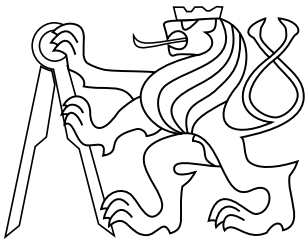
master's thesis

Network Service Anomaly Detection

Ivan Nikolaev



June 2014

Martin Grill

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Control Engineering

Acknowledgement

I would like to thank my advisor Ing. Martin Grill for dedicating a lot of his time and patience to help me write this thesis. I would also like to thank Ing. Tomáš Pevný, Ph.D. for his supervision on work on service modelling. I would like to thank Mgr. Jan Kohout for his collaboration on service modelling research.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Abstract

Tato diplomová práce studuje detekci a modelování běžně používaných síťových služeb. Informace o komunikaci na síti se získává v podobě statistických agregací ve formě NetFlow a proxy logů. První část této práce popisuje matematické modely použité k detekci síťových služeb. Druhá část se zabývá modelováním chování jednotlivých uživatelů těchto služeb a detekcí anomálních uživatelů. V poslední části jsou prezentované matematické modely experimentalně ověřené na datech z realných počítačových sítí.

Klíčová slova

NetFlow; Síťová služba; Detekce anomálií

Abstract

This thesis conducts a thorough study on detection and modelling of commonly used network services. Information about network communication is provided as statistical aggregates in the form of NetFlows and proxy logs. Mathematical models are used in order to detect network services as described in the first part of the thesis and then model the behaviour of service users and detect anomalous ones, described in the second part. In the last part we analyse the effectiveness of the mathematical models presented in the thesis using experiments on real network data.

Keywords

NetFlow; Network service; Anomaly detection

Contents

Abbreviations

In this thesis several abbreviations are used. Their meaning is explained below.

AUC	Area Under the Curve
C&C	Command and Control
CPU	Central Processing Unit
CTU	Czech Technical University
DGA	Domain Generation Algorithm
DoS	Denial of Service
DDoS	Distributed Denial of Service
EM	Expectation-Maximisation (algorithm)
FIN	Finish (TCP flag)
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDS	Intrusion Detection System
IP	Internet Protocol
RST	Reset (TCP flag)
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network

1 Introduction

In today's world the internet and IP networks have become an integral part of our society. Everything from basic social interactions to complex multinational operations has come to rely on computer networks. It is estimated that there are currently more mobile devices than human beings in the world [1].

The reliance on modern technology gives us a lot of advantages and opportunities but also creates new risks. Confidential data of individuals and corporations is more exposed to unauthorised access than ever before in history. A person skilled in network penetration can gain access to a corporate network and steal valuable information within days or even hours without leaving his home or even getting off his chair. Intellectual property theft is an ever-growing threat [2]. Worldwide costs due to cybercrime are estimated to be in the region of one hundred billion dollars every year [3]. This level of importance combined with a severe lack of security experts [4] makes this a very attractive field for research.

Traditional approach to network security is based on pattern matching — firewall rules, security policies, etc. As networks grow larger the amount of traffic increases. New types of services require new types of network protocols. The increase in network size, service and protocol types results in a huge number of new possible attack vectors. Threat detection methods based on deep packet inspection and fingerprinting are becoming infeasible due to their high computational demand and the constantly changing behaviour of malware and the attackers.

Increasingly, methods based on machine learning that leverage behaviour analysis principle and use aggregated information about network traffic are employed. These methods build statistical models of the network traffic and report deviations from the models. The advantage of these methods is their ability to detect new threats, their adaptability to specific networks and their use of aggregated data which lowers computational costs.

Motivation

This thesis is a thorough study of network services and their users inside large corporate networks which is part of a much larger field of network security. In the recent years there has been a constant increase in availability, variety, functionality and reliability of cloud services. Examples are many: Dropbox, Google Drive, Twitter, Facebook, Box, Salesforce, Microsoft Online and many others. More and more users come to rely on these services in their work on a daily basis. The network perimeter is increasingly becoming harder to define and protect, with a lot of users using cloud services outside the network, connecting to the network through VPN and constantly bringing in and out different devices like laptops, tablets and mobile phones. As the result, there is more and more opportunities for sensitive information to leave the network by means of upload to an outside cloud service, either through carelessness or ill–intent.

Another aspect of service usage are the internal services. These are usually abundant on the network and many of them may contain sensitive information such as customer

records, development code, company mail, financial records and so on. All this information can be of great value in espionage. It is clear that network services are of interest to network intruders.

The task of monitoring service usage is currently in high demand. There are different startup companies around the world, the best of them founded and led by PhD graduates in machine learning, statistics and mathematics, that specialise in monitoring cloud services. Skyhigh Networks [5], Elastica [6] and Netskope [7] are the three examples that come to mind.

The motivation for this thesis is to perform a deep study of network services behaviour, for both cloud and internal services. From being able to find them on the network, to monitoring their usage by the users and reporting abnormal behaviour. This thesis consists of three main parts: detection of services explained in Chapter 3, modelling of users' behaviour on those services in Chapter 4 and Chapter 5 which presents the experimental results.

2 Data Sources

Traditional approach to network security is deep packet inspection. This means that individual packets are captured. Every packet is opened and matched against different signatures to look for malicious behaviour. This approach has several disadvantages. The most obvious one is extensive computational complexity. The requirement to inspect every packet going through the network creates very high CPU load and makes this type of monitoring infeasible on large scale networks.

Another disadvantage concerns privacy. Many consider it unethical to inspect other people's communication by opening individual packets, akin to steam-opening other people's letters. Also, many modern protocols use encryption which prevents a third party from reading the transferred data. It is possible to circumvent encryption by doing a man-in-the-middle attack and serving the communication endpoints with fake encryption certificates [8, 9, 10], however this raises privacy concerns even higher. It also potentially weakens security and is illegal in some countries, e.g. Germany. There have also been reports of deep packet inspection devices themselves being vulnerable to specially crafted packets [11].

An example of an HTTP packet, as seen inside a popular network monitoring and packet inspection tool Wireshark is shown in Figure 1.

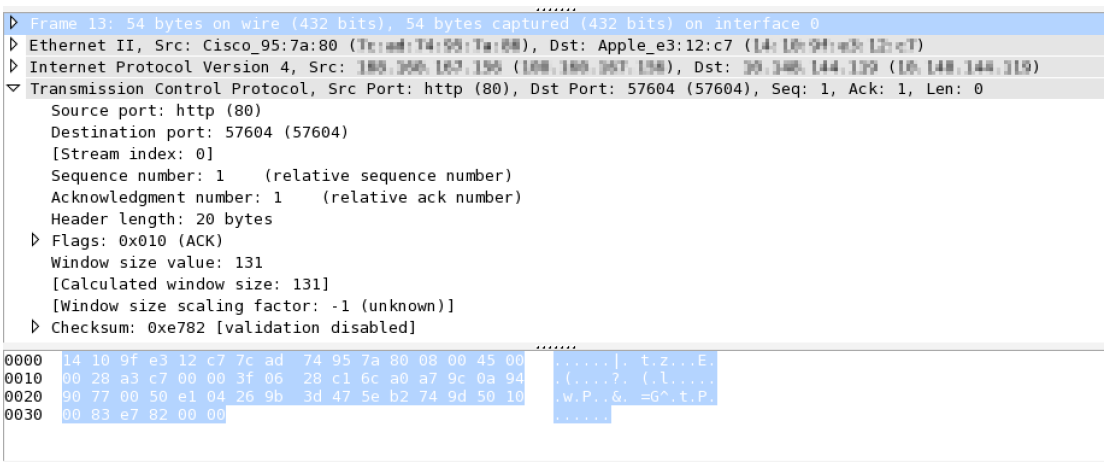


Figure 1 A screenshot of an HTTP packet displayed in a network monitoring tool Wireshark

An alternative to deep packet inspection is using statistical aggregates about the transferred packets. These aggregates only store the meta-data about communication, erasing a lot of the privacy concerns and easing computational load. The main advantage of the aggregates is also their main disadvantage — very limited information. It is possible to create behavioural models and learn statistical patterns using this information, but when an anomaly is found it is often hard to explain its cause in detail. But despite that, the advantages provided by this type of data sources are great enough for them to be used more and more by modern IDSs.

Table 1 NetFlow sample

Date flow start	Duration	Proto	Src IP Addr:Port		Dst IP Addr:Port	Flags	Tos	Packets	Bytes	Flows
2013-02-07 02:33:26.226	26.439	TCP	192.168.10.79:6667	->	66.166.77.146:37772	.A.RS.	0	16	704	1
2013-02-07 02:33:49.961	0.188	TCP	192.168.10.79:6667	->	66.166.77.146:39763	.A.RS.	0	4	176	1
2013-02-07 02:33:50.459	1.116	TCP	50.63.180.209:37244	->	192.168.10.79:6668	.AP.SF	0	10	654	1
2013-02-07 02:33:50.137	0.000	TCP	192.168.10.79:57180	->	50.63.180.209:113	....S.	0	2	120	1
2013-02-07 02:33:35.577	14.567	TCP	66.166.77.146:38587	->	192.168.10.79:6667	.A.RS.	0	12	512	1
2013-02-07 02:33:32.860	19.297	TCP	192.168.10.79:6667	->	66.166.77.146:38337	.A.RS.	0	12	528	1
2013-02-07 02:33:53.142	0.000	UDP	168.95.1.14:53	->	192.168.10.9:57368	.....	0	2	400	1
2013-02-07 02:33:41.713	12.261	TCP	192.168.10.79:6667	->	66.166.77.146:39114	.A.RS.	0	8	352	1
2013-02-07 02:33:52.276	0.190	TCP	66.166.77.146:39951	->	192.168.10.79:6667	.A.RS.	0	6	256	1

In this thesis two types of data sources are used: NetFlows and proxy logs. Both of them are a form of statistical aggregates.

2.1 NetFlows

A NetFlow [12, 13] is an aggregate of packets with the same source port, source IP, destination port, destination IP and protocol. The aggregate contains information about the number of packets transferred, the sum of bytes transferred by all packets, a logical OR of all the TCP flags, the starting time of communication and the duration of communication.

NetFlow collection is conducted by NetFlow probes. A NetFlow probe monitors all packets going through it. It has a cache where it aggregates information about the packets going through it. When a packet with a RST or FIN flag appears it purges the corresponding entry from the cache and creates a NetFlow with the aggregated values. It also purges the entry from the cache if no new packet arrives within a timeout window. Many common routers and switches are capable of generating NetFlows.

In this thesis the NetFlows used are processed in five-minute batches. The choice of the interval is based on related works [14, 15] which show that this interval gives optimal performance for detection. Table 1 shows a sample from a NetFlow batch. Each line represents a single NetFlow.

2.2 Proxy Logs

Proxy logs provide information about web communication. They are created by web proxy servers that the users of the monitored network are made to use, either explicitly or latently through network configuration. An example of a proxy server capable of producing this type of logs is Squid [16].

Each line of a proxy log represents a request made by a user to a web service. It contains the IP address and identity of the user, the IP address and domain name of the server. For HTTP it contains the URL of the request, but not for HTTPS. It also contains the User-Agent information provided by the user during the request and the Referrer field which contains the information about who referred the user to make a request to that particular server with that particular URL. It then contains information about bytes uploaded/downloaded, starting time of communication and duration of communication. There can be additional fields depending on the configuration of the proxy server. These are not considered in this thesis. Proxy logs do not show any data sent by the user or returned by the server, only the volumes transferred.

In the following chapters we will use the term flow which can mean either NetFlow or proxy log. As the information extracted from NetFlows and proxy logs is the same,

apart from the feature extraction process, NetFlows and proxy logs can be viewed as equal.

2.3 Networks

Datasets with data from four different networks were used in this thesis. The four networks are of different sizes, have different IP ranges, amount of users and services. The first network is the CTU university network. From this network we have NetFlow data generated from the traffic inside the subnet of Czech Technical University in Prague with approximately 1000 users. The other three networks are large corporate networks with over ten thousand users in each network.

3 Service Detection

In this and later chapters we will often use the terms endpoint and service. We define an endpoint as an IP–port–protocol triple. Service is an endpoint that passively waits for other endpoints to start communication with it and only responds to communication initiated by other endpoints (clients). A service never initiates communication itself. It is common for a service to have more than one client.

An important challenge in network security is detecting services in use on the network. This is important for several reasons. The first reason is simple network analytics. Server–client model is the classic communication model used for the majority of network communication, main exception being peer–to–peer communication. Therefore, by finding the services that are actively used in a network, we can find out what kind of communication is going on inside that network, what kind of communication the network is capable of and what kind of communication the users utilise the most.

