Nfsight: NetFlow-based Network Awareness Tool

Robin Berthier
Coordinated Science Laboratory
Information Trust Institute
University of Illinois
Urbana-Champaign, IL, USA
rgb@illinois.edu

Michel Cukier
The Institute for Systems Research
Clark School of Engineering
University of Maryland
College Park, MD, USA
mcukier@umd.edu

Matti Hiltunen, Dave Kormann, Gregg Vesonder, Dan Sheleheda
AT&T Labs Research
180 Park Ave.,
Florham Park, NJ, USA
{*hiltunen,davek,gtv*}*@research.att.com, dsheleheda@att.com*

Abstract

Network awareness is highly critical for network and security administrators. It enables informed planning and management of network resources, as well as detection and a comprehensive understanding of malicious activity. It requires a set of tools to efficiently collect, process, and represent network data. While many such tools already exist, there is no flexible and practical solution for visualizing network activity at various granularities, and quickly gaining insights about the status of network assets. To address this issue, we developed Nfsight, a NetFlow processing and visualization application designed to offer a comprehensive network awareness solution. Nfsight constructs bidirectional flows out of the unidirectional NetFlow flows and leverages these bidirectional flows to provide client/server identification and intrusion detection capabilities. We present in this paper the internal architecture of Nfsight, the evaluation of the service, and intrusion detection algorithms. We illustrate the contributions of Nfsight through several case studies conducted by security administrators on a large university network.

1 Introduction

Network awareness, i.e., knowledge about how hosts use the network and how network events are related to each other, is of critical importance for anyone in charge of administering a network and keeping it secure [11]. The goal of network awareness is to provide relevant information for decision-making regarding network planning, maintenance, and security. NetFlow is among the most-used information sources for gaining awareness in large networks because it offers a good trade-off between the level of detail provided and scalability. As a result, a majority of networks are already instrumented through their routers to collect and export NetFlow, and a variety of tools are available to process such data [18, 36, 27]. However, there is still no practical solution to visualizing network activity at various granularities and quickly gaining insight about the status of network assets. Numerous attempts have been made [37, 31, 5] and are detailed in Section 4, but none has gained a broad audience.

We developed a tool called *Nfsight* to address these challenges. The objective of Nfsight is to offer a comprehensive network awareness solution through three core functions: 1) passive identification of client and server assets, 2) a web interface to query and visualize network activity, and 3) a heuristic-based intrusion detection and alerting system. Nfsight is designed to be simple, efficient, and highly practical. It consists of three major components: a Service Detector, an Intrusion Detector, and a front-end Visualizer. The Service Detector component analyzes unidirectional NetFlow flows to identify client and server end points using a set of heuristics and a Bayesian inference algorithm. The Intrusion Detector component detects suspicious activity through a set of graphlet-based signatures [13], and the front-end Visualizer allows administrators to query, filter, and visualize network activity. We trained and evaluated the Service Detector using two different datasets of 30 minutes of packet dumps collected at the border of a large university network. The Intrusion Detector was evaluated by security experts over a period of four months. Based on several months of testing in a production environment of 40,000 computers, we believe Nfsight can greatly assist administrators in learning about network activity and managing their assets.

The rest of the paper is organized as follows. In Section 2, we provide an overview of Nfsight and we present the implementation and evaluation of the different components: the Service Detector (Section 2.2), the Intrusion Detector (Section 2.3), and the front-end Visualizer (Section 2.4). We discuss a number of use cases in Section 3 and we compare our approach to related work in Section 4. Finally, Section 5 offers some concluding remarks.

2 Architecture and Implementation

This section provides an overview of the architecture of Nfsight and describes in detail the implementations of the Service Detector, the Intrusion Detector and the front-end Visualizer.

2.1 Nfsight Architecture Overview

The architecture of Nfsight is presented in Figure 1. Nfsight uses non-sampled unidirectional NetFlow provided by a collector such as Nfdump/Nfsen [19]. A network flow is defined as a unidirectional sequence of packets that share source and destination IP addresses, source and destination port numbers, and protocol (e.g., TCP or UDP). A NetFlow flow carries a wide variety of network-related information about a network flow including the timestamp of the first packet received, duration, total number of packets and bytes, input and output interfaces, IP address of the next hop, source and destination IP masks, and cumulative TCP flags in the case of TCP flows.

The Service Detector component takes NetFlow flows and converts them into bidirectional flows in the IPFIX format (bidirectional flow format specified by the IPFIX working group [4]). During this process, it identifies client and server end points using a set of heuristics and a Bayesian inference algorithm. The bidirectional flows, denoted by *IPFIX* in Figure 1, are stored in flat files, while the server end points, denoted by *Assets* in Figure 1, are stored in a MySQL database. The Intrusion Detector component detects suspicious activity through a set of graphlet-based signatures [13] applied on the bidirectional flows. The high-level network activity and event alerts generated by the Intrusion Detector are stored in a MySQL database. An aggregation script runs periodically to maintain a round-robin structure in the database and to provide three aggregation levels: every five minutes, hourly, and daily. We detail the data storage and representation solution of Nfsight in Section 2.4. The front-end Visualizer allows administrators to query, filter, and visualize network activity. They can access the application simply by using a web browser and they can collaborate through a shared knowledge base of events reported either automatically by the Service Detector and Intrusion Detector or manually by operators.

2.2 Passive Service Detection

2.2.1 Definitions

In the rest of the paper we use the following definitions. A *server* is a network application that provides a service by receiving request messages from clients and generating response messages. A server is hosted on a computer identified by its IP address and accepts requests sent to a specific port. In this paper, we focus on servers using the UDP and TCP protocols. We are interested in both transient and permanent servers. Specifically, we consider P2P transactions a part of the client/server model, even if the server in this case may be handling client requests for only a few minutes and for only specific clients. We define an *end point* as a tuple $\{$IP address, IP protocol, Port number$\}$. An end point may represent either a client or a server.

We define a *network session* as a *valid communication* between one client end point and one server end point. All UDP flows are considered to be valid, but TCP flows are valid only if both the request and the reply flows carry at least two packets and the TCP acknowledgement flag. For example, if a server refuses a TCP connection handshake by sending a reset flag to the source end point, the communication is not considered valid. Finally, we use the term *network transaction* to describe any set of flows between two end points during a time window smaller than the maximum age limit of a flow (usually 15 minutes). There are two types of network transactions: unidirectional and bidirectional. We assume that bidirectional transactions are always between a client and a server and that bidirectional transactions are always initiated by a client.

2.2.2 Approach

The task of accurately detecting servers based solely on NetFlow flows is challenging because NetFlow does not keep track of the logic of network sessions between clients and servers. Specifically, we have to address the following challenges:

1. NetFlow may break up a logical flow into multiple separate flows;

2. NetFlow is made of unidirectional flows and therefore we need to identify the matching unidirectional flows to make up bidirectional flows and identify valid network sessions;

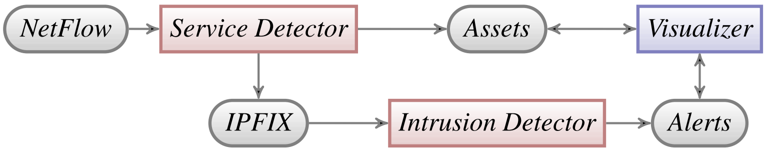


Figure 1: Nfsight architecture

3. Identifying the server end point in a network session is difficult because the TCP flags in the request and reply flows are typically identical for valid bidirectional flows. Furthermore, the flow timestamps have proven to be sometimes unreliable and more often, the request and reply flows have identical timestamps due to the granularity of the timestamps.

We solve the first and second challenges by matching and merging the NetFlow flows as follows. First, for each collection period (usually 5 minutes), we merge all network flows that have the same source and destination end points to eliminate any artificial breaking of unidirectional flows. Then, to address the issue of combining unidirectional flows into network sessions, we generate bidirectional flows by merging all flows collected during a given time window that have opposite source and destination end points. The network sessions are then selected based on the number of packets and flags and according to the definition of valid communication above. The last step is to address the third challenge, i.e., to identify client and server end points for every network session. We describe below the approach we developed to perform this task.

2.2.3 Server Identification Heuristics

To correctly identify client and server end points for every valid bidirectional flow, we developed a set of heuristics that determine if an end point is a server (or not). These heuristics were developed to cover a variety of intuitions gathered from network experts. A heuristic may be based on the attributes of an individual (bidirectional) flow or it may consider a set of flows.

The heuristics implemented are:

H.0 Flow timing. Let t_1 and t_2 be the timestamps of the unidirectional flows constituting a bidirectional flow. The source of the flow with the larger (more recent) timestamp is likely the server. The difference between t_1 and t_2 provides an indication on the probability that this heuristic will identify the correct end point as a server. If the timestamps are identical, they cannot be used to decide which end point is the server.

H.1 Port number. Let p_1 and p_2 be the port numbers associated with a bidirectional flow. The end point with the smaller port number is likely the server. If the port numbers are identical, they cannot be used to decide which end point is the server.

H.2 Port number with threshold at 1024. If an end point has a port number lower than 1024, then it is likely a server. The value of 1024 corresponds to the limit under which ports are considered privileged and designated for well-known services. If both ports are above or below 1024, this heuristic cannot be used to decide which end point is the server.

H.3 Port number advertised in /etc/services. If the port number of an end point is listed in the standard UNIX file /etc/services that compiles assigned port numbers and registered port numbers [12], then it is likely a server. If both or neither port numbers are in /etc/services, this heuristic cannot be used to decide which end point is the server.

