NTNU
Norwegian University of
Science and Technology

Protection Against DNS Tunneling Abuses on Mobile Networks

Terje Kristoffer Skow

Master of Science in Communication Technology
Submission date: June 2016
Supervisor: Van Thanh Do, ITEM
Co-supervisor: Hai Ngyuen, Telenor Research

Norwegian University of Science and Technology
Department of Telematics

Title: Protection Against DNS Tunneling Abuses on Mobile Networks
Student: Terje Kristoffer Skow

Problem description:

Lately, it has been discovered abuses on mobile networks that exploit the zero-rate websites to tunnel IP traffic without paying for service. The technique used in these abuses is called Domain Name System (DNS) tunneling and poses a significant threat for mobile networks. DNS tunnels can be detected by Deep Packet Inspection (DPI) i.e. analyzing each single DNS payload but this could be exhaustive with high traffic. This project is aiming at providing protection against DNS tunneling abuses by traffic analysis using machine learning techniques. An experiment will be done to gather data for the analysis. The experiment will use the open source software openGGSN which works as a real Gateway GPRS Support Node (GGSN) where the data are gathered. The project, more specifically, consists of the following tasks:

- Study of DNS tunneling abuses in mobile networks
- Study of machine learning methods applicable for DNS tunneling
- Study and installation of openGGSN, an open source GGSN
- Gather data from experiments
- Proposal, design and implementation of protection against DNS tunneling using machine learning techniques

Responsible professor: Thanh van Do, ITEM
Supervisor: Hai Thanh Nguyen, Telenor Research

Abstract

The use of mobile internet is increasing as the service becomes faster and more reliable. It is not only used by smartphones and tablets, but also regular computers are connected. With the increase in usage comes the need for an increased security. Companies have over the last 15 years been aware of Domain Name System (DNS) tunneling as means to perform data exfiltration and Command and Control (C&C) attacks in their networks. Before that DNS tunnels were used to access the internet at cafés and hotels without having to pay for it.

Mobile devices today contain more and more data which might be sensitive for both the user and his company and DNS tunnels are already in use on mobile devices to avoid paying for internet data usage. If history repeats itself, as it often does, will DNS tunnels soon be used to exfiltrate data from mobile devices without anyone noticing. This is what this study is trying to prevent. The study tries to find a viable machine learning classifier for detecting DNS tunnels.

Machine learning is a great tool to find statistical properties of datasets, and as DNS tunnels are irregularities should its properties be different. The K-means classifier, a cluster classifier, and the One-Class SVM (OCSVM) classifier, an outlier detector, are studied and tested in this study.

The data was planned to be gathered using the opensource software openGGSN. Using much time trying to set it up, did this plan have to change. The data was then gathered with Wireshark. It captured DNS traffic generated from four Virtual Machines (VMs) where one was using a DNS tunnel. At first the DNS tunnel stood for over 50% of the data collected, so it had to be reduced to be more representing of a larger network. The data was reformatted by merging the request and response in one line so the classifier could use those features together.

The precision, recall and F-score of the classifiers were tested on different initiation parameters and features. For the K-means the results started bad and neither changing the parameters nor features helped the results. The OCSVM has multiple kernels which were tested and the poly kernel looked very good on the first test. When changing the nu parameter and the features, did the results of the poly kernel change drastically for the worse. The Radial Basis Function (RBF) kernel kept a quite high score specifically on the recall of the outliers and the precision on the

inliers. More tests were executed using the RBF kernel changing both the gamma and the nu parameters, which are the most sensitive parameters for the kernel. Which in the end resulted in a 96% F-score where only the precision on outliers was under 90% which means the models largest weakness is a few false positive.

Sammendrag

Bruken av mobilt internett øker fort når tjenesten blir raskere og mer stabil. Den blir ikke bare brukt av smarttelefoner og nettbrett, men også vanlig datamaskiner er koblet til. Med den økte bruken blir sikkerheten viktigere og viktigere. Bedrifter har de siste 15 årene visst om bruken av Domain Name System (DNS) tunneller til å laste ned sensitiv data og utføre kontroll og kommander (K&K) angrep på deres private nettverk. Før DNS tunneller ble brukt til det, ble det brukt av personer som ville bruke internett gratis på hoteller og kaféer.

Mobile enheter inneholder i dag mer og mer informasjon som kan være sensitiv for både brukeren og bedriften hvis den kommer på avveie. DNS tunneller er allerede tilgjengelig som applikasjoner på mobile enheter for å slippe og betale for data bruken. Hvis historien gjentar seg selv, noe den ofte gjør, vil DNS tunneller snart bli brukt for å laste ned data fra mobile enheter uten at noen legger merke til det. Det er hva denne studien vil prøve å hindre. Studien vil prøve å finne en levedyktig maskin lærings metode for å detektere DNS tunneller.

Maskin læring er en god metode for å finne statistiske egenskaper av et datasett. Siden DNS tunneller er unormal bruk vil de ha annerledes statistiske egenskaper en vanlig DNS trafikk. K-means, en klynge klassifiserer, og One-Class SVM (OCSVM), utligger detektor, metodene ble studert og testet i denne studien.

Det var først planlagt å bruke gratis programvaren openGGSN for å samle inn data. Det viste seg å være tidkrevende og vanskelig å få satt ordentlig opp, så den planen måtte endres. Dataen ble derfor samlet inn med Wireshark. Den fanget opp DNS trafikken generert av fire virituelle maskiner (VM) der en av de brukte en DNS tunnel. DNS tunnelen stod for over 50% av dataen i starten, så mengden ble redusert for å kunne representere en mengde i et større nettverk. Dataen ble deretter formatert ved å slå sammen spørsmål og svar til en linje så metodene kunne bruke verdiene sammen.

Presisjonen, tilbakekallingen og F-poengsummen av metodene ble testet med forskjellige initierings parametere og verdier fra datasettet. K-means metoden sine første resultater var ikke noe bra, og verken endringer av parameterene eller verdiene i datasettet hjalp. OCSVM metoden hadde flere kjerner som ble testet og poly kjernen så ut til å være veldig bra ved første test. På de neste testene ble nu parameteren og verdier fra

datasettet endret, noe som førte til at poly kjernen fikk dårlige resultater. Radial Basis Function (RBF) kjernen gjorde det stabilt bra gjennom alle testene, spesielt på tilbakekallingen av utliggere. Det ble utført flere tester på RBF kjernen hvor både gamma og nu parameterene, som er de mest essensielle for kjernen, ble endret. Resultatet til slutt var at F-poengsummen var 96% og presisjonen av utliggere var det svakeste på 87%, noe som betyr at den får noen falske positive utliggere.

