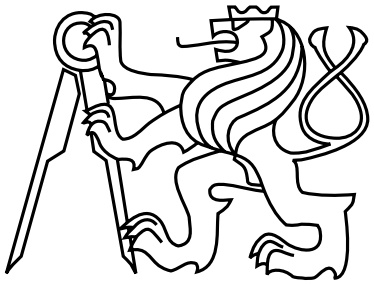
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Diploma Thesis

Anomaly detection of host roles in computer networks

Bc. Yury Kasimov

Supervisor: Sebastian Garcia, PhD.

Study Programme: Artificial Intelligence

Field of Study: Open Informatics

May 24, 2018

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Kasimov Yury

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Anomaly detection of host roles in computer networks

Guidelines:

1. Review the state-of-the art methods for detecting malicious behaviors with special attention to anomaly detection techniques.
2. Propose and implement anomaly detection method to detect changes in host behavior. Infected hosts change their behavior, this method would help to detect them.
3. Experimentally evaluate the proposed solution on datasets from Stratosphere project.
4. Analyze results of the implemented system and propose further improvements and applications of the solution in network security.

Bibliography/Sources:

[1] Arindam Banerjee Varun Chandola and Vipin Kumar. Anomaly detection : A survey. To Appear in ACM Computing Surveys, 2009.
[2] Northcutt, Stephen, and Judy Novak. Network Intrusion Detection. Indianapolis, IN: New Riders Pub., 2002. Print.
[3] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, and P. Dokas. MINDS - Minnesota Intrusion Detection System. In Next Generation Data Mining. MIT Press, 2004.
[4] Martin REHAK ,Michal Pechoucek ,Karel Bartos ,Martin Grill, Pavel Celeda, Vojtech Krmicek CAMNEP: An intrusion detection system for high-speed networks. In Progress in Informatics, No. 5, pp.65?74, (2008)

Diploma Thesis Supervisor: Ing. Sebastián García, Ph.D.

Valid until the end of the summer semester of academic year 2017/2018

prof. Dr. Michal Pěchouček, MSc.
Head of Department

prof. Ing. Pavel Ripka,CSc.
Dean

Prague, February 20, 2017

Aknowledgements

Foremost, I want to express my deep gratitude to my supervisor Sebastian Garcia. He guided and educated me in the fields of network security and anomaly detection. He gave me freedom in pursuing my research and providing me with valuable advice along the way. Sebastian gave me an opportunity to join Stratosphere lab, where he had created a great environment for learning and working. I am very grateful to my coworkers from Stratosphere lab: Maria, Raul, David, Ondra, Veronika, Frantisek, Maria Jose and Karel for creating a friendly atmosphere in which we were working and learning together. Especially, I would like to thank David Kubesa for sharing with me the idea and implementation of profiles. Last but not least I am thankful to my family for their love and support. Most importantly I would like to thank Elnaz for her love and encouragement.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

V Praze dne 25. 5. 2018 .....

Abstract

Detecting malware infections is one of the most challenging tasks in modern computer security. A very common approach in the last years has been to run an anomaly detection algorithm as a first detection approach and then complement the results with another classification algorithm. An anomaly detection method is usually designed to model normal traffic and then to find deviations from that model. However, anomaly detection techniques may suffer from different problems, including no labels for verification and a large number of false positives. This thesis proposes and describes a new profile-based method to detect anomalous changes in user network behaviors. The profiles describe the behavior of users from different perspectives in the network by using multiple features. Each profile encapsulates what the user did in a period of time. The features extracted inside the profile cover a wide range of actions from the user, trying to analyze different actions. Compared to other feature-based anomaly detectors, our profiles offer a more high-level view of the behavior of a user. We hypothesize that it is possible to look for anomalies using high-level features and at the same time produce less false positive while being effective at finding real attacks. The problem of obtaining real labeled data to train our anomaly detection algorithm was also addressed in this work. We designed and created new datasets that contain real normal actions of a human user, while the user is infected with real malware. These datasets were used to train and evaluate our anomaly detection algorithms. The algorithms studied were such as Local Outlier Factor and One-Class SVM. Results show that the proposed profile-based algorithm for anomaly detection produces very few false positive errors and reasonably high true positive detections. There are two main contributions in this thesis. First is the novel profile-based approach for anomaly detection in networks. Second, the datasets which contain the mix of real malware traffic and real user traffic. These datasets are available for download. The focus of the future work will be applying the proposed approach to the

protection of Internet of Things devices.

Abstrakt

Detekce infekcí malwarem je jedním z nejnáročnějších úkolů v oblasti moderní počítačové bezpečnosti. V posledních letech byla jako první detekční přístup běžná detekce anomálií, následně doplněná o výsledky z dalších klasifikátorů. Metody detekce anomálií jsou obvykle navrženy tak, aby modelovaly normální chování a poté hledaly odchylky od tohoto modelu. Nicméně techniky detekce anomálií mohou trpět různými problémy, včetně chybejících anotací pro ověření a velký počet falešně pozitivních detekcí. Tato práce navrhuje a popisuje novou metodu založenou na profilech pro zjištění anomálních změn v chování uživatelů v síti. Profily popisují chování uživatelů z různých pohledů pomocí různých příznaků. Každý profil je složen z informací o tom, co uživatel udělal v určitém časovém období. Příznaky extrahované v profilu pokrývají širokou škálu akcí uživatelů a snaží se analyzovat různé akce. Ve srovnání s jinými detektory anomálií na základě příznaků nabízejí naše profily pohled na chování uživatelů na vyšší úrovni. Předpokládáme, že je možné hledat anomálie využívající příznaky vyšší úrovně a současně produkovat méně falešně pozitivních detekcí a při tom efektivně nalézat skutečné útoky. Také problém získání skutečně označených dat pro trénování algoritmů detekce anomálií byl řešen v této práci. Navrhli jsme a vytvořili nové datasety, které obsahují skutečné normální akce uživatele, zatímco uživatel je napaden skutečným malwarem. Tyto datasety byly použity k trénování a vyhodnocení algoritmů pro detekci anomálií. Mezi zkoumanými algoritmy byly například Local Outlier Factor a One-Class SVM. Výsledky ukazují, že navržený algoritmus založený na profilu pro detekci anomálií způsobuje velmi málo falešně pozitivních chyb a poměrně vysokou míru skutečnou pozitivní detekcí. Dvěma hlavními příspěvky této práci jsou nový přístup založený na profilu pro detekci anomálií v sítích a dále datasety obsahující kombinaci skutečného malwaru a skutečného provozu uživatelů. Tyto datasety jsou dostupné ke stažení. Budoucí práce bude zaměřena na uplatnění navrženého přístupu k ochraně zařízení Internet of Things.

Contents

Chapter 1

Introduction

Detecting intruders and malware infections in local networks is one of the most difficult and most studied challenges in modern computer security [1]. From the huge amount of detection methods proposed, a large majority used static rules or reputation methods for performing the detection; until more modern behavioral techniques were introduced. Although very useful, these static techniques were not enough to detect the majority of attacks and malware. In particular we believe that the more important limitations of the current techniques are that (1) the detections are done per connection and not per user, (2) the classifiers are trained and tested on *only normal* and *only infected* datasets and, (3) the types of attacks and infections evolve and make classifiers quickly less useful.

Apart from the more traditional signature-based Intrusion Detection Systems (IDS), such as Snort [2] and Bro [3], there has been extensive research on behavioral detection methods during the last decade. From these new methods, the most used is anomaly detection (AD) due to its easy implementation and understandability. Anomaly-based IDS detect anomalies by assuming that more than half the data is normal and then searching for some *deviation* from that normality. The main benefits of AD techniques are its ability to detect previously unknown attacks and the identification of non-malicious problems within the network, such as corporate policy violation. Despite its extensive use, AD suffers from several issues that undermine its usefulness. First, it is hard to verify the results, leading to attacks being mixed with normal connections. Second, the anomalies found are not necessarily malicious, generating a high rate of false positives. Third, the nature of the network changes over time, making the original normal model obsolete and prone to more errors. Fourth, AD methods usually work with packets, making the methods a little less stable. The amount of errors generated by these issues tends to be so large that researchers give the output of AD to other algorithms to improve the detection. In consequence, AD models tend to need constant maintenance and supervision even to achieve acceptable results.

To improve the current situation, we propose an AD method that focuses on the high-level changes in the behavior of a user. The proposed anomaly detection method is analyzing time-fixed user profiles and how they change with regard to the past traffic of the same user. Each profile is composed of a set of features based on the flows received and sent during a time-window.

We hypothesize that it is possible to detect the infected users and to have a small number of false positives by focusing on the high-level behavior in time. Our detection approach

consists of monitoring a user's computer, and we collect flows during a fixed time-window. With the data collected in that time window, we create a profile for that user. The profile describes user's behavior during this time window. With the data obtained during the time window of each profile, we compute twelve features. We reference these feature as profile features (for more details see Section 3.3).

The features of a profile cover a wide range of possible changes in traffic to have enough representations of the behavior of the users. For example, one of the features analyzes the number of flows for all the TCP connections from a host, and another feature describes the number of packets for all the TCP connections aggregated by destination ports. If a user is infected with a malware which tries to connect to a large number of different computers, this last feature will show a very distinctive pattern for the number of packets on each destination port. As another example, if malware steals personal information of a user the features of the profile might show a higher volume of packets sent from the user's computer. The profiles generated for each time window may allow us to detect these changes and variances and generate alarms that may prevent the actions of an infected computer.

To be able to detect changes in the behavior of users we need to define a baseline of their behavior. We collected a set of profiles which covers different time intervals such as working hours, night activity, weekends, etc. Within each profile, each feature describes the user's behavior from a different perspective. Each perspective contributes a different type of information to the algorithm, and therefore obtaining detections for different behaviors. We evaluate each feature individually with the same feature in past profiles.

The initial phase of our anomaly detection method consists in preprocessing the collected data and preparing it for the algorithm. We apply dimensionality reduction technique to the features of a profile. Each feature has to have the same dimensionality. Then we normalize the features of a profile. Data normalization might significantly improve the performance of anomaly detection algorithms [4]. The normalization is done for each feature individually with respect to the same feature in other profiles.

After the preprocessing phase, we train the anomaly detection algorithms (see section 3.5) by using the normal behavior of a user. We create an anomaly detection model for each feature of a profile, based on all the data of that feature across all the normal profiles of the user. Each profile has twelve different features. Therefore, each feature has its unique normality model.

Once the twelve normal models were trained, it is possible to evaluate the performance of the algorithms using new unseen profiles. The new unseen profiles contain both normal data and normal plus attack data, and they were generated in the same way as the training profiles used to create the normal models. The anomaly detection algorithms, trained in the normal profiles, are used to evaluate the unseen profiles and assign a label to each feature in these profiles. For each feature, we have a label, and all these labels are used to generate the final label of the profile.

After assigning the labels, the detection method uses majority voting to get the final decision regarding a profile. If six out of twelve of the final labels assigned by the anomaly detection algorithms classify the profile as anomalous, then the final label of the profile is Anomalous.

One of the most important drawbacks of using anomaly detection algorithms is the lack of verified and trusted data. Therefore, several months were spent generating new datasets for

this research. These datasets were created in the Stratosphere IPS lab in the Czech Technical University. The datasets consist of normal profiles and anomalous profiles. A normal profile is a profile which was created before the computer was infected and an anomalous profile is a profile which was created after the infection of the computer. We used a clean virtual machine to create a packet capture for both datasets. The creation of the datasets involved several steps. The first step is to imitate a normal user doing standard things (e.g., checking emails). After some time we infected the machine with a malware while at the same time we continue to perform normal actions.

To evaluate the performance of the algorithm its necessary to have ground-truth labels. We assigned the normal label ($label = 1$) to all the profiles that are created before the infection of the virtual machine. We assigned the anomaly label ($label = -1$) to all the profiles that are created after the infection of the virtual machine. The reason to assign the anomaly labels in this way is the assumption that everything after an infection is worth detecting, and that malware produces changes in the behavior observed in the network. We are aware that not all the attacks are anomalies, but this assumption helps us better evaluate our algorithm.

The datasets and labels are used during the experiments to evaluate our hypothesis and our approach. Each experiment uses one normal dataset for creating the normal model and one mixed dataset of normal plus attacks for evaluating and testing the performance of the model. We split each dataset into three parts: train, validation, and test. The train set contains only normal profiles created before the infection. We use this set to create a model of normal user behavior. The validation set and the test set contains a mix of normal profiles and anomalous profiles.