H.4 Number of distinct ports related to a given end point. If two or more different port numbers (in different flows) are associated with an end point, the end point is likely a server. The number of different port numbers related to an end point provides an indication on the probability that this heuristic will correctly identify the server. This heuristic comes from the fact that ports on the client-side are often randomly selected. Therefore, ports on the client-side of a connection are less likely to be used in other connections compared to ports on the server-side. If both end points are related to the same number of ports, then this heuristic cannot be used to decide which end point is the server.

H.5 Number of distinct IP addresses related to a given end point. This heuristic is identical to the previous one but counts IP addresses instead of ports.

H.6 Number of distinct tuples related to a given end point. This heuristic is identical to the previous one but counts end points instead of single IP addresses. This heuristic is based on the observation that each server typically has two or more clients that use the service. Furthermore, even if only one real user accesses the service (e.g., identified by the IP address of the user's machine), the communication will likely require multiple connections and the client side of the access often uses different port numbers. Thus, multiple end points will be detected.

2.2.4 Evaluation of Individual Heuristics

We evaluated the accuracy of each heuristic by using bidirectional flows generated by Argus [26] as the ground truth. Argus is a flow processing application that generates bidirectional flows from packet data. We considered Argus to be more accurate than Nfsight, and able to produce a baseline dataset for our evaluation, since it uses detailed packet data as input instead of the high level flow data used by Nfsight. We collected a dataset of 30 minutes of network traffic from the border of a large university network and analyzed the data using Argus to identify bidirectional flows and their server end points. We then processed the data using Nfsight to generate bidirectional flows (6.2 million records) and applied the heuristics to determine the server end points. We define the accuracy of a heuristic as the probability that it correctly identifies the server end point of a bidirectional flow. The accuracy is estimated by dividing the number of bidirectional flows correctly oriented based on ground truth from Argus by the total number of bidirectional flows correctly and incorrectly oriented.

For heuristics H.1, H.2 and H.3 the accuracy probability is a single value. Specifically, based on our input data, we calculated the accuracies of these heuristics to be 0.78, 0.75, and 0.74, respectively. Heuristics H.0, H.4, H.5, and H.6 depend on parameter values, either on time difference or number of distinct ports, IP addresses, or tuples. Therefore, we can evaluate their accuracy with regard to the parameter value as demonstrated in Figures 2 to 5 (up to 10 seconds for H.0, and up to 100 ports, IPs, and tuples for H.4, H.5 and H.6). These plots show that the accuracy increases with the time difference between requests and replies (Figure 2), the number of related ports (Figure 3), IP addresses (Figure 4) and {IP, protocol, port} tuples (Figure 5) between source and destination end points. Note that the similarities between Figures 3 and 5 can be explained by the fact that the client ports are randomly selected among 64,511 values. Therefore, the number of client ports and the number of clients are different only in the case where two clients communicating with the same server select the same source port randomly.

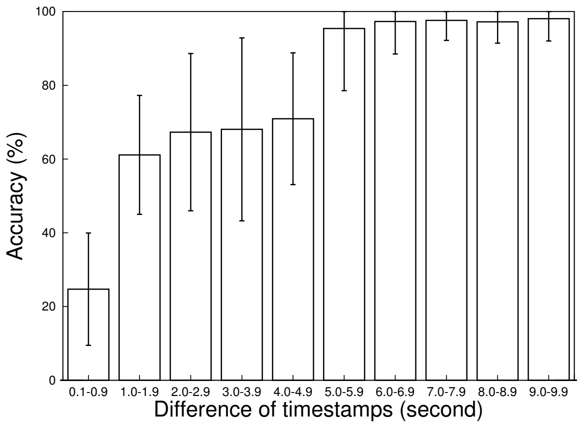


Figure 2: Bidirectional flow orientation accuracy increases with the timestamp difference between request and reply flows (H.0)

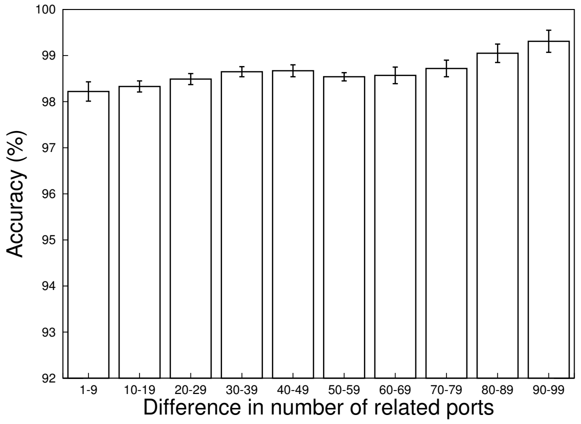


Figure 3: Bidirectional flow orientation accuracy increases with the difference between the number of source and destination related ports (H.4)

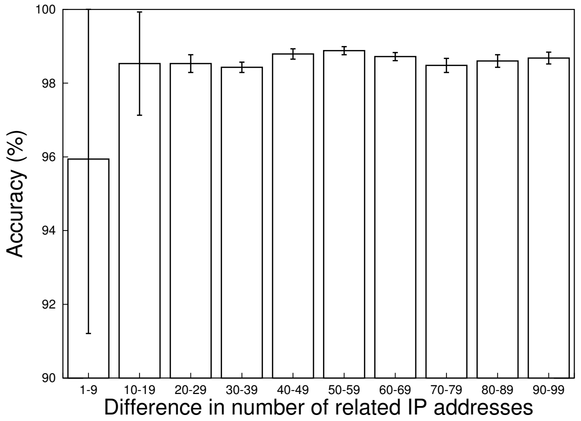


Figure 4: Bidirectional flow orientation accuracy increases with the difference between the number of source and destination related IP addresses (H.5)

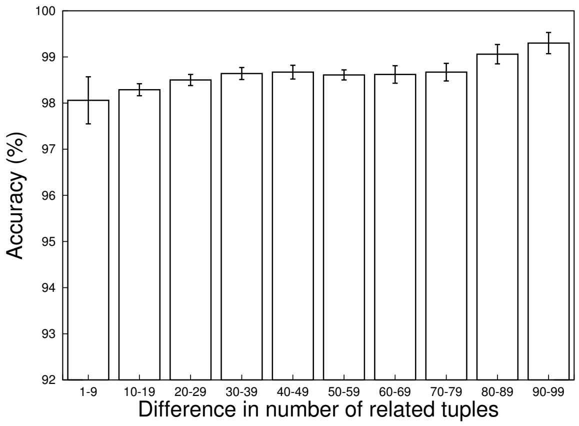


Figure 5: Bidirectional flow orientation accuracy increases with the difference between the number of source and destination related tuples (H.6)

2.2.5 Combining heuristics

While individual heuristics can be used to identify server end point, they cannot make a decision for all the flow processed. For example, some flows have similar request and reply timestamps, or similar source and destination port numbers. To address this issue and to get a better estimate, we combine the evidence provided by the different heuristics using basic Bayesian inference. We consider each end point that is present in at least one bidirectional flow. For each end point X, we have two possible hypotheses:

- H_s: end point X is a server.
- H_c: end point X is a client.

The different heuristics are used to identify evidence E in the bidirectional flows. For example, the fact that there is a difference in unidirectional flow timestamps provides evidence based on heuristic H.0. Bayesian inference combines any prior knowledge (the prior probability of hypothesis H_i being true denoted by $P(H_i)$) with information gained from new evidence E to produce a new estimate of the probability that the hypothesis is true using the formula:

$$P(H_i|E) = \frac{P(E|H_i) * P(H_i)}{\sum P(E|H_j) * P(H_j)}$$

where $P(E|H_i)$ denotes the probability that evidence E is present in a flow or set of flows given that hypothesis H_i is true, that is, that a heuristic we use to generate the evidence is accurate. While these conditional probabilities could be assigned using expert knowledge, we use the heuristic accuracies measured previously. We summarize these empirical results in Table 1.

Table 1: Individual heuristic accuracies used as conditional probabilities for Bayesian inference

Heuristic	*Output*	*Accuracy*
H.0	]0; 1.0[	0.25
	[1.0; 5.0[	0.7
	[5.0; ∞[	0.99
H.1	True	0.78
H.2	True	0.75
H.3	True	0.74
H.4	1	0.97
	[2; 29]	0.9825
	[30; 74]	0.9875
	[75; ∞[	0.99
H.5	1	0.95
	[2; ∞[	0.98
H.6	1	0.97
	[2; 29]	0.9825
	[30; 74]	0.9875
	[75; ∞[	0.99

Note that while the naive Bayesian formulation used assumes independence of evidence, and some of the heuristics are obviously correlated, we find the approach still useful for combining the heuristics. We are evaluating other combining techniques, such as Bayesian networks, that allow explicit representation of dependencies between heuristics.

2.2.6 Evaluation of Bayesian Inference

We evaluated the accuracy of Nfsight to address two related issues: 1) generating correctly oriented bidirectional flows, and 2) accurately identifying server end points. For the first issue, we applied the approach previously described to individually evaluate heuristics by using Argus to provide ground truth. For the second issue, we compared server end points discovered by Nfsight against Pads [23]. Pads is a packet-based passive service discovery tool. Similarly to Argus, we considered Pads to be more accurate than Nfsight and able to produce a baseline dataset for our evaluation, since it works from detailed packet data instead of high level flow data. In our evaluation, we are interested in measuring how much accuracy we lose by working only with flows.