Preface

I would like to thank Thanh Van Do from Department of Telematics (ITEM) at Norwegian University of Science and Technology (NTNU) and Hai Thanh Nguyen from Telenor Research for helping me find this problem and for helping me through both this thesis and in the course TTM4501, Telematics, specialization project which started this thesis. They have helped figuring out how to attack the problem and guided me towards smart solutions.

Rune Skow from Comcept also needs a thanks for helping with information on the mobile network I needed to understand how DNS is resolved in it. Thanks to Neels Hofmeyr at Osmocom who tried to help me with configuring openGGSN, even though I did not manage to set it up.

I would also like to thank my family for support and help through this thesis and all the years at NTNU.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI Artificial Intelligence.

ANS Authoritative Name Server.

BTS Base Transceiver Station.

C&C Command and Control.

csv comma-separated values.

DNS Domain Name System.

DPI Deep Packet Inspection.

GGSN Gateway GPRS Support Node.

GPRS General Packet Radio Service.

GSM Groupe Spécial Mobile.

GTP GPRS Tunnelling Protocol.

IANA Internet Assigned Numbers Authority.

IP Internet Protocol.

ITEM Department of Telematics.

NTNU Norwegian University of Science and Technology.

OCSVM One-Class SVM.

PCA Principal Component Analysis.

RBF Radial Basis Function.

RR	Resource Record.
SGSN	Serving GPRS Support Node.
SIM	Subscriber Identity Module.
SVM	Support Vector Machine.
TCP	Transmission Control Protocol.
TLD	Top Level Domain Name.
TTL	Time to live.
UDP	User Datagram Protocol.
VM	Virtual Machine.
VPN	Virtual Private Network.

Chapter 1 Introduction

1.1 Motivation

Mobile networks are expanding in usage and capacity and devices connected to them are working more and more like regular computers. This makes them able to use Domain Name System (DNS) tunnels to avoid paying for their services similar to how people used it on computers years ago. It is important to try to get ahead of the technology by finding out how to detect DNS tunnels on the mobile network, before the use becomes truly malicious.

Most of the people today have smartphones and tablets to connected to a mobile network, and they have more and more data stored on them. This data can be sensitive for both the user and his company so it is important that it can not be exfiltrated without anyone knowing. As companies tries to keep their private internal network safe from data exfiltration and Command and Control (C&C) attack, they do not think of the information employees has on their mobile devices. With this evolution of the devices and the mobile network does the possibility of data exfiltrate from a device on a mobile network increase.

The evolution of 4G mobile networks and also 5G, which is expected to provide extreme local coverage and capacity, will outperform regular home networks. This evolution will bring more data traffic into the mobile network. This fact combined with the situation where the mobile terminals holds more private and sensitive, or business restricted data than ever before, brings the security issue to a higher focus level.

The safest way to detect a DNS tunnel is to perform Deep Packet Inspection (DPI) which is a time consuming effort and on a mobile network, which cover over millions of people, almost impossible. There has to exist a better way and machine learning might be a good alternative. There are studies which have shown that the statistical properties of a DNS tunnel differ from regular DNS traffic and that is where machine learning excels.

1.2 Objectives

The main objective for this study is to find a way to detect DNS tunnels using machine learning. Machine learning is a great tool for statistical analysis and finding patterns, since DNS tunneling is not the regular use of DNS it should stand out from a pattern based on regular DNS traffic. To find out if it is possible the following objectives were set:

- Find different types of machine learning classifiers which is used for anomaly detection, cluster detection or categorization problems
- Gather data and reformat it so the classifiers can be trained and tested.
- Test the classifiers with different initiation parameters and different features from the dataset.

1.3 Limitations

This study has some limitations. It was not possible to use data gathered from a live network as we would not know if a DNS tunnel would have been used. Running an experiment on a live network to test a model or implement a program with the model was not possible either. The data gathered and used in the experiment is not from a mobile network, but it is good enough for this study.

1.4 Outline

Chapter 2 In this chapter will it be explained how DNS is built up and how it works. It also explains which weaknesses it has which are taken advantages of when performing DNS tunneling. The structure of mobile network is explained and related work are talked about.

Chapter 3 This chapter describes machine learning and the SciKit-learn library in Python, which was used in the experiment. The classifier models and measure techniques used in the experiment are explained in this chapter.

Chapter 4 The set up, data gathering and the results of the experiments are presented in this chapter.

Chapter 5 This chapter concludes the thesis and discusses future work.

Appendix A This appendix contain a snippet of the comma-separated values (csv) file as it were directly from Wireshark

Appendix B Here is a snippet of the reformatted csv file.

Appendix C The script used to reformat the csv file and divide it into training sets and testing sets are in this appendix.

Appendix D This appendix contain the prediction program used in the experiment to train and test the models.

Appendix E In this appendix are tables containing some of the test results.

Chapter 2 Background

To understand how to detect a tunnel, is the understanding of how DNS and DNS tunnels works very important. It is also good to know how the mobile networks work and what makes it different from a regular private network.

2.1 DNS

Since the introduction of the internet has connecting names with Internet Protocol (IP)-addresses been an important part. It is much easier to remember a name than four numbers between 0 and 255. It started out as a `txt` file called `HOST.TXT` stored in the core of the internet. As the number of domains and users grew started the system to encounter problems [MD88]. In the 1980s did the work start on a standardisation of a new system to replace the `HOST.TXT`. The new system became the start of the DNS.

Today DNS is one of the most important backbone components of the internet. It makes users able to use domain names instead of IP-addresses when browsing the internet, sending e-mail or any other interaction with the internet. The DNS will look up the domain name and translate it into an IP-address, e.g. `ntnu.no.` will be translated to `129.241.56.116` which the network needs to route requests and packages back and forth.

DNS is a hierarchy of servers all around the world, which makes it able to maintain fast response time as the number of domains and user grows. Each server either has the response or sends the request to the next server which is lower in the hierarchy. The request is passed on through the hierarchy until it reaches the Authoritative Name Server (ANS) for the domain name requested. It is only ANS nodes who have the information required to response to a request. A domain name consists of multiple domain name labels which is separated by a `"."` and when requested is the lookup resolved from right to left.

