Holmes: A Data Theft Forensic Framework

Ramya Jayaram Masti [#1], Vincent Lenders [*2], Mario Strasser [#3], Stefan Engel [*4], Bernhard Plattner [#5]

[#] *ETH Zurich, Switzerland*
[1] rmasti@inf.ethz.ch
[3] strasser@tik.ee.ethz.ch
[5] plattner@tik.ee.ethz.ch

[*] *Armasuisse, Switzerland*
[2] Vincent.Lenders@armasuisse.ch
[4] stefan.engel@armasuisse.ch

***Abstract*—This paper presents Holmes, a forensic framework for postmortem investigation of data theft incidents in enterprise networks. Holmes pro-actively collects potential evidence from hosts and the network for correlation analysis at a central location. In order to optimize the storage requirements for the collected data, Holmes relies on compact network and host data structures. We evaluate the theoretical storage requirements of Holmes in average networks and quantify the improvements compared to raw data collection alternatives. Finally, we present the application of Holmes to two realistic data theft investigation scenarios and discuss how combining network and host data can improve the efficiency and reliability of these investigations.**

I. Introduction

Digital forensics is the science of identifying, collecting, preserving, analyzing, and presenting digital evidence. It is an important aspect of computer system security as it allows reconstruction and investigation of the root cause of security incidents like computer infections, intrusions and data thefts. Despite the emergence of powerful forensic tools like Encase [1], Memoryze [2], or NetWitness Investigator [3], two main challenges remain in practice. First, although several potential sources of forensic evidence have been identified at the host- and network-level, they remain distributed and uncorrelated. Integration and correlation of information from these sources is key to understanding the root cause of security incidents and validating the integrity of the digital evidence. Next, given the large volume of potential forensic evidence, it is necessary to minimize the volume of data that has to be stored and ensure that it remains useful for different types of investigations. The efficiency of collection, storage and analysis of such information will determine the success of forensic investigations.

In this paper, we propose Holmes, a forensic framework that addresses the above challenges in the context of data theft investigations. Holmes is a support system for semi-automated postmortem investigation of data thefts in enterprise networks. Holmes continuously collects and preserves potential forensic evidence. The framework consists of two main components: *(i)* a compact data structure based on tagged bloom filters to infer data transfer relations from network-level observations and *(ii)* a host process data structure to recover past user actions from process and file access activities. Holmes achieves scalability and integrity by combining these two sources of observations. Furthermore, the correlation of network- and host-level information can not only improve the reliability of the forensic evidence (e.g., detection of host compromise due to anomalous network behavior) but can also result in more efficient investigations by narrowing the potential directions (e.g., number of files to be searched) of postmortem analysis.

We study the data storage efficiency of Holmes in detail and provide initial insight into its data processing efficiency. Specifically, we evaluate the storage requirements of Holmes analytically and describe its dependency on factors like network link utilization, rate of change of host memory and network heterogeneity. We show that the proposed data structures require significantly lesser storage than full network and host memory dumps but retain enough network and host process information to allow data theft investigations involving even partial file transfers. For example, on a 1Gbps network with a 1000 hosts and an average utilization of 10 percent, Holmes requires about 42GB of storage per day for network data, providing a cost reduction by a factor of 25 approximately over full network dumps. Assuming similar settings, the host data structure results in about 38GB per day providing a cost reduction of 10^4 over full memory dumps (assuming uncompressed data). Finally, we implemented the key components of the Holmes framework and applied it to an example data theft investigation. We show that using Holmes, we can identify the exact data stolen, given an attacker's IP address and estimated time of theft. Holmes can also be used to identify all hosts (IP addresses) that transferred a specific file during a given time frame.

The rest of this paper is organized as follows: In Section II, we discuss the goals of data theft investigations and derive a set of requirements for forensic frameworks that have to support such postmortem analysis. We describe the design of Holmes in Section III and present its implementation and an evaluation of its storage requirements in Section IV. Finally, Section V and VI contain related work and conclusions.

