ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester thesis

A Security Forensics framework

Ramya Jayaram Masti

Advisors:
Dr.Vincent Lenders
Dr.Stefan Engel
Dr.Mario Strasser
Prof.Dr.Bernhard Plattner
Prof.Dr.Srdjan Capkun

September 2009 - February 2010

Communication Systems group
Information Technology and Electrical Engineering Department
ETH Zurich

Abstract

With increasing use of IT infrastructure for crime, digital forensics has become important for law enforcement. Integration of digital evidence from several sources is important for improving the quality of data available for forensic analysis. As a first step in this direction, we evaluate the possibility of integrating host forensics and network forensics on a periodic basis rather than during post-mortem forensic investigation. We develop a support framework for the collection and storage of forensically relevant information from hosts and networks. We show the feasibility of collecting certain important host relevant data like process information (including path of the executable responsible for the process, process owner, handles, etc). The 'tagged' bloom filter structure we develop for storing network data, scales linearly with the number of hosts and the link utilization. The volume of storage required to collect data (network and host data) from a 1000 hosts with a 1Gbps link (10 percent utilization) in 24 hours is about 80GB which we consider reasonable. Finally, the use of the framework in the investigation of a data leakage scenario provides an illustration of how host information could improve time required for forensic analysis.

Acknowledgements

I would like to thank all the people who have helped me through the course of this project. At the outset, I would like to thank Dr.Vincent Lenders for giving me an opportunity to work with him on this project. He has been an excellent advisor and has been very patient with me. I have learned a lot from him, and not just academically. I am grateful to Dr.Stephan Engel, Armasuisse, Bern, for the insightful weekly discussions which he always made time for, despite his busy schedule. I have also been very fortunate to work with Dr.Mario Strasser whose guidance and support has been invaluable for the completion of this project. I would like to thank Prof.Dr.Bernhard Plattner and Prof.Dr.Srdjan Capkun for their valuable advice and feedback through the course of this project. I would also like to thank Dominik Schatzmann for his help regarding details of network flows. Finally, I would like to thank my family and friends for their unflinching support and cooperation.

Contents

Chapter 1

Introduction

1.1 Motivation

The increasing use of IT infrastructure for crime makes digital forensics an indispensable part of law enforcement. The science of digital forensics has been defined as 'the process of identifying, preserving, analyzing, and presenting digital evidence in a manner that is legally accepted' [1]. There are several sources of forensic evidence. Integration of information from these sources for improving the quality of forensic evidence poses an interesting challenge. The problems of collection, storage and analysis of the information with respect to feasibility and efficiency are important for analysis of security incidents after their occurrence.

1.2 Problem statement

The main goal of the forensic framework is to develop an approach to digital forensics that combines host forensics and network forensics to enable better reconstruction of security events. The framework must be able to provide details regarding the source of an attack, the location of the attack, its timing, its effect (details regarding compromised assets) and the reconstruction of the attack itself. The individual or the organization behind the attack and their motivation is not always deducible. For example, if it is known that some sensitive information was transferred over the network, the framework must be able to furnish information regarding what information was transferred, by whom, when and how it was transferred.

1.3 Related work

Recent work in digital forensics focuses on optimizing the collection, processing or storage of digital evidence from a single source as discussed in [2], [3], [4], etc. There are few pointers to the integration of information from

the various possible sources of evidence. Although it has been recognized for long that networks and hosts yield valuable data that can aid forensic investigations, there have been relatively few attempts to incorporate data from both sources into a forensic framework [5]. Most research focuses either on host forensics or towards network forensics but not in their combination.

Modern approaches to host forensics emphasize on 'live forensics' [6], [7] and development of tools for live forensics [8]. 'Live forensics' refers to collection of volatile data on the host under scrutiny before it is shut down to collect its disk contents. This is useful because volatile memory may contain information about recent activity on that host. However, there are no references to continuous volatile memory examination, processing and storage as we investigate in this thesis.

Contemporary research on network data retention for forensic purposes includes the use of hierarchical bloom filters [9], use of arithmetic coding for data compression [10] and storage of partial flow information [11]. However, notable disadvantages exist with each of these techniques. Hierarchical bloom filters do not allow the extraction of network flow data, arithmetic coding only compresses header size and storage of partial flow information loses data pertaining to longer flows. Hence, it was important to design a data structure for network data retention that overcomes these deficiencies but is still scalable with the number of hosts.

1.4 Contribution

Forensic investigation approaches differ according the exact nature of the incident under investigation like the time since the occurrence of the security incident, amount of information known about the incident, goals of the investigation, etc. A particular application of a forensic investigation deals with finding the source and/or the evidence pertaining to a known security incident. Given that a forensic examiner knows the exact nature of the breach, one may build a supporting framework to ensure that all the data required to construct the evidence has been collected. The exact data to be collected depends upon the nature of the security breach. This work focusses on the development of such a support framework through support for continuous data collection from hosts and the network.

This work examines and establishes the feasibility of periodic collection, processing and storage of forensically relevant data from hosts in contrast to 'live' examination of compromised hosts. Further, it also obviates the need for entire network traces and instead stores network data in the form on 'tagged' bloom filters. This approach has the advantage of allowing forensic analysis on security incidents long after they have occurred by reducing the

volume of data required to reconstruct the event.

Chapter 2

Background to Forensic Evidence collection

This chapter describes the background to forensic evidence collection which includes the characteristics of the data collected, the layers of abstraction of IT infrastructure and its relevance to forensic evidence collection.

2.1 Characteristics of forensic evidence

Forensic evidence is characterized by the following desirable properties[12]:

a. **Integrity:** It must be verifiable that the evidence has not changed since collection time.

b. **Reproducibility:** It must be possible to elicit the same evidence given a system in the same state, i.e., the process of evidence collection must be reproducible.

c. **Authenticity:** It must be possible to verify the source of evidence.

d. **Non-interference:** The examination process must not alter the examined systems or at least the exact effects caused must be clear.

e. **Minimization:** Only relevant data should be collected.

f. **Availability:** The process of evidence collection must not be hindered.

It is not always possible to ensure that all the properties hold. For example, it is not possible to achieve reproducibility in gigabit networks where large volumes of data cannot be stored for very long periods of time.

2.2 Layers of abstraction

In [13], the author discusses an approach to defining independent layers of abstraction that are amenable to forensic analysis. On applying a similar paradigm to a host, one can view IT infrastructure in terms of a number of layers as shown in Figure 2.1.

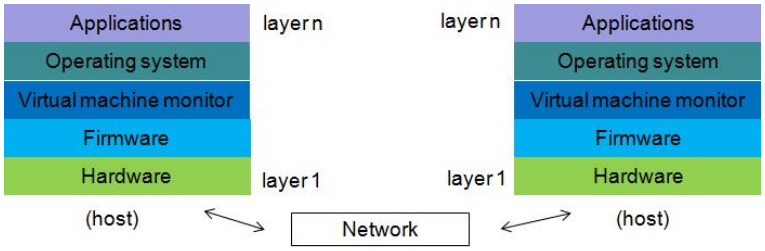

Figure 2.1: Layers of a host

An attacker could compromise any of the properties of the data collected as forensic evidence from any of these layers. Furthermore, if an adversary has complete control over a certain layer on a host, then the adversary also has partial or complete control over all the layers above the compromised layers on the host. The network layer is used for communication between hosts. Any layer on the host ideally uses the layers below it to reach the network layer. The effects of the compromise of a layer on a host is generally visible on the network layer.