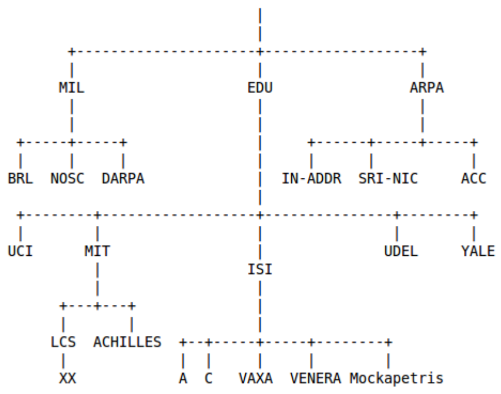


Figure 2.1: Early nodes of the DNS hierarchy tree [Moc87b]

The domain name `achilles.mit.edu.` indicates a path from the root node, which the first `"."` from the right indicates, to the node, ANS, which contains the information about the domain name. In Figure 2.1 can this path be seen. The `achilles` node only contains the information about the domain name `achilles.mit.edu.`. Each label in a domain name represents a domain level. `MIL.`, `EDU.` and `ARPA.` are the Top Level Domain Names (TLDs) in Figure 2.1, the next label, `MIT.EDU.`, is called second level and so on going down the hierarchy tree. Most ANS nodes are located either at the second or third domain level. When the structure is visualised, as in Figure 2.1, is a leaf node an ANS.

When a request is sent, e.g. `www.example.com.` will it first be sent to a root node, which looks up the the TLD `com.`. The root node then sends the request along the correct TLD node which looks up the node for `example.com.`. As the ANS for `example.com.` does this node have the complete knowledge of this domain. It look up the Resource Record (RR) for `www.example.com.` and sends it as a response directly to the user who sent the request. This process is visualized in Figure 2.2. It is only the ANS nodes which have this information and therefore are able to make the response

The RRs stored in the ANS follows a standard [Moc83] containing the following six fields:

– NAME – the domain name of who this record are related to.

– TYPE – what type of record this is.

– CLASS – define the class of the record, usually `IN` for internet.

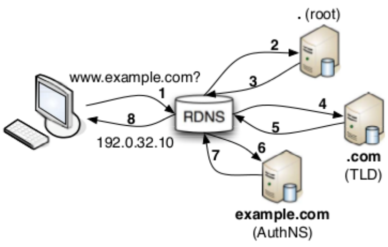


Figure 2.2: DNS lookup process [VH12]

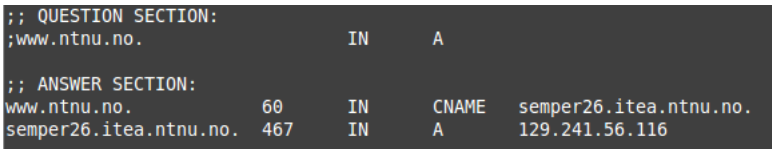
- TTL – an integer which says how long the record should be cached by the server receiving the response. If this field is `0` should the RR only be used in this transaction and not be cached.

- RDLENGTH – Specifies the length of the payload in number of octets.

- RDATA – the payload of the record. The format and length varies depending on the TYPE and CLASS of the RR.

The `TYPE` of a RR has many different uses and therefore different restrictions, which makes some of them easier and better to exploit. The `A` and the `AAAA` `TYPE` are domain names to IP-address RRs, ipv4 and ipv6 respectively. They are the far most used RR `TYPE` as they constitute 63-73% of regular DNS traffic [RDSC$^+$12]. An `A` RR has strict rules for what the `RDATA` can contain, this is also true for an `AAAA` RR. The `CNAME` RR is used to correct the domain name if entered a little wrong, e.g. `www` usually is not a part of the domain name. A request sent for `www.ntnu.no` receives both a `CNAME` response and an `A` response. This is seen in Figure 2.3. `CNAME` responses constitutes 20-30% of regular DNS traffic [RDSC$^+$12]. The `RDATA` field has some more possibilities in a `CNAME` response compared to an `A` and `AAAA`. 1-2% of regular DNS traffic is generated by `TXT` RRs [RDSC$^+$12]. This type of RR is used to store descriptive text on an ANS [Moc87a]. The `RDATA` field is therefore very limitless and can contain character strings. A `NULL` RR was created as a place holder for experimental extensions [Moc87a]. The `RDATA` of this type of RR can contain anything. It is still marked by Internet Assigned Numbers Authority (IANA) as experimental [ian].

The Time to live (TTL) field is an important field both for regular DNS use and for the tunnels. With the numbers of users on the internet, sites has multiple servers at different IP-addresses. Setting the TTL field to a low integer forces the users to request a new RR more often making it easier to load balance between the servers.



```
;; QUESTION SECTION:
;www.ntnu.no.                  IN      A

;; ANSWER SECTION:
www.ntnu.no.            60     IN      CNAME   semper26.itea.ntnu.no.
semper26.itea.ntnu.no.  467    IN      A       129.241.56.116
```

Figure 2.3: Response when running the command dig www.ntnu.no

With that in mind the TTL should not be too low forcing the user to do a DNS lookup for every request as that would overload the ANS.

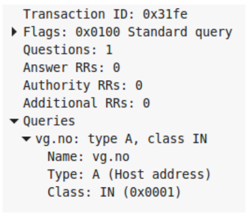
2.2 DNS Tunneling

DNS tunneling started out as a tool to exploit pay-for-service networks at hotels and cafés. It was set up to provide IP-over-DNS which function similar to a Virtual Private Network (VPN). DNS lookup was not a part of the paid service so when a client just sent DNS requests would it be able to access the internet without paying. The user had to set up an ANS with IP-over-DNS software or use a service with ready servers. It is required to control a real domain and have a server with static public IP to set up DNS tunnel server [iod].

DNS tunneling has later been used to perform data exfiltration and C&C attacks, making the attack harder to spot. By using a DNS tunnel for a C&C attack the victim's computer has to send out DNS requests regularly since a DNS response, which would contain the commands, can not be sent to a client without a corresponding request. This creates a large amount of traffic which makes a DNS tunnel more visible. It is therefore one of the downfalls of a DNS tunnel.

