked on by some of the same authors as the BackTracker system and has also served to influence the Forensix project. ReVirt utilizes single processor virtual machine monitoring techniques, as discussed in Section 5, to facilitate detailed logging of a system while trying to ensure a high level of integrity for those logs. These logs can then be used at a later time to perform replays of the virtual machine execution at an instruction-by-instruction level. The ability to replay the virtual machine execution at time points before, during, and after the attack, allows investigators to recreate and better understand an attack. ReVirt has been further extended to support multiprocessor VM logging as will be seen later on.

ReVirt's logging component utilizes the concepts of logging, checkpointing and roll-forward recovery to enable replaying virtual machine execution. A checkpoint is a previous state of the system that has been recorded. Roll-forward recovery involves starting from a checkpoint and resuming execution of the virtual machine. In order to replay virtual machine execution properly, certain non-deterministic events must be logged so that they can be reproduced during replay. Deterministic events such as (arithmetic, memory and branch instructions) are not logged by ReVirt since they are expected to be re-executed by the virtual machine during replay just as before. ReVirt authors classify non-deterministic events into two categories: time and external input. An example of a time event would be an interrupt, in which case ReVirt records the instruction at which the interrupt occurs so that it can be reproduced at the same instruction during replay. Many non-deterministic events occurring on the host machine that affect the operating system do not affect the execution of the virtual machine and are thus not recorded. Sources of external input include peripherals such as keyboard, mouse, CD-ROM, and the network interface card. Network messages are typically the largest type of data that gets logged. All relevant events are incrementally logged during the execution of the virtual machine.

ReVirt performs checkpointing by copying the virtual disk being used by the virtual machine while it is still powered off. The virtual disk contains the entire state while the virtual machine is powered off. Replays must begin with the virtual machine in a powered off state, and are initiated from a checkpointed state.

During a replay, ReVirt prevents new asynchronous interrupts from interfering with the replaying. An investigator can step into the virtual machine environment at any point during a replay to investigate its state by issuing commands. The ability to perform such an action greatly aids the investigator in analyzing the intrusion. However, this does perturb the replay and it is not possible to resume replaying the execution without going back to a checkpointed state first. It is also possible for an investigator to run tools such as debuggers and disk analyzers from outside the virtual machine environment during a replay to check its state information such as address space, registers, and disk data. Tools run from outside the virtual machine generally only have low-level access to state information, and they lack the flexibility of high-level commands one can run from inside the virtual machine. However these tools are still very useful. This is due to the fact that should the virtual machine itself be compromised by an attacker, the validity of any output obtained by running commands from within that environment is suspect.

The additional overhead introduced by ReVirt, as a result of running tasks in a virtual machine and logging, are important factors to take into consideration. The tests run by the ReVirt researchers reveal that the processing overhead and space requirements of ReVirt can vary greatly depending on the type of applications being executed. Running applications that have heavy dependencies on the guest kernel can incur an overhead of up to 58%, while the overhead for applications that constitute normal desktop usage are not significant [17]. The researchers believe that the security value added by ReVirt can outweigh the processing overheads observed. The logging action itself does not incur much of an overhead. The space requirements for logging can add up to a few gigabytes per day, with the largest logs being attributed to network traffic.

ReVirt's biggest shortcoming perhaps is the lack of any automated analysis tools to aid in the investigation process. Revirt fails to satisfy the fifth principle discussed in Section 2. The investigator still has to do a lot of manual work in order to identify the attack route and actions taken by the attacker after the intrusion. DACSA [18] on the other hand was recently proposed, overcoming that limitation by providing the ability of automated analysis. Yet this analysis is decoupled from the captured information (e.g., system calls), and could be done offline for preserving efficiency.

SMP-ReVirt. SMP-ReVirt is a modified version of ReVirt [23], which aims to log and reconstruct system state for multiprocessor VMs. The authors of SMP-ReVirt were motivated by the modern large-scale deployment of multi-core processors. The logging process on a single processor VM system differs from the one that uses multi-processor [23]. In multi-processor systems, processors compete for resources, particularly the memory. SMP-ReVirt advocates logging such race conditions in the proper manner for future retrieval when required.

5.2. BackTracker

The BackTracker [14] system is one of the prominent forensic event reconstruction systems proposed and has served as a solid foundation to a plethora of subsequent work in this area. It was worked on by two researchers from the ReVirt team, however BackTracker is not a direct continuation of that project.

The goal of BackTracker is to provide useful information to the forensic analyst in order to more easily understand the chain of events involved in an attack sequence. When an administrator detects a malicious modification to the system, he can feed this information into the BackTracker system. BackTracker would then try to go backwards from this detection point based on the logs it had collected and identify all the possible chains of events that could have led to this modification. This feature is in accordance with the fifth principle in Section 2 and should make the job of a forensic analyst much easier. In order to satisfy reasonable overhead and space requirements, BackTracker tries to collect enough information to cover most types of attacks as opposed to every possible attack.

There are two components to the BackTracker system: online event logging component called EventLogger, and an offline event analysis component called GraphGen, which generates dependency graphs. EventLogger is implemented using the virtual machine based approach. The log file is saved in the file-system of the host operating system. The VMM notifies EventLogger whenever a guest application process exits, or if it invokes or returns from a system call. The EventLogger then examines the state of the virtual machine and reads guest kernel data structures from its physical memory to collect the information it is interested in. EventLogger is able to access this high-level information in the guest kernel environment because it has been compiled with headers from the guest kernel. There is a downside to this approach that involves having to rely on large amounts of source code for the guest kernel; this could get very complicated [21].

