Payload Attribution via Hierarchical Bloom Filters

Kulesh Shanmugasundaram
kulesh@isis.poly.edu

Hervé Brönnimann
hbr@poly.edu

Nasir Memon
memon@poly.edu

Department of Computer Science
Polytechnic University
Brooklyn, New York

ABSTRACT

Payload attribution is an important problem often encountered in network forensics. Given an excerpt of a payload, finding its source and destination is useful for many security applications such as identifying sources and victims of a worm or virus. Although IP traceback techniques have been proposed in the literature, these techniques cannot help when we do not have the entire packet or when we only have an excerpt of the payload.

In this paper, we present a payload attribution system (PAS) that attributes reasonably long excerpts of payloads to their source and/or destination hosts. The system we propose is based on a novel data structure called a Hierarchical Bloom Filter (HBF). An HBF creates compact digests of payloads and provides probabilistic answers to membership queries on the excerpts of payloads. We also present the performance analysis of the method and experimental results from a prototype demonstrating the practicality and efficacy of the system. The system can reliably work with certain packet transformations and is flexible enough to be used if the query string is spread across several packets. The system, however, can be evaded by splitting or by "stuffing" the payload. Future work focuses on making the system robust against such evasions.

Categories and Subject Descriptors: C.2.0 [Computer-Communication Networks]: General- *Security and protection.*

General Terms: Algorithms, Performance, Security

Keywords: Payload attribution, Hierarchical Bloom Filters, ForNet, Security

1. INTRODUCTION

In networking, "attribution" is the problem of determining the source and/or the destination of some instance of traffic. For IP networks, this problem arises both at the level of individual packets and at the higher level of payloads. At the level of packets, attribution of source is difficult because the source IP can be spoofed. At the level of payloads, determining which source sent or received a particular sequence of data is difficult because of the lack of reliable logging mechanisms in networks. Attackers often use "zombie" or "stepping stone" hosts as intermediaries. Therefore, being able to attribute traffic to a particular source host is not a panacea. It does, however, bring us a step closer to the attacker, making it a valuable start in tracing attackers.

Several packet marking schemes have been proposed to meet the first hurdle [28, 25, 10, 4, 14, 15]. These tools, however, require a set of network packets to do traceback and are not useful when one does not have the packets but only an excerpt of the payload. In order to meet the second hurdle, tools have been developed that can record the entire network traffic and archive them for short periods of time [1,12,20,3]. Since the means of cybercrimes are usually not known a priori, for example signature of a new worm, we would like to store the network traffic for weeks or even months so that we can go back in time to investigate incidents. Unfortunately, recording raw network traffic not only makes it infeasible to archive the traffic for prolonged periods of time but also raises serious privacy issues thereby limiting the usefulness of these tools.

In this paper, we look at one aspect of attribution where given a payload (or a significant portion of payload henceforth referred to as an *excerpt*) and a time interval we identify the senders and/or the receivers of the payload. We call this process *payload attribution*. Whereas payload attribution may not be useful with flooding-like attacks, where the packets can be empty or always the same, it is useful when incidents cannot be characterized by superficial features such as packet headers, packet rate, or a fixed length prefix of the payload. For example, with viruses, worms, and vulnerability exploits, it is often difficult to distinguish a benign packet from a malicious one by simply logging and analyzing these superficial features. A worm exploiting a remote vulnerability in a web server, for instance, would send a request to port 80 much like any legitimate web browser. By the time the worm's signature becomes available to Intrusion Detection Systems, the worm has already infected most of the network. In this scenario, a payload attribution system can help us identify hosts that received the worm and hosts that propagated the worm. This information can be useful for a network administrator to quarantine or clean-up infected hosts.

CCS'04, October 25-29, 2004, Washington, DC, USA.

Our contribution. We describe and analyze a compact hash-based payload digesting data structure, which we call a *Hierarchical Bloom Filter (HBF)*. We then describe and implement a simple payload attribution system that utilizes HBFs to compactly store payload digests and is capable of attributing excerpts of payloads. Note that unlike previous schemes, the proposed attribution system does not require an entire packet for attribution but only a reasonably long excerpt (for example, 128 bytes) of a packet's payload. Compared to recording raw network traffic, the proposed solution has lesser storage requirements and provides better privacy guarantees. We should point out that the system as presented can be evaded by generating packets with small payloads (for example, 64 bytes) or by "stuffing" the payloads and future work is necessary to make the system robust against such evasions.

A prototype system using HBF has been implemented in software. It performs very well on a moderate-speed network (intranets, medium-bandwidth links). The prototype monitors network traffic, creates hash-based digests of payload, and archives them periodically. A query mechanism provides the interface to answer postmortem questions about the payload. The accuracy of attribution increases with the length of the excerpt and specificity of the query. While the error rates of the individual Bloom filters are fairly high, combined together in the hierarchical structure they achieve low overall false positive rates. With the proper extension, the solution is effective even if the excerpt spans several packets: it will be detected with the same low false positive rate. The method is robust against packet transformations that don't manipulate payloads, such as packet encapsulation, fragmentation, or re-packetization. The query string known to the analyst may also appear in the payload in a compressed or encoded form. If the transformation is known (e.g. base64, uuencode, gzip, or encryption with known key) then we may be able to transform the excerpt appropriately to query the system. The low false positive rate ensures that if there is a match, it is very likely the excerpt has been transmitted under that form. Of course, it may be impossible to attribute an encrypted payload with an unknown encryption key and other streaming transformations. We report on its performance in Section 5, both in terms of the systems parameters as well as in a real-life experiment. Hardware implementation and feasibility of a payload attribution system on high-speed networks are beyond the scope of this paper and are part of our future work.

The rest of this paper is organized as follows: the following section discusses related work in detail. Section 3 presents Hierarchical Bloom Filters followed by the description of the design and implementation of a Payload Attribution System in Section 4. We present the experiments and results of the prototype in Section 5 and conclude in Section 6 with a summary and future work.

2. RELATED WORK

Work related to the one presented in this paper falls into two categories: work related to Bloom filters and work related to attribution systems.

2.1 Bloom Filters

Ever since Bloom filters were introduced by Burton Bloom in [5] they have been used in a variety of contexts. Here we present work related to Bloom filters in the context of network monitoring and security and refer the readers to [6] for a comprehensive survey on the subject.

