Community-based analysis of netflow for early detection of security incidents

Stefan Weigert[†] Matti A. Hiltunen[‡] Christof Fetzer[†]

[†]TU Dresden
Dresden, Germany
{stefan,christof}@se.inf.tu-dresden.de

[‡]AT&T Labs Research
180 Park Ave.
Florham Park, NJ, USA
hiltunen@research.att.com

***Abstract*—Detection and remediation of security incidents (e.g., attacks, compromised machines, policy violations) is an increasingly important task of system administrators. While numerous tools and techniques are available (e.g., Snort, nmap, netflow), novel attacks and low-grade events may still be hard to detect in a timely manner. In this paper, we present a novel approach for detecting stealthy, low-grade security incidents by utilizing information across a community of organizations (e.g., banking industry, energy generation and distribution industry, governmental organizations in a specific country, etc). The approach uses netflow, a commonly available non-intrusive data source, analyzes communication to/from the community, and alerts the community members when suspicious activity is detected. A community-based detection has the ability to detect incidents that would fall below local detection thresholds while maintaining the number of alerts at a manageable level for each day.**

I. Introduction

Detection and remediation of security incidents (e.g., attacks, compromised machines, policy violations) is an increasingly important task of system administrators. While numerous tools and techniques are available, novel attacks and low-grade security events may still be hard to detect in a timely manner. Specifically, system administrators typically have to base their actions on observing the local traffic to and from their own networks as well as global security incident alerts from organizations such as SEI CERT[1], Arbor Atlas[2], or software and hardware vendors. However, stealthy targeted attacks may slip below detection thresholds both in the local data alone or on the global scale.

Furthermore, the nature of internet-based attacks is changing from random hacking to financially or politically motivated attacks. For example, botnets are increasingly leased out to highest bidders and DDoS attacks are often used as a means for blackmail. Moreover, attacks targeting industries with financial information (e-commerce, banking, gaming, insurance) are increasing and the threat of attacks against SCADA (supervisory control and data acquisition) systems in electrical power generation, transmission, and distribution (among other industrial process control systems) is even considered a potential target for terrorism [10].

[1]http://www.cert.org/
[2]http://atlas.arbor.net/

Targeted attacks might not leave a large traffic footprint in the targeted organization since one machine with access to the desired information or control system may be sufficient for the attacker to achieve their goals. It is often difficult to detect such low-footprint attacks based on local monitoring alone because it is often necessary to set local alerting thresholds high enough not to generate too many false positives and overwhelm the system administrators. But as a result, a stealthy attack or compromise may lay undetected. Therefore, it is possible for an attacker to target many such organizations without being detected. For example, the attacker may want to maximize profit by attacking multiple financial organizations concurrently before the vulnerability used is detected and corrected. Similarly, terrorists may require the control of many companies to achieve their goal of large scale damage.

In this paper, we present a novel approach for detecting stealthy, low-grade security incidents by utilizing information across a community of organizations (e.g., banking industry, energy generation and distribution industry). We will show by using an example that we can find possible attacks (or attempts) that only transfer very little data (e.g., a few bytes) and thus would remain undetected by conventional approaches.

The remainder of this paper is structured as follows. In Section II, we present the technical approach based on netflow data and construction of communities of interest. Section III describes the implementation of the system, including the algorithms used for the analysis. We evaluate the performance of our system in Section IVi and present selected case studies of suspicious activity we have identified in Section V . Section VI outlines related work in the area and Section VII concludes the paper.

II. Approach

A. Service vision

Our technique is based on the concept of *community*, in our case defined as a collection of (at least two) organizations. A community can be specified based on any criteria relevant for attack detection. For example, it could consists of businesses in a particular industry (e.g., banking, health care, insurance, etc), organizations within a country (e.g., businesses and government agencies in one country), or organizations with particular

type of valuable information (e.g., industrial espionage or customer credit card information). We detect stealthy security attacks by observing the communication to/from the member organizations of a community. The intuition being that within each organization only very few machines may be attacked or compromised and as a result an attack can be very hard to detect within each organization. However, by observing the communication behavior across multiple organizations in the community, such stealthy behavior may become visible.

Given that we analyze communication in the Internet, each organization is defined by the list or range of IP addresses belonging to the organization. We consider Internet communication connections (reported by netflow, for example) within the communities and between communities and external IP addresses who do not belong to any community. For our analysis, all the IP addresses within an organization can be collapsed into one identifier representing the organization. Any communication between two IP addresses where neither belongs to one of our communities and neither has communicated with a community in the past can be ignored. Furthermore, communication with IP addresses belonging to commonly used Internet services (e.g., search, news, social media) can be white listed and removed from consideration.

We construct a communication graph for each IP address that communicates with at least one organization in a community as illustrated in Figure 1. This figure shows the communication graph for an external IP address (i.e., some IP address outside any of the communities of interest). This node has communicated with two communities, one consisting of organizations 7 and 8, and the other consisting of organizations 1 through 6. A directed edge from some node A to some other node B in the graph indicates that A has sent messages to B. Although not depicted in the figure, each edge may contain additional information, such as the combinations of source and destination ports used.

The weight of the edge is used to quantify the importance of the communication. The importance can be based simply on the number of messages or bytes sent, or the number of contacted individual members in the targeted organization. However, some communication may be more important than others from security point of view. For example, some port numbers are more often involved with malicious activity (e.g., based on CERT reports) and communication using such ports can be weighted more heavily.

The weight is also used to limit the size of each graph. The size of the graph is determined by the number of nodes it contains. If the size exceeds a given threshold, we remove the weakest links until the threshold is reached. This is necessary because storing all communications would require too much space even for a single day. For example, in our data set consisting of heavily sampled netflow, a given weekday contains about 860 million entries. These 860 million recorded netflows originate in 28 million distinct IP addresses. Therefore, if we would not filter unimportant IP addresses, we would need to store 28 million graphs. Moreover, each of these 28 million IP addresses often connects with 1 to 2 million other IP addresses.