EventLogger is mainly interested in tracking the flow of information between OS level objects and high-control events; It does not concern itself with tracking the flow of information between application level objects and events. High-control events are those which are easiest for an attacker to use in order to exert some level of control over the target object. These events are basically invoked system calls. BackTracker observes the event type along with information identifying the calling process and the object affected by the event.

The following OS level objects are of interest to the EventLogger [14]:

1) Processes
2) Files and named pipes
3) File names

The following high-control events, which may cause dependencies between objects, are logged by EventLogger [14]:

1) Process creation through fork or clone
2) Load and store to shared memory
3) Load and store to mmap'ed files
4) Read and write of files and pipes
5) Opening a file
6) *execve* of files
7) Receiving data from a socket

In order for GraphGen to begin constructing chains of events, the administrator must first identify a detection point and feed this into it as input. A detection point has to be in the form of one of the objects tracked by BackTracker. Some possible detection points may be a modified, extra, or deleted file, or a suspicious or missing process. If the analysis is taking place a long time after the intrusion occurred, it may not be very easy to come up with suspicious files and PIDs. BackTracker doesn't provide any aid in identifying a detection point, which serves as one of its weaknesses. Once it has been fed with a detection point, GraphGen will analyze the log stored by EventLogger and try to go backwards in the logs starting from the detection point in order to build a dependency graph of all objects and events that have causally affected the state of the detection point. GraphGen does apply certain filters to take out objects and events from the produced graphs which it considers as not relevant.

5.3. Bi-directional Distributed BackTracker

The Bi-directional Distributed BackTracker (BDB) [24] is an evolution of the BackTracker system. From a forensics standpoint the goal of the BDB project is to extend BackTracker's capabilities beyond a single host in order to track multi-host attacks within a single administrative domain.

In the original BackTracker system, once it was fed with a detection point in the form of an OS level object, it would generate a backward causality graph based on the logs, showing all relevant objects and events that had causally affected the detection point. The BDB system takes this a step further and introduces forward graphs. As intuitively expected, these graphs go forwards in the logs from the detection point and display the relevant OS level objects that have been causally affected by the detection point object. Forward causality graphs can help the forensic analyst answer the question of what actions were taken by the intruder on the system. Both backward and forward causality graphs are required tools in order to track multi-host attacks.

BDB's multi-host forensic capabilities are limited to machines that have the BDB system installed on them. These machines would be in the same administrative domain since typically an administrator would install the software only on the machines he administers.

BDB extends BackTracker's logging capabilities to track packet sends and receives on the network. If a process on one machine sends a packet to a process on another machine, this creates an inter-host causal dependency between them. In order to associate a packet send event in one host's log with the packet receive event in another host's log, a mechanism is required to track packets. BDB only tracks TCP packets. It does so by using their source and destination IP address, as well as their sequence numbers. BDB is capable of employing a number of different ways to prioritize which packets to include and follow in its graphs. This prioritization reduces the size of the graphs and speeds up the tracking by weeding out packets that are unlikely to be involved in the attack.

The proper way to use BDB in a network after a detection point has been identified on a single host is to generate the backward graph on that host in order to identify the intrusion point. BDB should be able to identify the packet that caused the intrusion on this host, and if it came from another host within the same administrative domain, BDB can continue the backward analysis on that host and any other along the way until it finds the original point of entry into the administrative domain. For every host that was found to be infected or compromised, BDB's forward analysis should be run to identify all the other hosts these machines may have also compromised.

One of the limitations of BDB is that if its tracking leads to a host that doesn't have BDB installed then the tracking will not be able to proceed. Similarly to BackTracker, BDB only follows certain types of dependency causing events which are most likely to lead to attacks. This could lead to certain weak spots where attacks would be missed. An attacker may also try to cause a lot of dependency forming events and thus create a lot of noise in the graphs which will be generated. As well, an attacker could try to implicate non-malicious processes and hosts in the causality graphs to hide his actions and make the analysis much more difficult.

5.4. Improved BackTracker

[25] aim to improve upon the BackTracker system by reducing the size of the dependency graphs it produces. Their technique is referred to in this section as the Improved BackTracker. Improved BackTracker adds two modifications to the original BackTracker system in order to achieve its goals. The first modification involves the logging component which now also records the parameters and return values of all system calls. This is done so that it can capture

the file offset information when a read or write operation is perform. The second modification introduces data flow analysis within processes to better identify relevant events in the dependency graph. These improvements come at the expense of larger log files and additional processing overhead when generating the dependency graph.

A file offset interval identifies a specific portion of that file. By recording file offset intervals, Improved BackTracker is able to tell which location in a file was read from, or where exactly data was written to in a file. The purpose of this is to reduce the number of dependencies BackTracker would identify when multiple processes are interacting through the same file. For example, with the original BackTracker if process A writes to a file from which process B reads, BackTracker would form a dependency between the two processes through the file in question. BackTracker treats files as black boxes with no insight into them, therefore even if the two processes in the previous example write and read from completely different locations in the file, they still get linked. Improved BackTracker would be able to recognize that the two operations on the same file do not interact with the same region of data in the file and would not link the two processes. This feature should help make the generated dependency graphs less crowded and easier to analyze.

