Master System and Network Engineering

University of Amsterdam

Anomaly Detection on Internet Content Filter Data

Testing the effectiveness of statistical anomaly detection on user-agents

Author
P.T. van Bolhuis
peter.vanbolhuis@os3.nl

Supervisor
R. de Beer
ramses.debeer@shell.com

July, 2014

Abstract

The goal of this research was to find out what the effectiveness of statistical anomaly detection on user-agent strings is.

To increase effectiveness, a method was created to perform statistical analysis on the user-agents by splitting these into elements. By splitting the user-agent into multiple elements and checking these elements for anomalies, small differences in user-agents that would often occur could be removed. This helped to remove false positives that were not interesting.

Next, a scoring mechanism was used to score IP-addresses based on the amount of anomalous user-agent elements in their user-agent string. The more anomalous elements originated from an IP, the higher the score.

Plotting the host scores showed clear outliers when hosts are scored based on their user-agents. Outliers were often created by traffic from mobile phones, which mostly connected through proxies. Another found outlier was a host with a browser hijacker installed.

Performing this form of anomaly detection proves useful on a network where hosts are uniform. Networks with more variation in hosts (more mobile devices) create a larger amount of unique user-agents. This happens because of application using a unique user-agent when sending traffic to servers.

Contents

1 Introduction

This introduction functions to give a quick overview of the field of anomaly detection and user-agents. The last subsection will describe the relation of this research to Shell where it was performed.

1.1 Anomaly detection methods

Network anomalies are differences from *regular* traffic. These can be detected in multiple ways, namely statistically, knowledge-based and based on machine learning [1].

Statistical anomaly detection determines whether or not traffic is normal by comparing it to a previously generated profile. This profile holds statistics of parameters, such as amount of traffic, types of traffic, what IP-addresses it talks to and traffic rates. When current traffic is different from the previously determined 'normal', the NIDS can flag the traffic as anomalous.

Knowledge-based anomaly detection works with known-bads and known-goods. Setting up a knowledge-based NIDS can be time consuming, which is why it is often combined with either statistical or machine learning-based. Knowledge-based systems can also work with signatures, which are known-bads, the popular NIDS Snort uses signatures [2].

The last method, machine learning, is based on algorithms that take new information in and constantly adjust their behaviour based on it. These machines are resource intensive if they are configured to alter their behaviour based on live information. Some of the used machine-learning models are *Bayesian networks* and *Markov models*.

The Bayesian networks looks for probabilistic relations between certain variables of network traffic. This is often combined with statistical modeling of the traffic. However, the results of naive Bayesian networks are comparable to threshold-based systems, while requiring more resources [3].

Markov models work based on state transitions. A training set will create a model of probability on transitions between states. Anomaly detection is then done based on the probability combined with a fixed threshold. Unlikely transitions between states will trigger an alert.

1.2 User-agents

User-agents are programs that perform tasks on a computer, for example a web browser. This browser identifies itself to servers via the user-agent string and includes information about the system in this string. Information that can be included is the operating system, installed frameworks, browser version and compatibility information.

Some malware uses the HTTP protocol for command and control traffic and tries to fake the user-agent string. Often this user-agent string is hard-coded into the malware [4], and thus static. In some cases the user-agent string can be used for cross-site scripting (XSS) or forms of SQL-injection[5].

User-agent strings are part of the HTTP protocol, and are defined in RFC 2616 [6].

Note that in this report the term user-agent and user-agent string will be used interchangeably, but mean to indicate the user-agent string.

1.3 Relation to Shell

This research was performed at Royal Dutch Shell - Shell Information Technology International (SITI), in Rijswijk. SITI takes care of the incident detection and response within Shell. SITI was interested in malware detection through several methods, one of which was through user-agent information. The research would prove the feasibility of certain detection methods.

If a method proved feasible, it could be applied to the company network. The network at Shell was split between a guest network and the office network. The guest network was, as the name implies, for the guests, but also for mobile devices. The office network was for company laptops and desktops. Mobile devices were not allowed on this network.

For both networks, all clients connected to websites through an Internet Content Filter (ICF), which filtered the traffic. Access to certain sites was blocked. Traffic was logged on this ICF, and these logs were used for the data-set.

1.4 Related work

Anomaly detection was first thought of by James Anderson in 1980 [7]. Anderson suggested that anomalies could be detected in metrics like computing time. This was host based detection which was processed in batch daily.