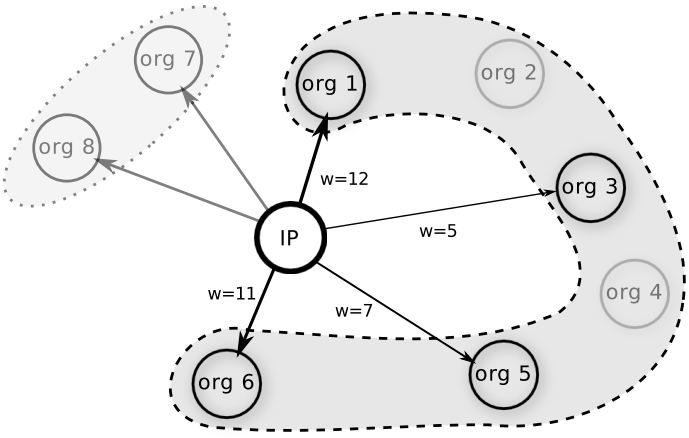


Fig. 1: Communication graph for an IP address

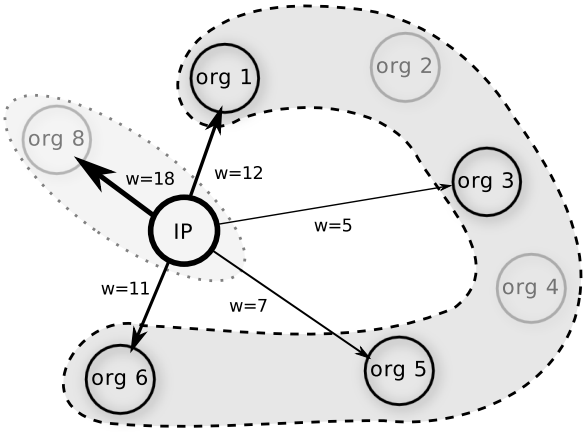


Fig. 2: Communication graph for a community member

Thus, if we did not limit the size of each graph, we would have some graphs that are too large to fit into memory. The situation would be even more challenging if we analyzed the data for one month or a week instead of the current one day at a time.

As already stated, we also consider communication within a community and across communities. With that, we are able to detect already compromised computers inside an organization when they try to attack further organizations as shown in Figure 2. To reduce the number of false positives (many organizations have frequent contact with other organizations of the same or other communities), a computer inside an organization that belongs to a community (or is contained in the whitelist) has to show more suspicious behavior than an external IP address before an alarm is generated. For example, we do not consider communication via port 443 with or across communities.

Given such communication graphs, a potential security incident is suspected when an IP address communicates with a specified number of community members. Typical examples of security threats that can be detected using this approach include botnet controllers managing a number of bots in the

community, compromised machines downloading stolen information on a dedicated server, an attacker targeting machines in multiple organizations, as well as many security policy violations (e.g., illegal software download sites, etc). The number of alarms can be controlled using thresholds and the system can memorize IP addresses that have already been reported recently. When there are false positives, the system administrators can extend the whitelist.

An IP address may contact a large number of community members either because the community is actually targetted or if the attacker is targetting all or most of the Internet (e.g., broad port scan). The system administrators may want to react differently to these alternative scenarios. Therefore, for each IP address that has contacted a community member, our system keeps track of how many times it has communicated with IP addresses outside our communities of interest.

B. Input data

Our community-based alerting service uses netflow as its input data source (although other types of information could be utilized as well). Netflow is a standard data format collected and exported by most networking equipment, in particular, network routers. It provides summary information about each network communication passing through the network equipment. Specifically, a network flow is defined as an unidirectional sequence of packets that share source and destination IP addresses, source and destination port numbers, and protocol (e.g., TCP or UDP). Each netflow record carries 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. Note, however, that the netflow record does not contain any information about the contents of the communication between the source and destination IP addresses.

The community-based alerting service requires access to netflow to/from each of the organizations in the community. Such data can be collected by each of the organizations in the community at their edge routers and then collected at a central location for processing. Alternatively, it can be provided by an ISP that serves a number of the organizations in the community. Note that the netflow data may be sampled (to reduce the volume of the data) and the actual IP addresses of the computers within each organization can be obfuscated prior to the analysis (e.g., all IP addresses belonging to an organization can be collapsed into one address) if desired.

Given the collected netflows and the IP address ranges belonging to each member organization in the community, our alerting service analyses the data (either real time or in daily or hourly batches) and generates alerts to the system administrators. The analysis algorithm is described in Section III. A whitelist can be used to eliminate any legitimate communication destinations from consideration (e.g., search engines, CDNs, banking, on-line retailers, etc).

III. Implementation

A. Architecture

The architecture of our system is presented in Figure 3. We use three different types of processing components that do not share any state and are executed as individual processes: the parse, the filter, and the graph components. Each component can be replicated and executed by any number of processes (e.g., L, M, $N1$, and $N2$ in the figure). Every process of every component has a unique id (from 0 to the number of processes for the component-1) that is used for message routing. Since the parse component is connected to the filter component, each parse process is connected to each filter process. The same is true for the filter and graph components. Note that the system supports multiple different kinds of graph components in one system configuration as illustrated by *Graph 1* and *Graph 2* in the figure. Different graph components can be used to realize different alerting conditions as we will describe below.

The communication between components is based on event messages that are sent via TCP-channels. A message consists of a key and a body that are defined by the pair of interacting components (e.g., parse and filter, or filter and graph) and may contain any information desired by these components. For the key, a hash function h must be available that maps the contents of a key into an unsigned integer, which is used to route the event message to the right receiving process. For example, if a parse process is connected to 2 filter processes (i.e., $M = 2$), the receiving filter process is chosen by calculating the modulo of the hash of the key and 2. Thus, in this particular example, all keys with even hashes would be routed to the first and all keys with odd hashes to the second filter process.

The internal state maintained by each component is partitioned by the same key, making it possible to distribute their processing load onto multiple cores efficiently.

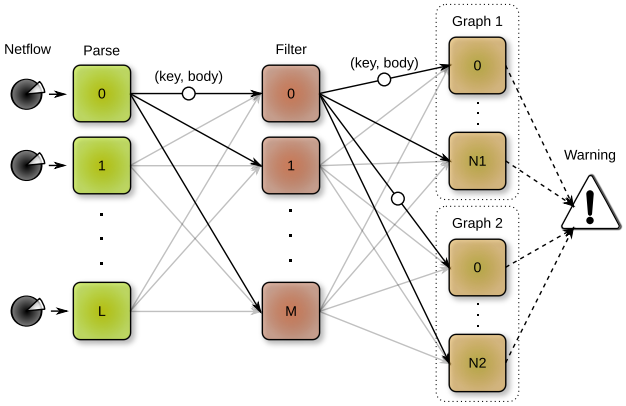


Fig. 3: Data processing architecture

Each network flow is processed as follows. First, the netflow data is read from a local storage device (it could also be received in real time from a router). The parse component transforms the IP addresses from their original string representation (i.e., "AAA.BBB.CCC.DDD") into an integer representing the

IP address[3] and constructs a message with 5 fields: sourceIP, source-port, destinationIP, destination-port, and transferred-bytes. The parse component sends this message to the filter component. It uses the sourceIP field of the message as the key. The filter component either forwards (using the same key) or discards the received message. This decision is based on various factors, like used ports and source and destination IPs. If the message is forwarded, it is forwarded to one process of every graph component (e.g., *Graph 1* and *Graph 2*). Finally, the graph components construct a *community graph* for each source IP. The filtering and community graph construction are described in detail below.

B. Filtering

The filter is an essential part of our analysis and its role is to remove irrelevant flow records and to reduce the amount of data that needs to be processed by the graph component. For example, commonly used search, news, social media, and entertainment web sites are used so frequently that they would appear with almost every community. Furthermore, any traffic that does not involve at least one community member is not relevant for the analysis and is filtered out. Other filtering actions can be chosen based on data volume and perceived threat vectors. For example, HTTP-traffic may be filtered to reduce data volume, but at the risk of missing attacks that use HTTP (port 80).