Detecting services is also very important from security perspective. Each service is a potential vulnerability. Given that it is possible to run a service on any computer in the network (unless prevented by a very strict firewall set up), unexpected vulnerabilities can present themselves anywhere in the network at any time. Also, running some services on some endpoints can be against company policy. It is therefore necessary for a network administrator to have an overview of actively used services on the monitored network.

Another reason for service detection is the possibility of doing detection of anomalous usage of those services. An IDS can detect active services on the network, then monitor the usage of those services by different users over a period of time. It can then establish models of each service's usage and report users that deviate from those models as anomalous.

It is important to note that for this thesis service detection only makes sense in the context of NetFlows which give low-level information about communication on IP level. For proxy logs, the service is always clearly given by the nature of proxy log creation. Also, proxy logs only provide information about web communication and not other types of services.

3.1 Related work

Previous work on service detection using NetFlows has been done by Berthier et al. [17] and Vaarandi [18]. Berthier et al. [17] performs service detection using NetFlows by creating several heuristics based on flow timings, port numbers, port numbers greater than 1024, port numbers that are well-known, number of distinct ports related to a given endpoint, number of distinct IP addresses related to a given endpoint and so on. The heuristics are then combined using a Bayesian network which gives the probability of an endpoint being a service. The biggest problem with that is assuming that services will use low ports and well-known ports. This simply does not have to be so. A service can run on any port whatsoever and it would be easy to trick this algorithm by using a high unknown port number in order to avoid discovery.

Vaarandi [18] does a somewhat simplified version of what is done by Berthier et al. [17]. They use privileged and unprivileged ports and timestamps in order to establish whether an endpoint is a service. This suffers from the same problems as Berthier et al. [17].

Our service detection algorithm focuses on behaviour of endpoints and does not use any prior knowledge of which ports the services should run on. This makes it more general and also more useful in the field of anomaly detection as it is behaviour-oriented and not signature-oriented.

3.2 Request-response matching

As was mentioned previously, services use server-client communication paradigm. This means that a service waits for a client to contact it. When a client contacts the server (sends a request) the server reacts to it (sends a response). This is the basis of server-client communication, therefore the first step in detecting services is finding request-response pairs.

A request-response pair of flows is a pair of such flows where one flow is the reaction to the other one. Such flows will have the same protocol and their IP-port source and destination pairs will be reversed. When we find such a pair we call it a request-response pair. The flow with the smaller starting timestamp is assumed to be the request. Flows that do not have a request-response pair are said to be requests without responses. An extensive study of requests without responses as well as request-response pair matching was done by [19].

Algorithm 1 presents a simple algorithm for matching request-response pairs. It iterates over flows sorted by starting time, putting new flows in the request list. Each new flow is matched against the flows in the request list. When a match is found it is said to be a request-response pair and the request is removed from the list. In case no match is found the new flow is put in the request list. At the end of the run the leftover flows in the request list are said to be requests without responses.

Algorithm 1 Simple Request-Response Matching Algorithm

```
F ← a list of all flows sorted by starting time
R ← an empty set of requests
P ← an empty set of request-response pairs
for all f ∈ F sequentially do
  if reverse flow to f (revf) is in R then
    revf is request
    f is response
    remove revf from R
    put pair f and revf in P
  else
    put f in R
  end if
end for
P contains request-response pairs
R contains requests without responses
```

3.3 Parallel request-response matching

The problem with the simple request-response matching algorithm is slow performance for large volumes of NetFlows. Parallel request-response matching algorithm was designed in order to address performance issues of Algorithm 1.

The algorithm works by splitting the NetFlows into smaller chunks and working on those in parallel. This is possible because the sum of source and destination port will be the same for both NetFlows in a request-response pair, since reverse flows have reverse source and destination IP-port pairs. Parallel request-response matching is described in Algorithm 2.

Algorithm 2 Parallel Request-Response Matching Algorithm

$h(f)$ sum of source and destination ports and starting time of a flow
$F \leftarrow$ a list of all flows sorted by h. Parallel sort is used.
split F into c chunks F_n. c depends on configuration.
boundaries of chunks F_n are chosen so that any port sum is only found in one chunk
apply Algorithm 1 to each chunk F_n, get P_n and R_n
$R \equiv R_1 \cup \cdots \cup R_c$
$P \equiv P_1 \cup \cdots \cup P_c$

The outputs of Algorithm 1 and Algorithm 2 are equivalent. Algorithm 2 is significantly faster for large batches of NetFlows. A comparison of the performance of the two algorithms is provided in Section 5.1. It is important to note that due to a large number of possible combinations of source and destination ports, the batch can be split into an arbitrarily large number of chunks, allowing for fast processing of giant NetFlow batches, if hardware permits.

It is therefore beneficial to use parallel request-response matching where large numbers of NetFlows are processed and performance is important.

3.4 Request-response anomaly detector

Using just the request-response pair matching algorithm, it is already possible to build a simple anomaly detector that performs surprisingly well for several types of malicious behaviour.

Request response anomaly detector is a detector of anomalous behaviour in the network. It is based on a simple principle. It calculates the amount of requests without responses that each endpoint makes. The endpoints that have a high amount of requests without responses are considered anomalous.

The detector uses request-response pair matching in order to identify requests without responses. It then calculates the amount of requests without responses for each endpoint. It uses those values to create a model of the network by calculating their mean and standard deviation. The anomaly values are based on the distance from the mean measured in standard deviations.

We create the network model by calculating the amount of requests without responses for each endpoint and then calculating the mean and standard deviation of those values for all the endpoints. The model is calculated over all the endpoints over all the five-minute batches.

Anomaly values are obtained using a fuzzy function given by

$$f(x) = \begin{cases} 0 & \text{if } x \leq \mu + t_1\sigma \\ \frac{x-(\mu+t_1\sigma)}{(t_2-t_1)\sigma} & \text{if } \mu + t_1\sigma < x < \mu + t_2\sigma \\ 1 & \text{if } x \geq \mu + t_2\sigma \end{cases},$$

where x is the value of the ratio for a given endpoint, μ and σ are the current model values and t_1 and t_2 are thresholds.

Thresholds t_1 and t_2 are important parameters. They are set manually and essentially determine the sensitivity of the detector. Based on our experiments, the thresholds were set to $t_1 = 1$ and $t_2 = 2$, selecting only the samples with probability smaller than 0.12099 and 0.026995 respectively.

The detector is able to detect C&C search for types of malware that perform the search using raw IPs. In our experiments we detect command and control search by Sirefef malware [20]. It is also possible to detect DoS and DDoS type of attacks as well as port scans using this approach.

3.5 Timestamp errors and service detection

In a perfect word, all that we would need to do in order to detect services would be to perform request-response pair matching and then calculate the number of requests and responses for each endpoint. A service would have a lot of responses and no requests, a client would have all requests and no responses. Unfortunately, things are not that simple.

The main problem with using timestamps for labelling requests and responses is that timestamps are not always accurate. Timestamp accuracy depends on the NetFlow probe. If the timestamps were accurate then most of endpoints with few exceptions such as NTP and peer-to-peer would either have only requests or only responses, which is not the case. This means that the accuracy of service detection is highly dependant on the accuracy of timestamp information produced by the NetFlow probe and requires sophisticated modelling in order to correct for timestamp errors.

Unfortunately, the accuracy varies quite a lot between different NetFlow probes and networks. Figure 2 shows the distribution of timestamp differences for two different networks. The timestamp difference is response timestamp minus request timestamp. The figure shows communication of different clients accessing HTTP servers on port 80. The requests and responses are determined using ports and the difference in their timestamps is calculated. The flow pairs whose timestamp difference is zero are not included. The plot on the left shows the results for the CTU network and the plot on the right a different, larger network where the NetFlows are collected by several probes around the network and then merged.

The distribution for the CTU network looks good, with only a small portion of flows having negative difference. The distribution of the other network looks much worse, with the distribution of the differences being almost symmetrical around zero, meaning that the feature values are essentially random and unusable for service detection in this network. This is the main limitation of the service detection model. It is necessary to make sure that the NetFlow probe is capable of producing reliable timestamps before running the model, otherwise accuracy of the results cannot be guaranteed.

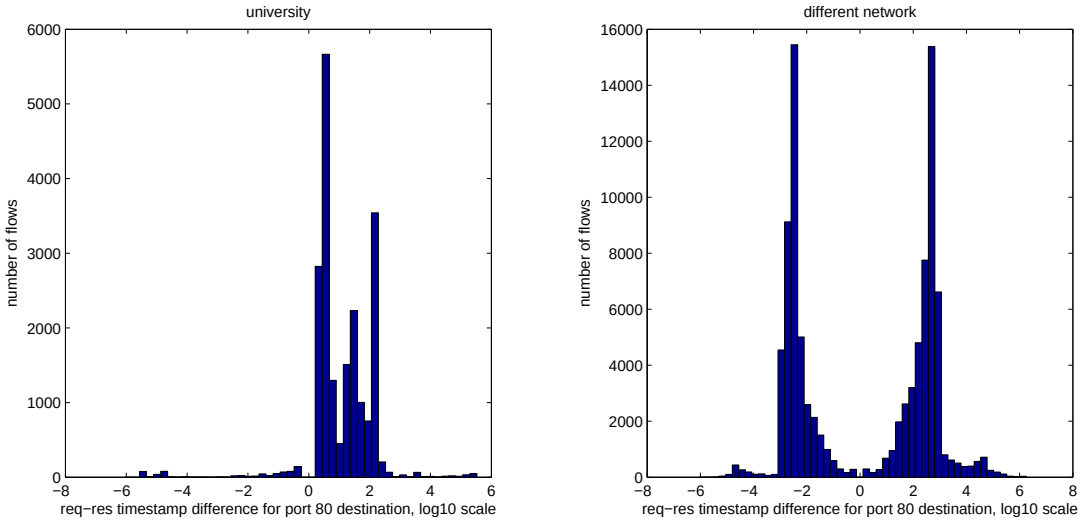


Figure 2 Distribution of res-req times for req-res pairs on two different networks.

3.6 Feature selection

When looking for services, it is important to realise that it is common for an IP address to act both as a client and as a service, e.g. a web server can download updates. We look for specific network services which can be tied to an endpoint — a service running on a specific IP address, using a specific port and protocol, as defined in the beginning of the chapter.

The first feature for service detection is the number of peers that an endpoint communicates with. An active service is likely to have more than one user, which means several endpoints. Even when there is only one user, the number of endpoints that connect to the service is going to be greater, because for most protocols, a random client port is chosen for a new connection, meaning that a single client can act as several endpoints. Active clients create many endpoints which are discarded after every connection resulting in a low number of peers for client endpoints.

The distribution of the number of peers for different endpoints is very wide and heavy-tailed, given by differently used services. This means that setting a threshold for this feature beyond which the endpoint is a service and below which it is a client is very hard. We therefore only use this feature as an initial filter, taking all the endpoints which have more than one peer and ignoring the rest.

The second feature used in service detection is the requests to requests with responses ratio given by:

$$r_{ep} = \frac{\text{requests}_{ep}}{\text{requests}_{ep} + \text{responses}_{ep}},$$

where r_{ep} is the ratio for endpoint ep, requests_{ep} is the number of requests for endpoint ep and responses_{ep} is the number of responses for endpoint ep. The values of that feature can range from 0 to 1. The services are likely to have lower values, while clients higher ones.

Figure 3 shows the distribution of that feature for the CTU network. This figure shows only endpoints that were associated with at least five flows. It is important to note that the labelling of endpoints is done based on their port numbers. Lower ports are said to be services and higher ports are said to be clients. While generally true, this

is not a very reliable indicator as many services run on high ports (such as 8080, 3368) and many protocols use lower ports for the clients (like 123 for NTP). This explains the mixed colours in the graph. Manual inspection of the endpoints with low r_{ep} ratio and high port numbers has shown them to be services running on high ports. Also, a lot of the endpoints with low port numbers and high r_{ep} ratio is clients using low port numbers. The labelling is more of a guidance and cannot be taken as ground truth. Manual inspection is required in order to establish whether an endpoint is a client or a service. This is also the reason that port numbers are not used as a feature in service detection.

