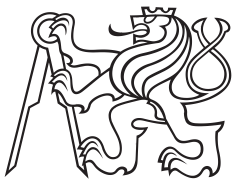
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Detection of HTTPS Malware Traffic

František Střasák
Open Informatics, Computer and Information Science

May 2017
Supervisor: Ing. Sebastián García, Ph.D.

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: František S t ř a s á k

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Detection of HTTPS Malware Traffic

Guidelines:

1. Review the state-of-the art methods for malware network behaviors analyses with special attention to HTTPs traffic detection techniques.
2. Propose and implement a detection method to identify HTTPs malware traffic. Malware using HTTPs is more difficult to detect because the content is encrypted. The features used mostly are those derived from SSL protocol and X.509 certificates.
3. Experimentally evaluate the proposed solution on real malware and normal datasets.
4. Critically analyze the results and propose further extensions of the solution with respect to its applicability in real time malware detection when part of the information is unavailable.

Bibliography/Sources:

[1] Anderson Blake, McGrew David - Identifying Encrypted Malware Traffic with Contextual Flow Data - Vienna, Austria October 28, 2016
[2] Sommer Robin, Paxson Vern - Outside the Closed World: On Using Machine Learning for Network Intrusion Detection - IEEE Symposium on Security and Privacy 2010
[3] Muehlstein Jonathan, Zion Yehonatan, Bahumi Maor, Kirshenboim Itay, Dubin Ran, Dvir Amit, Pele Ofir – Analyzing HTTPS Encrypted Traffic to Identify User Operating System, Browser and Application - Ariel University, Ben-Gurion University of the Negev, 2016
[4] Lokoč Jakub, Kohout Jan, Čech Přemysl, Skopal Tomáš, Pevný Tomáš - k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach - Faculty of Mathematics and Physics Charles University in Prague, Department of Computer Science and Engineering, FEE Czech Technical University in Prague, Cisco Systems, Inc., Cognitive Research Center in Prague, Prague 2016
[5] Garcia, S. (2016). Stratosphere Project. Retrieved January 24, 2016, from https://stratosphereips.org

Bachelor Project Supervisor: Ing. Sebastián García, Ph.D.

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

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 9, 2017

Acknowledgement / Declaration

I would like to express my sincere gratitude to my advisor Ing. Sebastián García, Ph.D. for his time, advice and experience that he dedicated to my project and for creating friendly working environment. Also, I would like to thank my parents for their support during my studies.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date 25.5. 2017

.....

Abstrakt / Abstract

V posledních letech lze zaznamenat nárůst malware, kteří ke své komunikaci používají HTTPS protokol. Tato situace s sebou přináší pro bezpečnostní analytiky řadu nových výzev, protože přenos je šifrován a je těžko rozlišitelný od běžné síťové komunikace. Z tohoto důvodu je potřeba najít nové metody pro detekci malware bez nutnosti dešifrovat síťovou komunikaci. Metoda, která nepotřebuje k detekci malware dešifrovat síťovou komunikaci, je levnější (protože není potřeba žádný dešifrovací přerušovač sítě), rychlejší a zajišťuje uchování soukromí, což je podstata šifrované komunikace v HTTPS. Cílem této práce je detekovat malware v síťové komunikaci vytvořením nových parametrů pro strojové učení a použitím dat, které zpracovává program Bro IDS. Jelikož není lehké získat data pro takovýto výzkum, použili jsme datasety, které jsou součástí projektu Stratosphere, a některé další jsme si vytvořili sami. Základní jednotkou pro náš datový model jsou informace, které lze získat z šifrované komunikace, jsou to flow, SSL data a x509 certifikáty, které generuje Bro program. Všechna tato data, jsou získána bez nutnosti jakéhokoliv dešifrování. Pro rozpoznání malware komunikace používáme několik algoritmů pro strojové učení, jako např.: Neuronové sítě, XGBoost a Random Forest. Výsledky výzkumu ukazují, že chování malware v síťové komunikaci se liší od běžné komunikace a že naší metodou jsme schopni detekovat malware s přesností až 96.64%.

Klíčová slova: Detekce Malware, Strojové učení, HTTPS, Síťová analýza, Umělá inteligence

Překlad titulu: Detekce Malware v HTTPS komunikaci

In the last years there has been an increase in the amount of malware using HTTPS traffic for their communications. This situation pose a challenge for the security analysts because the traffic is encrypted and because it mostly looks like normal traffic. Therefore, there is a need to discover new features and methods to detect malware without decrypting the traffic. A detection method that does not need to unencrypt the traffic is cheaper (because no traffic interceptor is needed), faster and private, respecting the original idea of HTTPS. The goal of this thesis is to detect HTTPS malware connections by extracting new features and using data from the Bro IDS program. Since the data for the research is hard to come by, we used data from the Stratosphere project and we created, by hand, our own datasets. Our unit of analysis is an aggregation of all the information that is possible to obtain without decrypting the data. We group together flows, SSL data and X.509 certificates data as they are generated by Bro. To classify the HTTPS malware traffic we used several algorithms, such as Neural Networks, XGBoost and Random Forest. Our results show that the HTTPS malware behaviour is distinct from normal HTTPS behaviour and that our methods are able to separate them with an accuracy of at least 96.64%.

Keywords: Malware detection, Machine Learning, HTTPS, Network Analysis, Artificial Intelligence

Contents

Tables /

Chapter 1
Introduction

The HTTPS protocol, also called HTTP Secure or Hypertext Transfer Protocol over SSL, is the standard for secure communications on the Internet and is dominantly used in any computer network. HTTPS is basically the encryption of Hypertext Transfer Protocol with either Transport Layer Security (TLS) or Secure Sockets Layer (SSL). The role of TLS/SSL in HTTPS is to encrypt the content of HTTP. Without encryption, the communication can be read by anybody that manages to see the packets between clients and servers.

According to a Google report from April 2017 [1] the usage of HTTPS is still growing. The report shows that desktop users load more than half of the web pages over HTTPS and that they spend two-thirds of their time on HTTPS pages. Windows users using Chrome browser load almost 60% of visited websites over HTTPS and Mac users using Chrome browser load almost 72% of visited websites over HTTPS.

With this increasing amount of encrypted network traffic in all internet, malware has also started to use the HTTPS to secure its own communication. In 2016 a report from Cisco [2] observed data that despite the fact that the majority of malware traffic is still using unencrypted HTTP there was a steady 10-12% of malicious communications using HTTPS.

The detection of HTTPS malware traffic is difficult and complicated because the encryption interferes with the efficacy of classical detection techniques. An increasingly common solution for dealing with HTTPS traffic in companies is to install HTTPS interceptor proxies. These hardware servers can open and inspect the HTTPS traffic of the employees by installing a special certificate in their computers. The HTTPS interceptor is placed between the client and the server, where the encrypted traffic is decrypted, scanned for malicious software, encrypted again and sent to the destination IP. This approach allows for classic detection methods to be used for detecting unencrypted malware traffic. The problems of using an interceptor are that it is expensive, computationally demanding, and that it does not respect the original idea of HTTPS which is to have private and secure communications.

We hypothesise that the detection of malware HTTPS traffic is possible with good accuracy without decrypting the traffic. Such a solution would be very important, because no HTTPS traffic interceptor would be needed, the privacy and security of communications would be respected and the detections would be faster. Also, such solution could be used together with some HTTPS traffic interceptor as the first detection layer on network traffic and if any traffic would be suspicious then the HTTPS traffic interceptor will be used for decryption.

This thesis proposes first to discover new features of HTTPS traffic, and then to apply methods to detect malware without decrypting the HTTPS traffic. The features for machine learning algorithms were created and extracted from data logs generated by the Bro IDS [3], that is able to process pcap files. Bro offers information about connections, SSL handshakes and X.509 certificates. These three types of data from

Bro give us enough information to create the powerful features and machine learning algorithms to detect the malicious HTTPS traffic with good accuracy.

The features are divided into three groups: connection features, SSL features and certificate features. The connection features describe common behavior of the traffic that is not associated with the encryption. The SSL features describe the SSL handshake and information about the encrypted traffic, the certificate features describe information about the certificate that the webser give us during the SSL handshake.

A data model for machine learning algorithms is a matrix where columns are feature values and rows are Connection 4-tuples. The Connection 4-tuple aggregates the records from Bro logs which share the same source IP, destination IP, destination Port, and protocol. Therefore, each Connection 4-tuple summarizes the behavior of the malware while connecting to the same C&C server. Such aggregation proved paramount for the success of our method. Each Connection 4-tuple contains 28 features that are used by the machine learning algorithms implemented in this thesis: XGBoost, SVM, Neural Networks and RandomForest.

A core part of our research was the production and selection of correct datasets. We used 13 datasets from the CTU-13 malware dataset [4], 59 malware datasets from the Stratosphere Malware Capture Facility Project (done by Maria Jose Erquiaga) [5] and I produced 13 normal datasets by hand. Each dataset was processed to extract the Bro files from the original pcap files. Afterwards, each dataset was labeled using our expert knowledge. The amount of malware and normal traffic in our entire dataset is balanced. Our data model from the entire dataset contains 17,384 Connection 4-tuples, where 8,132 of them are malware and 9,252 of them are normal. For the final experiments 20% from this data model was taken as a test data.

Our experiments show that the features and machine learning algorithms selected are able to detect the malicious HTTPS traffic with good accuracy. The best results from the final experiments on the test data are on accuracy: XGBoost 96.6%, Random Forest 94.1% and Neural Networks 79.9%.

Motivated by our good results, this thesis has a small extension of our methods, where we tried to detect only the malware certificates in our dataset by looking at the encrypted data. By analyzing our 3,246 normal certificates and 1,653 malware certificates, we obtained a detection accuracy of $\sim$73%.

All python scripts that are used for this thesis are stored in github [6] such as: computing statisics of dataset, labeling the dataset, generating the Connection 4-tuples, feature extraction, feature evaluation and machine learning algorithms. As python interpreter is used Python 2.7.13.

The remainder of this thesis is organized as follows: Chapter 2 reviews the related work, Chapter 3 details datasets and Chapter 4 details our feature extraction and creating data model. Chapter 5 details selected machine learning algorithms, Chapter 6 presents detection results and Chapter 7 presents our small experiment about certificate detection. Finally, Chapter 8 reviews our results, and we conclude in Chapter 9.

Chapter 2
Related work

To the best of our knowledge, the state of the art for using machine learning algorithms to detect malware HTTPS traffic is limited. Many research papers use decryption to detect malware HTTPS traffic. Then their features are mostly gathered from the unencrypted TLS/SSL handshake messages and from the certificates.

The research work "Deciphering Malware's use of TLS [without Decryption]" by Blake Anderson et al. [7] tries to detect the malware HTTPS traffic and malware families without decryption however their features are gathered from unencrypted TLS handshake messages and that's what our work is different from theirs because our features are gathered without decryption.

The work "Identifying Encrypted Malware Traffic with Contextual Flow Data" of Blake Anderson et al. [8] uses TLS flow, DNS flow, HTTP headers and TLS unencrypted header information to detect malware HTTPS traffic without decryption. Their final accuracy is 99.993% with 0% false discovery rate. Their work differs from ours in the use of DNS flows and also in the use of unencrypted TLS handshake messages. The DNS flows could be interesting idea for future work in our research.

The work "k-NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach" of Lokoč et al. [9] detects malware in HTTPS traffic using k-NN classification. They present the efficiency of metric indexing for approximate k-NN search over dataset of sparse high-dimensional descriptors of network traffic that reduce false positive rate.

Compared with other research works our method is unique and novel because is based on Connection 4-tuples and unencrypted Bro data. Also our datasets and python scripts for this research are published so the research community is able to reproduce the same experiments that are mentioned here and verify the credibility of results.

Chapter 3
Datasets

An essential and important part of our research was the production and selection of correct datasets, because the credibility of results is tied with the data. All dataset is published and contains only real malware and normal traffics consisting of 3 parts:

- CTU-13 dataset [4] — The CTU-13 dataset was captured in the CTU University, Czech Republic, in 2011. The CTU-13 dataset consists of a group of 13 different malware captures done in a real network environment. The captures include malware, normal and background traffic, but we just use Normal and Malware. The normal part of the dataset was captured using a Windows 7 running on virtual machine. The normal traffic uses unknown web browsers and the malware probably uses its own libraries to communicate with the Internet. Authors of this datasets are Sebastian Garcia, Martin Grill, Honza Stiborek and Alejandro Zunino with An Empirical Comparison of Botnet Detection Methods project.
- MCFP dataset [5] — There are datasets from the Stratosphere Malware Capture Facility Project done by Maria Jose Erquiaga. She generated these captures by selecting malware that uses HTTPS. This dataset consists of 53 malware captures and 6 normal captures. The capture of normal data was done using Internet Explorer on Windows 7 and Google Chrome on Linux Debian running on virtual machines between the years 2015 and 2017.
- My own normal dataset — Because of the lack of normal data we had to create more normal captures. The approach was to browse normal websites, such as facebook, twitter, gmail and dozens more. On most of them we obtained accounts and interacted for some time. We also used a list of websites from Moz.com [10] that contains the top 500 registered domains and first 700 websites from quantcast.com [11] which are most visited websites from people in the United States. This part of dataset contains 13 normal captures. The capturing was managed by Mozilla browser on Windows 7 running on virtual machine and by Iceweasel browser on Kali Linux in April 2017.

