Resource-Aware Multi-Format Network Security Data Storage

Evan Cooke, Andrew Myrick, David Rusek, Farnam Jahanian
Department of Electrical Engineering and Computer Science
University of Michigan
{emcooke, andrewmy, rusekd, farnam}@umich.edu

ABSTRACT

Internet security systems like intrusion detection and intrusion prevention systems are based on a simple input-output principle: they receive a high-bandwidth stream of input data and produce summaries of suspicious events. This simple model has serious drawbacks, including the inability to attach context to security alerts, a lack of detailed historical information for anomaly detection baselines, and a lack of detailed forensics information. Together these problems highlight a need for fine-grained security data in the short-term, and coarse-grained security data in the long-term. To address these limitations we propose *resource-aware multi-format security data storage*. Our approach is to develop an architecture for recording different granularities of security data simultaneously. To explore this idea we present a novel framework for analyzing security data as a spectrum of information and a set of algorithms for collecting and storing multi-format data. We construct a prototype system and deploy it on darknets at academic, Fortune 100 enterprise, and ISP networks. We demonstrate how a hybrid algorithm that provides guarantees on time and space satisfies the short and long-term goals across a four month deployment period and during a series of large-scale denial of service attacks.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations

General Terms

Measurement, Security, Darknet

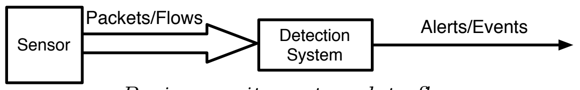
Keywords

Anomaly Detection, Anomaly Classification, Network-Wide Traffic Analysis

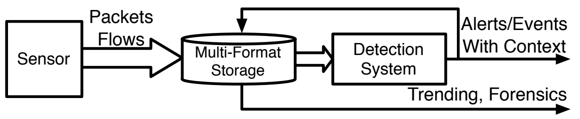
1. INTRODUCTION

The amount of malicious activity on the Internet has grown dramatically over the past few years. Some indicators of this alarming trend include the stream of critical patches for major operating systems and applications, frequent zero-day exploits, and widespread DDoS extortion. To counter the threat, enterprises, governments, and users have deployed detection, monitoring, and prevention systems like IDS's, IPS's, anti-virus programs, and firewalls.

These systems typically operate on a simple input-output principle. They receive a high-bandwidth stream of input data and produce high-level summaries of suspicious events. For example, an IDS such as Snort [10] or Bro [8] observes packets on a network link and produces high-level alerts based on violations of static or behavioral signatures.



Basic security system data-flow



Multi-format security system data-flow

This simple input-output model has serious limitations. First, the high-level alerts generated by security systems like IDS's lack context. That is, alerts typically provide information about the specific vulnerability exploited, but no information about the actions that preceded or followed the exploit (e.g., Did the attacker perform any reconnaissance before attacking the system? What did the attacker do to the compromised system after the exploit?). Another serious limitation with systems that only store data abstractions is that some detection systems require a long-term baseline to perform anomaly detection. Without fine-grained historical information it is much harder to predict the future with past information. Finally, event reports are often inadequate for detailed computer forensics work. More information is often required to track an intruder through the network, and recent regulations regarding data retention have established guidelines for storing this kind information over long periods.

Together these problems highlight two critical limitations with current network security systems, a lack of low-level, fine-grained security data in the short-term, and a lack of high-level, coarse-grained security data in the long-term. To address these limitations we propose a new model of network security data storage: *resource-aware multi-format security data storage*. The idea is to leverage the decreasing cost of storage to record different abstractions of the same input data stream. Thus, an IDS might store every packet it observes for a few days, every flow it observes for a few months, and events and aggregates for a few years. In this way the system provides complete information in the short-term, detailed information in the medium-term, and high-level information in the long-term.

Modifying existing network security systems to take advantage of this new data storage approach requires two basic steps. First, systems must choose which of many possible data abstractions to record. For example, should a system record full packets, network flows, counters, coarse-grained events, or alerts? The second modification step is to develop a storage allocation system. Newer network security data is generally more useful than older network security data, so old data is typically discarded to make room for new data (i.e., a drop-tail approach). A multi-format system must extend the idea to allocate finite storage resources between multiple streams of input data. For example, if a system records packet data and flow data, how does that system decide how much of a finite storage pool to allocate to packet data and how much to allocate to flow data? Said another way, when storage resources are exhausted, a decision must be made about what data format to delete to make room for new data.

We approach the first problem by presenting a novel framework for analyzing security data as a spectrum of information content and storage requirements. We use this analysis to choose distinct points along the spectrum and to design a prototype implementation. We approach the second problem by proposing two methods for capturing multi-format data and three algorithms for partitioning storage resources between the different formats. We describe the *dynamic transformation* and *concurrent capture* approaches for collecting multi-format data and the *fixed-storage*, the *fixed-time*, and the *hybrid* algorithms for allocating storage resources. These algorithms are based around two important metrics: *time* and *space*; that is, the time between the first sample and last sample for each data format, and the number of bytes of storage each data format requires.

