NetFlow Anomaly Detection; finding covert channels on the network

Joey Dreijer, student System and
Network Engineering

Abstract: The research focusses on detecting (well-known) Covert Channels and tunneling software by using NetFlow(v5) data. The research-data originates from generated known-good, known-bad and corporate Shell network flows. Based on the known-good and known-bad comparison, a possible anomaly detection based on profiles proof of concept was developed. The tunneling software being used during this research showed different behaviour compared to normalized data belonging to the used protocol(s). By using multiple metrics and variables originating from the NetFlow fields, abnormal behaviour could be detected and potential false-positives were able to be filtered.

Keywords: NetFlow, Covert Channels, Tunneling, Anomalies, OS3, SNE

1. Introduction

NetFlow is a standardized format (commonly used by Cisco) to gather/sent network flow data. NetFlow services give network administrators insight in the current network operation. The NetFlow [8] [1] standard is used in different multi-vendor switches and routers. NetFlow data can be used to monitoring the efficiency of the network, but can also be used for network security analysis. Different proprietary tooling exists to analyse NetFlow data for security purposes. An important sidenote is that NetFlow version 5 (used during this research) does not contain any details regarding packet content. Some example metrics that NetFlow provides are bytes sent, amount of packets sent and source/destination addresses. This data will be used to detect possible covert channels / tunnels. This research will focus on detecting covert channels by using the limited data the NetFlow v5 standard supplies.

The primary research question is stated as follows::

Can Covert Channels be detected by purely using NetFlow (v5) data?

1. What NetFlow(v5) metrics can be used for Covert Channel detection?
2. How can the researched detection method be implemented in an operational detection scheme?

1.1. Supervision and dataset

The research was performed at the Shell Cyber Centre (Rijswijk, the Netherlands). Shell uses several different monitoring platforms to analyse security threats. The data being analyzed also includes network flow data from different segments within the Shell network. Shell was able to provide a dataset originating from different un-managed (Guest Network) devices that have an active internet connection available. The data being used during the research mainly includes self-generated known-good and known-bad data, where Shell's provided dataset will be used as reference.

1.2. Ethical considerations

The NetFlow data being used during the research have the possibility to relate network flows to a specific user on the network. The data includes timestamps and destination/source addresses that

may provide user-behaviour statistics. User-specific data has not been included in this report. Concrete examples described have been stripped of any personlized information.

1.3. Related work

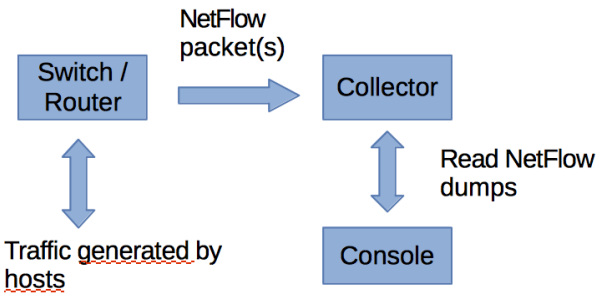
NetFlow security monitoring has been done in the past. However, the research topics mostly related to malware detection and 'generic' security analysis and monitoring. Research done by van Dijkhuisen and Romao [9] demonstrate a concept to detect DDoS attacks via NetFlow anomaly detection. Research done by Pao (et al) and Wei [3] show how 'abnormal' traffic can be efficiently detected.

2. Background; the Netflow standard

The term 'NetFlow' specifies a standard to effectively collect and send network (IP-based) traffic statistics. NetFlow was originally developed by Cisco for their routing equipment, but was standardized by the IETF since version 9 of the protocol. Using NetFlow, at least 2 different roles are required to analyse NetFlow data:

1. An exporter (ex. a switch or router)
2. A collector (ex. server)

An exporter is a networking device that sends NetFlow packets (ie. the statistics) to a receiving server (ie. the collector). The collector parses the NetFlow packets and stores them on either disk or database. Typically, a network administrator would configure a framework and/or interface to read the NetFlow data stored on the collector. An administrator would typically select a 'sample-rate' to determine the amount of NetFlow samples sent to a collector. For this research, no sampling was used on specific protocols. Sampling will introduce different results for detection, this will be discussed later.