2.3 Assumptions

We state the assumptions underlying the design of our framework. These assumptions must be realistic to ensure practicality of the framework. On a host (hardware, firmware, OS, application), it is reasonable to say that when an adversary controls a lower layer, he controls all the upper layers. The assumptions below, mitigate to an extent, the control an adversary can exercise over any layer without being detected.

In the following subsections, the assumptions at each layer are enumerated.

2.3.1 The hardware and firmware layer

a. Availability of trusted copies of important parts of the disk (boot sector, OS) for examination for changes at the byte level (not via an API)

b. Sufficient physical protection to prevent direct tampering of hardware (insertion of key loggers, modification of firmware by direct physical

access or insertion of other malicious hardware is very difficult).

2.3.2 The VMM and the operating system layer

a. Availability of trusted copies of the operating system and the VM monitor for static integrity checks.

2.3.3 The application layer

a. Availability of trusted copies of common applications (which are perhaps part of a known baseline of applications) for static file integrity checks.

b. No loss of evidence due to deletion by any protective or other mechanisms.

2.3.4 The network layer

a. Deployment of suitable mechanisms to prevent MAC spoofing and IP spoofing. For example, one could cross verify with a DHCP server which authenticates a user before assigning his host an IP address or one could deploy S-ARP[14] internally.

b. Deployment of suitable mechanisms to prevent exploitation of DNS vulnerabilities. For example, deployment of DNSSec internally on the network to mitigate the risk of exploitation of DNS related vulnerabilities.

The assumptions above are ideal for forensic data collection. The violations of these assumptions leads to less reliable forensic data depending upon the extent of compromise of the layers. For example, one could have mechanisms to detect IP and MAC spoofing instead of prevention mechanisms. If there is an appropriate deployment of sensors on the network, one might still be able to distinguish spoofed traffic from legitimate traffic.

Chapter 3

Framework Requirements

The requirements of the forensic framework can be categorized into generic and security requirements. They are discussed in detail below.

3.1 Generic requirements

1. **Scalable data collection:** The volume of data required to be stored by the framework must be 'reasonable' in economic terms (cost of disk space, etc.) and must exhibit not worse than linear increase with increase in number of hosts or network connections.

2. **Scalable Performance:** The framework must exhibit 'negligible' (to be defined) degradation in performance with increase in the number of hosts and/or network connections.

3. **Extensibility:** Incidence response uses attack signatures. It must be easy to extend the framework to detect new signatures.

4. **Platform independence:** The framework must be able to hide the heterogeneity of the hosts and network topologies and collect data in a uniform way.

3.2 Security requirements

The security requirements of the framework encompass the security requirements of the data collected and the security requirements of parts of the framework itself. The latter includes the authenticity, integrity and availability of parts of the framework for collection and analysis of forensic evidence. The integrity and authenticity of the data collected are somewhat equivalent because in either case the net effect is wrong data being collected.

In this work, the requirements of scalable data collection and extensibility is addressed. Platform independence can be achieved by the incorporation

of appropriate tools during the implementation stage to deal with different platforms. The security requirements with respect to the framework components can be met with the help of common techniques like integrity 'self-tests', 'heart-beat' generation for availability tests, authentication mechanisms, etc. Security violations of the data collected can be detected to an extent using cryptographic transformations like digital signatures, etc.

Under the assumptions stated in the previous section, Table 3.1 shows the effects of compromise of any layer in a host. It is to be noted that once a layer is compromised, evidence collected at that layer and all the above layers is rendered untrustworthy. Here, the integrity of evidence prior collection and post collection or in transit is dealt with separately. Authenticity of evidence refers to the authenticity of parts of the framework. This coupled with host authenticity is required to prevent framework components from impersonating each other. Finally, availability of evidence refers to the availability of the framework's evidence collection and analysis functionality. **The following discussion assumes that components of the framework that collect evidence are part of the application layer or the operating system layer.**

With compromise in the lower layers (hardware, firmware and VMM layers), it is difficult to assure the integrity of the data -both prior and post collection and the integrity of framework components because integrity violations at these layers may leave no traces in the upper layers. Similarly, authenticity of the framework components cannot be assured with compromise in these layers. Violation of the availability condition can be detected by appropriate implementation techniques (sufficient redundancy, 'heart beat' generation, etc.)

For compromise at the higher layers (OS and application layers), dynamic and static signature checks can be used to assure the integrity of the data as well as the framework. For example, checking for hooks which modify the behavior of the framework component or the data it collects. Authenticity of the framework components can be assured similarly. Violation of the availability condition can be detected as in the case of the compromise of the lower layers.

Layer under attacker control	Integrity of evidence		Integrity of framework components	Authenticity of framework components	Availability of evidence
	Prior collection	Post collection			
Hardware	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Unavailability detectable
Firmware	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Unavailability detectable
VMM	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Violation may or may not be detectable	Unavailability detectable
OS	Violation detectable	Violation detectable	Violation detectable	Violation detectable	Unavailability detectable
Application	Violation detectable	Violation detectable	Violation detectable	Violation detectable	Unavailability detectable

Table 3.1: Effects of compromise of different layers in a host

Chapter 4

Framework design

The design of the framework includes designing a generic architecture for the framework, definitions of the host and network data to collect, appropriate tools to collect the required data and design of data structures for storage of collected data. In this chapter, these aspects of the design and its evaluation are discussed.

4.1 Framework Architecture

In this section, a generic architecture of the online forensics framework including its components and communication between them is discussed.

4.1.1 Components of the framework

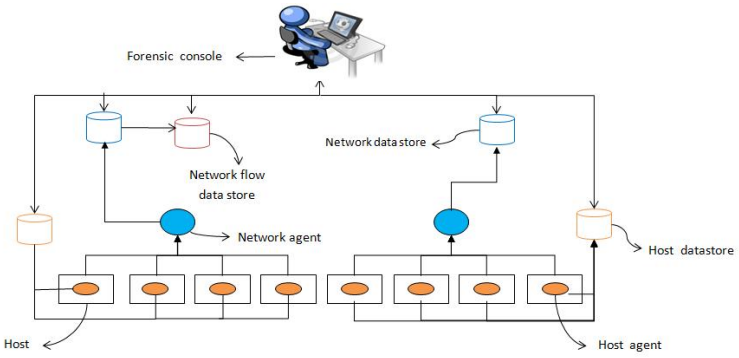

Figure 4.1: Components of the framework

Figure 4.1 shows the components of the framework.

a. **Host data store:** The host data store is used to store host information obtained from the host agents.

b. **Network data store:** The network data store is used to store network information obtained after processing from the network agents.

c. **Network flow data store:** This is an optional data store that can be used to hold flow information derived from the network data store. The processing to derive the network flow information can be done at the network data store. This will avoid transfer of large amounts of data over network.

d. **Network Agent:** The network agent is deployed on every LAN. It sniffs all the traffic on the LAN. Periodically, it processes the collected traffic, extracts the required information and stores it at the network data store in the tagged bloom filter format.

e. **Host Agent:** The host agent is deployed on every host. It periodically collects data about the host like the process information, port information,etc. and transfers it to the host data store.

f. **Forensic console:** This is a control and data mining interface for the entire framework. It connects to the various data stores for data retrieval during a forensic examination and can be used to control the various host agents and network agents.