From now on, all these different captures will be referenced completely as the 'dataset'.

3.1 Dataset structure and Bro IDS

Each capture in the dataset contains a pcap file of the traffic containing captured packets, Bro logs generated from the pcap file by the Bro IDS and readme file describing information about IP addresses, time of capturing, type of malware or visited websites and timeline of malware behavior.

The Bro IDS [3] is an open-source network traffic analyzer. It is mainly a security monitor system but it supports a wide range of traffic analysis tasks such as the log files that record a network activity from the pcap file. These logs describe the captured traffic and contain essential information for our feature extraction. Bro generates many log files, among the most common are:

- conn.log — TCP/UDP/ICMP connections
- dpd.log — A summary of protocols encountered on non-standard ports.
- dns.log — DNS queries along with their responses
- ftp.log — A log of FTP session-level activity.
- files.log — Summaries of files transferred over the network. This information is aggregated from different protocols, including HTTP, FTP, and SMTP.
- http.log — A summary of all HTTP requests with their replies.
- smtp.log — A summary of SMTP activity.
- ssl.log — A record of SSL sessions, including certificates being used.
- x509.log — X.509 certificate info

Our method only uses three files: conn.log, ssl.log and x509.log. They provide enough information about HTTPS traffic. In summary, for each file we obtain the following information:

1. Connection record — one line in conn.log file
   - Each line aggregates a group of packets and describes a connection between two endpoints. The connection records contain information about IP addresses, ports, protocols, states of connections, numbers of packets, labels and etc.
2. SSL record — one line in ssl.log file
   - They describe SSL/TLS handshakes and encryption establishment processes. There are versions of SSL/TLS, ciphers used, server names, certificate path, subjects, issuers and much more.
3. Certificate record — one line in x509.log
   - Each line in the x509.log is a certificate record describing certificate information such as certificate serial numbers, common names, time validities, subjects, signature algorithms, key lengths in bits, etc.

These terms are important because will be used for in next definition.

Must be noted that certificate record and certificate are not same. Each certificate has own certificate serial number that is unique. In case that we have 2 different HTTPS connection to the same server then we have two certificate records but both of them will contain same certificate because the certificate serial number will be same. Simply, more certificate records can describe one certificate.

3.2 Interconnection of data in Bro logs

A great advantage of Bro logs is the interconnection between them through unique keys. Every line in any log has unique key linking other lines in the rest of logs. For example, if we have some connection record from the conn.log containing information about SSL then there is a unique key of this connection record which some SSL record in the ssl.log has as well. Also with high probability this SSL record should have some certificate. The SSL records has certificate path pointing to certificates records in x509.log. This example is shown in Figure 3.1.

3.3 Certificate and certificate path

The certificate is represented by the certificate record from the x509.log in Bro. The role of the certificate in HTTPS is to ensure credibility of the web server with certificate authorities. There are two types of certificate authorities a root CA and an intermediate CA. In order for a certificate to be trusted that the certificate must have been issued by a CA that is included in your browser (or any device) as trusted CA. If the certificate was not issued by a trusted CA, then the web browser will check if the certificate of the issuing CA was issued by a trusted CA, and so on until a trusted CA is found. These certificates from the root certificate to the end-user certificate are called a certificate path or a certificate chain. The certificate path in Bro is stored in the SSL record in ssl.log where is a list of unique keys pointing to the x509.log where all certificates are describe.

Figure 3.2 shows an example of the google certificate path by the Google Chrome and the Bro logs. In Google Chrome example, at the bottom of the path is a *.google.cz certificate. This certificate is end-user certificate and is signed by Google Internet Authority G2. Next is the Google Internet Authority G2 certificate, that is signed by GeoTrust Global CA, is called Subordinate Certificate Authority that are used in a large organization, such as Google, for signing the many certificates they need to operate. The GeoTrust Global CA certificate is signed only by itself and should be stored in the browser as trusted CA. In Bro the SSL record has the same certificate path in the ssl.log. The left unique key is the end-user certificate for *.google.cz that shows to second line in x509.log. The middle unique key is the Google Internet Authority G2 certificate that shows to third line in x509.log and the right unique key is the root GeoTrust Global CA certificate shows to fourth line in x509.log.

3.4 Labels

In our dataset we used Normal and Malware labels. Our approach of labeling is that we know the source IP of infected and normal computers. So, according to the source IPs in the connection records we put Normal or Malware labels to each connection record in conn.log and if the source IP is not known we put Background label. However records with Background labels are not used in our method. The labeling was done by our python scripts.

3.5 Final dataset

In total our entire dataset contains 85 captures — 19 normal and 66 malware captures. Tables 3.1 and 3.2 show detailed information about entire dataset. There are numbers about the connection records, SSL records, certificate records and the certificates.

Type of Data	CTU-13	MCFP	Own normal
Normal connection records	358,768	9,394	402,239
Malware connection records	10,843,391	50,133,670	0
Normal SSL records	20,677	404	68,848
Malware SSL records	3,621	327,354	0
Normal certificate records	2,351	2,618	41,542
Malware certificate records	7,581	162,290	0
Normal certificates	34	35	3,177
Malware certificates	6	1,647	0

Table 3.1. Numbers of connection records, SSL records, certificate records and certificates in part of datasets.

Type of Data	**Entire dataset**
Normal connection records	**770,401**
Malware connection records	**60,977,061**
Normal SSL records	**89,929**
Malware SSL records	**330,975**
Normal certificate records	**46,511**
Malware certificate records	**169,871**
Normal certificates	**3,246**
Malware certificates	**1,653**

Table 3.2. Numbers of connection records, SSL records, certificate records and certificates in entire datasets.

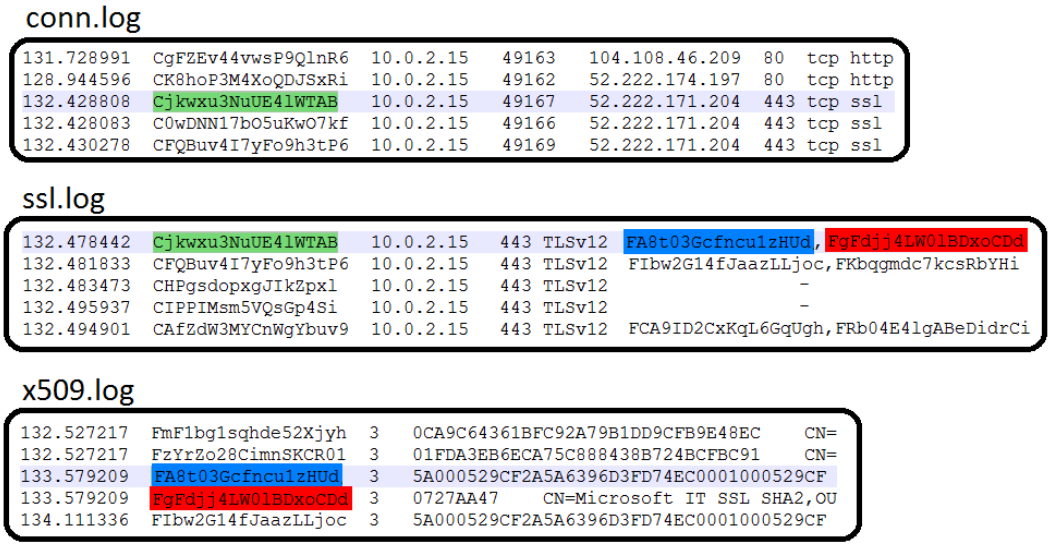


Figure 3.1. Example of the interconnection between logs. We can see that the third connection record in conn.log with unique key [Cjkwxu3NuUE41WTAB] has a SSL record in ssl.log with same key. Next the SSL record has 2 certificate keys. Both of them are in x509.log where the certificates are described. We can also see that the third and fifth certificate records in x509.log describing same certificate because the serial numbers in the fourth column are same.

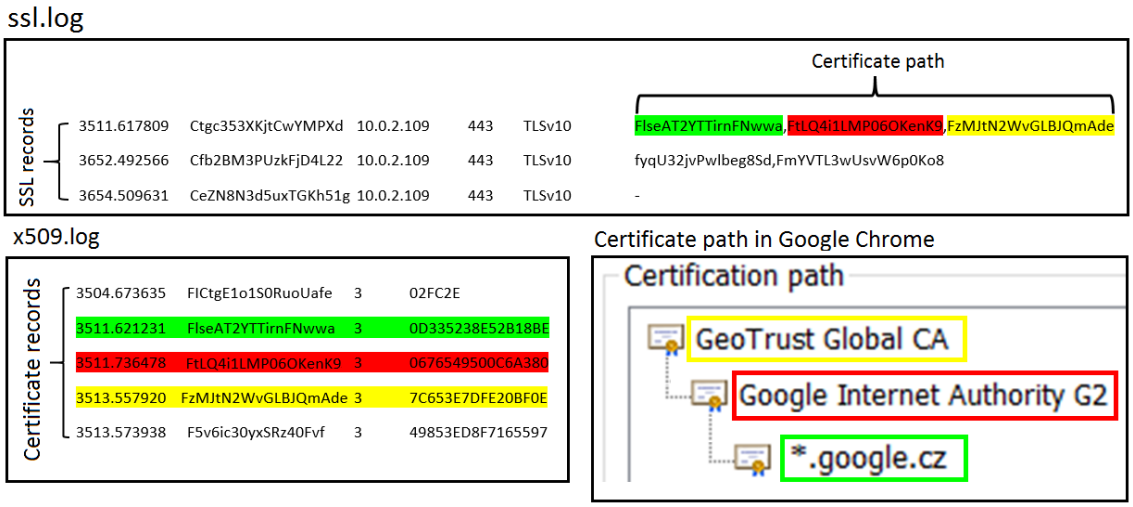


Figure 3.2. The example of the google certificate path in the SSL record in the Bro and in Google Chrome. Let's note that the third SSL record has no certificate path so no certificate.

Chapter 4
Features

The purpose of this chapter is to describe how we designed and created the features and the data model for machine learning algorithms. Our approach is based on Connection 4-tuples. A Connection 4-tuple is the basic unit of analysis of our algorithms and contains 28 features. To create Connection 4-tuples we use data from the conn.log, ssl.log and x509.log files. These three types of the Bro logs provide us enough information about the HTTPS traffic.

4.1 SSL aggregation

An SSL aggregation is a piece of data that is the concatenation of a triplet of records: one connection record, one SSL record and one certificate record. Where the connection record and the SSL record have the same unique key, and the unique key of the certificate record is the first unique key in the certificate path in the SSL record. Some SSL records have no certificate path, so no certificate, then the SSL aggregation contains only a pair of one connection record and one SSL record. Whether the SSL record has certificate path depends on many factors, for example, if the SSL handshake is successful, etc.

Figure 4.1 shows an example of four SSL aggregations and one connection record. SSL aggregation 1, 3 and 4 are very similar. Each of them has connection record and SSL record that have same unique key and also each of them has certificate path in the SSL record where first certificate unique key from this path gives us certificate record from x509.log. The SSL aggregation number 2 is also SSL aggregation according to definition but does not have the certificate path in the SSL record it means that this SSL aggregation 2 has no certificate.

4.2 Connection 4-tuples

The Connection 4-tuples are a group of SSL aggregations and some individual connection records. All of them share the same 4-tuple of source IP, destination IP, destination Port, and protocol. This 4-tuple is unique key for each Connection 4-tuple. Each Connection 4-tuple summarizes the behavior of the malware connecting to the C&C server or the behavior of the normal user connecting in most cases to normal websites. The label of the Connection 4-tuple is created according to its containing SSL aggregations and connection records that have their own label. In the vast majority of cases the Connection 4-tuples have all SSL aggregations and connection 4-tuples with same label but sometimes not. Then the final label is the majority one.

4.3 Generating Connection 4-tuples and the data model

An algorithm for generating Connection 4-tuples passes through each capture in entire dataset and finally creates the data model for machine learning algorithms. As

Figure 4.1. The example of creating SSL aggregations from our 3 imporatnt log files. Let's note that just the first unique key from the certificate path is used in the SSL aggregations. Next, the SSL aggregations 2 has no certificate because there is no certificate path in the SSL record.

mentioned each capture in dataset contains conn.log, ssl.log and x509.log, where all important information are prepared. The algorithm is described below.