From all the anomaly detection algorithms that are available, we selected those reported by the community as better for our problem. The algorithms we use in this thesis have different parameters that we can adjust to improve detection. The adjustment of these parameters is done based on the performance metrics computed on the validation set. The validation results allow us to select the best models (the model selection is described in 3.7) and later compute the final performance metrics on the test set.

The evaluation of the algorithms is done in two ways: per individual feature and for the whole profile. First, each feature gives a label for the profile from the point of view of this feature. Then the results are combined using the majority voting to get the final decision for the profile. Having a result per feature allows us to evaluate features individually and learn which ones contribute the most to the detection of the anomalous behavior produced by malware. The analysis of the performance of individual features might also help us better understand how malware communicates.

The analysis of the results is also done by analyzing individual features and then analyzing the result of the majority voting. The analysis of the results shows that most experiments achieved a very low FPR for individual features in our experiments. FPR does not exceed 0.01 in three out of four experiments for all features. When we use majority voting to produce the final label, the FPR was 0.0 in all experiments. The highest achieved TPR was 0.44. Although it may seem low, this is the final result among all tested profiles with the FPR 0.0. It means that the algorithm will detect one out of three anomalous profiles with 100% success, while at the same time it will not detect a normal profile as an anomalous.

One of the contributions of this thesis is a creation of public dataset with real user and malware being active at the same time. Another contribution is the proposed method which is novel in that it analyzes features individually across time. Results are promising and show that our high-level analysis may provide a good improvement over the current techniques. The proposed algorithm achieved a low False Positive Rate and reasonably high True Positive Rate.

Chapter 2

Related Work

In this chapter, we discuss related works in the area of anomaly detection for network security. There is much research that has been done in the field of anomaly detection in general and its application to the network security domain in particular.

There are multiple survey papers of algorithms for anomaly detection. Chandola et al. [5] reviewed different types of anomalies, the different fields where anomaly detection is used, challenges of anomaly detection and algorithms that could be used for anomaly detection. The paper mentioned that one of the main challenges of applying anomaly detection in the field of network security is that the nature of anomalies keeps changing with time and intruders try to adapt to evade detection.

A central premise of anomaly detection for security was defined by Patcha et al. [6] as that intrusive activity is a subset of anomalous activity. He mentioned that activities in a network could be split into four categories:

- Intrusive but not anomalous — the source of false negatives
- Not intrusive but anomalous — the source of false positive
- Not intrusive and not anomalous — true negatives
- Intrusive and anomalous — true positives

Among the systems that use flows for anomaly detection is Minnesota Intrusion Detection System (MINDS) [7] proposal. The system extracts the following features: source and destination IP addresses, source and destination ports, protocol, flags, number of bytes and number of packets. MINDS computes the anomaly score for each IP flow individually. As an anomaly detection algorithm, the creators of MINDS used Local Outlier Factor [8]. One of the main differences between MINDS and we propose that we search for anomalies closer to the actions of the user, and not to the network. We study anomalies in time, from several aggregations of the type of flows.

The methodology used in MINDS wass used by Rehak et al. [9] to develop an agent-based system to detect anomalies in networks by using multiple correlated anomaly detection techniques. Rehak et al. used NetFlow data and built a trust model to reduce the number of false positive alarms. They combined the output of each agent to build a trust model.

Each agent used not only past observations but also anomaly assessments obtained by other agents.

The algorithms that can be used for anomaly detection are varied and include any algorithm that can differentiate between distributions of data. This is the case of One-Class SVM, that has been used for anomaly detection by Zhang et al. [10]. Zhang et al. used One-Class Support Vector Machines to detect anomalies. They evaluated their approach on the dataset KDDCUP99 which was created in 1999. The algorithm showed very promising results compared to other methods. Authors mentioned the problem of obtaining a good dataset with labels to evaluate the anomaly detection methods.

Xu et al. [11] also used NetFlows to analyze the traffic. Their system created a cluster for each IP in the current time window. Clustering was based on the $srcIP$. For each cluster, the system computed the normalized entropy of $scrPort$, $dstPort$ and $dstIP$ and used it as a feature vector to represent clusters. Then the system applied behavior classification scheme to classify each sample in its behavioral class. The proposal of Xu et al. was similar to our proposal in the sense that they also saw the information about Source Ports and Destination Ports as important for anomaly detection.

Ideas of ensembling multiple anomaly detectors to lower the false positive rate were explored by Martin Grill, PhD. in his doctoral thesis [12]. The author developed new anomaly detectors which used NetFlows or HTTP logs. One of the proposed techniques in this work was to replace the output of an anomaly detector by an average of anomaly scores of similar flow in the past. The method was called local adaptive multivariate smoothing (LAMS), and the results of the evaluation showed a decrease in the number of false positives.

All the features in this thesis use the state field from IP flows to specify if the connection was established or not established. Mahoney et al. [13] also inspected TCP flags but based on individual packets. The proposed NETAD algorithm [13] built nine models to identify anomalies in nine subsets of packets. Packets were split into subsets based on TCP flag in the packet and on the port. The algorithm achieved 66 detections out 185 with only 20 false alarms.

One of the examples of creating normal traffic profiles is *FireSIGHT* tool from Cisco [14]. This tool allowed a user to specify a sliding time-window length and traffic profile would be created during this window. After the profile was created, the tool allowed detecting abnormal network traffic by comparing it to the profile. To detect abnormal traffic user should define correlation rules which would be triggered ones the traffic deviates from the normal profile. The paper [14] does not give details about profile creation.

Another example of profiling a user was presented by G.Pannell et al. [15]. The profile was created using multiple characteristics such as the number of running applications, keystroke analysis, websites viewed, application performance and the number of windows. Each characteristic was modeled separately, and then the evaluation results were combined using a weighting algorithm to produce the final decision. The results showed that combining results was producing a lower false positive rate than individual characteristics.

The features used for detecting anomalies in this thesis were a subset of features created by David Kubesa [16]. He defined a lot of different features to identify users in the network even if they would connect from a different place. We selected a subset of features which describe outgoing traffic from a computer of a user since we wanted to detect the infection of the computer.

Chapter 3

Anomaly Detection Method

To create a new anomaly detection method it is necessary to define what should be detected and why. The definition of anomaly depends on the goal of the system, the data available and the conditions of execution. In this thesis, we want to detect when a computer is infected by malware while its still acting normally in the network. Our focus is the malware infections of computers in networks. The data available are network packets, but we decided to use NetFlows to process all the information quickly and to preserve the privacy of users. In consequence, our method is also evaluating if the use of NetFlows may be enough for a good anomaly detection method. The constraints of our method are that detections should be reported as soon as possible and that the number of False Positives should be minimized.

Our proposed method analyzes anomalies in the behavior of the computers from a high-level perspective. This perspective is the actions of the user as they are reflected in flows in the network. Every time a user interacts with a computer, packets are sent via the network. These packets are grouped into flows according to their protocol. These flows are further grouped in this thesis in specific new features, such as the number of flows sent to each destination port. These features are a *higher level* perspective of the actions in the network.

To obtain a detection as soon as possible the traffic of each computer is separated in time windows of five minutes. These time windows allow our method to run quickly, to capture enough traffic to model behavior, but not to be too big to process. The decision taken by our anomaly detection method is *per time window*.

3.1 NetFlows

NetFlows are a data structure developed by Cisco Systems that allow to capture and aggregate information about network traffic Each packet which is forwarded through a router or a switch is examined for a set of IP packet attributes. Another way to generate NetFlows is to use a tool (such as argus) to generate them directly from network traffic. Usually, packets are identified by the following attributes [17]:

- IP source address
- IP destination address

- Source port
- Destination port
- Layer 3 protocol type
- Class of Service
- Router or switch interface

After the examination, all packets are grouped based on the attributes described in the list [17]. Constructing anomaly detection methods using NetFlow data has been a subject of research in many works, such as [18]. NetFlow data are easy to generate. We can generate flows from traffic captures or obtain it directly from a router. Also, NetFlow data preserves the anonymity of the users, because it does not contain the content of packets.

The flows in our experiments were generated from the capture *pcap* files, using the Argus tool [19]. One of the reasons that we use Argus is the option to generate bi-directional flows. Bi-directional flows contain information about packets sent in both directions. Argus can generate additional fields to the ones that used for flow creation.

We configure argus to give us some additional information about packets:

- *StartTime* — the time, when the first packet was forwarded
- *Dur* — duration of the communication
- *Proto* — protocol of the transport level
- *srcAddr* — source IP address
- *dstAddr* — destination IP address
- *dstPort* — destination port
- *Flag* — which flags were seen during the communication
- *SrcPackets* — the number of packets that were sent from the source IP to the destination IP during the communication
- *DstPackets* — the number of packets that were sent from the destination IP to the source IP during the communication
- *srcAppBytes* — the number of application bytes that were sent from the source IP to the destination IP during the communication
- *dstAppBytes* — the number of application bytes that were sent from the destination IP to the source IP during the communication
- *srcBytes* — the number of all bytes that were sent from the source IP to the destination IP during the communication
- *dstBytes* — the number of all bytes that were sent from the destination IP to the source IP during the communication

The *Flag* field in flows contains two parts separated by an underscore. These are the TCP flags used in the packets in the flow. In the state field generated by Argus, the letters to the left of the underscore character are the TCP flags used in the packets going from the source to the destination. The letters to the right of the underscore character are the TCP flags used in the packets going from the destination to the source.

3.2 Established and not Established Connections

We determine if a flow between two computers is established or not by the $flag$ field. An established TCP connection is the one which completed a three-way handshake [20]. For example, these are the flags of an established connection flag: $SRPA\_FSPA$. The Argus state field summarizes the TCP flags used in the packets. In flow state there are the following TCP flags:

- S — synchronization bit (SYN)
- R — reset bit (RES)
- P — push bit
- A — acknowledgment bit (ACK)
- F — final bit (FIN)

These packets could have been sent in any order.

An example state of a not established TCP connection is S. It means that the source IP address initiated a connection with SYN flag and did not get any response.

For the UDP protocol argus uses flags such as CON and REQ which are set by $argus$. CON flag is set in case of an established UDP connection. REQ flag is set if a client tried to establish a connection but a server did not send anything in response.

3.3 Profile to Identify Behavior

A profile is a high-level representation of user behavior in a network. To create a profile, we collect network traffic over a predefined time-window. Currently, the creation of the profile only includes the IP protocol version 4 (not version 6), and the TCP and UDP protocols. We do not include other protocols such as ICMP because they are by far the minority of the packets.

For each protocol, we define a set of features. After experimenting with different features, we decided to use the features created by David Kubesa in his work which is under publication.

Each feature is constructed in the following way: First, the purpose of the feature is decided. For example, to capture the variations in the destination ports, according to the flows used by the computer being analyzed when the connection is successful. Second, a subset of all the flows in the current time-window is selected according to the previous

criteria. In our example, only the established flows are selected. Third, the subset of flows is separated into two groups, one for the TCP protocol and one for the UDP protocol. This separation is done because the purpose of applications using the TCP and UDP protocols is very different and should not be mixed in a single feature. Fourth, the desired field is extracted for all the flows. In our case the destination port. So far there are two groups of data, one has all the destination ports for established TCP connections, and the other has all the destination ports for established UDP connections. Fifth, the extracted data is used to create a histogram of the number of flows per destination port. After these steps there are two features, both having a list of values that represents a probability distribution in a time-window. We would call the first feature *Client Destination Port Number Of Flows TCP Established* and the second feature *Client Destination Port Number Of Flows UDP Established*.

We represent features in a profile as vectors of real numbers $f_k = (x_1, x_2, ..., x_{65535})$, where x_i is a value for i-th port and $65,535$ is the maximum amount of ports available. A profile contains the following set of features:

- Client Destination Port Total Bytes UDP Established – distribution of a total number of bytes over ports for established UDP connections.
- Client Destination Port Number Of Flows TCP Established – distribution of a total number of flows over ports for established TCP connections.
- Client Destination Port Number Of Flows UDP Not Established – distribution of a total number of flows over ports for not established UDP connections.
- Client Destination Port Total Packets TCP Established – distribution of a total number of packets over ports for established TCP connections.
- Client Destination Port Number Of Flows UDP Established – distribution of a total number of flows over ports for established UDP connections.
- Client Destination Port Total Packets TCP Not Established – distribution of a total number of packets over ports for not established TCP connections.
- Client Destination Port Total Bytes UDP Not Established – distribution of a total number of bytes over ports for not established UDP connections.
- Client Destination Port Total Bytes TCP Established – distribution of a total number of bytes over ports for established TCP connections.
- Client Destination Port Total Packets UDP Not Established – distribution of a total number of packets over ports for not established UDP connections.
- Client Destination Port Number Of Flows TCP Not Established – distribution of a total number of flows over ports for not established TCP connections.
- Client Destination Port Total Bytes TCP Not Established – distribution of a total number of bytes over ports for not established TCP connections.