II. Problem and Security Requirements

In this section, we define the design goals for Holmes using the example of a large protected file server in an enterprise that

WIFS'2011, November, 16-19, 2011, Foz do Iguaçu, Brazil.
978-1-4244-9080-6/10/$26.00

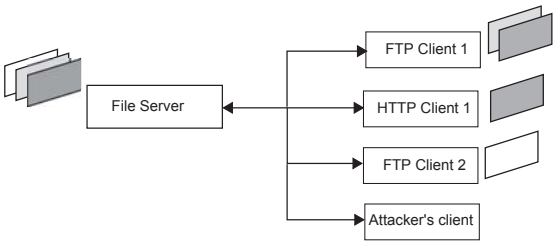


Fig. 1. A file server allows clients to retrieve files using different protocols (e.g., HTTP, FTP). One of the clients is controlled by an attacker. The attacker may be an authorized person who later misuses the privileges by handing the file(s) to a third party or an unauthorized client that gains access to the server (e.g., by exploiting a software vulnerability on the server).

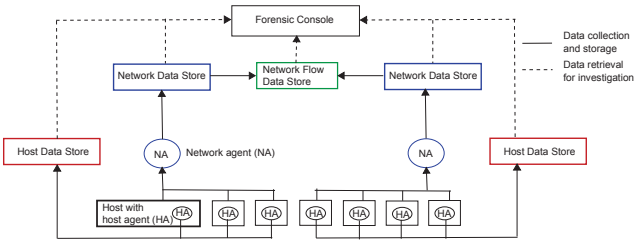


Fig. 2. The architecture of Holmes: Holmes consists of trusted network and host agents that periodically collect, process and transfer network and host data respectively to data stores. The data is stored in custom formats to reduce storage costs and can be accessed from a central forensic console.

allows file transfers using multiple protocols (e.g., HTTP, FTP, etc.). Many clients (Figure 1) can simultaneously retrieve files from the file server. We assume that an attacker may infect any such client but that he cannot compromise the network infrastructure. The attacker could be a person with granted access rights on the server who later misuses his privileges by handing the file(s) to a third party or an unauthorized client that gains access to the server (e.g., by exploiting a software vulnerability on it). Central to forensic analysis in these cases is the need to identify what data is forensically relevant, the on-line collection and correlation of this potential forensic evidence from different sources (e.g., mobile devices, third party information, host data, network data) and finally, efficient storage and processing of this data over extended periods of time. In this work, we focus on two such sources of forensic information, namely, the host and the network. Ensuring efficient collection, storage, analysis and integration of potential forensic evidence from these sources is challenging because it requires processing and correlating large volumes of data. In summary, our data theft forensic framework must support the following functionality:

1) It must enable collection of data on a continuous basis from hosts and networks.
2) It must ensure the integrity of the data collected prior collection and during storage.
3) Data collection and analysis must scale with increasing number of hosts and network connections.

Furthermore, a postmortem investigation must provide details regarding what data was stolen (which file(s)), by whom (e.g., IP of the unauthorized host used by the attacker or attacker's IP), the exact time of the incident and details regarding how the data was stolen (e.g., the protocol used, login information). However, such an analysis could have different goals depending on what information the investigator knows. We discuss two possible examples below.

1) Scenario I: Given the attacker's IP address and estimated time of the data theft, the forensic investigation should reveal the exact content downloaded by the attacker and the exact time of its transfer.
2) Scenario II: Given the exact content transferred by the attacker and estimated time of the data theft, the forensic investigation should identify the attacker's IP address and the exact time of the content transfer.

III. The Holmes Forensic Framework

Given the above design goals, we now present the architecture of the Holmes forensic framework (Figure 2). Holmes consists of trusted network agents and host agents that collect, process and transfer network and host data respectively. The network agent is a designated node that sniffs data off the LAN, processes it off-line into a custom format (that contains network flow and payload information per network packet) and transfers it periodically to the network data store. It may be implemented as a passive tab device or as an in-line device. Alternatively, a separate entity may post-process the data collected by the network agent into the custom format. Network agents must be able to intercept traffic of interest in unencrypted form. In an enterprise architecture, this could be achieved at the entry point of a data center or at network proxy interconnection points. Since network flow information is small and can be retained over longer periods of time, it is retrieved from the old data at the network data store and transferred to the network flow data store periodically.