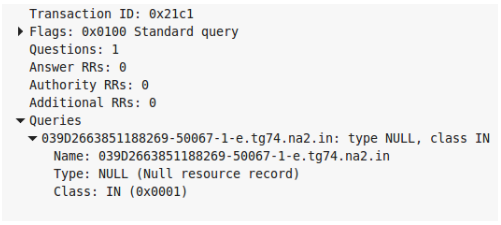
The way a tunnel works is that a client computer sends a DNS request of a controlled domain e.g. GET.vg.no.evilcorp.com. With the hierarchy of the system and how a DNS lookup is performed will this request always reach the same ANS. The ANS of evilcorp.com has DNS tunneling software installed, so it will understand that all the labels to the left of evilcorp.com are a command. It will in this example find the web page of `vg.no` and send as much of the page as possible in the `RDATA` field of the response. The client then sends a new request GET.part2.vg.no.evilcorp.com and the response contains a new part of the web page. This goes on until the whole page is downloaded by the client. The requests and responses are hashed so it is not possible to know exactly what the requests and responses sent are. The difference between a regular and malicious can be seen in Figure 2.4, where Figure 2.4b and Figure 2.4d are malicious request and response respectively and Figure 2.4a and Figure 2.4c are regular request and response respectively.

The `TYPE` of a RR has some important properties based on their regular use which



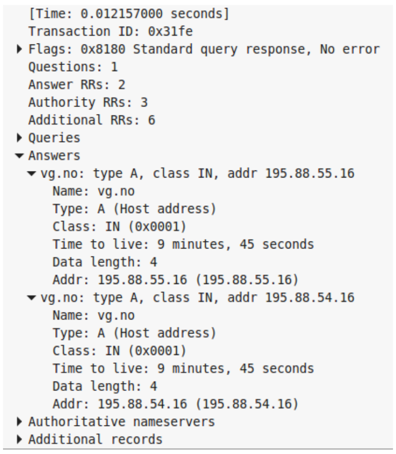
```
 Transaction ID: 0x31fe
▸ Flags: 0x0100 Standard query
 Questions: 1
 Answer RRs: 0
 Authority RRs: 0
 Additional RRs: 0
▾ Queries
  ▾ vg.no: type A, class IN
     Name: vg.no
     Type: A (Host address)
     Class: IN (0x0001)
```

(a) Regular DNS request



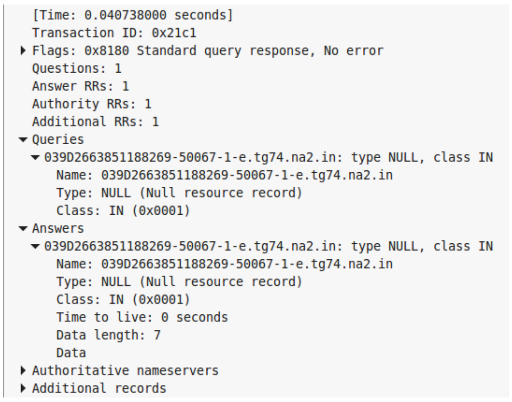
```
 Transaction ID: 0x21c1
▸ Flags: 0x0100 Standard query
 Questions: 1
 Answer RRs: 0
 Authority RRs: 0
 Additional RRs: 0
▾ Queries
  ▾ 039D2663851188269-50067-1-e.tg74.na2.in: type NULL, class IN
     Name: 039D2663851188269-50067-1-e.tg74.na2.in
     Type: NULL (Null resource record)
     Class: IN (0x0001)
```

(b) Malicious DNS request



```
 [Time: 0.012157000 seconds]
 Transaction ID: 0x31fe
▸ Flags: 0x8180 Standard query response, No error
 Questions: 1
 Answer RRs: 2
 Authority RRs: 3
 Additional RRs: 6
▸ Queries
▾ Answers
  ▾ vg.no: type A, class IN, addr 195.88.55.16
     Name: vg.no
     Type: A (Host address)
     Class: IN (0x0001)
     Time to live: 9 minutes, 45 seconds
     Data length: 4
     Addr: 195.88.55.16 (195.88.55.16)
  ▾ vg.no: type A, class IN, addr 195.88.54.16
     Name: vg.no
     Type: A (Host address)
     Class: IN (0x0001)
     Time to live: 9 minutes, 45 seconds
     Data length: 4
     Addr: 195.88.54.16 (195.88.54.16)
▸ Authoritative nameservers
▸ Additional records
```

(c) Regular DNS response



```
 [Time: 0.040738000 seconds]
 Transaction ID: 0x21c1
▸ Flags: 0x8180 Standard query response, No error
 Questions: 1
 Answer RRs: 1
 Authority RRs: 1
 Additional RRs: 1
▾ Queries
  ▾ 039D2663851188269-50067-1-e.tg74.na2.in: type NULL, class IN
     Name: 039D2663851188269-50067-1-e.tg74.na2.in
     Type: NULL (Null resource record)
     Class: IN (0x0001)
▾ Answers
  ▾ 039D2663851188269-50067-1-e.tg74.na2.in: type NULL, class IN
     Name: 039D2663851188269-50067-1-e.tg74.na2.in
     Type: NULL (Null resource record)
     Class: IN (0x0001)
     Time to live: 0 seconds
     Data length: 7
     Data
▸ Authoritative nameservers
▸ Additional records
```

(d) Malicious DNS response

Figure 2.4: Screen dump from Whireshark of regular and malicious DNS packets

makes some of them suitable for DNS tunneling and others impossible to use. The `RDATA` field of an `A` type RR has to contain an ordinary 32 bit internet address only using 4 octets, while it has no restriction other than a maximum size of 65,535 octets in a `NULL` type RR [Moc83]. This means it is possible to send a much larger message containing every possible character as the payload in a `NULL` type RR, which makes it much more suitable for DNS tunneling. The `RDATA` field in a `TXT` record type can also hold any character string, with little limitation to size, which also makes this suitable for malicious use.

Each domain name label has a limit of 63 characters, and the total length of a domain name can not exceed 255 characters. This limits the amount of data a client can send per request, making the stream of data up, from client to server, quite slow. The downstream on the other hand can be faster as the payload can be quite large, as mentioned earlier. The problem here is that normal DNS traffic is transferred over User Datagram Protocol (UDP) on which it is a limit of 512 bytes per packet [Far13]. DNS can use Transmission Control Protocol (TCP) on payloads larger than 512 bytes, but this will stand out and be easily detectable. Even so has speeds of

110 KB/s with delays down to 150 ms been measured when tunneling TCP packets in a DNS tunnels [vLCL08]. At that speed were the DNS traffic up 2000% compared to normal, which is an indication of how to detect the tunnels.