A Bloom filter is a simple, space-efficient, randomized data structure for representing a set in order to support membership queries. It uses a set of k hash functions of range m and a bit vector of length m. Initially, the bit vector is set to 0. An element in the set is inserted into the Bloom filter by hashing the element using the k hash functions and setting the corresponding bits in the bit vector to 1. To test whether an element was inserted into the filter, we simply hash the element with the same hash functions and if *all* corresponding bits are set to 1 then the element is said to be present in the filter. The space efficiency of a Bloom filter is achieved at the cost of a small probability of false positives as defined by Equation 1, where n is the number of elements in the set [13].

$$FP = \left(1 - (1 - \frac{1}{m})^{kn}\right)^k \approx (1 - e^{-kn/m})^k. \qquad (1)$$

As far as we know, Bloom filters were first used in the context of security in OPUS [29]. OPUS uses a Bloom filter to store the list of dictionary words in a space-efficient manner to eliminate weak passwords. In [27] Bloom filters have been used in the Source Path Isolation Engine (SPIE) to trace IP packets over networks. SPIE creates hash-digests of packets and stores them in a router using Bloom filters. With wide deployment, SPIE is capable of tracking a single packet to its source on the Internet. A SPIE-equipped router creates a packet digest for every packet it processes using the packet's non-mutable header fields and a prefix of the payload. These digests are then kept at the network component for a preset amount of time. When an attack is detected by a security component, such as a firewall or an intrusion detection system, it can present the offending packet to SPIE to trace the packet's path and the originator of a packet can be identified by launching enough queries along the packet's network path. With SPIE, any single packet can be traced back to its origin as long as all the nodes on the packet's path have not yet discarded the packet digest. This also makes the system unsuitable for forensics applications where information may be needed from a significantly distant past. Our payload attribution method described in the following section is similar to SPIE in that both are digesting schemes. However, whereas SPIE is a packet digesting scheme, the method we propose in this paper is a payload digesting scheme. This key difference between them is explained by the fact that SPIE requires that we present the whole packet, or at least the non-mutable headers and prefix of the payload, to do traceback. However, in most cases we may not have the exact packet that carried a certain payload.

More recently in [16] the authors propose a novel technique, called Space Code Bloom Filters, for approximate measurement of traffic flows. Unlike previous measurement schemes, a novel Bloom filter based data structure, known as Space-Code Bloom Filter enables the method to track *all* traffic flows instead of just "heavy hitters."

Finally, Dharmapurikar et al. [11] propose to use Bloom filters for intrusion detection by matching known signatures to the payload. The problem they tackle, often referred to as deep packet inspection, involves detection of predefined signature strings or keywords starting at an arbitrary location in the payload. Their system uses very similar principles but proceeds on the other end (the Bloom filter contains

the signatures) and for a different purpose (intrusion detection). Hence archiving and storing the Bloom filters is not a concern for them, while they face the challenge of performing their query at line speed over all possible alignments and various block lengths.

2.2 Attribution Systems

Over the past few years extensive research has been done in developing feasible solutions to trace back network traffic to its source on the Internet. Traceback systems can generally be grouped into three broad categories: 1) Traceback of single packets (e.g. SPIE) 2) Traceback of network floods 3) Traceback of connection chains. We already described SPIE in the previous sub-section. We now briefly summarize work in the other two areas below.

Distributed denial of service attacks create large uncorrelated network flows towards a particular host or a set of hosts. Since source IP addresses usually spoofed the traceback of the attack to its source a non-trivial task. Researchers have proposed some clever solutions to the problem of tracing IP packets back to their source (IP traceback) [27,25,28,4,10,8,18]. Most of this work can be grouped into two main categories: one in which no extra network packets are generated [27,25,28,10,8] and the other in which a few extra network packets are generated [4, 18]. These mechanisms, however, are not effective against attacks that require a relatively smaller amount of packets. An encoding strategy proposed in [14,15] requires that border routers establish a trusted region and encode the router's IP in all egress traffic. A destination host can then decode the IP address of the closet border router to source of a packet from the IP fragment-id field. None of these methods can be used against malicious network events that can only be defined by a packet's payload, like for example, uploading a company's intellectual property to a public FTP site.

A related problem, at a different level of abstraction however, is tracing connection chains. Attackers often obscure their identity and location by forming a connection chain by logging into a set of compromised systems before attacking a target– known as stepping stones. Tracing the attack from the victim takes us only to the last link in the chain but not to the location of the attacker. In [30, 31], methods are proposed to trace intruders through stepping-stones. The method proposed in [30] creates "thumb-prints" of connections using packet content which can be compared to determine whether two connections contain the same text and are therefore likely to be part of the same connection chain. However, the method fails when the connections are encrypted. To address the problem [31] proposes an algorithm that doesn't rely on traffic content, instead relies on packet sizes, packet intervals, etc. to identify stepping stones.

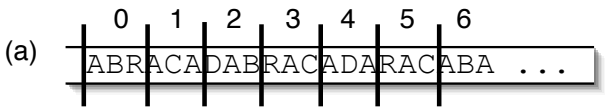
3. HIERARCHICAL BLOOM FILTERS

In this section we introduce a data structure, which we call a Hierarchical Bloom Filter (HBF), that can be used for payload attribution. In later sections we describe the design and implementation of a payload attribution system based on an HBF. A naive method to design a payload attribution system that consumes a small amount of storage and also provides some privacy guarantees, is to simply store hashes of payloads instead of the actual payloads. This effectively reduces the amount of data to be stored per packet to about 20 bytes (using SHA1, for example). Using a standard Bloom filter [5,6] with k hash functions, we can further reduce this space at the cost of a small false positive rate as defined by Equation 1. For a specific space usage of m bits, n strings (packets) inserted into the Bloom filter, the optimum value for FP is achieved for $k = \ln 2 \cdot (m/n)$ and $FP \approx 0.6185^{m/n}$. So, for example, storage per packet can be reduced from 20 bytes to 21 bits at a false positive rate of 4.27×10^{-5}. Compared to simple hashes, the only advantage of using standard Bloom filters is the space saving.

Unfortunately the approaches above restrict the queries to the whole payload. Attributing excerpts of payload is more useful and a simple approach to support queries on excerpts is to hash blocks of the payload instead. Indeed we present such a data structure in the next section. Two issues arise however: if the chosen block size is too small we get too many collisions as there are not enough unique patterns, yet too large a block size and there isn't enough granularity to answer queries smaller than a payload. In addition, one needs a mechanism to determine when two blocks appear consecutively in the same payload, or if their presence is merely an artifact of the blocking mechanism. In section 3.2, we propose a data structure that resolves these two issues simultaneously.

3.1 Block-Based Bloom Filter (BBF)

In order to extend support for attributions based on excerpts, we block the payload of each packet into a set of blocks of size s. Each block is then appended its offset in the payload: (content||offset), where content is the block content, $0 \leq \text{offset} \leq q = \lceil p/s \rceil$, and p is the size of the entire payload. It is then hashed and inserted into a standard Bloom filter. We call such a data structure a *block-based Bloom filter (BBF) with offsets*. See Figure 1(a).



(b)

	Alignment	Offset 0	1	2	3	4	5	6 ...
1.	BRA CAD AB	X	X	X	X	X	X	X
2.	BR ACA DAB		YY	X	X	X	X	X
3.	B RAC ADA B		X	YY	YY	X	YX	

Figure 1: (a) Inserting the string "ABRACADABRACABARACABA..." into a BBF with s=3. (b) Querying with "BRACADAB": the various blocks and offset combinations. Each line corresponds to an alignment, the letters in red do not get certified by the BBF; hence each alignment only gives two blocks to check. An X marks a mismatch of a block, and a Y marks a match; in case the first block is a match, the second block is tested, and so on. The YY in line 2, offset 1, is a real match. On the last line, the YY in offset 2 is a double false positive of the BBF, in offset 3 is a real match (both blocks), and in offset 5, the YX means the first block is a match but the second isn't, hence the combination isn't a match.