Let's have the set N of the Connection 4-tuples, then

- *For each capture in dataset:*
  - *For each SSL record in ssl.log:*
    - *Take an unique key from the SSL record and find the connection record in conn.log with this unique key. This found connection record contains a 4-tuple (SrcIP, DstIP, DstPort, protocol). If this connection record does not have Malware or Normal label, read next SSL record.*
    - *If the certificate path in the SSL record is not empty take the first certificate unique key from the certificate path and find the certificate record in x509.log with this certificate unique key.*
    - *These 3 or 2 found records are new SSL aggregation containing the 4-tuple. Find some Connection 4-tuple in the set N containing the 4-tuple from the SSL aggregation and add this SSL aggregation to this found Connection 4-tuple. If such Connection 4-tuple does not exist, create new Connection 4-tuple with key as 4-tuple from the SSL aggregation, add the SSL aggregation to the newly created connection 4-tuple and add this newly created connection 4-tuple to the set N.*

- *For each connection record in conn.log:*
  - *Take the 4-tuple from the connection record. If there is any Connection 4-tuple in the set N that has a key same as the 4-tuple from the connection record and the Connection 4-tuple does not contain this connection record in any SSL aggregation, add this connection record to the Connection 4-tuple.*

- *Compute features for each Connection 4-tuple in the set N and create the data model from the set N, where the rows are the Connection 4-tuples and the columns are feature values.*

Our algorithm for generating Connection 4-tuples begins by reading SSL records line by line from the ssl.log. For each SSL record we find the connection record and certificate record if it exists. The Connection record includes, among others, 4-tuple (SrcIP, DstIP, DstPort, protocol) and label as well. The triplet or pair of records is thus found SSL aggregation. The reason why we read the ssl.log first is that connection records in conn.log contain all kind of traffic such as HTTP, HTTPS, DNS and etc. However our connection 4-tuples contain only HTTPS traffic and the ssl.log gives us information about the SSL properties of the connection record in the conn.log. If the SSL aggregation has normal or malware label then we add this SSL aggregation to an existing Connection 4-tuple according to the 4-tuple which is the key of each Connection 4-tuple. If there is no existing Connection 4-tuple with this 4-tuple, we create a new Connection 4-tuple, where a key is the 4-tuple from the SSL aggregation. Then, we also add the SSL aggregation to the newly created Connection 4-tuple and this newly created Connection 4-tuple is added to the list of all Connection 4-tuples.

When the complete ssl.log is read then we start to read connection records in the conn.log line by line and if any Connection 4-tuple from the list of Connection 4-tuples has a same key (4-tuple) as the connection record and the Connection 4-tuple does not contain this connection record, we add the connection record to this Connection 4-tuple. The answer to this step is the fact that although these special connection records have no SSL records in ssl.log they still belong to some Connection 4-tuple because for example they don't reach the application layer and the Bro is not able to know if it is HTTP, HTTPS or DNS, only the transport layer is known such as TCP or UDP. As soon as these steps are applied on all captures in dataset the features are computed for each Connection 4-tuple.

Finally the data model is created. The data model is the basis of our experiments and what our detection algorithms classify. The data model contains 17,384 lines, where each line represents one Connection 4-tuple and has 28 columns representing feature values. The labels of Connection 4-tuples are stored in a second file containing 0 as Normal and 1 as Malware in the same order as the Connection 4-tuples. The data model contains 9,252 Normal Connection 4-tuples and 8,132 Malware Connection 4-tuples.

The last and very important step is to normalize the data model. For each feature column the biggest value is found and each value in this column is divided by the biggest value (except the -1 value). So in the end, the features values in data model are in the $\langle 0,1 \rangle$ interval or they have a value of -1.

4.4 Example of a Connection 4-tuple

To better understand the process of generating Connection 4-tuples, Figure 4.2 shows a more clear example of how this is done. There is a set of Connection 4-tuples where

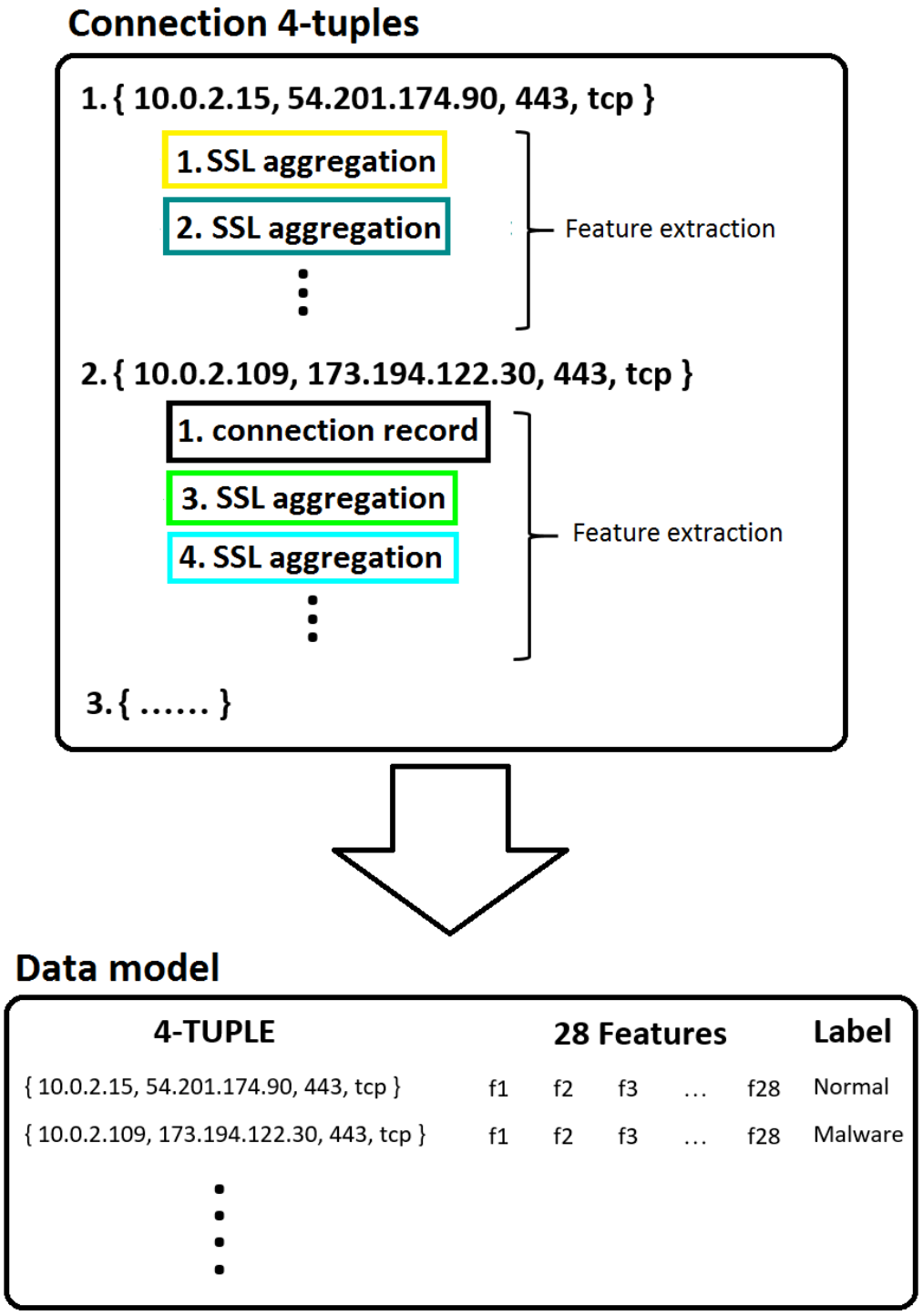


Figure 4.2. The example of Connection 4-tuples containing SSL aggregations and connection record. When all Connection 4-tuples are created the data model is created as well.

each of them has unique key as 4-tuple and contains at least one SSL aggregation and any number of connection records. From our previous example from Figure 4.1 there are four SSL aggregations and one connection record. Let's apply our algorithm to examples in Figure 4.1 and Figure 4.2.

The ssl.log in Figure 4.1 is read line by line. The SSL record number 1 with its unique key [C2OyAD2iBBP3ED5rnb] has connection record in conn.log with same unique key. This connection record has this [10.0.2.15, 54.201.174.90, 443, tcp] 4-tuple and Normal label. Next, the SSL record has also the certificate path. The first certificate unique key [F2HZJX1f4EeRGbdrk8] from the certificate path is also in the certificate record number 1 in the x509.log. These three records are the SSL aggregation number 1. We check if set of Connection 4-tuples contain some Connection 4-tuple that has our 4-tuple. Since our set is empty at this moment, we create new Connection 4-tuple with our 4-tuple and with our SSL aggregation number 1 and we add this Connection 4-tuple to the set. In Figure 4.2 there is this created Connection 4-tuple with the SSL aggregation number 1. According to our algorithm let's read next SSL record from the ssl.log. The second SSL record number 2 has also the connection record with the same 4-tuple as the previous SSL aggregation number 1 and Normal label as well. However this SSL record has no certificate path so this new SSL aggregation number 2 contains only 2 records. This SSL aggregation number 2 belongs to the previous Connection 4-tuple because it has same 4-tuple. SSL records number 3 and 4 create SSL aggregation number 3 and 4 with 4-tuple [10.0.2.109, 193.194.122.30, 443, tcp]. They are classic SSL aggregations that have certificate.

All lines from the ssl.log is read and our set contains two Connection 4-tuples where both of them have two SSL aggregations (Figure 4.2) Next step in our algorithm is to find connection records that belongs to some existing Connection 4-tuple. Connection records are connection records in conn.log that are not ssl (they don't have SSL record in the ssl.log with the same key). In Figure 4.1 the connection record number 1 in conn.log has same 4-tuple as the second Connection 4-tuple in our set and this second Connection 4-tuple doesn't contain this connection record then this connection record is added to this Connection 4-tuple. The rest of the connection records in conn.log are part of the SSL aggregations so conn.log is read. The last step is to compute features for each connection 4-tuple and create data model where each line is one Connection 4-tuple with feature values and the label. It is shown in Figure 4.2.

We can see that the second Connection 4-tuple in the set has two SSL aggregations. An attentive reader may notice that two certificate records from these two SSL aggregations have same certificate serial number (it is last column in the x509.log in Figure 4.1). This fact means that these two certificate records are the same certificate.

Let's note that these logs in examples don't have all columns. This is just example for understanding and viewing. The real logs have many columns.

4.5 Feature extraction

The generation of features is a very important part of this thesis. The features are new, precise and designed to detect malware traffic. Most of them were created based on our expert knowledge on the area and the thorough analysis of our malware data.

Our data model holds each of the 28 features, as they were computed for each Connection 4-tuple. As recalled, each Connection 4-tuple, in time, is an aggregation of other information. Therefore, each feature of the Connection 4-tuple is a complex extraction and summarization of information.

All of our 28 features are computed from each Connection 4-tuple. Even though our features are all together, for simplicity and description they are divided in the following three groups:

- Connection features

- SSL features
- Certificates features

The connection features are features from the connection records and describe common behavior of the traffic that is not associated with the certificates and the encryption. The SSL features are features from the SSL records and describe the SSL handshake and information about the encrypted traffic and the certificate features are features from the certificate records and describe information about the certificates that a webser give us during the SSL handshakes. Each feature is some float value and if this feature can not be computed for a lack of information the value is -1.

4.5.1 Connection features

All these connection record features are based on the connection records from the conn.log.

1. **Number of SSL aggregations and connection records** — Each Connection 4-tuple contains some number of SSL aggregations and connection records. This first feature is simply the sum of the two.
2. **Mean of duration** — Each connection record in the Connection 4-tuple contains a duration in seconds. With each incoming connection record the Connection 4-tuple store this duration value in a list and in the end the mean is computed from this list of the duration. Let set X containing duration values, then the mean is:

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

3. **Standart deviation of duration** — Same list of duration values from connection records as previous one but with standart deaviation. The σ is:

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

$$E(X^2) = \frac{x_1^2 + x_2^2 + x_3^2 + ... + x_n^2}{n}$$

$$\sigma = \sqrt{E(X^2) - (E(X))^2}$$

4. **Standart deviation range of duration** — This feature describe how many percent from all duration values is out of range. The range is has two limits. The upper limit is $mean + standart_deviation$ and the lower limit is $mean - standart_deaviation$. An exmaple is shown in Figure 4.3. There are 10 blue point values of the duration values. The mean of this values is 42,7 and the standart deviation is 27.5. The black line represents the mean a two red lines are upper and lower limit. All duratin values that are over the upper limit or under the lower limits are out of range. It is 3 points and all duratin values are 10 points so $3points/10points = 0.3$ is the standart deviation range of duration.
5. **Payload bytes from originator** — The number of payload bytes the originator sent for all connection records from the conn.log. With each incoming connection record we just add this value.
6. **Payload bytes from responder** — The number of payload bytes the responder sent for all connection records from the conn.log. With each incoming connection record we just add this value.

