CoreFlow: Enriching Bro security events using network traffic monitoring data

Ralph Koning[a,b,*], Nick Buraglio[b], Cees de Laat[a,b], Paola Grosso[a]

[a] *Universiteit van Amsterdam, Science Park 904, Amsterdam, The Netherlands*
[b] *Energy Sciences Network, Lawrence Berkeley Lab. Berkeley, CA, USA*

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 device 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.

Keywords: Security, Network, IDS, netflow, flow, detection, IPFIX, DDoS, Carrier networks, Transit networks

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 [1]; against commercial network providers such as end-user ISPs [2]; or against educational and research networks serving academia [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 [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 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.

*Corresponding author
Email addresses: r.koning@uva.nl (Ralph Koning), buraglio@es.net (Nick Buraglio), delaat@uva.nl (Cees de Laat), pgrosso@uva.nl (Paola Grosso)

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's 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 super computing facilities, as well as other other research networks in the world. Figure 1 shows the topology of the ESnet backbone network that spans the US and a part of Europe. The backbone consists mainly of 100Gbps links and allows sites to connect to ESnet at various speeds.

ESnet primarily transits data within the connected institutions and to other connected research facilities and resources

Aspect	Enterprise/Campus	Carrier/Transit
external connectivity	limited (single or redundant uplink)	many connected networks
application security	security can be tailored to application	need to allow everything
restrictions and policies	can be applied anywhere	subject net neutrality laws
impact of countermeasure	may affect users of a host or system	can affect many users and other networks
network capacity	accommodates one organization	accommodates many institutions

Table 1: Major differences between Enterprise/Campus networks and Carrier/Transit networks that are relevant from a security point of view

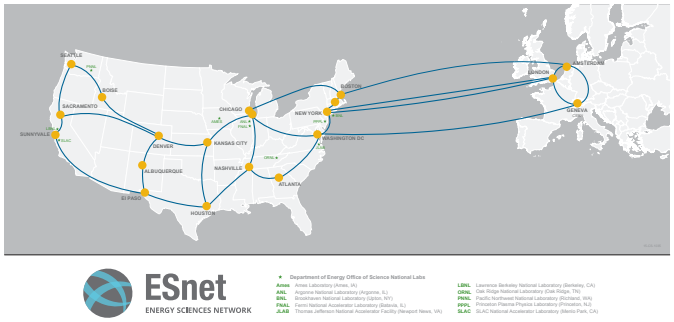

Figure 1: ESnet network. Source: `http://www.es.net`

and therefore operates as a carrier or transit networks 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 transits 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 super computing 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 100Gbps 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 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 IPv6, vlans, and MPLS labels.

IPFIX (IP Flow Information eXport) described in RFC5153[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.

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 Sec.5.1).

4. CoreFlow Architecture

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

[1] nfdump website: `https://github.com/phaag/nfdump`

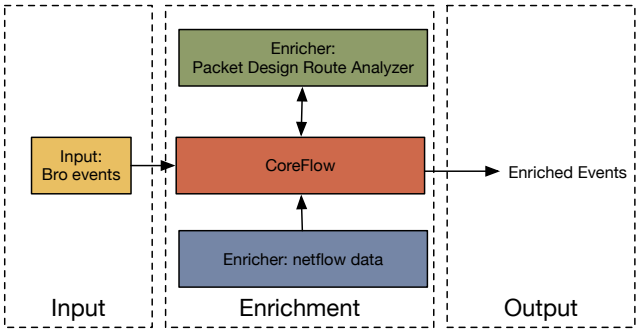

Figure 2: CoreFlow correlates input data from Bro to NetFlow and uses the enriched data to query the route analyzer. Finally, it outputs the security event with additional data from both enrichers.

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 Elasticsearch 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 Sec. 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.

field	type	description
ts	datetime	timestamp
uid	string	unique id to correlate to conn log
id.orig_h	string	ip address source
id.orig_p	string	source port
id.resp_h	string	ip address destination
id.resp_p	string	destination port
proto	string	protocol (TCP,UDP,ICMP)
...	...	...

Table 2: Bro notice.log field necessary for the correlation process

The *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 *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 discusses in Sec.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.

CoreFlow	proto	ip1	port1	ip2	port2
Bro	proto	orig_h	orig_p	resp_h	resp_p
NetFlow	pr	sa	sp	da	dp

Table 3: The CoreFlow flow tuple and the equivalent fields in Bro and NetFlow data

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 *nfdump* 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 Sec. 5.1).

