CoreFlow: Enriching Bro security events using network traffic monitoring data

Koning, R.; Buraglio, N.; de Laat, C.; Grosso, P.

DOI
10.1016/j.future.2017.04.017

Publication date
2018

Document Version
Final published version

Published in
Future Generation Computer Systems

License
Article 25fa Dutch Copyright Act (https://www.openaccess.nl/en/in-the-netherlands/you-share-we-take-care)

Citation for published version (APA):
CoreFlow: Enriching Bro security events using network traffic monitoring data

Ralph Koning\textsuperscript{a,b,*}, Nick Buraglio\textsuperscript{b}, Cees de Laat\textsuperscript{a,b}, Paola Grosso\textsuperscript{a}

\textsuperscript{a} Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands
\textsuperscript{b} Energy Sciences Network, Lawrence Berkeley Lab, Berkeley, CA, USA

HIGHLIGHTS

• Enriching IDS data with NetFlow information gives a better view of an attack.
• CoreFlow ingests data from the Bro IDS and augments this with flow data from the devices in the network.
• The augmented information can be the starting point for sophisticated countermeasures close to the origin.
• The CoreFlow prototype is tested in the ESnet network, using inputs from 3 Bro systems and more than 50 routers.

ARTICLE INFO

Article history:
Received 20 September 2016
Received in revised form
8 April 2017
Accepted 8 April 2017
Available online 13 April 2017

Keywords:
Security
Network
IDS
Netflow
Flow
Detection
IPFIX
DDoS
Carrier networks
Transit networks

ABSTRACT

Attacks against network infrastructures can be detected by Intrusion Detection Systems (IDS). Still reaction to these events are often limited by the lack of larger contextual information in which they occurred. In this paper we present CoreFlow, a framework for the correlation and enrichment of IDS data with network flow information. CoreFlow ingests data from the Bro IDS and augments this with flow data from the devices in the network. By doing this the network providers are able to reconstruct more precisely the route followed by the malicious flows. This enables them to devise tailored countermeasures, e.g. blocking close to the source of the attack. We tested the initial CoreFlow prototype in the ESnet network, using inputs from 3 Bro systems and more than 50 routers.

1. Introduction

As society becomes more reliant on cyber-infrastructures and computer networks, securing this infrastructure becomes increasingly more important. Large scale cyber attacks might be directed toward critical infrastructure components such as the DNS root servers \cite{1}; against commercial network providers such as end-user ISPs \cite{2}; or against educational and research networks serving academia \cite{3}. All these attacks show how fragile computer networks can be.

Given these continuous attacks carefully monitoring Internet systems and components for suspicious activities becomes imperative. There are many developments in monitoring and intrusion detection systems (IDS) that enable them to trigger alerts when such activities are present \cite{4,5}. When such an episode occurs it is the responsibility of the security and incident response teams that monitor this information to further investigate these events; this often requires them to look up and combine information from multiple sources to make a more informed judgment. In this paper we describe CoreFlow, a prototype framework to enrich IDS data with network flow data; this enhancement provides more context to security events and this in turn creates more targeted alerts and more advanced responses. This is in particular important for car-

http://dx.doi.org/10.1016/j.future.2017.04.017
0167-739X/© 2017 Elsevier B.V. All rights reserved.
carrier networks that due to their characteristics require to correlate information coming from distant elements in the network.

In Section 2 we will briefly review the different challenges carrier networks face to secure their networks, and we introduce ESnet, the network where we tested CoreFlow; in Section 3 we discuss the information sources used in this research. Section 4 and Section 5 describe CoreFlow architecture and implementation. In Section 6 we reflect on the functionality of the framework and discuss what can be improved. Section 7 covers related work and Section 8 contains the conclusion and future work.

2. Carrier network security

Carrier networks present different challenges than enterprise or campus networks due to their different characteristics. In Table 1 we list five aspects in which carrier networks differ from enterprise and campus networks when we consider them from a security perspective: external connectivity, application security, restrictions and policies, impact of countermeasures and network capacity. For example, in carrier networks it is unfeasible to run all traffic through a single or a set of security appliance devices due to very high data rates, as well as the large or numerous data flows and multiple ingress and egress points. Additionally, carrier networks are often tasked with adhering to network neutrality laws or policies which prevent filtering or otherwise altering traffic in any way other than to protect the infrastructure of the network.