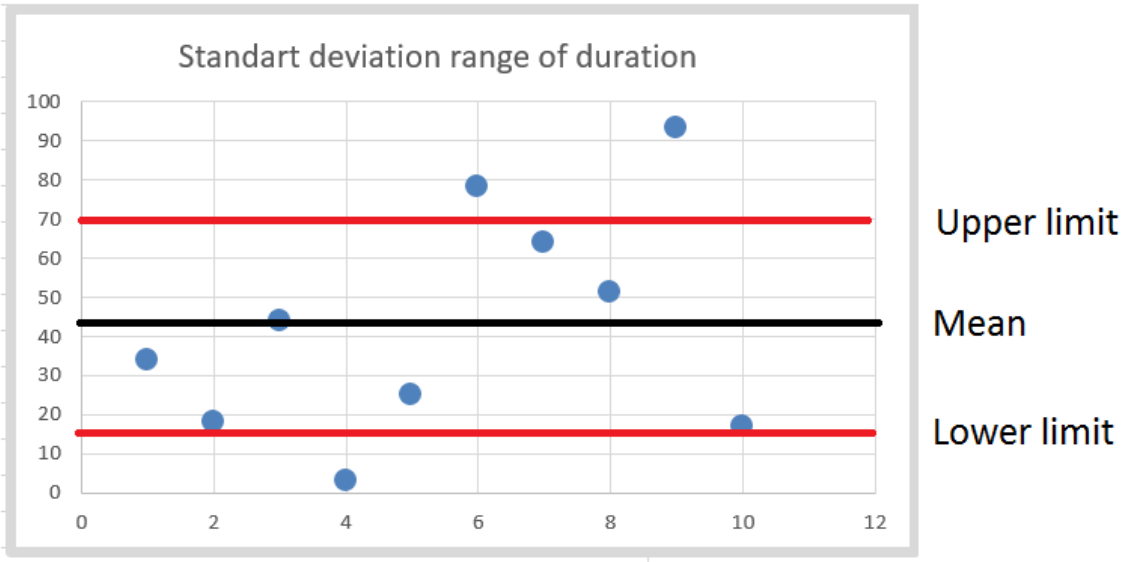


Figure 4.3.

7. **Ratio of responder bytes and all bytes** — All bytes are bytes from originator and bytes from responder. Then this feature is:

$$R = \frac{r}{r+o},$$

where r is number of bytes from responder and o is number of bytes from originator.

8. **Ratio of established states of connection** — Each connection record contains state of the connection. There are 13 types of these states. Our approach is that we divided these states to establihed connection states and non established states. These 2 groups of states describe if there is any TCP handshake or if it is just attempt to TCP handshake. The established states are [SF, S1, S2, S3, RSTO, RSTR] that contain a kind of successful TCP hanshake and non establihed states are [OTH, SO, REJ, SH, SHR, RSTOS0, RSTRH] that contains a kind of unsuccessful handshke. What each state excatly means is written in Bro documentation [3]. Our ratio R is:

$$R = \frac{e}{e+n},$$

where e is number of established states and n is number of non established states.

9. **Inbound packets number** — A value that is also included in connection record. With each incoming connection record to the Connection 4-tuple we just add this value.
10. **Outbound packets number** — A value that is also included in connection record. With each incoming connection record to the Connection 4-tuple we just add this value.
11. **Periodicity mean** — Each connection record has a time of capture. So we can measure group of them how they are periodic. An exmaple 4.4 shows five connection records. The time is fictitious, but for our exmaple is enough. The first step is to compute time diferences between the connection records in order. It is a first time difference. Next step is to compute second time diferences from the first time difference in absolute value. If the value is zero, it means that relevant connection records are periodic. In the end the values from the second time difference are stored in a list. From this list the mean is computed:

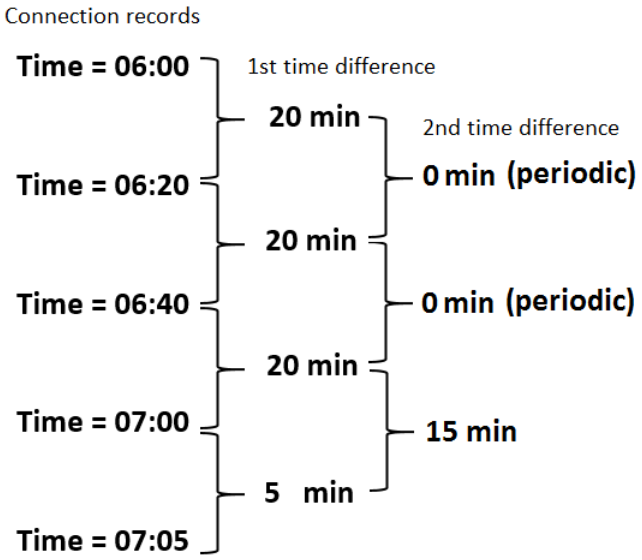


Figure 4.4.

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

12. **Standart deviation of periodicity** — The feature uses same list of the second time diferences as the feature 11, but with standart deviation.

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

$$E(X^2) = \frac{x_1^2 + x_2^2 + x_3^2 + ... + x_n^2}{n}$$

$$\sigma = \sqrt{E(X^2) - (E(X))^2}$$

4.5.2 SSL features

All these SSL features are based on the SSL records in the ssl.log.

13. **Ratio of connection records and SSL aggregations** — This feature describe ratio between connection records that are not SSL and connection records that are SSL. The ratio R is:

$$R = \frac{f_n}{f_s},$$

where f_n is number of connection records without SSL and f_s is number of connection records wit SSL.

14. **Ratio of TLS and SSL verison** — All SSL records have version of TLS or SSL protocols that are used for encryption. There are SSL 1.0, SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2 and TLS 1.3, where SSL protocol is older than TLS and almost all Normal traffic use TLS. This feature describe how many SSL records have TLS protocols. The ratio R is:

$$R = \frac{TLS}{TLS + SSL},$$

where TLS is number of SSL records that have TLS protocol and SSL is number of SSL records that have SSL protocol.

15. **Ratio of SNI** — The SNI is Server Name Indication that is included in the SSL record. This feature describes how many SSL records have SNI because some SSL records have empty SNI. Our results show that Malware SSL records have more often empty SNI than Normal SSL records. The ration R is:

$$R = \frac{F_s}{F_a},$$

where F_s is number of SSL record having SNI and F_a is number of all SSL records.

16. **SNI as IP** — Sometimes the SSL records have SNI as IP address. In this case the SNI IP should be same as destination IP address. This feature is -1 if any SSL record in its Connection 4-tuple has SNI as IP but the SNI is not same as DstIP. It is 0 if any SSL record has SNI as IP and the SNI is same as DstIP and it is 1 if there is no SSL record that is IP address.

17. **Mean of certificate paths** — As mentioned most of SSL records have certificate path. The certificate path is defined in chapter 3. This feature stores number of certificates in certificate path for each SSL record in a list. The mean is computed from the list.

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

18. **Ratio of self signed certificate** — Bro is able to recognize if the end-user certificate is self signed. This information is in the SSL record. This feature is ratio of the self signed certificates and all end-user certificate in Connection 4-tuple. The ratio R is:

$$R = \frac{s}{c},$$

where s is number of self signed certificates and c is number of all certificates.

4.5.3 Certificates features

All these certificate features are based on the certificate records in the x509.log and some of them on the SSL records in the ssl.log as well.

19. **Public key mean** — Each certificate record describing a certificate contains public key of the certificate. With each incoming certificate record to the Connection 4-tuple the public key is added to a list. From the list the mean is computed.

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

20. **Mean of certificate validatity periods** — Each certificate has a validity period. Exmaple of validity period is in Figure 4.5 where the certificate period is 10 years from 01.01.2010 to 01.01.2020. These bound dates are stored in certificate record in unix time. From each certificate is this validity period stored in a list in seconds. Then the mean is computed from this list:

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

21. **Standard deviation certificate validatity periods** — Same list of certificate validity periods in seconds as the previous feature but with standard deviation.

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

$$E(X^2) = \frac{x_1^2 + x_2^2 + x_3^2 + ... + x_n^2}{n}$$
$$\sigma = \sqrt{E(X^2) - (E(X))^2}$$

22. **Validity of the certificate period during the capturing** — With capture time and the validity period of the certificate we can find out if the certicate during the capturing is valid. If the capture time is inside the certificate valid period then it is ok. This feature is number of certificates that were out of validity period during the capturing the traffic. In Figure 4.5 the certificate is valid from 01.01. 2010 to 01.01. 2020. If you browse in 2025 or in 2005 in any webpage with this certificate, the certifiate woud not be valid. Our results show that Malware use more often notvalid certificates then Normal.

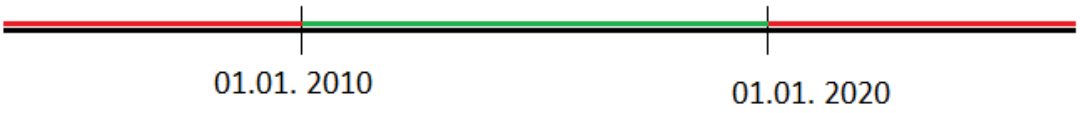


Figure 4.5.

23. **Mean age of Certificates** — This feature is ratio of two time lenghs. First lengh is certificate validity period and second length is a period from beginning of certificate validity period to capture time. So we know how the certificate is old. For each certificate is computed ratio of these periods and the result is stored in a list. Then the mean in computed from the list. In figure 4.6 is an exmaple of certificate validity period d and period of duration r. The certificate validity period from 01.01.2010 to 01.01.2020 is 315,532,800 seconds in unix time and the period from 01.01.2010 to 01.01.2015 is 157,766,400 seconds in unix time. Then period r is divided by the certificate validity period d.

$$E(X) = \frac{\frac{r_1}{d_1} + \frac{r_2}{d_2} + ... + \frac{r_n}{d_n}}{n}$$

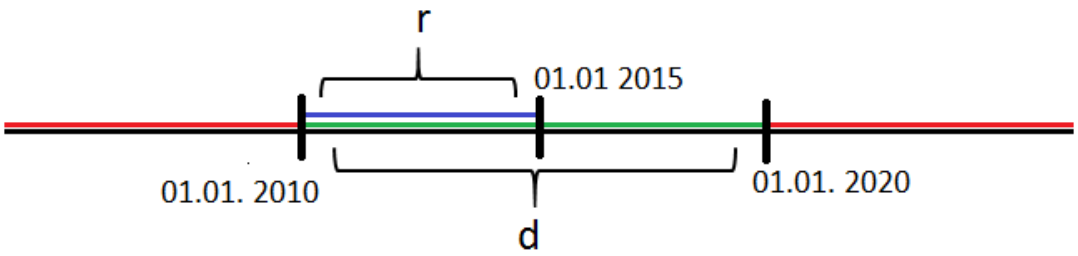


Figure 4.6.

24. **Amount of certificates** — The Connection 4-tuple ususally contain one certificate, but sometime more. So this feature is just number of certificates for one Connection 4-tuple. (Connection 4-tuples ususally contain a lot of certificate records in SSL aggregation but mostly it is still one certificate)
25. **Mean of number of domains in certificate SAN DNS** — The SAN is Subject Alternative Names describing which domains belong to this certificate. With each

new incoming certificate (not certificate record) the number of dns in SAN is stored in a list. Then the mean is computed from the list. An example of part of google certificate SAN dns: [∗.google.com, ∗.android.com, ∗.appengine.google.com, ∗.cloud.google.com, ∗.google-analytics.com, ∗.google.ca, ∗.google.cl, ∗.google.co.in, ∗.google.co.jp, ∗.google.co.uk, ∗.google.de]

$$E(X) = \frac{x_1 + x_2 + x_3 + ... + x_n}{n}$$

26. **Ratio of certificate records and SSL records** — The feature describe how many SSL record have certificate path, because certificate record can be added to SSL aggregation just in case that is contained as the first certificate in the certicate path. The ratio R is:
$$R = \frac{c}{s},$$
where c is number of certificate records and s is number of SSL record for one Connection 4-tuple.
27. **SNI in SAN DNS** — The SNI is Server Name Indication that is included in the SSL record. SAN DNS are domains in the certicate record that belong to the certificate. Usually the SNI is part of the SAN DNS. If any certificate record doesn't contain the SNI in the SSL record in one SSL aggregation then the value of feature is 0 and if all certificate records contain SNI in SAN DNS in each pair of SSL aggregation in the Connection 4-tuple then the value of feature is 1.
28. **CN in SAN DNS** — The CN is Common Name that is a part of the certificate record. CN should be part of the SAN DNS. This feature value is 0 if any certificate doesn't contain the CN in the SAN DNS and the feature value is 1 if all certificates contain the CN in the SAN.DNS.

Chapter 5
Method

5.1 Machine Learning Algorithms

The key of our research is to choose the right Machine Learning Algorithms. For our experiments we choose SVM, Neural Netowrk, Random forest and XGBoost. For using these machine learning algorithms we used scikit library [12] and XGBoost library [13] providing complex description of algorithms and many examples of usage.

The creating plots for visualization our data was managed by T-SNE. This method is for dimensionality reduction and is particularly well suited for the visualization of high-dimensional datasets. T-SNE library was provided by scikit library as well.

- SVM
- Neural Network
- Random Forest
- XGBoost

5.1.1 SVM

SVM (Support Vector Machines) is a supervised learning model used for classification and regression analysis. SVM can efficiently perform a non-linear classification using the kernel trick, mapping inputs into high-dimensional feature spaces. We use Radial Basis Function (RBF) kernel. The RBF kernel provides parameter C and parameter gamma. The C parameter tells how much you want to avoid misclassifying each training example and the gamma parameter defines how far the influence of a single training example reaches. For our experiments the parameter C is 110 and gamma is 0.1. Figure 5.1. shows classification of data with different values of parameter C and parameter gamma. This example is from scikit library examples [14].