An important factor is that NetFlow data works with **Flows** and not **Packets**. The exporter sends out NetFlow data for every (or ratio according to sampling) network flow passing through the device. A network flow consists out of the following characteristics:

- A finished TCP session (shut down by FIN).
- Inactive traffic exceeding a specified timeout
- Active but continuous traffic exceeding a specified timeout

Since UDP traffic does not make use of session states (while TCP does), a single UDP flow will be based on the timer and source/destination (port) addresses. By default, a flow is considered one-way traffic. The request and response and considered two different flows. Samples and parsed NetFlow data will not say anything about the packet content, but does provide a summary of the amount of packets and packet sizes. Since NetFlow has been standardized, other network hardware vendors make use of the NetFlow standard to analyse traffic statistics. Both version 5 and version 9 of the NetFlow specification can often be found on switches or routers. NetFlow version 5 is a static specification, meaning that all of the gathered data has to be send in one specific format in the following order and size:

B	Name	Description
4	srcaddr	Source IP
4	dstaddr	Destination IP
4	nexthop	IP address of next-hop
4	input and output	SNMP index
4	dPkts	Packets in flow
4	dOcters	Bytes in flow
4	First	Time start of flow
4	Last	Time stop of flow
2	srcport	Source port
2	dstport	Dstination port
2	pad1, tcpflags	TCP flags
1	prot	Protocol type
1	tos	Type of service (ToS)
2	srcas, dstas	AS Source and dest.
1	srcmask, dstmask	Mask bits
1	pad2	Padding unused

An open-source collector and parser such as Nfcapd/Nfdump [10] (by Ntop) is able to store these NetFlow packets in a binary format which can be analysed later on. Based on the basic NetFlow 5 specification, the Nfdump parser will be able to display data that looks like the format below:

```
"Start Time","End Time","Duration",
"Source Address","Source App.",
"Dest. Address","Dest. App.",
"In Interface","Out Interface",
"TCP flags","Packets", "Size"
```

```
1402704099267,1402704099267,
0,"208.67.220.220","53/UDP (domain)",
"10.141.174.7","54817/UDP",
"GigabitEthernet1/1","Tunnel3","...",
133,,1,
```

The NetFlow v5 fields that are relevant for (internet) anomaly detection are the source/destination addresses, source/destination ports, the start/stop time of flows, the packets in flows and protocol type. The remainder of the NetFlow metrics will not be used, since only data being sent over the 'internet' is included in this research. The NetFlow 9 standard makes use of user-defined template. This means that NetFlow 9 can include any form of data, as long as it's supported by the vendor. However, NetFlow 9 is not used during this research.

3. Detecting the covert channels

An important assumption is that leaking data out of the network via a Covert Channel is 'always' possible. Sending data via covert channel can be done by using many different protocols, but only a few are actively used and available to the public. A few of these covert channels are even available in standard Linux repositories (such as iodine [12], ptunnel [5] that were used during this research). The following sections will describe what tools and methods have been used to detect some of the popular covert channels.

3.1. Tooling

3.1.1. Collecting NetFlow data

The Nfcapd [10] was used to collect and read NetFlow data. Shell provided a CSV with network flow statistics, but were (unfortunately) not suitable for detection since several flow metrics were not stored. However, the Shell flow dump was used for backwards-reference and comparison. For this research, known-good and known-bad traffic was created using a company (with policies) laptop that generated mail (imap/smtp) web (http/https), tunnel (openVPN and IPsec) SSH, DNS and SMB traffic. This data was purely used to see how 'normal' (non-malicious) protocol statistics look like. Afterwards, covert channels and network scans were performed to compare the 'malicious' flows with the known-good dataset gathered earlier.

The softflowd [7] package was used to generate NetFlow data. Softflowd is able to take a pcap or network interface as input variable and send NetFlow data to a collector.