4.1.2 Communication between framework components

It is important to strategically position the various data stores as data transfers during an analysis can lead to considerable network traffic. This also makes the timing of the data transfer to the various data stores important.

One can assume without loss of generality that all transfers to the network data stores occur during off peak hours. However, data collected on hosts will have to be transferred more frequently because of the higher risk of compromise by an attacker. One may assume that transfers to host data stores occurs every few hours depending upon the amount of data generated and the frequency with which data is collected. Finally, one may assume that the forensic server can issue queries to the data stores instead of obtaining all the data and processing it itself. As a result, the network data stores can be closer to the forensic server than to the respective network agents. The host data agents are positioned on the LAN for proximity to the hosts.

The effects of having fewer host agents or fewer network agents in the framework is varied. If a host agent is not deployed on a certain machine, then process data on that machine is missing. However, network agents can be strategically positioned to collect the same information in different ways. For internal traffic, having an agent in the LAN of one of the communicating parties is sufficient. Network agents can also be positioned at the bridges (junction of LANs) to collect data from more than one LAN. This will mean increased resources and load for a single network agent instead of

less resources and less load for many network agents. The trade off in the cost of resources is situation dependent.

4.2 Forensic data collection

The network and the individual hosts are sources of forensically valuable data. In the following subsections, determination of the forensically useful data set and their storage data structures (for both, the network and the hosts) are discussed.

4.2.1 Network data collection and storage

Background

The emergence of gigabit networks poses a challenge to the preservation of network data over long periods of time for forensic analysis. Although,the cost of storage has decreased considerably during the same period, it is important to optimize the data stored in three dimensions - volume, content and processing. The project focuses on optimizing the volume of data vs. the content and not on the optimizing the processing. The following subsections include a survey of tools that can be used to collect this data, a description of the data structure used to store the network data and its evaluation.

Tool survey

There exist several open source tools (NetworkMiner, WireShark, TCPFlow, etc.) and commercial tools (NetDetector, NetIntercept, etc.) for network data collection and analysis. The commercial tools offer faster collection and analysis capabilities and some of them are customized for gigabit networks. The freeware tools have a comparable number of features like automatic flow extraction, deep packet inspection, etc. Further some of these tools are GUI-based, some have just a command line interface while a few have both.

For the purpose of this project, Wireshark is used to capture network traffic. The processing of this traffic is done offline. This ensures that the data collection is not affected by the processing of collected data.

Network data storage

The network agent can be used to store network data in various levels of detail. This may vary from entire network dumps to just netflow data. There is an inherent trade off in the volume of data that has to be archived

vs. the amount of information that can be retrieved from that data. For instance, netflow data is small in volume but gives no information about packet payloads. On the other hand, network dumps reveal all information about network traffic but are very large in volume. We propose a data structure that yields partial information about packet payloads and flow. We call this a 'tagged' bloom filter. Its purpose is to optimize the volume of data that accumulates for extraction of flow information, given known payloads.

'Tagged' bloom filter

The 'tagged' bloom filter we propose allows determination of which hosts have transferred what data from a set of known data. The data structure is best described for a single local area network (LAN) with replication for each LAN over the entire enterprise. Since most traffic is TCP/UDP based, this data structure primarily supports analysis of TCP/UDP traffic.

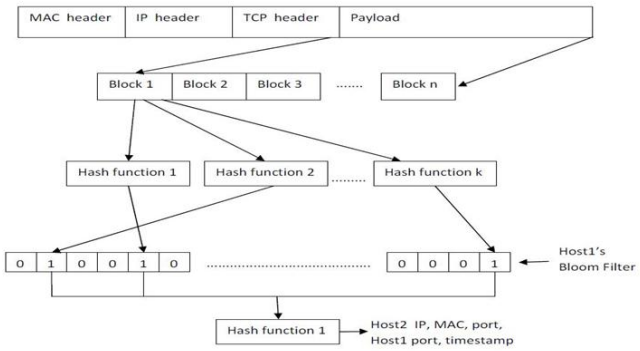

Figure 4.2: Inserting a link layer frame from host1 to host2 into a tagged Bloom filter

Consider a LAN of 'N' hosts. For each host, a Block based Bloom Filter is created. Let the number of blocks for each link layer packet (e.g. 1500 bytes) be 'b'. Figure 4.2 shows the processing of a link layer frame from host1 to host2. Without loss of generality, we may assume that host1 is the internal host talking to a host2 outside the enterprise. Assuming that IPs remain constant for a given time interval (e.g. 24 hours),

a. Extract the details pertaining to 'host1' (IP, MAC, (TCP/UDP) port) and the details pertaining to 'host2' (IP, MAC, (TCP/UDP) port) and create a timestamp.

b. For a packet which is IP, TCP/UDP based, the payload refers to the payload of the following application layer protocol or the payload of TCP/UDP layer itself . Split the 'payload' into 'b' blocks of equal

sizes. However, the last block is inserted without any padding even if its size is smaller than the block size.

c. Create a bloom filter (of size 'm' to hold 'n packets' using 'k' hash functions) for host1 if it doesn't exist already. Multiple hash functions are used to reduce the number of false positives especially when data blocks inserted differ only by a few bytes[15]. Insert each block into host1's bloom filter by interpreting the output of each hash function (k functions in total) as a number, applying the modulus operator (with respect to the size of the bloom filter) and then setting the corresponding bits (b_1-b_k) of the bloom filter to 1. Create a 'tag' with the details of host2 and host1's (TCP/UDP) port. For each inserted block, concatenate the values of (b_1-b_k) positions and hash them again using the bloom filter's first hash function modulo the size of the bloom filter. Add this value(referred to as a block-id henceforth) to the tag for each of the inserted blocks.

If host1 is the external node and host2 is an internal node, carry out the same operations on host2's bloom filter instead of host1. If both are internal nodes, then carry out the operations on the bloom filter of the packet's source.

Finally, re-create the entire data structure after the required time interval expires (24 hours in our case) because renewal of IP addresses requires new tagged bloom filters to be created for hosts. One can easily extract connection information using the tag data and dropping the bloom filters while transferring to the network flow data store.

The tagged bloom filter structure allows two types of queries. First, it allows queries to find the connection information corresponding to known payload transfers. Second, given a set of potential payloads, it can reveal if a certain connection transferrred any of those payloads. The granularity of these searches depends upon the block size. All searches proceed by first looking checking the bloom filters for containment of the payload under interest and then finding the tag (s) with the corresponding hash values to extract connection information.