2.1. ESnet

Our CoreFlow development and validation has taken place at ESnet. ESnet is a national research and education network (NREN) that interconnects multiple national labs in the US to each other, to supercomputing facilities, as well as other research networks in the world. Fig. 1 shows the topology of the ESnet backbone network that spans the US and a part of Europe. The backbone consists mainly of 100 Gbps links and allows sites to connect to ESnet at various speeds.

ESnet primarily transmits data within the connected institutions and to other connected research facilities and resources and therefore operates as a carrier or transit network for scientific traffic.

Given their architectures NRENs like ESnet fall in the category of carrier or transit networks and are therefore a suitable testing ground for CoreFlow.

3. Information sources

Different information sources can be used to identify and counteract network attacks.

IDS systems are able to perform in depth inspection of packets to detect security problems, yet they only have a limited end perspective of the network. NetFlow and other flow-based tools provide detailed network traffic information. This information can be collected from all routers over the entire extent of the network and can provide a global view and the origin of the traffic that transmits a network. Correlating data from both of these information sources may give a more detailed view on the origin of the malicious traffic and thus provide more context to act upon, this detailed multi-source view makes countermeasure less sensitive to spoofed traffic information.

This is particularly useful when an attack is volume based such as in the case of a Distributed Denial of Service (DDoS) attack. In this case instead of blocking traffic at the end systems, it may be preferable to prevent the malicious data from entering the network at the entry point, or contact an upstream provided to block the specific traffic. This reaction at the network edges is complicated by the fact that this attack traffic is often spoofed to cover its origin, causing it to have another entry point into the network than presented. Since the addressing information cannot be relied upon, one has to determine the origin by checking presence of this traffic pattern on all routers on the path.

In our development of CoreFlow we relied specifically on Bro data, on NetFlow information, on Splunk for data aggregation and on Route Explorer for path calculation.

3.1. Bro

Bro [6] is an open source network analysis framework developed at the International Computer Science Institute in Berkeley, CA and the National Center for supercomputing Applications in Urbana–Champaign, IL. Bro focuses on network security monitoring and offers functionality beyond traditional intrusion detection systems. It includes an event engine and a policy module in which one can write custom policies. Due to clustering capabilities, Bro can scale to 100 Gbps links [7]. Bro has an extensive policy system that can be used to react on or to trigger events. Events can thus also be correlated within the Bro framework itself as part of a policy. To implement policies Bro uses its own scripting language. This language is limited but it could in principle be used to implement the CoreFlow functionality as a plugin in the C language. This would require knowledge of two languages, the Bro domain specific language and C; for this reason it seemed more practical to us to implement CoreFlow as a stand-alone system using Python.

Building a stand-alone system makes CoreFlow more flexible since we are able to use multiple input sources or replace out Bro in favor of a different IDS. Python is a widely used and easy to learn language which became very popular among data scientists, therefore by using it we try to lower accessibility for potential collaborators that can help to extend CoreFlow with new features. Additionally, Python has a large set of libraries and tools available that are specifically useful for analysis and working with large data sets, these libraries can be used to aid the correlation and enrichment process.

3.2. NetFlow and IPFIX

NetFlow, originally developed by Cisco Systems, but now present on most modern routers is a protocol that allows routers and other network devices to export flow information. According to [8], Cisco traditionally distinguishes a flow based on 7 properties, two of which are not required:

- IP source address
- IP destination address
- source port
- destination port
- L3 protocol type
- Class of service (optional)
- Router or switch ingress port (optional).

These properties are extended in subsequent versions such that NetFlow supports IP6v, vlans, and MPLS labels.