Since 1980 a lot of research has been done into detecting both network and host-based anomalies in different ways [1]. Most research into statistical network anomaly detection has been based on header information of packets. One example is the research by Mahoney and Chan [8], which checks headers of Ethernet, IP and TCP/UDP for anomalies.

Other research has been done on detecting anomalies in the payload of packets [9]. In this research the complete payload size was used to create a frequency distribution and the standard deviation. If packets differed too much, they were flagged.

One research, done by N. Kheir [4] performed anomaly detection on user-agents. The used method was knowledge-based; blacklists and whitelists were created to match user-agents against. It managed to obtain a very low false-positive ratio and managed to find new botnet infections based on a detection pattern.

1.5 Research question

The research by N. Kheir [4] indicated that low false-positive ratios could be achieved by using signatures, while still detecting malicious traffic. This research hopes to have similar results by statistically comparing the user-agent fields.

Machine learning was too complex to start with and machine learning itself also relied on statistical detection (in the case of Bayesian network models) [3]. Therefore this research will try to detect anomalous traffic through statistical analysis of the user-agents. No previous research specifically checking the effectiveness of statistical detection on user-agents could be found.

The research will try to answer the following question: *What is the effectiveness of statistical anomaly detection, when applied to the user-agent string?*.

To answer this question, the following sub-questions are to be answered:

- How can user-agents be statistically analyzed effectively?
- How can false-positives be excluded?

1.6 Scope

Anomaly detection can be applied to many variables of network traffic. The anomaly detection in this research will be limited to the user-agent strings that are reported by the Internet Content Filter at Shell. This means the analysis will not be performed on a 'live' set of data that comes through, but rather on an exported set.

The research will also limit itself to statistical methods of detection. There will be no fingerprinting or white-listing of traffic.

2 Research

This section will describe the research performed. It will go into the data-set used and what methods were used to extract data from this set. Furthermore it will analyze some of the results.

The code used to process the output of the ICF can be found on `https://github.com/PTVB/agents`.

2.1 Data-set

As mentioned in the introduction, the data-set originated from the Internet Content Filter (ICF) server. This server logged all HTTP traffic, and blocked sites based on filtering rules.

The set of data was exported as a comma separated values (CSV) file. The file contained source information and destination information (IP-addresses), the file accessed (/index.html for example), the user-agent string and some other information about action taken by the ICF and based on what rule.

Only the source IP and the user-agent information from this set were used.

2.2 User-agents

To avoid one client distorting the statistics of user-agents, the user-agents were linked to source IP-addresses. Per IP, all user-agents that were unique for that IP were saved. This prevented one IP with many requests from not standing out, while the user-agent was unique to this IP.

Simply checking for unique (complete) user-agents showed a large amount of unique ones. This made it unusable for statistically detecting anomalies, since every unique user-agent would show up as an anomaly. However, a lot of elements in the user-agent were similar, sometimes only a version number differed, which wasn't necessarily an anomaly.

2.3 Splitting on elements

The small differences in complete user-agent strings led to splitting user-agent into elements. Elements were split on brackets, commas and semi-colons. In the user-agent string below, this would for instance results in "Mozilla/4.0" being one element and ".NET CLR 2.0.50727" being another one.

```
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64;
Trident/7.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729;
.NET CLR 3.0.30729; Media Center PC 6.0; eSobiSubscriber
2.0.4.16; BRI/1; MAAR; .NET4.0C; AskTbORJ/5.15.9.29495;
.NET4.0E; BRI/2) Funshion/1.0.0.1
```

All elements were then counted, leading to a list of elements with their number of occurrences over all IP-addresses. Plotting this, showed the graph shown in Figure

Clearly visible in Figure 1 is that a few user-agent elements are very common, but then at some point there is a large number of uncommon elements. Most of these uncommon elements originated from anti-virus update mechanisms, as well as from mobile phones. The mobile phones sometimes used an application specific user-agent for applications that contacted a server. Figure 2 shows the same graph, zoomed in on the first 50 elements.

A graph with the total number of elements per IP can be found in Appendix A.

2.4 Scoring host anomalies

For this research, only elements that occur very little are used to indicate an anomaly. Hosts will be scored based on them having anomalous elements.

While lack of often-used elements could also indicate an anomaly, these are not used in the system of scoring hosts. The reason is that anti-virus updating mechanisms used unique strings without any other information. Scoring these based on the lack of elements, would falsely give hosts a higher score, creating false-positives.