2.3 Mobile Network

In the mobile network goes all internet traffic through the Gateway GPRS Support Node (GGSN). This is also where the DNS is resolved in the network. When a connection is established is the packages tunnelled through the mobile network in GPRS Tunnelling Protocol (GTP) tunnels from the Serving GPRS Support Node (SGSN) to the GGSN before it is routed as regular internet traffic. In a mobile network is it normal to have multiple GGSNs, but they are mainly for redundancy not for traffic management [run]. All of the data traffic goes through one GGSN. This means that the data traffic for millions of users go through one GGSN. Going through all that traffic looking for a DNS tunnel is almost like looking for a needle in a haystack. Even though DNS tunnels generate a large amount of traffic, will it not be very noticeable since so few do it.

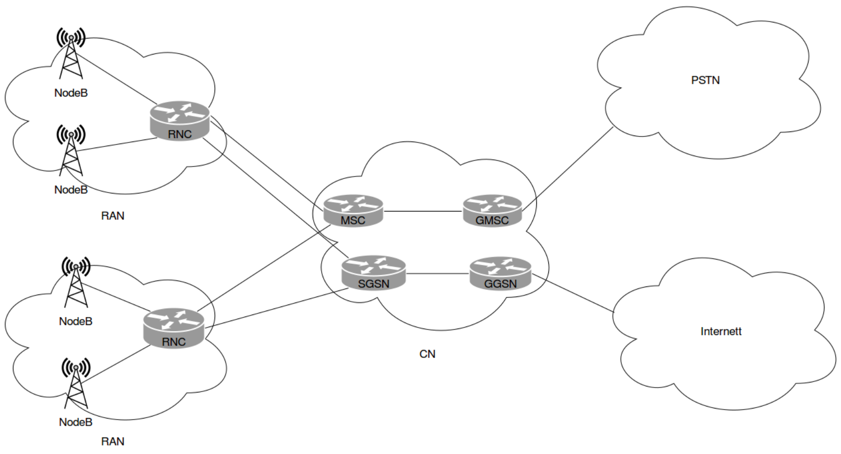


Figure 2.5: Cellular network structure for 3G

2.4 Related Work

Detecting DNS tunnels has been studied more and more since the early 2000s with different results. No fully functioning software or algorithm is yet available. In 2012 Rasmussen said 'most enterprises are wide-open to real attacks via this little-known vector' and 'there is little to no protection against them on most networks' [ras]. This is still a reality.

There are studies comparing different applications and methods to detect the tunnels, and the results seems to be quite similar. The main way to detect a DNS tunnel is with payload analysis, DPI [Far13, dnsa], which is slow and therefore not a great solution on a large network. DNS tunneling software are also being updated and new ones are created faster than the detection researches are able to find a good way to detect them. When using DPI is it possible to to detect a tunnel by looking at the type of record sent. Regular DNS traffic consist mostly of A, AAAA or CNAME, while tunnels wants to get maximum out of the packets and therefore use NULL or TXT [dnsa, dnsb] which allows larger and less structured `RDATA`.

The size of the a packet is also a way of detecting a tunnels as malicious use would want to maximise the load and therefore extend the label. It will try to use up to 63 characters per domain name label and the total domain name as close to the limit of 255 characters [Far13, dnsa, dnsb]. Following this, it is recommended to do a DPI of all packets where the domain name requests are longer than 52 characters [Far13]. The DPI can be combined with character frequency analysis to detect tunnels [BG10].

Chapter 3

Machine Learning

3.1 The Basics

When analysing large sets of data, machine learning is a great tool. It is algorithms which are trained to find patterns in datasets and categorizes them. There exists lots of different algorithms which serves different purposes and are specialized to solve different problems. These algorithms are called classifiers and are the base of a machine learning model. A model has to be trained and tested on datasets before they can be put into use. This is done either with or without labeled data, supervised or unsupervised respectively, indicating the correct category of the entry. If the model passes the tests with acceptable results is it ready to be put into use, if not must either the parameters of the model be changed or a different classifier has to be used.

Machine learning is not the same as Artificial Intelligence (AI). Where AI continuously learn over time, is a machine learning model set after it has been trained. If the model has to be updated, must a new model be trained with new parameters or with a new classifier algorithm.

3.2 SciKit-Learn

SciKit-Learn is a library for Python which contains functions to create machine learning classifiers and support for training and testing them. It contains a large selection of classifier models, where only the initial parameters have to be set. The library also contains functions to measure how well the models work. With the preference of coding in Python and some research, was this library chosen for this study. Python is not the fastest language when compiled, but for this study the time to run a test was not an issue. The main SciKit-learn modules used in this study is the classifiers One-Class SVM (OCSVM) and K-means and the metrics class to measure the results.

3.2.1 Classifiers

One Class SVM

Support Vector Machine (SVM) is a term used for multiple methods of machine learning. They are binary algorithms meaning their result is either in or out, used in classification and outlier detection. When used in classification will it say that the data is either in a category or not, and then traverse through the categories until either one fit or there are no more. The training is performed on both positive and negative examples, but the OCSVM extension made it possible to only use positive data in the training process [MY02].

The OCSVM is of great use in novelty detection where training is performed on a dataset only containing positive examples. It creates a boundary based on this data and when receiving new data sees if these are within those boundaries when predicting. In SciKit-learn OCSVM is implemented to run unsupervised, but with the knowledge that the dataset for training contains only, or at least mostly, positive data. It labels the data with `1` or `-1` for positive (inlier) or negative (outlier) respectively [out]. The OCSVM requires a kernel when initialized, and SciKit-learn has implemented four which are reay to be used. Those are linear, polynomial, Radial Basis Function (RBF) and sigmoid. It is also possible to use custom kernels, but it is not used in this study see [svm] for more information.

Algorithm 3.1 Algorithms used by the different kernels in OCSVM [svm]

$$\begin{aligned} Linear &= \langle x, x' \rangle \\ Polynomial &= (\gamma \langle x, x' \rangle + r)^d \\ RBF &= exp(-\gamma |x - x'|^2) \\ Sigmoid &= (tanh(\gamma \langle x, x' \rangle + r)) \end{aligned}$$

In algorithm 3.1 are the algorithms used by the different kernels. The γ is set by the parameter `gamma`, the `d` by `degree` and the `r` by `coef0` when the kernels are initiated. Setting these for the kernel linear will have no effect. The default value of degree is 3, coef0 is 0.0 and gamma is 'auto', which means $1/n\_features$ where `n_features` is the number of features in the dataset. The gamma value marks the area around a support vector which should be interpreted as part of the vector. If the value is low will a large area around the vector be used, and a high almost no extra area. This is a sensitive parameter and can have a large impact on the result. The parameters degree and coef0 does not effect the results as much. One parameter they all use are the `nu` parameter which indicates a upper bound on the percentage of training errors and a lower bound of support vectors used. The default value is 0.5 which means that the model is finished training if it has used atleast 50% of the support vectors and the training errors are under 50%.