IPFIX (IP Flow Information eXport) described in RFC515 [9] is a protocol developed by IETF that supersedes NetFlow v9. The major tools and collectors used to work with netflow information are adapted to also accept the IPFIX format. In this paper we use the term NetFlow to refer to both the NetFlow and IPFIX protocols. In CoreFlow the data we import from the routers uses the nfdump 1 format.

1 nfdump website: https://github.com/phaag/nfdump.
Table 1

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Enterprise/Campus</th>
<th>Carrier/Transit</th>
</tr>
</thead>
<tbody>
<tr>
<td>External connectivity</td>
<td>Limited (single or redundant uplink)</td>
<td>Many connected networks</td>
</tr>
<tr>
<td>Application security</td>
<td>Security can be tailored to application</td>
<td>Need to allow everything</td>
</tr>
<tr>
<td>Restrictions and policies</td>
<td>Can be applied anywhere</td>
<td>Subject net neutrality laws</td>
</tr>
<tr>
<td>Impact of countermeasure</td>
<td>May affect users of a host or system</td>
<td>Can affect many users and other networks</td>
</tr>
<tr>
<td>Network capacity</td>
<td>Accommodates one organization</td>
<td>Accommodates many institutions</td>
</tr>
</tbody>
</table>

Fig. 1. ESnet network.
Source: http://www.es.net.

3.3. Splunk

Splunk [10] is a search and analysis system for big data that is often used as a security information and event management (SIEM) system. It can be used to import logs from multiple sources for analysis. It provides a web interface that can be used to search and to make visualizations of the data for easy analysis. If needed, Splunk can also trigger and present security alerts. ESnet uses Splunk to aggregate and visualize log data, therefore we set up CoreFlow to consume the already aggregated Bro data in Splunk via a REST interface.

3.4. Packet design route explorer

Route Explorer [11] is a route analysis system developed by Packet Design. The appliance provides visibility into routing behavior for IGP and BGP routing protocols and VPNs. By peering with the routers it is able to track real-time changes in the network; it monitors routing tables and can store them for historical analysis. It can then be used by network administrators to debug problems in a complex network infrastructure. CoreFlow can use Route Explorer to perform path calculation (see Section 5.1).

4. Coreflow architecture

The architecture of CoreFlow is composed of three distinct phases: input, enrichment, and output. This is shown in Fig. 2.

The CoreFlow development was driven by a number of design requirements:

- support the Bro data format. The system needs to ingest and process Bro data;
- allow for multiple input sources. We wanted to be able to accept Bro data from different sources, for example reading from file or gathering it in real-time;
- process large amounts of NetFlow data. The system needs to process data from multiple routers.

![Diagram of CoreFlow architecture](https://example.com/coreflow_diagram.png)

Fig. 2. CoreFlow correlates input data from Bro to NetFlow and uses the enriched data to query the route analyzer. Finally, it outputs the security event with additional data from both enrichers.

4.1. Input phase

We support multiple ways to import the Bro data into CoreFlow:

- file operates on Bro log files in either text or gzip format
- stdin operates on output from the standard input in Bro log format
- splunk opens a socket to the Splunk server and starts a real time search for incoming events
- elasticsearch reads Bro data that has been imported into Elastic-search using an included import tool.

The stdin and splunk input methods support streaming of real time data. The file and splunk methods support reading historical data from within specified time window. We will elaborate on these two different uses in Section 4.2.

As main input we use the Bro notice log; this log file contains (security) events that are interesting enough to require further
investigation. The fields relevant for correlation are listed in Table 2.

The \( \text{uid} \) field contains a unique identifier which is a hash based on various properties of the event. This can be used to correlate the event between multiple Bro log files. To correlate Bro events to NetFlow data we cannot use this \( \text{uid} \) and we are required to match on the flow data contained in the event. Not all Bro events contain the required flow data and the events without this data are passed to the output queue without further enrichment.

We chose to represent the flow information in CoreFlow with a tuple consisting of 5 elements: protocol, source ip, source port, destination ip and destination port. These elements correspond to the mandatory NetFlow properties we discuss in Section 3. Each one of these properties correspond to a specific Bro field. Table 3 shows the mapping. Since we are working with data from multiple nodes, event time stamps may not be the same everywhere and it is not be used in the initial matching process.