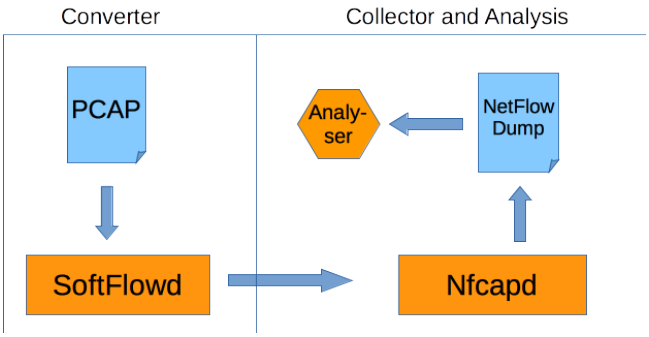


Figure 1: Collecting and storing NetFlow

When the NetFlow data (by softflowd) was sent over the network, the nfcapd daemon stored the information in binary format. This binary format can be read by nfdump, a tool automatically installed when nfcapd is used. Ntop designed their own binary standard for storing NetFlow data. Research done by the University of Twente [4] shows that the binary standard and nfdump tool are able to index and search specific data more efficiently compared to a MySQL [11] database (as example). Based on this conclusion, NetFlow data will be read from the binary format during this research. However, a MySQL database will also be used to compare 'live' NetFlow dumps with stored known-good protocol statistics.

3.1.2. Parsing NetFlow data

Different tooling was required to parse and analyse the NetFlow dumps. A Python module called Pynfdump [2] exists inside the PyPi repositories on Python.org. Pynfdump is used as a wrapper for Nfdump to read the collected NetFlow dumps. However, the wrapper does not support the bi-directional format that Nfdump supports itself. The default NetFlow format looks as follows (also when read by pynfdump):

- Source Address
- Destination Address
- Source Port
- Destination port
- Bytes sent
- Packets sent
- Time

During this research, anomaly detection will be based on both responses and requests. Nfdump supports the option to merge the requests and responses (using a simplistic source/destination comparison). Combining the response and requests will display the following additional data:

- Bytes received
- Bytes sent
- Packets sent
- Packets received
- Flow time

The official Python wrapper does not support the previously mentioned bi-directional format. A custom patch was made that enables users to provide the "multiflow=True" variable when calling the pynfdump.read() function. Supplying this variable will return a dictionary with the bidirectional flows.

```
#!/usr/bin/env python
...
readIn = pynfdump.Dumper(nfLocation,
    sources=SourceList)

readIn.set_where(start="2014-06-30
    10:00")

entries = ReadIn.search('',
    multiflow=True)
...
```

3.2. Methods

Previous research done by van Dijk [9] (et al), demonstrated a working concept of anomaly detection with NetFlow data. Based on this research, a similar aproach was researched used to identify possible covert channels. To perform anomaly detection, different NetFlow samples had to be used. These were:

1. Known-good
2. Known-bad
3. Flow-dump

The known-good consists out of traffic generated by a freshly installed Linux-based virtual machine. This virtual machine generates traffic that consists out of generic browsing, mail traffic, ssh traffic and VPN traffic. These samples do not contain any tunneled/covert channel traffic. The known-bad sample contains both generic traffic and known tunnelled traffic. And finally, a flow-dump provided by Shell was used for reference and test set.

The generated known-good NetFlow dumps were stored inside a MySQL database. For each unique destination port, metrics such as averages and standard deviations were calculated. This was done for each NetFlow field returned by the parser (such as bytes, packets etc). Whenever a specific NetFlow metric can be distributed via standard deviation, possible anomalies can be detected. The data stored looks like the example below:

Port	Packets Out	Bytes Out	Time	...
53	Avg: 1.5	Avg: 60	Avg: 0.242	...
	Std: 0.5	Std: 30	Std: 0.101	...