We construct a prototype multi-format data storage system based on these ideas and deploy the system on three diverse networks during the first four months of 2006. The deployments are located in a large academic network, inside the border of a Fortune 100 enterprise network, and in a regional ISP network. We present a preliminary evaluation based on these deployments and on tests of the system under simulated denial of service attacks. The idea is to evaluate how the system performs in a real-world setting and under a highly stressful condition.

We show that while no algorithm is perfect, the hybrid algorithm with the concurrent capture system appears to make the best set of tradeoffs. This combination satisfies the short-term and long-term goals while also guaranteeing some amount of data in all formats for detection systems during intensive attacks. Finally, we conclude with a discussion of the approach and future research directions such as adding a predictive capability.

2. BACKGROUND AND RELATED WORK

Network-based intrusion detection and prevention systems are now common on most academic, enterprise, and government networks. These systems analyze streams of packets or flows to identify anomalous behavior or suspicious activity using signatures. However, monitoring high data-rate network streams can be extremely resource intensive. Although storage and computational costs have dropped precipitously, archiving and processing fine-grained information on every packet on the network for long periods is currently impractical. For example, a campus router in our academic networks observes an average of 300Mb/s of traffic. If we were to record every packet for a year that would require about 1.1 petabytes of storage. Trying to store and process this volume of traffic at every detection point in the network would be massively expensive.

To reduce resource costs, existing systems have applied techniques that fall under two broad classes: *sampling*, and *data aggregation*. The sampling approach reduces computational and storage complexity by processing or storing only a subset of the members in a given data stream. The items chosen are kept in the same format as the input set. That is, if a stream of packets is sampled the result will also be a stream of packets. Another approach is data aggregation in which an input data stream is transformed into an output stream with a different format that is typically less storage and computationally expensive. For example, a NetFlow collector takes raw packets as input and produces network flows as output [6]. Research into these two methods of achieving scalability has resulted in important advances:

Sampling: Network measurement at high volume routers and switches for security, billing, and management can be extremely resource intensive. In order to avoid overloading router CPUs, collection systems, and detection systems incoming packets and flows are often sampled [9, 6]. This is typically achieved by processing only 1 in N packets or 1 in N flows. 1 in N sampling can significantly reduce the packet and flow rate, however, critical information and events can be lost due to sampling. For example, the distribution of flow size over time is heavy-tailed leading to an underestimation of total transfer sizes [4].

To achieve better reliability and capture more fine-grained information, several intelligent sampling approaches have been proposed. Duffield *et al.* proposed a proportional smart sampling approach and an architecture for collecting sampled flow data inside a large ISP network [3]. Estan *et al.* proposed a new adaptive sampling approach that limits router memory and CPU requirements by adaptively changing the sampling rate based on the observed traffic [5].

Data Aggregation: A second major approach to achieving scalability is to store specific events or summaries of raw data. These summaries can include fine-grained information like timestamps, source and destination addresses, or more coarse-grained information like the severity of the event and even possible mitigation strategies. For example, NetFlow is fine-grained summarization of packet-level data and IDS/IPS events are coarse-grained summarization of signature matches or behavioral abnormalities. The key idea is that scalability is achieved by using semantic knowledge of lower-level data formats to generate higher-level abstractions of the same data.

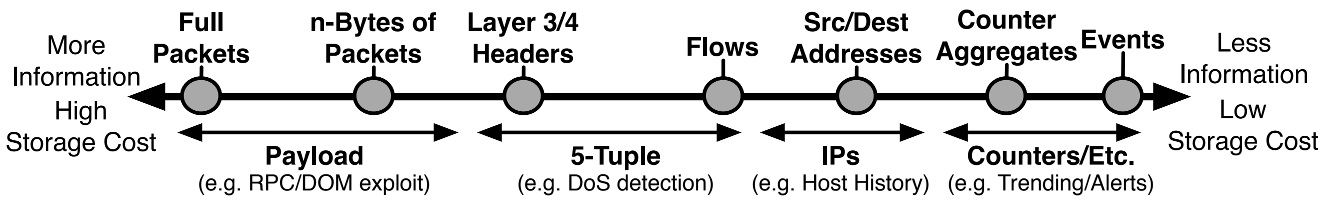

More Information High Storage Cost
Full Packets
n-Bytes of Packets
Layer 3/4 Headers
Flows
Src/Dest Addresses
Counter Aggregates
Events
Less Information Low Storage Cost
Payload (e.g. RPC/DOM exploit)
5-Tuple (e.g. DoS detection)
IPs (e.g. Host History)
Counters/Etc. (e.g. Trending/Alerts)

Figure 1: Network security data abstractions spectrum.

Sampling and data aggregation are complementary techniques and many detection systems use both methods to achieve scalability. However, detection systems today typically use and produce only one data abstraction. For example, an IDS might take full packets as an input and produce events as an output. Similarly, a DoS detection system might take NetFlow as input and produce events as output. This means forensics investigators are limited to the information provided in an event or a single data abstraction to pursue their investigation.