Given an excerpt x, a query proceeds as follows: since the excerpt may not start exactly on a block boundary, queries

should try all possible offsets (based on maximum packet size) with all possible first blocks (at most $s-1$) of the excerpt. To do this, we slide a window of size s through x and find matching block with confidence level FP as determined by Equation 1, where n is the number of blocks stored in the BBF. As soon as a match is found for the first block, the query can proceed with the next block at the next offset until all blocks are matched. It is also easy to extend the search over multiple packets. In the event that the excerpt x spans multiple packets, we must also check all the prefixes of each block; if a prefix of a block is found in the packet, then the query proceeds with the next block starting exactly where the prefix left off, at the offset 0. A possible query is depicted in Figure 1(b). For each packet of length p, a BBF method requires $(m/n) \times \lceil p/s \rceil$ bits as oppose to only (m/n) bits required by the standard Bloom filter. At the cost of extra storage, a BBF allows us to fine tune the granularity of excerpt attribution by way of the block size in the queries. For example, decreasing the block size s increases the amount of space required but provides better support to excerpt queries by reducing the granularity of the block. Unlike the standard Bloom Filter, a BBF requires $\lceil q/s \rceil \times (loffset - \lceil q/s \rceil)$ queries where $loffset$ is the largest $offset$ in the BBF and q is the length of the excerpt.

Note that if blocks of a given string occur in different packets at the appropriate offsets (an event we call *offset collision*), this method will see the set of substrings as if the whole string had occurred together in a single packet even if it did not. For example, for two packets made of blocks $S_0S_1S_2S_3S_4$ and $S_0S_2S_3S_1S_4$ (note the reordering), BBF would identify the string "S_2S_1" as if it occurred in a single packet when in fact it did not. This ambiguity is a result of inserting string $(S_2\|2)$ from packet (a) and string $(S_1\|3)$ from packet (b) into the BBF. The BBF could not recognize the fact that the strings in fact occurred in two different packets. For a BBF to work properly over multiple packets a unique packet identifier must be associated with each substring (content||offset||packetID). This, however, severely increases the number of queries required for attribution as it is not known a priori which packet contains the query string. Also note that we may have to maintain up to three Bloom filters to answer queries, one for (content), one for (content||offset), and one for (content||offset||packetID). Next we describe a simple technique that decreases the false positive rate due to collisions and fuses all three Bloom 5filters into one.

3.2 Hierarchical Bloom Filter

A Hierarchical Bloom filter (HBF) is simply a collection of BBFs for geometrically increasing block sizes. For instance, we may choose powers of two times the block size. A string is inserted into the hierarchy from bottom-up. A string of length p is broken into $\lceil p/s \rceil$ blocks which are inserted into the HBF at level 0. At the next level, two subsequent blocks are concatenated and inserted into the HBF at level 1 and so on. Figure 2 illustrates a simple example of such a hierarchy. In this example, string "$S_0S_1S_2S_3$" is blocked into blocks of size $(s = 1)$ at the bottom of the hierarchy. Then "S_0S_1" and "S_2S_3" are inserted at level 1, and "$S_0S_1S_2S_3$" at level 2. Thus, even if substrings have occurred at the appropriate offsets, going one level up in the hierarchy allows us to verify whether the substrings occurred together in the same or different packets.

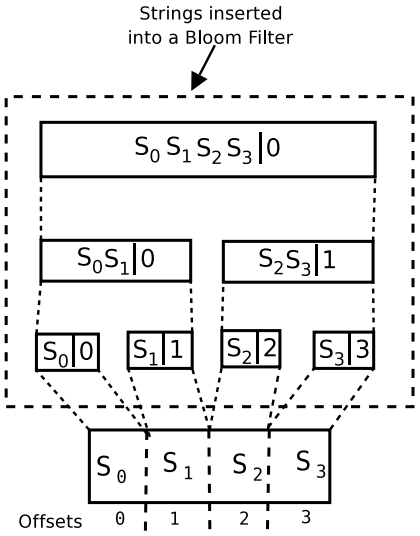


Figure 2: Inserting string "$S_0S_1S_2S_3$" into a Hierarchical Bloom Filter.

Aggregating results from multiple queries within a particular level and from the levels above in the hierarchy we can improve the confidence of the result and reduce the impact of collisions. Verifying an excerpt of length q on an HBF requires $\sum(\lceil q/s^i \rceil \times (loffset/2^i - \lceil q/s^i \rceil))$ queries where $loffset$ is the largest $offset$ in the HBF and i is the level at which the excerpt is being evaluated hence $0 \leq i \leq \log \lceil q/s \rceil$.

Note, however, that not all strings may be of exact length to fit in the hierarchy. For example, with d levels in the hierarchy, if $\left(p \gg \left(s \times 2^d\right)\right)$ we may not be able to store the entire string in the hierarchy as it is not deep enough. On the other hand, if for many packets $\left(p \ll \left(s \times 2^d\right)\right)$ then Bloom filters higher in the hierarchy will be underutilized. Implementing the hierarchy using a single Bloom filter with the offset of each element concatenated to it during insertion, like (content‖offset), improves the space utilization. For example, in order to store string "$S_0S_1S_2S_3$" in the hierarchy, we need to insert the following strings into the Bloom filter $\{(S_0S_1S_2S_3\|0), (S_0S_1\|0), (S_2S_3\|1), (S_0\|0), (S_1\|1), (S_2\|2), (S_3\|3)\}$. Having a single Bloom filter allows us to maximize its space utilization as we can determine the optimal number of elements inserted into it a priori.

It is intuitively clear that, HBF allows us to process excerpt queries with a higher accuracy than a BBF. Note that HBF does not rely on a packet identifier to resolve the offset collisions. The hierarchical nature of the HBF resolves collisions automatically. Furthermore, HBFs can also do limited pattern matching. Suppose we would like to verify if we have actually seen a string of the form "$S_0S_1 * S_3$". As in BBF, the string is broken down into three individual query strings $\{S_0, S_1, S_3\}$. By trying all possible offsets at the bottom of the hierarchy we can verify the existence of strings $\{(S_0\|i), (S_1\|i+1), (S_3\|i+3)$ with false positive rate FP. Since 'S_0' and 'S_1' are subsequent in the query string we can improve the confidence of the results by verifying query string $(S_0S_1\|i)$ at the level above. Now if we can make intelligent guesses for '$*$' and when a match S_x is found, we can verify the match at different levels of the hierarchy.

For example, we can verify the whole string "$S_0S_1S_xS_3$" all the way to the top of the hierarchy consequently improving the confidence of the result at each level.

4. PAYLOAD ATTRIBUTION VIA HBF

Although an HBF can be used for any string matching application, here we focus on its application to the payload attribution problem. In this section we describe the payload attribution problem, discuss the challenges faced in building a reliable payload attribution system, and how to adapt an HBF for such a system. Before we proceed, we would like to make a note on terminology. For the sake of brevity, in the rest of the paper, unless specified otherwise, we use the term payload when we actually mean some arbitrary excerpt from the payload.