An instance of the host agent runs on each host collecting information about the activities on it. Given the large variety of available host information, identifying and storing only the relevant information is a challenge. We discuss this further in Section III-B. The integrity of the host agent itself may be protected using trusted computing techniques [4] or by running it as an independent module that accesses host memory directly via DMA techniques [5]. Note that even if the host agent is compromised, the network agent will still provide trustworthy information because we assume that the attacker cannot compromise any network infrastructure components.

A. Network Data Collection and Storage

In gigabit networks, preservation of network data over long periods of time for forensic analysis requires optimizing the data stored in three aspects: volume, content and ease of processing. There is an inherent trade off in the volume of data that has to be archived versus the amount of information that can be retrieved from that data. For instance, network flow (e.g., netflow or IPFIX) data is smaller in volume than full network dumps but reveals only communication patterns and gives no information about packet payloads. On the

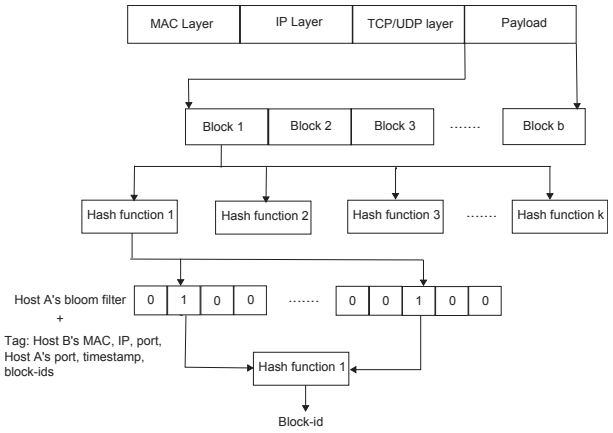


Fig. 3. Processing of a link layer frame from host *A* to host *B*: The payload is divided into equally sized blocks each of which is inserted (only block 1 is shown in the figure) into Host *A*'s bloom filter. Host *B*'s information (IP, MAC and port) and the block-id are combined into a new tag for this frame. The bloom filter and the tags together make up the Tagged Bloom Filter (TBF).

other hand, full network dumps reveal all information about network traffic. As a trade-off between the above approaches, we propose a data structure called the tagged bloom filter yields partial information about packet payloads and netflow information.

Tagged bloom filter

The tagged bloom filter (TBF) structure is tailored for tracking transfer of known payloads (e.g., a list of files residing on a server) by hosts. It is a variant of the Block based Bloom Filter described in [6] but, unlike in [6], each block is not appended with its offset before insertion into the bloom filter. Assuming that the IP lease period is 24 hours and IP-MAC-host associations are known, we describe how the network agent processes each link layer frame from host *A* to host *B* (Figure 3) to create a host *A*'s TBF. Without loss of generality, we assume that *A* is an internal host communicating with *B* outside the enterprise. In this paper, we restrict our analysis to TCP/ UDP-IP traffic.

- The IP, MAC address, TCP/ UDP port pertaining to *A* and *B* are extracted.
- The payload is split into b blocks of equal sizes. By payload, we mean the payload of the application layer protocol (e.g., HTTP or FTP) or the payload of TCP/ UDP layer itself. This makes bloom filter queries independent of the application layer protocol. The last block is not padded even if it is smaller than the chosen block size.
- A new bloom filter (of size m to hold n network packets of b blocks each, using k hash functions) is created for *A* if it does not already exist and is initialized to a set of zeros. Then, each block (say b_i of the packet is inserted into *A*'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 ($b_{i,1}$-$b_{i,k}$) of the bloom filter to 1. Using multiple hash functions reduces the number of false positives especially when data blocks inserted differ only by a few bytes [7].
- Furthermore, a 'tag' is created with the details of *A*'s (TCP/ UDP) port and *B*'s (TCP/ UDP) port, MAC and IP and a fresh timestamp. Tags corresponding to *A*'s incoming and outgoing network packets are stored separately. For each inserted block, the values of ($b_{i,1}$-$b_{i,k}$) positions are concatenated and hashed again using the bloom filter's first hash function. This value modulo the size of the bloom filter (which we call **block-id**) is added to the tag. The block-id can be used to identify the tag (and hence, flow information) corresponding to a block.