We collected a second dataset of 30 min of network traffic from the border of the same large university network. Note that the dataset used for determining the accuracy of individual heuristics (summarized in Table 1) and the dataset used for this evaluation were collected five months apart.

Concerning the issue of generating correctly oriented bidirectional flow, we analyzed 3,617,077 bidirec-

Table 2: Bidirectional flow orientation accuracy grouped by confidence level from Bayesian inference

Heuristic	*Able to decide*	*Accuracy*
H.0	11.49%	94.54%
H.1	63,98%	85.54%
H.2	48.14%	98.15%
H.3	47.73%	98.17%
H.4	63.28%	93.72%
H.5	55.51%	88.76%
H.6	63.38%	92.58%

tional flows generated by both Nfsight and Argus. On this dataset, Argus could decide on the orientation for 2,356,616 flows (65.15%) while Nfsight could make a decision for 3,616,942 flows (99.996%). When Argus could decide, we evaluated that Nfsight agreed on the orientation for 2,183,440 flows. This represents an accuracy of 92.65%.

To understand further the contribution of the Bayesian inference to combine heuristics, we expand the comparison against Argus for each individual heuristic in Table 2. These results reveal that individual heuristics provide high accuracies but they are able to decide for only a fraction of the flows. For instance, H.0 agrees with Argus for 94.54% of the flows, but could decide for only 11.49% of the flows. The accuracies of H.1 to H.6 range from 85.54% to 98.17%, while the decision capabilities of H.1 to H.6 lie between 47.73% and 64.98%. These results show the importance of the Bayesian inference to combine heuristics, because it allows the overall decision capability to reach almost 100% while keeping the overall accuracy above 92%.

The final step of the evaluation was to address the second issue of accurately identifying server end points. We compared server end points identified by Nfsight and Pads. Out of 57,985 TCP servers detected by Pads from the packet data, Nfsight was able to identify 45,932, which represents an accuracy of 79.21%. We investigated the services detected by Pads and not by Nfsight, and we found that the majority of them were source end points of unidirectional flows. This pattern indicates that our evaluation dataset did not contain both directions of network sessions for some flows. The lack of request or reply flows can come from asymmetric routing or sampling. We discuss in Section 3.4 the need to develop additional heuristics that would allow Nfsight to handle such cases.

2.3 Intrusion Detection

Once bidirectional flows have been generated by the Service Detector, the Intrusion Detector identifies malicious activity using a set of detection rules based on the graphlet detection approach [13]. In this approach, the patterns of host behavior are captured based on the flows, and then these patterns are compared with intrusion detection signatures. Patterns are generated for each host and contain statistical information such as host popularity, number of ports used, number of failed connections, and total number of packets and bytes exchanged. Note that working with bidirectional flows simplifies the definition of the detection rules and the pattern lookup since the source and destination end points of each network transaction are already known. We describe in detail the data structure and the different detection rules we evaluated in the remainder of this section.

2.3.1 Data Structures

The intrusion detection algorithm processes each bidirectional flow generated over the last batch of NetFlow flows collected (5 minutes in our setup) and creates or updates two dictionary structures: one for the source and the other for the destination IP addresses of the flow under review. The structure for source IP addresses captures the fan-out relationships, while the other captures the fan-in relationships. These dictionaries are organized in a three-level hierarchy, where the IP address and the protocol are used as keys for the first and second levels, respectively. The different fields at the third level are therefore all related to a specific {IP, protocol}. These fields are:

- *Peer*: the set of distinct related IP addresses;
- *Port*: the set of distinct related destination or source ports;
- *TCP flag*: the set of distinct flag combinations used;
- *Packet*: the total number of packets sent or received;
- *Byte*: the total number of bytes sent or received;
- *Flows*: the total number of bidirectional flows sent or received;
- *Failed connections*: the total number of unidirectional flows sent or received;
- *Last source end point*: the source port, IP address and TCP flag of the last flow captured;
- *Last destination end point*: the destination port, IP address and TCP flag of the last flow captured.

The last two fields are not used by the detection rules but were requested by our team of administrators as an additional time-saving information when classifying alerts sent by email. For example, consider a case where a host is detected as initiating a large number of failed connections over the last 5 minutes. If the last source port appears to be random and the last destination port is TCP/445, then the host will be immediately classified as compromised by a malware that spreads over the Netbios service. On the other hand, if the last source port is TCP/80 and the last destination port appears to be random, then the host will likely be classified as a victim of a denial-of-service attack.

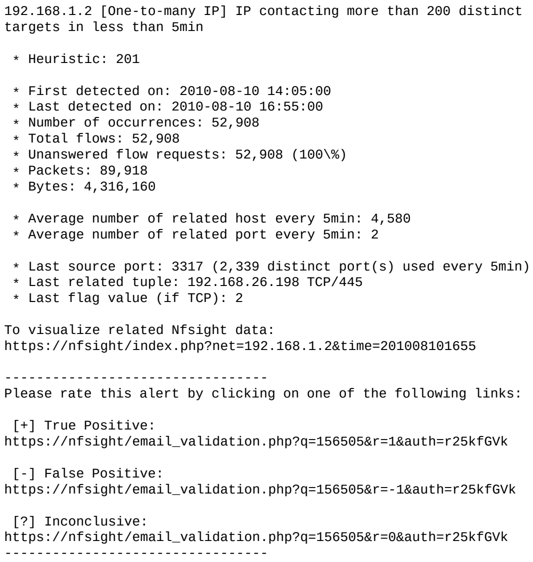
2.3.2 Detection Rules

The next step performed by the intrusion detection algorithm is to process each bidirectional flow again and to try to match flow information and source and destination host patterns against a set of signatures. We created a set of 13 rules organized in 3 categories: malformed flows, one-to-many, and many-to-one relationships. These rules and categories are described in Table 3. They were based on expert knowledge and on a study of attack traces to cover noisy malicious activity such as scanning and denial-of-service activities generated by compromised hosts. We note that Nfsight provides the data structures and rule matching algorithm to enable administrators to create and evaluate more fine-grained rules.

As shown in Table 3, rules in the one-to-many and many-to-one categories use thresholds. We defined these thresholds empirically from a study of attack traces and the feedback we received during the testing of the different detection rules. These thresholds are likely specific to a given network and a given time window of analysis. Thus, they are subject for future tuning. The threshold values used in our experiments were *max_dst_ip* = 200, *max_dst_port* = 250, *max_src_ip* = 500, and *max_src_port* = 500. Rules in the malformed flow category use three data structures to catch incorrectly formed packets. These are: *invalid_code* to detect incorrect ICMP type and code combinations; *invalid_ip* to detect forged or misconfigured IP addresses sent to private or unallocated subnets; and *invalid_flag* to detect incorrect TCP flag combinations.

2.3.3 Evaluation

Flow-based intrusion detection implementations often suffer from two problems: 1) the difficulty to validate and tune anomaly detection rules and 2) the difficulty to access and understand the root cause of the malicious activity detected. The first problem is illustrated in the context of application detection in [14], where the authors observe that the tuning of the 28 configurable threshold parameters of the original graphlet approach [13] is too cumbersome. To simplify rule tuning and validation, we developed an evaluation process using email alerts. The objective is to leverage administrator expertise while minimizing the time and effort required to validate detection rules. Specifically, each alert emailed to security administrators contains three embedded links that allow the alert receiver to rate the alert as true positive, false positive, or unknown. A fourth link allows administrators to open the front-end Visualizer and display the network activity related to the alert under review. An example of an alert email with validation links is given in Figure 6.



```
192.168.1.2 [One-to-many IP] IP contacting more than 200 distinct
targets in less than 5min

 * Heuristic: 201

 * First detected on: 2010-08-10 14:05:00
 * Last detected on: 2010-08-10 16:55:00
 * Number of occurrences: 52,908
 * Total flows: 52,908
 * Unanswered flow requests: 52,908 (100\%)
 * Packets: 89,918
 * Bytes: 4,316,160

 * Average number of related host every 5min: 4,580
 * Average number of related port every 5min: 2

 * Last source port: 3317 (2,339 distinct port(s) used every 5min)
 * Last related tuple: 192.168.26.198 TCP/445
 * Last flag value (if TCP): 2

To visualize related Nfsight data:
https://nfsight/index.php?net=192.168.1.2&time=201008101655

--------------------------------
Please rate this alert by clicking on one of the following links:

 [+] True Positive:
https://nfsight/email_validation.php?q=156505&r=1&auth=r25kfGVk

 [-] False Positive:
https://nfsight/email_validation.php?q=156505&r=-1&auth=r25kfGVk

 [?] Inconclusive:
https://nfsight/email_validation.php?q=156505&r=0&auth=r25kfGVk
--------------------------------
```

Figure 6: Example of an alert email with validation links

The second problem is due to the fact that flows are based on aggregated header information and lack details on the payloads required to precisely identify attack exploits. It is not possible to fully address this problem if we restrict ourselves to Netflow, but we note that the different visualization solutions offered by Nfsight and described in Section 2.4 help to understand and assess the illegitimate nature of suspicious network activity.