Evaluation of the data structure

The data structure is evaluated across several parameters. In the following discussion, it is assumed that the network agent is monitoring a 1 Gbps Internet link. Also, the size of the bloom filter is chosen such that it ensures that the False Positive Rate is about 0.02 (by m=8*n) [15]. The variation of the total data size per day with respect to variations in number of hosts per LAN, differing behavior of hosts on a LAN (some hosts generating a lot of traffic, some hosts generating lesser traffic), number of blocks per link layer

packet and link utilization is discussed below. The calculation is described in the Appendix 7.3

Effect of number of hosts

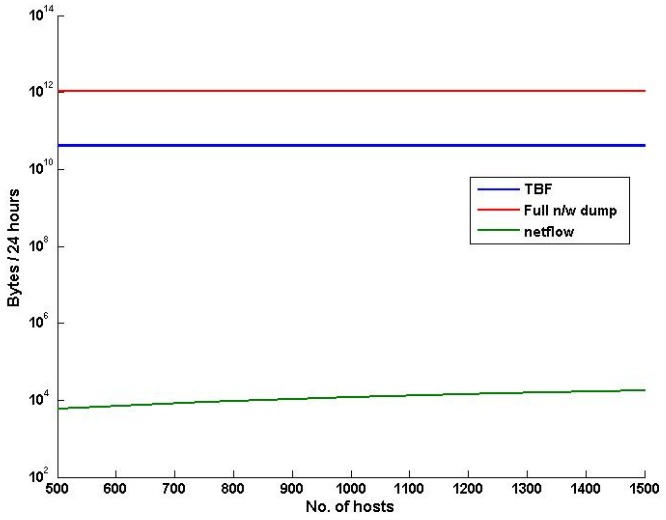

Figure 4.3: Comparison of storage requirements for full network dumps, tagged bloom filter structures and netflow data assuming a 1Gbps link with 10 percent utilization

Figure 4.3 shows the comparison of storage requirements for tagged bloom filter structures compared to full network dumps and netflow data assuming 10 percent link utilization and 8 blocks per link frame. The volume increases by 10 bytes for every host added as these 10 bytes are used to store the new host's information (host IP, host MAC). Also, the tagged bloom filter structure reduces storage requirements for storing full network dumps by a factor of 25. It requires much more place than required to store flow information (source IP, destination IP and flow length (4 bytes)) but provides forensic details about the payload unlike netflow information. For 1000 hosts, about 40GB of data accumulates in 24 hours which we believe is feasible to store for a couple of days to weeks.

Effect of link utilization

Figure 4.4 shows the comparison of storage requirements for our tagged bloom filter structures compared to full network dumps and netflow data assuming a 1Gbps link with differing link utilization and 8 blocks per link frame. The volume of data generated varies linearly with respect to the utilization of the link. Here we again see that the tagged bloom filter structure requires significantly less storage than to store the whole network dump but

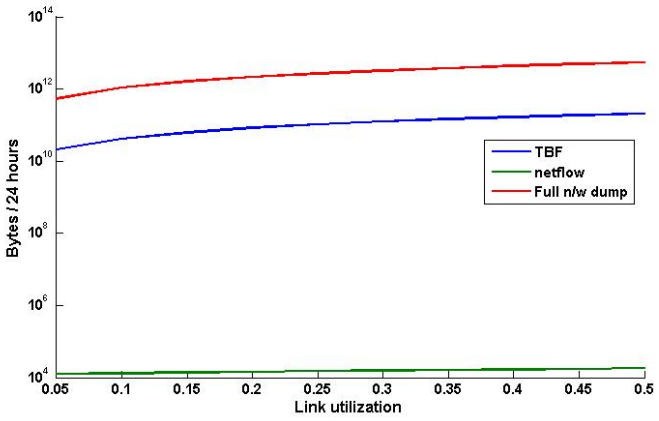

Figure 4.4: Comparison of storage requirements for full network dumps, tagged bloom filter structures and netflow data assuming a 1Gbps link with differing link utilization

a lot more than to store just netflow information.

Effect of block size

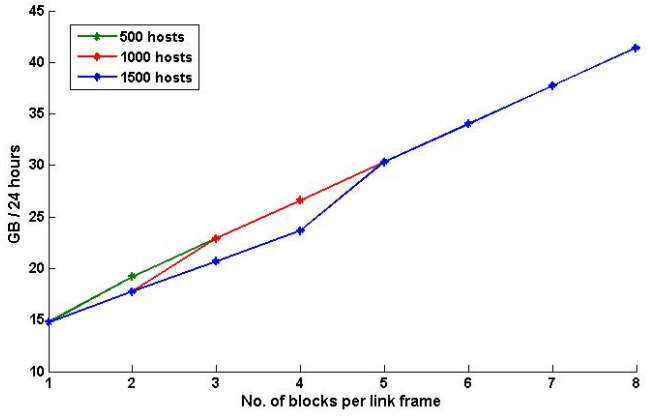

Figure 4.5: Total volume of data per 24 hours vs. number of blocks per link layer packet assuming a 1Gbps link with 10 percent utilization

Figure 6.1 shows the variation in the volume of data accumulated in 24 hours versus the number of blocks per link layer packet for different number of hosts assuming a 1Gbps line with 10 percent utilization. Increasing the number of blocks per link layer frame helps improve the granularity of searching for a given payload in the bloom filter. However, a side effect may be false positives if many files have a common block of data.

The size of the block-id in the tag depends upon the size of the bloom filter

which depends upon the number of blocks that the frame is divided into and the number of hosts. As the number of hosts increases, the size of the bloom filter decreases and hence the size of the block-id decreases. However, after a threshold number of blocks, because of byte boundaries, the block-id size remains the same irrespective of the number of blocks and the number of hosts for the considered spectrum of block sizes and number of hosts. The other parameters adding to the data volume like the source and destination information depends only upon the number of link layer frames which is a constant for a given link speed and link utilization.

Thus, the volume of data converges for varying number of hosts for a given link speed and utilization above a corresponding block size.

Effect of heterogeneity in network behavior of hosts

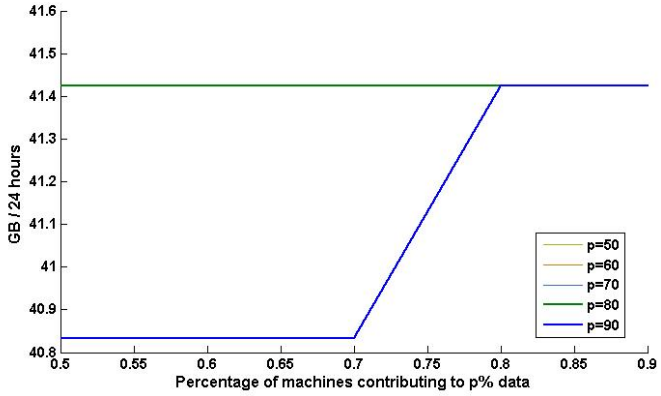

Figure 4.6: Total volume of data per 24 hours vs. Percentage of 1000 hosts contributing to 'p' percent data assuming a 1Gbps link at 10 percent utilization

Figure 4.6 shows the effect of having hosts that generate different amounts of network traffic on the same LAN on the total volume of data generated in 24 hours, assuming a 1 Gbps link at 10 percent utilization and 8 blocks per link frame. We observe that the total amount of data generated is a constant irrespective of varied host behavior in generation of network data. This is understandable because the total volume of data depends only upon the number of link layer frames and the number of blocks into which each frame is divided. The difference at 'p=90' percent is again due to difference in the block-id size compared to the other scenarios.

An optimization of the above approach would be to use smaller bloom filters for all hosts at the beginning and add new bloom filters when the number of packets inserted into the old filters are the maximum that the filters were designed to hold for a certain False Positive Rate.