- Client Destination Port Total Packets UDP Established – distribution of a total number of packets over ports for established UDP connections.

We reference these features as "profile features". The profile features describe the behavior of a user from different perspectives.

Figure 3.1a shows a feature "Client Destination Port Total Bytes UDP Established" of a normal profile and Figure 3.1b shows the same feature of an anomalous profile. A logarithmic scale is used for the y-axis to allow a large range to be displayed without small values being compressed down into the bottom of the graph.

3.4 Host Behavior Using Profiles

In the previous section, the complete content of a unique profile was described, including its twelve features. These profiles are the basic unit of analysis of our anomaly detection method, and together they are part of the complete behavior of the user. This section first describes how the profiles are used to build the behavior of a user, and then it describes which is the behavioral analysis method used by our technique.

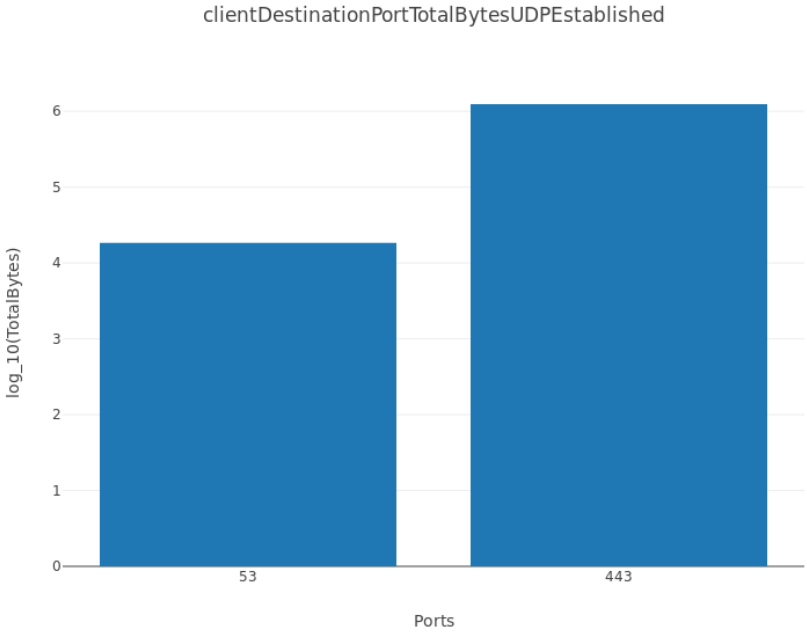
The behavior of a user is defined by all the actions and decisions taken by the user in a certain period. Their actions are transformed into packets and flows, which are then captured in the already described features of a profile. This allows each profile to describe the behavior of a user from twelve different points of view, each capturing a different perspective. As time goes by and the user generates more traffic, more and more profiles are generated.

The behavior of the user is then defined by all these profiles and their features. However, each feature describes the same data differently and therefore it does not make much sense to compare each feature with each other. Instead, we propose to compare each feature in the profile, with the same feature in the *rest* of the profiles for this user. The idea is that anomalies will arise when the same feature is analyzed in the concatenation of profiles. Figure 3.2 shows the idea of searching for anomalies on the same feature on successive profiles.

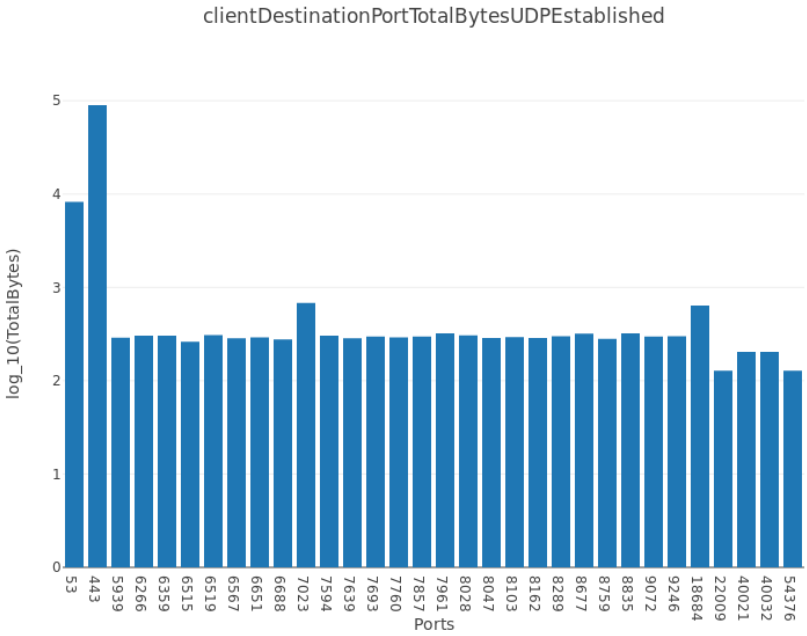
Each feature in the profile corresponds to some measurement of data per port. Interpreted as a histogram, each feature is defined in the space of $65,535$ dimensions. Working with such a large space has two main limitations. First, data in the histogram are sparse, since most of the ports are never used. Second, the algorithms that analyze this data will have to deal with an increased, and probably unnecessary, complexity. Therefore, a reduction in the dimension of the space of each feature is necessary.

3.4.1 Dimensionality Reduction of the Feature Space

After normal profiles are collected, we aggregate features by type of data it meausers over all profiles, and we arrange them into matrices P_i (i is a feature number). The matrix P_i is a sparse matrix, and it has dimensionality $n \times 65,535$, where n is the number of profiles we have collected and $[0 - 65535]$ is the range of ports. Such number of dimensions might cause a problem with scalability and with the performance of anomaly detection algorithms. To improve the scalability and performance we need to reduce the number of dimensions of the data.



(a)



(b)

Figure 3.1: The Figure on top (a) shows an example histogram for the feature 'Client Destination Port Total Bytes UDP Established' for a normal profile. It can be seen that the amount of ports is small. The Figure at the bottom (b) shows an example histogram for the same feature but for an anomalous profile (an attack was being done). It can be seen a large number of ports used.

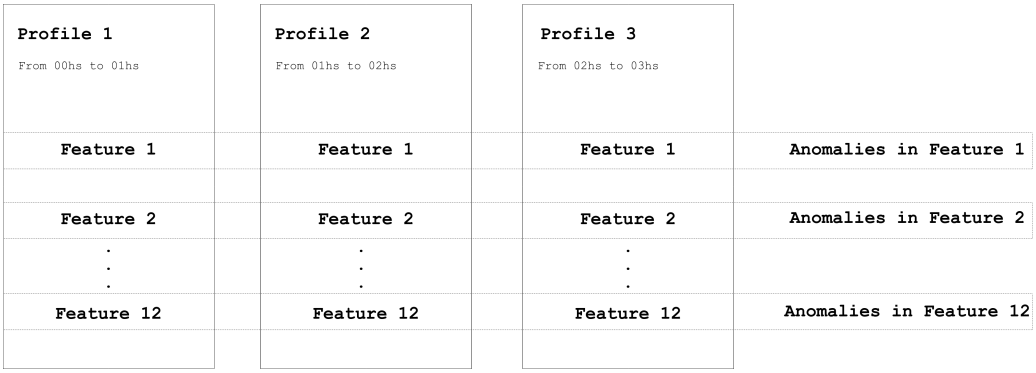


Figure 3.2: Design of our anomaly detection technique. Instead of comparing each feature with each other, our method searches for anomalies on the same feature on successive profiles.

A common dimensionality reduction approach is is *Principal Component Analysis* (PCA). It is a well-known dimensionality reduction technique [21]. PCA was successfully used in [22] to reduce the number of dimensions in features derived from web logs.

PCA derives a reduced set of the most significant uncorrelated features (principal components) that are linear combinations of the original set of features [23]. The new principal components are vectors in the direction of the largest variance of the dataset.

Given m features, PCA selects $d < m$ principal components which define a new k-dimensional space based on normal profiles. There are two ways of specifying d: one can either set d to a fixed number or specify a percentage of variance to preserve, and d will be computed based on this percentage. In our case, the PCA algorithm was configured to preserve 99.9% of variance. We have observed that we could reduce dimensions to $d = 8$ in some cases and due to a high percentage of preserved variance we were not losing much information.

However, after some experimentation, we realized that we had a problem with using PCA in our approach. The problem was that an infected computer used ports which were not commonly used by a normal computer. Because ports were not used by the normal computer, the matrix P_i always had 0 values in the columns corresponding to these ports. PCA was not using these columns when learning how to transform P_i to P_i' in a new basis.

When we applied the transformation learned on the normal profiles to an anomalous profile in which an anomaly was reflected in columns which were not used in creation of the new basis, we lost information about the anomaly.

Because of this problem we have decided against using PCA in our approach. Due to a problem of unseen ports, we apply a different method of dimensionality reduction. We combine ranges of ports which greatly reduces the number of the feature while preserving anomalies. To reduce dimensions of a profile we are doing the following steps:

1. Ports in the range $1 - 1,000$ are left as they are with bin size equal to one port. This is done because the majority of servers use ports in this range for listening.

2. Ports in the range $1001 - 65535$ are split into bins with the size of 100 ports

3. The Value of each bin is an average of values of the ports in the bin

The size of the bin was chosen to be 100 because it brings us a significant reduction of dimensions and also preserves enough granularity of the data so we can detect anomalies.

As a result of this method, we get a reduced histogram with $1,647$ bins. The new matrix P_{red} has a dimensionality $n \times 1647$.

3.4.2 Anomaly Detection Methodology

In this section, we describe our complete methodology step by step. The assumption of the approach is that after an infection the behavior of a host is changing, and the proposed algorithm have to be able to detect these changes.

Each profile feature is represented by a vector of real numbers. We propose to use the Euclidean distance to compare a profile feature along with other profiles. Figure 3.3 shows the training process of the proposed approach. During the training, we use normal profiles which are collected at different time intervals. The collected profiles cover the time when a user is active and when the user is idle, for example during the night. Providing a model with different types of the behavior of the user, allows the model to better generalize the behavior.

The dimensionality reduction does not require training, and it is the same for each profile feature. Therefore it is applied during profile creation. The profile features are grouped based on the information they capture. After features have been grouped we center and scale the profile features so all components of individual profile features will have 0 mean and unit variance. The original mean value and standard deviation of components are saved for future because they are required to center and scale new profiles. We use *StandardScaler* from python library *sklearn* for this task.

Then we use profile features as an input to the anomaly detection algorithm to train models. The AD algorithm is applied to each group of features and for each group it learns a model. After the training phase there are the following models:

- Twelve pairs of parameters ($[\mu,\ \sigma]$) for scaling and centering of features.
- Twelve models to classify profile by individual features.

Figure 3.4 depicts the inference phase of the proposed algorithm. The algorithm uses the models created during the training phase. The scaling and centering is applied to individual profile features of a testing profile. The scaled features are used as an input to the trained models. Each model produces one of two labels: 1 if the model considers the testing profile normal according to this feature or -1 if it considers the testing profile anomaly according to this feature. Each output of the models contributes equally to the final decision. If six out of twelve models classify the profile as normal, the final label will be 1. Otherwise, the profile is labeled with -1.