1: Example Filter algorithm

```
input : (src-IP, src-port, dst-IP, dst-port, transferred-bytes)
output: The same as the input, if not filtered
//collapse IP addresses
src-IP, dst-IP = collapse(src-IP), collapse(dst-IP);
//filter IPs of commonly used web sites
if src-IP ∈ whitelist then
    return ∅;
end
//filter web-accesses to community-members
if dst-IP ∈ community then
    if src-IP ∉ community then
        if src-port = 80 then
            return ∅;
        end
    end
end
//only forward if one of the IPs is in the community
if dst-IP ∈ community OR src-IP ∈ community then
    return (src-IP, src-port, dst-IP, dst-port, transferred bytes);
end
```

Algorithm 1 shows an example filter component that filters connections based on their ports, and source and destination IP addresses. First, the algorithm collapses IP addresses for an organization into one address. If, for example, an organization has the IP range from `141.1.0.0` to `141.85.255.255` and either the src-IP or dst-IP are within this range, it is set to `141.1.0.0`. We then discard every connection from IP addresses that are contained in the whitelist. Second, accesses to a community member's web-server are filtered. Finally, we only forward the event message if at least one of the connection end-points is contained in the community.

[3]We will continue calling this identifier an IP address to enforce the one to one connection between these numerical IDs and the IP addresses.

C. Community Graph

We build a fixed size (K) Community of Interest (COI) graph for each IP address that is received by the graph component. Essentially, we use a windowed top-K algorithm, as described in [3]. However, there are two significant differences in our implementation compared to [3]. First, our window is not based on a fixed time interval, but rather on the observed connections. This has the benefit that the COIs of IP addresses with many connections will be updated more often than of those with very few. Second, we introduce several COI views ($\{\mathcal{V}_1, \ldots, \mathcal{V}_n\}$) that use different methods to determine the weight of a connection. We can, for example, favor connections that transfer many bytes over those that only transfer a few by using the transferred bytes as the edge weight. Obviously, in this case we would not be able to detect attacks that transfer only a small set of data if these connections are dominated by large file transfers. Therefore, we define another view that uses the port numbers involved in security incidents to weight the edges (i.e., the more reported security incidents for a port, the larger the weight). Our system supports any number of such views running in parallel, as depicted in Figure 3 (with *Graph 1* implementing a different view than *Graph 2*).

Algorithm 2 shows how the COI is constructed in more detail. The algorithm uses two main data structures: a window that is used to collect recent data and a COI graph that stores the COI graph as seen from the beginning of the analysis run. We first add the received connection to the window. If more than 1000 connections have already been added, the window is merged with the COI graph. To this end, for each *IP* in the window, the weight of each edge is calculated, multiplied with a damping factor $1 - \theta$ and added to the weight in the COI, which is first multiplied with θ. Since $\theta = 0.85$, the influence of the new connections in the window is dampened. We also merge the port-mapping per destination-IP. It maps the source-port to the destination-port and a counter, counting how often this port-combination was used. Thereafter, the weights of all contacts in the COI that have not been observed during the current window are decayed by multiplying them with θ. To keep the COI at a maximum size of K, we remove the weakest links until the size of the COI is equal to K. Finally, the window and the counter are reset.

D. Generating Alarms

We showed above how the COI graph is constructed. Here, we provide two complementary algorithms to detect suspicious IP addresses.

The first, shown in Algorithm 3, is used to pre-filter all IP addresses that belong to a community. However, if a computer inside the community is compromised, we still want it to be checked further. To this end, we iterate over all connections in

2: Example Community graph construction

```
input : (src-IP, src-port, dst-IP, dst-port, transferred-bytes), s =
        State[src-IP], F
output: None

//Save connection in window
s.window[dst-IP].transferred_bytes += transferred-bytes;
s.window[dst-IP].port_map[src-port][dst-port]++;
s.counter++;
//Merge window into topK after 1000 events
if s.counter > 1000 then
    foreach IP ∈ s.window do
        //θ has a value of 0.85 in our analysis.
        s.topk[IP].weight = 1 - θ * V(s.window[IP])
                          + θ * s.topk[IP].weight;
        //Merge the window's port map with the top-k's
        foreach {source-port, dest-port} ∈
        s.window[IP].port_map do
            s.topk[IP].port_map[source-port][dest-port] +=
            s.window[IP].port_map[source-port][dest-port];
        end
    end
    //Decay weight of old connections
    foreach IP ∉ s.window do
        s.topk[IP].weight = θ * s.topk[IP].weight;
    end
    //Remove the weakest links
    while size(s.topk) > K do
        remove_weakest_link_from(s.topk);
    end

    s.window = ∅;
    s.counter = 0;
end
```

3: Suspicious IP detection (1)

```
input : IP, community, s = State[IP]
output: IP, if suspicious; ∅, if not

//blacklisted IPs are always suspicious
if IP ∈ blacklist then
    return IP;
end

//check if IP is in the community
if IP ∈ community then
    //iterate over all of IP's connections
    foreach conn ∈ s.topk do
        //iterate over all ports of one connection
        foreach p ∈ s.topk[conn].port_map do
            //check if src_port and dst_port are suspicious
            if is_suspicious(src_port, dst_port) then
                return IP;
            end
        end
    end

    //no strange ports -¿ skip
    return ∅;
end

//not in community -¿ check
return IP;
```

4: Suspicious IP detection (2)

```
input : IP, community, min_cnt, min_part, s = State[IP]
output: Alarm

//check if top-K connections of this IP are in the community
  often enough
cnt = count_community(community, s.topk);
part = cnt / size(s.topk);

if IP ∉ blacklist then
    if cnt ≤ min_cnt OR part ≤ min_part then
        return false;
    end
end

return true;
```

the IP's top-K and check each pair of ports. The pairs of ports, considered suspicious, are specified using a configuration file.

We call Algorithm 4 for all IP addresses returned by Algorithm 3. It assures that (1) only those IP addresses that connected to at least `min_cnt` members of the community will be reported and (2) that the connections to the community make at least `min_part` percent of all the connections of the current IP address.

The detection algorithm can be run either for all IP addresses at once or individually for each IP address. Therefore, it is possible to provide different detection latencies. For example, to detect a suspicious IP address the earliest possible, the algorithm must be executed as soon as a message is received for its source-IP's top-K. If this is not necessary, the algorithm can be run for all top-Ks in one graph process at any desired interval.

The generated alarms can be emailed to the system administrators in the affected organizations or posted on a security dashboard. The reports contain the complete top-K for each suspicious IP address, including the port mappings.