Payload Attribution. Given a payload, a payload attribution system reduces the uncertainty that we have about the actual source and destination(s) of the payload, within a given target time interval. The more this uncertainty can be reduced, the better the attribution system. More specifically, let $S = \{s_1, s_2, \ldots, s_m\}$ be the set of all possible hosts that could have originated a payload and let $D = \{d_1, d_2, \ldots, d_m\}$ be the set of all possible hosts that could have received it. Now, given a payload P and a time interval (t_i, t_j), a *source attribution system* $\mathcal{S}$, returns a candidate subset X of S such that any element not in this subset is definitely not the source. Any element in the subset that is not an actual source is a false positive. We can define a destination attribution system in a similar manner and also a full attribution system which reduces the uncertainty of both source and destination. Ideally a PAS would have no uncertainty in its association. However, any practical design of a PAS faces some serious challenges that need to be overcome.

4.1 Design Challenges of PAS

An implementation of a payload attribution system has two main components: a payload processing component and a query processing component. In payload processing, a payload is examined, transformed in some manner and transferred to a permanent storage device. Depending on the application, it can process every single packet it sees or it can selectively process packets. In the query processing component, a query is received, appropriate data is retrieved from storage, interpreted and attributions are sent back. Each component presents its own set of challenges. For example, during payload processing, the system must process packets at line-speed and store the results to a much slower permanent storage device. During the query phase, the system must be aware of and comply with accuracy and privacy requirements set forth by the security policies that govern a network. Although the following do not represent any hard-and-fast design rules, we believe a reliable payload attribution system should at least exhibit these basic properties:

1. *Succinct Representation of Payload:* Storing raw payload presents three major hurdles. First, it requires a lot of memory in the network component itself. Second, transferring raw network data to permanent storage creates a bottleneck due to slower speeds of current storage devices. Third, longevity of stored data depends on the capacity of the storage device. Capacity of storage devices is still a limiting factor for storing raw network data for a prolonged period of time. In order to overcome these hurdles, payloads must be represented in a succinct form.
2. *Efficient Utilization of Resources:* Processing of payload should ideally be done at line-speed so that it does not create any bottlenecks. Moreover, fast memory is a scarce resource therefore processing should utilize the memory efficiently.
3. *Robustness Against Transformations:* Packets can go through two types of transformations: (1) network-induced transformations (2) malicious transformations. Packet encapsulation, and fragmentation are examples of network-induced transformations. Although such transformations are rare [19], a payload attribution system must handle them and NAT translations consistently such that results are not impacted negatively. Malicious transformation, on the other hand, is a more serious threat to the reliability of the system [23].
4. *Accuracy of Results:* Depending on the data structures and algorithms used to store payload and answer queries, results from the system may at best be probabilistic. Therefore, the system must be able to quantify the confidence in its results and preferably send it along with the results. Acceptable accuracy of attribution depends on how the results from the system are used. For instance, to be used in a court of law we would like to have the highest level of accuracy possible whereas to be used for network troubleshooting we might not require such a high level of accuracy.
5. *Privacy:* Obviously, examining and storing payload raises many privacy concerns. Therefore, a payload attribution system must have proper mechanisms in place to guarantee the privacy of users in the network where it is deployed. Also, proper authorization mechanisms must be in place to ensure information is disclosed only to authorized parties. Necessary precautions must also be taken to minimize the exposure of information in the event system itself is compromised.
6. *Compatibility with Existing Systems:* Although a payload attribution system can function independent of any traceback mechanisms, from a pragmatic point of view it is useful if the system can complement many proposed traceback systems. (See Section 4.3 for details.)

4.2 Adapting HBF for Payload Attribution

In this section we describe in detail a payload attribution system that we have designed and implemented, using an HBF that meets some of the design challenges.

Adapting HBF. Note that the construction of an HBF described in Section 3 can only verify whether a string queried was seen by the HBF or not. However, if we would like to attribute an excerpt to a host then payloads must be tied to a particular host (or a pair of hosts). This is accomplished by inserting an additional substring of the form (content||offset||hostID) for each block inserted into HBF, where $hostID$ could be a string that identifies the host that originated or received the payload. For most practical purposes $hostID$ can simply be $(SourceIP, DestinationIP)$. During attribution if the source and destination hosts are not known or if there is any uncertainty about them (See Section 4.3) then the attribution system needs a list of candidate $hostID$s from which it can choose a set of possible attri-

butions. For this purpose, a list of $(SourceIP, DestinationIP)$ can either be maintained by the PAS itself or be obtained from connection records maintained by firewalls, intrusion detection systems or hosts themselves.

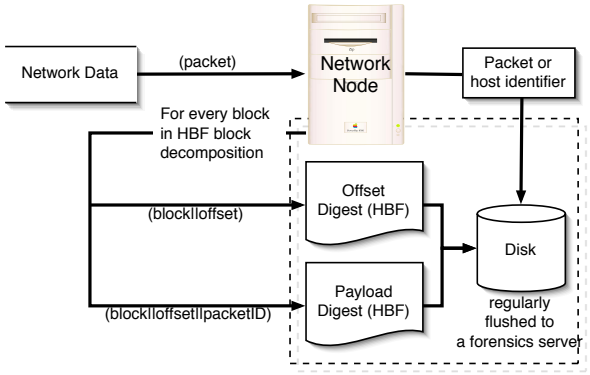


Figure 3: A high level view of the system, with emphasis on packet processing and HBF. The optional block digest is not represented.

As depicted in Figure 3, the system is organized in several tiers. The system sniffs the network and produces the blocks in the hierarchical block decomposition of the packet payload. For every such block, the information (content, offset, $hostID$) is available, and the system maintains:

1. a **block digest** (optional): a HBF storing the hashes of blocks of payload, (content).
2. an **offset digest**: a HBF storing the hashes of content concatenated with its offset in the payload, (content‖offset).
3. a **payload digest**: a HBF storing, for every block (content‖offset) in the offset digest, the corresponding (content‖offset‖hostID).

One may use a BBF instead of an HBF but HBF leads to fewer false positives. The main advantage of using a block digest is to have better accuracy answering whether a block has been seen at all (without knowing the offset). Without it, one must query the offset digest with all possible offsets: although the extra space afforded by not having a block digest increases the accuracy of the offset digest, the testing of every offset gives both designs roughly equivalent accuracy (see Section 5.1). So, we can omit the block digest and save storage to increase the accuracy of the offset digest. Nevertheless, if there are lots of queries for small excerpts, it may be beneficial to keep a block digest.

Payload Processing. Based on network load, required accuracy and granularity of attribution, FP_o, block size, and time-interval to flush an HBF to disk are determined a priori. When deployed PAS maintains an HBF of offset digests and payload digests. It may also maintain a list of $hostID$s if necessary. Upon the predetermined time-interval, the HBF and the list of $hostID$s for the interval are flushed to disk. Our implementation of PAS maintains a list of $hostID$s of the form $(SourceIP, DestinationIP)$ for each HBF.