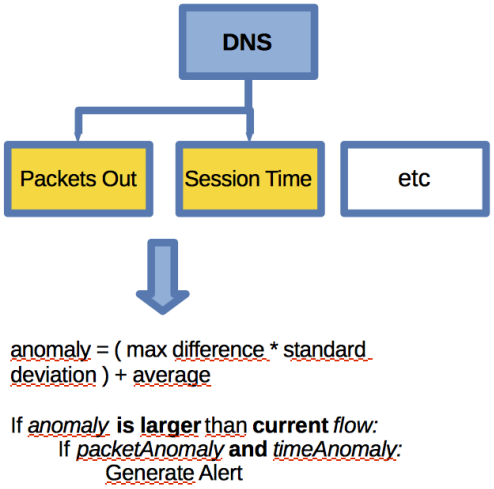
When the 'known-good' data was stored inside the database, each new flow will be compared to the stored averages and deviations. Basic anomalies can be defined according to the rules of default distribution: Whenever a metric such as the amount of outgoing packets is higher than (3*Std) + Avg, the sample belongs to the 0.1% of the outliers. However, 0.1% of a million flows is still a large number. This may detect anomalies, but won't provide concrete alerting for covert channel detection. Doing anomaly detection purely based on the default distribution rules will result in a high false-positive rate. Only using the default distribution rules is not the only problem, another issue occurs when reporting anomalies based on a single metric.

The NetFlow parser used during this research (Nfdump) tries to correlate average session times for network flows. The collector stored the dumps every 5 minutes. A simple DNS requests will take around 0,2 seconds to get a valid reply. If by pure coincidence another DNS request occurs while using the same source/destination values, the Nfdump parser assumes that these two requests are part of the same flow. If each DNS flow would send 2 packets. Nfdump will display a single flow consisting out of 4 outgoing packets with an average session time of (time of first request) + (time of second request). Depending on the interval when Nfdump stores the NetFlow data, a single flow could look like the below example:

Source:10.10.0.2:50001 - **Destination:**8.8.8.8:53
Packets: 4, **Time:** 4001 seconds

This flow may actually consists out of two DNS requests each only lasting 0,2 seconds and sending 2 packets each. When performing anomaly detection only based on the 'time' metric (as example), alerts would have been generated and resulting in false-positives.

A proposed detection method is by manually creating profiles and maximum offsets for each unique protocol to be analyzed.



The image shown above demonstrates a brief example how the proof-of-concept detection works. By combining multiple metrics possible false-positives can be limited. An NetFlow analist would specify that a violation of both the time and packet number exceeds the allowed width of the standard deviation (using the Std) should result in an alert. The code snippet below demonstrates how an anomaly (based on a single metric) is detected. The example fetches the average, standard deviation and allowed 'anomaly size' (ie. width of standard deviation) from the MySQL database that holds the known-good data for each unique protocol. If the 'allowedRange' value is larger (or smaller, not applicable in this case) than the current protocol statistics, an alert is generated.

```
#!/usr/bin/env python
...
if str(resultMetric[2]) == "packets_out":
  maxDiff = float(resultMetric[3])
  avgPackOut = float(resultAvg[11])
  medPackOut = float(resultAvg[26])
  allowedRange = (maxDiff * medPackOut)
      + avgPackOut

  if float(entry['out_packets']) > aRange:
          genAlert(data, metric)

...
```

The next section will demonstrate how this example can be used to detect DNS and ICMP tunnels.

3.3. Examples; DNS and ICMP

The known-good and known-bad traffic used during the research contains valid and tunnelled DNS requests. To apply anomaly detection based on the methods explained in the previous sub-section, specific NetFlow metrics will have to be chosen that contain a 'most' commonly used value. A clear example of a NetFlow metric that can be used is the outgoing/incoming packet amount for DNS and ICMP.