We configured the email validation script to send no more than five alert emails in two batches per day to four experts: two security administrators and two graduate students working in network security. Alerts were ranked according to the number of flows and the number of detection occurrences. Then the top five internal IP addresses for which no alerts email had been previously sent were selected. Table 4 presents the validation results collected over a period of four months for the five detection rules that triggered alerts. In this table, *TP* denotes the number of alerts labeled as “true positives”, *FP* denotes the number of alerts labeled as “false positives”, and *Unknown* represents alerts for which experts could not decide if the activity was malicious. The results in-

Table 3: Intrusion detection rules

Id	*Name*	*Category*	*Filter*
101	Identical source and destination	Malformed flow	src_ip = dst_ip
102	Invalid ICMP flow size	Malformed flow	proto = ICMP *and* total_byte $\leq$ 64000
104	Invalid ICMP code	Malformed flow	proto = ICMP *and* icmp_code $\in$ invalid_code
105	Invalid IP address	Malformed flow	(src_ip *or* dst_ip) $\in$ invalid_ip
106	Invalid TCP flag	Malformed flow	proto = TCP *and* flag $\in$ invalid_flag
201	One-to-many IP	One-to-many	failed_connection $\geq$ 1 *and* unique_dst_ip $\geq$ max_dst_ip *and* unique_flag $\leq$ 1
301	One-to-many Port	One-to-many	failed_connection $\geq$ 1 *and* unique_dst_port $\geq$ max_dst_port *and* unique_flag $\leq$ 1
401	Many-to-one IP on TCP flows	Many-to-one	proto = TCP *and* flag $\notin \{19, 27, 30, 31\}$ *and* unique_src_ip $\geq$ max_src_ip *and* unique_flag $\leq$ 1
402	Many-to-one IP on ICMP flows	Many-to-one	proto = ICMP *and* unique_src_ip $\geq$ max_src_ip
403	Many-to-one IP on UDP flows	Many-to-one	proto = UDP *and* unique_src_ip $\geq$ max_src_ip
501	Many-to-one Port on TCP flows	Many-to-one	proto = TCP *and* flag $\notin \{19, 27, 30, 31\}$ *and* unique_src_port $\geq$ max_src_port *and* unique_dst_port = 1 *and* unique_flag = 1
502	Many-to-one Port on ICMP flows	Many-to-one	proto = ICMP *and* unique_src_port $\geq$ max_src_port *and* unique_dst_port = 1
503	Many-to-one Port on UDP flows	Many-to-one	proto = UDP *and* unique_src_port $\geq$ max_src_port *and* unique_dst_port = 1

dicate that rules *105* and *201* are relatively accurate. We note that these two rules allowed our team of administrators to detect 18 internal compromised hosts. However, rules *106*, *301*, and *501* have a high rate of false positives. The poor performance of rule *106* can be explained by the facts that invalid TCP flag combinations may be due to misconfigured hosts or legitimate TCP connections may be broken over different flows. The false positives for rules *301* and *501* are mainly due to heavily used servers for which the thresholds *max_src_ip* and *max_src_port* were too low. The feedback offered by this validation process and the labeled alerts help adjusting the parameters and thresholds of the detection rules. We are working towards implementing an automated process to adjust these values and revise the detection rules.

2.4 Data Visualization

The front-end Visualizer allows administrators to query, filter, and visualize network activity. This section presents the web interface of Nfsight and the underlying data storage solution.

2.4.1 Hybrid Data Storage

Alerts and client/server end points identified by the Service Detector and Intrusion Detector modules are stored in a MySQL database at three aggregation levels: five minutes, hourly, and daily. An aggregation script that expires data at different granularities runs periodically to maintain a round-robin structure in the database. This structure allows the storage of a large volume of data (88 million records organized in 107 tables in our implementation) while offering a fixed database size (11GB in our implementation) and a fast access to network end points at different time granularity levels. We configured the 5-minute granularity data to expire after two weeks.

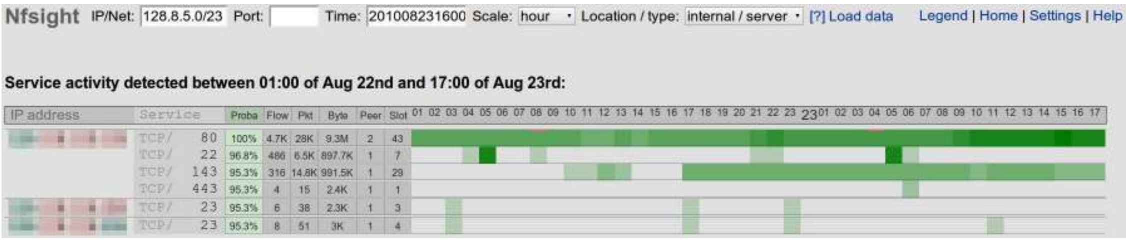
2.4.2 Web Front-end

The front-end is developed in PHP and consists of a search engine, a dashboard, and a network activity visualization table. The dashboard presents the latest generated alerts and the top 20 servers, services, scanned services, and internal scanners. The search form and the network activity visualization table are represented in Figure 7. We note that IP addresses in Figure 7 and in Section 3 have been pixelated on purpose. The search form enables administrators to filter activity per subnet, IP, time period, and type of activity (i.e., internal or external client and/or server).

The visualization table is organized by host IP, port number, and type of activity (either client for source port or server for destination port). For each end point, the tool provides both statistical information and a visualization of the activity over the given time period. The statistical information includes the confidence value given

Table 4: Validation results for each detection rule triggered

Id	*Total Validated*	*TP*	*FP*	*Unknown*	*Accuracy: TP/(FP+TP)*
105	23	11	4	8	73.3%
106	27	3	19	5	13.6%
201	68	40	21	7	65.6%
301	94	30	41	23	42.3%
501	78	21	38	19	35.6%



Nfsight IP/Net: 128.8.5.0/23 Port: Time: 201008231600 Scale: hour Location / type: internal / server [?] Load data Legend | Home | Settings | Help

Service activity detected between 01:00 of Aug 22nd and 17:00 of Aug 23rd:

IP address	Service		Proba	Flow	Pkt	Byte	Peer	Slot
[redacted]	TCP/	80	100%	4.7K	28K	9.3M	2	43
	TCP/	22	96.8%	486	6.5K	897.7K	1	7
	TCP/	143	95.3%	316	14.8K	991.5K	1	29
	TCP/	443	95.3%	4	15	2.4K	1	1
[redacted]	TCP/	23	95.3%	6	38	2.3K	1	3
[redacted]	TCP/	23	95.3%	8	51	3K	1	4

Figure 7: Nfsight front-end Visualizer

by the Bayesian inference algorithm and the number of flows, packets, and bytes. The network activity is represented as a time series using a heat map that visually reveals the number and type of flows detected over the time period. A color code enables network operators to separate client activity (blue) from server activity (green), and also to identify the fraction of invalid, i.e., non-answered (red), flows sent/received by an end point. The intensity of the color is used to represent the number of flows. Some servers may receive both unidirectional and bidirectional flows, represented by a block divided into green and red parts that represent the proportion of unidirectional and bidirectional flows received by the server. These unidirectional flows may be due to invalid packets that the server rejected, an overwhelming number of requests, or unidirectional flows that the Service Detector component failed to pair correctly. Additional examples of the visualization capabilities of Nfsight are provided in Section 3.

3 Use Cases

We present in this section different use cases to demonstrate how Nfsight can help security administrators and network operators in their daily tasks.

3.1 Network Awareness

3.1.1 Server Identification

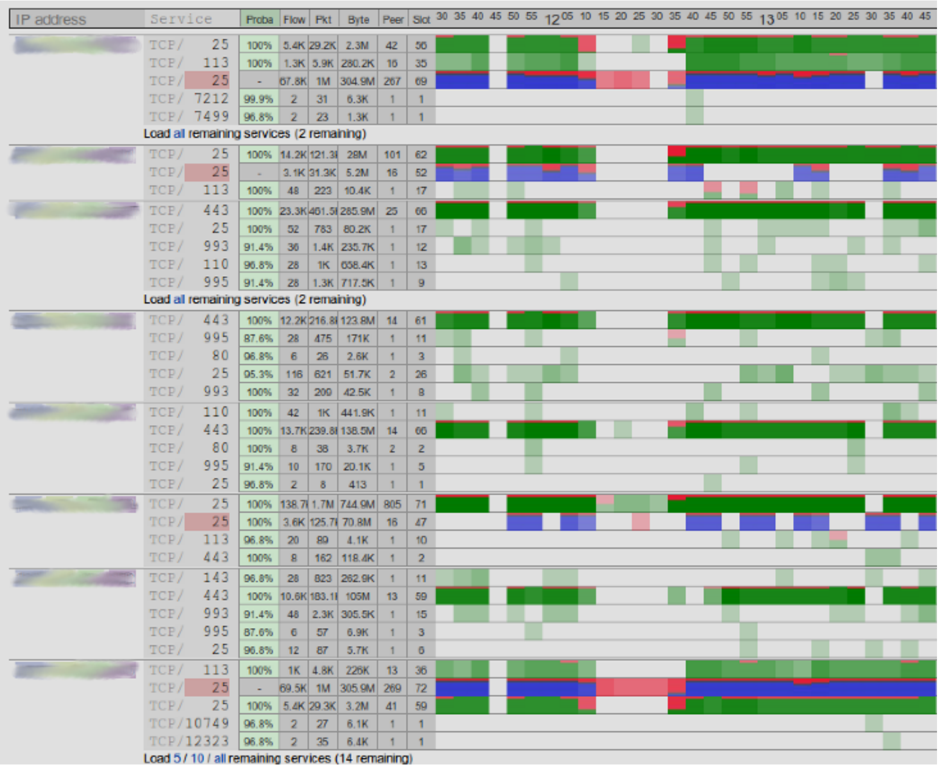
Nfsight can be used to rapidly identify the population of internal servers. The passive service detection algorithm identifies servers actively used in the organization network. Through the front-end, operators can query monthly, weekly, or daily network activity by port number. For example, one can query all internal IP addresses hosting a VNC server (port TCP/5900), and display the daily average number of peers each of the IP addresses has been connected to over the past few weeks. The dashboard also provides the top 20 hosted services ranked by the number of internal servers.