IV. Evaluation

A. Input data and general setup

We currently run the experiment on a per-day basis. This means we fetch the netflow entries of the last 24 hours and run our analysis. We do not carry any state from one daily run to the next. In principle, we could leave the system running continuously or checkpoint the graph component and re-initiate its state on the next day. However, we found it useful to start with a clean system every day since this makes it easier to reason about the impact of changes in the community and white lists.

Moreover, we introduced the concept of different views in June 2011. Since then, we use three different views: one that weighs the bytes transferred, another that weighs the number of connections made, and the last one that weighs the security risk for the ports used (as described in Section III). For any measurements that were conducted before this date, we only used the view based on the bytes transferred.

Our input data-set is heavily sampled netflow from an ISP. In the first step, we remove all unimportant fields, leaving only the source-IP, destination-IP, source-port, destination-port, and the number of transferred bytes. This sums up to roughly

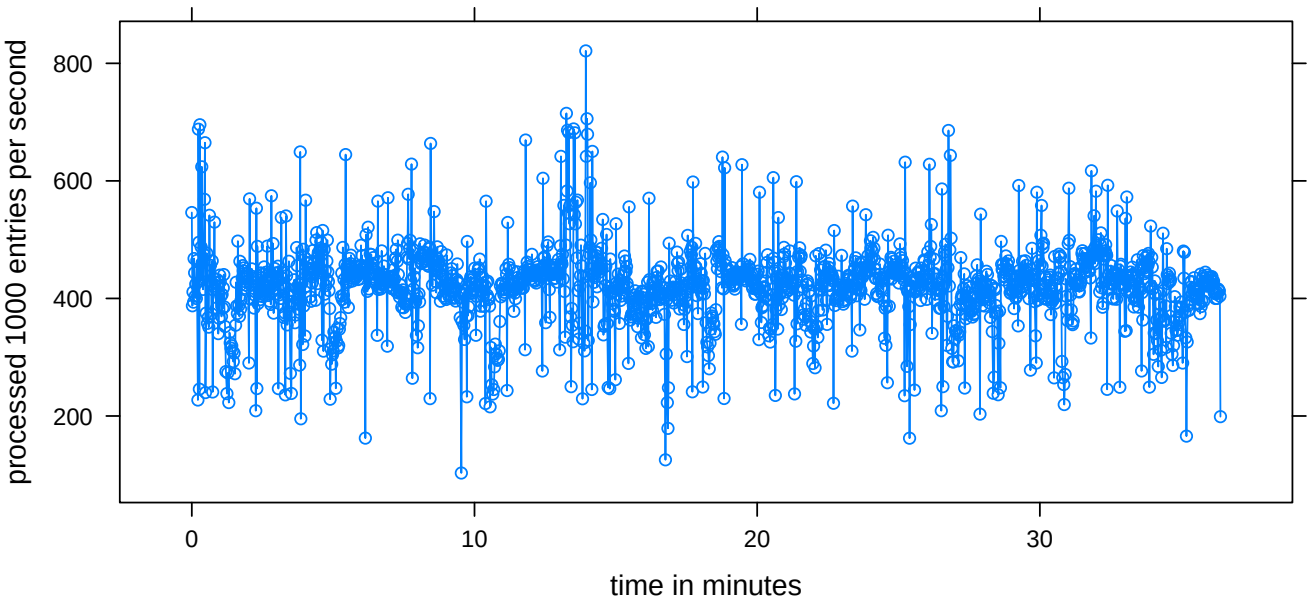


Fig. 4: Processed netflow-entries per second over the complete analysis execution

50GB of processed netflow per day.

The community lists define a community with the IP address ranges of all its members and each community is stored in a separate file (the white list is simply a "special" community). For example, if we wanted to add "TU-Dresden" to a "universities community" we would add the following line into the corresponding file:

```
141.1.0.0 - 141.85.255.255 TU.DRESDEN.DE
```

If a company or institution has more than one IP address range assigned, we can simply add each range as a separate entry. Moreover, an entry in one community is allowed to be a member in other communities as well.

B. Performance

We implemented the parse, filter, and graph components on top of StreamMine [12], a highly scalable stream processing system. While StreamMine supports scaling to hundreds of physical machines, a scalability and performance evaluation involving multiple machines is out of the scope of this paper. Therefore, we only used a single machine with 24GB of RAM and 16 processing cores for the analysis. For the top-K algorithm we used a value of 100 for K.

Figure 4 shows the read-throughput of the parse component of one such run in which we processed one day of netflow data (using only one view). The measurement was taken every second throughout the whole run. The parse component can read around 400,000 netflow entries per second with this single machine. Each entry is converted into a message and sent to the filter component. The filter component discards a large fraction of these messages and only send around one in a hundred of the incoming messages to the graph component. Naturally, the read throughput varies over time, since the amount of processing that needs to be done in the system depends heavily on the content of the input data. However, it is important to note that the mean throughput stays constant, i.e., the system performance does not decline with time as more graphs are added.

In the experiments reported in this paper, the filter component uses 13 of the available cores, since it has to filter the 400,000 netflow entries arriving every second. The graph component uses only one core since the amount of data it has to process is only a fraction of the data the filter receives. Note that even if one would assign more processing resources (i.e., cores) to the graph component, it would still be impossible to process unfiltered traffic (i.e., system without the filter component)— the system would simply run out of memory. The parse component uses the remaining two cores for reading the input files and parsing their contents.

To avoid queuing, StreamMine uses the TCP back-pressure mechanism on the network-connections. Hence, if a message cannot be processed by the filter component because all its threads are already busy processing other messages, the parse component will eventually stop sending new messages (the TCP send blocks if messages are not read fast enough on the other side). This will eventually lead to the parse component not reading any new netflow entries, because all its threads are blocked trying to send messages.

Figure 5 shows the size of the daily alarm report (= number of suspicious IPs communicating with the community) and community sizes (approximately the number of member organizations) over time for several months. The size of the