Deciding what elements increase a hosts score is done by setting the amount of unique element occurrences. The lowest x element-occurrences will then cause an increase of host score. The choice for x can be made based on

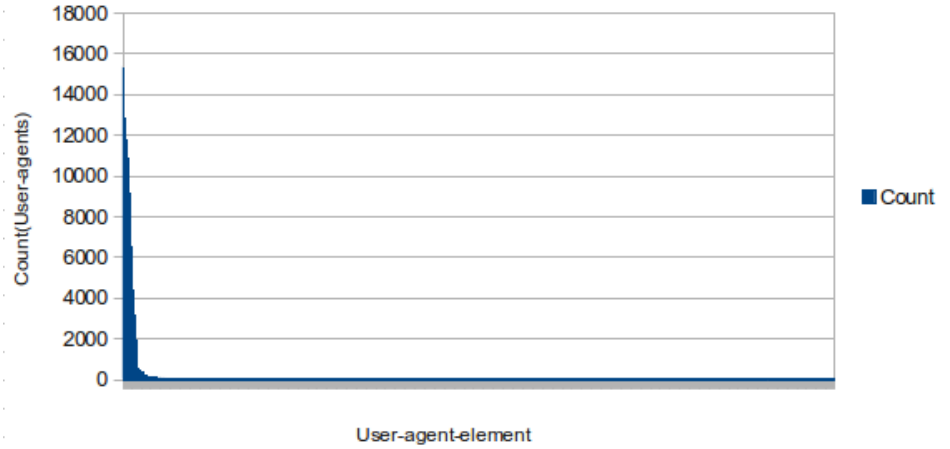


Figure 1: Count of user-agent elements

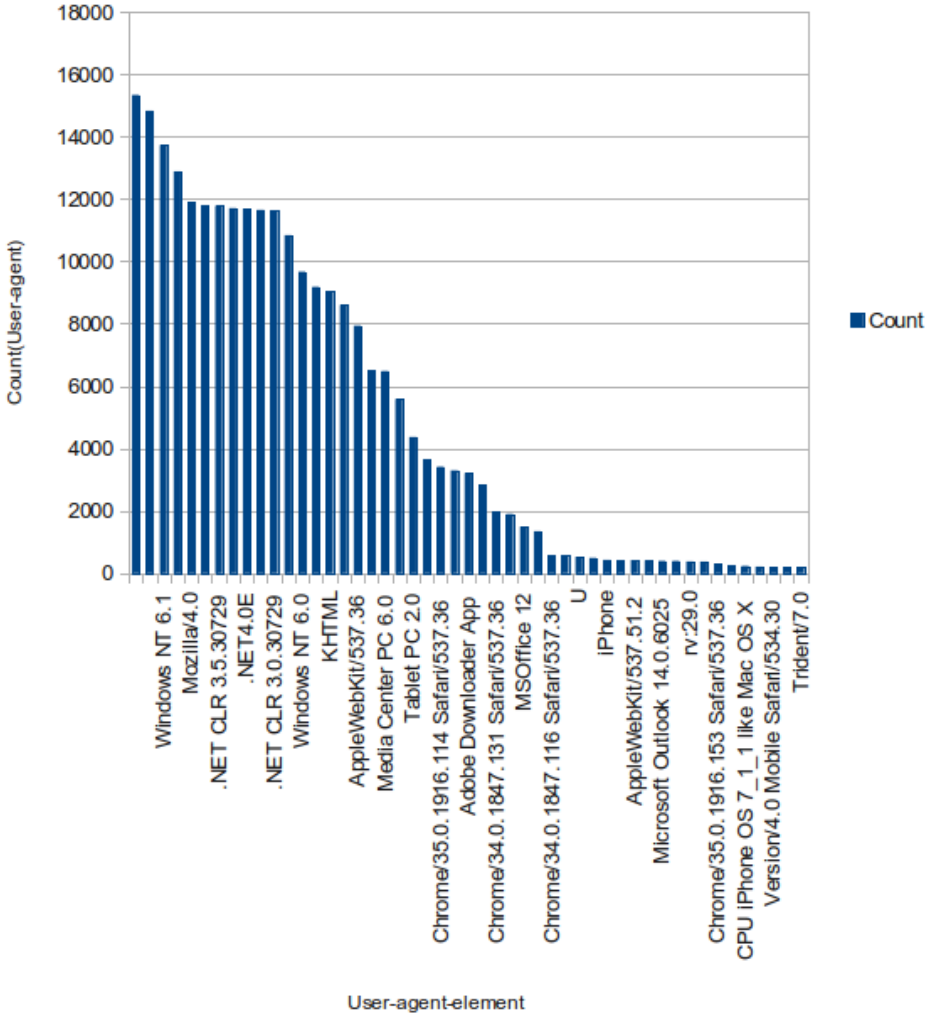


Figure 2: Count of user-agents elements, zoomed in on first 50 elements

Figure 1, the point where the graph clearly starts to flatten out is a good point.