3.1.2 Network Monitoring

In addition to filtering activity by port, one can query activity by subnet to check for anomalies in a specific part of the network. An example of anomaly is the loss of network connectivity for a set of hosts. We illustrate this case in Figure 8, which represents the effect of a power outage from the perspective of both the servers which lost power (activity in green) and the clients which could no longer reach the servers (activity in red). The visualization provided by Nfsight makes it easy to determine the duration of the event (it started at 12:10 PM and activity was fully restored at 12:40 PM) and the list of internal hosts affected.

3.1.3 Policy Checking

In most organizations, critical subnets are subject to a tight security policy to prevent exposure of sensitive hosts. Nfsight can be used to check that these policies are properly implemented and are not compromised. The front-end Visualizer organizes assets per IP address and service, providing the operators an instant view to detect rogue hosts or rogue services. A watchlist allows one to register hosts with a service profile and be alerted when an unknown service is detected. For example, the pro-



IP address	Service	Proba	Flow	Pkt	Byte	Peer	Slot
[redacted]	TCP/ 25	100%	5.4K	29.2K	2.3M	42	56
	TCP/ 113	100%	1.3K	5.9K	280.2K	16	35
	TCP/ 25	-	67.8K	1M	304.9M	267	69
	TCP/ 7212	99.9%	2	31	6.3K	1	1
	TCP/ 7499	96.8%	2	23	1.3K	1	1
	Load all remaining services (2 remaining)						
[redacted]	TCP/ 25	100%	14.2K	121.3	28M	101	62
	TCP/ 25	-	3.1K	31.3K	5.2M	16	52
	TCP/ 113	100%	48	223	10.4K	1	17
[redacted]	TCP/ 443	100%	23.3K	461.5	285.9M	25	66
	TCP/ 25	100%	52	783	80.2K	1	17
	TCP/ 993	91.4%	36	1.4K	235.7K	1	12
	TCP/ 110	96.8%	28	1K	658.4K	1	13
	TCP/ 995	91.4%	28	1.3K	717.5K	1	9
	Load all remaining services (2 remaining)						
[redacted]	TCP/ 443	100%	12.2K	216.8	123.8M	14	61
	TCP/ 995	87.6%	28	475	171K	1	11
	TCP/ 80	96.8%	6	26	2.6K	1	3
	TCP/ 25	95.3%	116	621	51.7K	2	26
	TCP/ 993	100%	32	209	42.5K	1	8
[redacted]	TCP/ 110	100%	42	1K	441.9K	1	11
	TCP/ 443	100%	13.7K	239.8	138.5M	14	66
	TCP/ 80	100%	8	38	3.7K	2	2
	TCP/ 995	91.4%	10	170	20.1K	1	5
	TCP/ 25	96.8%	2	8	413	1	1
[redacted]	TCP/ 25	100%	138.7	1.7M	744.9M	805	71
	TCP/ 25	100%	3.6K	125.7	70.8M	16	47
	TCP/ 113	96.8%	20	89	4.1K	1	10
	TCP/ 443	100%	8	162	118.4K	1	2
[redacted]	TCP/ 143	96.8%	28	823	262.9K	1	11
	TCP/ 443	100%	10.6K	183.1	105M	13	59
	TCP/ 993	91.4%	48	2.3K	305.5K	1	15
	TCP/ 995	87.6%	6	57	6.9K	1	3
	TCP/ 25	96.8%	12	87	5.7K	1	6
[redacted]	TCP/ 113	100%	1K	4.8K	226K	13	36
	TCP/ 25	-	69.5K	1M	305.9M	269	72
	TCP/ 25	100%	5.4K	29.3K	3.2M	41	59
	TCP/10749	96.8%	2	27	6.1K	1	1
	TCP/12323	96.8%	2	35	6.4K	1	1
	Load 5 / 10 / all remaining services (14 remaining)						

Figure 8: Effect of a power outage on connectivity

file for an email server could consist of three services: TCP/25 (mail), TCP/143 (IMAP), and TCP/993 (IMAP over SSL). Any additional open port detected on this host would raise an alert automatically. This functionality can also be achieved by active scanning tools such as PBNJ [24], but the passive approach provided by Nfsight is less intrusive and offers a continous view of the service activity.

3.2 Malicious Activity

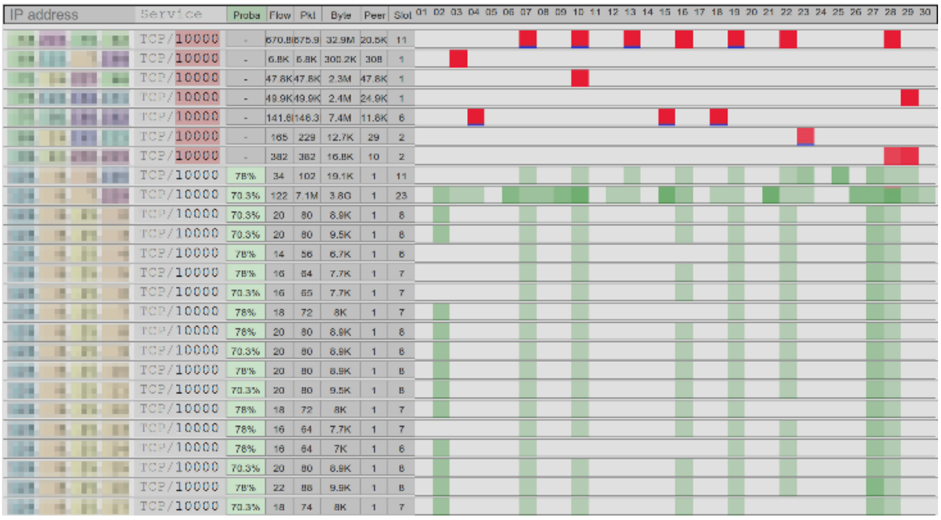
3.2.1 Scanning Activity and Vulnerable Servers

The filtering features of the front-end Visualizer allows one to query external clients generating unidirectional flows. These clients are often scanners targeting the organization IP addresses randomly or sequentially, and trying to find open services to compromise. As shown in Figure 9, the dashboard of Nfsight also provides the top 20 probed services ranked by number of scanners. Operators can click on a service to display the details of the scanning activity and more importantly, the list of internal hosts that scanners were able to find. This information is critical when a new vulnerability linked to a specific service is discovered, because security administrators can use Nfsight to learn, first, if attackers are actively trying to exploit it, and, second, what are the internal hosts that potentially need to be patched or closed.

Figure 10 illustrates this feature by showing the activity for port TCP/10000 over a period of 19 days. This port is known to host the Webmin application, which has been vulnerable to remote exploits [34]. We can see two parts in Figure 10: the top part in red shows external hosts scanning the organization network to find vulnerable applications on port TCP/10000. The bottom part in green represents internal hosts listening on port TCP/10000. The coloring is automatic based on the number of unanswered unidirectional flows (red) versus valid bidirectional flows (green). Moreover, the average number of peers displayed for each end point in the metric section clearly discriminates scanning activity (between 16 and 27,200 peers scanned per day) and server activity (1 client on average per day).

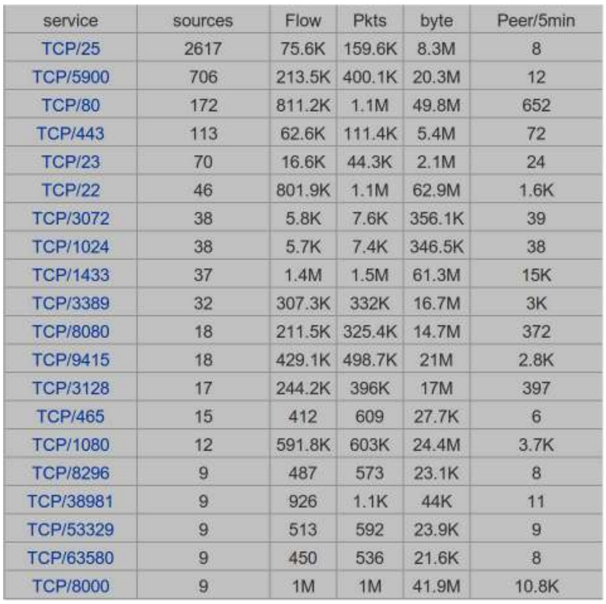
3.2.2 Compromised Hosts

In addition to external scanners, Nfsight can detect and display internal hosts generating an abnormal volume of unidirectional flows. These hosts are often compromised by a malware that tries to spread. The Intrusion Detector notifies the operators by means of automatically generated alarms when such a host is observed in the network. As described in Section 2.3.3, each alert contains a link that shows the service activity detected by Nfsight and