### 4.2. Enrichment phase

We distinguish two modes of correlation: historical and real time.

Historical correlation specifies a time window in which to match the flows. CoreFlow first processes the Bro data, correlates it with the NetFlow data and then exits. The sizes of the log files can easily exceed gigabytes; the data workflow is customized to minimize memory utilization and random IO and to retain reasonable speeds.

Real time correlation works by streaming the latest events from the Bro notice log. Since we are using \( nf\)dump files for NetFlow processing and do not have a source that was able to stream real time NetFlow information, there is a time delay introduced in processing the events. CoreFlow periodically sends out NetFlow searches and queues events until the previous search is completed, this approach prevents slowdown caused by many searches blocking on disk I/O.

After the matching process the Bro event is enriched with one or more NetFlow records, one for every router it was seen on. When combined with sufficient topology information one can now estimate the exact path of the event flow and the ingress and egress router and ports (see Section 5.1).

### 4.3. Output phase

When the NetFlow and path information is merged with the Bro events summary output is written to stdout. Listing 1 shows an example of the stdout output. The output includes: alert id, flow(protocol, source ip, source port, destination ip, destination port), alert message, and an ordered list of routers provided by the route estimation described in Section 5.1. Besides the summarized output, CoreFlow provides an output module that exports the full enriched output as json to a log file. There is also support for Elasticsearch. Other outputs are being considered, for example, a Bro output such that the enriched output can be used to create new alerts. This idea is discussed in Section 6.4.

### 5. Implementation

The first prototype of CoreFlow is implemented in Python 3.5 using the Python requests and Elasticsearch libraries. CoreFlow has a main loop that routes messages from input to the output via the NetFlow enricher.

![Fig. 3. Execution flow of CoreFlow, with its three threads: main loop (orange), input thread (blue), search thread (cyan/green). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)](image)