Query Processing. Now given an excerpt and a time interval, the PAS first retrieves the HBF's and list of $hostID$s that fall within the target time interval from the disk. Then we would first like to verify whether the excerpt was seen by the HBF. In order to achieve this we need to try all possible sliding windows and offsets (as in Figure 1). For each possible alignment, simply block the excerpt and verify if all the blocks are present in the HBF. If any of the blocks cannot be found, then the query string has not been seen by the HBF. If every single block is verified by the HBF, then we need to make sure they appear in the same order as in the query string. To verify the order, we append all possible offsets to the strings (content‖offset) and verify their positions. Based on their offset we may be able to go to a higher level in the HBF hierarchy and increase the confidence as described earlier. Now, in order to attribute the query string we simply append the $hostID$s from the list being maintained by our PAS for the particular HBF being queried and verify the (content‖offset‖hostID). Figure 4 depicts how a query is processed in such a setup.

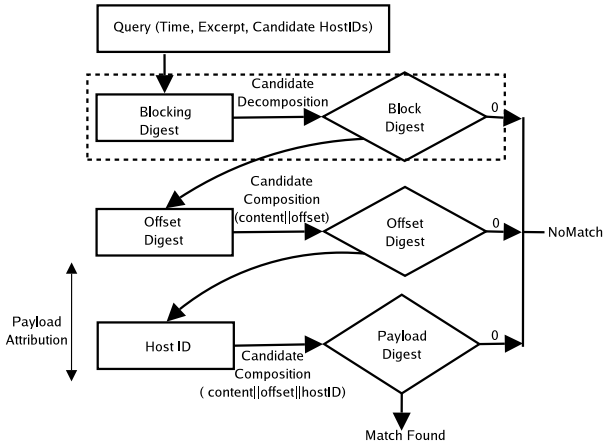


Figure 4: Query Processing in HBF. The block digest will filter out some of the alignments, but it can be omitted (as proposed in the text). In that case, all alignments and offset combinations are passed directly to the offset digest.

4.3 Deployment Challenges of PAS

Now the obvious question is where and how do we deploy a PAS in a network? Ideally, deploying the system at a traffic concentration point of a network would be enough. However, source IP addresses can be spoofed which means the PAS will not be able to attribute an excerpt to its rightful owner. Note, however, PAS is concerned with attributing payload. This can be anything from a mundane web session to a single-packet UDP worm. Thanks to randomization of TCP initial sequence numbers it is difficult to spoof a useful TCP session when the hosts are in different broadcast domains. Only blind-attacks can succeed by spoofing. A good example would be a worm like Slammer which uses single UDP packet to propagate blindly across networks. Keeping this in mind we can divide the attribution process into four different scenarios. What follows is a discussion of various deployment strategies that accommodates spoofing on the Internet and applications of PAS.

Destination Attribution: In this case we would like to use a PAS to attribute an excerpt to its destination. Since it is not possible (or more precisely, useless) to spoof a destination IP address, deploying a PAS at a traffic concentration

point in the local network is sufficient. Viruses and worms spread so fast they often infect most of the network before an administrator can obtain enough information to contain them. With the help of PAS the administrator can look at past network traffic to identify hosts that have received a particular worm and quarantine them. In this scenario, we only need to maintain a list of destination addresses within our networks. In addition, destination attribution is not affected by spoofing in anyway.

Local Source Attribution: We can also use a PAS to attribute an excerpt to a local source. Since source addresses can easily be spoofed, in order to make the attribution reliable PAS has to be deployed in the local network on every subnet. This would help us narrow down the source to its subnet. Also note that the HBF parameters (space vs. accuracy) can be tuned to optimize space utilization in this hierarchical deployment. For example, a PAS at the edge of the network which sees lot of traffic can maintain HBFs with coarse parameters to save space whereas the ones in the subnets can have more accurate HBFs. Local source attributions are useful to prove or disprove an alleged policy violation like, uploading a file that contains trade secrets.

Foreign Source Attribution: Foreign source attribution is when we use a PAS to attribute an excerpt to a source beyond the network boundary where PAS is installed. Deploying PAS at the core of a network to make source IP reliable is impractical. We can, however, use some of the source attribution techniques proposed for flooding attacks at the core. For example, SPIE can be used to trace an excerpt to a foreign source if a PAS maintains MD5 hashes of "non-mutable headers" of packets instead of ($SourceIP$, $DestinationIP$). When the trace reaches the edge of the network this MD5 hashes can then be used by SPIE to trace the excerpt to its actual source through the core of the network. Effectiveness of this method relies on the ability of SPIE to keep the MD5 hashes for a prolonged period of time. Another method, is to use a packet marking scheme proposed in [14,15] which does not require any storage at the core. When using this method a PAS (or a firewall for that matter) can simply replace the source address of a packet with the IP address encoded in the packet's Fragment-Id field. Then, the source can be traced to the closest router that employs this address encoding scheme. In both cases, note that PAS only needs to be deployed in the traffic concentration point of the destination network. Again, such an elaborate setup is needed only when we need to traceback an excerpt involved in a blind-attack, like the Slammer worm. Excerpts of most connection oriented sessions can be attributed to its source just by using the PAS at the destination network.

Full Attribution: For the sake of completeness, full attribution is simply a combination destination attribution and one of the source attributions. Therefore, deployment strategy is a superset of the ones discussed above.

4.4 Attacks, Evasions, & Possible Defenses

Now let us look at how an adversary can evade the system and how we can mitigate the effects. In general, attacks on PAS can be grouped into four major categories:

Malicious Transformations: As discussed in the Design Challenges malicious transformation is one of the biggest threats to any system processing payloads. An attacker, for example, can choose a rather low TCP segment sizes (MSS) to send packets smaller than the block size chosen for the HBF. PAS will not be able to perform attributions because the payloads are smaller than the block size. One of the solutions to this problem is to make the PAS stateful so that it reassembles packets to a minimum size, say 128-bytes, prior to processing. Another approach is to use value-based hashing [24]. In this method, the block size depends on the content as determined by a finger printing algorithm such as Rabbin Fingerprints as oppose to a fixed block size. The dynamic block sizes makes the evasion harder as the attacker not only needs to break up the payload into smaller blocks but also need to beat the odds of triggering the finger printing algorithm. Both of these methods incur additional memory and processing penalties.

Stuffing: An attacker may stuff a packet with no-ops and escape characters such that the application view of the payload is different from that of the network layer. If the excerpt is obtained from the application layer PAS would not be able to attribute the excerpt as the network view is different from the application view. In comparison to simply hashing the whole payload, HBF is more robust against stuffing because with HBFs, an adversary has to affect multiple blocks (32-bytes, in our experiments) to instead of just the end or beginning of the packet.

Exploiting Collisions: There are two types of collisions involved in HBF: hash collisions in Bloom filter and offset collisions in blocks. An attacker may exploit the collisions of hash functions used by the PAS to create false attributions. Using strong hash functions, such as MD5, can resist hash collision attacks at the expense of performance. Besides, choosing a random seed for each HBF created can help resist the attack further. The offset collisions can be easily solved if the query string is long enough. Since the whole packet is also digested by the HBF, if the query string overlaps the colliding blocks the HBF can easily detect the collisions.