4.2.2 Host data collection and storage

Background

The extent of change to the main memory largely depends on the usage patterns of the host. In 'Forensic Discovery' (Chapter 8), the authors show that certain memory pages like those corresponding to files do not change much. This implies that the feasibility of differential storage of such data would be interesting to examine. In this project, the theoretical feasibility of periodic data collection and differential storage is evaluated. The feasibility of such an approach is evaluated by measuring how much data accumulates under different conditions. The following subsections describe the type of data collected, tools that can be used to collect this data, the data structure used to store them and its evaluation.

Selection of the dataset

There is a large variety of information that one could collect from a running host. The choice of the subset of data to be collected will largely influence the amount of data generated for storage. The challenge is to strike a balance between the amount of useful information gathered versus the amount of data that is generated for storage. One approach to determining the data set of interest is examination of the various attack models whose forensic ivestigation the framework is expected to support and deriving a common set of data that could be used as evidence for the attacks.

In the context of the project, for a feasibility study, the data set chosen includes process-ids, process names, the corresponding executable and the current handles of each process. Handles includes open ports, directories, files and registry keys. The file handles should be processed into filenames using appropriate Windows APIs.

Tool survey

There are a number of tools that are useful in gathering volatile memory information. These include tools from sysinternals like psList.exe, handles.exe, etc., EnCase Forensic, Volatility, and Memoryze. Some of these work only for certain operating systems while others are platform independent.

In this project, Memoryze is used for gathering host data. This requires an installation of the tool on every host. The tool exports the gathered data in the form of XML. The tool offers a number of features besides the ones required on the project like ability to an image file or from a live host. An

important aspect of this tool regarding security is that it does not rely on operating system APIs but carries out an examination of the memory by itself.

It is to be noted that most of these tools have been designed for 'live forensics' in the traditional sense[16]. They are not optimized for running repeatedly at small intervals. As a result, they are relatively slow. Running these tools periodically rather than continuously as a monitor also makes the framework vulnerable to missing data because of timing issues.

Representation of host data

The data collected from the host will be stored in the form of XML according to the XML schema in Appendix 7.3 for the ease of processing and analysis.

Feasibility study

The amount of host data depends largely on the usage patterns of the hosts. In practice, it is difficult to develop a baseline which can be used as the minimum amount of data that the examination of a host will yield because it is not easy to ensure that one is baselining the right spectrum of machines. As a result, we adopt a slightly different approach to determine the data growth rates.

On fresh installations of Windows XP Professional and Windows Vista, the initial size of volatile memory data in each case is determined. The initial size of this data on a fresh Windows XP Professional installation is about 66KB and on a fresh Vista installation is about 200KB. Then, assuming that a host's process data relates to its operating system baseline, the volume of data collected differentially per host depends upon the initial size of data from that host, its memory changes and sampling frequency.

Figures 4.7 and 4.8 indicate that the total volume of host data generated varies linearly with the number of hosts and the initial size of the data per host. The maximum volume of data generated from 1000 hosts with an initial data size of 500KB is about 38GB which is feasible to collect and store on a daily basis. Figure 4.9 shows the comparison between storage requirements for the differential memory dumps of a host with 1GB RAM and for differential process information storage. The latter is significantly smaller and hence, feasible to store.

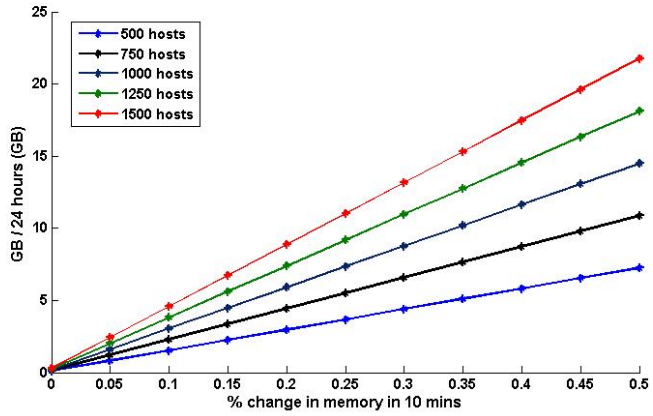

Figure 4.7: Variation in volume of host process data vs. percentage change in memory for different number of hosts assuming initial process data size of 200KB.

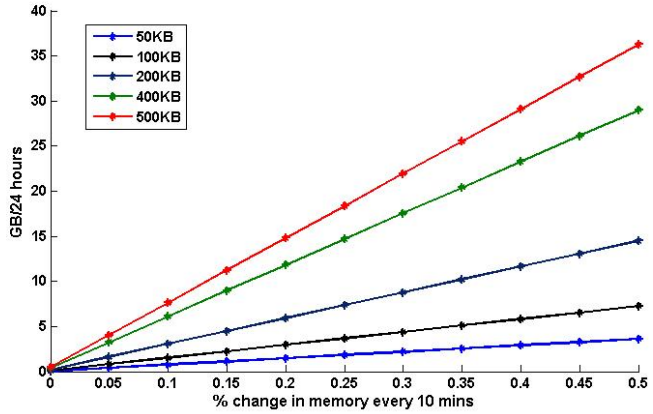

Figure 4.8: Variation in volume of host process data vs. percentage change in memory for different initial host process data sizes for a 1000 hosts

4.3 A combined forensic investigation approach

Combining forensic data from several sources allows cross verification of information and detection of any anomalies in the information derived from individual sources. For instance, malware like Mebroot, modify kernel drivers for communication across the network but no network activity is observed on the application layer. But the network agent detects the malware's communication activity. Despite the malware's ability to hide from the host data collection agent, an analysis of the network and host data would indicate an unusual network activity. Although the data collected from the host may no more be reliable, such cross verification can help trigger a more detailed investigation. Since most incidents have a remote attacker, securing the network layer to yield trustworthy data can help detect host compromise

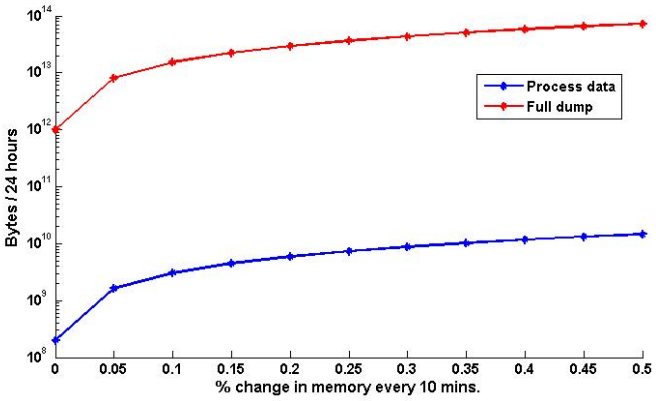

Figure 4.9: Comparison of storage requirements of limited process data vs. differential memory dumps of a 1GB RAM for a 1000 hosts

(hidden processes, ports, etc.) to a large extent.

Chapter 5

Framework Implementation

The implementation of the framework includes the choice of the programming language used for implementation, choice of libraries to parse capture files and create tagged bloom filters and finally, implementation of the logic for searching through the bloom filters. It is to be noted that the focus of the implementation was just proof of concept rather than optimality of performance.

We choose Java as the language of implementation. We used the jnetcap1.3 library (from Sly Technologies, Inc.) to parse network data capture files. We also used a slightly modified bloom filter implementation by Magnus Skjegstad to create the tagged bloom filters. Finally, we used a simple linear search algorithm to search through the tags and for different file blocks.