IP address	Service	Proba	Flow	Pkt	Byte	Peer	Slot
[redacted]	TCP/10000	-	670.8	675.9	32.9M	20.5K	11
[redacted]	TCP/10000	-	6.8K	6.8K	300.2K	308	1
[redacted]	TCP/10000	-	47.8K	47.8K	2.3M	47.8K	1
[redacted]	TCP/10000	-	49.9K	49.9K	2.4M	24.9K	1
[redacted]	TCP/10000	-	141.6	146.3	7.4M	11.6K	6
[redacted]	TCP/10000	-	165	229	12.7K	29	2
[redacted]	TCP/10000	-	382	382	16.8K	10	2
[redacted]	TCP/10000	78%	34	102	19.1K	1	11
[redacted]	TCP/10000	70.3%	122	7.1M	3.8G	1	23
[redacted]	TCP/10000	70.3%	20	80	8.9K	1	8
[redacted]	TCP/10000	70.3%	20	80	9.5K	1	8
[redacted]	TCP/10000	78%	14	56	6.7K	1	6
[redacted]	TCP/10000	78%	16	64	7.7K	1	7
[redacted]	TCP/10000	70.3%	16	65	7.7K	1	7
[redacted]	TCP/10000	78%	18	72	8K	1	7
[redacted]	TCP/10000	78%	20	80	8.9K	1	8
[redacted]	TCP/10000	70.3%	20	80	8.9K	1	8
[redacted]	TCP/10000	78%	20	80	8.9K	1	8
[redacted]	TCP/10000	70.3%	20	80	9.5K	1	8
[redacted]	TCP/10000	78%	18	72	8K	1	7
[redacted]	TCP/10000	78%	16	64	7.7K	1	7
[redacted]	TCP/10000	78%	16	64	7K	1	6
[redacted]	TCP/10000	70.3%	20	80	8.9K	1	8
[redacted]	TCP/10000	78%	22	88	9.9K	1	8
[redacted]	TCP/10000	70.3%	18	74	8K	1	7

Figure 10: Scanners targeting port 10000 and internal servers hosting a service on this port



service	sources	Flow	Pkts	byte	Peer/5min
TCP/25	2617	75.6K	159.6K	8.3M	8
TCP/5900	706	213.5K	400.1K	20.3M	12
TCP/80	172	811.2K	1.1M	49.8M	652
TCP/443	113	62.6K	111.4K	5.4M	72
TCP/23	70	16.6K	44.3K	2.1M	24
TCP/22	46	801.9K	1.1M	62.9M	1.6K
TCP/3072	38	5.8K	7.6K	356.1K	39
TCP/1024	38	5.7K	7.4K	346.5K	38
TCP/1433	37	1.4M	1.5M	61.3M	15K
TCP/3389	32	307.3K	332K	16.7M	3K
TCP/8080	18	211.5K	325.4K	14.7M	372
TCP/9415	18	429.1K	498.7K	21M	2.8K
TCP/3128	17	244.2K	396K	17M	397
TCP/465	15	412	609	27.7K	6
TCP/1080	12	591.8K	603K	24.4M	3.7K
TCP/8296	9	487	573	23.1K	8
TCP/38981	9	926	1.1K	44K	11
TCP/53329	9	513	592	23.9K	9
TCP/63580	9	450	536	21.6K	8
TCP/8000	9	1M	1M	41.9M	10.8K

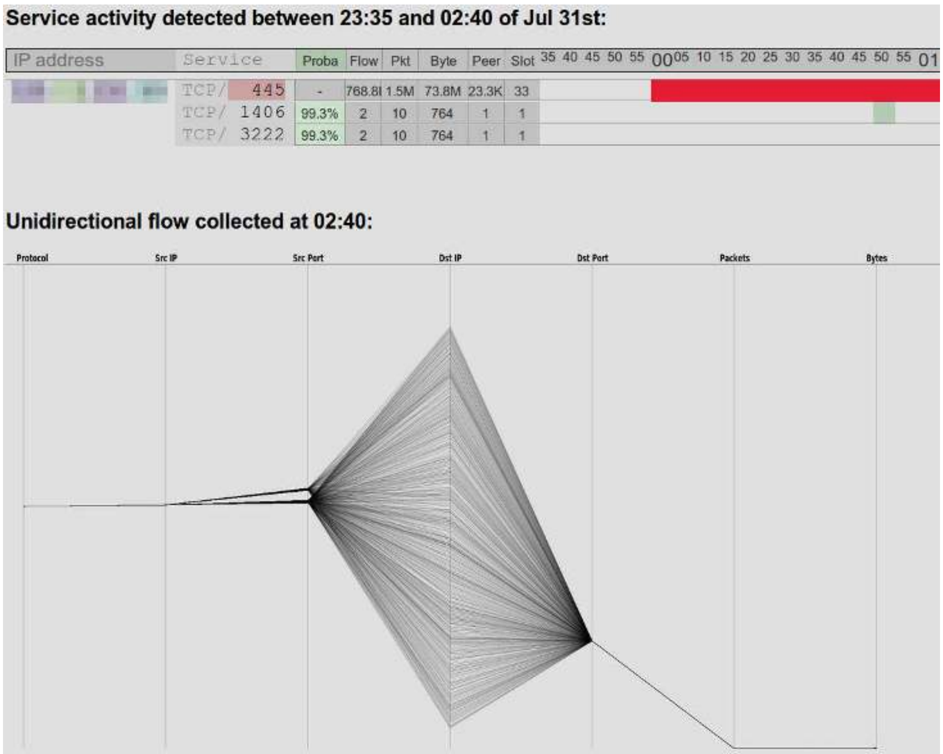
Figure 9: Top 20 scanned services

the details of flows related to the event. Consequently, operators can check if these alerts are due to malicious behavior or normal server behavior.

Figure 11 illustrates the activity of an internal host which was compromised and started at midnight to send a massive number of probes to random destination IP addresses on port TCP/445. Nfsight provides information about the scanning rate, on average 23,300 IP every 5 minutes, and the uniform distribution of targets from the parallel plot provided by Picviz [33]. Security administrators who tested Nfsight indicated that they cannot configure their IPS devices to detect and block this type of massive scanning activity, because the IPS devices would be at risk of becoming overloaded. Therefore, Nfsight complements other security solutions by leveraging NetFlow for scalable security monitoring.

3.2.3 Distributed Attacks

The visualization feature of Nfsight enables security administrators to identify coordinated attacks and to understand their scope. An example of a distributed scan originating from a set of internal SSH servers is provided in Figure 12. A total of 19 servers were compromised because the password for one shared account was determined through brute-force attack. Attackers installed a remote control software on each host and then launched a distributed scan at 8 PM to find additional SSH servers to compromise. The timeseries representation and the distinction between client/server activity allows administrators to immediately see the coordinated nature of the attack.

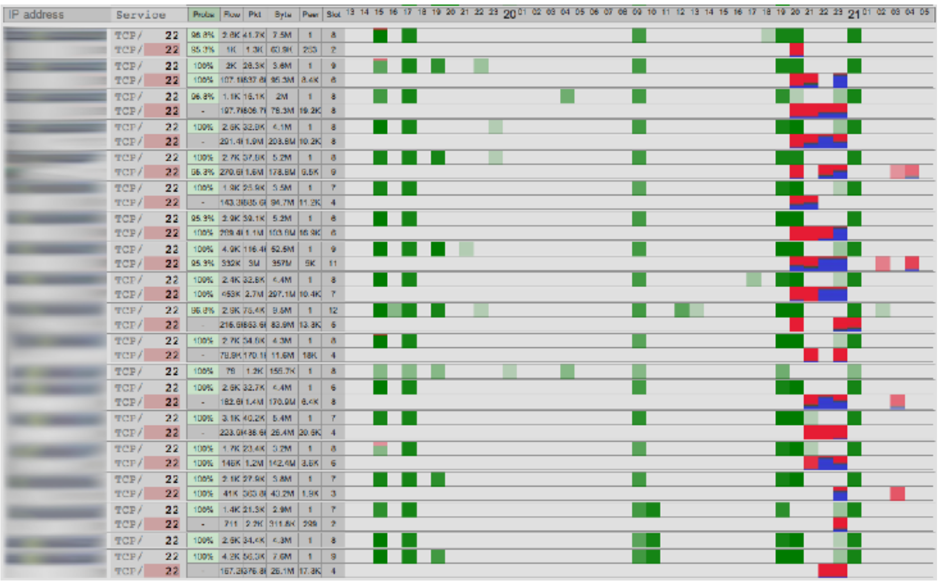


Service activity detected between 23:35 and 02:40 of Jul 31st:

IP address	Service	Proba	Flow	Pkt	Byte	Peer	Slot
[redacted]	TCP/ 445	-	768.8l	1.5M	73.8M	23.3K	33
	TCP/ 1406	99.3%	2	10	764	1	1
	TCP/ 3222	99.3%	2	10	764	1	1

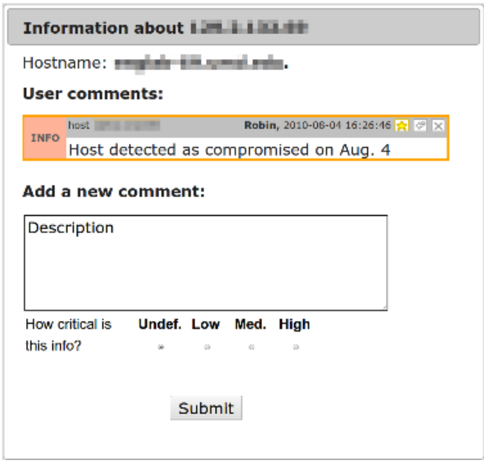
Unidirectional flow collected at 02:40:

Figure 11: Compromised internal host scanning a large range of destination IP on port TCP/445 (Netbios service)