3. MULTI-FORMAT STORAGE

To provide fine-grained information in the short-term and coarse-grained information in the long-term, we propose *resource-aware multi-format storage*. Our approach is to develop a technique for scalably recording different granularities of security data simultaneously. To accomplish the goal of keeping both short-term and long-term data, we propose that a multi-format storage system should store security data in many different formats. In this section we explore the range of network security data summarization and abstraction methods and the utility each provides for detection, forensics, and trending.

At the lowest level are packets. Packets provide the most complete source of information available to most network security devices. Full packets include complete headers and payloads which enable important network security operations like differentiating specific application-level protocols like HTTP. However, storing full packets is also extremely resource intensive.

To reduce the cost of storing and processing full packets, data can be summarized into different abstractions such as flows or events. The key idea is that each of these summaries trade off information for lower resource cost. We propose that these tradeoffs can be illustrated as a spectrum as shown in Figure 1. Full packets that provide the most information and require the most resources are shown on the left, and event summaries that require the least resources but provide the least information are shown on the right (note that certain systems may produce more detailed event summaries that would be placed closer to left of the spectrum).

The important implication of Figure 1 is that there are many different data abstractions, and while it is hard to quantitatively compare them, each abstraction provides uniquely important information useful for forensics and alerting. Several points along the data abstraction spectrum are particular common today. *Packets* are used as the input to many IPS and IDS systems such as Snort and Bro, *flows* are used as the input to large-scale systems such as DoS detectors, and *events* and alerts are produced by detection and mitigation systems. These three abstractions provide excellent coverage of the complete data abstraction spectrum.

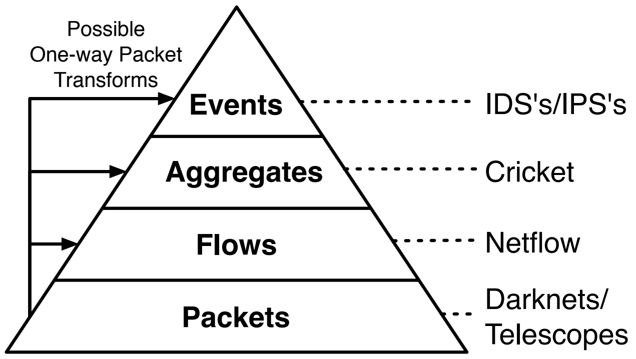


Figure 2: Network security data format hierarchy. One-way data transforms can be performed between lower and higher levels in the hierarchy.

The information and storage relationship between the different formats can also be visualized as a pyramid as shown in Figure 2. The key idea is that formats lower in the pyramid can be transformed into formats higher in the pyramid. For example, there is some function that takes packets as input and transforms them into flows. Clearly these transformation functions are one-way, as information is lost in the process e.g., raw packets cannot be accurately reconstructed from flow data.

The implication of this analysis is that there are many points on the data format spectrum that provide unique resource and information tradeoffs and specific detection, forensics, and trending value. Furthermore, there are a set of one-way transformation functions that provide the capability to convert more fine-grained data formats such as packets into more coarse-grained formats such as flows.

4. STORAGE ALLOCATION

The second major component needed to convert existing network security systems into multi-format storage systems is a storage allocation system. Existing network security systems record data at a higher abstraction level and thus do not worry about storage resources. Newer data is generally more useful than older data in network security so old data is discarded to make room for new data (i.e., a drop-tail approach). This approach is inadequate for multi-format data storage. The problem is that a finite storage resource must be allocated to multiple streams of input data. For example, if a system records packet data and flow data, how does that system decide how much of a finite storage pool to allocate to packet data and how much to allocate to flow data. Said another way, when storage resources are exhausted, data from which format should be deleted.

We now present three algorithms for allocating finite storage resources and two methods of capturing the incoming

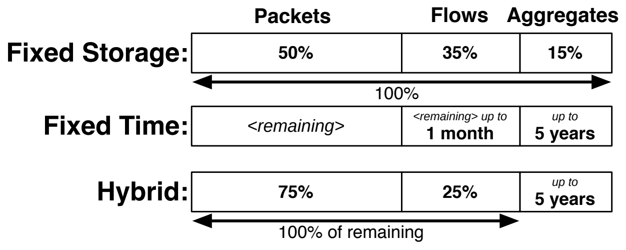


Figure 3: Multi-format storage allocation algorithms.

data in multiple formats. The algorithms are based around two important metrics: *time* and *space*; that is, the time between the first sample and the last sample for each data format, and the number of bytes of storage each data format requires.

4.1 Storage Allocation Algorithms

The development of a storage allocation algorithm requires a method of assigning priority to data formats. When storage resources become scarce, a decision must be made about what lower-priority data to delete. We now present two high-level objectives that we use to help develop priority enforcement algorithms.

The first goal is to guarantee some data will exist over a long period. To keep some a higher level abstractions over months or years. This long-term data is useful for satisfying data retention requirements, trending, and other long-term analysis and characterization. The second goal is to guarantee that detailed data will exist for at least a short period such as a few days or weeks. Highly detailed data provides essential forensic details about the outbreak of new threats and details during an intrusion investigation.