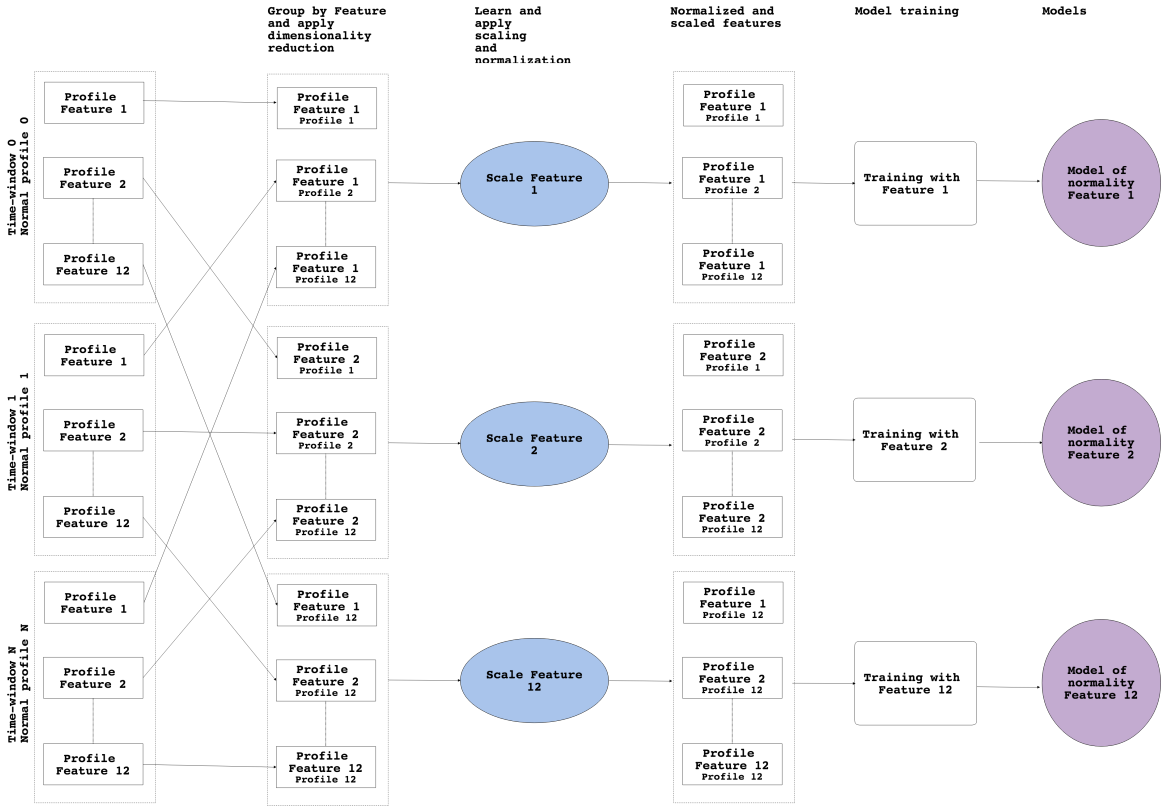


Figure 3.3: The diagram shows the training process of the proposed approach. The training uses N collected normal profiles. There are twelve pairs of parameters ($[\mu,\ \sigma]$) for scaling and centering of features and twelve models after the training.

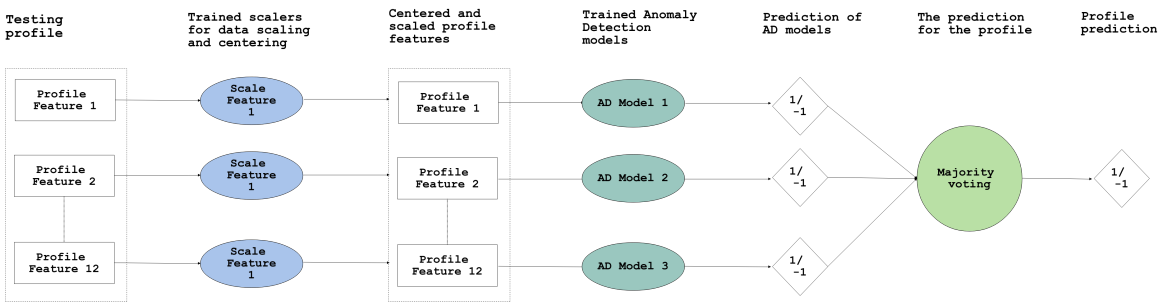


Figure 3.4: The diagram shows the inference process of the proposed approach. There is one profile which we want to classify. We scale and center each profile feature to the same scale as in training profile features. We use the trained models to give predictions for each feature and then apply the majority voting to get the final prediction for the profile.

3.5 Anomaly Detection Algorithms

The previous sections described how the flows are processed and how the features are created to obtain a suitable set of data to work on. This section describes all the anomaly detection algorithms selected and tested in order to find the best one. Among all the possible options, the following algorithms were selected:

1. Local Outlier Factor [8] — uses collected normal profiles to compute an anomaly score of a new profile.
2. One Class SVM [24] — outputs a boundary around normal data.
3. Isolation Forest [25] — method based on Random Forest. Outputs a model of normal data.

The following subsections describe the details of how each of these algorithms works.

3.5.1 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm assigns an anomaly score to each data point based on the idea of density. The LOF measures how density around a point differs from the density of its neighbors. It detects outliers in data on regions with different densities.

In Figure 3.5 there are two clusters C_1, C_2 and two additional points o_1 and o_2. It can be seen that the C_2 cluster is much denser than the C_1.

According to Hawkins' [26] both points (o_1 and o_2) are outliers. However, it can be shown [27] that there does not exist any distance-based detector that can mark o_2 and not mark all points in the C_1 cluster.

This example shows that distance-based methods have a problem if there are regions with different densities in the data. The Local Outlier Factor (LOF) algorithm presented in [27] solves this problem by assigning a value to each object which represents its anomaly score.

This example shows that the distance based methods have a problem if there are regions with different densities in data. The Local Outlier Factor (LOF) algorithm presented in [8] solves this problem by assigning a value to each object which shows the degree of it being an anomaly.

To use the LOF algorithm, authors in [28] define several notions: ϵ-neighborhood and k-distance.

Let p be an object from a database D, let ϵ be a distance value, let k be a natural number and let d be a distance metric on D. Then:

Definition 3.5.1 *: (ϵ-neighborhood)*
The ϵ-neighborhood are the objects x with $d(p,x) \leq \epsilon : N_\epsilon(p) = \{x \in D | d(p,x) \leq \epsilon\}$

Definition 3.5.2 *: (k-distance) The k-distance of p is the distance $d(p,o)$ between p and an object $o \in D$, such that it holds at least for k objects $o' \in D$ it holds that $d(p,o') \leq d(p,o)$ and for at most $k-1$ objects $o' \in D$ it holds that $d(p,o') \leq d(p,o)$*

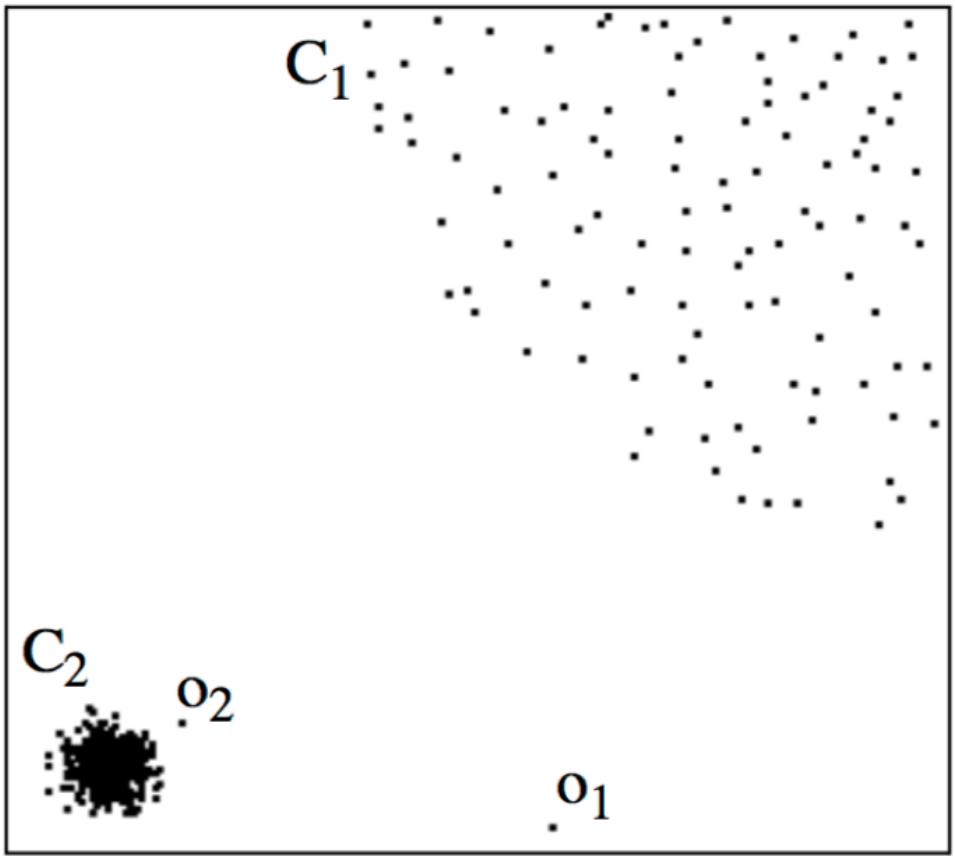


Figure 3.5: Figure shows an example situation when there are clusters with different density. It is a demonstration of the advantage that LOF have over other distance based algorithms. This example is taken from *LOF: Identifying Density-Based Local Outliers* [8]

Definition 3.5.3 *: (reachability distance of object p with regard to object o)*
Let $k \in \mathbb{N}$. The reachibility distance of object p with respect to object o is defined as $reach\text{-}dist = max\{k\text{-}distance(o), d(p,o)\}$

In words, all objects that belong to the k- neighborhood of an object p are considered to be equally distant from the object p.

The next equation is the equation of *local reachability density*. It is an inverse of the average reachability of the object p from its neighbors.

$$lrd(p) = \frac{1}{\frac{\sum_{o \in N_k(p)} reach-dist_k(p,o)}{|N_k(p)|}} \tag{3.1}$$

The Local Outlier Factor is computed by comparing with local reachability distances of the neighbors:

$$LOF_k(p) = \frac{\sum_{o \in N_k(p)} \frac{lrd(o)}{lrd(p)}}{|N_k(p)|} \tag{3.2}$$

For any object which is inside a cluster, the local outlier factor will be around 1. It does not depend on the density of a cluster, and it will be the same for objects inside cluster C_1 and objects inside C_2 [29] as is depicted in Figure 3.5.

The main drawback of the LOF algorithm is its time complexity. Computation of the local outlier factor has the complexity $\mathcal{O}(n^2)$, because it requires computing pairwise distances between all data point. In the paper [7] a suggested approach is to sample training data and compare all data points to this sample.

3.5.2 One Class SVM

One-class SVM is an algorithm that identifies regions of space by their support vectors, of which there are far fewer than data points. The one-class SVM algorithm solves the following optimization problem to compute the support vectors [24]:

$$\begin{aligned} &\min_{w \in F, \xi \in R, \rho \in R} \frac{1}{2} w^2 + \frac{1}{mv} \sum_{i=1}^{m} \xi_i - \rho \\ &\text{subject to} \\ &w \cdot \Phi(x_i) \geq \rho - \xi_i \; \forall i = 1, \ldots, m \\ &\xi_i \geq 0 \; \forall i = 1, \ldots, m \end{aligned} \tag{3.3}$$

Where $\Phi : R^n \mapsto F$ is a nonlinear mapping from data space R^n to feature space F, ξ_i is a slack variable one for every data sample, ρ is a distance from the hyperplane to the origin in feature space, and v is the expected fraction of data samples outside the estimated support.

The one-class SVM algorithm depends on the choice of two parameters: ν and σ. The parameter ν controls the sensitivity of the model. More precisely it controls the ratio of outliers in the data. The parameter σ controls the number of support vectors. The lower value of σ leads to "remembering" the training dataset and the model overfits. The larger value leads to oversimplifying dependencies in the dataset, in other words, it leads to high bias.

The other important decision one has to make when training an OC-SVM model is a choice of kernel. The general advice is to use an RBF kernel because usually, it performs better on different datasets [30]. We have experimented with different kernels, and the RBF kernel was showing the best results. RBF kernel adds one more parameter which can be adjusted to change the performance of the OC-SVM algorithm. The parameter γ controls how far the influence of a single training example reaches. The smaller value of γ means that a single example influences other examples far away and the larger value means its influence is shorter.