IP address	Service	Proba	Flow	Pkt	Byte	Peer	Slot
[redacted]	TCP/ 22	96.8%	2.8K	41.7K	7.5M	1	8
	TCP/ 22	95.3%	1K	1.3K	60.9K	253	2
[redacted]	TCP/ 22	100%	2K	28.3K	3.6M	1	9
	TCP/ 22	100%	107.1l	837.8l	95.3M	8.4K	6
[redacted]	TCP/ 22	96.8%	1.1K	16.1K	2M	1	8
	TCP/ 22	-	197.7l	806.7l	78.3M	19.2K	8
[redacted]	TCP/ 22	100%	2.6K	32.8K	4.1M	1	8
	TCP/ 22	-	201.4l	1.9M	203.8M	10.2K	8
[redacted]	TCP/ 22	100%	2.7K	37.5K	5.2M	1	8
	TCP/ 22	96.3%	270.6l	1.6M	178.8M	9.5K	9
[redacted]	TCP/ 22	100%	1.9K	25.9K	3.5M	1	7
	TCP/ 22	-	143.3l	885.6l	94.7M	11.2K	4
[redacted]	TCP/ 22	95.9%	2.9K	39.1K	5.2M	1	8
	TCP/ 22	100%	209.4l	1.1M	103.8M	16.9K	6
[redacted]	TCP/ 22	100%	4.9K	116.4l	52.5M	1	9
	TCP/ 22	95.9%	332K	3M	357M	5K	11
[redacted]	TCP/ 22	100%	2.4K	32.6K	4.4M	1	8
	TCP/ 22	100%	453K	2.7M	297.1M	10.4K	7
[redacted]	TCP/ 22	96.8%	2.8K	75.4K	9.5M	1	12
	TCP/ 22	-	216.6l	853.6l	83.9M	13.3K	6
[redacted]	TCP/ 22	100%	2.7K	34.6K	4.3M	1	8
	TCP/ 22	-	78.9K	170.1l	11.6M	18K	4
[redacted]	TCP/ 22	100%	79	1.2K	155.7K	1	8
[redacted]	TCP/ 22	100%	2.6K	32.7K	4.4M	1	6
	TCP/ 22	-	182.6l	1.4M	170.9M	6.4K	8
[redacted]	TCP/ 22	100%	3.1K	40.2K	5.4M	1	7
	TCP/ 22	-	223.0l	438.6l	26.4M	20.6K	4
[redacted]	TCP/ 22	100%	1.7K	23.4K	3.2M	1	8
	TCP/ 22	100%	146K	1.2M	142.4M	3.5K	6
[redacted]	TCP/ 22	100%	2.1K	27.9K	3.8M	1	7
	TCP/ 22	100%	41K	363.8l	40.2M	1.9K	3
[redacted]	TCP/ 22	100%	1.4K	21.3K	2.9M	1	7
	TCP/ 22	-	711	2.2K	311.8K	299	2
[redacted]	TCP/ 22	100%	2.6K	34.4K	4.3M	1	8
[redacted]	TCP/ 22	100%	4.2K	56.3K	7.6M	1	9
	TCP/ 22	-	167.2l	376.8l	26.1M	17.3K	4

Figure 12: Set of 19 compromised SSH servers remotely controlled (server activity in green) and launching a synchronized distributed scan towards port TCP/22 (client activity in blue and red)



Information about [redacted]

Hostname: [redacted].

User comments:

INFO host [redacted] **Robin,** 2010-08-04 16:26:46

Host detected as compromised on Aug. 4

Add a new comment:

Description

How critical is this info?	**Undef.**	**Low**	**Med.**	**High**
	●	○	○	○

Figure 13: User comment window for information sharing about a specific host

3.3 Forensic and Collaboration

The different case studies described previously show that Nfsight can be efficiently used to perform forensic tasks. The overview representation and detail-on-demand capability offer a fast and easy solution to understand what happened in the network. This functionality is augmented by several collaboration features. First, operators can click on any IP address or service to leave a comment and rate its criticality (low, medium or high). The comment window is illustrated in Figure 13. Second, email alerts contain links that the operators can use to rate the alert as true positive, false positive, or unknown. The web page displayed after clicking on these links allows operators to write a comment and rate the criticality of the alert. These comments are displayed on the dashboard of Nfsight and colored by criticality. Operators can reply to comments left by others and share their finding or expertise.

3.4 Limitations and Future Work

Nfsight provides a practical network situational awareness solution based on NetFlow flows. The main contributions are 1) passive service discovery, 2) intrusion detection and 3) automated alert and visualization. We showed with different use cases how Nfsight can help network administrators and security operators in their monitoring tasks. However, Nfsight has still important limitations that we plan to address in our future work.

First, Nfsight works with non-sampled flows. We note that results from other evaluations of passive detection techniques indicate that sampling has a limited impact on the overall accuracy. For example, [1] reports that capturing only 16% of the data results only in an 11% drop in discovered servers. However, we believe that random flow sampling will likely break our algorithm for identifying bidirectional flows. We plan on assessing the effect of sampling on the detection accuracy of the different heuristics. Furthermore, asymmetric routing can challenge our approach. Specifically, we assumed in this study that NetFlow collectors covered the pathways for both requests and replies. In some organization networks, replies and requests can sometimes take different routes for which there is no NetFlow collector deployed and therefore, we would not be able to pair the unidirectional flows into bidirectional flows.

We also note that Nfsight works at the network layer and therefore heavily relies on port numbers. As a consequence, it can be difficult or impossible for a network operator to identify the application behind a service detected by Nfsight. This issue arises from the fact that some applications use random ports or hide behind well-known ports. For example Skype is famous for using port 80 or port 443, normally reserved to web traffic, in order to evade firewall protection. Related work [6] on flow-based traffic classification proved that it is possible to accurately identify applications using only NetFlow. We plan on developing additional heuristics for Nfsight to be able to classify traffic regardless of the port numbers used. These heuristics can work on 1) relationships between flow characteristics, such as the ratio between number of packets and number of bytes or the time distribution of flows, and 2) relationships between hosts. We believe that discovering communication patterns between hosts would be critical to identify not only applications but also large communication structures such as those used by P2P networks or botnets.

Finally, the current intrusion detection rules are rudimentary and the fact that most of them are threshold-based means that they are prone to generate a significant volume of false positives. We implemented a feedback mechanism to leverage human expertise and facilitate the task of tuning the detection rules, but this process still involves important manual development. We plan to automate this task and integrate a machine-learning approach to create and tune rules based on samples of true and false positives.

4 Related Work

NetFlow is highly popular among network operators and researchers because it offers a comprehensive view of network activity while being scalable and easy to deploy in large networks. As a result, an important number of tools and publications have been produced over the past decade, as shown by [28] and [17]. We present in this section an overview of these resources organized according to our areas of interests: Netflow processing and visualization, and service detection.

4.1 NetFlow Processing and Visualization Applications

Working with NetFlow is a multi-step process. First, flow records are generated by a compatible network device, typically a router, or by a software probe such as [29, 21, 36]. These flows are then sent over the network in UDP packets to collectors according to the NetFlow protocol. The role of a collector is to store flow records in flat files or in a database. The collector is often linked to a set of processing tools to allow a network operator to read and filter flow records. Processing tools include CAIDA Cflowd [2], OSU flow-tools [27], SiLK [8] and more recently Nfdump [18].

In addition to command line tools, several graphical user interfaces exist to visualize and query network activity. NTOP [22] and Nfsen [10] are two popular solutions that provide a web interface to network operators. We note that we developed Nfsight as a plugin of Nfsen because of its simplicity, extensibility and processing capability.

An important body of research has been conducted on the topic of NetFlow visualization. The NCSA research center at the University of Illinois produced NvisionIP [16] and VisFlowConnect [38]. NvisionIP provides a two-dimensional map to visualize the network characteristics of up to 65,536 hosts in a single view. It has been extended to include a graphical filtering rule system [15] to allow operators to easily spot abnormal activity. VisFlowConnect offers a parallel-plot view with drill-down features. Compared to Nfsight, the main limitation of these two tools is that they work offline, while our solution processes NetFlow flows in near real time.

Researchers at the University of Wisconsin developed FlowScan [25] and NetPY [3]. NetPY is an interactive visualization application written in Python on top of flow-tools. It provides an automated sampling algorithm and enables operators to understand how network traffic is used through heatmaps, timeseries and hierachical heavy hitters plots. FlowScan works at a higher level by providing traffic volume graphs of network applications. The architecture of FlowScan, which consists of Perl scripts and uses RRDTool, is very similar to the architecture of Nfsen. Also, Nfsight shares with FlowScan the idea of using heuristics to classify flow records. However, FlowScan lacks alerting capabilities and does not determine client/server relationships.

Other research projects on the topic of flow visualization include FloVis [31], VIAssist [5] and NFlowVis [7]. FloVis offers a set of modules such as Overflow [9] and NetByte Viewer [30] to display the same network activity through different perspectives in order to gain a better understanding of host behavior. VIAssist and NFlowVis adopt the same objective with drill-down features and multiple visualization techniques. NFlowVis integrates state-of-the art plots by making use of treemap and a hierachical edge bundle view. Similarly to Nfsight, VIAssist offers collaboration features to allow operators to share items of interest and to communicate findings. We note that none of these three visualization frameworks are publicly available.