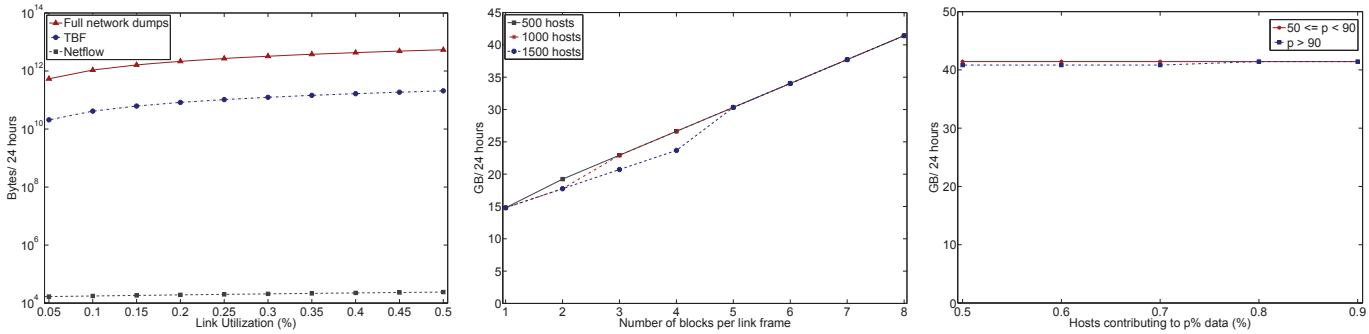
If *A* is outside the enterprise and *B* is on the internal network, the same operations are performed on *B*'s TBF instead of *A*'s TBF. If both hosts are on the internal network, then the operations are performed on the TBF of the network packet's source. Finally, the entire data structure is re-created periodically (e.g., 24 hours in our case) when the IP address changes. We note that one can easily extract connection information using only the tag data and dropping the bloom filters before transfer to the network flow data store.

TBF sizes are chosen to ensure a given False Positive Rate (FPR) on containment queries. Inserting more packets into a fixed size TBF (than it was designed to hold originally) increases the FPR. In order to keep the FPR constant, each host's TBF size must be tailored to its network activity. However, this makes TBF management complex on a heterogeneous network. Instead, a simpler solution is to tailor the initial TBF size to fit the average network activity (number of packets sent and received) of hosts on a LAN. For a host with above average network activity, a new TBF is created after its current filter contains the maximum number of packets it was designed to hold. This obviates the need for large TBFs for all hosts.

A TBF 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 transferred any of them. The search granularity depends upon the block size. All queries proceed by first finding the TBFs that contain block(s) of the concerned payload. Then, the tag(s) corresponding to these blocks are used to extract connection information. We refer the reader to [7] for details regarding bloom filter querying.

B. Host Data Collection and Storage

There is a large variety of information that one could collect from a running host. In traditional host forensics [8], [9], the entire volatile memory and disk storage are used to extract useful information from hosts. However, collecting and storing full volatile memory and disk images on a periodic basis over the network, is not a scalable solution. Holmes therefore collects only a subset of host data. The host data storage requirements largely depends on the choice of this subset of data. One way to choose what data has to be collected and stored is to examine various attacks whose forensic investigation the framework is expected to support and derive a common data set that can be used as evidence for them.



(a) Comparison of TBF, full network dumps and netflow volumes

(b) Varying block size

(c) Heterogeneous network

Fig. 4. The Holmes framework reduces network storage requirements by a factor of 25 compared to full network dumps. The storage requirement depends on the block size used during the tagged bloom filter creation and remains fairly stable for a given link utilization and block size even on a heterogeneous network. For a 1Gbps network with a 1000 hosts, at 10 percent utilization, Holmes generates about 40GB of network forensic data over 24 hours.

From the perspective of data theft investigations, the Holmes host agent periodically collects *(i)* all process ids, *(ii)* all process names, *(iii)* the corresponding executable names and *(iv)* all file handles including open ports, directories, file names and registry keys. Process ids and names reveal the types of applications (e.g., an ftp client) that were running at the time of an incident but not their precise activity. File handles provide file or directory access information. This information can be correlated with information from the network agent to speed up data theft investigations. We propose the differential storage of this data using an XML structure. Exploration of more efficient storage structures for host data is intended as part of future work.