5.1.2 Neural Network

Our model of Neural Network uses MLP Classifier (Multi-layer Perceptron classifier) that optimizes the log-loss function using stochastic gradient descent. We choose the stochastic gradient descent with Adam (Adaptive Moment Estimation) that is method for minimizing an objective function. Parameter of the method is Alpha. Alpha is a parameter for regularization penalty, that combats overfitting by constraining the size of the weights. Increasing alpha may fix high overfitting by encouraging smaller weights. Similarly, decreasing alpha may fix an underfitting by encouraging larger weights, potentially resulting in a more complicated decision boundary. In our method the parameter alpha is 1e-05. Figure 5.2 shows classification of data with with different values of parameter alpha.

5.1.3 Random Forest

We used Random Forest Classifier model that is an estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to

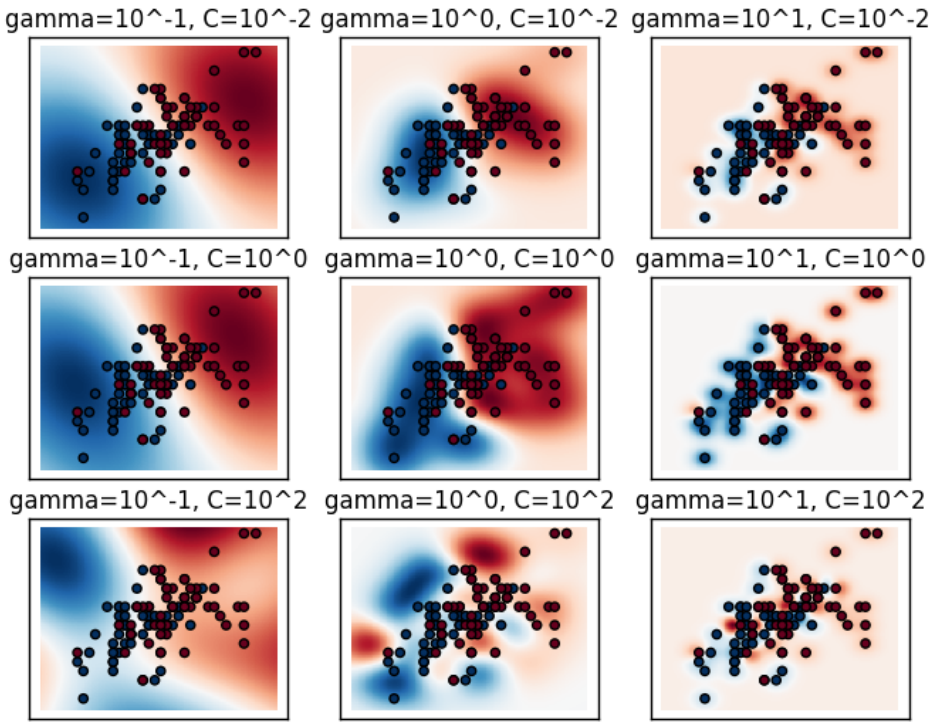


Figure 5.1. Example of SVM with RBF kernel from scikit library examples [14]. Several images with different values of C and gamma.

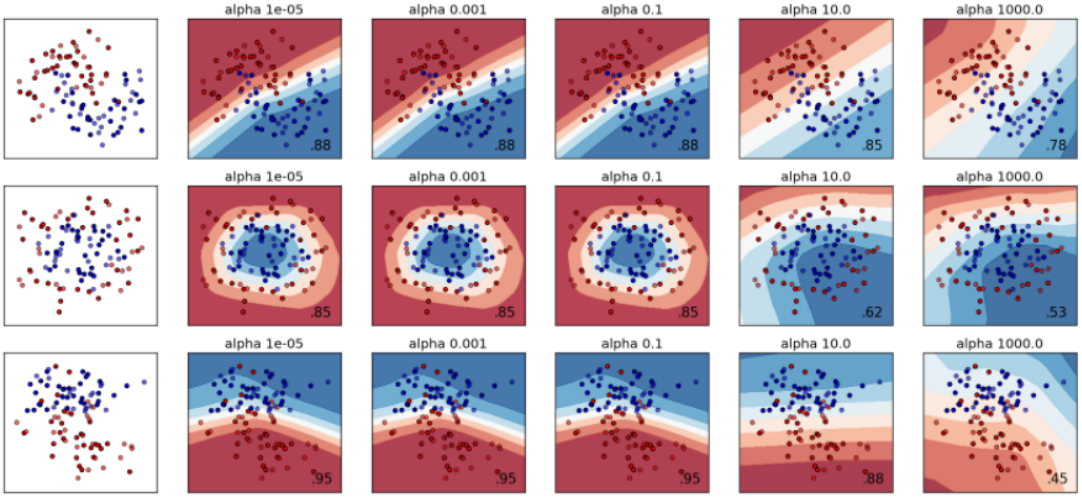


Figure 5.2. Example of Neural Network with Adam solver from scikit library examples [15].

improve the predictive accuracy and control over-fitting. In our method the number of trees in the forest is 1000.

5.1.4 XGBoost

XGBoost (eXtreme Gradient Boosting) contains a lot of algorithms such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines. We used tree booster with logistic regression that is for binary classification. XGBoost has a lot of parameters but the most important for us are:

- max depth — describe maximum depth of a tree
- gamma — minimum loss reduction required to make a further partition on a leaf node of the tree.
- min child weigh — minimum sum of instance weight (hessian) needed in a child.

Our experiments use 2 types of settings for XGBoost, called XGBoost 1 and XGBoost 2.

- XGBoost 1
  - n estimators = 1000
  - max depth = 10
  - gamma = 0
  - min child weigh = 1
- XGBoost 2
  - n estimators = 1000
  - max depth = 3
  - gamma = 0.1
  - min child weigh = 5

Chapter 6
Experiments

This chapter shows our results. There are the content of data model, data analysis by plotting, training experiments and testing experiments.

6.1 Data model

Our data model is a matrix of values where each row is identified with the Connection 4-tuple ID and the columns are the feature values. Each feature ranges from 0 to 1, or has the -1 value. The data model has 17,384 rows and 28 columns. From the data model, 20% is taken as the testing data and the remaining 80% as the training data. The training data are used for a cross-validation and for the final training and the testing data are used only for the final test evaluation. Table 6.1 shows the distribution of labels in the data model.

Connection 4-tuples	Train data (80%)	Test data (20%)	All data (100%)
Normal	7,402	1,850	9,252
Malware	6,505	1,627	8,132
Normal + Malware	13,907	3,477	17,384

Table 6.1. Normal and Malware Connection 4-tuples in the data model splitted to train and test data.

6.2 Data plotting

For the analysis of our data, we plot it using the T-SNE (T-Distributed Stochastic Neighbor Embedding) method. This method is for dimensionality reduction and is particularly well suited for the visualization of high-dimensional datasets. T-SNE takes our 28 dimensional space and transforms it into two dimensional space. We also try to use this two dimensional space instead of the 28 dimensional space for machine learning algorithms, but the results were not interesting.

Figure 6.1 shows the training and testing data together where Malware Connection 4-tuples are ones and Normal Connection 4-tuples are zeros. Figure 6.2 contains only the testing data.

6.3 Training experiments

In this section all the experiments only use the training data. The testing data is left to the final experiments. The training experiments consist of a cross-validation and a learning curves analysis. The training experiments give us possibility to find the best parameters for machine learning algorithms and show how our data looks like.

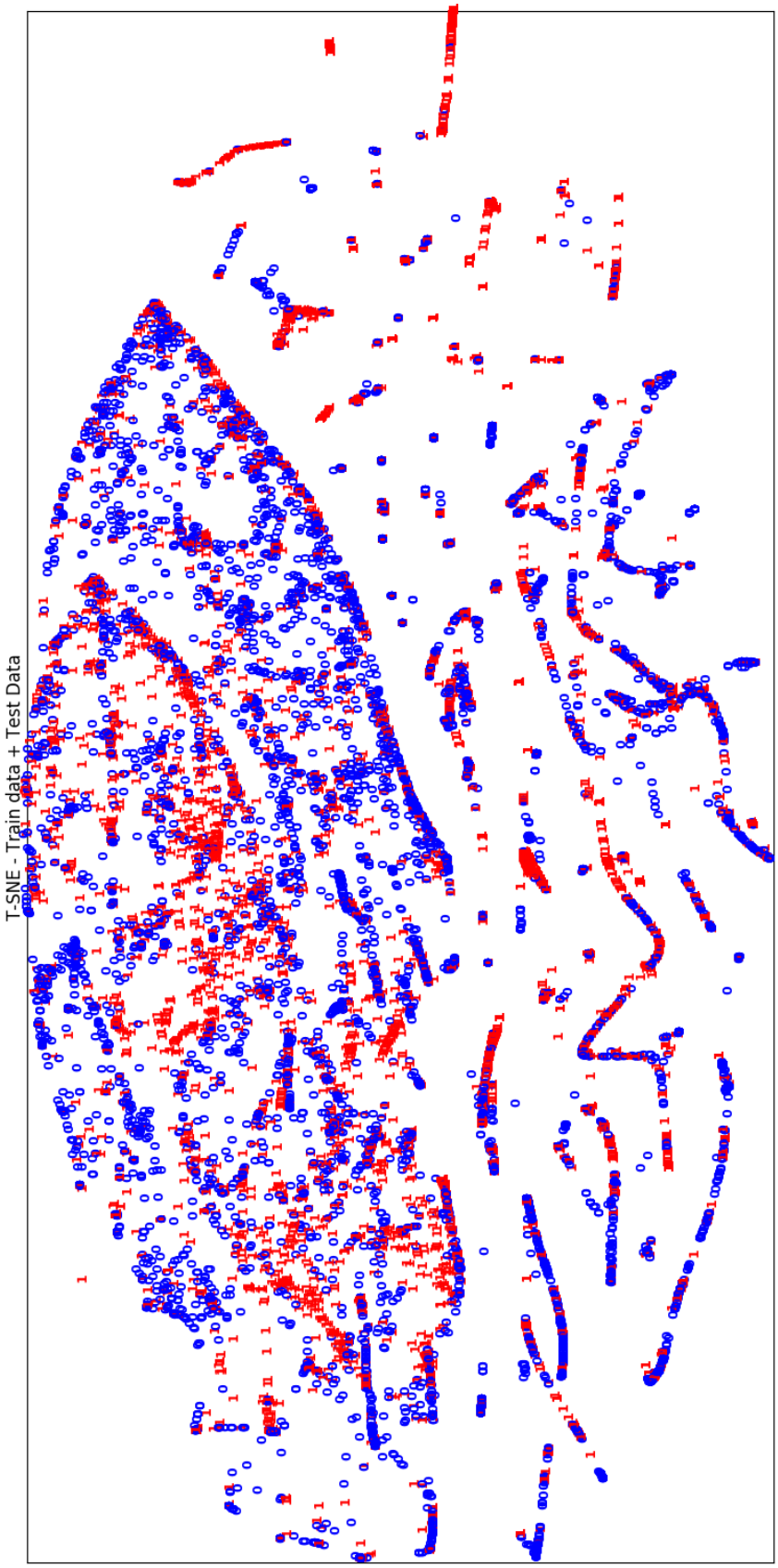


Figure 6.1. The figure shows all data. The ones are Malware Connection 4-tuples and the zeros are Normal Connection 4-tuples.

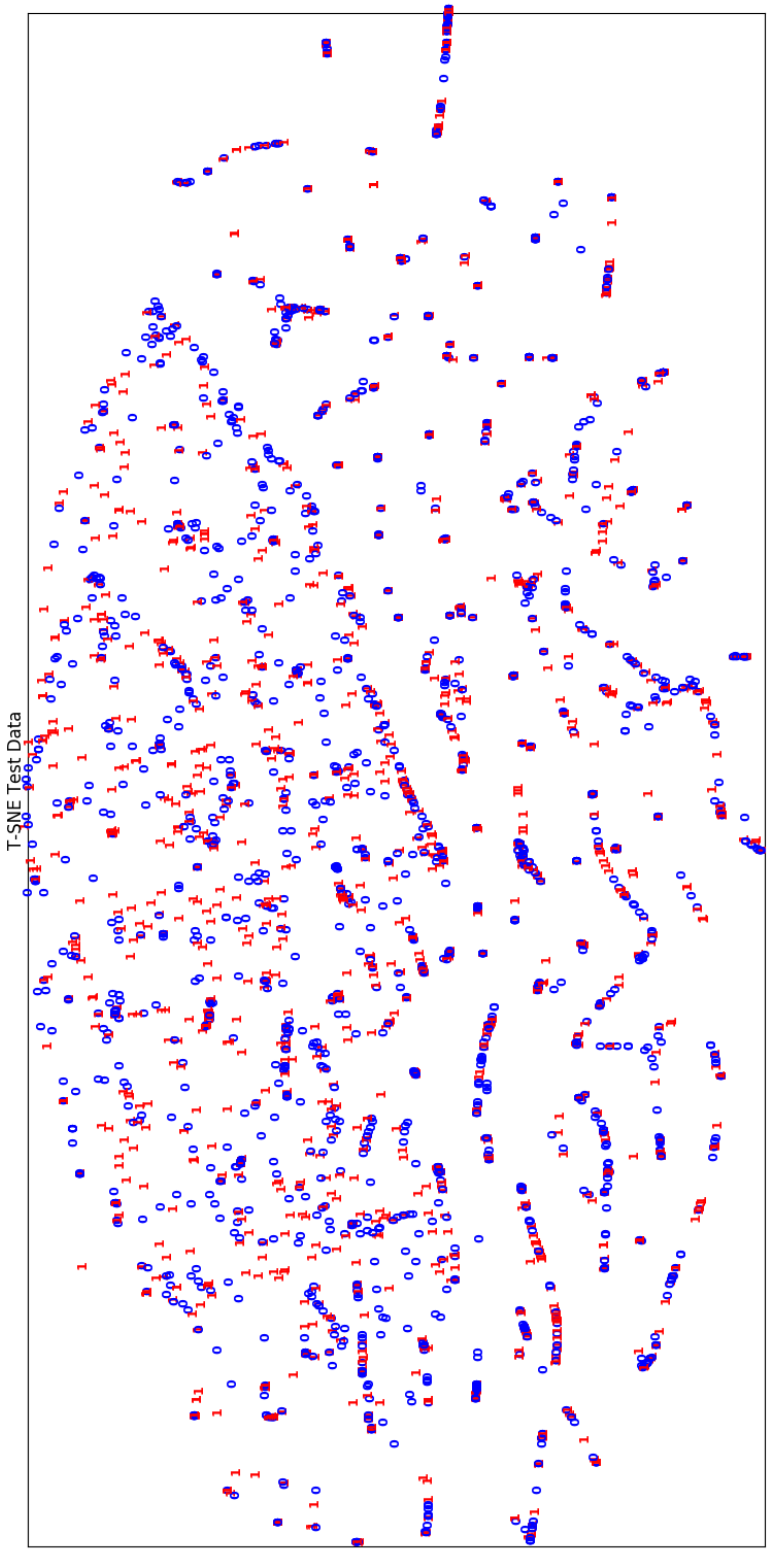


Figure 6.2. The figure shows testing data. The ones are Malware Connection 4-tuples and the zeros are Normal Connection 4-tuples.