The implemented framework consists of three parts. The first part processes a .pcap file into a tagged bloom filter array. The second part searches for a given block of data in the bloom filter and returns the correspoding tags. Finally, the third part implements the forensic investigation procedures for two data leakage scenarios.

5.1 The jnetpcap library

The jnetpcap library is essentially a wrapper over the popular libpcap library for capture and analysis of network packets. Its supports various Ethernet protocols and several application layer protocols for protocol based processing like HTTP, VoIP, etc. However, in this implementation, it is used to extract the payload of every application layer protocol using TCP/UDP as its transport protocol. This makes the implementation independent of the application layer protocol used.

5.2 The Bloom filter implementation

The Bloom filter implementation by Magnus Skjegstad uses a java.util.BitSet to store the bloom filters bits. The number of elements inserted(say n) and the size of the bloom filter(say m) are configurable. The number of hash functions is calculated as ($k = m/n * ln2$). Instead of using several hash functions, each block is hashed the required number of times (k) by appending to it values between 0 to $(k-1)$ one at a time. The hash function used was MD5. The FPR for the bloom filter can be configured by choosing m and n approporiately.

This implementation was enhanced to obtain the intermediate values of the hashes when inserting an element for tag creation.

5.3 The capture file processor

Given a network data capture file (.pcap file), this application iterated over every packet, identified packets that used TCP/UDP, then extracted the application payload of such a packet and inserted it into an appropriate bloom filter along with the corresponding tags exactly as described in the design. Since the maximum payload size is 1460 bytes for an Ethernet packet, the block size was chosen as 292 bytes, i.e., each payload consisted of a maximum of five blocks. However, the block size is configurable.

```
prompt> java -jar pcapfileproc.jar <capture_file> <bf_file>
```

5.4 Forensic procedure implementation

The inputs to the forensic investigation include the file source IP and suspect/attacker IP, the appropriate tagged bloom filter file for the time frame to be searched, a directory consisting of a set of potentially transferred files and the byte offset of the block to be searched. The implementation returns a subset of those files that were transferred to the destination IP along with the exact timestamp.

```
prompt> java -jar case1.jar <suspect_IP> <bf_file> <dir_of_files>
<server_IP> <byte_offset>
```

For the second scenario, given the appropriate tagged bloom filter file for the time frame when the incident occured and the file that was transferred from a source whose IP is known, the implementation returns a set of IPs that received that file from that source during that time frame along with corresponding timestamps.

```
prompt> java -jar case2.jar <bf_file> <transferred_file>
<server_IP> <byte_offset>
```

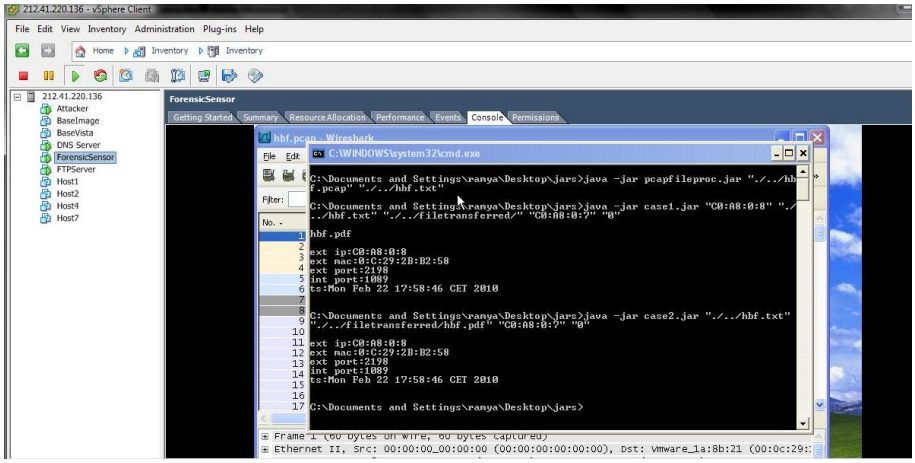

Figure 5.1: Snapshot of the analysis resultsin both scenarios when the attacker(C0:A8:0:8) transfers 'hbf.pdf' from an FTP server C0:A8:0:7 which is recorded in 'hbf.pcap'.

Chapter 6

Framework Evaluation

A scenario based evaluation of the framework involved study of two data leakage scenarios and infections like Mebroot [17], Torpig [18], Slammer [19]. The infections exploit vulnerabilities in the Windows operating systems or related applications and are discussed in detail in Appendix 7.3. The forensic investigation of simple data leakage scenarios provides insight into the advantages of combining host forensics and network forensics. The two scenarios considered for the purpose of demonstration are described in the following subsections.

6.1 Two data leakage scenarios

The security incident under consideration was the illegitimate transfer of file(s) by an attacker (known or unknown) from a large file server. It was assumed that all the information about the server was known including its IP and all the files on it.

For the purpose of illustration, the functionality of the network agent, both the network data stores and the forensic console as depicted in the forensic framework in the section 4.1 were combined into a single host (and is referred to as the 'forensic sensor' in the rest of the chapter). The data collection for the purpose of host forensics was treated only in theory.

A DNS/DHCP server, an FTP server (victim), an attacker with a suitable FTP client and the forensic agent were setup on four virtual machines running Windows XP Professional(SP 2) on an ESX server. We use a DNS/ DHCP server for windows from here. We also use the Firezilla server and client Firezilla server and client as FTP server and client respectively. Wireshark is used to capture the network packets and run on the forensic agent. Finally, processing of the capture files is done using a custom java implementation. The setup is illustrated in Figure 6.1

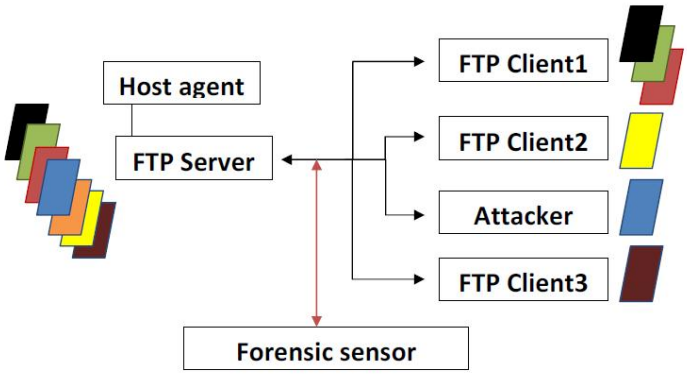

Figure 6.1: The framework evaluation setup

6.1.1 Scenario 1

The first scenario considers the case where the attacker's IP and an estimate of the time frame during which the incident occured are known. The purpose of the forensic investigation is to find the information that was transferred by the attacker and the exact time of transfer.

The forensic procedure consisted of finding the appropriate set of server bloom filter(s) to search using the time frame, followed by finding the particular bloom filter containing the attack's IP by a search on all tags of the bloom filter. Given this bloom filter and a set of files accessed by the server during that time frame (say from server logs or file system information), the files were checked for containment in the bloom filter resulting in a set of hosts that transferred that file and subsequently for transfer by the attacker by searching through this set.

6.1.2 Scenario 2

The second scenario considered a case where the exact file transferred by the attacker and an estimate of the time frame during which the incident occured are known. The purpose of the forensic investigation was to find the attacker's IP and the exact time of file transfer.