3.5.3 Isolation Forest

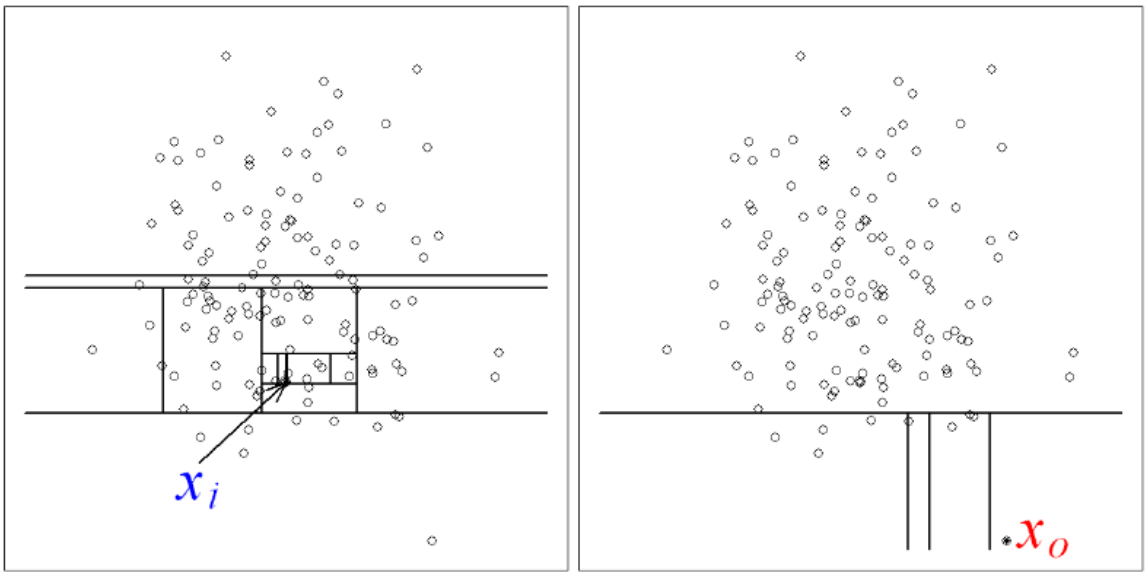
Isolation Forest is a model-based approach to detect anomalies. In the context of Isolation Forest "*isolation*" means "separating an instance from the rest of the instances" [25].

The algorithm constructs trees which isolate every single instance of data. Because anomalies are different, they will be isolated faster by the algorithm, which means they will be closer to the root of a tree [25].

To achieve this, Isolation Forest takes advantage of two properties of anomalies [25]:

1. they are the minority
2. attribute-values of anomalies significantly differ from normal samples

An example of isolation can be seen in the Fig 3.6



(a) Isolating x_i (b) Isolating x_o

Figure 3.6: The left Figure depicts the isolation of a normal data sample. The right Figure depicts the isolation of an anomalous data sample. Image is taken from *Isolation forest* [25]

Authors of the paper define *path length* of a point x as the number of edges the point traverses until it terminates in the end node.

There are two stages of anomaly detection with Isolation Forest. The first stage is a training stage. In this stage, the trees are constructed recursively until all instances are isolated or the specified tree height is reached. The theory behind growing the trees up to some height is that we are mostly interested in points which have a shorter than average path distance [25]. There is no need to grow the trees until each point is isolated. The second stage is the evaluation. The anomaly score of a sample is based on the average path length from root to its termination node.

3.6 Train and validation

We are going to use cross-validation which is a common technique to estimate a test error of a model [31]. We split a dataset and hold out a small subset of data to test our model performance and to make sure that the model does not just memorize the dataset.

We want to make sure that our model does not have a high FPR and also that it detects anomalies. Also, we want to prevent the model from overfitting. It means that a model is trained too much and it is fitted too close to the train set. The main sign of overfitting is that a model has the very low error on the train set, but a much higher error on a validation set.

We split our data into three sets: train, validation, and test. The train set consists of normal profiles only. The validation and test sets consist of normal and anomalous profiles. We take all anomalous profiles randomly select half of them and insert them into validation set. The other half is inserted into the test set. The split is the following:

- Train set — 70% of normal data.
- Validation set — 15% of normal data and 50% of anomalous profiles.
- Test set — 15% of normal data and the other 50% of anomalous profiles

As we mentioned before our data are scaled and normalized to have 0 mean and unit variance. To avoid information leak from the training data to test data, we learn a mean value and variance for scaling and centering only on train data, and then we apply scaling to validation and test data. The information leak might lead to test error underestimating the actual error.

3.7 Model Selection

In section 3.5 we have described the algorithms that we have tried for anomaly detection. Local Outlier Factor and One-Class SVM have both hyperparameters that we can tune to improve the performance. When we search for the optimal set of parameters for our model, we conducted a grid search over a range of possible parameter values. If a hyperparameter is restricted to some range of values we use that range. If a hyperparameter is not restricted we select some reasonable size and search inside it.

The evaluation of a network anomaly detection algorithm is a very important step in showing the advantages and efficiency of the proposed method. The main challenge is to acquire labeled data to measure the performance of the model.

To select a model we use the following criteria:

1. Find a set, A, of models with the lowest FPR.
2. Select a model s_{lowest} with the highest TPR from the set A
3. Find a set of models B with the FPR less than 0.01
4. Select a model $s_{threshold}$ with the highest TPR from the set B

5. Select the final model with the highest FPR by comparing FPR of s_{lowest} and $s_{threshold}$

The FPR metric is very important in anomaly detection. If a model has FPR around 0.05 in a small network with 10 computers and 5 minutes profiles, it will generate 144 false reports daily. Since each report should be checked manually by a system administrator, it would consume many resources.

We try to find model parameters which will have a good trade-off between FPR and TPR. The goal is to maximize TPR while at the same time keeping the FPR below 0.01 or 1%.

Chapter 4

Dataset

The characteristics of the dataset used for training and testing a machine learning algorithm are very important for the results obtained. In fact, the dataset is so important that it completely define if the algorithm works or not, it defines its performance and its generalization power. It does not make much sense to talk about if an algorithm is good or bad without discussing the dataset used.

The dataset used in this thesis was created from scratch to fulfill our requirements. In particular, it was very important to have a dataset of real malicious activities *at the same time* that the normal user is also using the computer. This was achieved through a large process of configurations and infections.

The setup to create the datasets consisted of a Windows 7 virtual machine running on VirtualBox. The traffic was captured by VirtualBox in a pcap file. After the capture was finished, the pcap file was processed with the Argus tool to obtain bidirectional flows.

To work with an anomaly detection algorithm, its necessary to have a dataset which contains two main parts: normal user activity, and a mix of normal activity and malicious activity. The normal user activity is used for creating a model of normal traffic. Later on, the models are evaluated using the mix of normal traffic and malicious traffic.

To generate normal activity, we have created multiple accounts in different services such as Facebook, Gmail, Dropbox, and Twitter. All these accounts were used to generate real normal traffic, where the user creates and writes in new documents, chat with friends and synchronizes data in the cloud. There is also normal activity such as visiting websites, searching for information and downloading files, including executable files. In all our datasets the normal activities lasted several hours, up to several days.

After the normal activity was done, the computer was infected with some malware while the user keeps doing normal actions. This mixed traffic was kept for several hours also. During all the experiments all the activities done by the normal user were logged and stored. This log was later used for labeling the profiles with normal and anomalous labels.

In total three complete datasets were generated for this thesis. The following sections describe in detail the actions and characteristics of each of the three datasets.

4.1 First dataset

The first dataset consists of two parts. Each part is a traffic capture of the same normal user and the same malware. Both parts consist of a normal user and the execution of the malware called Emotet (MD5 8a5d3cada819fe7fd9db67d8c0af120e). This dataset has been created during a complete week, although most of the traffic was seen in few hours. It has, in total, 6 hours of active normal user traffic before the infection, 113 idle normal user traffic before infection, 5.5 hours of normal active user plus malicious traffic, and 22 hours of idle user traffic and malicious traffic.

Summary of the first part of the first dataset:

- Total Duration: 140 hours
- Before infection: 116 hours
  - Active User: 3 hours
  - Idle User: 113 hours
- After Infection: 24 hours
  - Active User: 3 hours
  - Idle User: 21 hours

This first part of this dataset is completely available and can be downloaded from the dataset repository of the Malware Capture Facility Project[1].

To obtain the second part, the Virtual Machine was restored as clean, and a new capture was started. This second part of the first dataset lasted 6.5 hours. It contains 3 hours of normal user activity and 3.5 hours of normal activity and malware activity. The malware used for this second dataset is the same as the first part of the dataset.

Summary of the second part of the first dataset:

- Total Duration: 6.5 hours
- Before infection: 3 hours
  - Active User: 3 hours
  - Idle User: 0 hours
- After Infection 3.5 hours:
  - Active User: 2.5 hours
  - Idle User: 1 hours

[1] https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-9-1/

This second part of this dataset is completely available and can be downloaded from the dataset repository of the Malware Capture Facility Project[2].

During the training and evaluation of our algorithm, these two parts of the first dataset were combined. So both normal parts were treated as one, and both normal plus infected parts were treated as one. For our anomaly detection algorithm, this way of using the dataset does not affect the results, since we split the datasets in time-windows.

The training part of this dataset consisted of 7 hours of normal user traffic before infection (active user) and 30 hours of normal idle user traffic before infection. We selected 30 hours of idle traffic to represent the idle traffic during a day and a night. The verification part of this dataset consisted of 3 hours of flows from the second dataset.

4.2 Second Dataset

The second dataset consists of a normal capture that runs for 43 hours. The normal traffic includes activities such as browsing the web, accessing and using a real Facebook account, watching Youtube videos, sending and receiving emails, etc. After creating the normal traffic, we took a snapshot of the virtual machine to restore it later. Therefore the normal only part of this dataset, which lasted 43hs could be shared with the third dataset. The idea is that since the user and the computer are the same, this baseline may be used to detect the different type of anomalies. The second dataset was infected with the Malware called DarkVNC (MD5 01968695156dc80fcbd9a2b43a88d09c). After infection, the activities continued for 41 hours. This dataset is completely available and can be downloaded from the dataset repository of the Malware Capture Facility Project[3].

Summary of the second dataset:

- Total Duration: 84 hours
- Before infection: 43 hours
  - Active user: 43 hours
  - Idle user: 0 hours
- After the infection: 41 hours
  - Active user: 41 hours
  - Idle user: 0 hours

4.3 Third Dataset

The third dataset consists of the same, only normal part as the second dataset, but a new malware. By restoring the clean state of the VM after the infection of the second dataset, it was possible to recover the computer to infect again. This third dataset

[2]https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-9-2/
[3]https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-6/

was infected with the malware called Simba, which belongs to the Simba botnet (MD5 eb7c74c66f801abde07e0d1a72cbec79). This dataset is completely available and can be downloaded from the dataset repository of the Malware Capture Facility Project[4].

Summary of the third dataset:

- Total Duration: 49.9 hours
- Before infection: 43 hours
  - Active user: 43 hours
  - Idle user: 0 hours
- After infection: 6.9 hours
  - Active user: 6.9 hours
  - Idle user: 0 hours

4.4 Assignment of Ground-Truth Labels

In this section, we describe our labeling process. For the evaluation of the performance, we need to have ground-truth labels. Our captures of traffic are split into two parts: before and after an infection. When we create profiles from the flows we know if it belongs to pre- or post-infection part.

We assign the normal label ($label = 1$) to all profiles which are created before the infection of the virtual machine. We create a virtual machine before each capture which ensures that it is clean and does not contain any malicious software. We put the anomaly label ($label = -1$) to all profiles which are created after the infection of the virtual machine. The reason we assign the anomaly labels this way is the assumption that everything after an infection worth detecting and that malware produces changes in behavior. We are aware that not all the attacks are anomalies, but this assumption helps us better evaluate our algorithm.

[4] https://mcfp.felk.cvut.cz/publicDatasets/CTU-Mixed-Capture-7/

Chapter 5

Experiments and Results

With the anomaly detection method designed and a good dataset generated it was possible to train and test our hypothesis by running experiments. There were five experiments in total, each one verifying a different algorithm with a different dataset. The goal of these experiments was to verify if our hypothesis was true. Our hypothesis was if *it is possible to detect the infected users and to have a small number of false positives by focusing on the high-level behaviors in time.* A side effect verification was to see if our method was capable of generalizing to malware which it has not seen during the training.

All our experiments were evaluated in two ways. First, we label each feature individually using the trained models. Then second way is to use a voting mechanism to decide if the analyzed profile was anomalous or not. The voting is described in 3.4.2.

A summary of the experiments follows:

- The first experiment uses one-class SVM trained on the normal part of the first dataset and validated on the mixed part of the first dataset.
- The second experiment uses one-class SVM trained on the normal part of the second dataset and verified on the mixed part of the second dataset (and the mixed part of the third dataset).
- The third experiment uses Local Outlier Factor trained on the normal part of the first dataset and validated on the mixed part of the first dataset.
- The fourth experiment uses Local Outlier Factor trained on the normal part of the second dataset and validated on the mixed part of the second dataset.
- The fifth experiment uses Isolation Forest trained on the normal part of the first dataset and validated on the mixed part of the first dataset.