In the training and testing experiments the accuracy of detection is used. The definition of the accuracy of detection is:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

In this equation TP is True Positive , FP is False Positive, TN is True Negative and FN is False Negative. The TP is detected as malware and it is malware, the FP is detected as malware but it is normal, TN is detected as normal and it is normal and FN is detected as normal but it is malware.

6.3.1 Cross-validation

We use a 10-fold cross-validation that randomly splits the training data into 10 subsets and iteratively trains and tests them. In each iteration the machine learning algorithm is trained with 9 training subsets and tested to the one testing subset. At the end there are 10 results of the detection accuracies from which the average is created. Table 6.2 shows results from 10-fold cross-validation where XGBoost 1 and XGBoost 2 achieve the best results.

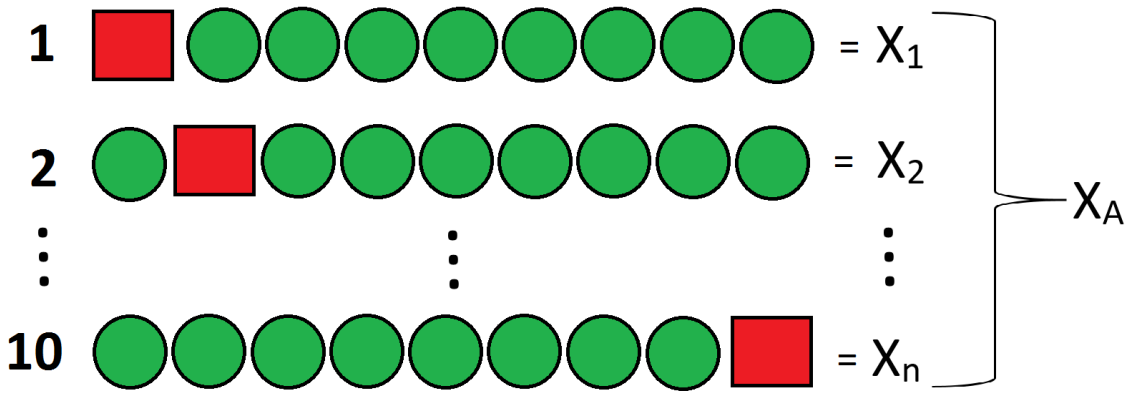


Figure 6.3. Example of 10-fold cross-validation. The rectangle is testing sample and the circle is training sample. In each iteration the accuracy of detection X_n is computed and then the average X_A of all 10 iteration is computed.

Algorithm	Cross-validation
SVM	0.7842
NN	0.8072
Random Forest	0.9391
XGBoost 1	0.9635
XGBoost 2	0.9579

Table 6.2. The accuracies of detection by the 10-fold cross-validations.

6.3.2 Lerning curve

A learning curve is a graphical representation of the validation and training score with an increasing training data samples. It shows us how much we benefit from adding more training data. The learning curve consists of the validation curve and the training curve where the vertical axis is the score or accuracy of detection and the horizontal axis is amount of the train data.

- Training curve — The current amount of the training data is taken, the machine learning algorithm is learnt to this amount of data and then is tested to the same amount of data as it is learnt. The accuracy from the testing is the training score on vertical axis.
- Validation curve — The current amount of the training data is taken and the 10-fold cross-validation is applied for the current amount of the training data. It gives us 10 results of detection accuracies from which the average is computed. This average is the validation score on vertical axis.

Learning curves give us information about an overfitting and underfitting. The overfitting means that learning curves have high variance. Usually the training curve has small error and the validation curve has big error. So between the training curve and validation curve is big gap. The underfitting has high bias. It means that training curve has big error and the training data is not fit well. Our Figure 6.4 shows exmaples of underfitting and overfitting.

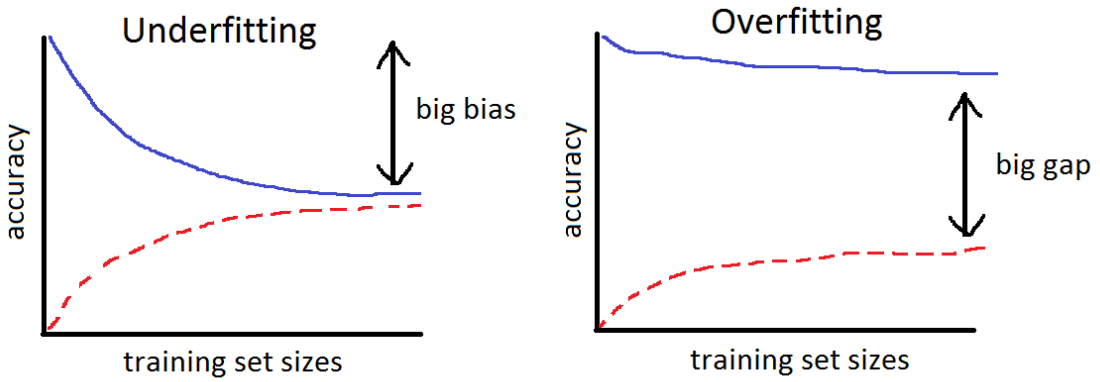


Figure 6.4. An example of underfitting (left) and overfitting (right). The training curve in underfitting example has big error. In the overfitting example the training curve has very small error and the validation curve has big error.

We create the learning curves for each machine learning algorithm. The training curve and validation curve have 10 points showing the values of detection accuracy (score) for the amount of training data.

In Figure 6.5 there is the learning curve for SVM. It looks that the SVM underfits because the training curve has an high error but there is a steady growth of the training and validation curve.

In Figure 6.6 there is the learning curve for Neural Network that is similar to the previous learning curve with SVM. The training curve has also an high error so it is underfitting as well but there is not a steady growth of the training and validation curve. Around 5000 training samples the training and validation curves drop. This phenomenon is repeated with 10,000 train samples just with the validation curve. We would need more data to improve results.

In Figure 6.7 there is the learning curve for Random Forest. We can see that the training curve has 0However the validation curve has very high detection accuracy (score). So the gap between the training and validation curve is small.

In Figure 6.8 and Figure 6.9 there are the learning curve for XGBoost 1 and XGBoost 2. The first figure 6.8 with XGBoost 1 has the best training score 96.3% from all algorithms. But his learning curve overfits, because the training curve has zero error.

From this reason we try to find other parameters for XGBoost algorithm, that would have better learning curve without the overfitting. The figure 6.9 shows XGBoost 2 with the learning curve that doesn't have overfitting. The training curve slowly falls and approaches the validation curve. Despite the fact that this validation curve from XGBoost 2 is not so good as the validation curve from XGBoost 1, it is still very good detection accuracy with 95.8%.

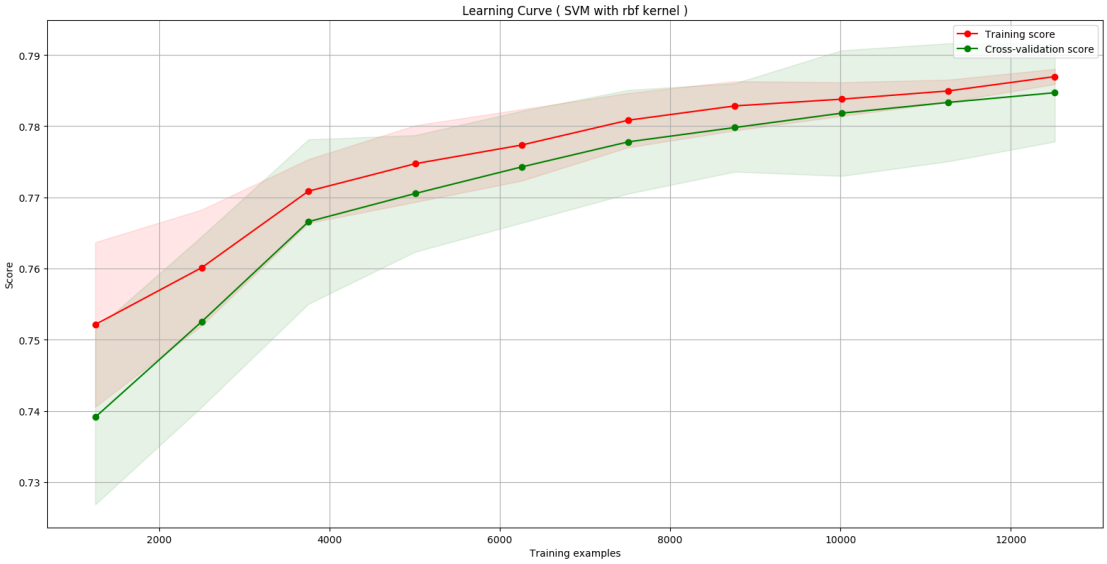


Figure 6.5. The learning curve for SVM. It looks that SVM underfits because the training curve has an high error. However there is a stedy growth of the training and validation curve. Probably more training data would help.

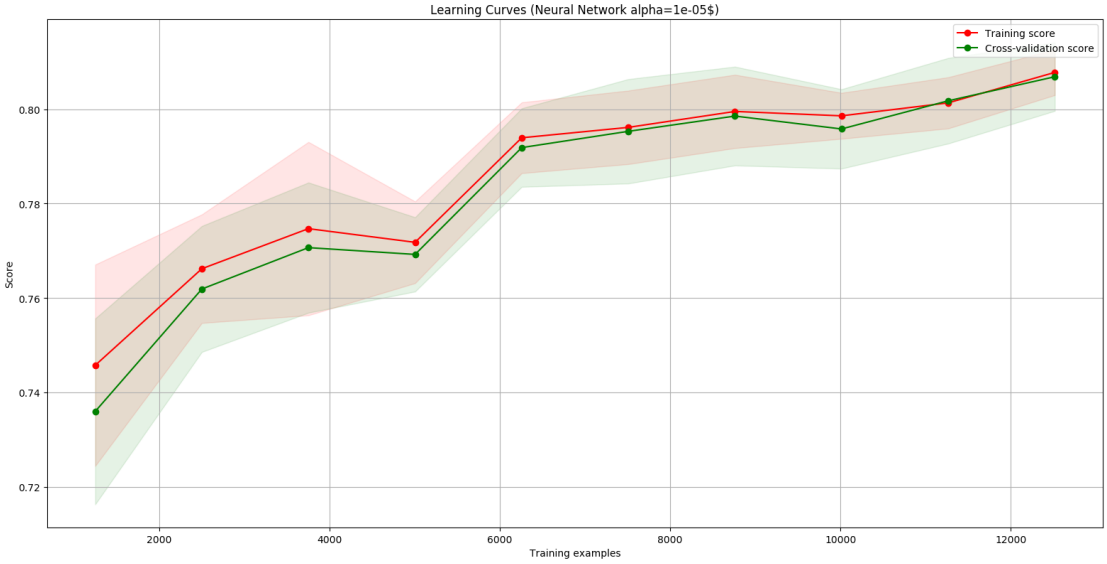


Figure 6.6. The learning curve for Neural Network. The training curve has an high error so, it is probably underfitting. However there is not a stedy growth of the training and validation curve. Around 5000 train samples the training and validation curves drop. This phenomenon is repeated with 10,000 train samples just with the validation curve. Probably more training data would help.