Traffic Injection: An adversary may send a specially-crafted packet with incriminating content to frame an innocent host. For example, an adversary could spoof a packet with incriminating data. Although a valid session may not have been established by the packet and the host did not really receive any data, PAS would indicate otherwise. Such attacks are possible because PAS is not aware of the underlying protocol context and can be fooled. In such situations we need to rely on corroborating sources, such as firewall connection records, to verify the context in which the events took place.

Denial of Service: Much like any system that monitors a network our PAS is also vulnerable to flooding attacks. However, the system would withstand more flooding than a traditional packet logger (for the same amount of storage) as there is considerable data reduction thanks to HBF. However, PAS suffers from denial of service attack as an attacker can overflow the list of host IDs used for full attribution.

5. EXPERIMENTS & RESULTS

In this section we discuss some experiments to evaluate the effectiveness of the prototype PAS we have implemented. As noted before, HBF and BBF are constructed on top of a standard Bloom filter. Decomposing the payloads into blocks gives us the ability to query excerpts and also improves the effective false positive rate. In the rest of the paper we refer to the false positive rate of the standard Bloom

filter upon which our extensions are built as *basic false positive rate* (FP_o) and refer to the false positive rate resulting from our extensions as *effective false positive rate* (FP_e). An analysis of the relationship between the false positive rates can be found in [7], where we show that HBF is always better than BBF. In the experiments, we will confirm this and show that $FP_e \ll FP_o$ for reasonably large excerpts.

We first evaluate the effective false positive rate of the HBF (FP_e) so that we can determine the appropriate operational parameters to the system, such as the basic false positive rate (FP_o), block size, minimum length of query string, storage and processing requirements. Finally, we experiment with a real network incident by tracing the propagation of a mass-mailing virus.

5.1 Experimental Evaluation of the Effective FP rate FP_e

In this section we test the effective false positive rates of HBF and BBF. For this purpose we used a packet trace of all email-related traffic (IMAP, SMTP, and POP3) between pairs of $1,500$ hosts for a 24-hour period. The trace was about 1.5GB and contains approximately 3.3 million packets and is large enough to estimate the quantities involved. Also, the results are independent from the nature of the traffic. The trace was digested with HBFs whose block size was 32 bytes, for varying base false positive rates (FP_o). We manufactured queries by taking excerpts of actual payloads and perturbing everything but the first 6 bytes of the query. We removed any duplicates and made sure the queries did not represent strings inserted into the HBFs. Although an HBF is capable of producing partial matches for a given query, in order to clearly quantify the results of our experiments a query string was considered "matched" if and only if all blocks queried by the HBF for the string returned true. For this experiment every such match contributed to FP_e. Table 1 lists the FP_e rates of the HBF determined using $100,000$ queries each, for various lengths and FP_o.

The first thing that one notices is the extremely good FP_e, even with a FP_o as low as 0.2370. If FP_o is set to a moderate value of 0.1090 then a query that is at least four blocks long has an effective false positive rate of only 2×10^{-5}. For an HBF block size of 32 bytes, this means that if we can capture at least 192 bytes, then we are guaranteed (no matter how they are aligned) to have at least 4 blocks in the query and thus an effective false positive rate of less than 2×10^{-5}. As we can see the effective false positive rate depends on (1) the FP_o and (2) the length of the query string. The FP_o decreases the effective false positive rate as each query made on the Bloom filter would return more accurate answer. Queries that are at least 3 blocks long traverse the hierarchy of the HBF. At each level of the hierarchy, queries performed at lower levels are evaluated again. Such a repeated evaluation eliminates the false positives found in the lower levels and contributing to better effective false positive rates in HBFs.

It should be noted that excerpts of length exactly one block result in higher effective false positive rates than the basic false positive rate. This is due to the fact that the our HBF implementation does not insert block content without the offset (i.e just (content)). Therefore, to find a single block the HBF has to try the excerpt with all possible offsets.

In order to evaluate the benefits of the hierarchical structure we did measure the effective false positive rate of BBF and HBF with identical memory footprint: given that HBF stores about twice as many blocks as BBF for the same block size, we took $FP' = \sqrt{FP}$ for both FP_o and FP_p, where FP' is the basic FP rate of the HBF, and FP that of the BBF. Table 2 lists the measured false positive rates of the digests under various sizes of queries. HBF has a clear advantage over BBF of identical memory footprint in almost all cases, as is suggested by the analysis (included in the Appendix).

5.2 Resource Requirements

The following table lists the storage size of HBFs (including $hostID$s in the form of a list of unique (source, destination) IP address pairs) of various basic false positive rates and block sizes for the trace. With a moderate FP_o of 0.109 and a block size of 128 bytes we achieve a 136 : 1 data reduction by using an HBF while the effective false positive rate is as low as 2×10^{-5} (for query string of 512 bytes). Such a block size is good enough to trace worms and viruses on the Internet. Even if we reduce the block size to 32 bytes we get a reduction of 39:1. Even comparing to the compressed email trace (at 684MB) we achieve an order of magnitude reduction in storage space. To further minimize storage of the archive contents, a natural strategy is to compress the filters. The randomized nature of Bloom filters makes them difficult to compress. By populating the bit-vector sparsely (by choosing $k \ll m/n$), however, it would possible to compress the Bloom filters better and at the same time improve the false positive rates [21] at the cost of more high-speed memory at the network component.

5.3 An Actual Case: Tracking MyDoom

We now describe an actual case where we used HBF to track the propagation of mass-mailing virus MyDoom [9] in a large network with thousands of hosts and multiple mail-servers. The attributions obtained as a result can be used for containing the virus within our network and from spreading to other networks. As noted before, with PAS we can find instances of viruses that infected the hosts before signature information was ever available to intrusion detection systems or virus scanners. The PAS was deployed at the traffic concentration point of the network and was setup to monitor all email related traffic (POP, IMAP, SMTP) in and out of the network. Although we were not aware of MyDoom at the time, we were also collecting the network traffic for the experiments in the previous sections from the same vantage point. It was fortunate that the events coincided. The raw packet trace was used to determine the "actual attribution rate" of MyDoom.

Payload Processing. Using the effective false positive rates in Table 1 we chose 0.1090 as the FP_o of Bloom filter on which the HBF is built. Our PAS implementation uses MD5 as the hash function for the Bloom filter. Each MD5 operation yields 4 32-bit integers and two of these are used to achieve the required FP_o. Using the email traffic statistics of the network we concluded that on average $70,000$ blocks will be inserted into an HBF every minute. For the chosen false positive rate of the Bloom filter we need to commit 5 bits per block ($m = 5$) and the optimal number of elements the filter can contain is $70,000$ ($n = 70,000$) which in turn translates to a filter of size 43.75KB (i.e., $n \times m = 43.75$KB). HBF is flushed to disk when the filter is full ($70,000$ blocks are inserted) or 60 seconds have elapsed, whichever comes

	Basic False Positive Rates (FP_o)							
Blocks	0.3930	0.2370	0.1550	0.1090	0.0804	0.0618	0.0489	0.0397
1	1.000000	0.999885	0.996099	0.976179	0.933179	0.870477	0.798657	0.728207
2	0.063758	0.064569	0.048981	0.036060	0.026212	0.021024	0.015881	0.012538
3	0.012081	0.002620	0.000744	0.000275	0.000172	0.000046	0.000023	–
4	0.000820	0.000230	0.000060	0.000020	–	–	–	–
> 4	–	–	–	–	–	–	–	–

Table 1: Measured effective false positive rate (FP_e) of an HBF as a function of both the basic false positive rate (FP_o) and the length of the query (in blocks; 1block=32 bytes). Note that for $blocks > 4$, we encountered no false positives, hence the measured FP_e is equal to 0 (indicated by –).