Figure 2 shows personal known-good and known-bad DNS requests displayed in a histogram. In this dataset, the majority of each 'DNS' flow uses 2 incoming packets to fully complete the request. A smaller amount uses 1 single packet and even less for the other numbers. It is important that a clear majority and minority of flows can be seen per metric to create a proper histogram. After generating the known-good traffic, a DNS tunnel was started. The red arrow on the graph shows the amount of packets the tunnel required to initialize the connection (the blue bar isn't visible). Around 14 packets are sent out in a single DNS flow without any user-data actually being transferred. If the user starts using the tunnel, the amount of packets and session time in a single DNS flow will start to increase.

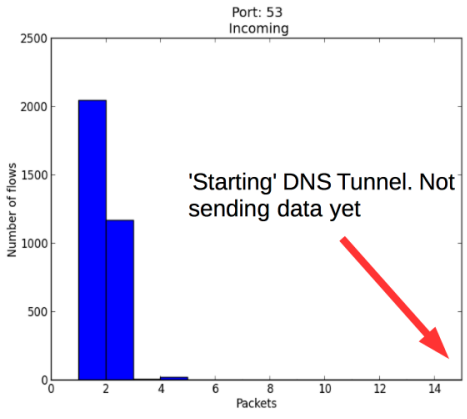


Figure 2: Packet for most common packet count (in whole packets)

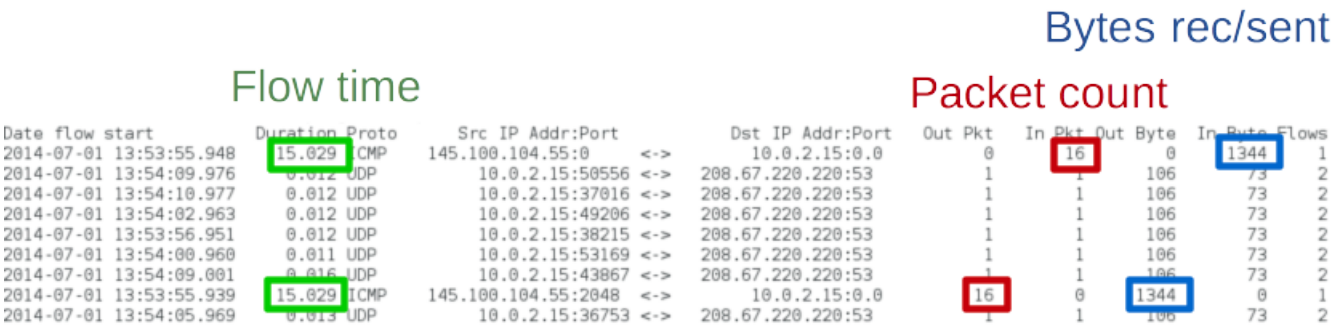


Figure 3: Nfdump displaying ICMP traffic

Figure 3 shows the distribution of total session times for a DNS request. The time includes the total flow length combining both the request and response. The histogram rounds the session timers to 0,10 seconds. Most of the DNS flows take between 0,0-0,10 seconds to complete. The DNS tunnel was inactive during the time of use and is displayed far outside (to the right) the histogram shown above.The combination of longer session time and amount of packets sent may indicate an anomaly compared to the normalized DNS traffic statistics stored earlier.

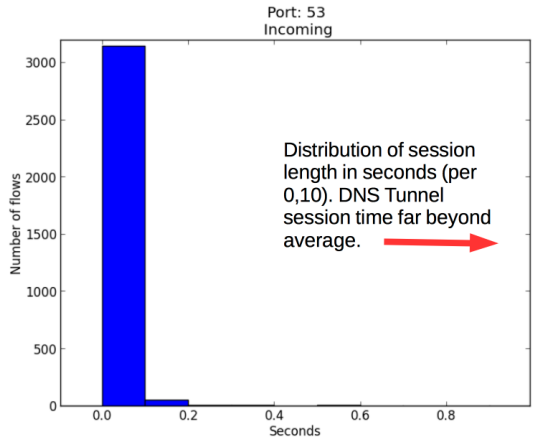


Figure 4: Packet for most common flow times (per 0.10 seconds)

A similar aproach can be done via possible ICMP tunnels. Statistics of ICMP traffic can be monitored for possible anomalies based on several different Metrics. However, having a low false-positive rate depend on nature of the known-good traffic (same applies for DNS tunnels). Simple ping requests can be very common at IT-departments to check for uptime or test connectivity. When no variables are altered, a ping is sent every second with a fixed amount of bytes using a single ICMP packet. Both the request and response should contain (somewhat) identical data.