Listing 1: CoreFlow stdout summary output

```plaintext
1 CCRhH1P1x4ukkv7Fe2c ('tcp', '192.0.2.170', '36099', '203.0.113.212', '5308') ['203.0.113.232 has a certificate on port 5308/tcp that has not been baselined.'] ['deserver']
2 C0H6qYtCGGUSpr0J ('tcp', '192.0.2.150', '42090', '203.0.113.175', '80') ['A local user used curl or wget to download a file. They downloaded: W/A'] ['aofa', 'sacramento']
3 C2xdQ38J8zu3c6K9 ('tcp', '192.0.2.170', '36033', '203.0.113.222', '5308') ['203.0.113.232 has a certificate on port 5308/tcp that has not been baselined.'] ['deserver']
4 CbgfFyogoMV2fKYS ('tcp', '192.0.2.170', '36551', '203.0.113.222', '5308') ['203.0.113.232 has a certificate on port 5308/tcp that has not been baselined.'] ['kansas', 'newyork']
5 CtaF3vFuir1lpv1V ('tcp', '192.0.2.214', '18779', '203.0.113.184', '443') ['SSL certificate validation failed with (self signed certificate in certificate chain)'] ['chicago']
6 CZyAKWuHOPP7e3g1 ('tcp', '192.0.2.170', '35587', '203.0.113.222', '5308') ['203.0.113.232 has a certificate on port 5308/tcp that has not been baselined.'] ['aofa', 'chicago', 'kansas', 'newyork', 'sacramento', 'washington']
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>Datetime</td>
<td>Timestamp</td>
</tr>
<tr>
<td>uid</td>
<td>String</td>
<td>Unique id to correlate to conn log</td>
</tr>
<tr>
<td>id.orig_h</td>
<td>String</td>
<td>Ip address source</td>
</tr>
<tr>
<td>id.orig_p</td>
<td>String</td>
<td>Source port</td>
</tr>
<tr>
<td>id.resp_h</td>
<td>String</td>
<td>Ip address destination</td>
</tr>
<tr>
<td>id.resp_p</td>
<td>String</td>
<td>Destination port</td>
</tr>
<tr>
<td>proto</td>
<td>String</td>
<td>Protocol (TCP, UDP, ICMP)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3</th>
<th>The CoreFlow flow tuple and the equivalent fields in Bro and NetFlow data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoreFlow</td>
<td>proto</td>
</tr>
<tr>
<td>Bro</td>
<td>proto</td>
</tr>
<tr>
<td>NetFlow</td>
<td>pr</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>proto</td>
<td>String</td>
<td>Protocol (TCP, UDP, ICMP)</td>
</tr>
<tr>
<td>id.resp_p</td>
<td>String</td>
<td>Destination port</td>
</tr>
<tr>
<td>id.resp_h</td>
<td>String</td>
<td>Destination Ip Address</td>
</tr>
<tr>
<td>id.orig_p</td>
<td>String</td>
<td>Source port</td>
</tr>
<tr>
<td>id.orig_h</td>
<td>String</td>
<td>Source Ip Address</td>
</tr>
<tr>
<td>uid</td>
<td>String</td>
<td>Unique id to correlate to conn log</td>
</tr>
<tr>
<td>ts</td>
<td>Datetime</td>
<td>Timestamp</td>
</tr>
</tbody>
</table>

Table 2 | Bro notice.log field necessary for the correlation process. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
<td>Type</td>
</tr>
<tr>
<td>ts</td>
<td>Datetime</td>
</tr>
<tr>
<td>uid</td>
<td>String</td>
</tr>
<tr>
<td>id.orig_h</td>
<td>String</td>
</tr>
<tr>
<td>id.orig_p</td>
<td>String</td>
</tr>
<tr>
<td>id.resp_h</td>
<td>String</td>
</tr>
<tr>
<td>id.resp_p</td>
<td>String</td>
</tr>
<tr>
<td>proto</td>
<td>String</td>
</tr>
</tbody>
</table>
CoreFlow reads the events queue and when it finds new events it extracts the flow tuple. This together with the event id is inserted in a queue for the NetFlow enricher. When the NetFlow enricher is idle it picks up all items in the queue at once; it creates a filter for all the flows and their reverse that can be passed to nfdump. A reverse flow is simply the flow detected by Bro with source and destination IP/port swapped. We need both flow and its reverse given that we want to have visibility in the bidirectional traffic. Creating such a bulk request, a filter with multiple flows/reverse, is significantly faster than passing each event one by one because now we have to search through the flow data only once. Depending on the amount of routers in the network the NetFlow enricher will spawn one search thread per router that runs nfdump with the previously compiled filter. The results of the bulk request come back out of order, and we need to mapped them back onto the original Bro data.

Now the NetFlow data is mapped back to the event id’s and it put into another queue to CoreFlow. CoreFlow now adds the NetFlow data to the existing events and passes them on to the output module that logs the enriched data.

The enriched event data contains multiple occurrences of the flow reported by multiple routers and together with topology information CoreFlow tries to reconstruct the path of the flow with a route estimation procedure (see Section 5.1). For more detailed route estimation CoreFlow can interface with products such as Route Explorer by Packet Design.

Finally, after the routes have been identified, CoreFlow exports the results.

5.1. Route estimation

ESnet uses OSCARS [12] for provisioning links across its network and OSCARS therefore maintains a database with topology information. To create the required topology information for CoreFlow, we extract the topology information from the OSCARS topology publisher. The extracted information does not contain policy information or any routing metrics used to select preference. Therefore, we decided upon finding the shortest path with the constraint of traversing all the routers for a single flow as an approximation. We designed Algorithm 1 with the following requirements in mind:

- The input list may have missing routers; for example a flow may traverse a router but may not be recorded due to the sample rate.
- The path may traverse a router multiple times; flows may be observed on a router twice for example using different vlan/mls label.

The algorithm works as follows.

We take as a starting point the first router in list D, start (line 5); then we use the topology to build a tree from start limited to a depth and return the paths as an array P (line 6). To include all possibilities the depth should be set to the maximum spanning tree distance of the network graph.

We reverse all the paths (lines 7–9) and then we concatenate the result R with the original paths in P (lines 10–14). This gives us list A of all paths that traverse the start node. We then filter A to only include paths that contain all routers in D and store this as F (lines 15–18). We select the minimum length paths in F and return this list as value O (lines 20–23).

The output of this algorithm can be illustrated with a simple example. Fig. 4 shows a topology with nodes r1 – r12; a flow entered the network at r2 and exited at r10. The routers that observed the flow are D = [r1, r12, r3]. Our algorithm is able to interoperate that r4 and r9 are part of the path and it returns [r1, r4, r3, r9, r12] as estimated route together with its reverse. Note that r2 and r10, the ingress and egress nodes, are not part of the reconstructed path as they had not observed the flow themselves directly. A current limitation of the algorithm is that is not capable to determine which one was the actual ingress and egress router; this is because the topology information we rely upon does not distinguish edge routers.

6. Evaluation and discussion

We tested the prototype on the ESnet infrastructure by enriching incoming events from three different Bro nodes with NetFlow data collected by over 50 routers. Fig. 5 shows the latest set-up we used for CoreFlow at ESnet. There were two specific limitations in ESnet that we had to deal with.

We had 3 Bro detectors sending their logs to Splunk. CoreFlow was reading the logs from Splunk and performing searches on NetFlow data of all routers. The NetFlow data was exposed to CoreFlow via an NFS share. Every 5 min a new NetFlow log of a router gets saved and gets copied to the NFS server. Under normal circumstances this copy time took less than 3 min. This meant that we had to delay the retrieval of incoming events from Splunk by 5 + 3 = 8 min.

When the flow is detected on multiple routers CoreFlow performed route estimation as described in Section 5.1 and prepared queries for the Route Explorer to further refine the found route. Due to access restrictions we could not query the Route Explorer database directly.

### Algorithm 1 route estimation algorithm

1. **topology** ← topology graph of the network
2. **depth** ← max search depth
3. D ← detected routers in the path
4. **procedure** ESTIMATE PATH(D)
5. **start** ← D[0]
6. P ← all paths up to depth from **start** in **topology**
7. **for** each p ∈ P **do**
8. R ← add reverse(path)
9. **end for**
10. **for** each p ∈ P **do**
11. **for** each r ∈ R **do**
12. A ← add r + p[1 :]
13. **end for**
14. **end for**
15. **for** each p ∈ A **do**
16. if D ⊆ p **then**
17. F ← add p
18. **end if**
19. **end for**
20. **for** each p ∈ F **do**
21. O ← min(length(p))
22. **end for**
23. return O
24. **end procedure**

![Fig. 4. Route estimation: given a list with routers [r1, r12, r3] the algorithm is able to interpolate that r4 and r9 must also be included in the path resulting in [r1, r4, r3, r9, r12]. However with the current information the algorithm cannot deduct the edge routers r2 and r10.](image-url)
provided by CoreFlow into the IDS, alerts can be raised for possible to alert on this event. By feeding back the additional information by the IDS. Based on the new context the IDS can now consider that previously did not pass the alert threshold can be re-evaluated.

This results in events being sent into CoreFlow that was previously not seen by and acted upon by the security team. The added contextual information provided by CoreFlow on the events that previously did not pass the alert threshold can be re-evaluated by the IDS. Based on the new context the IDS can now consider to alert on this event. By feeding back the additional information provided by CoreFlow into the IDS, alerts can be raised for possible

6.3. Sample rate

An another point of attention is the sample rate of NetFlow. In ESnet, for example, the sample rate of the data was set to 1:1000 on each router. The unfortunate side effect of a low sample rate is that the probability to find flows related to the IDS alerts is also very low, since the sample rate needs to be multiplied by rate of malicious flows to all traffic on each router. This can be improved by increasing the sample rate on all the routers. In ESnet this was not possible since we were running this on a production network and higher sample rates may result in degraded network performance because more samples require more processing on the production routers. Another way to improve the chance of finding flows which can have less impact on the network is using high sample rates at the edges. This approach may be feasible in carrier networks since the bulk traffic streams are located in the core. Additionally, this approach also increases the chance of finding the flow on the ingress router which benefits path estimation and can help to apply countermeasures at the point of entry. Methods described in [13] can also help improve the sampling algorithms in to detect smaller flows.