We now describe three algorithms for allocating storage resources based on these two goals: the fixed-storage, fixed-time, and hybrid algorithms (illustrated in Figure 3). To explore these algorithms we model a system that captures packet data, flow data, and counter aggregate data.

4.1.1 Fixed Storage

The fixed-storage algorithm is based on the idea that each data format should be allocated a fixed proportion of the total available storage. For example, a system might allocate 50% of the available space for packet storage, 35% for flow storage, and 15% for counter aggregates. In this way, each data format is independent, and an overflow in one format doesn't impact the allocations of other formats. When the data in a given format exceeds the space available in a partition, the oldest data in that format is deleted so that new data can fit. The problem with this scheme is that there is no way to guarantee how long data in a given format will be available. For example, the partition for network flows might be allocated 35% of the total storage but there is no simple way to know the amount of time that the flow data will cover. That is, how many hours, minutes, days, etc. of flow data will be available for forensics investigations.

4.1.2 Fixed Time

The fixed-time algorithm takes the opposite approach, and provides guarantees on the length of time that a given data format will exist. Each data format is assigned a unique priority and a time range over which the format is guaranteed to cover. For example, counter aggregate data might be assigned the highest priority and the algorithm configured to keep counter aggregates for at most 5 years. Flow data could then be assigned the next highest priority and the algorithm configured to keep flows for at most 1 month. Finally, packet data would be assigned the lowest priority and the algorithm would allocate any storage not used by counter aggregates or flows to packet data. The fixed-time algorithm will then guarantee that if storage is available, there will be 5 years of counter aggregates. Then, if there is still storage left over, there will 1 month of flow data. In this way, network security devices can prioritize certain formats and make a best-effort attempt to store them over long periods of time without the chance that an extremely storage intensive and bursty format like packet data will overwrite them.

The main drawback of this algorithm is that low priority data (like the packet data in our example) can be overwritten easily if the size of higher priority formats suddenly increases. For example, a DoS attack can cause the number of flows to increase to the point that packet data is completely lost (which could hinder subsequent investigations).

4.1.3 Hybrid

The hybrid algorithm attempts to combine the best features of the fixed-storage and fixed-time algorithms. The key idea is to use the fixed-time algorithm when the size of the format can be estimated and the fixed-storage algorithm when it cannot accurately predicted. Thus, the hybrid algorithm can guarantee *some* information will exist for a long period of time and more fine-grained data formats will exist when possible. For example, counter aggregates can be constructed in such a way as to accurately estimate the sample rate and the size of each sample. Thus, counter aggregates can reliably be assigned to cover a fixed time range (without taking all the storage from other formats) and the remaining space partitioned between flows and packets. Using this scheme we can guarantee coarse-grained data will exist for long period and fine-grained will exist for at least a short time.

4.2 Data Capture

As raw data comes into a monitoring system, data abstractions can be generated in one of two ways. First, higher-level abstractions can be generated dynamically as a result of transforms from lower-level abstractions. Second, they can be constructed concurrently with lower-level abstractions. We now detail each approach:

Dynamic Transformation: In this approach, data that enters the system is recorded only in the lowest-level format. As the lower-level format data ages, that data is transformed dynamically into higher-level abstractions. For example, packets are recorded as they enter the system. As that packet data gets older, it is transformed into higher-level abstractions. Those aggregates are subsequently transformed into higher-level abstractions as they age. The advantage of this approach is that storage requirements are reduced because a given time period is stored only in a single data abstraction. The cost of performing the transformation can be distributed across time by pre-transforming and caching the next time unit of data. Dependencies between samples can be reduced by restricting timeouts in protocols like NetFlow to the sample interval size.

Concurrent Capture: This method stores data in multiple data abstractions simultaneously. For example, as packets enter the system, packets, flows, and counter aggregates are generated and recorded simultaneously from the same input packet stream. The advantage of this approach is that recent data is available from different abstractions simultaneously. Therefore, alerting and forensics applications will always have recent data.

5. EVALUATION

In this section we construct a prototype multi-format data storage system and deploy it on three diverse networks. We then evaluate the *dynamic transformation* and *concurrent capture* approaches for collecting multi-format data and the *fixed-storage*, the *fixed-time*, and *hybrid* algorithms for allocating storage resources.

5.1 System Implementation

We constructed a prototype consisting of four daemons: three for capturing data and one for managing storage resources. The system was designed as four separate daemons for reliability, scalability, and flexibility. Each capture process is independent so a failure of one process does not impact another. This redundancy helps ensure data availability under conditions like the outbreak of a new threat or an attack.