'True' automized anomaly detection is very difficult to perform due to the dynamic nature of networks. The sample (reference) data received from Shell contains 2 million DNS requests. When performing a query that checks for DNS flows with more than 12 packets outgoing in a single flow (the aprox. amount required to set up a tunnel), 0,0005% of the two million records came back as a possible alert. However, the flow time was quite long as well (several minutes, even tens of minutes). The exact nature is still unknown, since it is not known if the traffic was 'bad' or not. The question still remains if these 'anomalies' are the cause of a flow-parsing bug (as explained on page 5) or if a long-lasting DNS flow was legitimately performed.

3.4. Examples; Other uses

Using NetFlow data can also be used to detect other types of malicious traffic; such as DDoS attacks or portscans. Both portscans and (traffic-based) DDoS attacks can be detected by means of anomaly detection. Alerts can be generated whenever a large

amount of traffic is detected to many unique (previously un-used) ports or when an 'abnormal' amount of traffic is detected to an active service. However, in the case of portscans another detection method can be implemented. Whenever an attacker attempts to detect open ports on a system with a filtering firewall, the bi-directional NetFlow dump will display non-responsive requests as seen in Figure 5. **Note:** As explained in the 'Background' section, the NetFlow collector being used tries to wait for finished or timed-out TCP sessions. When performing a half-open portscan (only checking if a SYN is received), the actual portscan flows will be stored when the time-out expires. This means realtime portscan detection is not possible with the configuration used during this research. During this research, the portscan was seen 20 minutes later.

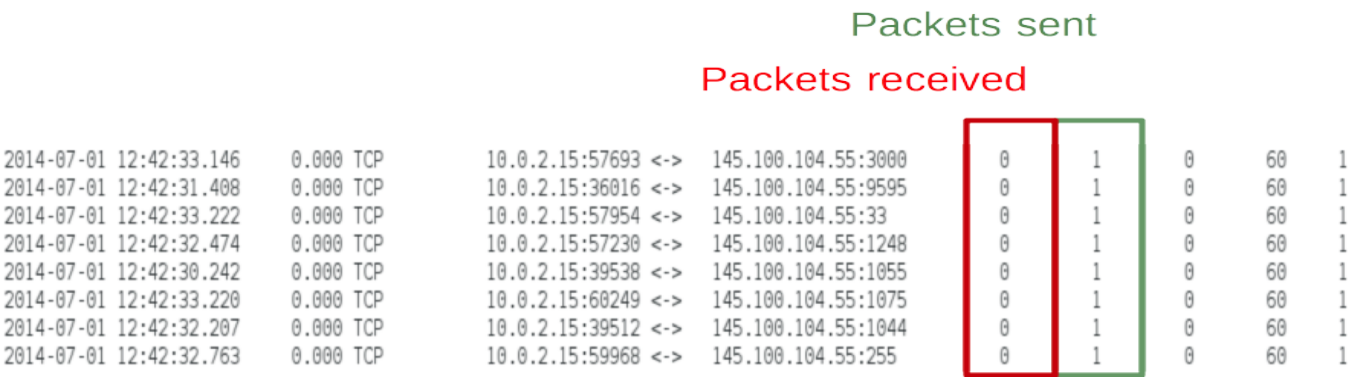


Figure 5: NetFlow dump snippet of a portscan

4. Conclusion

Is it possibe to detect Covert Channels via NetFlow (v5) data?
It is indeed possible to detect some of the popular covert channel / tunnelling packages available. However, NetFlow v5 only provides limited data and detection purely relies on flow metrics. If a malicious user manages to craft a covert channel that 'behaves' (ie. taking time and packet numbers into account), it will be much harder to detect.