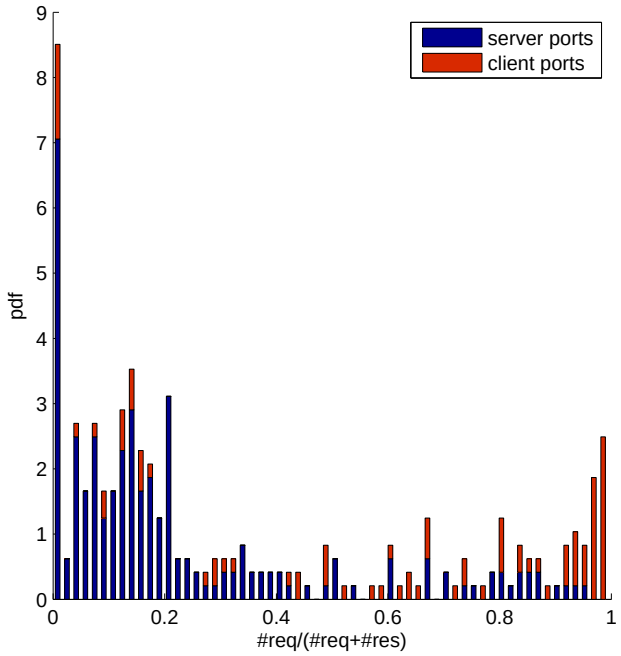


Figure 3 Distribution of r_{ep} for endpoints in CTU network (only endpoints with at least five flows shown)

3.7 Threshold setting using EM algorithm

The last step in service detection model is setting the threshold for r_{ep} that separates services from clients. Given the fact that the timestamp error distributions are different for different networks and also the behaviour of the users of the networks can vary, the distributions of r_{ep} is also different for different features. This makes it necessary to set the treshold online, based on the r_{ep} distribution of that specific network.

We assume that the distribution of r_{ep} comes from two different sources: services and clients. We assume that r distributions for both services and clients are exponential. The distribution r_{pdf} of r_{ep} can be described using the following equation:

$$r_{pdf}(x) = (1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2(x-1)},$$

where x is the ratio value, π is the mixture coefficient, λ_1 is the parameter of the service distribution and λ_2 is the parameter of the client distribution.

We use expectation–maximisation algorithm in order to estimate the parameters of the mixture distribution. Expectation–maximisation algorithm works in two steps. In the expectation step it estimates the probabilities of belonging to a certain distribution for each data point. In the maximisation step it uses maximum likelihood estimation weighted by the calculated probabilities to estimate the parameters of each distribution. The two steps are repeated until the parameters converge to a stable value. A derivation of the expectation–maximisation algorithm is provided below, based on the derivation for a Gaussian mixture model in [21, p. 423].

$$p(x) = \sum_{\mathbf{z}} p(z)p(x|z) = (1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2(x-1)},$$

where z is the latent state and $p(x|z)$ is the conditional distribution of x given state z. The conditional probability of z given by x is defined as follows:

$$\gamma(z_k) \equiv p(z_k = 1|x) = \frac{p(z_k = 1)p(x|z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(x|z_j = 1)}$$

In our case there are two possible latent states: z_1 when the endpoint is service and z_2 when the endpoint is client. Therefore, the conditional probabilities of z are given by:

$$\gamma(z_1) = \frac{(1-\pi)\lambda_1 e^{-\lambda_1 x}}{(1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2(x-1)}}$$

$$\gamma(z_2) = \frac{\pi\lambda_2 e^{\lambda_2(x-1)}}{(1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2(x-1)}}$$

Log-likelihood function is given by:

$$\sum_{i=1}^{N} \ln\{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}\}$$

Maximum-likelihood estimates of distribution parameters are given by:

For λ_1:

$$\frac{\delta}{\delta\lambda_1} \sum_{i=1}^{N} \ln\{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}\} =$$

$$= \sum_{i=1}^{N} \frac{(1-\pi)\left(e^{-\lambda_1 x_i} - x_i\lambda_1 e^{-\lambda_1 x_i}\right)}{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}} =$$

$$= \sum_{i=1}^{N} \frac{(1-\pi)e^{-\lambda_1 x_i}}{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}} - \sum_{i=1}^{N} \frac{(1-\pi)x_i\lambda_1 e^{-\lambda_1 x_i}}{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}} =$$

$$= \sum_{i=1}^{N} \frac{\gamma(z_{i1})}{\lambda_1} - \sum_{i=1}^{N} x_i\gamma(z_{i1}) = 0 \implies \lambda_1 = \frac{\sum_{i=1}^{N}\gamma(z_{i1})}{\sum_{i=1}^{N} x_i\gamma(z_{i1})}$$

For λ_2:

$$\frac{\delta}{\delta\lambda_2} \sum_{i=1}^{N} \ln\{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}\} =$$

$$= \sum_{i=1}^{N} \frac{\pi e^{\lambda_2(x_i-1)} + \pi\lambda_2(x_i-1)e^{\lambda_2(x_i-1)}}{(1-\pi)\lambda_1 e^{-\lambda_1 x_i} + \pi\lambda_2 e^{\lambda_2(x_i-1)}} =$$

$$= \sum_{i=1}^{N} \frac{\gamma(z_{i2})}{\lambda_2} + \sum_{i=1}^{N} x_i \gamma(z_{i2}) - \sum_{i=1}^{N} \gamma(z_{i2}) = 0$$

$$\implies \lambda_2 = \frac{\sum_{i=1}^{N} \gamma(z_{i2})}{\sum_{i=1}^{N} \gamma(z_{i2}) - \sum_{i=1}^{N} x_i \gamma(z_{i2})}$$

Applied to our problem the expectation-maximisation steps are as follows:

Expectation:

$$\gamma(z_1) = \frac{(1-\pi)\lambda_1 e^{-\lambda_1 x}}{(1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2 (x-1)}}$$

$$\gamma(z_2) = \frac{\pi\lambda_2 e^{\lambda_2 (x-1)}}{(1-\pi)\lambda_1 e^{-\lambda_1 x} + \pi\lambda_2 e^{\lambda_2 (x-1)}}$$

Maximisation:

$$\lambda_1^{\text{new}} = \frac{\sum_{i=1}^{N} \gamma(z_{i1})}{\sum_{i=1}^{N} x_i \gamma(z_{i1})}$$

$$\lambda_2^{\text{new}} = \frac{\sum_{i=1}^{N} \gamma(z_{i2})}{\sum_{i=1}^{N} \gamma(z_{i2}) - \sum_{i=1}^{N} x_i \gamma(z_{i2})}$$

$$\pi^{\text{new}} = \frac{\sum_{i=1}^{N} \gamma(z_{i2})}{N}$$

The initial parameters are set to $\lambda_1 = \lambda_2 = 7$, $\pi = 0.5$. In order to avoid the distributions growing out of bounds, the λ_1 and λ_2 parameters are bounded by 10. Figure 4 shows the distribution from Figure 3 fit with the EM algorithm. The threshold is set at the intersection of the two curves.

The expectation–maximisation does not use a convergence criterion. It is run continuously, a new expectation–maximisation cycle run for every NetFlows batch. This is done because we want the model to be able to continuously adapt to the changing network parameters. The behaviour of the endpoints in the network is constantly changing. It is different during the day and at night, during weekday or weekend. We therefore need to always be able find the best threshold parameter, according to the current behaviour of the network. That is why no convergence criterion is used.

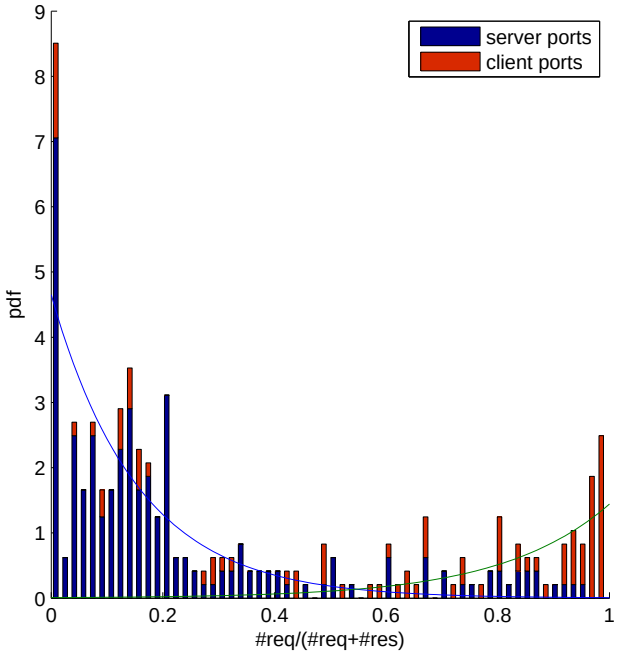
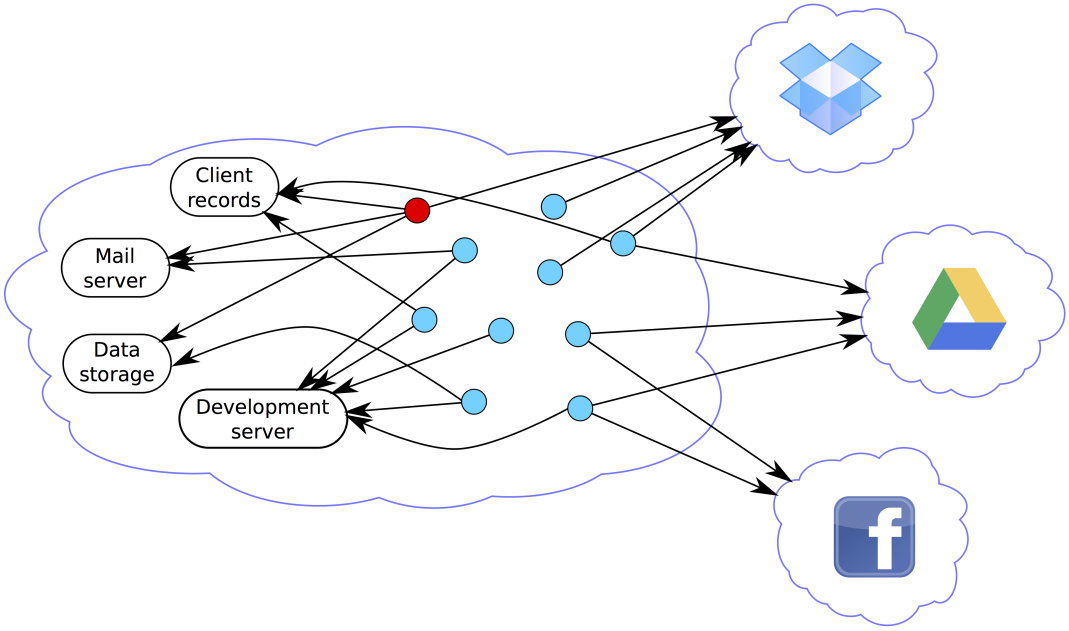


Figure 4 Distribution of r_{ep} for endpoints in CTU network fit with EM algorithm

4 Service Modelling and Anomaly Detection



Often, the real malicious behaviour does not come from botnets or malware. It comes from ordinary users with malicious intent or from somebody who has acquired remote access to a user's machine and is using it for malicious purposes. This can be a corrupt employee or a hacker who has conducted a successful spear phishing [22] attack and managed to install a backdoor inside the computer of one of the regular users inside the network. This user can have access rights to some protected information inside the network. In case of corrupt employee, he may want to download the information and then propagate it somewhere outside the network. In case of a hacker he may need to look for the information first, before he can send it to a safeplace outside the network. The process of finding information is called reconnaissance and the process of propagating it outside the network is called exfiltration.

Another type of service misuse can come from malware. It is incrasingly more common for malware to use cloud services like Google Drive, Twitter, Flickr and others as command and control servers. This is done for several reasons. First reason is the ease of use. There is no need to set up a server, have a publicly available IP address or DNS name. Cloud services are available to anyone, from anywhere. Another reason is increased difficulty of detection. If done right, the command and control channel will look like a normal user using a cloud service to a system administrator. There is no need to perform complex and detectable operations to search for command and control, like DGA [23] or fast flux [24], which can set off alarms. Also, using cloud services is attractive to malware designers, because using these services is free which makes for cheaper malware.

An interesting challenge for an IDS is to learn the behaviour of the users of different services and then report unexpected changes and deviations from the normal behaviour. This type of anomaly detection is increasingly more and more in demand. There are three startup companies that focus on solving this exact challenge: Skyhigh Networks [5], Elastica [6] and Netskope [7]. These companies provide an overview of the services used on the network, detect and report excessive usage by specific users, create risk estimates based on service usage and the qualities of those services and also offer functionality such as cost optimisation for various paid services based on the service usage within the company and the offered contract options.

The focus of this chapter is on service modelling for the purpose of detecting anomalous usage and misuse. In this chapter we look at different ways of modelling users' behaviour on different network services. We also look at ways of reporting anomalies which may often indicate the types of activities such as reconnaissance and exfiltration.