To achieve best results with the file offset tracking feature of the Improved BackTracker, the analyst choosing a detection point that is a file should also try to identify the suspicious offset interval within that file. This may or may not be possible for the analyst to do depending on the file type (more difficult with binary files as opposed to text) and how much information is available about the attack.

The data flow analysis improvements use program slicing techniques on processes to better understand the execution paths within them. For example, if there are multiple inputs from different sources into a process and it is unclear which has lead to a particular output, using program slicing it may be possible to narrow down the set of input sources that may have led to that particular output. In the original BackTracker, processes are treated as black boxes and any input into them could be the cause of the output, therefore they are all further pursued in the event logs and displayed in the graph. Applying program slicing techniques could further prune irrelevant events and objects from the generated dependency graph.

Program slicing is not very straight forward to apply however. It may require access to the source code for the programs being analyzed, as well as requiring object files to have been compiled with special debugging parameters beforehand. Depending on which particular program slicing technique is being used, it may introduce large overheads into the log analysis and graph generation phase.

5.5. Plato

Plato [21] is not a forensic event reconstruction system on its own, rather it is platform that can help in the development of VM-based tools such as event reconstruction systems. It has been developed by the BackTracker researchers, as well as another researcher who worked on ReVirt. The goal of Plato is to bridge the semantic gap that exists between the low-level information that can be accessed by a VM service regarding a guest OS, and the high-level information that is accessible from within the guest OS context. A VM service is software that runs outside of the virtual machine and uses the VMM interface to access information or manipulate a VM. The interface provided by the VMM typically provides access to low-level hardware events such as network packets and disk I/O, rather than high-level information available within the guest OS context such as sockets and files. This makes it difficult for VM services to easily access the information they need. Plato tries to address this problem in order to make VM services more powerful and easy to develop.

Without using Plato, there are two alternate ways to achieve similar end results. These are: copying or re-implementing parts of the guest OS, as is done by the BackTracker system, or to use debugging tools like `gdb`. Copying or re-implementing guest OS code can quickly become very complicated and difficult to do, both in terms of the amount of code required, as well as getting that code to run in the VM service process. Using a debugging tool to directly call guest OS functions can incur very large overheads. As well, both of these approaches could lead to perturbing the sate of the virtual machine as a result of trying to inspect it.

Plato, which comes in the form of a C++ library, provides functionality beyond what is offered by a debugger at a substantially reduced overhead. Plato's capabilities can be categorized as follows:

1) Interposition: Plato lets the VM service register callback routines on VMM events such as I/O activity, CPU exceptions, or on functions names and source code line numbers. A callback routine can monitor or modify the state of the virtual machine. During the execution of a callback routine the virtual machine is suspended.
2) Access to variables: During a callback routine Plato allows the VM service to read or modify local variables in the context of the guest OS.
3) Calling guest kernel functions: Plato allows the calling of both exported and local functions in the guest OS.
4) Checkpoint and rollback: Plato implements a checkpoint and rollback feature in order to avoid perturbing the VM state as a result of VM service activity. This feature allows the VM to be rolled back to a previous state before the VM service caused any perturbations. There are some limitations to this feature in the area of networking, where rolling back may disrupt the state of network connections and cause them to be dropped.

5.6. Trail of Bytes

Trail of Bytes [26] is a monitoring approach that, as with Plato, takes into consideration and bypasses the “semantic gap" challenge that exists between a VM and running services. Trail of Bytes aims to reconstruct the events that occur on multiple abstraction levels found in virtual machines by implementing mechanisms for monitoring them. These events are defined as operations (typically read/write) made by entities to some locations. Obviously, these monitored events tend to grow tremendously. As a result, Trail of Bytes employs a query interface for the ease of manipulating the stored events.

To achieve event monitoring, first an abstraction of fundamental system calls is created in order to implement rapid and efficient event monitoring. Second, a logging framework that consists of an array of modules is created. The logging framework resides in the hypervisor, with none of its modules running in the VM. However, to the hypervisor, the internals of the VM are viewed as a black box, which imposes some difficulties on the logging framework. To solve this, the modules work cooperatively in an intelligent manner to be able to track the events that traverse the “semantic gap" between the virtual machine and the host operating system.

Mining the stored events is essential for the forensic analyst to understand the vulnerabilities and the exploits which occurred during an attack. The query interface of Trail of Bytes consists of four queries for mining the events:

- A two variable query that reports accesses to any memory block within β blocks that occurred during time range ω. The returned report shows the ID that made the access.
- A two variable query that reports accesses to any memory block done by ID during time range ω.
- A three variable query that reports all operations of certain type $access$ that were made to any block within β blocks that occurred during time range ω. This query can be either be given the range of blocks required β or certain ID.
- A three variable query that reports all operations caused by the same $causal$ that are made to any block within β blocks that occurred during time range ω. This query can be either be given the range of blocks required β or certain ID.

As Trail of Bytes fully resides in the hypervisor, it suffers from its vulnerability to hypervisor-detection attacks. Therefore, should the hypervisor get compromised, so will the event monitoring process. This is considered one of the shortcomings of Trail of Bytes. Also, Trail of Bytes is built over Xen VMM [27], and hence it inherits its vulnerabilities. In addition, Trail of Bytes can be hindered when faced by a resource-exhaustion attack. If an attacker executed high frequency operations that can overpass the ability of the logging framework to report all such events, he should succeed in escaping some events from being properly logged. Finally, Trail of Bytes has another logging limitation which is the lack of monitoring the outcomes of network operations.