K-Means

K-means is an algorithm used to categorize data in `n` clusters. It works in three steps:

1. Choose an initial centroid for each cluster.
2. Assign each sample to the nearest centroid.
3. Create new centroid with the value calculated from the mean of all samples to their nearest centroid.

The algorithm loops between point 2 and 3 until the distance between the new and the old centroid is lower than a given value. This value is given as a parameter called `tol`, which default is 0.0001. The `init` parameter of K-means is how the initial centroids is set. SciKit-learn has programmed two methods, `k-means++` and `random`, which could be used, or the user can enter an ndarray [kmeb]. The `k-means++` is an algorithm which tries to choose the the initial centroids in such a way that the convergence is sped up. `Random` simply chooses `n` points from the data at random to be the initial centroids, where `n` is the number of clusters. If a custom ndarray is used must it contain `n` points which are in the same space as the data, meaning it has to have the same number of features as the dataset used. With a class in SciKit-learn called `decomposition` are there functions which could help create good points for a K-means model to start. No matter how your point are initialized with enough time will the algorithm converge, at least to a local minimum.

The distance from each point to their assigned centroid are called `inertia`, or within-cluster sum-of-squares. Even though it is how the algorithm test it self to see how good the result is, is the metric in itself not very good. It is known that zero is the optimal value, but it is not normalized. In high-dimensional, multiple feature, spaces is it almost impossible to get the inertia to zero [kmea].

3.2.2 Metrics

The metrics class in SciKit-Learn contains many functions to evaluate classifiers. Different classifiers are measured based on different criteria. For a outlier and categorization classification are the normal way to determine the success of a classifier by measuring `precision`, `recall` and `F-score`. The precision of a classifier determine the percentage of the elements selected are true positives and the recall determines the percentage of the relevant elements was selected. This is explained in Figure 3.1.

The F-score measurement is derived from both precision and recall and gives a result which better represents the overall character of the classifier. The closer to 1 the

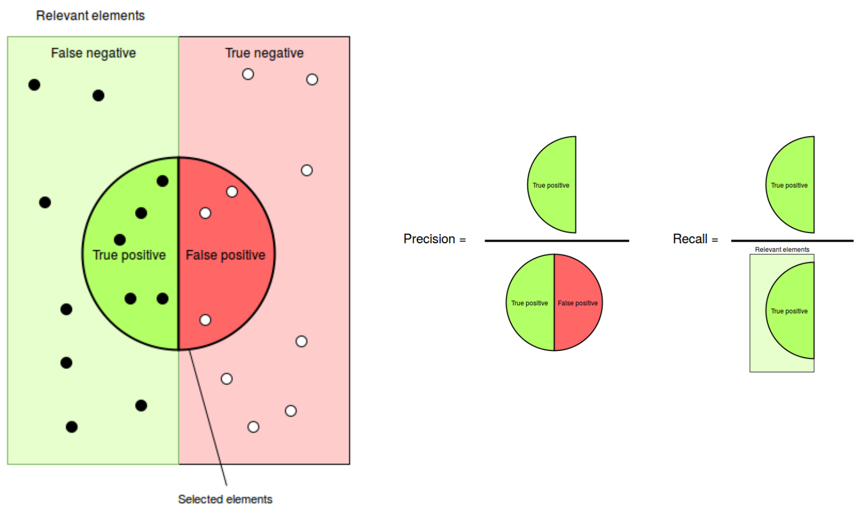


Figure 3.1: Venn diagram explaining precision and recall

F-score value is indicates that both precision and recall are high, and the classifier is working well [VH12]. How to calculate the precision, the recall and F-score in algorithm 3.2.

Algorithm 3.2 The algorithms for calculating precision, recall and F-score

$$precision = \frac{True\ positives}{Selected\ elements} = \frac{Selcted\ elements\ \cap\ Relevant\ elements}{Selected\ elements}$$

$$recall = \frac{True\ positives}{Relevant\ elements} = \frac{Selected\ elements\ \cap\ Relevant\ elements}{Relevant\ elements}$$

$$F = 2 * \frac{Precision\ *\ Recall}{Precision\ +\ Recall}$$

Chapter 4 Experiment

4.1 Setup

To gather data for testing the machine learning models, was the plan to use OpenGGSN from Osmocom [ope]. This is an opensource software developed as a part of a complete opensource mobile network. The distribution of OpenGGSN also contains a sgsnemu program which is used to emulate a SGSN setting up a GTP tunnel to the GGSN. This program has earlier been used to improve the security of the core mobile network [Dim07] and in a man-in-the-middle attack setting up a fake Base Transceiver Station (BTS) connected to the internet [PP11]. Using OpenGGSN therefore seemed like good idea, but setting it up would turn out to be a real problem. The software is poorly documented, and we were not able to get the connection between the GGSN and the SGSN software to work properly. The best way to set it up seemed to be as part of a complete mobile network setup [bscb], which require hardware not available for this thesis. Other problems with this are that BTS hardware is required to be registered to the public Groupe Spécial Mobile (GSM) frequency spectrum. It also requires the use of special Subscriber Identity Module (SIM) cards [bsca]. The plan of how to gather data had to be revised and a new solution had to be found.

The new solution was to gather data using Oracle VirtualBox and Wireshark. Four Virtual Machines (VMs) were created, one running Android 4.4 and the three others running Ubuntu 14.04. On the Android VM was a DNS tunneling application called `Slow DNS` installed. The DNS tunnel was established on the Android before browsing the internet and the others browsed the internet regularly. The data was then gathered by Wireshark capturing all the DNS traffic these VMs generated.

4.2 Reformatting

The DNS packets were filtered out in Wireshark and saved as a csv file, a snippet of this can be seen in appendix A. Each line in the csv file contained the meta data for

Time	Source	Destination	Protocol	LengthUp	LengthDown	Info	Label
0.021829999999997796	192.168.1.60	192.168.1.1	DNS	89	276	Standard query 0x0cf7 A safebrowsing-cache.google.com	1
0.5176109999999987	192.168.1.14	192.168.1.1	DNS	209	274	Standard query 0x3a8c NULL 149N2546851188122-246-109-MHoQF3dk88nOvvCbaPdyeLvknsPKAAAAdgAbh.u4KADQAMgAOAA0AGQALAAwAGAAJAAoAFgAXAAgABgAHABQAFQAEAAUAEgATA.AEAAgADAA8ie.tg16.m7q.in	-1

Table 4.1: A line with regular DNS traffic and one with malicious traffic

one packet, with the features:

– No. – Packet number since the start of the capture.