4.1 Framework overview

The service modelling and anomaly detection system consists of several cascading layers. The overview of the layers is illustrated in Figure 5. The first layer is the data source layer. The system currently uses two sources of data: Netflows and Proxy logs both of which are described in detail in Chapter 2. Due to the nature of the features and the models, the system is easily extensible by other sources of data. The second layer extracts the features from the data sources. The features and the extraction process is explained in greater detail in Section 4.2.

This chapter focuses on the third layer. This is the layer where behavioural models of different users are created. Deviations from those models are reported as anomalous and are propagated further to the fourth layer. The fourth layer aggregates the outputs of the behavioural models, across the individual models as well as in time and provides a final system output, which reports anomalous service users. The aggregation is discussed in a different chapter in Section 6.4. The rest of this chapter deals with the service usage models.

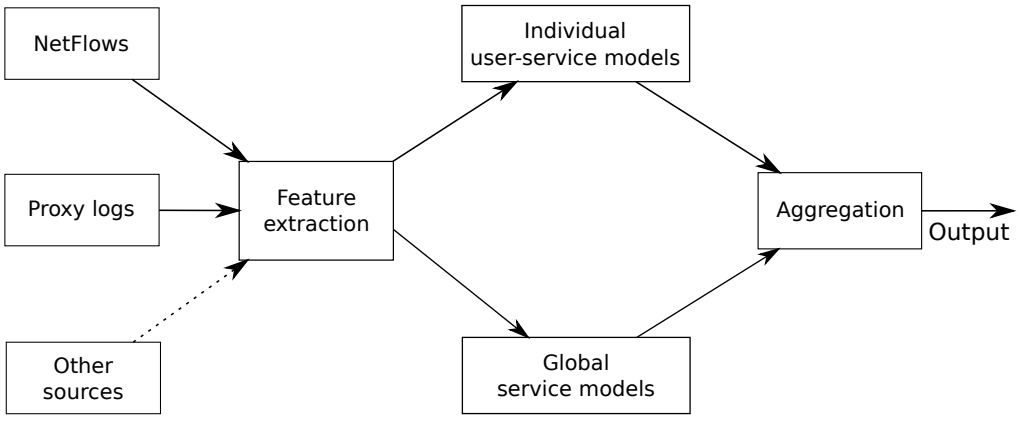


Figure 5 Service modelling framework overview

4.2 Feature extraction

In this layer the features are extracted from the data sources and provided to the models. The features used by the models are very simple, allowing for easy extensibility of the framework by other data sources. The three features that are currently used are: the number of requests, bytes uploaded and bytes downloaded.

The features are provided in the form of time series. A time series is a sequence of data points representing measurements over successive uniformly distributed time periods. The time series are collected for the behaviour of an individual user on an individual service and provided to the model. The model can then make different combinations of the time series or work on separate time series, depending on the type of the model.

The time series can be created with a different time step. The optimal time step depends on many factors, such as model type, service type and activity of users. The smaller the time step, the greater the level of detail provided, but also the greater the noise. A bigger time step filters out a lot of noise, but also filters out the fine details of users' activities.

The three monitored features are calculated by aggregating all of the feature values occurring over all the flows in the specified time frame. As an example, the function for calculating the bytes uploaded feature from proxy logs is as follows:

$$v_{bup}(t, u, s) = \sum_{p \in P_{t,u,s}} p_{bup},$$

where $v_{bup}(t, u, s)$ is the function for calculating the value of time series, t is the time step for which the value is calculated, u is the user, s is the service, $P_{t,u,s}$ is the set of proxy logs with timestamps in the time period $(t, t + t_l)$, t_l being the length of one time step, u being the user and s the service. p_{bup} is the amount of bytes uploaded recorded by proxy log p.

The function for calculating the same feature from NetFlows is slightly different, due to the fact that NetFlows show communication in both directions, while proxy logs only in one. The function for calculating the bytes uploaded from NetFlows is:

$$\nu_{bup}(t, u, s) = \sum_{f \in F_{t,u,s}} f_{tr},$$

where $\nu_{bup}(t, u, s)$ is the function for calculating the value of time series, t is the time step for which the value is calculated, u is the user, s is the service, $F_{t,u,s}$ is the set of flows with timestamps in time period $(t, t + t_l)$, t_l being the length of one time step, u being the user and s the service. The u must correspond to the source endpoint and s to the destination endpoint of the flow. f_{tr} is the amount of bytes transferred by flow f.

The amount of request is calculated by counting the number of corresponding flows in the specified time period. The function for calculating the number of requests from proxy logs is defined as:

$$v_{req}(t, u, s) = |P_{t,u,s}|,$$

with all the variables and parameters being the same as those defined for calculating v_{bup}.

Finally, all the values are logaritmized in order to provide more stable time series. The logaritmized values have the following relationship with the original:

$$v_l = \ln(v + 1),$$

where v_l is the logaritmized value and v is the original. v is in the interval $[0, \infty]$. A zero is added to it to allow for one-to-one mapping to the same interval in the logaritmized time series. The values for other features and sources are calculated correspondingly.

An illustration of timeseries of a number of requests is shown in Figure 6. The time series is of one user of a DNS server, his number of requests monitored over the period of three weeks, with time step of five minutes. This is an active user, and his behaviour clearly shows the different days, which are represented by regular peaks of activity. Also, weekends can be easily distinguished from the work days, where the activity peaks are absent. This is a very active user, unfortunately most users do not exhibit such periodic and predictable behaviour.

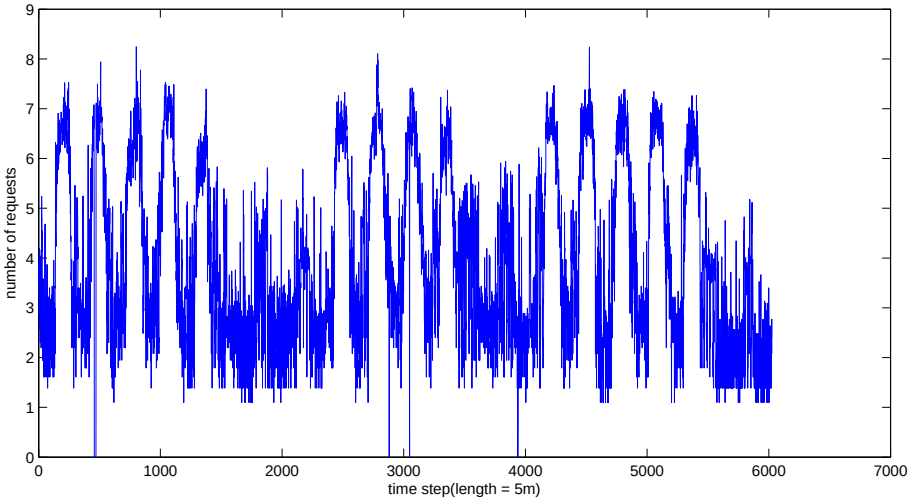


Figure 6 Time series example for a DNS server user, y–axis is on a logarithmic scale

4.3 Individual user–service models

Individual user–service models create a model of a single user using a single service. It is a one-to-one relationship. The idea is that a user will have an established pattern for using a specific service. When the user significantly deviates from his long-term behavioural pattern, it is reported as an anomaly.

The disadvantage of this model is that it only works for users who actively use a service. If the user doesn't use a service, then there is no way to establish a pattern of his interaction with the service and therefore an individual model cannot be applied. The downside is that for many services the majority of users do not use the service consistently. However, for those who do, this type of model can be used.

Below we present different mathematical representations of a user–service individual model. Later on, their performance is compared in Chapter 5.

4.3.1 Holt–Winters prediction model

The Holt–Winters Forecasting algorithm is an algorithm that builds upon normal exponential smoothing. The method is described in [25] and a more sophisticated version

with two seasonal trends is described in [26]. A very thorough analysis of Holt–Winters and other exponential smoothing methods is done in [27].

The Holt–Winters algorithm breaks down the prediction of the time series into three components, each of which is updated via exponential smoothing by a linear combination of the other components and the currently measured value. The Holt–Winters prediction is given by the sum of the three components:

$$\hat{y}_{t+1} = a_t + b_t + c_{t+1-m}.$$

The components are updated using the following equations:
Baseline:

$$a_t = \alpha(y_t - c_{t-m}) + (1-\alpha)(a_{t-1} + b_{t-1}).$$

Linear trend:

$$b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1}).$$

Seasonal trend:

$$c_t = \gamma(y_t - a_t) + (1-\gamma)c_{t-m},$$

where y_t is the measured value at step t, a_t, b_t and c_t are the three components of the prediction, α, β and γ are the exponential smoothing update parameters and m is the length of the season cycle.

The baseline is adjusted by the difference between the seasonal trend and the current value. The linear trend represents the growth of the baseline between observations. The seasonal trend is adjusted based on the difference between the measured value and the baseline.

Szmit[26] presents a slightly more complex Holt–Winters algorithm with two seasonal trends. The algorithm is described by the following equations:

$$\hat{y}_t = L_{t-1} + T_{t-1} + D_{t-r_1} + W_{t-r_2}$$

$$L_t = \alpha(y_t - D_{t-r_1} - W_{t-r_2}) + (1-\alpha)(L_{t-1} + T_{t-1})$$

$$T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1}$$

$$D_t = \gamma(y_t - L_t - W_{t-r_2}) + (1-\gamma)D_{t-r_1}$$

$$W_t = \delta(y_t - L_t - D_{t-r_1}) + (1-\delta)W_{t-r_2}$$

Which is very similar to the one-seasonal model. $\hat{y}_t$ is the predicted value for step t, y_t is the measured value at step t. L_t is the baseline component, T_t is the linear trend and D_t and W_t are the two cycles, their lengths given by r_1 and r_2 respectively.

4.3.2 Autoregressive model

The autoregressive model uses classical autoregression in order to learn and predict the values of a time series. Autoregressive model assumes that a measurement at time t is a linear combination of k previous measurements. In order to find the linear coefficients for the combination we solve a linear squares problem. We use three different types of autoregressive models: without a cycle, with a single cycle and with two cycles.

Autoregressive model without a cycle

The autoregressive model without a cycle is the simplest of the three. It uses a linear combination of the k last time series values in order to predict the next value. The coefficients of the linear combination are found by solving least squares problem.

The value y_t at time t can be represented as:

$$y_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k} + \epsilon_t$$

where k is the number of parameters, y_t is the value at time t, α_p, $p \in 1..k$ are the linear combination coefficients and ϵ_t is the error at time t. The least square problem attempts to minimise the error sum $\sum_{i \in (k,t)} \epsilon_i^2$. It is solved by writing out the linear combinations of the values at different times in matrix format and then finding the inverse as shown below:

$$y = \begin{bmatrix} y_{k+1} \\ y_{k+2} \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} \alpha_1 y_k + ... + \alpha_k y_1 \\ \vdots \\ \alpha_1 y_{n-1} + ... + \alpha_k y_{n-k} \end{bmatrix} =$$

$$= \begin{bmatrix} y_k & y_{k-1} & \cdots & y_1 \\ y_{k+1} & y_k & \cdots & y_2 \\ \vdots & & & \\ y_{n-1} & y_{n-2} & \cdots & y_{n-k} \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_k \end{bmatrix} = Y \cdot A$$

In our specific case of modelling user behaviour we are only interested in time steps where the user was active, therefore we prune the y vector and Y matrix by removing rows of y which are zero and removing the corresponding rows of Y. y_{nz} and Y_{nz} are selected rows of y and Y where all elements of y_{nz} are non-zero. The model parameters are calculated by solving:

$$A = Y_{nz} \backslash y_{nz}$$

The prediction $\hat{y}_t$ at time t is given by:

$$\hat{y}_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k}$$

The advantage of the autoregressive model without a cycle is the small memory length needed to make a prediction. Only k last values are required. The disadvantage is that it does not consider long-term behaviour, which makes it less precise for long-term users, compared to models with a cycle.

Autoregressive model with a cycle

The autoregressive model with a cycle extends the previous model with a cycle that looks back in history and uses the values that correspond to some period in history. This predictor is based on the idea that a lot of users' behaviour is periodic — someone who used a service on Monday is likely to use it again on Tuesday. This model can be represented by the following equation:

$$y_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k} + \beta_1 y_{t-r+\frac{k}{2}-1} + \cdots + \beta_k y_{t-r-\frac{k}{2}} + \epsilon_t$$

where r is the length of the cycle and β_i, $i \in 1..k$ are the parameters for the cycle values. Once again, this is a least squares problem where we want to minimise $\sum_{i \in (k, r+\frac{k}{2})} |\epsilon_i|$.