4.3. Output phase

When the NetFlow and path information is merged with the Bro events summary output is written to *stdout*. Additionally, CoreFlow provides a simple output module that exports enriched output as json to a log file. There is also experimental output support to 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 Sec. 6.

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.

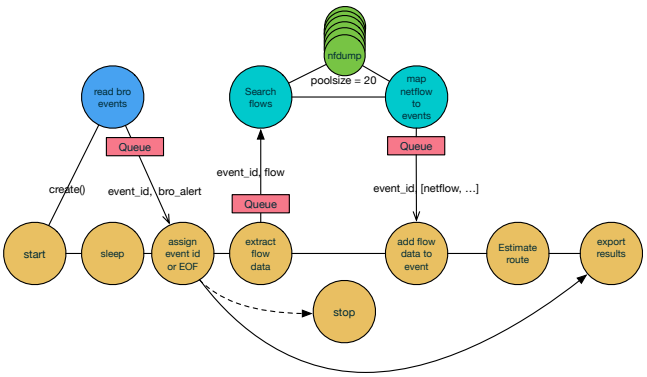

Figure 3: Execution flow of CoreFlow, with its three threads: main loop (orange), input thread (blue), search thread (cyan/green).

Figure 3 shows the execution flow of CoreFlow. There are three threads: a main loop, an input thread and a search thread.

The input modules run in a separate thread that is being watched and when necessary restarted by CoreFlow. CoreFlow receives the data from the import thread via an event queue which contains the *bro_alert* and the *event_id*.

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 Sec.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/mpls 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. Figure 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 interpolate 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.