As an example, see Table 1, where x is set to 3. If this was the entire set, the element *DragonUpdater* would be in the lowest three occurrences (1, 2 and 6), and would therefore increase a hosts score. For every marked element a host has, a hosts score will be increased with 1, see Table 2 for scores based on Table 1. A weighed score would likely be a more efficient way to score hosts. However, for the proof-of-concept, this was not needed. A weighed score for anomalies is recommended for the future work.

Table 1 User-agent elements + whether or not they create a trigger

User-agent element	# occurrences	Increases score
compatible	14823	No
Windows NT 6.1	13723	No
Mozilla/5.0	12882	No
DragonUpdater	6	Yes
Device	2	Yes
SPID=3755	2	Yes
Edition Next	2	Yes
AppleWebKit/532.9	1	Yes

Table 2 Host scores (example data)

Host	User-agent string	Score
10.100.0.1	Mozilla/5.0 (compatible; AppleWebKit/532.9)	1
10.0.10.20	Mozilla/5.0 (DragonUpdater; Device)	2
10.13.37.30	Edition Next (AppleWebKit/532.9; Device: SPID=3755)	4

2.5 Alert threshold

Scoring hosts with the previously mentioned mechanism will result with each host having a score. A visualization of these scores on the provided data-set is shown in Figure 3, where the value of x was set to five.

Comparing the result with a raw count of elements per IP, as shown in Appendix A, will show that the four highest outliers still exist[1]. However, all outliers below 500 have been reduced to scores practically equal to all others.

[1]The order is different due to the way the written program processes the data

This shows that removing user-agents that occur often is an effective way to reduce the score of hosts, and will therefore make setting a threshold easier. It also reduces the chance on false-positives. On the other hand, IP-addresses with a lot of user-agents originating from it, will likely score a lot higher than others. The amount of user-agents originating from one IP has a significant impact on its score.

The graph in Figure 3 shows four very clear outliers, labeled A, B, C and D in the figure. Any threshold between 10 and 150 would show these four as the only outliers. Thanks to this large difference between the hosts, false-positives can be easily excluded. Note that the network is fairly uniform, with hosts having either a little score, or enormous outliers. Networks with less uniformity will show smaller differences between normal hosts and outliers. The smaller difference in score would make detecting outliers more difficult and could potentially raise the number of false-positives.

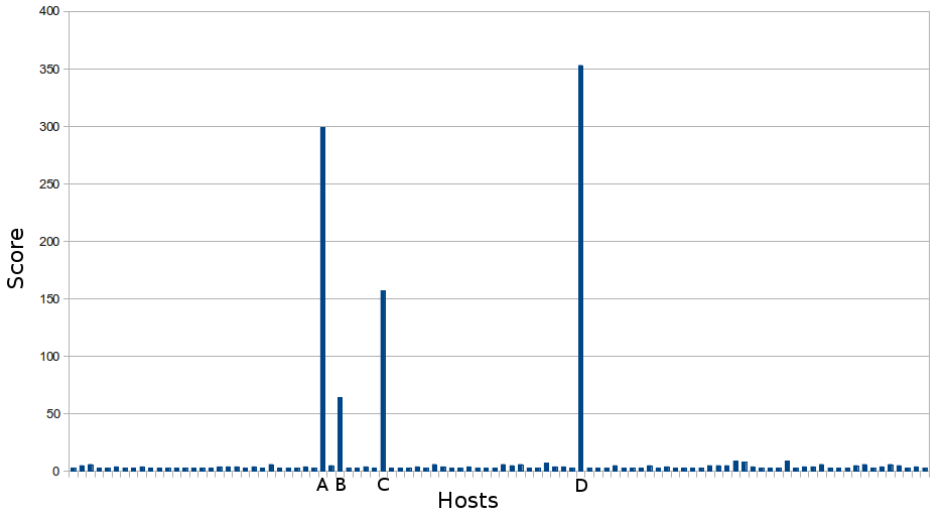


Figure 3: Scores per host, only for hosts with score >1

2.6 Verification

As a final step, the found outliers were inspected as to what caused them to stand out. Table 3 shows the findings.