What metrics can be used to detect Covert Channels?
The two examples mentioned in this report make use of the packet numbers and session time. Of the two examples mentioned, DNS is the most variable protocol; packet sizes and count differ frequently depending on the DNS requests. Abnormal ping behaviour could possibly be easier to detect. On all modern platforms (Windows/Linux/Mac) a standard ping is sent out with an interval of a single second per packet. In some cases, the exact amount of bytes stay the same. In short; to detect tunnels via these non-http protocols, the time/session length metric is most important, since the amount of packets and bytes sent in relation to the total time may indicate possible tunnel activity.

Can these detection methods be implemented in a function monitoring scheme?
Proprietary tooling already exists to detect abnormal activity on the network. ArcSight is able to analyse NetFlow for possible malicious behaviour. However, anomaly detection can be seen in it's broadest form; detecting sudden peaks in traffic, machines not receiving traffic at all, (D)DoS attacks etc. For this research a proof-of-concept detection tool was developed. The proof-of-concept compares known-good example data with recent network flows received by a NetFlow collector. For each unique protocol, different metrics were selected that are analysed for anomalies. Whenever all these metrics (ex. both time and packets) exceed the allowed border value compared to the known-

good, an alert is raised. In practice, this method is very difficult to perform. The PoC assumes that the flows are not sampled (as is often the case with NetFlow). Limiting the amount sample rate may introduce missed alerts. Next to that, depending on the NetFlow collector and parser, false-positives may be generated by aggregated flows.

5. Future work

During this research, no NetFlow sampling was used. Future research should test the effectivity of similar detection methods when NetFlow sampling is in place. Furthermore, many anomaly detection tools are already available. FlowMatrix [6] is one of the few open-source tools out there but is very outdated (2010). Future research could test the effectivity of other (proprietary) NetFlow analysis tools and see if they can detect popular covert channels without verifying packet content or using static signatures. Furthermore, the Shell sample-data used during this research was not verified (ie. know whether malicious traffic took place or not) and the known-good and known-bad data was generated based on limited network traffic. Future research could implement similar detection methods on a larger (and verified) dataset.

6. Annex

6.1. Proof-of-concept

To effectively test the dataset, a proof-of-concept web-interface was developed. The web-interface makes use of the functionality as explained in the anomaly detection section. The statistics page shows the averages, standard deviations and variable width of deviation for each unique protocol. A combination of multiple metrics (for alerting) can be selected here.

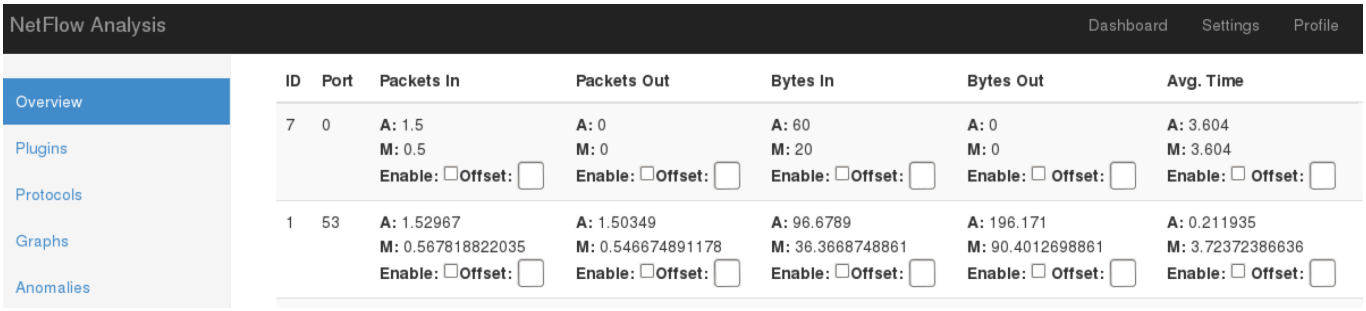


Figure 6: Screenshot of statistic page

Whenever an anomaly is detected based on the values seen in the statistics page, the Python module would push an alert to the database.