4.2 Service Detection and Bidirectional Flows

Solutions for service discovery can be divided into active and passive techniques. Active techniques send network probes to a set of targets to check the presence of any listening service, while passive techniques extract information about services from network sniffing devices. A well-known open source active scanner is Nmap [20]. The drawbacks of active techniques are: 1) they provide only a snapshot in time of the network, 2) they cannot detect services protected by firewalls, 3) they are intrusive and not scalable, and 4) aggressive scanning may also cause system and network disruptions or outages [35, 1]. Passive solutions offer a continuous view of the network, their results are not impacted by firewalls, and they are highly scalable. The main limitation of the passive approaches is that they detect only active services, i.e., any unused services with no incoming traffic cannot be discovered. However, by providing a low overhead continuous passive discovery approach, services that do communicate will be detected. A well-known open source passive service detector working on packet data is Pads [23].

A passive and accurate detection of network services working on network flows would be trivial with bidirectional flows where request flows initiated by clients and reply flows initiated by servers can be easily identified. However, most organization networks are currently instrumented with traditional unidirectional flow solutions such as NetFlow, and they lack the capability to generate and collect bidirectional flows. This motivated us to design a solution based only on unidirectional flow. We note that the IPFIX IETF working group has recently introduced a new standard format to export network flows based on NetFlow version 9 [4], which includes the capability to export bidirectional flows generated directly at the measurement interface [32]. We see our approach as a robust intermediate solution between the current large scale deployment of NetFlow, which is unidirectional, and the future implementation by router vendors and deployment by organization networks of IPFIX, which can be bidirectional.

Rwmatch from SiLK [8] shares the same motivation of generating correctly oriented bidirectional network flows from unidirectional flows. Rwmatch uses two heuristics to decide on the orientation of bidirectional flows: timestamp of request and reply flows, and server port number being below 1024. However, we have observed that both of these heuristics can be fallible by themselves. Therefore, we use five additional heuristics and combine heuristic outputs through Bayesian inference in order to improve the accuracy of server detection over time. We note that another tool similar to rwmatch called flow-connect, developed as part of the OSU Flow-tools framework, has been suggested in [27] but has actually never been implemented.

Finally, two alternative approaches YAF from CERT [36] and Argus [26] generate bidirectional flows not from unidirectional flows but from packet data. Both tools work by processing packet data from PCAP dump files or directly from a network interface, and then export bidirectional flows following the IPFIX format.

5 Conclusion

Timely information on what is occurring in their networks is crucial for network and security administrators. Nfsight provides an easy to use graphical tool for administrators to gain knowledge on the set of services running in their networks, as well as on any anomalous activities. Nfsight is non-intrusive since it relies on passively collected NetFlow data, provides a near real-time report on network activities, allows data to be viewed at different time granularities, and supports collaboration between system administrators. Nfsight uses a combination of heuristics and Bayesian inference to identify services and graphlet-based technique to detect intrusions. In this paper, we described the architecture and heuristics used by Nfsight, evaluated its accuracy in service discovery, and presented a number of real use-cases. Our future work includes development and evaluation of additional server discovery heuristics. We also plan to revise the intrusion detection rules and to complete the implementation of the feedback mechanism to adjust detection thesholds automatically.

6 Acknowledgments

We would like to thank Gerry Sneeringer, Kevin Shivers and Bertrand Sobesto for their ideas and their help labeling and investigating malicious activity. We are also grateful to Virginie Klein for her contribution on the Intrusion Detector. Finally we would like to thank William Sanders, Jenny Applequist, Danielle Chrun, Eser Kandogan and the anonymous reviewers for their guidance and comments on the paper.

7 Availability

The documentation and the source code of Nfsight are freely available at:

`http://nfsight.research.att.com`

References

[1] Bartlett, G., Heidemann, J., and Papadopoulos, C. Understanding passive and active service discovery. In *Proc. 7th ACM Internet Measurement Conference* (2007), pp. 55–60.

[2] Caida cflowd. http://www.caida.org/tools/measurement/cflowd/, 2010.

[3] Cirneci, A., Boboc, S., Leordeanu, C., Cristea, V., and Estan, C. Netpy: Advanced Network Traffic Monitoring. In *Proc. Int Conf. on Intelligent Networking and Collaborative Systems (INCOS'09)* (2009), pp. 253–254.

[4] Claise, B., Quittek, J., Bryant, S., Aitken, P., Meyer, J., Trammell, B., Boschi, E., Wenger, S., Chandra, U., Westerlund, M., et al. RFC 5101 Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information, 2008.

[5] D'Amico, A., Goodall, J., Tesone, D., and Kopylec, J. Visual discovery in computer network defense. *IEEE Computer Graphics and Applications 27*, 5 (2007), 20–27.

[6] Erman, J., Mahanti, A., Arlitt, M., and Williamson, C. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *Proceedings of the 16th international conference on World Wide Web* (2007), ACM, p. 892.

[7] Fischer, F., Mansmann, F., Keim, D., Pietzko, S., and Waldvogel, M. Large-scale network monitoring for visual analysis of attacks. In *Proc. Workshop on Visualization for Computer Security (VizSEC)* (2008), Springer, p. 111.

[8] Gates, C., Collins, M., Duggan, M., Kompanek, A., and Thomas, M. More NetFlow tools: For performance and security. In *Proc. 18th USENIX Large Installation System Administration Conf. (LISA)* (2004), pp. 121–132.

[9] Glanfield, J., Brooks, S., Taylor, T., Paterson, D., Smith, C., Gates, C., and McHugh, J. OverFlow: An Overview Visualization for Network Analysis. In *Proc. 6th Int. Workshop on Visualization for Cyber Security (VizSec)* (2009), pp. 11–19.

[10] Haag, P. Watch your Flows with NfSen and NFDUMP. In *50th RIPE Meeting* (2005).

[11] Hughes, E., and Somayaji, A. Towards network awareness. In *Proc. 19th USENIX Large Installation System Administration Conf. (LISA)* (2005), pp. 113–124.

[12] Iana assigned port numbers. http://www.iana.org/assignments/port-numbers, 2010.

[13] Karagiannis, T., Papagiannaki, K., and Faloutsos, M. BLINC: multilevel traffic classification in the dark. In *Proc. ACM SIGCOMM Conference* (2005), pp. 229–240.

[14] Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., and Lee, K. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference* (2008), ACM, pp. 1–12.

[15] Lakkaraju, K., Bearavolu, R., Slagell, A., Yurcik, W., and North, S. Closing-the-loop in NVisionIP: Integrating discovery and search in security visualizations. In *Proc. IEEE Workshop on Visualization for Computer Security (VizSEC)* (2005).

[16] LAKKARAJU, K., YURCIK, W., AND LEE, A. NVisionIP: netflow visualizations of system state for security situational awareness. In *Proc. ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)* (2004), pp. 65–72.

[17] LEINEN, S. FloMA: Pointers and Software, NetFlow. Tech. rep., SWITCH, 2010.

[18] Nfdump. http://nfdump.sourceforge.net, 2010.

[19] Nfsen. http://nfsen.sourceforge.net, 2010.

[20] Nmap. http://www.nmap.org, 2010.

[21] Nprobe: Netflow/ipfix network probe. http://www.ntop.org/nProbe.html, 2010.

[22] Ntop: Network traffic probe. http://www.ntop.org, 2010.

[23] Pads. http://passive.sourceforge.net, 2010.

[24] Pbnj. http://pbnj.sourceforge.net, 2010.

[25] PLONKA, D. Flowscan: A Network Traffic Flow Reporting and Visualization Tool. In *Proc. 14th USENIX Large Installation System Administration Conf. (LISA)* (2000), pp. 305–318.

[26] QOSIENT, L. Argus: Network Audit Record Generation and Utilization System.

[27] ROMIG, S., FULLMER, M., AND LUMAN, R. The OSU flow-tools package and CISCO NetFlow logs. In *Proc. 14th USENIX Large Installation System Administration Conf, (LISA)* (2000), pp. 291–304.

[28] SO-IN, C. A Survey of Network Traffic Monitoring and Analysis Tools. Cse 576m computer system analysis project, Washington University in St. Louis, 2009.

[29] Softflowd: fast software netflow probe. http://www.mindrot.org/projects/softflowd/, 2010.

[30] TAYLOR, T., BROOKS, S., AND MCHUGH, J. NetBytes viewer: An entity-based netflow visualization utility for identifying intrusive behavior. pp. 101–114.

[31] TAYLOR, T., PATERSON, D., GLANFIELD, J., GATES, C., BROOKS, S., AND MCHUGH, J. FloVis: Flow Visualization System. In *Proc. Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)* (2009), pp. 186–198.

[32] TRAMMELL, B., AND BOSCHI, E. RFC 5103: Bidirectional Flow Export Using IP Flow Information Export (IPFIX), 2008.

[33] TRICAUD, S., AND SAADÉ, P. Applied parallel coordinates for logs and network traffic attack analysis. *Journal in computer virology 6*, 1 (2010), 1–29.

[34] Webmin vulnerability, cve-2006-3392, 2006.

[35] WEBSTER, S., LIPPMANN, R., AND ZISSMAN, M. Experience using active and passive mapping for network situational awareness. In *Proc. 5th IEEE Int. Symp. on Network Computing and Applications (NCA)* (2006), pp. 19–26.

[36] Yaf. http://tools.netsa.cert.org/yaf/, 2010.

[37] YURCIK, W. Visualizing NetFlows for security at line speed: the SIFT tool suite. In *Proc. 19th Large Installation System Administration Conf. (LISA)* (2005), USENIX, pp. 169–176.

[38] YURCIK, W. VisFlowConnect-IP: a link-based visualization of Netflows for security monitoring. In *18th Annual FIRST Conf. on Computer Security Incident Handling* (2006).