It shows that two out of four (C and D) were proxy servers. These servers aggregated data from multiple clients and therefore stood out.

Host A was a host that sent out multiple user-agents, different applications (mostly games) all used their own user-agent. Most notably, this client

also had software installed that identified itself with "`SkyNet/1.2.0`". This Skynet used the user-agent as some form of covert channel, exporting information about the phone, including phone number, Google account and the IMEI number. This information clearly stood out, as almost all parameters were unique to this host. As the data-set was gathered from a network where phones were not supposed to be, this was a compliance incident.

Host B was an IP address that was leased with DHCP, so multiple clients had used it. One of these clients was infected with the Conduit browser hijacker. It showed up because of unique strings and special user-agents, but also because multiple clients had used this IP.

Table 3 Verification of anomaly on hosts

Host	**Result of inspection**
A	Host was a phone, which is a compliance incident on this network.
B	Host infected with Conduit browser hijacker, infection was missed by AV
C	Host was a proxy, combining traffic of multiple clients
D	Host was a proxy, same as C

3 Conclusion

The goal of this research was to test the effectiveness of anomaly detection on transmitted user-agent strings.

User-agents can effectively be analyzed by splitting them up into elements. By scoring hosts or IP-addresses on the amount of anomalous elements, a large difference appeared between the hosts in the used data-set. By applying this method, scores for hosts with a lot of common user-agents will not stand out more than hosts with a few common user-agents.

Because of the large differences in score, a threshold can easily be placed. Based on this threshold, it will decided when a score triggers an alert. While the large difference will make exclusion of false-positives easy, the difference is due to the fact that the network is very uniform. Less uniform networks will give a smaller difference between positives and negatives, making the descision for placing the threshold more difficult.

Found hosts appeared to be mostly phones and proxies. This is due to the fact that applications on phones sometimes identify themselves with a user-agent that is unique to the application. The more apps that do this, the more a host will stand out. The proxies stood out because a lot of (different) clients connected through it. These clients are likely to have differing user-agent strings. If this method would be applied to network traffic that does not contain aggregators, such as network proxies, it would filter out phones and computers with anomalous packages installed.

Software on computers that installs browser-addons may identify themselves by adding an element to the user-agent. This will also show as an anomaly. An example of which was shown by the detected anomaly for the host with the browser hijacker

Malware that is installed on computers will sometimes try to fake a user-agent, which may stand out, but this depends on the user-agent it is faking. Other (malicious) functions that the user-agent string may be used for, such as a covert-channels would be detected by a statistical method.

The used method will likely pick out hosts with a lot of different software packages installed, but will not necessarily pick out hosts infected with malware. Using this method on a network with uniform hosts, will show the best results.

In conclusion: statistical anomaly detection on user-agents can provide an indication of hosts having installed malicious software. However, this detection works best in a uniform environment.

3.1 Future work

This research created a very basic method for scoring host anomalies based on their user-agents. By giving hosts a weighed score, based on how rare a user-agent element is, better results may be achieved. This will especially show in a less uniform network environment.

4 References

[1] G. Maci-Fernndez E. Vzquez P. Garca-Teodoro, J. Daz-Verdejo. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28:18–28, 2009. http://web.info.uvt.ro/~dzaharie/cne2013/proiecte/aplicatii/intrusion_detection_systems/IDS_general.pdf.

[2] Snort home page. http://www.snort.org.

[3] W. Robertson F. Valeur C. Kruegel, D. Mutz. Bayesian event classification for intrusion detection, 2003. http://www.cs.ucsb.edu/~chris/research/doc/2003_07.pdf.

[4] N. Kheir. Behavioral classification and detection of malware through http user agent anomalies. *Journal of Information Security and Applications*, 18:2–13, 2013.

[5] Adrian Crenshaw. Xss, command and sql injection vectors: Beyond the form.

[6] R. Fielding et al. Hypertext transfer protocol – http/1.1. RFC 2616, 1999. https://tools.ietf.org/html/rfc2616.