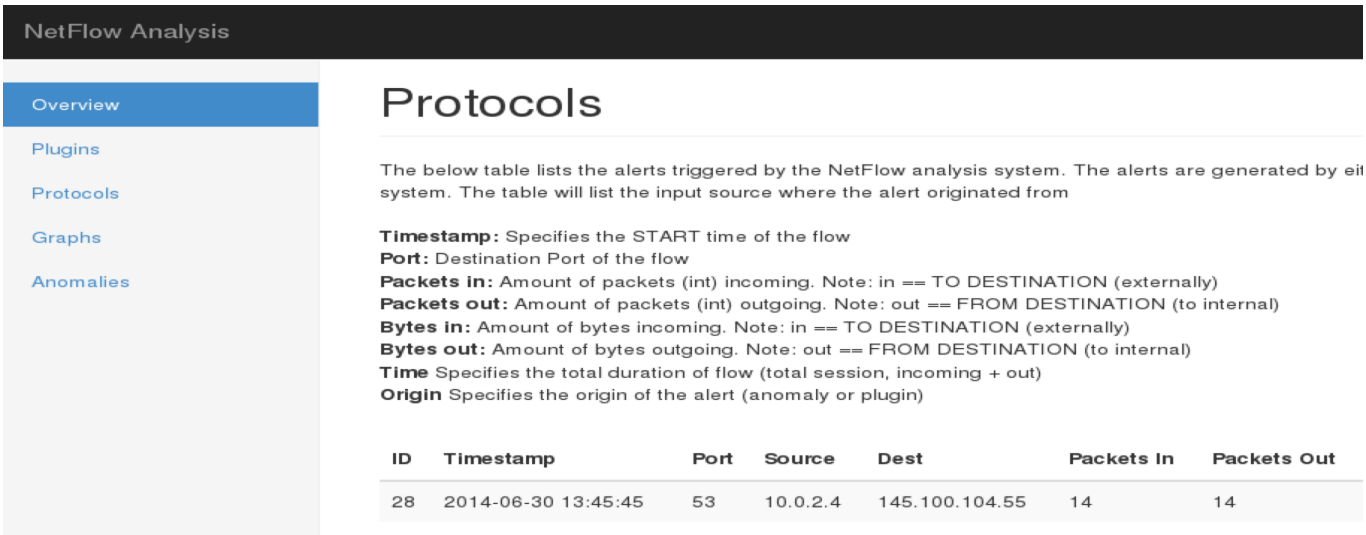


Figure 7: Screenshot of alerts page

References

[1] Various authors. Netflow. http://en.wikipedia.org/wiki/NetFlow, 2014.

[2] Justin Azoff. Netflow based intrusion detection system. https://pythonhosted.org/pynfdump/, 2009.

[3] Pao et al. Netflow based intrusion detection system. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1297037, 2004.

[4] Rick Hofstede. The network data handling war: Mysql vs. nfdump. http://eprints.eemcs.utwente.nl/18060/01/fulltext-1.pdf.

[5] Kryo. iodine dns tunnel. http://code.kryo.se/iodine/.

[6] AKMA Labs. Flowmatrix; network behavior analysis system. http://www.akmalabs.com/home.php, 2010.

[7] Damien Miller. Softflowd. http://www.mindrot.org/projects/softflowd/, 2011.

[8] N/A. Netflow v5 documentation. http://netflowv5.com, 2014.

[9] Daniel Romao Niels van Dijkhuizen. Ddos detection and alerting. http://delaat.net/rp/2013-2014/p47/report.pdf, 2014.

[10] Ntop. Netflow parser/collector. http://nfdump.sourceforge.net, 2013.

[11] Oracle. Mysql server. http://www.mysql.com, 2014.

[12] Daniel Stodle. Ping tunnel. http://www.cs.uit.no/~daniels/PingTunnel/, 2011.