Query Blocks	2	3	4	5
BBF	0.049621	0.035129	0.000560	0.000088
HBF	0.016457	0.000720	0.000110	0.0

Table 2: Performance comparison of a BBF and an HBF with the same memory footprint. (Query strings of size > 5 resulted in 0 measured false positives for both BBF and HBF, hence are not listed.)

first. The PAS also maintained a list of $hostID$s of the form $(SourceIP, DestinationIP)$ per HBF so that the system does not rely on other sources for candidate $hostID$s. During the experiments, we noted on average each HBF had about 260 $hostID$s. In summary, the PAS was run on a 3GHz Pentium4 machine with $1GB$ of RAM. The average incoming rate of email traffic was 1MB/minute and the average HBF output, including $hostID$s, was 46KB/minute. On average, inserting a packet into the HBF took $28.6\mu s$ including the MD5.

Query Processing. Given the HBFs of the email traffic we now set to look for the presence of the MyDoom virus. To query the HBF we need three parameters, namely: an excerpt, time interval, and candidate $hostID$s. Each copy of the virus comes with a 22KB attachment part of which can be used as the excerpt. Note, however, at the network layer the attachment is MIME-encoded so we MIME-encoded one of the attachments and used the first 96-bytes to 256-bytes as the signatures of MyDoom. In email parlance, these signatures are two to seven line-long excerpts. Time interval and $hostID$ were left open in which case the query processor tries to attribute the excerpt using *all* available data over *all* $hostID$s. For this particular use case, the query processor used data observed over a five day period (more precisely, 138 hours and 13 minutes) over 136,631 unique $hostID$s. For the sole goal of quantifying the accuracy, we used the raw packet trace and the actual attribution rate was obtained by grepping the raw packet trace with `ngrep` for the virus signature. In actual deployment of the PAS, the raw packet trace is of course not needed.

Discussion. Figure 5 shows the number of MyDoom instances given full attribution every hour over the five day period whereas Figure 6 zooms in on a 24-hour period. As we can see from the figures, the actual number of attributions forms the lower bound and the attributions using the smallest excerpt (96-byte signature) forms the upper bound. Figure 6 clearly illustrates how increasing the length of excerpts reduces number of false attributions. When we use a 256-byte excerpt the number of attributions converges to that of actual attributions. More precisely, using the 256-byte signature correctly found all the 25328 actual attributions observed during the five day period–hosts that received at least one copy of the virus–along with 33 incorrect attributions– hosts that did not receive the virus but was identified as if they did because of false positives. The following table lists the number of incorrect attributions found for the whole five days for various lengths of the excerpts.

Length	96	128	160	192	224	256
Incorrect	1375	932	695	500	293	33

Table 4: For a total number of 25328 actual attributions of MyDoom over the five day period, the table lists number of incorrect attributions for varying lengths of excerpts used for querying the PAS.

In conclusion, the system as deployed was quite effective in finding all the instances of the virus in the email traffic during a five day period, with acceptable false positive rate and no false negatives. With the help of PAS attributions we were also able to obtain the following facts about the hosts that were infected by the virus. Over the five day period 679 unique source addresses originated at least a copy of the virus, of which only 52 machines were from our networks and rest of the machines were outside our networks. Of 52 local machines 24 of them sent more than 50 copies of the virus (not including 4 known mail-servers). Furthermore, we found one particular host still live in our network which sent out more than 5000 copies of the virus! The sources inside our network sent copies of the virus to 2011 unique IP addresses outside our network of which 74 got more than 50 copies of the virus. These statistics would have helped a network administrator modify the security policies to abate the severity of the infection. For example, network administrator could have blocked the traffic from infected machines. Although we were lucky in collecting the packet traffic during that time period, we only learned about the virus afterwards, and still had all the data needed for our analysis. This is the main advantage of PAS as opposed to intrusion detection, where we would only have been able to gather the portion of the data after the virus was identified and its signature isolated.

Finally, for most practical purposes PAS can be deployed just at a traffic concentration point of a network. Large enterprises and broadband service providers, for example, can use PAS to monitor network traffic and identify victims

	Basic False Positive Rates (FP_o)							
Block Sizes	**0.3930**	**0.2370**	**0.1550**	**0.1090**	**0.0804**	**0.0618**	**0.0489**	**0.0397**
32	19.42	19.48	28.32	38.62	47.00	56.30	65.65	74.95
64	9.88	9.88	15.40	19.48	25.47	30.48	33.85	38.62
128	6.00	6.03	8.91	11.80	14.69	17.55	18.77	21.39
256	4.10	4.10	6.73	7.95	9.28	11.07	12.05	13.72

Table 3: Storage requirements (in MB) of HBF (including the $hostID$ {source, destination IP}) for varying basic false positive rates (FP_o) and block sizes.

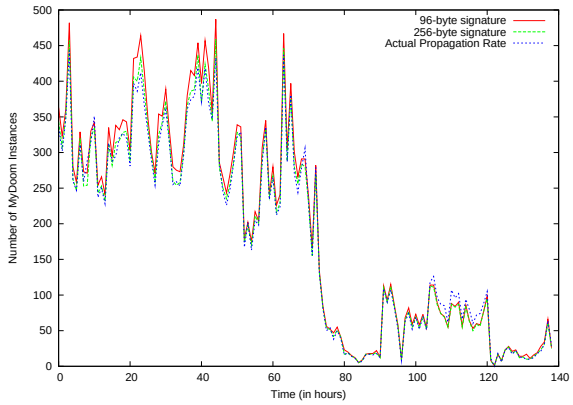


Figure 5: Number of MyDoom attributions in the monitored network for five days.

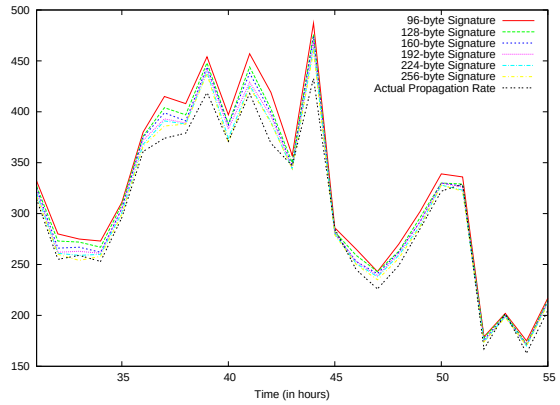


Figure 6: Zooming in on a 24-hour period in Figure 5 with more signatures.

of worms and viruses. For such uses PAS does not rely on any IP traceback mechanisms and the list of $hostID$s that needs to be maintained is much less than what is required for full attribution.