The forensic procedure in this case consisted of listing all clients that transferred the given file during the given time frame, using the appropriate tagged bloom filter(s) of the server corresponding to the given time frame followed by an analysis of whether those clients were authorized to transfer that file.

6.2 Results

The time required to obtain the results of the above forensic procedures is indicative of the usefulness of the framework despite possible inefficiencies in the implementation. For the following discussion, it is assumed that the tagged bloom filter to be searched is known. Also, it is assumed that the time required to process file handles to file names and finding these handles for a particular process during a particular timeframe is negligible. Finally, it is assumed that the search to check if an IP is contained in a set of IP's takes negligible time. Hence, the bottle neck is essentially the number of search operations performed in the bloom filter. The second scenario only requires one search in the bloom filter followed by examination of the tags. Hence, it takes constant time.

For the first scenario, the time function depends on some concrete details of the scenario. The implemented framework can recognize any transferred block corresponding to any size instead of the whole file transfer although the search maybe quite expensive if the block that was transferred is unknown. The growth function of the time required to find the file block transferred is examined theoretically in the following subsections.

6.2.1 Complete file transfer

Each look up in the bloom filter takes constant time. Assuming that the candidate files (say N in number) have a known unique block of size corresponding to the block size used to create the tagged bloom filter, the search would require one operation per file. The search to check if the attacker transferred a given file is assumed to take negligible amount of time. Thus, the time required to find the files transferred is linear in the number of candidate files.

Time for analysis α N

6.2.2 Partial file transfer with fixed block boundaries

Assuming that the file server transfers blocks of files instead of entire files based on fixed block boundaries (and this block size is the same as that used to create the tagged bloom filter), the number of look-ups into the tagged bloom filter is proportional to the number of blocks in each file each of which takes a constant time to be looked up in the tagged bloom filter.

Time for analysis $\alpha\ \Sigma_{i=1}^{N}\ size(file_i)/blocksize.$

If the file handles that were active during that timeframe are available from host forensic data (say H in number), then instead of searching for all files, the search space is reduced to the set of active files.

Time for analysis $\alpha\ \Sigma_{i=1}^{H}\ size(file_i)/blocksize.$

The gain in time for forensic analysis is proportional to the ratio of the number of active files to the total number of files.

The time taken to search for the all blocks of different file sizes was measured thrice and approximated on a Windows Vista machine with 4GB RAM. It was noticed that the search function is memory intensive as it holds the entire bloom filter in memory. For a block size of 292 bytes, it took about 1 second to find all blocks of a file of size 250KB, approximately 15 seconds to find all blocks of a file of size 4MB. This behavior seems linear. However for a file of size 90MB, it took between 35-40 minutes in each run to find all the blocks. This does not seem to be linearly increasing from the time taken in the previous two cases. The reason for this maybe high memory consumption due to very large data structures that are kept in memory which causes paging inefficiencies.

From the above analysis, for a file server with a 1000 4MB files, the amount of time required to find all blocks of data transferred to an attacker without any host file handle information is about 5 hours. But assuming that a 100 handles were active during the given timeframe, the time required reduces to 30 minutes.

This indicates that host forensic data (in this case the file handles) can help reduce the amount of computation (time) and hence improve the efficiency of forensic analysis.

6.2.3 Partial file transfer with arbitrary byte boundaries

Assuming that the file server is capable of streaming any portion of a file (but of a fixed block size), the number of look-ups into the tagged bloom filter is proportional to the sum of the $(sizeofthefiles - blocksize)$. As with the earlier scenario, one may be able to reduce this search space (and hence time) dramatically if the active file handles from the host forensics data is available.

6.3 Verification of evidence consistency

Combining evidence from various sources can not only improve the amount of information that is available for forensic analysis but also, help reinforce information gathered from one source by information gathered from another. This cross validation of the evidence gathered improves the confidence in the results of the forensic analysis, as well as, helps to detect any anomalies that could trigger further investigation. More specifically, having trustworthy

network data can help detect host compromise. If a hidden process transfers data such that it is not detectable by the host data collection agent, the network information will still yeild valuable details of what was transferred and when.

Chapter 7

Conclusion

In this chapter, a summary of the results, the limitations of the framework developed and the outlook are discussed.

7.1 Summary of results

Our work aimed at combining mechanisms for host and network forensics for improving the quality of data available for forensic analysis and incidence response. A framework architecture for the collection and processing of host and network information was developed. The components of the framework include the host agents, the network agents, the host and network data stores and the forensic console. The architecture developed is extensible with respect to the types of incidents whose forensic analysis it is able to support. The extensibility is provided by the flexibility of the data collected. The network data structure for storing the collected data results in about 42GB of data per 24 hours assuming a 1Gbps line with line utilization of 10 percent for a 1000 hosts. The host data structure based on information derived from fresh installations of Windows XP Professional (SP2) and Windows Vista results in about 38GB of data per 24 hours for 1000 hosts sampled every 10 mins assuming that the volatile memory changes 50 percent every 10 mins. In summary, the framework results in about 80GB of data per 24 hours for 1000 hosts which we consider economically feasible to store. In addition, the processing of the raw data collected and the placement of sensors/agents has been discussed.

As an illustration of the advantages of combining host forensics and network forensics, a data leakage scenario with a victim file server has been implemented and discussed. It has been shown that it is possible to find the attacker, given the data leak that occured and that it is possible to find the exact information that was lost, given the IP of the attacker. The search function in the latter case was found to be time consuming for very large files(about 90MB). The advantage of having access to the file handles in this case greatly reduces the time required for a forensic investigation by

reducing the search space.

7.2 Limitations of the framework

The framework is limited its application by the environment in which it is deployed as well as some of the assumptions that have been discussed previously. A few limitations of the framework have been listed below.

a. The network agents are not applicable to networks where data is encrypted.

b. The framework creates a bloom filter for every new internal IP that is recognizes. In a highly dynamic network where hosts join and leave the network very often or if IP addresses change very often in a network, this would result in wasted storage. The given data volumes are based on the assumption that the IP address remains constant for a certain length of time (e.g. 24 hours).

c. The host data collection tools are not optimized for periodic execution and information collection. This results in time intervals where data from the host is lost.

d. There may also be privacy concerns against running a host agent on every host. Also the data collected may contain personal information and is subject to privacy laws and regulations. It will have to be protected against unauthorized access.

e. Finally, the implementation is not optimal in memory consumption. For very large bloom filters, the execution of forensic procedures is memory intensive and is hence, quite slow.

7.3 Outlook

This work considered the integration of host forensic mechanisms and network forensic mechanisms only. There are serveral other sources of information like physical entry-exit logs to a building, video surveilance, etc. that could prove be automated for integration into a forensic data store. There are also interesting complementary problems. One such is the development of new tools that can report differtial changes to volatile memory, hooks, etc. efficiently. One could also use better storage structures like databases instead of raw XML to speed up analysis and improve storage space requirements for host forensics. The network data structure currently uses a single bloom filter and could be enhanced with a hierarchy to improve search efficiency. Finally, the current implementations of the network data processing, host data processing and the forensic investigation could be optimized for performance with respect to time and memory.