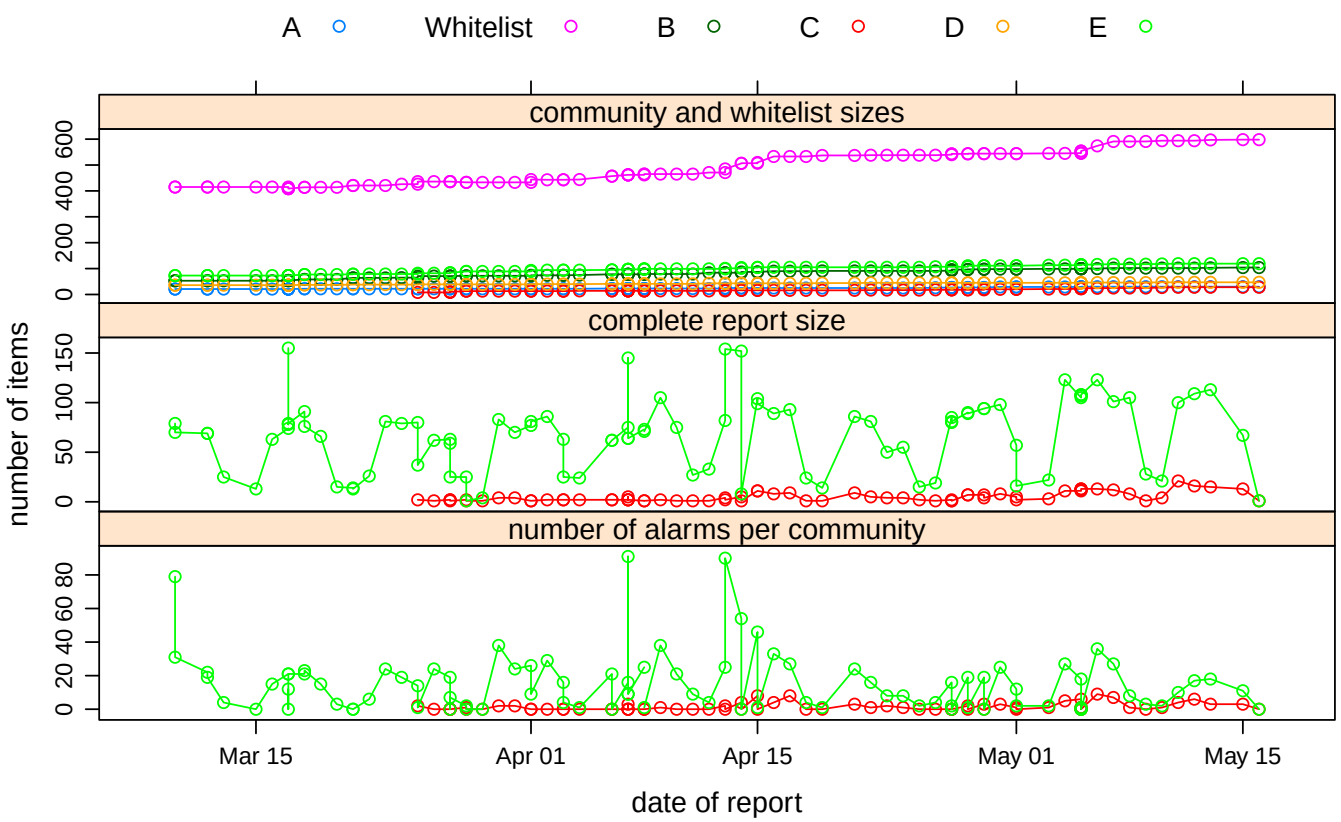


Fig. 5: Community and alarm sizes over time

alarm report is subject to a weekly pattern with larger sizes for weekdays (for alarm reports produced from Tuesday to Saturday) and smaller for weekend traffic. The community lists and the white list were updated manually on a daily basis. Given a fixed community, the community list would typically stay relatively fixed but in our case we occasionally identified additional community members. For the alarm reports, we only plot the report sizes for communities E and C. We did not generate reports for the other communities because (1) we found E and C to be the most interesting ones and (2) because of time-constraints as we need to scan the reports manually for attacks and new members of the community or white lists. It is natural that the reports, especially initially, contain a number of false positives. Some of them will be new community members that have to be added to the community list, while others are companies and organizations that can be added to the white list. The white list is used to filter out trusted traffic, i.e., from well known search engines, entertainment web sites, social media, popular CDNs, banking, government services, etc.

In an actual usage of the system, the system administrators analyzing the alarm reports would also add other known "good" IP addresses to the white list to prevent them from being reported daily. Lacking such domain knowledge, our experiments used the white list conservatively. The bottom plot approximates the size of the daily alarm report under real usage scenario where suspected IP addresses are processed daily and either added to the white list or the suspect communication is stopped (e.g., clean up infected machine, add firewall rules). This alarm size is approximated simply by only listing the IP addresses that have not been reported before.

V. Case studies

While we do not typically know the ground truth, we have observed a number of suspicious cases in our analysis. In this section, we outline some of these examples.

A. Case 1

Table I shows an anonymized part of the report, generated for the netflow on May 13th, 2011. The report was obtained using the view based on the number of bytes transferred. It depicts the anonymized source-IP address (X.Y.Z.W) and the communities it was connected to, which ports were used (to help identify the application or service used), and a measure of the frequency of communication—the "Occurence" field indicates how often this connection was observed in the COI. In the actual report, the IP address and the exact community member are visible, of course.

In the next step, we usually use the *whois* service, to determine to whom the IP address belongs. This way, we may also find new members of the community by looking up the company names, displayed in the *whois* information. For this

IP Address	Src Port	Community	Dst port	Occurrences
X.Y.Z.W	6000	E	1433	2
X.Y.Z.W	6000	E	1433	2
X.Y.Z.W	6000	E	1433	1
X.Y.Z.W	6000	E,C	1433	1
X.Y.Z.W	6000	B	1433	1
X.Y.Z.W	6000	E,C	1433	1

TABLE I: Anonymized report-snippet (port-mapping) from May 13th, 2011

particular example, the only information we could get, was that it belongs to an Asian ISP. Since the IP address likely does not belong to a company that the community members would typically collaborate with, we have a closer look at the ports being used. We assume that the lower port number (1433) belongs to the server and the higher port-number to the client (6000). Figures 6 and 7 show the output of the "SANS Internet Storm Center" web-site[4] related to port 1433. The web-site shows the services that usually run on these ports—in this example, "Microsoft-SQL-Server". The SANS reports indicate many potential vulnerabilities, which may be used, for example, to steal data.

Unfortunately, this is usually everything we are able to derive from the netflow alone. While we consider this to be a potential attack, final certainty could only be provided by the system administrators of the individual companies, given they have deeper knowledge about legitimate communication connections of each organization and access to lower-level logs on the targeted machines.

B. Case 2

Table II shows a summary of the COI of another anonymized IP address for August 8th, 2011. It shows the IP address, each community and two numbers. The report was generated using the view based on the security risk of used ports. The first number is simply a count of how many members of the current community had an entry in the COI of this IP address. The second number shows how often the IP address connected to other IP addresses that are in none of the communities. We stated in Section II that this number is a good indicator of the severity and specificity of an attack. Here, it is relatively low, which leads to the assumption that the connections were not driven by a brute-force or port-scan-like technique.

To verify this intuition, Table III shows the used ports for each community member individually. In contrast to the previous example, the source port is not constant anymore but seems to be chosen randomly. The destination port, however, is constant 445. Port 445 is usually used by "Win2k+ Server Message Block". Note that every connection only appeared once in the netflow. This either means there was in fact just one connection being used or the attempt to connect failed.

In the next step, we use again the *whois* service, to determine that the IP address belongs to an European ISP. However,

[4]http://isc.sans.org

IP Address	Community	# in Top-K	# outside Community
X.Y.Z.W	A	0	42
X.Y.Z.W	B	0	42
X.Y.Z.W	C	1	42
X.Y.Z.W	D	0	42
X.Y.Z.W	E	1	42
X.Y.Z.W	F	6	42

TABLE II: Anonymized report-overview-snippet from August 8th, 2011. The last two columns contain the following numbers: (1) Number of members of the current community which had an entry in the COI of the current IP address and (2) number of connections to non-community members after the first connection to a community-member.