All the experiments were run in docker container using Ubuntu 16.04, Python 3.6 and the following versions of libraries:

- *pandas*: 0.20.3
- *numpy*: 1.13.1
- $scikit-learn$: 0.19.0

5.1 The First Experiment

In the first experiment, we used a One-Class SVM algorithm, as it was described in Section 3.5.2. This algorithm was trained with the normal part of the first dataset 4.1, and it was validated on the mixed part of that same first dataset. The SVM algorithm used the Euclidean distance and the RBF kernel function.

To train the SVM we used a grid search over the following parameters:

- *gamma* parameter for RBF kernel: values in range $[10^{-9}, 10^{3}]$ with step 10^{2}.
- *nu* parameter for lower bound of fraction of support vectors: values in range $[0.01, 0.99]$ with step 0.01.

The training of the algorithm was performed by splitting the normal part of the first dataset into three sets: *train*, *validation* and *test*. This was done using the *train_test_split* function from the python library *sklearn*. To make the data split repeatable, we specify that the random seed is 42. The mixed part of the first dataset (that contains both normal and malicious traffic) was split into two sets: *validation* and *test*. Both were generated by randomly selecting half of the profiles in that dataset.

With the sets of data defined we trained a model for each of the twelve features in a profile (see 3.3). After we have trained all models, we select the best model by using the validation set. The best model was selected for each feature (see 3.7).

The following are the validation results for each of the twelve features in a profile:

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.0	0.389	1.0	0.560
clientDestinationPortNumberOfFlowsTCPEstablished	0.0	0.0	NaN	NaN
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.0	0.389	1.0	0.560
clientDestinationPortTotalPacketsTCPEstablished	0.009	**1.0**	0.947	0.973
clientDestinationPortNumberOfFlowsUDPEstablished	0.009	0.389	0.875	0.539
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesUDPNotEstablished	0.009	0.389	0.875	0.539
clientDestinationPortTotalBytesTCPEstablished	0.009	**1.0**	0.947	0.973
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.389	1.0	0.560
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.389	1.0	0.560

Table 5.1: Validation results for the first experiment. One-class SVM algorithm trained on the normal part of first dataset and tested on the mixed part of the first dataset. The validation set consists of some normal profiles and some malicious profiles.

The first analysis of these results is done according to each feature. Table 5.1 shows that one-class SVM achieved low FPR for each feature. It does not exceed 0.01 in any case, and for eight of the features, it is 0.0. The algorithm also has 1.0 TPR for five features. All of these five features are TCP features. It means that the malware actively uses TCP protocol

to communicate and this is very anomalous compared with the normal user. The interesting observation is that the feature *clientDestinationPortNumberOfFlowsTCPEstablished* has 0.0 TPR.

After running the validation using grid search on the parameters, we selected the best model using the methodology explained in Section 3.7. The winning parameters for these one-class SVM models were:

	gamma	*nu*
clientDestinationPortTotalBytesUDPEstablished	10^{-9}	0.01
clientDestinationPortNumberOfFlowsTCPEstablished	10^{-9}	0.35
clientDestinationPortNumberOfFlowsUDPNotEstablished	10^{-3}	0.02
clientDestinationPortTotalPacketsTCPEstablished	10^{-4}	0.01
clientDestinationPortNumberOfFlowsUDPEstablished	10^{-4}	0.01
clientDestinationPortTotalPacketsTCPNotEstablished	10^{-4}	0.01
clientDestinationPortTotalBytesUDPNotEstablished	10^{-9}	0.37
clientDestinationPortTotalBytesTCPEstablished	10^{-9}	0.35
clientDestinationPortTotalPacketsUDPNotEstablished	10^{-9}	0.14
clientDestinationPortNumberOfFlowsTCPNotEstablished	10^{-9}	0.52
clientDestinationPortTotalBytesTCPNotEstablished	10^{-9}	0.52
clientDestinationPortTotalPacketsUDPEstablished	10^{-9}	0.64

Table 5.2: The winning parameters for One-Class SVM models in the first experiment. These results were obtained using the validation set of the first dataset.

After the winner parameters were selected during the validation phase, we can test the model for its generalization power using the test set. Table 5.3 shows the results on the test set. The results are very similar to the validation set results, which means that there is a good generalization. This test set contains the same malware that was used in the validation set.

After testing the one-class SVM using our first method of evaluation, we now evaluate the results of applying a majority voting mechanism on the output generated for each feature to decide if the profile was anomalous or not. As in the previous testing, we tested the majority voting model on the test set from the first dataset. Our approach to a majority voting is described in section 3.4.2. The results are shown in Table 5.4.

Table 5.4 shows that the majority voting among the models produces a good result on the test set. The TPR of 0.444 or 44% may seem low, but this is the final result for all the profiles in time, with a 0 FPR and a 100% precision. These results mean that on average we will detect one anomaly out of three with probability 100%. It also means that if we use the five-minute time-windows to create profiles, we can raise the alarm after first 15 minutes after a malware becomes active. Considering that the evaluation is per profile, we believe that these results are very good in the area.

5.2 The Second Experiment

In the second experiment, we also used the One-Class SVM, as it was described in Section 3.5.2. This algorithm was trained with the normal part of the second dataset 4.2,

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.0	0.389	1.0	0.560
clientDestinationPortNumberOfFlowsTCPEstablished	0.0	0.0	NaN	NaN
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.0	0.389	1.0	0.560
clientDestinationPortTotalPacketsTCPEstablished	0.011	**1.0**	0.947	0.973
clientDestinationPortNumberOfFlowsUDPEstablished	0.011	**1.0**	0.947	0.973
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesUDPNotEstablished	0.011	0.389	0.875	0.539
clientDestinationPortTotalBytesTCPEstablished	0.011	**1.0**	0.947	0.973
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.389	1.0	0.560
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.389	1.0	0.560

Table 5.3: Test results for the first experiment. This is the one-class SVM algorithm trained on the normal part of the first dataset. The test of the algorithm was done on the test set for the first dataset. The test set contains normal profiles and profiles created during the infection with the malware.

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.444	1.0	0.615

Table 5.4: Test results for the first experiment using majority voting. We apply majority voting to the twelve results on each profile to get the final decision about a profile. The test set for the first dataset was used for the testing. The test set contains normal profiles and profiles created during the infection with the malware.

and it was validated on the mixed part of that same dataset. The SVM algorithm used the Euclidean distance and the RBF kernel function.

To train the SVM model we used a grid search over the following parameters:

- *gamma* parameter for RBF kernel: values in range $[10^{-9}, 10^{3}]$ with step 10^{2}
- *nu* parameter for lower bound of fraction of support vectors: values in range $[0.01, 0.99]$ with step 0.01

The training of the algorithm was performed by splitting the normal part of the dataset into three sets: *train*, *validation* and *test*. This was done using the *train_test_split* function from the python library *sklearn*. To make the data split repeatable, we specify that the random seed is 42. The mixed part of the second dataset (that contains both normal and malicious traffic) was split into two halves: the first half was added to the *validation* set and the second half was added to the *test* set. Both were generated by randomly selecting half of the profiles in that dataset.

With the sets of data defined we trained a model for each of the twelve features in a profile (see 3.3). After we have trained all models, we select the best model by using the validation set. The best model was selected for each feature (see 3.7).

After the validation is finished we select the winner model. The following are the validation results for each of the twelve features in a profile:

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.008	0.016	0.667	0.031
clientDestinationPortNumberOfFlowsTCPEstablished	0.008	**0.388**	0.980	0.556
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.008	0.039	0.833	0.075
clientDestinationPortTotalPacketsTCPEstablished	0.008	**0.364**	0.979	0.531
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	**0.418**	1.0	0.590
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesUDPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortTotalBytesTCPEstablished	0.0	**0.341**	1.0	0.509
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalPacketsUDPEstablished	0.008	**0.574**	1.0	0.729

Table 5.5: Results on the validation set for the second dataset. The validation set contains profiles created during the infection with the first type of malware.

The first analysis of the results is done with regard to each feature. Table 5.5 shows that the models achieved low FPR for each feature. It does not exceed 0.01 in any case, and for seven of the features, it is 0.0. However, TPR is lower than in the first experiment. This might be caused by the nature in which this malware communicates. The mixed capture of the second dataset was generated using DarkVNC malware. DarkVNC is used to silently control the computer of a victim, and it does not generate much additional traffic.

This is an example of how our technique might help better understand the communication details of malware. By analyzing results, the analyst could very fast see which protocols are used by malware, if it generates many connections and sends much information.

The winning parameters for the best One-Class SVM models were:

After the winner parameters were selected during the validation phase, we can test the model for its generalization power using the test set. Table 5.7 shows the results on the test set. This test set contains the same malware that was used in the validation set.

We wanted to test the generalization power of the models even further, and we used the mixed capture from the third dataset 4.3 to test the performance of the models. Since the mixed capture from the third dataset contains traffic from a different type of malware and it was not used during the selection process of models, this evaluation could be a good estimate for the real error. Table 5.8 contains results obtained for the mixed capture from the third dataset using the models trained on the second dataset:

These results are first analyzed per each feature. Table 5.8 shows that different features contribute to the detection of different types of malware. For example, on the one hand, the feature $clientDestinationPortNumberOfFlowsUDPNotEstablished$ contributes a lot to the detection of Simba malware and on the other hand the same feature does not contribute to the detection of DarkVNC (see Table 5.8).

	gamma	*nu*
clientDestinationPortTotalBytesUDPEstablished	10^{-9}	0.50
clientDestinationPortNumberOfFlowsTCPEstablished	10^{-3}	0.03
clientDestinationPortNumberOfFlowsUDPNotEstablished	10^{-4}	0.02
clientDestinationPortTotalPacketsTCPEstablished	10^{-3}	0.03
clientDestinationPortNumberOfFlowsUDPEstablished	10^{-4}	0.01
clientDestinationPortTotalPacketsTCPNotEstablished	10^{-9}	0.50
clientDestinationPortTotalBytesUDPNotEstablished	10^{-8}	0.49
clientDestinationPortTotalBytesTCPEstablished	10^{-2}	0.01
clientDestinationPortTotalPacketsUDPNotEstablished	10^{-9}	0.99
clientDestinationPortNumberOfFlowsTCPNotEstablished	10^{-7}	0.60
clientDestinationPortTotalBytesTCPNotEstablished	10^{-9}	0.50
clientDestinationPortTotalPacketsUDPEstablished	10^{-3}	0.01

Table 5.6: The winning parameters for One-Class SVM models in the second experiment. These parameters were obtained using the validation set of the second dataset.

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.011	0.016	0.667	0.031
clientDestinationPortNumberOfFlowsTCPEstablished	0.010	**0.388**	0.980	0.556
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.011	0.039	0.833	0.075
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.364	1.0	0.534
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	**0.418**	1.0	0.590
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesUDPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortTotalBytesTCPEstablished	0.0	**0.341**	1.0	0.509
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalPacketsUDPEstablished	0.010	**0.574**	0.987	0.726

Table 5.7: Results on the test set for the second dataset. The test set contains profiles created during the infection with the first type of malware.

After testing the one-class SVM using our first method of evaluation, we now evaluate the results of applying a majority voting mechanism on the output generated for each feature to decide if the profile was anomalous or not. As in the previous testing, we tested the majority voting model on the test set from the second dataset. Our approach to a majority voting is described in section 3.4.2. The results are shown in Table 5.4.

Table 5.9 shows that the majority voting among the models produces FPR of 0.0. However, the TPR is very low, which could be explained by the difficulty of detecting this particular malware. In the future we want to experiment with approaches other than majority voting, for example, we will train a neural network to summarize the outputs of the twelve models into one final result.

We also apply majority voting to classify the profiles which were generated during the

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.011	**0.390**	0.968	0.556
clientDestinationPortNumberOfFlowsTCPEstablished	0.011	**0.455**	0.972	0.620
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.011	**0.610**	0.979	0.752
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.0	NaN	nan
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	**0.636**	1.0	0.778
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.025	1.0	0.049
clientDestinationPortTotalBytesUDPNotEstablished	0.0	0.246	1.0	0.395
clientDestinationPortTotalBytesTCPEstablished	0.0	0.0	NaN	nan
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.247	1.0	0.396
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.026	1.0	0.051
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortTotalPacketsUDPEstablished	0.010	**0.597**	0.978	0.741

Table 5.8: Results on the test set for the third dataset. The test set contains profiles created during the infection with Simba malware. The models were created using the second dataset. *NaN* in the *Precision* column means that we do not have any true positives and we do not have any false positive.