6. Instrumented Kernel Approach

In this approach, the kernel is instrumented to create logs when certain events are observed. Since the logging component is integrated with the kernel of the operating system to be monitored, it has much better visibility into the system’s activities compared with the virtual machine approach. One of the downsides of this approach in comparison with the virtual machine approach is the inferior isolation between the logger and the operating system to be monitored, which increases the risk of log integrity being jeopardized should the kernel be compromised.

There are two ways to instrument the kernel to perform the logging required. Either the kernel code can be modified and have the logging capability built into it before compilation, or the logging functionality can be implemented as a separate loadable kernel module.

6.1. Forensix

Forensix [15] is another one of the prominent forensic event reconstruction systems proposed. The goal of Forensix is to allow reliable reconstruction of all system activity. Forensix facilitates this by capturing system calls and logging extensive information about these calls to a secure remote backend system for storage. The reasoning behind tracing only system calls in Forensix is based on the research team’s observation that successful attacks can only be caused by system calls issued by processes running on that system. Forensix was further extended by the same team to generate a timeline with the system state in order to simplify the job of analysis queries that use the generated log for event reconstruction.

The main component of Forensix is the logging facility which is installed on the machine to be monitored. Forensix uses the instrumented kernel approach towards logging. The instrumentation is done through the use of a loadable kernel module that traps system calls. Every time a system call is executed on the system, the Forensix logger logs the system call as well as other information relevant to it such as its time-stamp, parameters, return value, PID and owner of the calling process. This level of logging would store all network traffic.

In order to reduce the possibility of a kernel compromise, and thus losing log integrity, Forensix uses LIDS [28] to disable the following functionalities [29]:

1) user-level writes to kernel memory
2) user-level writes via the raw disk interface
3) writing to the kernel or the Forensix binary files
4) loading kernel modules

Disabling these functionalities could prevent certain applications that rely on them from running, such as X11 [16].

The system call data collected by the Forensix logging facility is periodically sent over a private network to a secure backend system that will store this data in an append-only fashion. While it is a good idea to store the logs on another secure system rather than the target system itself, the downside of this approach is that the network activity due to the log transmissions could lower the overall throughput of the target system. On the backend system, the incoming data is stored in log files temporarily. Every 24 hours, the log file contents are loaded into a MySQL database that has been set up in such a way so as to provide easy and powerful query access to the system call information for forensic purposes. In order to ensure that the data being generated on the target system does not overwhelm the backend system's ability to store logs, the backend system has been fitted with the ability to throttle the activity on the target system.

The backend system comes with numerous scripted queries and commands that allow a forensic analyst to easily retrieve specific information from the system call logs. Some examples of these scripted queries are:

- List all active processes during a given time interval
- List the children of a given process
- List the file descriptors written to by a given PID during a specified time interval, as well as the time of the write operation
- List all PIDs that wrote to a specific file descriptor in a given time interval

While Forensix does not provide automatic forensic analysis of an intrusion, the high-level queries it provides can still be very helpful to a forensic analyst trying to understand the activities that took place on the target system. It is possible in Forensix to run a series of iterative queries on the logged data and use the returned results to calculate a dependency graph similar to those generated by Back-Tracker [30]. As well, depending on the type of intrusion being investigated, the analyst may be able to construct a particular query in order to identify an exploit. As an example, [15] were able to come up with a query that would go through the database records and identify local privilege escalation attacks. For the most part however, it is up to the analyst to initially identify suspicious elements on the system and then investigate these further using queries in Forensix.

In order to get an idea of the performance overhead and space requirements associated with Forensix, the researchers ran two benchmarks. The first was a kernel build benchmark, and the second was a webstone web server throughput benchmark. With Forensix enabled, the kernel build time increased by 8%, while the web server throughput decreased by 36%. A large part of this decrease in the web server throughput is probably due to the transmission of logs from the target system to the backend system. The size of the compressed log files were also measured during the benchmarks. The compressed log file grew at a rate of 8.8GB/day for the kernel build benchmark, and 30GB/day for the webstone test.

Motivated by enhancing the performance of the reconstructed system through querying the database of stored events, the authors of Forensix have developed a tool [31] to analyze queries before reaching the SQL server. With this modification, Forensix is now considered a tool that follows the fifth principle in Section 2; as it greatly refines the way data is presented and manipulated by the forensic investigator.

6.2. Taser

The Taser [29] intrusion recovery system has been developed on top of the Forensix project in order to help recover file-system data that has been damaged due to an attack. With some input from an analyst, Taser can identify which file-system operations are due to the actions of an attacker and can restore the disk to a legitimate state by rolling back these malicious operations. There are several challenges in carrying out such an operation, which will be explored further in this section, such as legitimate operations interacting and becoming dependent on malicious modifications to the file-system. The Taser system consists of the following three components: auditor, analyzer, and resolver.

The auditor component is exactly the same as the Forensix system. Its function is to collect and log information regarding operations taking place on the system related to files, processes, and sockets. As was explained in Section 6.1, Forensix does this by capturing system calls and additional information associated with them.