Holmes provides forensic data from host and network resources which can be used to cross-verify and detect any anomalies in them. For example, malware which may be invisible to a host agent may be detected by the network agent due to its network activity. Securing the network layer to yield trustworthy data can help detect host compromise (hidden processes, ports and so on). Besides, correlation of information from various sources can also help improve the efficiency of a forensic investigation. For instance, trustworthy file handle information from the host agent can be used to reduce the search space in data theft investigations.

IV. Implementation and Evaluation of Holmes

In this section, we present our prototype implementation of Holmes, evaluate its storage requirements, and demonstrate its applicability to data theft investigations. Evaluation of the efficiency of TBF creation from raw network data and querying are intended as part of future work.

A. Implementation

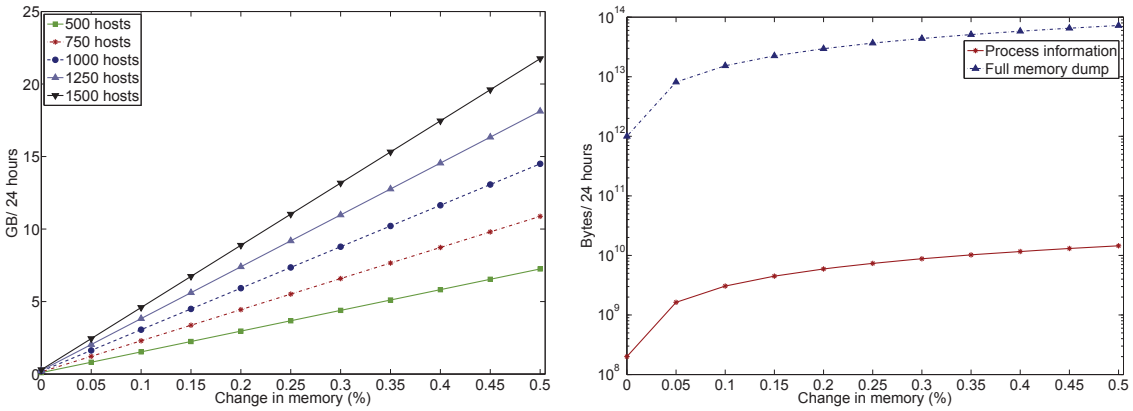
We have implemented a prototype of the network agent, host agent, and data stores of Holmes. Our network agent prototype captures raw network data (e.g., in a .pcap file), and then, extracts each packet from it and inserts it into the appropriate TBF. The FPR of the TBF is configured by choosing n and m appropriately. Instead of using k different hash functions, each block is concatenated with values between 0 to *k-1* and then hashed k times. The block size is configurable. The host agent uses Memoryze [2] to retrieve meta information from host memory, extracts relevant data from it and sends it to the host data store periodically.

B. Network Storage Requirements Evaluation

We now discuss the factors affecting the network storage requirements of Holmes. We assume a single network agent monitoring a 1Gbps link. The TBF size is chosen such that the FPR is about 0.02 [7]. Compression techniques can further improve our projected storage requirements.

Figure 4(a) shows that the network data storage requirements of Holmes varies linearly with respect to the link utilization. The use of TBFs reduces storage requirements for network data by a factor of 25 approximately compared to full network dumps. Although it is bigger in volume than netflow information (source and destination IPs and ports and flow length (16 bytes per record)), unlike netflow data, it provides information about the payload. Using TBFs, the storage 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 address). For 1000 hosts, about 42GB of data accumulates in 24 hours assuming 10 percent link utilization without compression

Increasing the number of blocks per link layer frame improves the granularity of searching for a given payload in the bloom filter but may also result in false positives if many files have a common block of data. Figure 4(b) shows the variation in the storage requirements with respect to the number of blocks into which the network packet payload is divided. The total storage requirement can be categorized into bloom filter storage space and storage space for tags that contain connection information (IP, MAC, ports, timestamp) and block-ids (Section III-A). For a given FPR and link utilization, the storage requirement depends on the size of the block-id. As the number of hosts increases, the size of the bloom filter (and hence, the block-id) decreases. This results in smaller block-id storage requirements initially. However,