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.031	1.0	0.060

Table 5.9: The second experiment. We apply majority voting to the twelve results to get the final decision about a profile. We show results on the test set for the second dataset and the first malicious part. The test set contains normal profiles and profiles created during the infection with the first malware.

infection with the Simba malware in the third dataset. The results are shown in Table 5.10

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.233	1.0	0.378

Table 5.10: The second experiment. We apply majority voting to the twelve results to get the final decision about a profile. We show results on the test set for the second dataset and the second malicious part. The test set contains normal profiles and profiles created during the infection with the second malware.

The TPR of 0.233 or 23% is lower than in the first experiment. However, this result means that the algorithm will detect one anomalous profile out of five with probability 100%. If we use five minutes time-windows to create profiles, the alarm will be raised during the first 25 minutes after a malware becomes active.

5.3 The Third Experiment

In the third experiment, we used the LOF algorithm, as it was described in Section 3.5.1. This algorithm was trained with the normal part of the first dataset 4.1, and it was validated

on the mixed part of that same first dataset. The LOF algorithm used the Euclidean distance to compute density estimation.

To train the LOF we used a grid search over the following parameters:

- k parameter for number of neighbours: values in range $[1 - 10]$ with a step 1
- *contam* parameter for contamination rate (the ratio of anomalies): values in range $[0.01, 0.1]$ with a step approximately 0.002

The training of the algorithm was performed by splitting the normal part of the dataset into three sets: *train*, *validation* and *test*. This was done using the *train_test_split* function from the python library *sklearn*. To make the data split repeatable, we specify that the random seed is 42. The mixed part of the first dataset (that contains both normal and malicious traffic) was split into two halves: the first half was added to the *validation* set and the second half was added to the *test* set. Both were generated by randomly selecting half of the profiles in that dataset.

With the sets of data defined we trained a model for each of the twelve features in a profile (see 3.3). After we have trained all models, we select the best model by using the validation set. The best model was selected for each feature (see 3.7).

After the validation is finished we select the winner model. The following are the validation results for each of the twelve features in a profile:

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.0	0.389	1.0	0.560
clientDestinationPortNumberOfFlowsTCPEstablished	0.0	0.056	1.0	0.106
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.0	0.333	1.0	0.500
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.056	1.0	0.106
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	0.0	NaN	NaN
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesUDPNotEstablished	0.011	0.333	0.857	0.480
clientDestinationPortTotalBytesTCPEstablished	0.0	**0.778**	1.0	0.875
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.278	1.0	0.435
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalBytesTCPNotEstablished	0.0	**1.0**	1.0	1.0
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.0	NaN	NaN

Table 5.11: The third experiment. We use Local Outlier Factor algorithm to detect anomalies. We show results on the validation set for the first dataset. The validation set contains normal profiles and profiles created during the infection with the malware.

The first analysis of these results is done according to each feature. Table 5.11 shows that LOF achieved low FPR for each feature. It exceeds 0.01 only for one profile and only by 0.001, and for other eleven of the features, it is 0.0. The algorithm also has 1.0 TPR for three features. All of these three features are TCP features. It means that the malware actively uses TCP protocol to communicate and this is very anomalous compared with the normal user. The feature $clientDestinationPortNumberOfFlowsTCPEstablished$ has very low TPR of 0.056 which is consistent with the result of the first experiment.

After running the validation using grid search on the parameters, we selected the best model using the methodology explained in Section 3.7. The winning parameters for these LOF models are shown in Table 5.12.

	k	$contam$
clientDestinationPortTotalBytesUDPEstablished	1	0.01
clientDestinationPortNumberOfFlowsTCPEstablished	6	0.01
clientDestinationPortNumberOfFlowsUDPNotEstablished	2	0.01
clientDestinationPortTotalPacketsTCPEstablished	1	0.01
clientDestinationPortNumberOfFlowsUDPEstablished	3	0.01
clientDestinationPortTotalPacketsTCPNotEstablished	3	0.01
clientDestinationPortTotalBytesUDPNotEstablished	1	0.01
clientDestinationPortTotalBytesTCPEstablished	1	0.01
clientDestinationPortTotalPacketsUDPNotEstablished	1	0.01
clientDestinationPortNumberOfFlowsTCPNotEstablished	3	0.01
clientDestinationPortTotalBytesTCPNotEstablished	4	0.01
clientDestinationPortTotalPacketsUDPEstablished	4	0.01

Table 5.12: The winning parameters for LOF models in the third experiment. These results were obtained using the validation set of the first dataset.

After the winner parameters were selected during the validation phase, we can test the model for its generalization power using the test set. Table 5.13 shows the results on the test set. The results for TPR are similar to the results obtained on the validation set, but the FPR is slightly higher for nine profile features. This test set contains the same malware that was used in the validation set.

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.011	0.389	0.875	0.539
clientDestinationPortNumberOfFlowsTCPEstablished	0.011	0.056	0.5	0.101
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.022	0.333	0.75	0.461
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.056	1.0	0.106
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	0.0	NaN	NaN
clientDestinationPortTotalPacketsTCPNotEstablished	0.022	**1.0**	0.9	0.947
clientDestinationPortTotalBytesUDPNotEstablished	0.011	0.333	0.857	0.480
clientDestinationPortTotalBytesTCPEstablished	0.022	**0.778**	0.875	0.824
clientDestinationPortTotalPacketsUDPNotEstablished	0.022	0.278	0.714	0.400
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.022	**1.0**	0.90	0.947
clientDestinationPortTotalBytesTCPNotEstablished	0.022	**1.0**	0.90	0.947
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.022	1.0	0.043

Table 5.13: The third experiment. We use Local Outlier Factor algorithm to detect anomalies. We show results on the test set for the first dataset. The test set contains normal profiles and profiles created during the infection with the malware.

After testing the LOF using our first method of evaluation, we now evaluate the results of applying a majority voting mechanism on the output generated for each feature to decide

if the profile was anomalous or not. As in the previous testing, we tested the majority voting model on the test set from the first dataset. Our approach to a majority voting is described in section 3.4.2. The results are shown in Table 5.14.

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.333	1.0	0.496

Table 5.14: The third experiment. We apply majority voting to the twelve results to get the final decision about a profile. We show results on the test set for the first dataset. The test set contains normal profiles and profiles created during the infection with the malware.

Table 5.4 shows that the majority voting among the models produces a good result on the test set. The TPR of 0.333 or 33.3% may seem low, but this is the final result for all the profiles in time, with a 0 FPR and a 100% precision. These results mean that on average we will detect one anomaly out of three with probability 99.9%. It also means that if we use the five-minute time-windows to create profiles, we can raise the alarm during the first 15 minutes after a malware becomes active. Considering that the evaluation is per profile, we believe that these results are very good in the area.

5.4 The Fourth Experiment

In the fourth experiment, we used the LOF algorithm, as it was described in Section 3.5.1. This algorithm was trained with the normal part of the second dataset 4.1, and it was validated on the mixed part of that same second dataset. The LOF algorithm used the Euclidean distance to compute density estimation.

To train the LOF we used a grid search over the following parameters:

- k parameter for number of neighbours: values in range $[1 - 10]$ with a step 1
- *contam* parameter for contamination rate (the ratio of anomalies): values in range $[0.01, 0.1]$ with a step approximately 0.002

The training of the algorithm was performed by splitting the normal part of the dataset into three sets: *train*, *validation* and *test*. This was done using the *train_test_split* function from the python library *sklearn*. To make the data split repeatable, we specify that the random seed is 42. The mixed part of the second dataset (that contains both normal and malicious traffic) was split into two halves: the first half was added to the *validation* set and the second half was added to the *test* set. Both were generated by randomly selecting half of the profiles in that dataset.

With the sets of data defined we trained a model for each of the twelve features in a profile (see 3.3). After we have trained all models, we select the best model by using the validation set. The best model was selected for each feature (see 3.7).

After the validation is finished we select the winner model. The following are the validation results for each of the twelve features in a profile:

The first analysis of the results is done with regard to each feature. Table 5.15 shows that the models achieved low FPR for each feature. It does not exceed 0.01 in any case, and

Feature Name	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.011	0.016	0.667	0.031
clientDestinationPortNumberOfFlowsTCPEstablished	0.0	0.07	1.0	0.131
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.0	0.02	1.0	0.039
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.054	1.0	0.102
clientDestinationPortNumberOfFlowsUDPEstablished	0.0	0.008	1.0	0.016
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesUDPNotEstablished	0.011	0.016	0.667	0.031
clientDestinationPortTotalBytesTCPEstablished	0.0	**0.470**	0.639	
clientDestinationPortTotalPacketsUDPNotEstablished	0.0	0.008	1.0	0.031
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.016	1.0	0.031
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.016	1.0	NaN
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.0	NaN	NaN

Table 5.15: Fourth experiment where we use LOF algorithm. Results on the validation set for the second dataset. The validation set contains profiles created during the infection with DarkVNC malware. *NaN* in the *Precision* column means that we do not have any true positives and we do not have any false positive.

for seven of the features, it is 0.0. However, TPR is lower than in the first experiment. This might be caused by the nature in which this malware communicates. The mixed capture of the second dataset was generated using DarkVNC malware. DarkVNC is used to silently control the computer of a victim, and it does not generate much additional traffic.

After running the validation using grid search on the parameters, we selected the best model using the methodology explained in Section 3.7. The winning parameters for these LOF models are shown in Table 5.16.

	k	*contam*
clientDestinationPortTotalBytesUDPEstablished	3	0.01
clientDestinationPortNumberOfFlowsTCPEstablished	2	0.02
clientDestinationPortNumberOfFlowsUDPNotEstablished	1	0.01
clientDestinationPortTotalPacketsTCPEstablished	8	0.02
clientDestinationPortNumberOfFlowsUDPEstablished	5	0.01
clientDestinationPortTotalPacketsTCPNotEstablished	6	0.01
clientDestinationPortTotalBytesUDPNotEstablished	1	0.02
clientDestinationPortTotalBytesTCPEstablished	5	0.03
clientDestinationPortTotalPacketsUDPNotEstablished	3	0.01
clientDestinationPortNumberOfFlowsTCPNotEstablished	7	0.01
clientDestinationPortTotalBytesTCPNotEstablished	3	0.01
clientDestinationPortTotalPacketsUDPEstablished	2	0.01

Table 5.16: The winning parameters for LOF models in the fourth experiment. These results were obtained using the validation set of the second dataset.

After the winner parameters were selected during the validation phase, we can test the model for its generalization power using the test set. Table 5.17 shows the results on the

test set. The results for TPR are similar to the results obtained on the validation set, but the FPR is slightly higher for five profile features. This test set contains the same malware that was used in the validation set.

	FPR	TPR	Precision	F1	TNR	Accurac
clientDestinationPortTotalBytesUDPEstablished	0.0	0.016	1.0	0.031		
clientDestinationPortNumberOfFlowsTCPEstablished	0.032	0.078	0.769	0.142		
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.021	0.023	0.6	0.044		
clientDestinationPortTotalPacketsTCPEstablished	0.0	0.061	1.0	0.115		
clientDestinationPortNumberOfFlowsUDPEstablished	0.011	0.007	0.5	0.014		
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.016	1.0	0.031		
clientDestinationPortTotalBytesUDPNotEstablished	0.011	0.016	0.667	0.031		
clientDestinationPortTotalBytesTCPEstablished	0.011	**0.472**	0.983	0.638		
clientDestinationPortTotalPacketsUDPNotEstablished	0.011	0.007	0.5	0.014		
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.016	1.0	0.031		
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.016	1.0	0.031		
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.0	NaN	NaN		

Table 5.17: Fourth experiment where we use LOF algorithm. Results on the test set for the second dataset. The test set contains profiles created during the infection with DarkVNC malware. *NaN* in the *Precision* column means that we do not have any true positives and we do not have any false positive.

We wanted to test the generalization power of the models even further, and we used the mixed capture from the third dataset 4.3 to test the performance of the models. Since this mixed capture contains traffic from a different type of malware and it was not used in the during the selection process of models, this evaluation could be a good estimate for the real error. Table 5.18 contains results obtained for the mixed capture from the third dataset using the models trained on the second dataset:

After testing the LOF using our first method of evaluation, we now evaluate the results of applying a majority voting mechanism on the output generated for each feature to decide if the profile was anomalous or not. As in the previous testing, we tested the majority voting model on the test set from the second dataset. Our approach to a majority voting is described in section 3.4.2. The results are shown in Table 5.19.