This is done by constructing the following matrices and solving them in the same way as before.

$$y = \begin{bmatrix} y_{r+\frac{k}{2}+1} \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} y_{r+\frac{k}{2}} & y_{r+\frac{k}{2}-1} & \cdots & y_{r-\frac{k}{2}+1} & y_k & \cdots & y_1 \\ \vdots & & & & & & \\ y_{n-1} & y_{n-2} & \cdots & y_{n-k} & y_{n-r+\frac{k}{2}-1} & \cdots & y_{n-r-\frac{k}{2}} \end{bmatrix} \cdot \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_k \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix} =$$

$$= Y \cdot A$$

Once again, we select rows of y and Y which are non-zero for y and use

$$A = Y_{nz} \backslash y_{nz}$$

as parameter estimate. The prediction is then given by:

$$\hat{y}_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k} + \beta_1 y_{t-r+\frac{k}{2}-1} + \cdots + \beta_k y_{t-r-\frac{k}{2}}$$

The advantage of using a cycle is the ability to look back further in history and be able to learn from long-term usage. The disadvantage is that more values need to be retained in memory, increasing hardware demands.

Autoregressive model with two cycles

Autoregressive model with two cycles is done by extending the autoregressive model with one cycle with another cycle of greater length r_2 and then solving the problem in the same way. The two cycle model can be an improvement on one cycle if we consider that user's behaviour can have two periods: daily and weekly. A common pattern is for activity to repeat from day to day and then stop over the weekend. This can be accounted for by using the second cycle. The prediction of the two-cycle autoregressive model is then given by:

$$\hat{y}_t = \alpha_1 y_{t-1} + \cdots + \alpha_k y_{t-k} + \beta_1 y_{t-r_1+\frac{k}{2}-1} + \cdots + \beta_k y_{t-r_1-\frac{k}{2}} + \gamma_1 y_{t-r_2+\frac{k}{2}-1} + \cdots + \gamma_k y_{t-r_2-\frac{k}{2}},$$

where r_1 is the length of the first cycle and r_2 is the length of the second cycle and γ_i, $i \in 1..k$ are the parameters for the second cycle values.

The advantage is increased ability to learn from long-term history. The disadvantage is increase in memory demands.

Autoregressive model with aggregated memory

One of the problems of predictive models with cycles is that they need to retain a lot of values in memory (longest cycle length) in order to make a prediction. A way around it is to aggregate values that are far in memory into values with a greater time step. This way less memory is needed to store the time series and similar performance results can be achieved by the predictor. A description of an autoregressive predictor with one cycle with aggregated memory is given below:

$$y_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k} + \beta_1 \tilde{y}_{\tilde{f}(t-r)+\frac{l}{2}-1} + \cdots + \beta_k \tilde{y}_{\tilde{f}(t-r)-\frac{l}{2}} + \epsilon_t,$$

where r is the length of the cycle and β_i, $i \in 1..k$ are the parameters for the cycle values, $\tilde{y}_{\tilde{t}}$ is the aggregate of values for time $\tilde{t}$ defined as

$$\tilde{y}_{\tilde{t}} = \frac{1}{w} \sum_{i=0}^{w-1} y_{f(\tilde{t})+i},$$

where w is the width of aggregation for past values. The corresponding time transformations defined as follows:

$$t = f(\tilde{t}) = w \cdot (\tilde{t} - 1) + 1$$

$$\tilde{t} = \tilde{f}(t) = \lfloor \frac{t}{w} \rfloor$$

The matrices for the least square problem are formulated similarly to the autoregressive model with one cycle and are solved in the same way. The prediction value is given by:

$$\hat{y}_t = \alpha_1 y_{t-1} + ... + \alpha_k y_{t-k} + \beta_1 \tilde{y}_{\tilde{f}(t-r)+\frac{l}{2}-1} + \cdots + \beta_k \tilde{y}_{\tilde{f}(t-r)-\frac{l}{2}}$$

This model can be easily extended to a model with two cycles.

4.3.3 Quantile Regression Model

Quantile regression model is similar to normal autoregressive model. The difference is in the minimization function. Quantile regression is described more thoroughly in [28, p. 18]. Quantile regression is achieved through solving the following minimisation problem:

$$\arg\min_{\beta} \sum_{i=1}^{n} \rho_\tau(y_i - \xi(x_i, \beta))$$

$$\rho_\tau(x) = \begin{cases} \tau * x & \text{if } x \geq 0 \\ (\tau - 1) * x & \text{if } x < 0 \end{cases}$$

where τ is the quantile desired, ξ is the projection function, x_i is the input variable, y_i is the output variable, β is the set of parameters for ξ to be optimised and i is the step.

Quantile regression works by minimising an absolute error function. Quantile regression can fit a function into different quantiles of the y distribution. This is achieved by giving different weight to errors of different sign through parameter τ, allowing to set the quantile into which the function is to be fit. The parameters β that minimise the error function are found by solving a linear programming problem.

Quantile regression model without a cycle

A predictor without a cycle similar to the one in autoregressive model can be created if we take ξ to be a linear combination of the k values preceding y_i. The quantile regression model is then formulated as follows:

$$\arg\min_{\beta \in \Re^k} \sum_{t=k+1}^{n} \rho_\tau(y_t - \xi(y_{t-1}, \cdots, y_{t-k}, \beta))$$

$$\xi(y_{t-1}, \cdots, y_{t-k}, \beta) = \beta_1 y_{t-1} + \cdots + \beta_k y_{t-k},$$

where y_t is the value at time t, β is the set model parameters and k is the number of model parameters. The prediction is given by

$$\hat{y}_t = \beta_1 y_{t-1} + \cdots + \beta_k y_{t-k}$$

This model is somewhat similar to the autoregressive model without a cycle. The advantage is the ability to choose the quantile into which to fit the function, which in turn could allow to control the amount of alerts generated by the model. The disadvantage is very high computational complexity, compared to other models.

Quantile regression model with one cycle

A quantile regression model with one cycle can be created analogously to the autoregressive model. The quantile regression model with one cycle is formulated as follows:

$$\arg\min_{\beta \in \Re^k} \sum_{t=r+k/4+1}^{n} \rho_\tau(y_t - \xi(y_{t-1}, \cdots, y_{t-k/2}, y_{t-r-k/4}, \cdots, y_{t-r+k/4-1}, \beta))$$

$$\xi(y_{t-1}, \cdots, y_{t-k}, \beta) = \beta_1 y_{t-1} + \cdots + \beta_{k/2} y_{t-k/2} + \beta_{k/2+1} y_{t-r-k/4} + \cdots + \beta_k y_{t-r+k/4-1},$$

where r is the length of the cycle. The model prediction is given by:

$$\hat{y}_t = \beta_1 y_{t-1} + \cdots + \beta_{k/2} y_{t-k/2} + \beta_{k/2+1} y_{t-r-k/4} + \cdots + \beta_k y_{t-r+k/4-1}$$

Quantile regression model with two cycles

The quantile regression model with two cycles is created by adding a second cycle to the single cycle model. Its output is given in an analogous manner.

Quantile regression model - linear programming

The task of learning the parameters of the quantile regression model is non-trivial. We can reformulate the problem by creating an additional optimisation variable $t \in \Re^l$. The optimisation problem is then:

$$\min \sum_{i=1}^{l} t_i$$

$$\begin{bmatrix} -\tau Y & -I \\ -(\tau - 1)Y & -I \end{bmatrix} \cdot \begin{bmatrix} P \\ t \end{bmatrix} \leq \begin{bmatrix} -\tau y \\ -(\tau - 1)y \end{bmatrix},$$

where Y and y are the matrix and vector defined in Section 4.3.2. $Y \in \mathbb{R}^{l \times k}$ and P are the model parameters to be found. This is a form that can be solved by standard linear programming solvers.

4.3.4 Anomaly values from predictor models

Since the predictor models described above give only an estimate of what the time series value should be at a specific time step, we need a way to calculate anomaly values from these predictions. In order to calculate the anomaly values we first calculate the deviations of the measured value y_t from the predicted value $\hat{y}_t$ at time step t. As we are only interested in excessive usage, we only take deviations that are greater than $\hat{y}_t$. Deviation for time step t is given by:

$$\delta_t = \begin{cases} 0 & \text{if } y_t - \hat{y}_t \leq 0 \\ y_t - \hat{y}_t & \text{if } y_t - \hat{y}_t > 0 \end{cases}$$

Anomaly value is then given as a CDF of the exponential distribution of non zero deviations δ. The distribution parameter is calculated as follows:

$$\lambda = \frac{1}{|\Delta_p|} \sum_{\delta \in \Delta_p} \delta,$$

where Δ_p is the set of positive δ_t. The anomaly value for measurement y_t for time step t is then given by:

$$\text{anomaly } (y_t) = \begin{cases} 0 & \text{if } y_t - \hat{y}_t \leq 0 \\ 1 - \exp\{\lambda(y_t - \hat{y}_t)\} & \text{if } y_t - \hat{y}_t > 0 \end{cases}$$

4.3.5 Parzen window cumulative distribution model

The Parzen window cumulative distribution model is different from the previous models in that it does not take the time order into account. The idea is simple: in order to detect excessive usage we report usage that is higher than what was seen previously, regardless of when it had happened before.

The Parzen window cumulative distribution model uses the Parzen window method to estimate the probability density function of the values from the time series and then uses it to numerically calculate the cumulative distribution function. The Parzen window estimation of the probability density is defined below, as given in [21, p. 123]

$$p(x) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h} k\left(\frac{x - x_n}{h}\right),$$

where x_n, $n \in 1..N$ are the samples from which the probability density function is estimated, h is the bandwidth and k is the Parzen window function, sometimes referred to as kernel function. The Parzen window (kernel) function is subject to the following two conditions:

$$k(u) \geq 0$$

$$\int_{-\infty}^{\infty} k(u) = 1$$

The cumulative density function can be estimated numerically using

$$cdf(x) = \sum_{i=1}^{m} b \cdot \frac{p(x_{i+1}) - p(x_i)}{2},$$

where b is the length of the integration step, $p(x)$ is the Parzen window estimation function and x_i, $i \in 1..m$ are samples with distance b between two consecutive sample, sample $x_m = x$ and $p(x_1) < \epsilon$, where ϵ is a tolerance value.

Parzen window cumulative distribution model uses the time series as the samples for estimating the probability density function and gives the numerically estimated cumulative density function $cdf(x)$ as its output. Figure 7 show an example of a Parzen window cumulative distribution model for a single user. Anomaly value is given by the cdf.

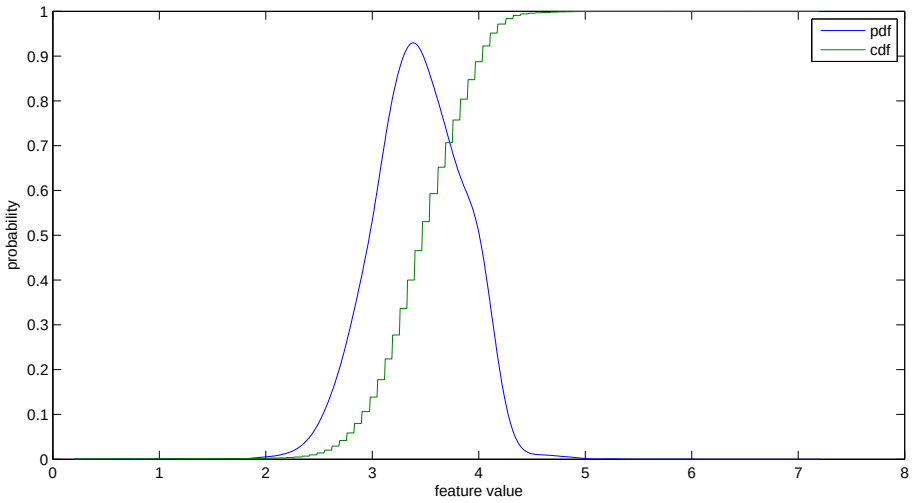


Figure 7 Parzen window cumulative distribution model

4.4 Global service models

A global service model considers the behaviour of all the users of the modelled service. It then reports those users who have different behaviour from the majority as anomalous. Its main advantage over the individual user–service models is that this type of model does not require user history, it works just as well for the users who use the service sporadically or even those who use it for the first time ever.