The goal of the analyzer component is to identify the set of file-system objects tainted as a result of the intruder's activity so that they can be fed into the resolver for recovery. The analyzer also reports the time point at which the objects became tainted. In order to begin the analysis, the analyzer needs to be fed with a set of detection point(s) from either an IDS or by the analyst. This is somewhat similar to the way BackTracker works. If the provided detection points are not the source of the attack, then the analyzer needs to start a tracing phase and go back in the auditor logs until it finds the object(s) that are the source of the attack. It does this by following the dependencies between the detection point objects and other objects in the logs. The dependencies are formed when information flows from one object to another due to a system-call operation. After the analyzer identifies the potential set of objects that are the source of the attack, it requires manual feedback from the analyst to confirm this set. The analyst could reduce this set for example if he knows certain objects identified were not involved in the attack.

Once the analyzer has finished the tracing phase, it will begin the propagation phase. The propagation phase takes the set of objects identified during the tracing phase as the source of the attack and traces their dependencies forwards in the auditor logs in order to identify all the tainted file-system objects.

An analysis policy were all system-call operations that could lead to dependencies are followed could result in a large number of false positives. As such, Taser does allow the analyzer to use a number of other, more relaxed and optimistic policies, where certain potentially dependency causing operations are ignored. This can significantly reduce the number of false positives reported by the analyzer, but it may also lead to conflicts occurring during the resolver's operation.

The purpose of the resolver component is to revert the tainted file-system objects back to a legitimate state. The inputs to the resolver are: the set of tainted objects identified by the analyzer during the propagation phase, the logs stored by the auditor component, a file-system snapshot, as well as user preferences regarding certain recovery actions. There are two algorithms that the resolver can use here. The first is called the *Simple Redo Algorithm*. This algorithm starts with a file-system snapshot from a time before the objects identified by the analyzer became tainted. It then uses the data in the logs to replay all successful legitimate operations that caused modifications to the file-system from the time the snapshot was taken onwards. Any operations in the logs that were carried out on the tainted objects after the time the object became tainted, as determined by the analyzer, are considered malicious and not replayed. This is a straightforward approach, but it may end up taking a long time to replay all the operations on file-system objects which have been modified since the snapshot was taken. The other algorithm is called the *Selective Redo Algorithm*. This algorithm starts with the file-system state at the recovery time. At this point in time all the untainted objects are in their final state and no action is required regarding them. Only the tainted objects need to be recovered. The resolver obtains untainted versions of the tainted objects from the file-system snapshot and only performs the legitimate operations on these objects based on the logs. This approach is generally preferred if the number of illegitimate operations is small compared to the number of legitimate modifications since the snapshot.

Depending on how optimistic and relaxed a policy is used by the analyzer, the resolver may run into conflicts while trying to recover file-system objects. Some examples of these conflicts could be a legitimate file being created in a directory created by the attacker, legitimate renaming of a file created by the attacker, deleting a file renamed by the attacker, a legitimate update to the tainted contents of a file is made. The resolver may ignore some of these conflicts, recreate and recover some of the files with special extensions so that they can be manually inspected later, require manual fixing, or use application-specific conflict resolvers.

The Taser researchers tested the tool by presenting it with several different attack scenarios and then using Taser to recover from the attacks. Their results can be summarized as follows:

1) Taser analysis usually achieves high accuracy with few false positives and negatives.
2) Performing recovery one day or one week after the attack does not significantly affect the accuracy.
3) The selective redo algorithm is generally preferred because attacks usually have small footprints in comparison to the number of legitimate modifications to the file-system.

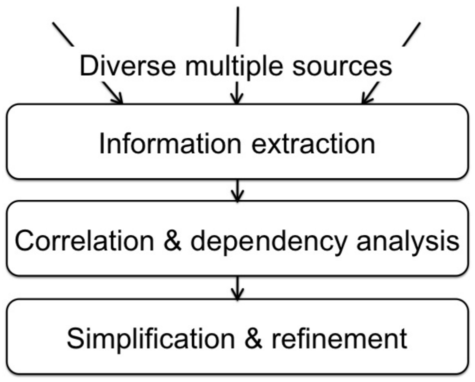
Diverse multiple sources
Information extraction
Correlation & dependency analysis
Simplification & refinement

Figure 1. Flowchart for the SLog tool [32]

4) None of the analyzer policies perform ideally under all circumstances.

6.3. The SLog tool

SLog [32] is a framework that is designed to process huge amounts of logs into an abstracted view for analyzing illegal system intrusion. It doesn't employ any logging component, rather, it is designed to be provided with different types of logs that should be available with system administrators. The sources of these logs can be intrusion detection systems, default system logging, network/application activity logging, etc. SLog is built upon three key assumptions:

1) An ease in distinguishing legitimate events from illegitimate ones out of huge log files should speedup the recovery process.
2) The tool should be able to recover information about an attack that happened prior to its installation.
3) The tool should be able to face and deal with huge amounts of logs.

Figure 1 shows the sequential functionality for the operation of SLog. All three steps are integrated together to constitute a declarative tool that provides the forensic analyst a simplified, yet comprehensive, interface to the enormous amounts of logs.