IP Address	Src Port	Community	Dst port	Occurrences
X.Y.Z.W	4798	F	445	1
X.Y.Z.W	1238	F	445	1
X.Y.Z.W	1256	F	445	1
X.Y.Z.W	1682	F	445	1
X.Y.Z.W	3143	C,E,F	445	1
X.Y.Z.W	4243	F	445	1

TABLE III: Anonymized report-snippet from August 8th, 2011

it is not clear if this address belongs to a community member. An attempt to *ping* the address did not succeed. A query to "SANS Internet Storm Center" (Figure 8) shows a long list of reports about worms using this port with the famous "Conficker" being one of them.

As with the previous example, we cannot determine if this case is a true attack. To this end, we would need the help of the system administrators of the various community members who have access to the log-files of the corresponding machines. However, there are two interesting points concerning this IP address. First, there are only a total of 69 entries in the netflow, where this address is the source of communication. Second, all connections transfer only a very small amount of data—around 60 bytes each. Even in total, this only sums up to several kilo bytes. Therefore, this address only appears in the ports view and not in the other views that consider either the number of bytes or connections. Hence, an administrator would need to set the detection threshold very low to see an alarm concerning this address.

C. Case 3

In contrast to the previous two cases, this case is not an attack. It occurred in all views and if one only looks at the report (an excerpt is shown in Table IV), it is not immediately clear what service is being used since the address seems to be using random ports on both ends of the communication. The query to *whois* does also not reveal any useful information, except that the address belongs to a US ISP.

However, looking at the connections with IP addresses outside of the communities provides a hint that this is not targeted against any of our specified communities as shown in

User Comment

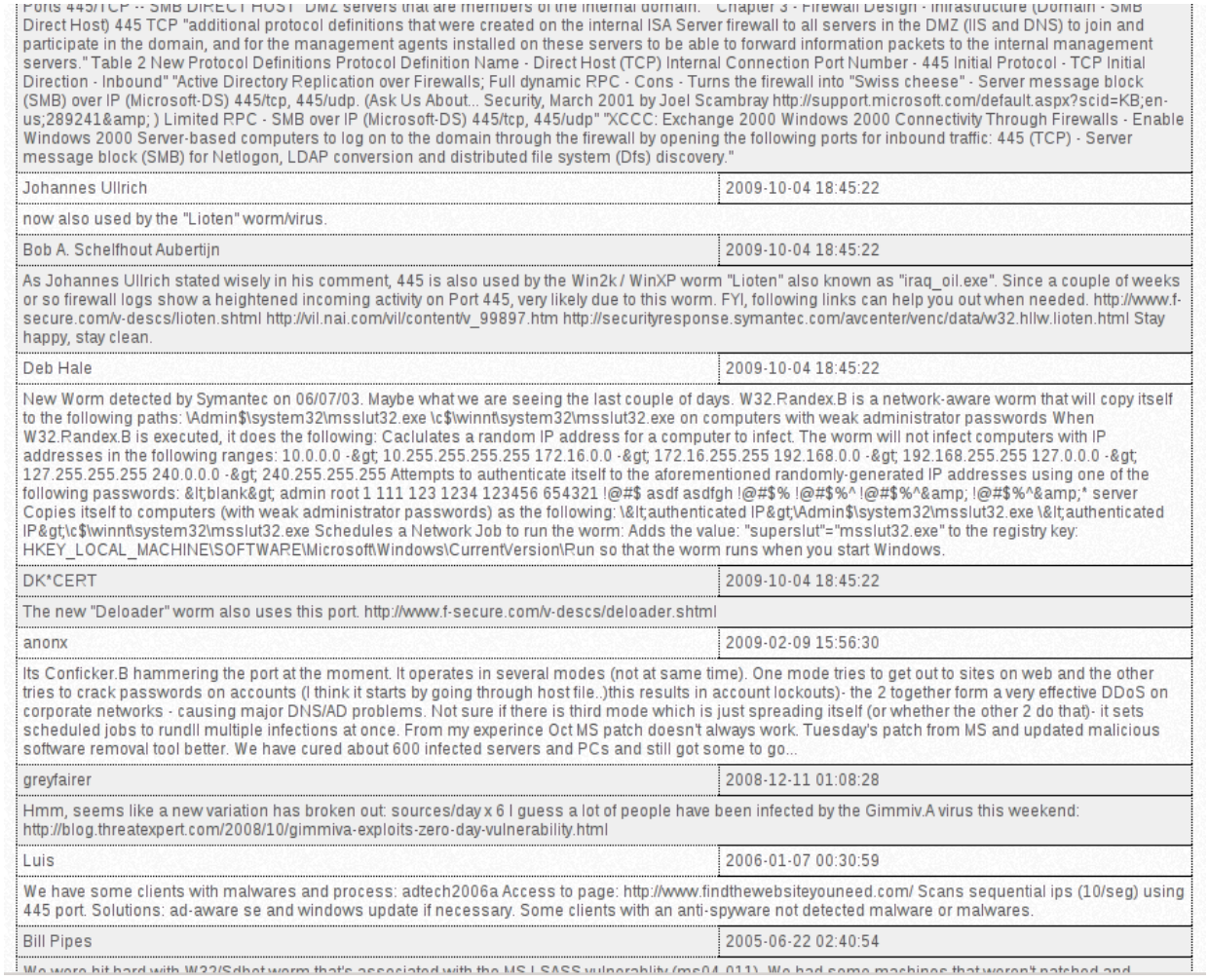
Submitted By	Date
Comment	
Marcus H. Sachs, SANS Institute	2003-10-10 00:50:59
SANS Top-20 Entry: W2 Microsoft SQL Server (MSSQL) http://isc.sans.org/top20.html#w2 The Microsoft SQL Server (MSSQL) contains several serious vulnerabilities that allow remote attackers to obtain sensitive information, alter database content, compromise SQL servers, and, in some configurations, compromise server hosts. MSSQL vulnerabilities are well-publicized and actively under attack. Two recent MSSQL worms in May 2002 and January 2003 exploited several known MSSQL flaws. Hosts compromised by these worms generate a damaging level of network traffic when they scan for other vulnerable hosts.	
Johannes Ullrich	2002-10-10 17:21:35
Port 1433 is used by Microsoft SQL Server. SQLSnake is one worm taking advantage of SQL Server installs without password. As SQL Server is able to run batch files and command line programs, it can be used to download and install malware. Basic Protection: Use good passwords for all SQL Server accounts.	