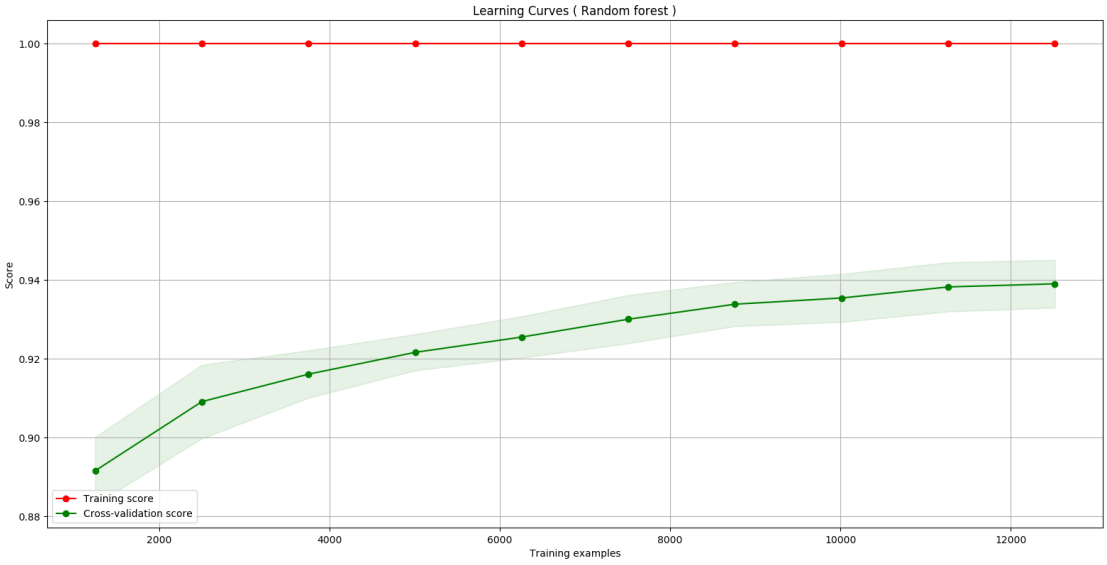


Figure 6.7. The learning curve for Random Forest. We can see that the training curve has 0% error that is the overfitting but the validation curve has very high detection accuracy (score). The validation curve increases and the achieved result is almost 93.9%.

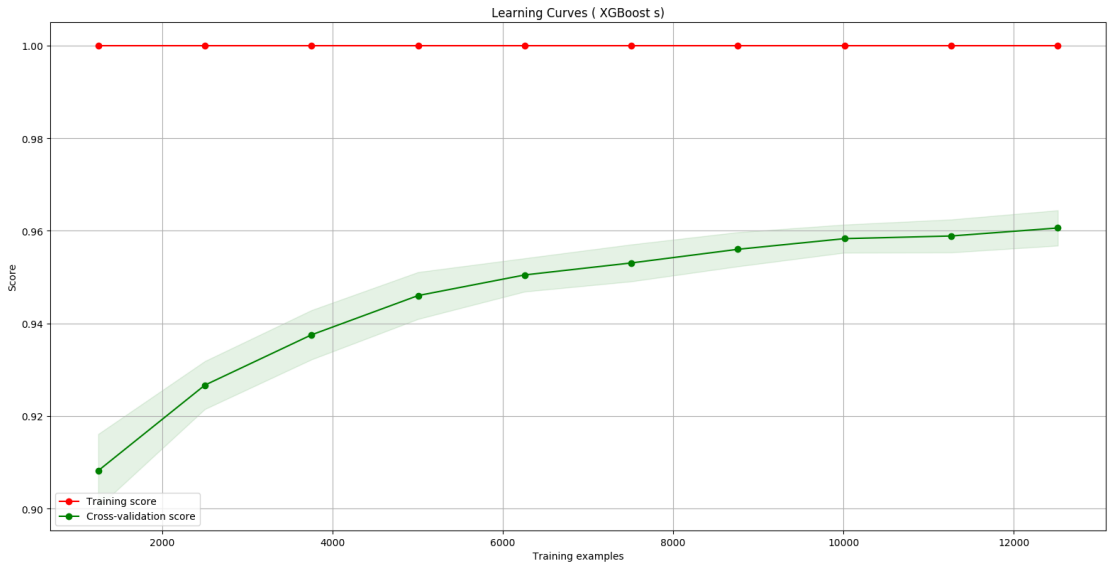


Figure 6.8. The learning curve for XGBoost 1. There is also the training curve with 0% error that is overfitting, but the validation curve has high score.

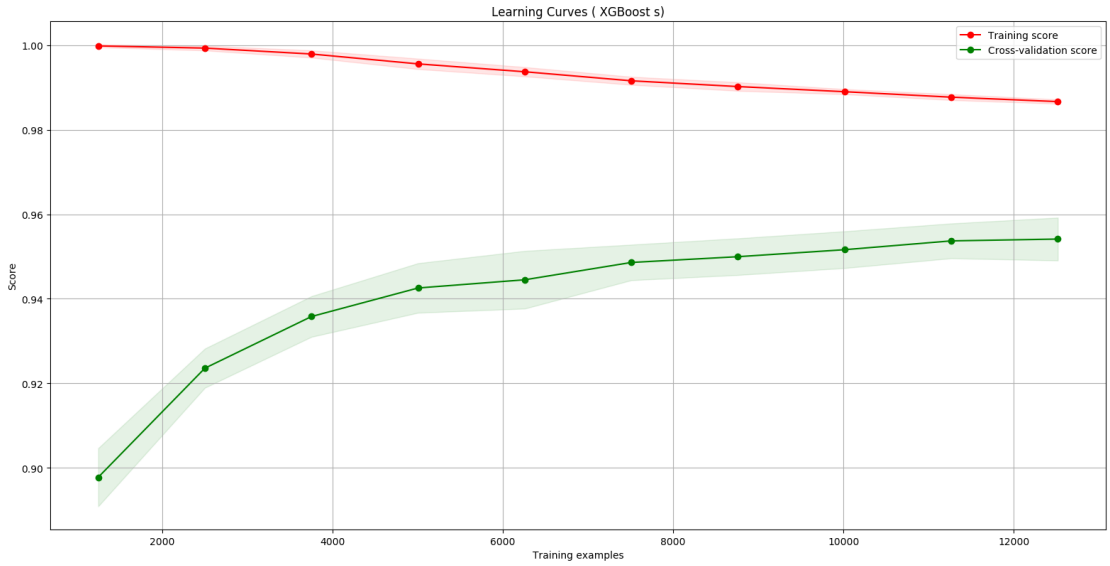


Figure 6.9. The learning curve for XGBoost 2. This learning curve has low variance, so it doesn't overfit. The training curve slowly falls and approaches the increasing validation curve.

6.4 Testing experiments

Finally our testing data are used. The machine learning algorithms are learnt from the training data and then tested with the testing data that has not yet been used. Each detection gives us confusion matrix containing numbers of True negative, False negative, True positive and False positive. From the Confusion Matrix we can compute

next measures describing our detection results such as false positive rate, sensitivity, specificity etc. For our experiment the most important measures are the accuracy and the false positive rate (FPR). FPR describes how many percent is detected as malware but it is just normal. The definitions of all used measures is below.

Tables 6.3 and 6.4 contain values of measures computed from the Confusion Matrix. Figures 6.10 and 6.11 show the measures in the graphical representation. And finally table 6.5 compares the detection accuracy between the 10-fold cross-validation training and the final testing.

- Accuracy
$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$
- False Positive Rate
$$FPR = \frac{FP}{FP + TN}$$
- Sensitivity
$$TPR = \frac{TP}{TP + FN}$$
- Specificity
$$TNR = \frac{TN}{TN + FP}$$
- Precision
$$PPV = \frac{TP}{TP + FP}$$
- Negative Predictive Value
$$NPV = \frac{TN}{TN + FN}$$
- False Negative Rate
$$FNR = \frac{FN}{FN + TP}$$
- False Discovery Rate
$$FDR = \frac{FP}{FP + TP}$$
- F1 Score
$$F1 = \frac{2TP}{2TP + FP + FN}$$
- Matthews correlation coefficient (MCC)
$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

ML Algorithm	Cross-validation accuracy	Test data accuracy
SVM	0.7842	0.7745
NN	0.8072	0.7987
Random forest	0.9391	0.9416
XGBoost 1	0.9635	0.9664
XGBoost 2	0.9579	0.9572

Table 6.5. Result comparison of the cross-validation and the final testing accuracy.

Measure	SVM	NN	Random Forest
Sensitivity	0.6030	0.6490	0.8961
Specificity	0.9254	0.9303	0.9816
Precision	0.8767	0.8911	0.9772
Negative Predictive Value	0.7260	0.7509	0.9149
False Positive Rate	**0.0746**	**0.0697**	**0.0184**
False Discovery Rate	0.1233	0.1089	0.0228
False Negative Rate	0.3970	0.3510	0.1039
F1 Score	0.7145	0.7511	0.9349
MCC	0.5643	0.6099	0.8849
Accuracy	**0.7745**	**0.7987**	**0.9416**

Table 6.3. Confusion Matrix for Machine Learning Algorithms

Measure	XGBoost 1	XGBoost 2
Sensitivity	0.9496	0.9348
Specificity	0.9811	0.9768
Precision	0.9778	0.9725
Negative Predictive Value	0.9568	0.9446
False Positive Rate	**0.0189**	**0.0232**
False Discovery Rate	0.0222	0.0275
False Negative Rate	0.0504	0.0652
F1 Score	0.9635	0.9533
MCC	0.9326	0.9143
Accuracy	**0.9664**	**0.9571**

Table 6.4. Confusion Matrix for Machine Learning Algorithms

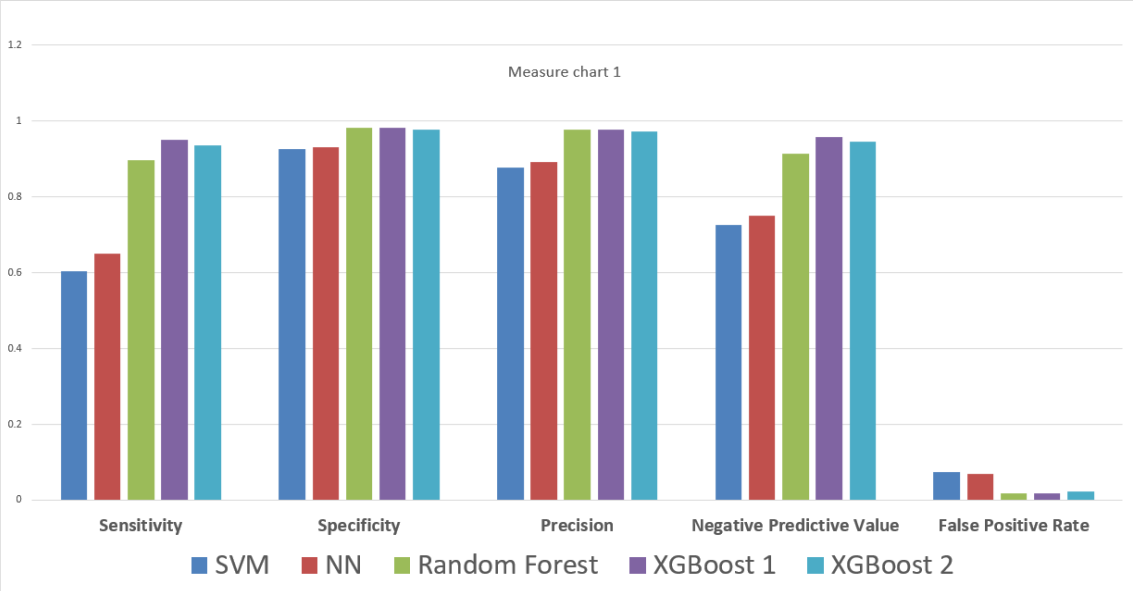


Figure 6.10. This Figure describes Sensitivity, Specificity, Precision, Negative Predictive Value and False Positive Rate for each machine learning algorithm. The first columnn is always SVM, second is Neural Network, third Random Forest, fourth XGBoost 1 and fifth column is always XGBoost 2.