As seen in Figure 1, the first step, namely information extraction, is built over the extraction techniques made in [33], where the Datalog –a prolog based nonprocedural query language that simplifies writing queries– language is modified to employ "embedded procedural predicates" [32, p. 190]. With procedural predicates, arguments are passed from Datalog programs and information is returned in the same way. One advantage of this mechanism is the smooth extraction of complex information from sources that lack certain structure. This mechanism has the ability to scale as the dataset grows, it has an understandable extraction specification that is easy to construct and its results are reliable and comprehensive as it extends the accurately defined semantics of Datalog.

The next step after information extraction is correlation and dependency analysis, which is to decide weather an event, defined by a set of information, is relevant to the intrusion or not. To decide this, the relation of data dependence between events must be analyzed. Four factors are primarily used to carryout the analysis:

- Value equivalence: *identical values of events make them correlated.*
- Type information: *if two events are of the same type, they are considered correlated.*
- Kill events: *if the triggering of event A causes the killing of event B, then A and B are said to be correlated.*
- Definition & use information: *if event A used a data location that is defined by B, then A and B are said to be correlated.*

Finally, before viewing the extracted events, refinement is done to abstract the low-level data in the logs and present a refined summary of information with the events that occurred. The tool allows the forensic investigator to define his preferred abstraction methodology.

7. Assessing the Proposed Solutions

7.1. Comparison

Amongst the proposed solutions discussed in Sections 5 and 6, the two competing camps are the BackTracker based and the Forensix based systems. These two sets of solutions are compared briefly in the remainder of this section in addition to a quick comparison with other tools.

7.1.1. Operation-based Comparison. The BackTracker and Forensix systems share many similarities. Both systems are primarily concerned with capturing and recording system calls. Both systems do not consider user space, application objects or events in their tracking. While BackTracker is only interested in certain system calls, Forensix records extensive information about all system calls, including their parameters and return values which are not recorded by BackTracker. This does allow much more detailed analysis of system activity during the forensic investigation, but comes at a cost of larger log file sizes. However, later evolutions of BackTracker such as Improved BackTracker and Bi-directional Distributed BackTracker do record more information than their predecessor. Improved BackTracker records the parameters and return values of system calls just like Forensix so it can determine which file offsets were involved in read and write operations. Bi-directional Distributed BackTracker derives information from the header of each TCP packet and records it for later analysis. Trail of Bytes has a slightly similar approach to Backtracker where it operates on virtualized environments, though it differs in its theory of operation. It focuses on recording events of data access while residing in the hypervisor of the virtual machine. SLog has a completely different approach. It doesn't perform any logging by itself, rather, it depends on either the default system logging or any other logging scheme that should be externally provided. Its main objective is to process huge sizes of logs from diverse sources by: extracting data, correlating it and simplifying it without loosing important information.

7.1.2. Goal-based Comparison. The main difference between these systems is their goals. The purpose of BackTracker is to analyze causality information and generate graphs which identify the source of an attack on a system. Further work on the BackTracker project -which resulted in the Bi-directional Distributed BackTracker- aims at extending the tracking of intrusions to multiple hosts so that the initial source of an attack in a network, as well as other hosts compromised can be identified. Trail of Bytes aims to present the analyst with answers to three main questions regarding a compromised system: what, when and how. Forensix is somewhat more open-ended, and provides a set of high-level SQL queries for the analyst to use in order to investigate the system events in any way desired. The Taser system which is an evolution of Forensix steers this project towards identifying and recovering damage done to the file-system rather than pinpointing the attack source. SLog aims to provide an abstracted view of all the events that happened on the system as well as identify the ones related to an intrusion.

While SQL queries and logged data in Forensix can be used in a similar manner to BackTracker in order to identify the source of an attack on a system, Forensix does not come with analysis tools that automatically perform this operation. On the other hand, Forensix stores logged data in a MySQL database and provides high-level SQL queries for analyzing it which is much more user friendly than BackTracker.

In terms of logging, both BackTracker and Forensix provide reasonable solutions. We believe that both logging components in the instrumented kernel approach (such as that of Forensix) and the virtual machine approach (such as that of BackTracker) are vulnerable to attacks. Forensix transmits the logs to a backend system for immutable storage [15]. As such, should the system being monitored get compromised, at least the integrity of the logs from the time before the attack could be maintained. Although this technique isolates the logs, the transmission of the logs to the backend system by Forensix tends to reduce the throughput by a large amount on the system being monitored [15]. With the virtual machine approach, in the event that the attacker can exploit a vulnerability in the VMM from inside of a VM and affect the native host OS, the entire set of logs stored on the host OS could be compromised.

7.2. A summary of the Empirical Study by Jeyaraman et al.

Although there have been quite a few event reconstruction systems proposed over the past several years, there have not been many independent studies performed on their effectiveness. One possible reason for this might be that it is difficult to come up with a good suite of benchmarks for this task. Since it is known that coming up with fair, accurate and unbiased benchmarks for intrusion detection systems is a difficult challenge [34], and since event reconstruction systems do have somethings in common with IDSes, this could be a plausible explanation for the lack of studies in this area. Amongst the things IDSes and event reconstruction systems have in common is that they are both complex systems and their operating environment is a significant factor in their effectiveness [35].

In this section, we summarize the work done by [35] for general event reconstruction system evaluation. The researchers broke down the mechanisms behind causal relationships into two types: *operating system* and *program dependencies (PD)*. Operating system dependencies are easy to spot because they are due to system calls whose semantics are well defined [35]. On the other hand, when dealing with the program dependence mechanism, if a dependency is formed entirely in the context of a single process and its address space, it is somewhat more difficult to analyze. Therefore, the evaluation criteria was chosen as the ability to infer causal relationships based on PDs. In particular, [35] focused on evaluating Forensix and Backtracker based on their rate of false positives in figuring out dependencies.