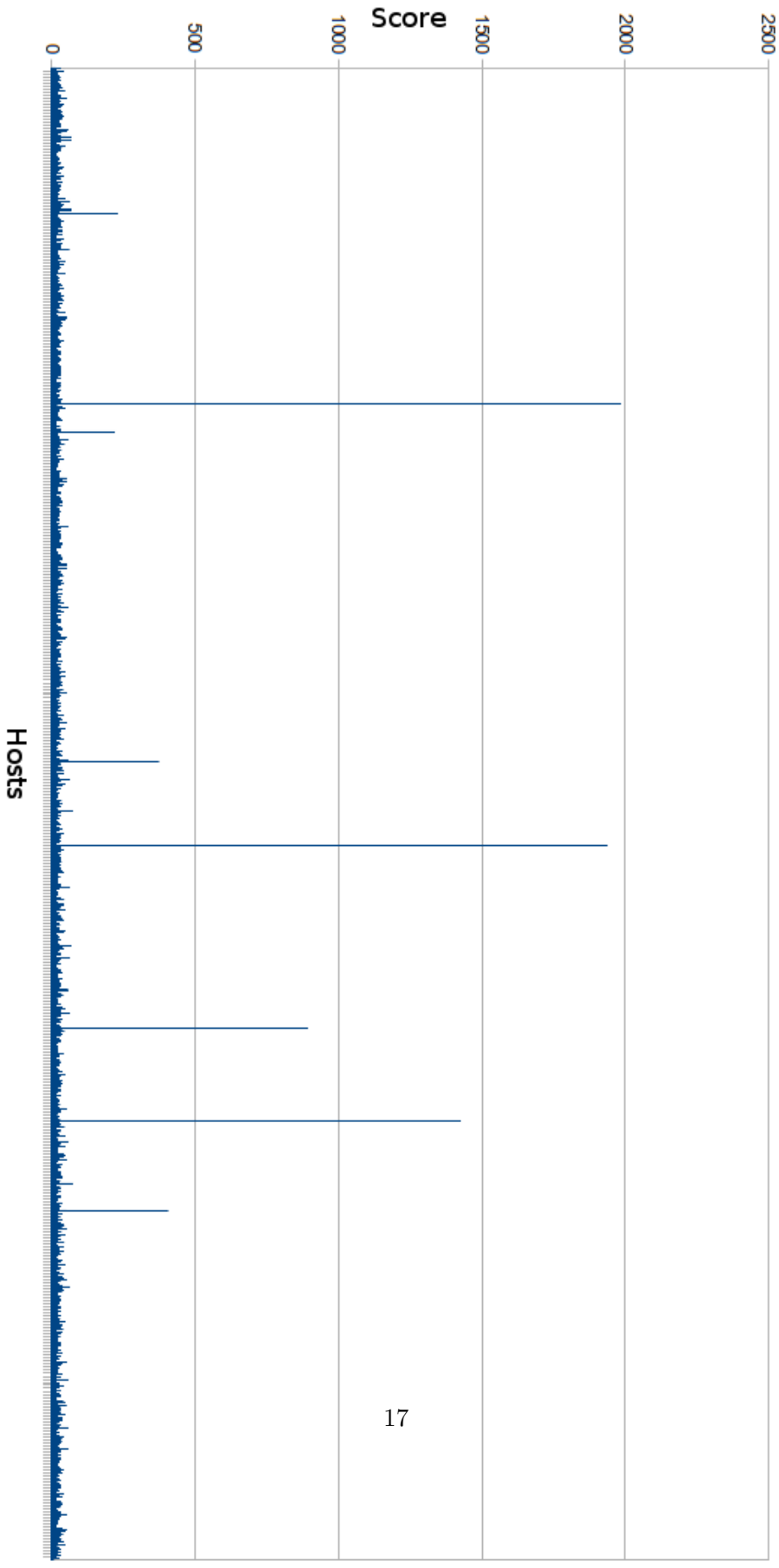
[7] J.P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Company, 1980. http://csrc.nist.gov/publications/history/ande80.pdf.

[8] P.K. Chan M.V. Mahoney. Phad: Packet header anomaly detection for identifying hostile network traffic. Master's thesis, Florida Institure of Technology, 2001. https://repository.lib.fit.edu/bitstream/handle/11141/94/cs-2001-04.pdf?sequence=1.

[9] S.J. Stolfo K. Wang. Anomalous payload-based network intrusion detection. Master's thesis, Columbia University, 2004. http://academiccommons.columbia.edu/download/fedora_content/download/ac:125705/CONTENT/RAID4.pdf.

Appendix

A Total number of elements per IP


Score
0
500
1000
1500
2000
2500
Hosts