4.4.1 Global median model

The global median model works by taking the values for all the active users in a specific time step. It then calculates the median of those values.

$$m = \text{median } v,\ v \in V_{at},$$

where v is the value of the modelled feature for a specific user and V_{at} is the set of values of this feature for all users active at time t. The next step is finding deviations from the median. As we are only interested in excessive usage, we only take deviations that are greater than m. Deviation for user u is given by:

$$\delta_u = \begin{cases} 0 & \text{if } v_u - m \leq 0 \\ v_u - m & \text{if } v_u - m > 0 \end{cases},$$

Anomaly value is given as a CDF of the exponential distribution of non zero deviations δ. The distribution parameter is calculated as follows:

$$\lambda = \frac{1}{|\Delta_p|} \sum_{\delta \in \Delta_p} \delta,$$

where Δ_p is the set of positive δ_u. The anomaly value for measurement v_u for user u is then given by:

$$\text{anomaly } (v_u) = \begin{cases} 0 & \text{if } v_u - m \leq 0 \\ 1 - \exp\{\lambda(m - v_u)\} & \text{if } v_u - m > 0 \end{cases}$$

4.4.2 Global Parzen window cumulative distribution model

The global parzen window model works on the same principle as the individual user–service Parzen window cumulative model described in Section 4.3.5. The difference is that instead of taking the distribution of values of an individual user over time, the global model takes the distribution of values of all active users in current time step. The rest is exactly the same. The anomalies are given by the numerically calculated cumulative density function over the active user values distribution.

5 Experiments

In this chapter we present the results of experimenting with the mathematical models and algorithms described in the previous chapters. First, in Section 5.1 we compare the performance of the simple and parallel request-response pair matching algorithms, as were described in Sections 3.2 and 3.3. Then, in Section 5.2 the experiments and detection results for the request-response detector (see Section 3.4) are described. In Section 5.3 we evaluate the service detection algorithm as described in Sections 3.5, 3.6 and 3.7. Section 5.4 does a comparison of the service models presented in Chapter 4. Subsection 5.4.1 does an error comparison for different predictors on multiple datasets. Finally, Subsection 5.4.2 evaluates the performance of detectors on mixed attacks.

5.1 Request-response pair matching

The performance of the two versions of request-response pair matching algorithms, described in Sections 3.2 and 3.3 was compared. The performance was compared on several different NetFlow batches, each containing a different amount of NetFlows. The NetFlow batches of different sizes were generated from an existing dataset by merging a required amount of NetFlows into a single batch. Five batches were created altogether, each successive batch containing roughly twice the amount of NetFlows as the previous batch.

Each algorithm was run on the NetFlows batch twenty times and the time was measured. The average time was given as the run time for each algorithm. Parallel algorithm was allowed to use four threads. Both algorithms were run on the same machine with a dual-core CPU. Figure 8 shows the comparisons of performance of the two algorithms. Parallel version is significantly faster than the normal version for all batch sizes.

5.2 Request-response anomaly detector

The request-response anomaly detector was run on several different datasets captured on the CTU network. Different types of malicious and legitimate activities were labelled in the datasets. AUC values were calculated from the outputs of the detectors using the dataset labels as ground truth.

Table 2 shows the AUC values for different datasets that contain different types of malicious behaviour: malware C&C search, port scans and DoS attacks. The AUC values show high detection quality for all three types of malicious behaviour. The scan detection quality depends on how successful and extensive the scan is.

Successful scans get lower AUC values, because the request-response detector only reports flows that are requests without responses as anomalous. In the dataset, all the flows belonging to the scan are labelled as anomalous, including those that got a response. This means, that even though request-response is successful at detecting a scan, it will only label a subset of the flows belonging to the scan as anomalous, which results in a lower AUC value.

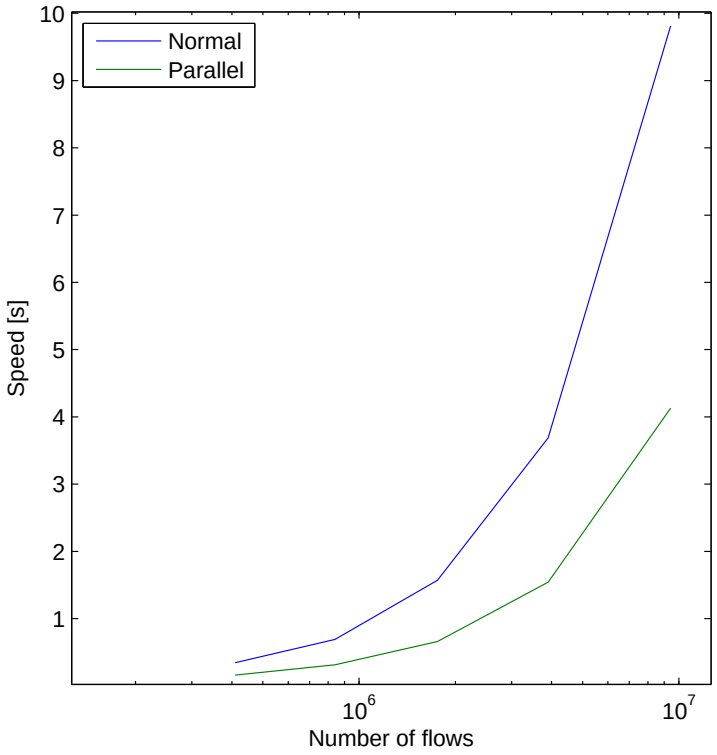


Figure 8 Request-response pair matching algorithm comparison

Sirefef and Pushdo are different malware families that perform C&C search by going to lot of different IP addresses. Most of those addresses are not active, which results in a large number of requests without responses generated by the malware and aids in its detection.

Horizontal scan is a scan of certain port across many different IP addresses in the network. This type of scan is more likely to be successful, as the ports chosen are usually of commonly used services that the intruder wants to discover and which are more likely to respond to a request. SSH scan and SQL scan are examples of horizontal scan for a specific service type.

A vertical scan is a scan of a single IP address across a large set of ports, in order to discover what services are run on that IP address. Normally, this type of scan is less likely to be successful as there is a very large number of ports, while it is unusual to run more than a couple of services on a single IP address, therefore only a small portion of requests is likely to get a response.

ICMP and SYN flood are types of DoS attacks, where the attacker sends a huge amount ICMP or SYN packets to a certain service. The ideas is to consume enough server resources by these packets to make it unresponsive. This means that a successful ICMP or SYN flood attack will also have a large number of requests without responses.

From the experiments we can conclude that the detector is successful at detecting C&C search by two different families of malware, horizontal and vertical port scans, as well as denial of service attacks. High AUC values show high detection quality.

Dataset	Label	AUC value
Malware C&C search		
ctu_sirefef_november(15_11)	Sirefef	0.9606
ctu_sirefef_november(19_11)	Sirefef	0.9648
ctu_pushdo_november	Pushdo	0.8481
Port scans		
ctu-dataWithout0-firstDay	SQL scan	0.8873
ctu-dataWithout0-firstDay	Horizontal scan	0.8541
ctu-dataWithout0-firstDay	Vertical scan	0.9893
ctu-sshScan-bruteforce_large	SSH scan	0.8996
ctu-sshScan-bruteforce_small	SSH scan	0.8959
ctu_scanMsssql	Vertical scan	0.9055
felk-probing _attack	Vertical scan	0.9078
DoS		
ctu_icmp_attack_november	ICMP attack	0.9534
ctu_synFlood_attack_november	SYN flood	0.9533
ctu_synFlood_attack_november	ICMP attack	0.9196

Table 2 Results of request response anomaly detector. AUC values are shown for datasets that contain different types of malicious behaviour.

5.3 Service detection

Evaluating the results of service detection is somewhat difficult due to the lack of ground truth labels for services. The only way to confirm if a detected service is a service is by manual inspection of NetFlows or by obtaining additional information, e.g. asking the administrator of the network or contacting the service. This has its own problems, as the administrator might not know about the service and contacting it may not work, as a service that was active at the time when NetFlows were captured may not be active when you attempt to contact it.

In our evaluation of service detection we chose a simple if somewhat manual approach. We did an analysis of common network services such as DNS and HTTP. As the first step we found all the endpoints inside the network that seemed to be running one of these common services. This was done by iterating over all the flows in the dataset and taking those flows whose source port was a port of one of the common services and whose TCP flags were not R or AR in order to filter out scans. These flows represented responses from common services, and their source endpoints were taken as potentially running one of the common services in the network. Service detection algorithm was then run on the same dataset and its output was compared to the estimate of the services using port numbers. Table 3 shows the results of the evaluation. The service type and port number columns represent the information about the service types, the detected services column is the number of services detected by the service detector and all endpoints column is the number of endpoints counted using that port. The last column represents the ratio between the number of detected services and all endpoints using that port.

From looking at the Table 3, it seems that the service detection quality is not great. However, if we list all the endpoints for a specific service type and go through them manually the picture is different: there are five endpoints that are confirmed DNS servers, all of which are detected by the service detector. The other one detected is also

Table 3 Service detection results

Service type	Port number	Detected services	All endpoints	Detected/all
HTTPS	443	30	218	0.14
RDP	3389	28	90	0.31
HTTP	80	62	232	0.27
DNS	53	6	12	0.5
SSH	22	109	315	0.35
HTTP	8080	9	48	0.19
SMTP	25	4	23	0.17
MySQL	3306	18	72	0.25

a service, however it is a form of service misuse through DNS tunnelling. The rest of the endpoints are clients who use DNS port as their source port and two of which have only one flow in the whole dataset. This shows that the detector has good recall and precision. Manual inspection for other services reveals similar resuts. The clients who use DNS port as the source port is an example of why using the port numbers is a bad feature in detecting services, as done by [17] and [18] and it is better to model services using the behavioural approach.

5.4 Service modelling

In this section we compare different predictors and anomaly models from Chapter 4.

5.4.1 Predictor errors

In this section we compare the errors of the user-service individual model predictors. We define two types of errors for the purposes of this comparison. The first type is normalised root mean squared error for non-zero values, defined as:

$$\text{rmse} = \sqrt{\frac{1}{|T_{nz}|} \sum_{t \in T_{nz}} \left(\frac{y_t - \hat{y}_t}{y_t} \right)^2},$$

where y_t is the measurement at time t, $\hat{y}_t$ is the predicted value for time t and T_{nz} is the set of times for which y_t is non-zero. We take only non-zero values of y_t, because in the service modelling anomaly detection a zero value of y_t means inactivity of the user at time t. As we are interested in detecting excessive usage, we disregard those periods when the user was inactive. The normalisation is done in order to make the influence of less intensive and more intensive users equal in the mean predictor error.

The second error is the mean normalised heaviside error and it is defined as follows:

$$\text{mhe} = \frac{\sum_{t \in T_{nz}} \left(\frac{y_t - \hat{y}_t}{y_t} \right) \cdot \text{H}\left(y_t - \hat{y}_t\right)}{\sum_{t \in T_{nz}} \text{H}\left(y_t - \hat{y}_t\right)},$$

where $\text{H}(x)$ is an indicator function defined as:

$$\text{H}(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

The mean normalised heaviside error captures how much a predictor undershoots the measured values on average. This is an important error measure when detecting

Table 4 Predictor errors for Dropbox, different time aggregation windows, active users

	rmse	**mhe**	**sum**	**rmse**	**mhe**	**sum**
	Dropbox (5m)			Dropbox (30m)		
Holt-winters	**0.057**	0.042	**0.099**	**0.101**	0.053	**0.153**
AR no mem	0.083	0.053	0.135	0.138	0.045	0.184
AR day mem	0.081	0.051	0.133	0.126	0.050	0.176
AR day week mem	0.079	0.050	0.129	0.116	0.048	0.163
AR agg day mem	0.082	0.052	0.133	0.131	0.050	0.182
AR agg day week mem	0.080	0.050	0.130	0.124	0.048	0.171
Quan day 0.5	0.103	0.034	0.137	0.139	0.041	0.180
Quan day 0.75	0.102	0.065	0.167	0.138	0.065	0.203
Quan day week 0.5	0.102	**0.033**	0.135	0.134	**0.039**	0.173
Quan day week 0.75	0.101	0.064	0.164	0.134	0.060	0.193
	Dropbox (1h)			Dropbox (2h)		
Holt-winters	0.153	0.114	0.267	0.213	0.185	0.398
AR no mem	0.200	0.092	0.291	0.238	0.140	0.378
AR day mem	0.177	0.092	0.269	0.225	0.130	0.354
AR day week mem	**0.153**	0.073	**0.226**	0.179	0.093	0.271
AR agg day mem	0.184	0.101	0.285	0.178	0.110	0.287
AR agg day week mem	0.168	0.080	0.248	**0.166**	**0.072**	**0.237**
Quan day 0.5	0.194	0.074	0.268	0.245	0.107	0.352
Quan day 0.75	0.207	0.108	0.315	0.269	0.156	0.425
Quan day week 0.5	0.180	**0.062**	0.242	0.206	0.079	0.286
Quan day week 0.75	0.194	0.083	0.278	0.255	0.094	0.349

excessive usage, because it reflects how much a predictor is able to distinguish natural increase in usage from anomalous excessive usage.