[35] used a suite of real-world applications including: `gnuPG`, `wget`, `find`, `locate`, `ls`, `cp`, `wc`, `tar`, `gzip`, `grep`. The reason for choosing such applications for testing instead of coming up with intrusion scenarios, such as those used by the researches of the proposed systems themselves, was to avoid any inherent bias and inaccuracies that the researchers themselves may unintentionally introduce. Also, as previously mentioned, coming up with an all encompassing set of fair and accurate attack scenarios for a benchmark in this field is a difficult task. One of the downsides of this choice is that these are not applications one would typically encounter while investigating security incidents.

[35] note that all the event reconstruction systems considered are quite good at recognizing proper dependencies resulting from operating system mechanisms. As such, they focused their testing on the rate of false positives registered by reconstruction systems when dealing with PD relationships. BackTracker, Improved BackTracker, and Forensix are conservative in their approaches towards tracking causality relationships, therefore they only result in false positives and no false negatives.

Their results revealed that on average, the rate of false positives for the technique used by BackTracker and Forensix is very high when dealing with PD relationships. The false positive rate does vary greatly between different applications. The highest rate was 95.6% for `gpg`, while the lowest was 31.78% for `wget`. Improved BackTracker also has high false positive rates and in general does not provide a significant improvement over BackTracker. While in most tests its rate of false positives was slightly lower than that of BackTracker, in one test it registered almost double the number of false positives. The researchers do acknowledge that testing Improved BackTracker's data flow analysis technique is dependent on a variety of parameters and that testing using a different implementation with different parameters could return different results.

8. Conclusion

Forensic event reconstruction systems are aimed at helping administrators and forensic analysts investigate a security breach. A forensic event reconstruction system needs to be installed on the target system before it has been exposed to a hostile environment. These systems carry out their own extensive logging of events on the target machine and it is these logs which are used at a later stage in order to understand the nature and consequences of the exploited security breach.

We summarized the work done in the field of forensic event reconstruction systems mainly by surveying the peer-reviewed literature and other sources. This paper was not intended to be a product evaluation study nor a comprehensive survey that covers every publication in the area. Our contribution is rather a survey covering basic principles, a broad taxonomy of the techniques proposed in literature and summary of some of their reported assessment.

We covered several systems proposed by the research community over the past few years. All of the systems discussed either use a virtual machine introspection approach, or an instrumented kernel approach towards monitoring the sequence of events on the system and logging them for later analysis. ReVirt, along with BackTracker and related projects all use the virtual machine introspection approach towards logging, while Forensix, Taser and SLog utilize the instrumented kernel technique. The ReVirt system provides the analyst with the ability to replay the execution on the system at an instruction-by-instruction level from any time period during which logging was taking place. ReVirt lacks any automatic analysis tools. The BackTracker system is capable of taking as input a suspicious process ID or file presented to it by the analyst and based on the logs it had collected, identify all the possible chains of events that could have affected the given object and display these in the form of a dependency graph. In doing so it would identify the original source of the attack on the system. The other proposed solutions which are based off of BackTracker try to improve its accuracy, as well as extending its tracking capabilities across multiple systems in an administrative domain. Trail of Bytes is another solution underlying the virtual machine approach. It employs some heuristics to correlate the collected events and trace them to figure out the process with the backdoor used by an attacker. Forensix has somewhat similar, yet more in-depth, logging capabilities compared to BackTracker, but instead of focusing on presenting the analyst with possible chains of events involved in an attack, it provides a high-level SQL query interface to the logged system activities. The Taser system builds on the work done in Forensix and focuses on the ability to identify file-system objects tainted as a result of the attack, as well as the ability to revert these objects back to a clean state. Finally, SLog is a framework that should allow external forensic tools to work on top of it. The authors of SLog have designed the SLog programming language with some rules in order to help the development of forensic tools with the SLog framework.

Acknowledgments

Ashraf Matrawy acknowledges funding from NSREC through the Discovery Grant Program.

References

[1] S. Peisert, M. Bishop, S. Karin, and K. Marzullo, "Principles-driven Forensic Analysis," in *New Security Paradigms Workshop (NSPW)*. New York, NY, USA: ACM Press, 2005, pp. 85–93.

[2] ——, "Toward Models for Forensic Analysis," in *Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*. IEEE, 2007, pp. 3–15.

[3] T. Vidas, "The Acquisition and Analysis of Random Access Memory," *Journal of Digital Forensic Practice*, vol. 1, no. 4, pp. 315–323, December 2006.