One might argue whether is necessary to increase the sample rate to detect small flows on the network in the context of cybersecurity. Volume based attacks such as DDoS attacks will for example be clearly visible in sampled data. Yet, given the right circumstances, an attacker can do much damage using only a few packets. Moreover, there are instances on attackers using volume based attacks to distract the victim from the real attack [14]. Therefore, it is important to provide as much context to every event that the intrusion detection system marks as malicious.

6.4. Other use cases

CoreFlow can also be used in multi-domain defense strategies. When it is possible to establish the ingress point of the spoofed malicious traffic it is possible to contact the neighboring domain to take action. If the neighbor also has such a system it can subsequently contact its neighbors, eventually tracing the traffic back to the source. Taking action closer to the source of the problem can unburden networks of large volume based attacks.

During Supercomputing 2015, the University of Amsterdam demonstrated SARNET (Secure Autonomous Response Networks) using an interactive demo [15]. Users had to defend a network under attack by applying countermeasures at points in the network to recover the throughput to the services. The demo showed that responses can become complex and even counter intuitive when networks increase in size and when information is limited. SARNET can greatly benefit from CoreFlow since it provides richer information and more context to enhance the decision making leading to autonomous response.

7. Related work

Much work is done on applying statistical methods and machine learning approaches to NetFlow data in order to detect anomalous behavior on computer networks. These anomalies can be caused by network changes, outages, content changes or security related events. Sperotto et al. published a comprehensive overview of flow based intrusion detection in [16].

The goal of CoreFlow is not to identify security threats. CoreFlow does not do any intrusion detection. It assumes there are already
facilities in place that generate these security events. CoreFlow uniqueness is that it focuses on the correlation and enrichment of already identified events, by using multiple data sources such as NetFlow to create a more comprehensive view of what occurred in order to enhance decision making.

Xu et al. describe a system that can group low level events from several inputs based on similarities or relations [17]. When the low level events trigger at the same time, as a group, a more meaningful high level event (alert) can be created to act upon. Our approach is different since we correlate the triggered events to data sources that may not have triggered events themselves, in order to expose more contextual information for further analysis. If we would accept multiple input sources, then grouping triggered events becomes relevant for CoreFlow, yet this is considered future work.

There is numerous work done on IP traceback [18,19]. A more recent example from Tian et al. [20] uses sample netflow data from different autonomous systems on a hop by hop basis to find the origin of the traffic. CoreFlow works differently: First, CoreFlow is not a solution purely designed for route estimation. The intra-domain route estimation is currently provided as a working application that uses the new context, in this case router identifiers, given by CoreFlow based on NetFlow information. Secondly, in stead of tracing back to the source on a hop by hop basis, the route estimation is provided with a full topology view and a list of routers a-priori, based on the enriched data from CoreFlow. Finally, CoreFlow does not interact with routers or the traffic itself such as some other intra-domain traceback methods. CoreFlow relies on other systems to make decisions on the enriched data and take action when necessary.

In Secure Autonomous Response Networks [21] we try to create networks that autonomously respond to cyber attacks. SARNET uses control loops to detect, classify, analyze, decide, respond and learn about cyber attacks on a network. Using CoreFlow in such a framework can benefit the analyze phase by drawing in information from multiple data sources and provide a more comprehensive picture to the decide and response phases.

8. Conclusion and future work

Enriching IDS data with NetFlow information gives a better view of an attack. CoreFlow provides a correlation framework that can combine these data sources based on the flow tuples. The successfully enriched data can be used for more advanced attack detection and reaction.