(a) Variation with host memory change

(b) Comparison with complete memory dumps

Fig. 5. The volume of the process data stored by Holmes is significantly lesser than the volume of full memory dumps. It varies linearly with the percentage change in host memory for a given operating system baseline (baseline = 200KB in the figure) and data collection frequency (1 sample/10 mins in the figure) .

for the considered range of block sizes and number of hosts, the block-id size remains constant above a threshold number of blocks (3 blocks for 1000 hosts or 5 blocks for 1500 hosts from Figure 4(b)) because of byte boundaries. Thus, the required storage volume converges for varying number of hosts for a given FPR, link speed and utilization below a corresponding block size. A similar argument holds in the case of heterogeneous networks in which some hosts generate more data than others (Figure 4(c)).

C. Host Data Storage Requirements Evaluation

The required storage space for host data depends largely on the rate and degree of change in host memory which in turn depends on host usage patterns. The heterogeneity in host platforms (differences in operating system, applications and so on) makes it difficult to derive a generic baseline for host data storage requirements. We handle these differences between host platforms partially by relating each host's process data volume to its operating system baseline. First, we measure process data volume on fresh installations of different operating systems. Then, the volume of data collected differentially per host depends upon the initial size of data from that host (which we assume is a multiple of its operating system baseline), degree of change in its memory over time and frequency of process data collection.

We used Memoryze to collect the information relevant to data theft analysis (Section III-B). Memoryze directly examines host memory instead of depending upon operating system primitives and hence, its reports are unaffected even if the operating system is compromised. However, live analysis using such tools requires more resources and takes longer than using hooks on operating system calls. As a result, the meta information is available only periodically which is a disadvantage. Presently, this tool works only on the Windows platform and exports data in XML format which is further processed to keep only the relevant parts. Such analysis on fresh installations of Windows XP Professional and Windows Vista, revealed that the baselines for Windows XP Professional and Windows Vista are about 66KB and 200KB respectively. We believe that the results using any other tool should not be very different but this has to be verified.

Figure 5(a) shows how host data storage requirements vary with respect to rate of change in host memory assuming that hosts are sampled once every 10 minutes. More frequent sampling of host memory will result in larger storage requirements. We note that the storage requirements of Holmes for different levels of host memory change is significantly lesser than for storing full memory dumps (Figure 5(b)). Storing information about a 1000 hosts with over two times the process information as Windows Vista (with 500KB of process data) requires about 38GB (prior compression) assuming that each host's memory changes 50 percent every ten minutes over 24 hours.

D. Data Theft Investigations using Holmes

We applied our Holmes prototype to two forensic investigations scenarios (Section II) as shown in Figure 6. In each scenario, first, the search is narrowed down to a relevant set of TBFs using the time frame. In scenario I, these TBFs are searched for the attacker's IP. Then, using the server's host data, the files accessed by the server during this time are found and checked for transfer by the attacker. On the other hand, if the exact file stolen is known (scenario II), then the attacker's IP is found by listing all clients that transferred that file and checking if they were all authorized to access it. Holmes can also be applied to scenarios where an authorized client downloads a file and transfers it to a third-party over the network by combining the above approaches.

E. Discussion

While a thorough performance analysis of Holmes is necessary to establish its usability, we provide initial insight into factors affecting it. The practicality of investigations using Holmes largely depends on the efficiency of creating and querying the host and network data structures. While creating the data structures can be done offline, a better estimate of the time of theft will result in faster queries because it reduces the number of TBFs to be searched. If several files share

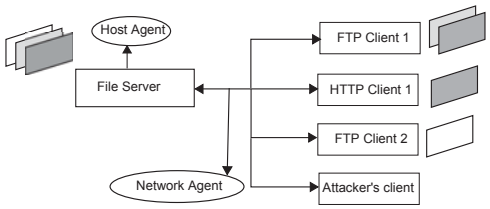


Fig. 6. Holmes for data theft investigations: We run the host agent on the file server and a network agent on a LAN containing the server and its clients.

common data blocks, fine search granularities will result in false positives. However, this can be handled by searching for more data blocks from the same file. An initial study shows that it takes about 1s to search for all blocks of a file of size 250KB, and about 15s for a file of size 4MB (on a host with 4GB RAM and running Windows Vista). However, searching for all blocks of large files is very inefficient (e.g., for a file of size 90MB, it takes about 4 hours). This may be improved by running several search threads in parallel but it needs further investigation. File handle data collected from the server can also reduce the search space but presently, host data collection tools are not optimized for periodic information collection. This results in time intervals where host data is not captured. Finally, although Holmes has been designed for use in networks where data is not encrypted end-to-end, it only collects meta-data at the host and compressed network data using bloom filters and hence, mitigates concerns regarding central collection of end-user information.