The capture daemons include one program for recording packets, one for recording flows, and one for recording counter aggregates. The `pcapture` daemon stores packets, the `nfcapture` daemon stores flows in NetFlow version 9 format, and the `scapture` daemon stores counter aggregates at five different time resolutions. The input to each capture daemon is raw packets or other lower-level data abstractions. This enables the system to support both concurrent capture and dynamic transformation. Finally, `formatmgr`, the storage management daemon, is responsible for enforcing the different storage management algorithms and transforming and purging old data files.

pcapture: The `pcapture` daemon reads packets from a network interface using the libpcap packet capture library and stores each complete packet with link-level headers. `pcapture` automatically rotates files each hour in order to keep individual files from getting too large. `pcapture` output files are compatible with tcpdump and other packet inspection tools.

nfcapture: The `nfcapture` daemon reads packets from a network interface or `pcapture` file using the libpcap packet capture library and aggregates them into flows uniquely identified by the layer-3/4 connection 5-tuple. Flows are stored according to Cisco's NetFlow version 9 specification which provides a flexible format that allows customized flow storage fields. `nfcapture` stores the: source IP address, source port, destination IP address, destination port, flow start time, flow end time, protocol, total packets, total bytes, and TCP flags in each flow record. Each flow record consumes a total of 27 bytes. Flows are ordered by start time and are written to hourly flow files. The flow output files are compatible with existing flow tools that support NetFlow version 9.

Time-scale	Sample Rate	Samples Per Period	Average Bytes/Second
Hour	4 *secs*	900	24
Day	90 *secs*	960	1.067
Week	10 *mins*	1000	0.16
Month	45 *mins*	960	0.356
Year	9 *hours*	973	0.00296

Table 1: Counter aggregate samplers implemented in `scapture` (96 bytes per sample).

scapture: The `scapture` daemon reads packets or flows and counts the number of observed bytes, packets, and unique source addresses observed. `scapture` builds these aggregates in four bins: over all packets, over all TCP packets, over all UDP packets, and over all ICMP packets. A 64-bit counter is used for each data point, so a complete sample takes 96 bytes. The counter aggregates are meant for quick analysis and trending so `scapture` stores counter aggregates to enable analysis over 5 common time ranges: hourly, daily, weekly, monthly, and yearly. Sample rates at each of these time ranges were chosen to provide approximately 1000 data points. 1000 data points is typically enough to produce high fidelity graphs and to perform basic trending. A summary of the different time scales and the corresponding data rates (at 96 bytes/sample) are shown in Table 1.

formatmgr: The `formatmgr` daemon manages storage resources between multiple data abstractions. In our system it handles allocations for the `pcapture`, `nfcapture`, and `scapture` daemons. `formatmgr` tracks the amount of space used by each daemon and transforms or deletes old data files to free up resources. The `formatmgr` daemon implements the fixed-storage, fixed-time, and hybrid algorithms described in the previous section. The daemon automatically adapts allocations as storage resources are removed and added. Thus, if a new disk is added to the system, the `formatmgr` will detect the disk and increase the amount of space allocated to each storage format according to the partitioning algorithm.

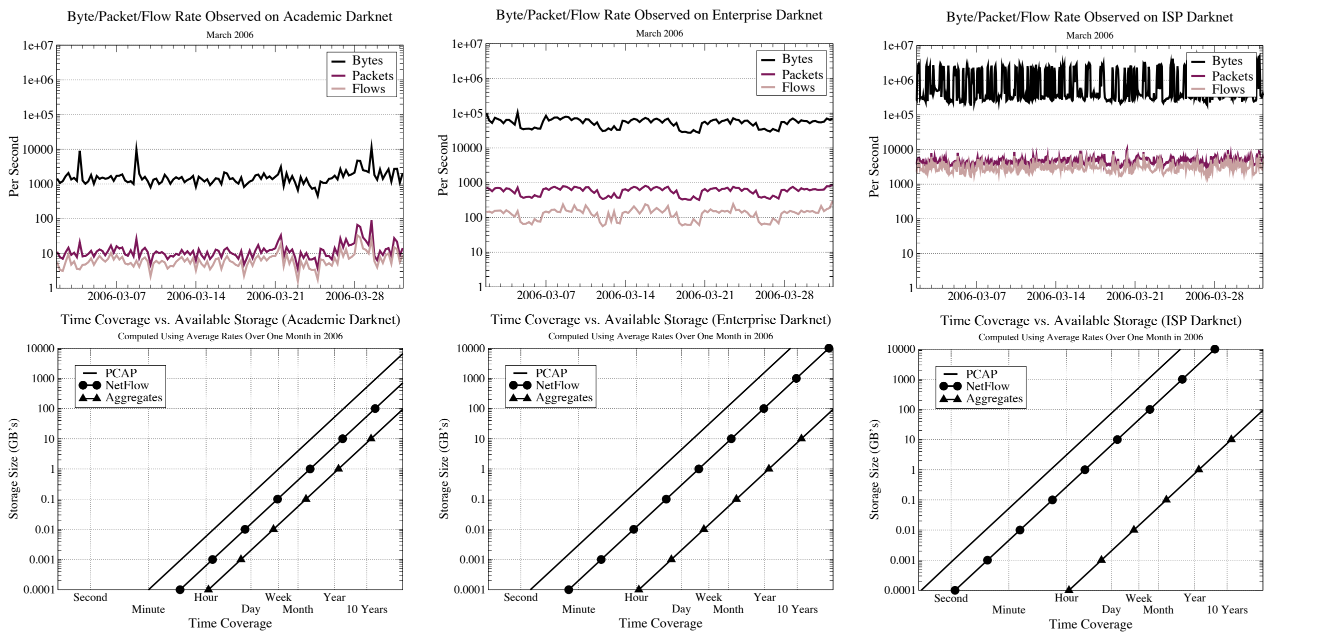
5.2 Deployment and Evaluation Environment

To evaluate the prototype multi-format security data capture system, we deployed it on three large production networks and tested the system under simulated DoS attacks. The idea was to evaluate how the system performed in a real-world setting and under a highly stressful condition.