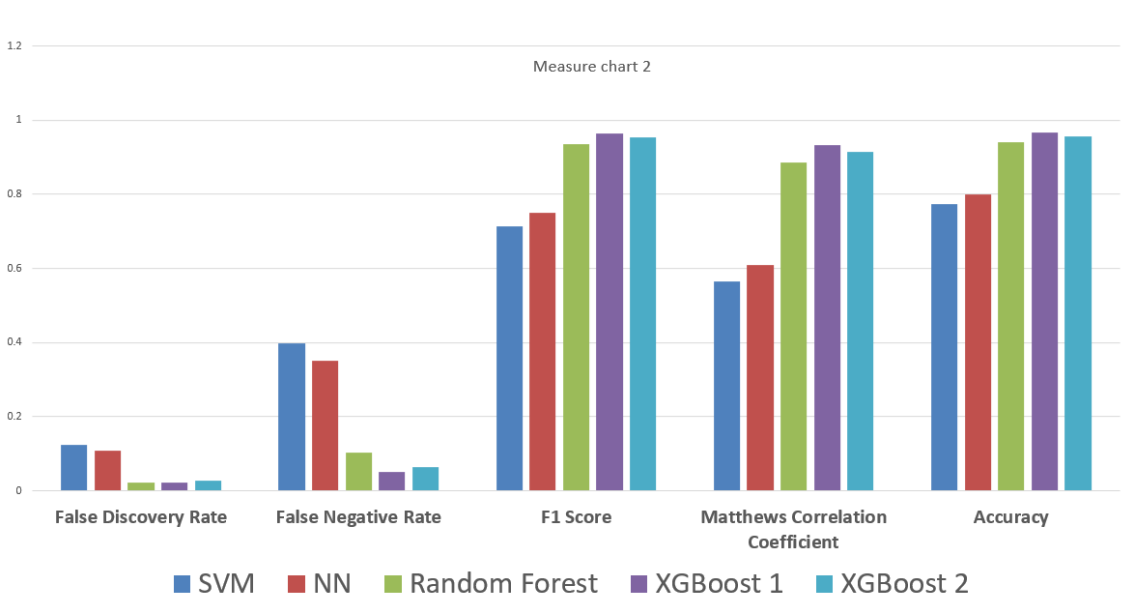


Figure 6.11. This Figure describes False Discovery Rate, False Negative Rate, F1 Score, Matthews correlation coefficient and Accuracy for each machine learning algorithm. The first columnn is always SVM, second is Neural Network, third Random Forest, fourth XGBoost 1 and fifth column is always XGBoost 2.

Chapter 7
Certificate experiments

Motivated by our good results, this chapter is a small extension of our methods, where we tried to detect only the malware certificates in our dataset by looking at the encrypted data. This was not in the original plan of the research, but seems to us interesting.

7.1 Certificate data

The certificate is sent from the server if there is successful SSL handshake between the client and the server. The certificate information is taken from certificates records in x509.log file describing all information about the certificates. Our dataset contains 4,899 certificates with unique serial number where 3,246 of them are normal certificates and only 1,653 of them are malware certificates. The label of the certificate is the majority of certificates records that belong to this certificate.

7.2 Features of certificate

The certificates contain same certificate feature as the Connection 4-tuple and some others. These features are created during the algorithm for the creating Connection 4-tuples. if any new certificate is found in this algorithm (according its certificate unique serial number) the information from the certificate record is stored in a set of certificates. If any certificate incoming from x509.log already exists in our set of certificates, the time of capture and Server Name Indication (SNI) are added. Finally each certificate contains its basic information from the certificate record in x509.log and also list of captured times and SNI names. Used features are described below.

- **CN in SAN DNS** — The CN is Common Name that is a part of the certificate record and SAN DNS are domains in the certicate record that belong to the certificate. CN should be part of the SAN DNS. This feature value is 0 if the certificate doesn't contain the CN in the SAN DNS and the feature value is 1 if the certificate contains the CN in the SAN.DNS.
- **Public key number** — The certificate contains a public key in bits.
- **Number of domains in SAN DNS** — It is number of domains in SAN DNS.
- **Number of SNI** — This feature is number of SNI included with this certificate. Usually one certificate is used in more SSL handskaes. And also some SSL handshake doesn't have the SNI. So this feature is sum of SNI list that is sotred in this certificate set.
- **Mean of valid certificates** — The captured time list contains captured times of using this certificate. Anytime this certificate is used, the captured time is stored and at the end each captured time is checked if it is inside of validation period of this certificate. The value of fearure is ratio of non-valid cases, when the certificate is valid and all certificates cases.

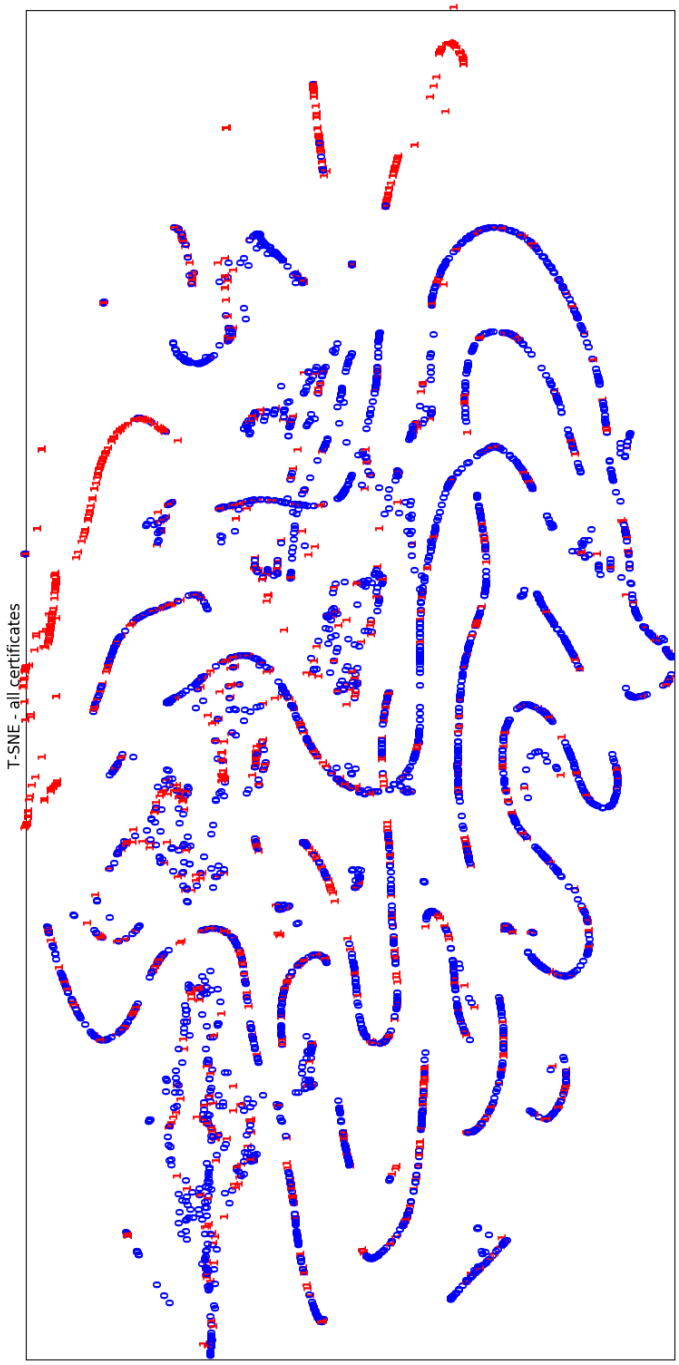


Figure 7.1. Example of all 4899 certificates. The ones are malware certificates and zeros are normal certificates.

- **Mean of certificate validatity periods** — The captured time list contains captured times of using this certificate. Anytime this certificate is used, the captured time is stored. This feature is mean age of all of all cases when the certificate is used.

At the end there is the data model of 4899 certificates where each of them has these 6 features. Next this data model is normalized for each column.

7.3 Experiments

The data model is splitted to 20% testing data and 80% training data. The training data is used for 10-fold cross-validation in each machine learning algorithm. Then the machine learning algorithms are learnt to all training data ant tested on testing data. The results are shown in Table 7.1.

ML Algorithm	Cross-validation accuracy	Test data accuracy
SVM	0.7512	0.7592
NN	0.7514	0.7642
Random forest	0.6989	0.7041
XGBoost 1	0.7043	0.7245

Table 7.1. Certificate result comparison of the cross-validation and the final testing accuracy.

Chapter 8
Analysis of results

Our experiments consist of the training and testing experiments. The training experiments use training data and the testing experiments use training data for learning and testing data for testing. Training data contain 80% and testing data 20% from all available Connection 4-tuples. The testing data is left for the final testing experiments. The most important measures for us are the accuracy and the false positive rate. It is important that FPR is low because too frequent false alarm is expensive and untrustworthy.

8.1 Training experiments

The training experiments show that SVM and Neural Network have underfitting problem, because the training curves have high bias so, high error. Instead of this fact with incoming training samples the accuracy of the training and validation curves still improves. To solve this issue we would probably need more training samples or other features and also in case of Neural Network there should be more experiments with algorithm parameters. The 10-fold cross-validation experiments on training samples are 78.42% for SVM and 80.72% for Neural Network.

As far as the training experiments for Random Forest the training curves have zero error and the validation curve has a steady growth. It is overfitting problem but the validation has high score with 93.91%.

XGBoost 1 and XGBoost 2 are same machine learning algorithms with other parameters. XGBoost 1 achieves the best cross-validation results with 96.35% but with overfitting, where training curve has zero error. Our endeavor was to limit this overfitting. We tried to learn about XGBoost as much as possible and we also mix the parameters of XGBoost to limit the overfitting. Finally we found it. The XGBoost 2 has 95.79% cross-validation accuracy and no overfitting. The Figure 6.9 shows that the training curve slowly falls and approaches the increasing validation curve.

8.2 Testing experiments

In the testing experiment the machine learning algorithms are learnt for all training data and then tested on testing data that has not yet been used. According to the Table 6.5 the best detection accuracy have XGBoost 1 with 96.64% and XGBoost 2 with 95.72%. Also Random forest has also very good detection accuracy with 94.16%. As far as SVM and Neural Network the accuracy is not so good. SVM achieves 77.45% and the Neural Network achieves 79.87%. The best results in the false positive rate (FPR) has Random Forest with 1.84% XGBoost 1 with 1.89% and XGBoost 2 with 2.32%.

8.3 Certificate experiments

The results of certificate experiments show that malware certificates are difficult to detect. According to the Table 7.1 the best detection accuracy has Neural Network with 76.42%. The reason is that malware use a many of normal certificate in our dataset such as google etc.

Chapter 9
Conclusion

The aim of our research was to detect malware HTTPS traffic without decryption and to create novel features to differentiate the malware and normal HTTPS usage. The features were extracted from our datasets containing real malware and normal traffic. The set of features consisted in features for connections, SSL data and certificates, helping to describe the behaviour of HTTPS traffic. The final table of features is our data model. Our machine learning algorithms were trained and tested from the data model and the XGBoost 1 that is the best method, achieved 96.64% detection accuracy with 1.89% false positive rate. These results seem to be good enough for our datasets and context. If we had so good results on big datasets as well, then the result would be more trustworthy.

However even though the False Positive Rate of XGBoost and Random Forest may seem low, it may relatively high for larger networks, because it means that almost two connections out of 100 between two endpoints would be false alarms. We conclude that the usage done by the malware of HTTPS is distinct from normal usage and that these differences can be used for creating powerful features for machine learning algorithms that achieve satisfactory results.

9.1 Feature work

There are a lot of possibilities how to improve the final accuracy and the false positive rate in future work. One of them is to gather a greater amount of training and testing samples, also to find other available information in BRO logs describing HTTPS traffic in different way or to find different open source software describing HTTPS traffic. Next more in-depth to analyze how malware works in HTTPS and also to try more machine learning algorithms with more different parameters.

References

[1] GOOGLE. *HTTPS Usage*. Website was accessed on May 2017.
https://www.google.com/transparencyreport/https/metrics/.

[2] CISCO Blake Anderson. *Hiding in Plain Sight: Malware's Use of TLS and Encryption*. January 25, 2016; Website was accessed on May 2017.
https://blogs.cisco.com/security/malwares-use-of-tls-and-encryption.

[3] Bro. *Bro IDS*. Website was accessed on May 2017.
https://www.bro.org/.

[4] Honza Stiborek Sebastian Garcia, Martin Grill, and Alejandro Zunino. *An empirical comparison of botnet detection methods*. 2014.
http://mcfp.weebly.com/the-ctu-13-dataset-a-labeled-dataset-with-botnet-normal-and-background-traffic.html.

[5] Maria Jose Erquiaga. Website was accessed on May 2017.
https://mcfp.felk.cvut.cz/publicDatasets/.

[6] Frantisek Strasak. *Scripts for thesis*. Website was accessed on May 2017.
https://github.com/frenky-strasak/My_bachelor_thesis.

[7] David McGrew Blake Anderson, Subharthi Paul. *Deciphering Malware's use of TLS (without Decryption)*. 6 Jul 2016.

[8] David McGrew Blake Anderson. *Identifying Encrypted Malware Traffic with Contextual Flow Data*. 2016.

[9] J. Lokoc, J. Kohout, P. Cech, T. Skopal, and T. Pevny. *k NN Classification of Malware in HTTPS Traffic Using the Metric Space Approach*. 2016.

[10] www.moz.com. Website was accessed on May 2017.
https://moz.com/top500.

[11] www.quantcast.com. Website was accessed on May 2017.
https://www.quantcast.com/top-sites.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. *Scikit-learn: Machine Learning in Python*. 2011.
http://scikit-learn.org/stable/index.html.

[13] XGBoost. *XGBoost library*. Website was accessed on May 2017.
https://github.com/dmlc/xgboost.

[14] scikit-learn developers. *RBF SVM parameters*. Website was accessed on May 2017.
http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html.

[15] scikit-learn developers. *Varying regularization in Multi-layer Perceptron*. Website was accessed on May 2017.
http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html.

Appendix A
Content of the CD

- project_code — Source code for all research for python Python 2.7.13.
- data_models — Prepared data models for machine learning algorithms.
- tex_folder — Thesis in tex.
- strasak_thesis_2017.pdf — The electronic version of this thesis.