We also apply majority voting to classify the profiles which were generated during the infection with the Simba malware in the third dataset. The results are shown in Table 5.20

As in the second experiment results of the majority, voting are not satisfactory, and we are going to address this problem in the future work.

5.5 The Fifth Experiment

In this experiment, we have tried to use Isolation Forest on our data. The results were surprising because we were not able to train a model with any true positive detection. The Isolation Forest is shown to out-perform many algorithms including Local Outlier Factor and One-Class SVM [32], [33].

	FPR	TPR	Precision	F1
clientDestinationPortTotalBytesUDPEstablished	0.0	0.026	1.0	0.051
clientDestinationPortNumberOfFlowsTCPEstablished	0.031	**0.519**	0.930	0.666
clientDestinationPortNumberOfFlowsUDPNotEstablished	0.021	0.0	0.0	NaN
clientDestinationPortTotalPacketsTCPEstablished	0.0	**0.948**	1.0	0.973
clientDestinationPortNumberOfFlowsUDPEstablished	0.015	0.0	0.0	NaN
clientDestinationPortTotalPacketsTCPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortTotalBytesUDPNotEstablished	0.015	0.0	0.0	NaN
clientDestinationPortTotalBytesTCPEstablished	0.015	**0.766**	0.983	0.861
clientDestinationPortTotalPacketsUDPNotEstablished	0.015	0.013	0.50	0.025
clientDestinationPortNumberOfFlowsTCPNotEstablished	0.0	0.026	1.0	0.051
clientDestinationPortTotalBytesTCPNotEstablished	0.0	0.039	1.0	0.075
clientDestinationPortTotalPacketsUDPEstablished	0.0	0.0	NaN	NaN

Table 5.18: Fourth experiment where we use LOF algorithm. Results on the test set for the third dataset. The test set contains profiles created during the infection with Simba malware. *NaN* in the *Precision* column means that we do not have any true positives and we do not have any false positive.

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.0	NaN	NaN

Table 5.19: The fourth experiment. We apply majority voting to the twelve results to get the final decision about a profile. We show results on the test set for the second dataset. The test set contains normal profiles and profiles created during the infection with DarkVNC malware.

Feature Name	FPR	TPR	Precision	F1
Majority voting	0.0	0.0	NaN	NaN

Table 5.20: The fourth experiment. We apply majority voting to the twelve results to get the final decision about a profile. We show results on the test set for the third dataset. The test set contains normal profiles and profiles created during the infection with Simba malware.

It could be caused by the nature of anomalies in our dataset. As we described in section 3.3 each feature of a profile is represented by a vector. When we detect anomalies among one feature of a profile, it means that we detect anomalous vectors.

Anomalies in our datasets are reflected in vector components which are irrelevant during the training. They are irrelevant because normal data have only 0 values in these components and these components do not contribute to model training. When we run interference on a testing profile, the model does not use these components to isolate the profile faster. As a result, the profile is labeled as normal by the model.

Another reason for the poor results could be a mistake in the way we have trained the Isolation Forest model. We will investigate more closely this issue in the future.

5.6 Analysis of Results

The experiments proposed in our analysis try to find how the anomaly detection algorithms may work in a realistic setup where a normal user is infected at the same time that they continue to work. In this sense, this is the first research work in the security area that publishes results using a mixed dataset of real normal actions and real malware actions.

Our experiments were designed, so they were trained with real users and tested with real malware. They were also designed to detect if a profile is anomalous and not the IP address of a user. Finally, the time-window of the profiles is five minutes, which also may affect the algorithm if changed.

Using the one-class SVM, it was possible to obtain a good TPR of 44% with 0% FPR. Since these results are per five-minutes profile, it means that the algorithm will, out of **three** anomalous profiles, detect **one** with 100% probability (or 2 out of 5). It also means that there will be a detection at most every **15** minutes. Moreover, these results are based on a majority voting mechanism, which is not considered to be the best way of improving the results. In particular, there are some individual features that may have better results under specific circumstances. In the first experiment, six out of twelve features reach 100% TPR with 0% FPR.

In the case of the Local Outlier Factor algorithm, the results are very similar, with an average of 33% TPR detection and 0% FPR using majority voting. This means that LOF can also detect **one** profile correctly out of **three** anomalous profiles with 100% probability. The detection time is similar to one-class SVM: one anomalous profile every 15 minutes. If the detection of profiles is not done with majority voting, then LOF can reach 77.8% TPR with 2% FPR by using the feature called *clientDestinationPortTotalBytesTCPEstablished*.

LOF also had very good results on the third experiment, that used the first dataset. In this case the algorithm can have a 100% TPR, but only at the expense of a 2% FPR. The good part was that these results were obtained with three different features *clientDestinationPortNumberOfFlowsTCPNotEstablished*, *clientDestinationPortTotalBytesTCPNotEstablished*, *clientDestinationPortTotalPacketsTCPNotEstablished*.

The LOF algorithm also had good results on the fourth experiment, which used the second dataset. In this case, the algorithm achieved 47.2% TPR with a 1.1% FPR. These results were obtained also with the feature *clientDestinationPortTotalBytesTCPEstablished*. In the forth experiment, with LOF on the third dataset, the algorithm obtained a very good 94.8% TPR with 0% FPR with the feature *clientDestinationPortTotalPacketsTCPEstablished*.

Chapter 6

Conclusion

TThe detection of attacks and malware using anomaly detection techniques is a very well known topic in the area. This thesis proposes a new perspective on the problem by analyzing the behavioral features of users in the network and by applying a high-level detection on features in time. By using a completely novel dataset and known anomaly detection methods, we were able to obtain promising results.

The contributions of this thesis are the following:

- A new approach for anomaly detection by defining profiles of the user in fixed time-windows.
- A novel real datasets in which normal user actions were mixed with malware actions. This dataset was published online.

To the best of our knowledge, this thesis presented the first anomaly detection method where users were profiled using their network traffic to create behavioral features, and these features were analyzed from different perspectives. The presented anomaly detection method was based on a high-level view of the traffic by generating behavioral profiles of the activity of the users inside fixed time-windows. The new profiles of the users were compared to the past profiles to classify them as anomalous or normal. Our approach is different from other anomaly detection algorithms because we model user behavior by combining detailed features that describe all actions of the user from different perspectives.

We classified profiles by comparing each feature with the same feature in other past profiles. The decision on whether there was an anomaly or not was taken for *each feature* and then the final label of the profile was decided by majority voting. The anomalies along each feature were found using well-known algorithms such as Local Outlier Factor and One-Class SVM.

To evaluate our approach we needed the data from real normal users and the data from real infection. The produced datasets were unique because they contain real malware activities at the same time that the real normal user was using the computer. The datasets were made open to the public and future research.

The datasets were used to test and evaluate how our approach would detect different types of malware. The multiple experiments show that our approach could help in reducing

the number of false positive alarms while at the same time being effective in detecting true anomalies. In multiple experiments, we were able to detect one out of three anomalous profiles with 100% success, and we had 0 false alarms.

Even though we are satisfied with the results, there is much research to be done. One of the problems we want to work in the future is to solve how to combine twelve different results in order to get the final decision. We would like to experiment with training another model which would accept the output of the twelve models described in this work and give the final decision. If we provide enough data during the training, it may be possible that such a model could help find some non-obvious relationship between data.

Also, our approach lacks the very important process of updating the model of normal behavior in order to adapt to new traffic. Since the behavior of a user might change with time, we need to find a way how to keep our models up-to-date. For this, we need to create larger datasets which would cover an extended period in different situations.

A very promising research direction we may work on is the use of anomaly detection methods to Internet Of Things (*IOT*) devices such as IP cameras, thermostats, printers, etc. The number of attacks on *IOT* devices is growing as well as the amount of malware designed to target these *IOT* devices. Our approach could be useful in protecting *IOT* devices because the traffic from these devices is far more stable than a human computer, and therefore it changes less often than the traffic generated by a human. As a result, the profiles may have less diversity, and the results from an anomaly detection method may be easier to obtain.

Bibliography

[1] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman, "An overview of issues in testing intrusion detection systems," 2003.

[2] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." in *Lisa*, vol. 99, no. 1, 1999, pp. 229–238.

[3] V. Paxson, R. Sommer, S. Hall, C. Kreibich, J. Barlow, G. Clark, G. Maier, J. Siwek, A. Slagell, D. Thayer *et al.*, "The bro network security monitor," 2012.

[4] W. Wang, X. Zhang, S. Gombault, and S. J. Knapskog, "Attribute normalization in network intrusion detection," in *Pervasive systems, algorithms, and networks (ISPAN), 2009 10th international symposium on.* IEEE, 2009, pp. 448–453.

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[6] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[7] L. Ertoz, E. Eilertson, A. Lazarevic, P.-N. Tan, V. Kumar, J. Srivastava, and P. Dokas, "Minds-minnesota intrusion detection system," *Next generation data mining*, pp. 199–218, 2004.

[8] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[9] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, P. Celeda, and V. Krmicek, "Camnep: An intrusion detection system for high-speed networks," *Progress in informatics*, vol. 5, no. 5, pp. 65–74, 2008.

[10] M. Zhang, B. Xu, and J. Gong, "An anomaly detection model based on one-class svm to detect network intrusions," in *Mobile Ad-hoc and Sensor Networks (MSN), 2015 11th International Conference on.* IEEE, 2015, pp. 102–107.

[11] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Reducing unwanted traffic in a backbone network." *SRUTI*, vol. 5, pp. 9–15, 2005.

[12] M. Grill, "Combining network anomaly detectors," 2016.

[13] M. V. Mahoney, "Network traffic anomaly detection based on packet bytes," in *Proceedings of the 2003 ACM symposium on Applied computing*. ACM, 2003, pp. 346–350.

[14] CISCO, "Creating profiles," <https://www.cisco.com/c/en/us/td/docs/security/firesight/541/user-guide/FireSIGHT-System-UserGuide-v5401/Traffic-Profiles.pdf>, unknown.

[15] G. Pannell and H. Ashman, "Anomaly detection over user profiles for intrusion detection," 2010.

[16] D. Kubeša, "parse-binet," <https://github.com/Kobtul/parse-binet/tree/thesis>, 2018.

[17] C. I. NetFlow, "Introduction to cisco ios netflow a technical overview," *White Paper, Last updated: February*, 2006.

[18] M. Rehak, M. Pechoucek, K. Bartos, M. Grill, and P. Celeda, "Network intrusion detection by means of community of trusting agents," in *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. IEEE Computer Society, 2007, pp. 498–504.

[19] C. Bullard, "Argus, the network audit record generation and utilization system," 2005.

[20] J. Postel, "Transmission control protocol," 1981.

[21] K. Pearson, "Principal components analysis," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 6, no. 2, p. 559, 1901.

[22] T. Sipola, A. Juvonen, and J. Lehtonen, "Dimensionality reduction framework for detecting anomalies from network logs," *Engineering Intelligent Systems*, vol. 20, no. 1/2, 2012.

[23] T. Huang, H. Sethu, and N. Kandasamy, "A new approach to dimensionality reduction for anomaly detection in data traffic," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 651–665, 2016.

[24] S. Dreiseitl, M. Osl, C. Scheibböck, and M. Binder, "Outlier detection with one-class svms: an application to melanoma prognosis," in *AMIA Annual Symposium Proceedings*, vol. 2010. American Medical Informatics Association, 2010, p. 172.

[25] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 2008, pp. 413–422.

[26] D. M. Hawkins, *Identification of outliers*. Springer, 1980, vol. 11.

[27] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM sigmod record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[28] ——, "Optics-of: Identifying local outliers," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 1999, pp. 262–270.

[29] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[30] M. H. S. C. S. Singla and Y. Shen, "Kernel selection and dimensionality reduction in svm classification of autism spectrum disorders."

[31] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[32] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "On detecting clustered anomalies using sciforest," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 274–290.

[33] A. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, "A meta-analysis of the anomaly detection problem," *arXiv preprint arXiv:1503.01158*, 2015.

Appendix A

Obsah přiloženého CD

- CD
  - `README.TXT` ..... The file with contents of attached CD
  - `text` ..... The folder with the text of the diploma thesis
  - `source` ..... The folder with the source code
    - `AnomalyDetection` ..... The folder with anomaly detection notebooks
      - `OCSVM_Models` ..... The folder with One-Class SVM models
      - `Results` ..... The folder with experiment results
    - `parse-binet` ..... parse-binet GitHub repository [16]