We deployed the system on three diverse networks during the first four months of 2006. We monitored security data feeds from three Internet Motion Sensor [1] darknet sensors. Darknets monitor traffic to unused and unreachable addresses [7, 2]. The darknets we monitored were passive (i.e., did not actively respond to incoming packets) and were located in a large academic network, inside the border of a Fortune 100 enterprise network, and in a regional ISP network. The darknets covered approximately a /16 network (65 thousand addresses), a /8 network (16 million addresses), and /8 network, respectively.

To provide a baseline for the subsequent analysis we measured the byte, packet, and flow rates at each of the deployments. The top of Figure 4 shows the number of bytes, packets, and flows observed at the three darknet deployments during March 2006. These graphs demonstrate the huge differences in the relative quantities of data in different formats.

The bottom of Figure 4 shows the storage resources required to store each data abstraction using the `pcapture`,



(a) Academic Darknet (b) Enterprise Darknet (c) ISP Darknet

Figure 4: (Top) Overall byte, packet, and flow rates observed at three darknet deployments during March 2006. Note the daily and weekly cyclical behavior at the enterprise darknet. (Bottom) Storage resources required for storing different lengths of time in different with different data abstractions.

nfcapture, and scapture daemons for different periods of time. The packet and flow rates were computed by averaging the traffic rates over March 2006. These graphs demonstrate the vast difference in storage requirements for different security sensors. For example, 100 GB of storage is enough to store more than a year of packet data on the academic /16 darknet, but only enough to store one day of packet data on the larger /8 ISP darknet.

We then used the traffic data that we observed during the live deployment to generate a packet trace simulating a high volume denial of service (DoS) attack. The baseline traffic rate in the trace was fixed at the average traffic rate observed at the /8 ISP darknet over March 2006. Next, a series of five DoS attacks were injected into the packet stream. These attacks increased the byte, packet, and flow rate by ten times over the normal rate. The resulting packet trace was then used to evaluate the dynamic transformation and concurrent capture approaches.

5.3 Dynamic Transformation

In this subsection we evaluate the fixed-storage, fixed-time, and hybrid algorithms using the dynamic transformation storage approach. Recall that the dynamic transformation capture approach translates one data format into another when a time or space allocation becomes full. For example, when the storage allocated to packets becomes full, older packets are automatically transformed into flows.

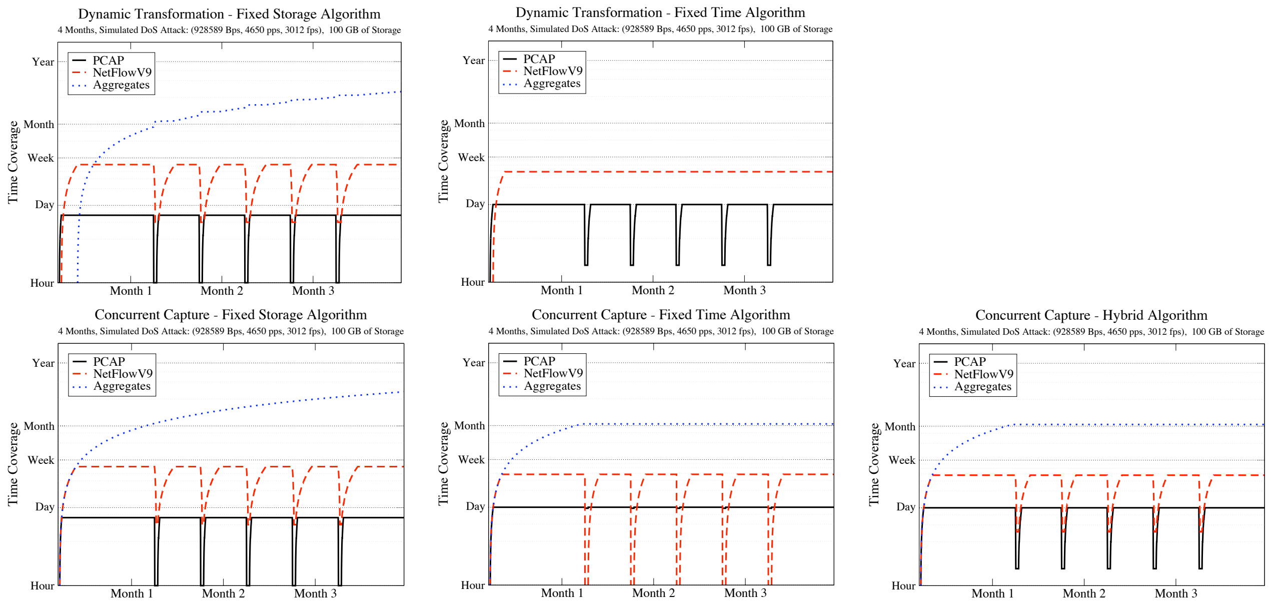
We started by replaying the simulated attack packet trace and capturing it using the dynamic transformation system. The results are shown in the top of Figure 5. Looking first at the top of Figure 5(a), we find that the fixed-storage algorithm was able to successfully keep results as packets, flows, and aggregates. However, notice the dropout in time coverage (the difference between the first and last data timestamp) with the packets and flows corresponding to the five DoS attacks. In addition, notice how the time coverage of the aggregates also spikes with the attacks as data is transformed between formats more quickly. This unpredictability makes this algorithm less desirable.