Appendix A

A.1 Network data size calculation

We know that for a FPR=0.02, the size of a bloom filter must be eight times the number of elements it is expected to hold. Assuming all hosts on a LAN generate equal amount of traffic,

$N : Numberofhostsper LAN$
$M : NumberofLANs$
$u : Percentageutilizationofthelink$
$b : Numberofblocksperpacket$
$bsize : Sizeofbloomfilter$
$h : Numberofhoursoftraffic$
$linkspeed : speedoftheoutgoingInternetline$
$packetsize : sizeofalinklayerpacket\ 1500bytes$

$Numberofpacketsperhost = (linespeed{*}utilization{*}h{*}60{*}60{*}x)/packetsize$
$Sizeofthebloomfilter = (8 * Numberofpacketsperhost * b)/8bytes$
$Hostdatasize = 32bitsfortheIP + 48bitsfortheMAC = 10bytes$
$Datasizepertag = 32bitsfortheIP{+}48bitsfortheMAC{+}32bitsfor2ports+$
$16bitsforthetimestamp = 16bytes$
$Totalsizeoftagdata = Datasizepertag * Numberofpacketsperhost$
$Sizeofeachpointer = ceil(ceil(log2(Sizeofthebloomfilter))/8)$
$Totalsizeofpointers = b * Numberofpacketsperhost * Sizeofeachpointer$
$Totaldataperhost = Hostdatasize{+}Sizeofthebloomfilter{+}Totalsizeoftagdata+$
$Totalsizeofpointers$
$Totaldataperdayforallhosts = Totaldataperhost * N * M$

A.2 Host process data schema

The following XML schema is used to store host process information. It includes the process-id, name of the process, the path of the executable responsible for the process and the list of file, directory and port handles of the process. The file and directory handles can be processed into filenames and directory names using Windows APIs.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Details" type="DetailsType"/>
<xs:complexType name="processType">
<xs:sequence>
<xs:element type="xs:string" name="pid"/>
<xs:element type="xs:string" name="name"/>
<xs:element type="xs:string" name="Username"/>
<xs:element type="xs:string" name="pathof"/>
<xs:element type="xs:string" name="parentpid"/>
<xs:element type="HandleListType" name="HandleList"/>
<xs:element type="PortListType" name="PortList" minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="DetailsType">
<xs:sequence>
<xs:element type="processType" name="process" maxOccurs="unbounded"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="HandleListType">
<xs:sequence>
<xs:element type="xs:string" name="Handle" maxOccurs="unbounded"
minOccurs="0"/>
</xs:sequence>
</xs:complexType>
<xs:complexType name="PortListType">
<xs:choice maxOccurs="unbounded" minOccurs="0">
<xs:element type="xs:string" name="protocol"/>
<xs:element type="xs:string" name="port"/>
</xs:choice>
</xs:complexType>
</xs:schema>
```

A.3 Framework evaluation

In this section, a brief description of the infections in terms of the vulnerabilities they exploit in each layer of a host and the network is discussed. The data required for reconstuction of these infections can be derived from the effects of the infections on the various layers.

Layer		Mebroot	Torpig	Slammer
Hardware layer	Vulnerability exploited			
	Effect	Malware executable on file system,boot sector changes	Modified system .dlls in System32 folder.	
OS layer	Vulnerability exploited	Inadequate user privilege enforcement, unprotected hooking functionality	unprotected hooking functionality	
	Effect	Creation of new services, devices, registry keys Modification of system executables like NTOSKRNL.exe, etc. and modification of system data structures like disk.sys,IRP_MJ_, etc.*	Modified memory signature of system utilities due to injection of Torpig	
Application layer	Vulnerability exploited	Browser vulnerability	Browser and other application vulnerabilities	Buffer overflow vulnerability in unpatched Microsoft SQL server
	Effect	Drive by download	Data from applications like password manager, etc. is accessible to Torpig	

Table A.3-1: Effects of Mebroot, Torpig and Slammer worm on various layers of IT infrastructure.

Layer		Mebroot	Torpig	Slammer
Network layer	Vulnerability exploited	Access and modification rights to internal NDIS structures		
	Effect	Modification of NDIS structures and encrypted communication	Encrypted communication, opening of ports	Generation of network packets containing the same worm

Table A.3-1 continued: Effects of Mebroot, Torpig and Slammer worm on various layers of IT infrastructure.

Bibliography

[1] R. McKemmish. What is forensic computing? *Australian Institute of Criminology Trends and Issues*, 1999.

[2] N. Savant A. Bronnimann H. Shanmugasundaram, K. Memon. Fornet: A distributed forensics network. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 1–16, 2003.

[3] W.-C.; Maier D.; Walpole J. Goel, A.; Feng. Forensix: A robust, high-performance reconstruction system. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 155–162, 2005.

[4] Timothy Fraser William A. Arbaugh Nick L. Petroni Jr., AAron Walters. Fatkit: A framework for the extraction and analysis of digital forensic data from volatile system memory. *Digital Investigation*, 3(4):197–210, 2006.

[5] Samuel T. King and Peter M. Chen. Backtracking intrusions. *ACM Transactions on Computer Systems*, 23(1):51–76, 2005.

[6] Frank Adelstein. Live forensics: diagnosing your system without killing it first. *Commun. ACM*, 49(2):63–66, 2006.

[7] Bruce J. Nikkel. Generalizing sources of live network evidence. *Digital Investigation*, 2(3):193–200, 2005.

[8] Bradley Schatz. Bodysnatcher: Towards reliable volatile memory acquisition by software. *Digital Investigation*, 4(1):126–134, 2007.

[9] Kulesh Shanmugasundaram, Hervé Brönnimann, and Nasir Memon. Payload attribution via hierarchical bloom filters. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 31–41, 2004.

[10] Yongping Tang and Thomas E. Daniels. A simple framework for distributed forensics. In *ICDCSW '05: Proceedings of the Second International Workshop on Security in Distributed Computing Systems (SDCS) (ICDCSW'05)*, 2005.

[11] Stefan Kornexl, Vern Paxson, Holger Dreger, Anja Feldmann, and Robin Sommer. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 23–23, 2005.

[12] Sarah Mocas. Building theoretical underpinnings for digital forensics research. *Digital Investigation*, 1(1):61 – 68, 2004.

[13] Brian Carrier. Defining digital forensic examination and analysis tools using abstraction layers. *International Journal of Digital Evidence*, 1(4), 2003.

[14] E. Rosti D. Bruschi, A. Ornaghi. S-arp: a secure address resolution protocol. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 66– 74, 2003.

[15] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[16] Aaron Stanley Eoghan Casey. Tool review remote forensic preservation and examination tools. *Digital Investigation*, 1(4):284–297, 2004.

[17] Elia Florio Kimmo Kasslin. Your computer is now stoned (...again!). the rise of mbr rootkits. In *VIRUS BULLETIN CONFERENCE*, 2008.

[18] Brett Stone-Gross and et.al. Cova. Your botnet is my botnet: analysis of a botnet takeover. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 635–647, 2009.

[19] Paxson V. Savage S. Shannon C. Staniford S. Weaver N. Moore, D. Inside the slammer worm. *IEEE Security and Privacy,*, 1(4):33–39, 2003.

[20] David Solomon Mark E. Russinovich. *Microsoft Windows Internals*. Microsoft Press, A Division of Microsoft Corporation, 2005.

[21] Wietse Venema Dan Farmer. *Forensic Discovery*. Addison-Wesley Professional Computing Series, 2005.