Fig. 6: Screenshot of "`http://isc.sans.org/port.html?port=1433`" from September 8th, 2011

CVE Links

CVE #	Description
CVE-1999-287	"Vulnerability in the Wguest CGI program."
CVE-2000-1081	"The xp_displayparamstmt function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1082	"The xp_enumresultset function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1083	"The xp_showcolv function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1084	"The xp_updatecolvbm function in SQL Server and Microsoft SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1085	"The xp_peekqueue function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1086	"The xp_printstatements function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2000-1088	"The xp_SetSQLSecurity function in Microsoft SQL Server 2000 and SQL Server Desktop Engine (MSDE) does not properly restrict the length of a buffer before calling the srv_paraminfo function in the SQL Server API for Extended Stored Procedures (XP)
CVE-2001-542	"Buffer overflows in Microsoft SQL Server 7.0 and 2000 allow attackers with access to SQL Server to execute arbitrary code through the functions (1) raiserror
CVE-2002-642	"The registry key containing the SQL Server service account information in Microsoft SQL Server 2000

Fig. 7: Screenshot of "`http://isc.sans.org/port.html?port=1433`" from September 8th, 2011


Ports 445/TCP -- SMB DIRECT HOST "DMZ servers that are members of the internal domain." "Chapter 3 - Firewall Design - Infrastructure (Domain - SMB Direct Host) 445 TCP "additional protocol definitions that were created on the internal ISA Server firewall to all servers in the DMZ (IIS and DNS) to join and participate in the domain, and for the management agents installed on these servers to be able to forward information packets to the internal management servers." Table 2 New Protocol Definitions Protocol Definition Name - Direct Host (TCP) Internal Connection Port Number - 445 Initial Protocol - TCP Initial Direction - Inbound" "Active Directory Replication over Firewalls; Full dynamic RPC - Cons - Turns the firewall into "Swiss cheese" - Server message block (SMB) over IP (Microsoft-DS) 445/tcp, 445/udp. (Ask Us About... Security, March 2001 by Joel Scambray http://support.microsoft.com/default.aspx?scid=KB;en-us;289241&) Limited RPC - SMB over IP (Microsoft-DS) 445/tcp, 445/udp" "XCCC: Exchange 2000 Windows 2000 Connectivity Through Firewalls - Enable Windows 2000 Server-based computers to log on to the domain through the firewall by opening the following ports for inbound traffic: 445 (TCP) - Server message block (SMB) for Netlogon, LDAP conversion and distributed file system (Dfs) discovery."

Johannes Ullrich — 2009-10-04 18:45:22

now also used by the "Lioten" worm/virus.

Bob A. Schelfhout Aubertijn — 2009-10-04 18:45:22

As Johannes Ullrich stated wisely in his comment, 445 is also used by the Win2k / WinXP worm "Lioten" also known as "iraq_oil.exe". Since a couple of weeks or so firewall logs show a heightened incoming activity on Port 445, very likely due to this worm. FYI, following links can help you out when needed. http://www.f-secure.com/v-descs/lioten.shtml http://vil.nai.com/vil/content/v_99897.htm http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.lioten.html Stay happy, stay clean.

Deb Hale — 2009-10-04 18:45:22

New Worm detected by Symantec on 06/07/03. Maybe what we are seeing the last couple of days. W32.Randex.B is a network-aware worm that will copy itself to the following paths: \Admin$\system32\msslut32.exe \c$\winnt\system32\msslut32.exe on computers with weak administrator passwords When W32.Randex.B is executed, it does the following: Caclulates a random IP address for a computer to infect. The worm will not infect computers with IP addresses in the following ranges: 10.0.0.0 -> 10.255.255.255.255 172.16.0.0 -> 172.16.255.255 192.168.0.0 -> 192.168.255.255 127.0.0.0 -> 127.255.255.255 240.0.0.0 -> 240.255.255.255 Attempts to authenticate itself to the aforementioned randomly-generated IP addresses using one of the following passwords: <blank> admin root 1 111 123 1234 123456 654321 !@#$ asdf asdfgh !@#$% !@#$%^ !@#$%^& !@#$%^&* server Copies itself to computers (with weak administrator passwords) as the following: \<authenticated IP>\Admin$\system32\msslut32.exe \<authenticated IP>\c$\winnt\system32\msslut32.exe Schedules a Network Job to run the worm: Adds the value: "superslut"="msslut32.exe" to the registry key: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run so that the worm runs when you start Windows.

DK*CERT — 2009-10-04 18:45:22

The new "Deloader" worm also uses this port. http://www.f-secure.com/v-descs/deloader.shtml

anonx — 2009-02-09 15:56:30

Its Conficker.B hammering the port at the moment. It operates in several modes (not at same time). One mode tries to get out to sites on web and the other tries to crack passwords on accounts (I think it starts by going through host file..)this results in account lockouts)- the 2 together form a very effective DDoS on corporate networks - causing major DNS/AD problems. Not sure if there is third mode which is just spreading itself (or whether the other 2 do that)- it sets scheduled jobs to rundll multiple infections at once. From my experince Oct MS patch doesn't always work. Tuesday's patch from MS and updated malicious software removal tool better. We have cured about 600 infected servers and PCs and still got some to go...

greyfairer — 2008-12-11 01:08:28

Hmm, seems like a new variation has broken out: sources/day x 6 I guess a lot of people have been infected by the Gimmiv.A virus this weekend: http://blog.threatexpert.com/2008/10/gimmiva-exploits-zero-day-vulnerability.html

Luis — 2006-01-07 00:30:59

We have some clients with malwares and process: adtech2006a Access to page: http://www.findthewebsiteyouneed.com/ Scans sequential ips (10/seg) using 445 port. Solutions: ad-aware se and windows update if necessary. Some clients with an anti-spyware not detected malware or malwares.

Bill Pipes — 2005-06-22 02:40:54

We were hit hard with W32/Sdbot worm that's associated with the MS LSASS vulnerability (ms04-011). We had some machines that weren't patched and

Fig. 8: Screenshot of "`http://isc.sans.org/port.html?port=445`" from September 8th, 2011

IP Address	Src Port	Community	Dst port	Occurrences
X.Y.Z.W	13397	B	38426	1
X.Y.Z.W	41748	F	41387	1
X.Y.Z.W	49534	C	23068	1
X.Y.Z.W	16249	C	22654	1
X.Y.Z.W	29167	C	43183	2
X.Y.Z.W	20	F	7205	4
...	...	...	...	...

TABLE IV: Anonymized report-snippet from August 8th, 2011

IP Address	Community	# in Top-K	# outside Community
X.Y.Z.W	A	0	14250
X.Y.Z.W	B	2	14250
X.Y.Z.W	C	1	14250
X.Y.Z.W	D	0	14250
X.Y.Z.W	E	0	14250
X.Y.Z.W	F	6	14250

TABLE V: Anonymized report-overview-snippet from August 8th, 2011. The last two columns contain the following numbers: (1) Number of members of the current community which had an entry in the COI of the current IP address and (2) number of connections to non-community members after the first connection to a community member.