[4] "EnCase Forensic," http://www.guidancesoftware.com/, September 2007. [Online]. Available: http://www.guidancesoftware.com/

[5] "ASR Data's SMART," http://www.asrdata.com/, September 2007. [Online]. Available: http://www.asrdata.com/

[6] T. Sleuth Kit, "The Sleuth Kit (TSK) & Autopsy: Open Source Digital Investigation Tools," http://www.sleuthkit.org/, September 2007. [Online]. Available: http://www.sleuthkit.org/

[7] "CERT's Live View," http://liveview.sourceforge.net/, September 2007. [Online]. Available: http://liveview.sourceforge.net/

[8] "SANS SIFT," https://computer-forensics.sans.org/, August 2011. [Online]. Available: https://computer-forensics.sans.org/community/downloads

[9] K. Guðjónsson, "Mastering the Super Timeline With $log2timeline$," https://computer-forensics.sans.org/, June 2010. [Online]. Available: http://computer-forensics.sans.org/community/papers/gcfa/mastering-super-timeline-log2timeline_5028

[10] Timestomp, "TimeStomp - Metasploit Unleashed," http://www.offensive-security.com/metasploit-unleashed/Timestomp, September 2007. [Online]. Available: http://www.offensive-security.com/metasploit-unleashed/Timestomp

[11] "Defiler's Toolkit," http://www.phrack.org/issues.html?issue=59&id=6&mode=txt, September 2007.

[12] J. Rutkowska, "Beyond The CPU: Defeating Hardware Based RAM Acquisition," in *Proceedings of BlackHat DC 2007*, ser. Black Hat DC, February 2007.

[13] "CA Network Forensics," http://www.ca.com/, September 2007. [Online]. Available: http://www.ca.com/

[14] S. T. King and P. M. Chen, "Backtracking Intrusions," in *ACM symposium on Operating Systems Principles*. ACM Press, 2003, pp. 223–236.

[15] A. Goel, W. chang Feng, D. Maier, W. chi Feng, and J. Walpole, "Forensix: A Robust, High-Performance Reconstruction System," *Workshop on Security in Distributed Computing Systems (ICDCSW)*, pp. 155–162, 2005.

[16] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Network and Distributed Systems Security Symposium (NDSS)*, 2003.

[17] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay," *SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 211–224, 2002.

[18] J. Gionta, A. M. Azab, W. Enck, P. Ning, and X. Zhang, "Dacsa: A decoupled architecture for cloud security analysis," in *Workshop on Cyber Security Experimentation and Test (CSET)*. USENIX, 2014.

[19] D. Farmer and W. Venema, *Forensic Discovery*. Addison-Wesley Professional, 2004.

[20] S. Ma, X. Zhang, and D. Xu, "Protracer: towards practical provenance tracing by alternating between logging and tainting," in *Annual Network and Distributed System Security Symposium (NDSS)*, 2016.

[21] S. T. King, G. W. Dunlap, and P. M. Chen, "Plato: A Platform for Virtual Machine Services," University of Michigan, Tech. Rep. CSE-TR-498-04, 2004.

[22] P. Chen and B. Noble, "When Virtual is Better Than Real," in *Workshop on Hot Topics in Operating Systems*, 2001, pp. 133 – 138.

[23] G. W. Dunlap, D. G. Lucchetti, M. A. Fetterman, and P. M. Chen, "Execution Replay of Multiprocessor Virtual Machines," in *SIGPLAN/SIGOPS international conference on Virtual Execution Environments (VEE)*. ACM, 2008, pp. 121–130.

[24] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching Intrusion Alerts Through Multi-Host Causality," in *Network and Distributed System Security Symposium (NDSS)*, 2005.

[25] S. Sitaraman and S. Venkatesan, "Forensic Analysis of File System Intrusions Using Improved Backtracking," in *International Workshop on Information Assurance (IWIA)*. IEEE, 2005, pp. 154–163.

[26] S. Krishnan, K. Z. Snow, and F. Monrose, "Trail of Bytes: Efficient Support for Forensic Analysis," in *Conference on Computer and Communications Security (CCS)*. ACM, 2010, pp. 50–60.

[27] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Symposium on Operating Systems Principles (SOSP)*. ACM, 2003, pp. 164–177.

[28] H. Xie, "Linux Intrusion Detection System (LIDS) Project," http://www.lids.org. [Online]. Available: http://www.lids.org

[29] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara, "The Taser Intrusion Recovery System," in *Symposium on Operating Systems Principles (SOSP)*. ACM, 2005, pp. 163–176.

[30] A. Goel, W. chang Feng, W. chi Feng, and D. Maier, "Automatic High-performance Reconstruction and Recovery," *Computer Networks*, vol. 51, no. 5, pp. 1361–1377, 2007.

[31] A. Goel, K. Farhadi, K. Po, and W.-c. Feng, "Reconstructing System State for Intrusion Analysis," *SIGOPS Operating Systems Review*, vol. 42, pp. 21–28, 2008.

[32] M. Fredrikson, M. Christodorescu, J. Giffin, and S. Jhas, "A Declarative Framework for Intrusion Analysis," in *Cyber Situational Awareness*, ser. Advances in Information Security, S. Jajodia, P. Liu, V. Swarup, and C. Wang, Eds. Springer US, 2010, vol. 46, pp. 179–200.

[33] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan, "Declarative Information Extraction Using Datalog with Embedded Extraction Predicates," in *International conference on Very Large Data Bases (VLDB)*. VLDB Endowment, 2007, pp. 1033–1044.

[34] M. J. Ranum, "Experiences Benchmarking Intrusion Detection Systems," http://www.snort.org/docs/Benchmarking-IDS-NFR.pdg, December 2001. [Online]. Available: http://www.snort.org/docs/Benchmarking-IDS-NFR.pdg

[35] S. Jeyaraman and M. J. Atallah, "An Empirical Study of Automatic Event Reconstruction Systems," *Digital Investigation*, vol. 3, no. 1, September 2006.