The two errors measure different criteria of predictor quality. If we choose a predictor based on one error, it is possible that it will perform badly in the other. It is therefore necessary to create a measure that will reflect both aspects of predictor quality. Root mean square error and mean heaviside error measure error in the same units so we can take their sum and then take that as the unified measure that reflects both criteria.

Evaluation was performed by running evaluated predictor on a set of users in a dataset, measuring their error, then taking the average across the users and reporting it as the error for the predictor. The users on which the predictor was evaluated were chosen based on their activity level using the following ratio:

$$\mathrm{ar} = \frac{|T_{nz}|}{|T|},$$

where T_{nz} is the set of time steps where the user was active and T is the set of time steps in which user's activity was measured. Active users were defined as those whose ar is greater than 0.25 and highly active users as those whose ar is greater than 0.5.

Experiments were run for different predictors with different configurations. Holt-winters predictor was run using the two-cycle version, with parameters set to

$$\alpha = f(0.75, 3)$$

$$\beta = f(0.2, 288 \cdot 2/a)$$

Table 5 Predictor errors for Dropbox, different time aggregation windows, highly active users

	rmse	**mhe**	**sum**	**rmse**	**mhe**	**sum**
	Dropbox (5m)			Dropbox (30m)		
Holt-winters	**0.368**	0.201	0.569	0.185	0.133	0.318
AR no mem	0.445	0.253	0.698	0.187	0.088	0.275
AR day mem	0.422	0.241	0.662	0.165	0.079	0.244
AR day week mem	0.375	0.199	0.574	**0.103**	0.054	**0.156**
AR agg day mem	0.422	0.229	0.651	0.156	0.075	0.231
AR agg day week mem	0.403	0.177	0.580	0.124	0.059	0.183
Quan day 0.5	0.420	0.228	0.648	0.179	0.077	0.256
Quan day 0.75	0.574	0.239	0.813	0.217	0.083	0.300
Quan day week 0.5	0.381	0.183	**0.565**	0.120	0.041	0.161
Quan day week 0.75	0.535	**0.174**	0.709	0.187	**0.038**	0.225
	Dropbox (1h)			Dropbox (2h)		
Holt-winters	**0.082**	0.049	0.132	0.124	0.081	0.205
AR no mem	0.099	0.031	0.130	0.128	0.079	0.207
AR day mem	0.093	0.042	0.135	0.121	0.073	0.194
AR day week mem	0.084	0.040	0.124	0.108	0.054	0.162
AR agg day mem	0.096	0.042	0.138	0.106	0.084	0.191
AR agg day week mem	0.087	0.037	0.124	**0.101**	0.047	**0.149**
Quan day 0.5	0.101	0.022	0.123	0.134	0.041	0.174
Quan day 0.75	0.102	0.040	0.142	0.138	0.068	0.206
Quan day week 0.5	0.097	**0.021**	**0.118**	0.122	**0.035**	0.157
Quan day week 0.75	0.098	0.038	0.136	0.142	0.052	0.194

$$\gamma = f(0.5, 288/a)$$
$$\delta = f(0.5, 288 \cdot 7/a)$$
$$f(w, p) = 1 - \exp\left\{\frac{\log(1-w)}{p}\right\},$$

f is a function that sets the parameters in such a way so that the last p time points account for $w \cdot 100\%$ of the prediction. $w \in (0, 1)$, $p \in \{1, 2, 3, ...\}$. a is the number of five-minutes aggregated into one time step. The procedure for setting Holt-winter model parameters was taken from [25].

Autoregressive model parameters were set to $k = 10$. For autoregressive model with aggregated memory $l = 2$ and $w = 12$ for the two-cycle version $l_1 = 2$, $w_2 = 12$, $l_2 = 2$ and $w_2 = 36$. *Mosek* [29] linear programming solver was used to solve the LP problem for the quantile regression model. The cycles for all the models, where appropriate, were set to $r_1 = 288/a$ and $r_2 = 288 \cdot 7/a$, which is a one day and one week cycles, respectively.

Experiments were run for several different datasets with different network services and different aggregation steps and the results were compared. Tables 4, 5, 6 and 7 show the results of the evaluation.

Table 4 shows the results of evaluation run with different time step length on the same service Dropbox. Dropbox was chosen because it is a popular data storage cloud service and can be easily used for data exfiltration. It is also used by a lot of users in the dataset which allows for more reliable evaluation. For the shorter time step the

Table 6 Predictor errors for various services, 2 hour aggregation, active users

	rmse	**mhe**	**sum**	**rmse**	**mhe**	**sum**
		Twitter			Salesforce	
Holt-winters	**0.399**	0.329	0.728	0.302	0.249	0.551
AR no mem	0.563	0.351	0.913	0.430	0.266	0.696
AR day mem	0.531	0.318	0.848	0.403	0.246	0.649
AR day week mem	0.460	0.249	0.709	0.307	0.223	0.530
AR agg day mem	0.471	0.248	0.718	0.308	0.172	0.480
AR agg day week mem	0.457	0.198	**0.656**	**0.284**	0.164	**0.447**
Quan day 0.5	0.551	0.308	0.858	0.424	0.233	0.658
Quan day 0.75	0.773	0.311	1.084	0.533	0.266	0.799
Quan day week 0.5	0.503	0.234	0.736	0.304	0.223	0.527
Quan day week 0.75	0.751	**0.192**	0.943	0.512	**0.130**	0.642
		Live			Dropbox (Smurf network)	
Holt-winters	**0.403**	0.256	0.660	0.262	0.216	0.478
AR no mem	0.504	0.301	0.805	0.296	0.175	0.472
AR day mem	0.470	0.281	0.751	0.260	0.153	0.414
AR day week mem	0.408	0.231	0.638	**0.114**	0.072	0.186
AR agg day mem	0.456	0.230	0.687	0.242	0.133	0.375
AR agg day week mem	0.420	**0.182**	**0.602**	0.117	0.058	**0.175**
Quan day 0.5	0.479	0.269	0.748	0.286	0.141	0.427
Quan day 0.75	0.665	0.275	0.940	0.352	0.148	0.501
Quan day week 0.5	0.419	0.217	0.635	0.130	0.082	0.212
Quan day week 0.75	0.617	0.193	0.810	0.188	**0.055**	0.243

smallest error is attained by Holt-winters predictor, while for the greater time steps the error is lowest for the autoregressive model with day week cycles. Table 5 shows the results of the same evaluation for highly active users. Here the results are always the best for autoregressive models with aggregated memory and day week cycles.

Tables 6 and 7 show the results of the same evaluation run for different services with two hours time step length. For the active users the most successful predictor is the autoregressive with aggregated memory and day week cycle. For the highly active users the most successful is Holt–Winters for Twitter and Salesforce and Live and Dropbox it is quantile and autoregressive predictors with day week cycles respectively.

Autoregressive predictors with day week cycles and autoregressive predictor with aggregated memory and day week cycle often have the lowest error values. In those cases where their values are not the lowest, they are among the top three lowest values. From this we can conclude that from the point of view of prediction error the autoregressive predictors with day week cycles and autoregressive predictor with aggregated memory and day week cycles are the best ones.

5.4.2 Mixed attacks

In this section we describe mixing of attacks into normal user traffic in order to test performance of the detectors. The mixing process is relatively simple, due to the additive nature of the features used. An attack is said to be an excessive usage of a service during some period of time, for example a one gigabyte upload to Dropbox during an

Table 7 Predictor errors for various services, 2 hour aggregation, highly active users

	rmse	**mhe**	**sum**	**rmse**	**mhe**	**sum**
	Twitter			Salesforce		
Holt-winters	**0.277**	0.149	**0.427**	**0.221**	0.184	**0.405**
AR no mem	0.371	0.154	0.525	0.289	0.217	0.506
AR day mem	0.355	0.153	0.508	0.261	0.203	0.464
AR day week mem	0.323	0.135	0.458	0.254	0.199	0.452
AR agg day mem	0.334	0.146	0.480	0.231	0.192	0.423
AR agg day week mem	0.335	0.127	0.463	0.236	0.185	0.420
Quan day 0.5	0.358	0.154	0.512	0.247	0.200	0.448
Quan day 0.75	0.446	0.165	0.611	0.363	0.176	0.539
Quan day week 0.5	0.339	**0.122**	0.461	0.243	0.198	0.441
Quan day week 0.75	0.421	0.132	0.554	0.360	**0.158**	0.519
	Live			Dropbox (Smurf network)		
Holt-winters	**0.368**	0.201	0.569	0.185	0.133	0.318
AR no mem	0.445	0.253	0.698	0.187	0.088	0.275
AR day mem	0.422	0.241	0.662	0.165	0.079	0.244
AR day week mem	0.375	0.199	0.574	**0.103**	0.054	**0.156**
AR agg day mem	0.422	0.229	0.651	0.156	0.075	0.231
AR agg day week mem	0.403	0.177	0.580	0.124	0.059	0.183
Quan day 0.5	0.420	0.228	0.648	0.179	0.077	0.256
Quan day 0.75	0.574	0.239	0.813	0.217	0.083	0.300
Quan day week 0.5	0.381	0.183	**0.565**	0.120	0.041	0.161
Quan day week 0.75	0.535	**0.174**	0.709	0.187	**0.038**	0.225

Table 8 Twitter mixed attack, mean rank out of 46 users

Attack size (#req/h)	**AR day week mem**	**Global median**
5	9.83	10.41
50	5.044	7.59
500	2	2.96
5000	1.28	2.85
50000	1.18	2.02

hour. Any usage can only be described as excessive within some context, so a small upload to one service can be absolutely normal, while the exact same upload to a different service, or even to the same service by a different user can be considered excessive.

The evaluation is done using the following procedure. For the evaluated service, we select a group of users that we want to evaluate the detector on. We usually choose active or highly active users, as the detectors based on individual user-service models are only effective for them. We mix the attack into each user one by one. The time series of the user with the mixed attack are given by

$$\text{mixed time series} = \{v_1, v_2, ..., v_{start} + a_1, ..., v_{end} + a_m, ..., v_n\},$$

where v_k, $k \in 1..n$ are the non-logarithmic values of the user time series and a_k, $k \in 1..m$ are the non-logarithmic values of the attack time series. *start* is the first time step into

Table 9 DNS enumeration attack, mean rank out of 548 users, detector comparison, attack at night

Detector	Full[rank]	Full[prob]	Half[rank]	Half[Prob]
Holt-winters	16.98	0.97	26.00	0.95
AR no mem	6.85	0.97	8.33	0.96
AR day mem	5.94	0.98	7.19	0.97
AR day week mem	15.07	0.96	13.75	0.95
AR agg day mem	6.77	0.98	8.93	0.97
AR agg day week mem	13.86	0.95	15.17	0.95
Quan day 0.5	15.46	0.96	15.10	0.95
Quan day 0.75	19.41	0.96	19.96	0.94
Quan day week 0.5	22.73	0.95	25.79	0.94
Quan day week 0.75	30.20	0.94	27.89	0.93
Individual Parzen model	1.31	0.99	1.90	0.98
Global median model	9.98	1	9.98	1
Global Parzen model	2.00	1	6.88	1

which the attack is to be mixed and *end* is the last. We take one user from the set of evaluated users, mix the attack into his time series and run the evaluation process, which calculates anomaly scores for all the users at the time of the attack. The users are then sorted by their anomaly scores, in descending order. The position of the user with the mixed in attack in that list is said to be his rank. We repeat the procedure for all the users, calculating their ranks. The the average rank and probability of detection can then be calculated from the ranks.

Table 8 gives the ranks for an attack that was mixed into the users of Twitter. Twitter is a popular service among malware creators who like to use it as C&C for their botnets. The feature that the attack was mixed into is the number of requests. The length of time step is one hour and the length of the attack is five time steps. As can be seen from the table, the mean rank decreases as the attack size increases.

Table 9 shows the results of mixing DNS enumeration attack into active users of the DNS service. DNS enumeration [30] is listing of all the available DNS information by repeatedly querying a DNS server. This can be used to gather a lot of crucial information about the network, such as available services and network infrastructure without setting off security alerts. Here we report the outputs of the detectors for two different attack configurations: full intensity and half intensity. We report the mean rank of the attacker, as well as probability that the attacker is within the top ten anomalous users.

Results in Table 9 show good detection quality. It is interesting to note that the results of the detection evaluation are not very correlated with the predictor model error measurements. The two predictors with the lowest error sum as measured in Section 5.4.1 were autoregressive predictor with day week cycle and autoregressive predictor with day week cycle and aggregated memory. In the detection evaluation their performance is average and the best results are achieved by autoregressive predictor with day memory, autoregressive predictor with aggregated day memory and autoregressive predictor with no memory. Another interesting result is the performance of the Parzen models. Both individual and global Parzen models outperformed all the other models by a large margin, achieving very small mean ranks for both full and half sized

attacks.