– Time – Time elapsed since the capture started.

– Source – The source IP-address of the packet.

– Destination – The destination IP-address of the packet.

– Protocol – What kind of protocol does the packet belong to.

– Length – The size of the packet.

– Info – A description of the content of the packet.

The raw data had to be reformatted to be used for the machine learning models. To do this a python script were used, see appendix C. It went through the csv file to find the response to each request and creating a new csv file. The features in the new file was `Time, Source, Destination, Protocol, LengthUp, LengthDown, Info, Label`. The `Time` feature now was the time between the request and response, not the time since the capture started. `LengthUp` was the size of the request and `LengthDown` the size of the response. The feature `Label` was 1 or -1 for regular or malicious packet respectively. Table 4.1 shows how two lines of the new csv file looks like, one line with regular DNS traffic and one with malicious traffic. Reformatting the files by combining the request and response packets reduced the number of entries with 50%, making the new file easier to use and understand. A snippet of the reformatted csv file is in appendix B.

Reducing the number of malicious DNS packets was necessary to be more representative of the real world. Without any reduction of the malicious packets, did they accounted for over 50% of all the DNS traffic. This was in a small environment only consisting of four users, while in a normal mobile network can it be millions of users connected to one GGSN as mentioned in section 2.3. Without reducing would the dataset represent that one in four mobile devices used DNS tunnels. The amount of malicious packets were therefore reduced down to 7% of the data set. Reducing the number of malicious packets lower would result in too few packets for the testing of the models. The total dataset consisted of 1,124 malicious data entries and 13,690 regular data entries, a total of 14,814.

To be able to test the models had the dataset to be divided into pairs of training and testing sets. Models based on the OCSVM classifier should be trained on dataset only containing "good" data, meaning data that should be categorized as inliers [out]. So for training and testing OCSVM a training set containing around 75% of the "good" data was created and testing set containing the remaining 25% and all of the malicious data. K-means on the other hand needs data of both categories to be able to find a centroid for each of them. So for that model was a test set containing 75% of the malicious and 75% of the good data created, and a training set containing the remaining 25% of both types of data created.

4.3 Results

4.3.1 One-Class SVM

Testing the model is done by letting the model predict if the input is an inlier or an outlier. The results of these predictions are compared to an array containing the true label of each input with a function called `classification_report` from the metrics class. This functions calculates the precision, recall and F-score for each model. In this test is the `precision` how many percent of the predicted outliers was `True positives` and the `recall` is how many percent of outliers were predicted as outliers. The `f1-score` is a way of measuring a total based on the precision and recall, mentioned in section 3.2.2. The test program is seen in appendix D. Previous test has revealed that these models tends to work better on scaled data, so all the tests were executed with scaled data. The data was scaled with a function `scale` from the class `preprocessing` in SciKit-learn.

OCSVM has multiple kernels, and each kernel different parameters which affect the results of the model. So the first test was executed with each kernel with all other parameters at `default`. This was to decide which should be focused on. Each model was trained on the same set of data, which contains 75% of the good data, and then tested on the dataset containing all the malicious data and the remaining 25% of the good data.

Table 4.2 shows the results of the test and it is clear to see that the `poly` kernel had great results. More tests were executed changing the features used and the parameter `nu` of the models. Only changing the nu is because this is the only parameter that effects every kernel, see section 3.2.1. In appendix E is some of the results from these tests. After these tests did the kernel `RBF` show the best results. The RBF got a total `f1-score` of 0.93, with some small changes to the `nu` parameter. On the other hand

		precision	recall	f1-score	support
Kernel = rbf	Outlier	0.40	1.00	0.57	1124
	Inlier	1.00	0.51	0.67	3394
	avg / total	0.85	0.63	0.65	4518
Kernel = sigmoid	Outlier	0.25	1.00	0.40	1124
	Inlier	0.00	0.00	0.00	3394
	avg / total	0.06	0.25	0.10	4518
Kernel = linear	Outlier	0.24	0.94	0.38	1124
	Inlier	0.57	0.03	0.05	3394
	avg / total	0.49	0.25	0.14	4518
Kernel = poly	Outlier	0.86	0.94	0.90	1124
	Inlier	0.98	0.95	0.96	3394
	avg / total	0.95	0.95	0.95	4518

Table 4.2: Classification report for OCSVM with different kernels and default parameters using the features LengthUp, and LengthDown from the dataset.

	precision	recall	f1-score	support
outliers	0.87	0.98	0.92	1124
inliers	0.99	0.95	0.97	3428
avg / total	0.96	0.96	0.96	4552

Table 4.3: Classification report for OCSVM with the RBF kernel, nu=0.05 and gamma=0.01. The features used was LengthUp and LengthDown

did the results for the poly kernel become worse when the parameter and features were change. This lead to more tests on the RBF kernel, to see if it could surpass the 95% f1-score that the poly kernel got with default parameters.

The parameters `nu` and `gamma` have the most effect on the results of the RBF kernel, see section 3.2.1. More tests were executed, changing these parameters up and down. Some of the test results are in appendix E. The final result was a f1-score of 96% shown in table 4.3, with `nu` set to 0.05 and `gamma` set to 0.01. The recall on outliers had suffered a little compared to some of the previous tests, but the precision of the outliers was much higher than on any other test executed with the RBF kernel. This shows how much the parameters has to say when configuring a machine learning model.