The top of Figure 5(b) show the results with the fixed-time algorithm. The fixed-time algorithm was configured with data format priorities consistent with our short-term and long-term goals. That is, aggregates (long-term data) were given the highest priority followed by packets (short-term data) followed by flows (medium-term). The most critical feature of the resulting graph is that we see no aggregates. The reason is that packet and flows take all the available space, *starving* the aggregates. This is a critical result because there is always the chance that a lower-priority format can be starved for resources, meaning that we won't record that format. This means we are not able to meet the goal of having fine-grained information in the short-term and coarse-grained over the long-term.

Because the hybrid algorithm is based on the fixed-time algorithm, it also suffers from the same starvation problem when data is dynamically transformed. These limitations mean that there are very weak guarantees on the availablity of more coarse-grained formats such as aggregates. Thus, the dynamic transformation approach does not appear to meet our goal of having fine-grained data in the short-term and coarse-grained data in the long-term.

5.4 Concurrent Capture

In this subsection we analyze the utility of the fixed-storage, fixed-time, and hybrid algorithms when data is captured and stored using the concurrent capture approach. The concurrent capture approach differs from the dynamic transformation approach because all data formats are recorded



(a) Fixed Storage (b) Fixed Time (c) Hybrid

Figure 5: (Top) Performance of the fixed-storage and fixed-time allocation algorithms using the dynamic transformation capture approach under a series of simulated DoS attacks. (Bottom) Performance of the fixed-storage, fixed-time, and hybrid allocation algorithms using the concurrent capture approach under the same attacks.

simultaneously. For example, when a packet enters the system it is recorded as a packet, a flow, and an aggregate simultaneously

To evaluate the concurrent capture approach, we replayed the simulated attack packet trace and recorded it using the concurrent capture approach. The results are shown at the bottom of Figure 5. Looking first at the fixed-storage algorithm, the bottom of Figure 5(a) shows that some of each format was captured with the fixed-storage algorithm. The graph does show significant dropouts in both packet and flows during each of the DoS events. However, because it is a fixed-storage approach, we can guarantee some amount of packet and flow data will be available during the attacks.

The bottom of Figure 5(b) shows the results for the fixed-time algorithm. As with the dynamic transformation approach, the fixed-time algorithm was configured with data format priorities consistent with our short-term and long-term goals. That is, aggregates (long-term data) were given the highest priority followed by packets (short-term data) followed by flows (medium-term). The bottom of Figure 5(b) shows that we were able to provide good guarantees for both aggregates and packets. The only difficulty is that there are periods during the attacks where no flow data is recorded. This is potentially dangerous if a detection system using data from our system operates only on flows. During the period of the attacks the detection system would have no flow data with which to generate alerts.

The hybrid algorithm provides both a long-term guarantee for aggregate data and the guarantee that some packet and flow data will exist. Moreover, because the data rate of the aggregates is fixed (the bit-rate is constant), it does not change when the system comes under attack, and we can guarantee it will not cause storage starvation for other data formats. The bottom of Figure 5(c) demonstrates how the hybrid algorithm with the concurrent capture system is able to provide long-term guarantees on coarse-grained aggregates and guarantee some amount of short-term data in both packet and flow formats. It provides both short-term packet and flow data during each attack so detection systems based on packets or flows can continue operating effectively.