Having tested all the service usage models against a simulated attack we can conclude that all of them work well, being able to detect the attack. The performance of the models is not correlated with the errors of the predictors. The best results for both individual and global models are achieved by the Parzen window cumulative distribution models.

6 Applications and Discussion

In this chapter we discuss current and potential applications of the methods presented in the thesis.

6.1 Request–response pair matching

Request-response pair matching is a task commonly performed by IDSes that use NetFlows as their data source. In this thesis we presented an efficient algorithm for matching request–response pairs that is easily parallelisable. The request–response pair matching algorithm can provide valuable information about communication relationships in the network.

6.2 Request–response detector

On top of request-response pair matching algorithm we have built a request–response detector which models the amount of requests without responses and reports abnormally high amounts. It is able to detect different types of malicious activity such as command and control search by different malware families, vertical and horizontal scans and denial of service attacks. It is used as a part anomaly detection layer in Camnep [31]. Camnep is a network intrusion detection system that uses an ensemble of detectors and other layers in order to detect anomalies in network traffic.

6.3 Service detection

In this thesis we presented an algorithm that is able to detect actively used network services using NetFlow data. This algorithm is able to actively adapt to different networks and changing conditions in those networks. Service detection can have many potential applications. It can be used by network administrators in order to monitor their networks and have a better of overview of available services and their usage. The service detection algorithm can reveal possible policy violations or misconfigurations of servers which may often lead to undesirable results and monetary losses.

Another aspect of service detection algorithm is malware detection. It is common for malware to use infected servers as command and control channels, often creating new services by opening new ports. The service detection algorithm can be used to detect this type of activity.

It can also be generally useful as part of intrusion detection system, as it can not only be used for detecting command and control channels, but also for creating a more sophisticated model of the network which can be used for creating more advanced anomaly detection algorithms.

6.4 Service modelling - aggregation

As was explained in Section 4.1, the outputs of the service detectors are fed to the aggregation layer which provides a unified output of anomaly values. The aggregation layer was designed by Mgr. Jan Kohout. The aggregation is done in time, taking last k anomaly outputs, as well as across models. The anomaly values are aggregated using different aggregation strategies, ranging from *mean* to *max*. The aggregation strategy is chosen for every aggregation time step using game theory which calculates a Nash equilibrium from a pay–off matrix with different attacker and defender strategies. Using game theory in aggregation maximises the chances of discovering the attacker.

6.5 Service modelling - potential applications

Service modelling has very many potential uses. As was mentioned before, there are several companies that specialise in service monitoring and modelling.

It is common for network administrators to set alarm policies based on fixed thresholds. For example, report an upload larger than 100MB. This has the problem of deciding what is the right threshold. Too high means poor detection and too low means an endless flood of alerts. The service modelling algorithms can be used in order to adaptively set the administrator alert thresholds in order to report abnormal service usage. This can be used to detect security breaches and help prevent service misuse which can result in intellectual property theft.

The output of the service modelling framework can also be used together with other intrusion detection systems and their outputs can be correlated in order to lower the false positive rates. For example, a sign of malware activity followed by a large upload to a service outside the network is something that network security response teams will definitely take an interest in. Service modelling can also be used to detect certain policy violations, which can be defined by the network administrator.

7 Future work

There is always room for improvement. Here we discuss future work which can build on existing methods or improve their weaknesses.

7.1 Service detection

In the future it would be useful to improve the service detection algorithm in order to make it more robust to errors in timestamps. Currently it only works on networks where the NetFlows are gathered with relatively small timestamp errors and breaks when the errors get worse.

7.2 Service classification

As part of future work it would be interesting to design a service classification algorithm. This algorithm would cluster different services together, based on their behaviour. The same service types would fall in the same cluster. This could also be used as an anomaly detector. It would detect when a service of a certain type (determined by port number) falls far away from the cluster of same service types. This could be a sign of a malicious service pretending to be a service of different type in order to avoid firewall rules. This detector would be able to detect various types of malicious activity such as DNS tunnelling.

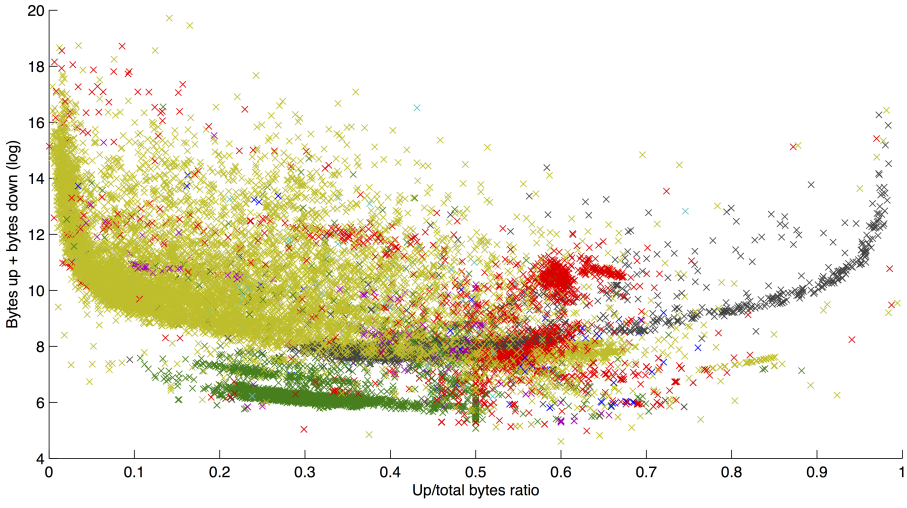


Figure 9 Distribution of individual user-service communications in feature space

Some preliminary work has already been done in this area. Figure 9 shows a distribution of individual user-service communications in feature space. Different colours

represent different service types (determined by port number). It is clear that different service types form clusters. Using more features and different clustering techniques it should be possible to separate different service with high accuracy.

7.3 Service modelling

Future work on service modelling will involve creating more advanced models of service usage by users. It would be interesting to try a third type of user-service relationship and model the usage of several service by one user as one model. It is normal for a user to have a long-term established pattern of services that he uses on a daily basis and only deviate by a small amount. This type of model could monitor this pattern and report deviations, which could mean reconnaissance activity by an infected host.

Different features can be used in service modelling, such as periodicity, persistence and potentially many others in order to create more sophisticated models of user behaviour.

Correlations between different service and service usage in different networks can be studied and used to create different models which will allow to monitor more aspects of user behaviour and service misuse.

An alert system can be created that will adapt based on network administrator priorities and network policies. Services can be categorised in order to create different monitoring policies and alert levels for them. Similarities between services can be used in order to automatically apply policies to uncategorised services based on categorised services with similar behaviour and usage patterns.

8 Conclusion

In this thesis we performed a thorough study of network services. In Chapter 3 we described a request-response pair matching algorithm and an anomaly detector based on it. This detector is able to detect several types of malicious behaviour including various types of scans, ongoing DoS attacks and C&C search as was shown in the experiments (see Chapter 5).

We then proposed a service detection algorithm which uses expectation-maximisation for adaptive learning. Using this approach we were able to correctly discover network services within the observed network. This method relies heavily on the accuracy of timestamp information included in the NetFlow. We have shown that there are network probes that are not able to produce accurate timestamps and we need to check the timestamp accuracy before applying the service detection algorithm.

In Chapter 4 we described several service models that monitor service usage by the users and report anomalous usage. Majority of the models is based on the time series prediction that can be used to detect anomalies with respect to previous user behaviours. We have also introduced Parzen window model that is able to detect anomalies based on feature distributions. Aggregated value of all presented models is used as a final anomaly of specific service user.

In Chapter 5 we experimentally evaluated the proposed algorithms on real network data. We showed that the proposed service models are able reliably detect attacks that were mixed into the real network data.

Bibliography

[1] *More mobiles than humans in 2012, says Cisco*. Feb. 2012. URL: http://www.bbc.com/news/technology-17047406.

[2] *Intellectual Property Theft*. URL: http://www.fbi.gov/about-us/investigate/white_collar/ipr/ipr.

[3] *Norton Study Calculates Cost of Global Cybercrime: $114 Billion Annually*. Sept. 2011. URL: http://www.symantec.com/about/news/release/article.jsp?prid=20110907_02.

[4] Danny Palmer. *World needs 21 million cyber security professionals - but there's only 3,000 now, warns expert*. Oct. 2013. URL: http://www.computing.co.uk/ctg/news/2301230/world-needs-21-million-cyber-security-professionals-but-theres-only-3-000-now-warns-expert.

[5] *Skyhigh Networks*. URL: http://www.skyhighnetworks.com/.

[6] *Elastica*. URL: http://elastica.net/.

[7] *Netskope*. URL: http://www.netskope.com/.

[8] Peter Burkholder. *SSL Man-in-the-Middle Attacks*. Feb. 2002. URL: https://www.sans.org/reading-room/whitepapers/threats/ssl-man-in-the-middle-attacks-480.

[9] Steven J. Vaughan-Nichols. *How the NSA, and your boss, can intercept and break SSL*. June 2013. URL: http://www.zdnet.com/how-the-nsa-and-your-boss-can-intercept-and-break-ssl-7000016573/.

[10] *SSL Visibility Appliance*. URL: http://www.bluecoat.com/products/ssl-visibility-appliance.

[11] Dr. Thomas Porter. *The Perils of Deep Packet Inspection*. Oct. 2010. URL: http://www.symantec.com/connect/articles/perils-deep-packet-inspection.

[12] B. *RFC 3954 - Cisco Systems NetFlow Services Export Version 9*. Oct. 2004. URL: http://www.rfc-editor.org/rfc/rfc3954.txt.

[13] Benoit Claise, Brian Trammell, and Paul Aitken. *RFC 7011: Specification of the IPFIX Protocol for the Exchange of Flow Information*. Sept. 2013.

[14] Fernando Silveira et al. "ASTUTE: detecting a different class of traffic anomalies." In: *SIGCOMM*. Ed. by Shivkumar Kalyanaraman et al. ACM, 2010, pp. 267–278. ISBN: 978-1-4503-0201-2. URL: http://dblp.uni-trier.de/db/conf/sigcomm/sigcomm2010.html#SilveiraDTG10.

[15] Y. Zhang et al. "Network anomography". In: *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. IMC '05. Berkeley, CA, USA: USENIX Association, 2005, p. 30.

[16] *Squid proxy server*. URL: http://www.squid-cache.org/.

[17] Robin Berthier et al. "Nfsight: netflow-based network awareness tool". In: *Proceedings of the 24th USENIX LISA* (2010).

[18] Risto Vaarandi. "Detecting Anomalous Network Traffic in Organizational Private Networks". In: *2013 IEEE CogSIMA Conference*. IEEE. 2013, pp. 285–292.

[19] Eduard Glatz and Xenofontas Dimitropoulos. "Classifying internet one-way traffic". In: *Proceedings of the 2012 ACM conference on Internet measurement conference*. ACM. 2012, pp. 37–50.

[20] K McNamee. *Malware Analysis Report: ZeroAccess*. Tech. rep. Sirefef, 2012. Technical Report by Kindsight Security Labs.

[21] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. ISBN: 9780387310732. URL: `http://books.google.co.uk/books?id=kTNoQgAACAAJ`.

[22] Norton. *Spear Phishing: Scam, Not Sport*. URL: `http://us.norton.com/spear-phishing-scam-not-sport/article`.

[23] P Amini and C Pierce. *Kraken botnet infiltration*. 2008.

[24] Thorsten Holz et al. "Measuring and Detecting Fast-Flux Service Networks". In: *NDSS*. 2008.

[25] Jake D Brutlag. "Aberrant Behavior Detection in Time Series for Network Monitoring." In: *LISA*. 2000, pp. 139–146.

[26] Maciej Szmit and Anna Szmit. "Usage of modified Holt-Winters method in the anomaly detection of network traffic: Case studies". In: *Journal of Computer Networks and Communications* 2012 (2012).

[27] Everette S Gardner. "Exponential smoothing: The state of the art". In: *Journal of forecasting* 4.1 (1985), pp. 1–28.

[28] W. Härdle, Z. Hlavka, and S. Klinke. *XploRe® - Application Guide: Application Guide*. Springer Berlin Heidelberg, 2000. ISBN: 9783540675457. URL: `http://books.google.com.ua/books?id=92Z1YW-IP84C`.

[29] *Mosek*. URL: `http://mosek.com/`.

[30] *DNS enumeration*. URL: `https://pentestlab.wordpress.com/tag/dns-enumeration/`.

[31] Martin Rehak et al. "CAMNEP: An intrusion detection system for high-speed networks". In: (2008).