Table V. Moreover, the use of port 20 (the last line in Table IV) gives a hint that at least some part of the communication involved anonymous ftp, which uses port 20 to initiate the connection but uses random ports thereafter. Finally, using an ftp-client (i.e., a web-browser) revealed indeed that this is simply an ftp server hosting software updates. As a result of this analysis, we added the address to the white list.

D. Building Communities

In real use of the system, the community members might be known a priori and even stay relatively fixed. However, in our case we built the community lists incrementally by identified new community members based on the COIs gener-

ated. Specifically, we assumed that members of a community exchange information with one another and often the data exchange is encrypted. Therefore, we focused on new IP addresses that used the https-port (443) for communication. However, a certain minimal set of known members is needed before reports can be generated. This set should be as large as possible for two reasons. First, the likelihood that an unknown address that belongs to the community (and thus, should be added) connects to one or more entries of a large set of members is higher than if the set contains only very few entries. Second, if the set is large, one can set the reporting threshold higher and reduce the amount of noise.

Building a community this way is a task that lasts for weeks, depending on how much communication is observed between the individual members and how large the community is initially. We start by adding the new community to the list of communities. With every subsequent report, we scan for new members and add them to the corresponding lists. This way, the community grows every day, and with it the likelihood of finding any missing members. The community stabilizes eventually with fewer and fewer new members per day.

VI. Related work

A number of tools and techniques have been developed to process and visualize netflow data(see [17] for a survey). Netflow processing tools include OSU flow-tools [16], SiLK [7], and Nfdump[5]. In addition to command line tools, numerous graphical user interfaces exist to visualize and query network activity, including NTOP[6], Nfsen [9], NfSight [1], VisFlowConnect [20], FlowScan [14], NetPY [2], FloVis [18], VIAssist [5], and NFlowVis [6]. While visualization tools allow the users to view the netflow data from different perspectives to locate suspicious activity, our approach analyzes the data and produces small number of meaningful alarms each day. Also, our focus on communities allows us to detect attacks and suspicious behavior that is focused on a potentially small community, but would not show significantly on a global scale.

Detection of similar communication behavior in multiple hosts has been used previously to raise suspicion that hosts with the correlated behavior may be members of the same botnet. For example, [21] uses netflow data to identify sets of suspicious hosts and then uses host level information (collected on each host by a local monitor) to confirm or reject the suspicions. However, detection of botnets is simplified by the fact that the bots typically act in unison (e.g., start spamming or DDoS attack against a target at the same time). Indeed, much of the work in this area (e.g., BotMiner [8]) specifically build detection mechanisms based on the assumptions of the communication behavior required for a botnet. Furthermore, to our knowledge, prior work is limited to detecting similar behavior within one organization.

The concept of using a community to help detect security events has been used in the past. For example, the Ensemble [15] system detects applications that have been hijacked by using the idea of a trusted community of users contributing system-call level local profiles of an application to a common merging engine. The merging engine generates a global profile that can be used to detect or prevent anomalies in application behavior at each end-host in real time. A similar concept of collaborative learning for security [13] is applied to automatically generate a patch to the problematic software without affecting application functionality. PeerPressure [19] automatically detects and troubleshoots misconfigurations by assuming that most users in the community have the correct configuration. Cooperative Bug Isolation [11] leverages the community to do statistical debugging based on the feedback data automatically generated by community users. Vigilante [4] apply the community concept for containment of Internet worms by community members running detection engines on their machines, where the detection engines distribute attack signatures to other community members when a machine is infected.

[5]http://nfdump.sourceforge.net

[6]http://www.ntop.org

VII. Conclusions

In this paper, we have presented a community-based analysis and alerting technique for detecting small-footprint attacks targeting communities of interest for attackers such as financial institutions, e-commerce web site, or the electricity generation and distribution infrastructure. By comparing communication behavior across the member organizations in the community, it is possible to detect suspect behavior that may fall below detection thresholds at individual member organizations. A white list can be used to avoid repeating false positives. We have implemented the analysis algorithm in a scaleable distributed architecture that can process large volumes of netflow data efficiently.

References

[1] R. Berthier, M. Cukier, M. Hiltunen, D. Kormann, G. Vesonder, and D. Sheleheda. Nfsight: netflow-based network awareness tool. In *Proceedings of the 24th USENIX LISA*, 2010.

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

[3] Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. In Frank Hoffmann, David Hand, Niall Adams, Douglas Fisher, and Gabriela Guimaraes, editors, *Advances in Intelligent Data Analysis*, volume 2189 of *Lecture Notes in Computer Science*, pages 105–114. Springer Berlin / Heidelberg, 2001. 10.1007/3-540-44816-0_11.

[4] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: End-to-end containment of internet worms. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2005.

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

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

[7] C. Gates, M. Collins, M. Duggan, A. Kompanek, and M. Thomas. More NetFlow tools: For performance and security. In *Proc. 18th USENIX LISA*, pages 121–132, 2004.

[8] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the USENIX Security Conference*, 2008.

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

[10] V. Igure, S. Laughter, and R. Williams. Security issues in scada networks. *Computers & Security*, 25(7):498 – 506, 2006.

[11] Ben Liblit. *Cooperative bug isolation: winning thesis of the 2005 ACM doctoral dissertation competition*. Springer-Verlag, Berlin, Heidelberg, 2007.

[12] André Martin, Thomas Knauth, Stephan Creutz, Diogo Becker de Brum, Stefan Weigert, Andrey Brito, and Christof Fetzer. Low-overhead fault tolerance for high-throughput data processing systems. In *ICDCS '11: Proceedings of the 2011 31st IEEE International Conference on Distributed Computing Systems*, page TBD, Los Alamitos, CA, USA, June 2011. IEEE Computer Society.

[13] J. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sullivan, W.-F. Wong, Y. Zibin, M. Ernst, and M. Rinard. Self-defending software: Automatically patching security vulnerabilities. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[14] D. Plonka. Flowscan: A Network Traffic Flow Reporting and Visualization Tool. In *Proc. 14th USENIX LISA*, pages 305–318, 2000.

[15] F. Qian, Z. Qian, Z. M. Mao, and A. Prakash. Ensemble: Community-based anomaly detection for popular applications. *5th International ICST Conference on Security and Privacy in Communication Networks*, May 2009.

[16] S. Romig, M. Fullmer, and R. Luman. The OSU flow-tools package and CISCO NetFlow logs. In *Proc. 14th USENIX LISA*, pages 291–304, 2000.

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

[18] T. Taylor, D. Paterson, J. Glanfield, C. Gates, S. Brooks, and J. McHugh. FloVis: Flow Visualization System. In *Proc. Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH)*, pages 186–198, 2009.

[19] H. Wang, J. Platt, Y. Chen, R. Zhang, and Y.-M. Wang. Automatic misconfiguration troubleshooting with peerpressure. In *In OSDI*, pages 245–258, 2004.

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

[21] Y. Zeng, X. Hu, and K. Shin. Detection of botnets using combined host- and network-level information. In *Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010.