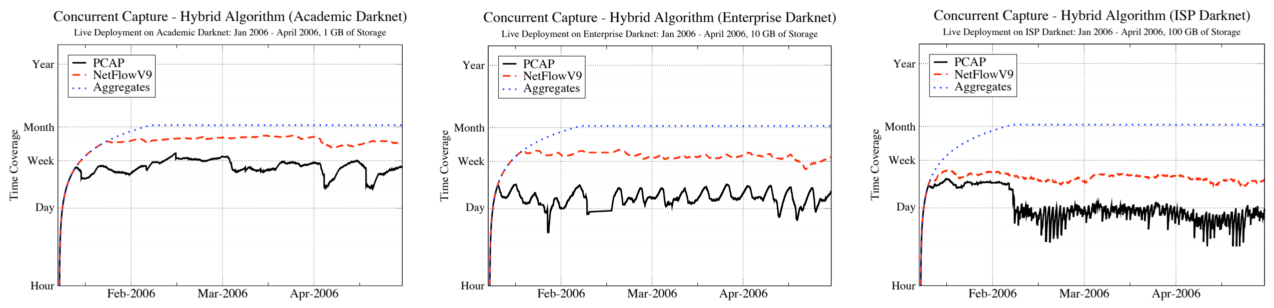
While no algorithm is perfect, the hybrid algorithm with the concurrent capture system appears to make the best set of tradeoffs. This combination satisfies both our short and long-term goals while also guaranteeing some amount of data in all formats during intensive attacks.

5.5 Darknet Results

We deployed the concurrent capture system with the hybrid algorithm on the academic, enterprise, and ISP networks over four months during the beginning of 2006. To evaluate the effectiveness of the system under conditions with limited resources, we fixed the storage pool for the academic darknet at 1GB, the storage pool for the enterprise darknet at 10GB, and the storage pool for the ISP darknet at 100GB. The results are shown in Figure 6.

Figure 6 demonstrates that the proposed multi-format storage system was able to meet our goal of providing fine-grain information in the form of complete packets in the short-term and a guaranteed amount of coarse-grained aggregates in the long-term. However, there was also a high degree of variability in the time coverage due to different events. For example, drop in time coverage during February 2006 in Figure 6(c) is due to the emergence of a large amount of extremely aggressive single-packet UDP Windows popup spam. This variability also means that it is difficult predict the availability a format like packets based on historical information.

Finally, we performed a preliminary evaluation of overall performance of the monitoring system and found that



(a) Academic Darknet (b) Enterprise Darknet (c) ISP Darknet

Figure 6: Deployment results using concurrent data capture and the hybrid allocation algorithm on three darknets networks during the first four months of 2006.

the primary bottleneck was the storage system. Thus, while there is additional overhead to storing multiple formats simultaneously, the amount of resources required to store packet data vastly dominate other formats as shown in Figure 4. Therefore, the overhead of storing multiple formats is directly related to the resource requirements of the most fine-grained data format (in our case packets).

6. DISCUSSION AND FUTURE WORK

We have presented *resource-aware multi-format security data storage*, a framework for archiving fine-grained security data in the short-term, and coarse-grained security data in the long-term. We demonstrated how security data formats can be placed along a spectrum of information content and resource cost. We then proposed three algorithms for collecting and storing multi-format data. We deployed prototype implementation of a multi-format storage system on three darknets in an academic network, a Fortune 100 enterprise network, and an ISP network. Finally, we demonstrated how a hybrid algorithm that provides guarantees on time and space satisfies the short and long-term goals across the four month deployment period and during a series of large scale denial of service attacks.

While the multi-format approach performed well, there are still many open questions. For example, what is the optimal method of configuring the time and size of the different allocations with the fixed-storage, fixed-time, and hybrid algorithms? Understanding what different data formats should be captured, the typical data rate of those formats, and how the stored information will be used are all critical to approaching this problem.

Related to the partition sizing problem is the development of some predictive capability. That is, it would be extremely helpful if previous historical data could be used to predict future data rates. Given the variability in data rates we observed during the evaluation, this a difficult problem and may depend heavily on the type of security data being monitored.

Another important question is how the system scales to more resource intensive applications like high-volume IDS's. It would be very helpful to understand how a multi-format storage system performs with gigabits per second of input traffic. Is the bottleneck recording data to persistent storage or generating data abstractions? An evaluation on a live IDS deploy would help to answer this question.

In all, the multi-format approach is very promising and there are many interesting research questions remaining.

7. ACKNOWLEDGMENTS

This work was supported by the Department of Homeland Security (DHS) under contract number NBCHC040146, and by corporate gifts from Intel Corporation and Cisco Corporation.

8. REFERENCES

[1] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A distributed blackhole monitoring system. In *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[2] E. Cooke, M. Bailey, F. Jahanian, and R. Mortier. The dark oracle: Perspective-aware unused and unreachable address discovery. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, May 2006.

[3] N. Duffield and C. Lund. Predicting Resource Usage and Estimation Accuracy in an IP Flow Measurement Collection Infrastructure. In *ACM SIGCOMM Internet Measurement Conference*, 2003.

[4] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *ACM SIGCOMM Internet Measurement Workshop*, 2001.

[5] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. In *Proc of ACM SIGCOMM*, Portland, Oregon, USA, 2004.

[6] C. S. Inc. Netflow services and applications. `http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.h%tm`, 2002.

[7] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 27–40. ACM Press, 2004.

[8] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[9] S. Phaal, S. Panchen, and N. McKee. RFC 3176: InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. 2001.

[10] M. Roesch. Snort — lightweight intrusion detection for networks. In USENIX, editor, *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII): November 7—12, 1999, Seattle, WA, USA*, Berkeley, CA, USA, 1999. USENIX.