We determined that the success of the NetFlow correlation largely depends on the sampling rate of the NetFlow data. We showed how to use the enriched information to do route estimation; this in turn can be the starting point for sophisticated countermeasures close to the origin needed when the attack traffic is spoofed and for carrier networks to determine where the traffic entered their network.

The focus of our future work is to evaluate the effectiveness of CoreFlow with different sample rates (1:1) and other sampling algorithms to see what settings are most beneficial, while not affecting the performance of a production network.

CoreFlow can be extended to allow multiple in and output plugins for other data sources such as PerfSonar and syslog and to include analysis methods that can help to interpret the information and improve the context of an event.

This new context can lead to improved and more advanced alerts, research needs to be done on how to act upon this new information by for example feeding this back into the IDS system to reduce false positives; it may even be beneficial to lower the threshold for IDS events sent into CoreFlow to discover malicious events that previously went undetected.

Acknowledgments

This work is funded by the Dutch Science Foundation project SARNET (grant no: CBVSEC.14.003/618.001.016) and by the Dutch project COMMIT (WP20.11). Special thanks go to our research partners CIENA, TNO and KLM. This work was supported by the Director, Office of Science, Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Ralph is grateful for the financial support given by ESNet during his stay at LBNL, and he thanks the ESNets team for the interesting discussions during the CoreFlow development.

References


2 PerfSonar website: https://www.perfsonar.net/.
Ralph Koning received his M.Sc. in System and Network Engineering in 2007 at the University of Amsterdam. After being employed in the System and Network Engineering research group at the UvA he started his Ph.D. in 2015 on the SARNET project. He contributed work to several projects such as GigaPort, CineGrid, GN3plus and COMMIT. His current interests include computer networks, SDN infrastructures, semantic descriptions and digital security. https://staff.fnwi.uva.nl/r.koning/.

Nick Buraglio has been involved in the networking industry in varying roles since 1997. Prior to joining the Network Engineering group at ESnet, Nick was employed by the University of Illinois as the Lead Network Engineer working on research and HPC, campus and wide area connectivity. In this role, Nick also functioned as the Lead Network Engineer and IP architect for the National Association of Telecommunications Officers and Advisors (NATOA) broadband project of the year, UC2B. Nick has also held Network Engineering positions at early regional broadband internet providers as well as at the National Center for Supercomputing Applications. Nick has participated in the SCinet working group on several occasions and has been involved in R&E, high performance networking and security since 2002. In addition to Network Engineering positions, Nick has been involved in cybersecurity from the campus, enterprise and service provider perspective and acted as a resource and trainer for the Federal Bureau of Investigation RCAT agents. Nick has been active in the SDN community since 2009 and is currently actively involved in several SDN related initiatives and projects. https://es.net/about/esnet-staff/network-planning/nick-buraglio/.

Cees de Laat chairs the System and Network Engineering (SNE) laboratory in the Informatics Institute of the Faculty of Science at University of Amsterdam. The SNE lab conducts research on leading-edge computer systems of all scales, ranging from global-scale systems and networks to embedded devices. Across these multiple scales our particular interest is on extra-functional properties of systems, such as performance, programmability, productivity, security, trust, sustainability and, last but not least, the societal impact of emerging systems-related technologies. Prof. de Laat serves on the Lawrence Berkeley Laboratory Policy Board for ESnet, is co-founder of the Global Lambda Integrated Facility (GLIF), founder of GRIDforum.nl and founding member of CineGrid.org. His group is/was part o/a.o. EU projects GN4-2, SWITCH, CYCLONE, ENVRIplus and ENVRI, Geysers, NOVI, NEXTGRID, Egee, and national projects SARNET, COMMIT, GIGAport and VL-e. He is a member of the Advisory Board Internet Society Netherlands and Scientific technical advisory board of SURF Netherlands. See: http://delaat.net/.

Paola Grosso is an Assistant Professor in the SNE group at UvA. Her research interests are green ICT, development of information models for hybrid, multi-domain, multi-layer networks, and software-defined networks. She has numerous publications on these topics and has participated in numerous international and national projects. Grosso is currently involved in the EU-funded projects GN4-2 and ENVRIplus, and in the Dutch projects SARNET, RoN and COMMIT. http://www.science.uva.nl/~grosso.