6. CONCLUSION & FUTURE WORK

In this paper, we introduce the problem of payload attribution in a network. Although we focused on IP networks, our ideas apply to other types of networks as well. Unlike previous systems, the proposed system is able to work with arbitrary fragments or excerpts of the payload. Our contention is that in many situations a payload based attribution system is more useful as we often do not have the complete network packets of interest but only an excerpt thereof. For example, we may only possess a code fragment of a virus, as shown in the use case, or the knowledge that a particular file was transferred over the network but no idea when, where and how. In order to construct a payload attribution system that works with excerpts, we propose a novel packet digesting mechanism, namely, a Hierarchical Bloom filter (HBF). We show both by analysis and experimentation that HBFs yield a performance superior to that of a simpler block-based strategy, that involves blocking the payload and inserting blocks along with their offsets into a Bloom filter. Furthermore, our experimental results with actual network data give reasonable effective false positive (FP_e) rates for reasonably long queries. Essentially, our results show that if a query is longer than four blocks, then the FP_e obtained is much less than the basic Bloom filter FP_o. This is because an HBF consolidates many queries to the underlying Bloom filter, and combined together these queries make up a precise answer to the payload query. Our experimental results also indicate that our system is practical. We observe an order of magnitude reduction in data compared to raw network traffic. Privacy is achieved by one-way hashes in the Bloom filters therefore even if the system itself is compromised no raw data is ever exposed.

The system we have described is part of a larger system for facilitating network forensics over wide area networks [26]. The system we have implemented monitors network traffic, creates hash-based digests of payload, and archives them periodically. A user-friendly query mechanism provides the interface to answer postmortem questions about the payload. There are several interesting problems to address as part of our future work. How do we handle compressed data or encrypted data on the network? How can the system be more robust towards various attacks proposed in the paper? What are the advantages of amalgamating such a PAS with current IDS systems [17, 22, 2]? Will the ability of PAS to remember payloads of prolonged past increase the effectiveness of intrusion detection systems which currently work with a much narrower field of vision? We hope to answer these questions in future.

7. ACKNOWLEDGMENT

We thank Vern Paxson, Adrian Perrig, and the anonymous reviewers for many helpful suggestions. This work was partially supported by a NSA/DoD capacity building grant.

8. REFERENCES

[1] Infinistream. http://www.networkgeneral.com/.

[2] Snort. `http://www.snort.org/`.

[3] C.J. Antonelli, M. Undy, and P. Honeyman. The packet vault: Secure storage of network data. Santa Clara, April 1999. Proc. USENIX Workshop on Intrusion Detection and Network Monitoring.

[4] S. M. Bellovin, M. Leech, and T. Taylor. ICMP traceback messages. In *Internet Draft draft-ietf-itrace-01.txt (Work in progress)*. IETF, Oct 2001.

[5] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. In *CACM*, pages 422–426, 1970.

[6] A. Broder and M. Mitzenmatcher. Network applications of bloom filters: A survey. In *Annual Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, Illinois, USA, October 2002.

[7] H. Brönnimann, K. Shanmugasundaram, and N. Memon. String matching on the internet. In *Workshop on Combinatorial and Algorithmic Aspects of Networking*, Banf, Canada, August 2004.

[8] H. Burch and B. Cheswick. Tracing anonymous packets to their approximate source. In *Proc. USENIX LISA*, Dec 2000.

[9] CERT. CERT incident note in-2004-1. `http://www.cert.org/incident_notes/IN-2004-01.html`.

[10] D. Dean, M. Franklin, and A. Stubblefield. An algebraic approach to IP traceback. In *Proceedings of NDSS*, Feb 2001.

[11] S. Dharmapurikar, M. Attig, and J. Lockwood. Design and implementation of a string matching system for network intrusion detection using FPGA-based bloom filters. Technical Report, CSE Dept, Washington University, 2004. Saint Louis, MO.

[12] Sandstorm Enterprises. NetIntercept. `http://www.sandstorm.com/`.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proceedings of ACM SIGCOMM'98*, 1998.

[14] I. Hamadeh and G. Kesidis. Packet marking for traceback of illegal content distribution. In *Proceedings of International Conference on Cross-Media Service Delivery (CMSD)*, Santorini, Greece, May 2003.

[15] I. Hamadeh and G. Kesidis. Performance of ip address fragmentation strategies for ddos traceback. In *Proceedings of IEEE IPCOM*, Kansas City, October 2003.

[16] Abhishek Kumar, Jun Xu, Jia Wang, Oliver Spatschek, and Li Li. Space-code bloom filter for efficient per-flow traffic measurement. In *Proceedings of IEEE INFOCOM*, Hong Kong, China, March 2004.

[17] S. Kumar and E. H. Spafford. An application of pattern matching in intrusion detection. Purdue University Technical Report CSD-TR-94-013, 1994.

[18] A. Mankin, D. Massey, C. L. Wu, S. F. Wu, and L. Zhang. On design and evaluation of "intention-driven" ICMP traceback. In *Proc. IEEE International Conference on Computer Communications and Networks*, Oct 2001.

[19] S. McCreary and K. Claffy. Trends in wide area ip traffic patterns: A view from ames internet exchange. In *ITC Specialist Seminar on IP Traffic Modelling, Measurement, and Management*, March 2000.

[20] A. Mitchell and G. Vigna. MNEMOSYNE: Designing and implementing network short-term memory. In *International Conference on Engineering of Complex Computer Systems*. IEEE, Dec 2002.

[21] M. Mitzenmacher. Compressed bloom filters. In *Proceedings of the 20th Annual ACM Symposium on Principles of Distributed Computing*, pages 144–150, 2001.

[22] V. Paxson. Bro: A system for detecting network intruders in real-time. 7th Annual USENIX Security Symposium, January 1998.

[23] T. H. Ptacek and T. N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Secure Networks, Inc., January 1998.

[24] Sean C. Rhea, Kevin Liang, and Eric Brewer. Value-based web caching. In *Proceedings of the twelfth international conference on World Wide Web*, pages 619–628. ACM Press, 2003.

[25] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for IP traceback. In *Proceedings of the 2000 ACM SIGCOMM Conference*, pages 295–306, Stockholm, Sweden, Aug 2000.

[26] K. Shanmugasundaram, A. Savant, H. Brönnimann, and N. Memon. Fornet: A distributed forensics network. In *The Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security*, St. Petersburg, Russia, October 2003.

[27] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent, and W. T. Strayer. Hash-based IP traceback. In *ACM SIGCOMM*, San Diego, California, USA, August 2001.

[28] D. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *IEEE Infocomm*, 2001.

[29] Eugene H. Spafford. OPUS: Preventing weak password choices. In *Computers & Security*, pages 273–278, May 1992.

[30] S. Staniford-Chen and L.T. Heberlein. Holding intruders accountable on the internet. Oakland, 1995. Proceedings of the 1995 IEEE Symposium on Security and Privacy.

[31] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, Denver, Colorado, USA, August 2000.