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(lenght(p))
22:     end for
23:     return O
24: end procedure
```

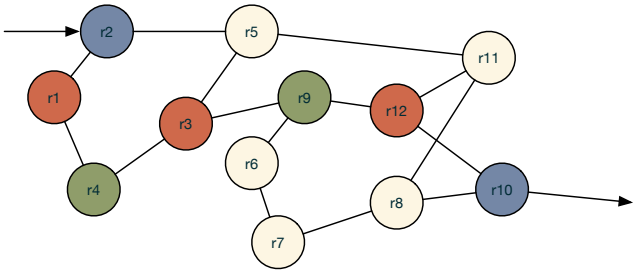

Figure 4: Route estimation: given a list with routers [r1 r12 r3] the algorithm is able to interpolate that r4 and r9 must also be included in the path resulting in [r1, r4, r3, r9, r12]. However with the current information the algorithm cannot deduct the edge routers r2 and r10.

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. Figure 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 minutes 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 minutes. This meant that we had to delay the retrieval of incoming events from Splunk by 5 + 3 = 8 minutes.

When the flow is detected on multiple routers CoreFlow performed route estimation as described in Sec. 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 directly, so we verified this functionality by sending the query from another host.

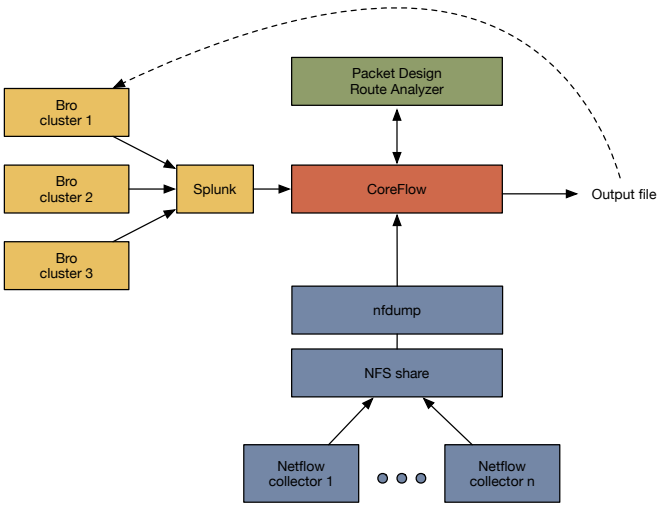

Figure 5: CoreFlow set-up at ESnet

6.1. Route estimation

The route estimation can be optimized in multiple ways. The data structures contains redundant information and for large networks this data structure may get too big. The algorithm does not deal with metrics and routing policies and any traffic engineering that can manipulate the flow of traffic because this information was not available at the time. Improvements to the route estimation can be made by calculating paths based on live routing tables of the network. For historical paths we can rely on products such as Packet Designs Route Explorer that records changes in the routing table over time. By recording this information Route Explorer can provide paths from an ingress router to a destination prefix at any point in time. However, this requires us to determine the ingress router of the specific flow and when NetFlow traffic is sampled we may not be able to because see the flow on the ingress router. If we find the flow on one or more routers in some situations we can use the route estimation explained in section 5.1 to extrapolate a potential ingress router that we can use for the full path calculation.

Adding reconstructed paths and NetFlow information to security events allows for more targeted monitoring or mitigation techniques, e.g. blocking at the source or redirecting the traffic somewhere along the path for further analysis. Additionally, one can feed the enriched data back into the IDS to enhance filtering on relevant alerts. e.g. by lowering the threshold for data going to CoreFlow and create more specific event filters on CoreFlows output.

6.2. 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 counter measures 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.

6.3. 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 Super Computing 2015, the University of Amsterdam demonstrated SARNET (Secure Autonomous Response Networks) using an interactive demo[15]. Users had to 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].

CoreFlow goal 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.

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.

CoreFlow needs to be evaluated using 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[2] 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.

9. Acknowledgments

This work is funded by the Dutch Science Foundation project SARNET (grant no: CYBSEC.14.003 / 618.001.016) and by the Dutch project COMMIT (WP20.11). Special thanks go to our research partners CIENA, TNO and KLM.

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

[2] PerfSonar website: `https://www.perfsonar.net/`

[1] K. McArtney, Internet's root servers take hit in DDoS attack, `http://www.theregister.co.uk/2015/12/08/internet_root_servers_ddos/`, 2015. Accessed on 2016/09/15.

[2] P. Mishra, Internet in Mumbai Goes Slow As ISPs Suffer Massive DDoS Attacks, `https://www.hackread.com/ips-in-mumbai-suffer-ddos-attacks/`, 2016. Accessed on 2016/09/15.

[3] K. Hall, Academic network Janet clobbered with DDoS attacks again, `http://www.theregister.co.uk/2016/04/18/janet_clobbered_with_ddos_attacks_again/`, 2016. Accessed on 2016/09/15.

[4] H. Debar, M. Dacier, A. Wespi, Towards a taxonomy of intrusion-detection systems, Computer Networks 31 (1999) 805–822.

[5] M. V. Mahoney, A machine learning approach to detecting attacks by identifying anomalies in network traffic, Ph.D. thesis, Florida Institute of Technology, 2003.

[6] V. Paxson, Bro: a system for detecting network intruders in real-time, Computer networks 31 (1999) 2435–2463.

[7] S. Campbell, J. Lee, Intrusion detection at 100g, in: State of the Practice Reports, ACM, p. 14.

[8] NetFlow, Cisco IOS, Introduction to cisco ios netflow a technical overview, `http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`, 2012. Accessed on 2016/09/15.

[9] E. Boschi, L. Mark, J. Quittek, M. Stiemerling, P. Aitken, RFC 5153: IP flow information export (IPFIX) implementation guidelines, IETF, April (2008).

[10] D. Carasso, Exploring splunk, published by CITO Research, New York, USA, ISBN (2012) 978–0.

[11] Packet Design, Route Explorer, `http://www.packetdesign.com/products/route-explorer/`, 2016. Accessed on 2016/09/15.

[12] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, W. Johnston, Intra and interdomain circuit provisioning using the oscars reservation system, in: 2006 3rd International Conference on Broadband Communications, Networks and Systems, IEEE, pp. 1–8.

[13] G. Androulidakis, V. Chatzigiannakis, S. Papavassiliou, Network anomaly detection and classification via opportunistic sampling, IEEE network 23 (2009) 6–12.

[14] S. Mansfield-Devine, Under the radar, Network Security 2015 (2015) 14–18.

[15] R. Koning, B. de Graaff, C. de Laat, R. Meijer, P. Grosso, Interactive analysis of sdn-driven defence against distributed denial of service attacks, in: 2016 IEEE NetSoft Conference and Workshops (NetSoft), IEEE, pp. 483–488.

[16] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, B. Stiller, An overview of ip flow-based intrusion detection, IEEE communications surveys & tutorials 12 (2010) 343–356.

[17] D. Xu, P. Ning, Alert correlation through triggering events and common resources, in: Computer Security Applications Conference, 2004. 20th Annual, IEEE, pp. 360–369.