V. 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 [10]–[12]. 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 [13]. Most research focuses either on host forensics or network forensics but not on their combination. Modern approaches to host forensics emphasize on 'live forensics' (retrieval of volatile data on the host before it is shut down to collect its disk contents) [8], [14] and development of tools for it [9]. However, there are no references to continuous volatile memory examination, processing and storage as investigated here.

Contemporary research on network data retention for forensic purposes includes the use of bloom filters [6], [15], use of arithmetic coding for data compression [16] and storage of partial flow information [17]. However, notable disadvantages exist with each of these techniques. Bloom filters as used in [6], [15] do not allow the extraction of network flow data directly, 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 still scales with the number of hosts.

VI. Conclusions

We proposed Holmes, a forensic framework to support the investigation of data thefts. The framework combines information from the network and hosts for improved efficiency and correctness. We presented the design of tagged bloom filters that support queries which are independent of the application layer protocol and performed a preliminary study of periodic and differential collection of process data from hosts. We analyzed the storage requirements of Holmes by varying a number of parameters and showed its application to two data theft scenarios. Using Holmes, it is possible to find the IP address of the attacker's host, provided that the exact data leak that occurred is known. It is also possible to find the exact information that was lost, given the attacker's IP. We intend to study the feasibility and performance of processing the information collected using Holmes as part of future work.

References

[1] "Encase from guidance software," http://www.guidancesoftware.com/.
[2] "Mandiant memoryze," http://www.mandiant.com/products/.
[3] "Netwitness investigator," http://netwitness.com/.
[4] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, 2004, pp. 223–228.
[5] N. L. PetroniJr., T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot - a coprocessor-based kernel runtime integrity monitor," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 179–194.
[6] K. Shanmugasundaram, H. Brönnimann, and N. Memon, "Payload attribution via hierarchical bloom filters," in *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, 2004, pp. 31–41.
[7] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Internet Mathematics*, 2002, pp. 636–646.
[8] F. Adelstein, "Live forensics: diagnosing your system without killing it first," *Commun. ACM*, vol. 49, no. 2, pp. 63–66, 2006.
[9] B. Schatz, "Bodysnatcher: Towards reliable volatile memory acquisition by software," *Digital Investigation*, vol. 4, no. 1, pp. 126–134, 2007.
[10] K. Shanmugasundaram, N. Memon, A. Savant, and H. Bronnimann, "Fornet: A distributed forensics network," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 1–16, 2003.
[11] A. Goel, W.-C. Feng, D. Maier, and J. Walpole, "Forensix: A robust, high-performance reconstruction system," in *25th IEEE International Conference on Distributed Computing Systems Workshops*, 2005, pp. 155–162.
[12] N. L. PetroniJr., A. Walters, T. Fraser, and W. A. Arbaugh, "Fatkit: A framework for the extraction and analysis of digital forensic data from volatile system memory," *Digital Investigation*, vol. 3, no. 4, pp. 197–210, 2006.
[13] S. T. King and P. M. Chen, "Backtracking intrusions," *ACM Transactions on Computer Systems*, vol. 23, no. 1, pp. 51–76, 2005.
[14] B. J. Nikkel, "Generalizing sources of live network evidence," *Digital Investigation*, vol. 2, no. 3, pp. 193–200, 2005.
[15] M. Ponec, P. Giura, J. Wein, and H. Brönnimann, "New payload attribution methods for network forensic investigations," *ACM Trans. Inf. Syst. Secur.*, vol. 13, pp. 15:1–15:32, 2010.
[16] Y. Tang and T. 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.
[17] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. 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*, 2005, pp. 23–23.