The model with RBF kernel can be represented as a graph which is shown in Figure 4.1. Each dot in the figure is a data entry. The white ones are from the training set and green and red are from the test set, where green is inliers and red

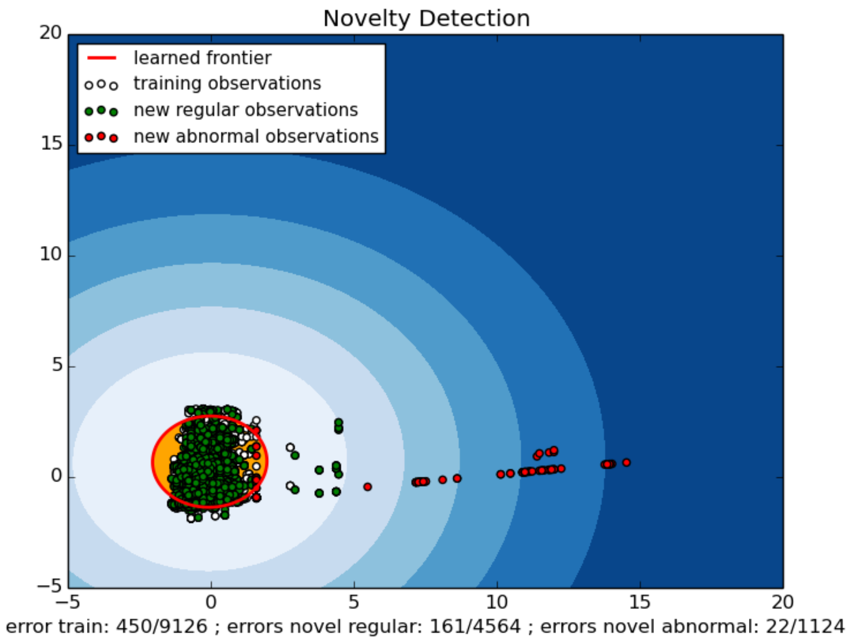


Figure 4.1: Graph presenting all data points used both in the training phase and testing phase and the decision function the OCSVM created with RBF as kernel, nu=0.05 and gamma=0.01

are outliers. Inside the red circle is the area the model defines as inlier area. This figure shows that malicious data has some points near and inside the area, which means that some malicious data has properties quite similar to regular DNS traffic. There are also regular DNS traffic quite far out from the learned area, meaning there are irregularities in regular traffic.

4.3.2 K-Means

The K-means classifier was also tested for which it is three main different initiation methods, either `k-means++`, `random` or by using an `ndarray`, see section 3.2.1. The `ndarray` used in the testing is the result of a Principal Component Analysis (PCA) decomposition. That is a function which creates an array containing the most significant singular vectors of the data [pca]. K-means needs to have elements of both categories when training, so the data is not the same as for OCSVM although it is from the same main dataset.

The result of the initial test, where the data used was scaled similar to the tests of

		precision	recall	f1-score	support
init = ndarray	Outlier	0.14	0.99	0.24	286
	Inlier	1.00	0.48	0.65	3441
	avg / total	0.93	0.52	0.62	3727
init = k-means++	Outlier	0.14	0.99	0.25	286
	Inlier	1.00	0.50	0.66	3441
	avg / total	0.93	0.53	0.63	3727
init = random	Outlier	0.00	0.01	0.00	286
	Inlier	0.86	0.50	0.64	3441
	avg / total	0.79	0.47	0.59	3727

Table 4.4: Classification report for K-means models with different init values, using the features LengthUp, and LengthDown from the dataset.

OCSVM, is shown in table 4.4. All the initiation methods have a quite even total f1-score, but looking at each line reveals a weakness in that measurement. With `init` set to `random` is the model almost not able to predict any outlier, with recall at 1% and both precision and f1-score at 0%.

More tests were executed and some of the results are located appendix E. The models were trained and tested with unscaled data, different features and different number of features. The best results had a f1-score total of 100%, but then one of the features was the source IP-address and in this dataset only one source were used to produce DNS tunneling data.

The models with k-means++ and random initiation has one more parameter which is interesting to look at which is the `n_init`. This is a parameter which states how many times the model shall set the initial centroids. If 10 the models runs 10 times with different initial centroids for each time and then returns the model where the `inertia` is lowest. With the ndarray is this not possible since a value for the initial centroids are given and can not be change between initiations, the `n_init` is therefore always 1. It drastically increased the time it took to test, but the changes did not help the result.

Chapter 5 Conclusion

5.1 Summary

The purpose of this study was to see if it was possible to make good detection program for detection of DNS tunnels in mobile networks based on machine learning. The data was suppose to be gathered from a GGSN using openGGSN. Unfortunately we were not able to get the openGGSN to run properly. The data was instead gathered using Oracle VirtualBox to run VMs in and Wireshark to capture the DNS traffic. This data is comparable to data captured from a GGSN.

The results show that the OCSVM classifier is supreme compared to the K-means for this problem. K-means is a cluster classifier and works best when the clusters are even. It did manage to sort out the uneven clusters, but only when given the data had clear indication. The problem is that the data it will be used on does not have labels or is only sent by one user. There are cluster classifiers which are more versatile and works better on uneven clusters. Some of those were tested in this study, but they did not support the size of the dataset used. As this is a small scale test, they would not be able to work in a real mobile network.

OCSVM gave great results with the `poly` kernel with default parameters, and with the RBF kernel with both `gamma` and `nu` parameters changed. As the poly kernel only seemed to work with the default parameters and with two features from the dataset, does it seem to be quite unstable and might not be the best to use in a real network. The RBF kernel had a recall of close to 100% on the outliers in nearly all the tests, which means it was able to categorize all the outliers correct. This is important for a detection program to be able to do. The weakness of the model was the precision of outliers and recall of inliers, which means it produced some false positives. By working with the initiation parameter of the model it was possible to get the number of false positives down, which resulted in a good model to base an implementation on.

This study shows that it is possible to use machine learning to detect DNS tunnels. The best and most versatile solution is the OCSVM classifying algorithm with the RBF kernel which is a model used for novelty and outlier detection. The feature `time` which was the time between request and response did not seem to do much as the results with and without it was equal to each other. Length of the requested domain name is directly connected to the the size of both request and response. The results did not change when this feature was used or not. The features which gave the best results was the size of the request and response, which therefore is recommended for future studies.

5.2 Future Work

The results of this study has been good, and detecting DNS tunnels in the mobile network will only become more important as the amount of devices connected increases. Since this study was not able to use data from openGGSN, would a study based on data gathered from a GGSN be interesting to look at. It will be of specific interest to evaluate whether data is similar enough to simply copy the model set up or if the parameters have to change or different features have to be used. The model also needs to be tested on a larger dataset. This study was not able to produce a large enough dataset to really represent a mobile network regarding the number of users. To drop the percentage of malicious DNS traffic to a representing amount would result in a too small amount of data to be able to run tests.

Further areas of studies is to use this model to create a program which flags traffic assumed to be malicious and test it in live traffic. This could be done in small scale initially, by setting up a private BTS. This should be followed up with a large scale test where the program runs in a live mobile network.

It is also interesting to study if this model could be used in detecting DNS tunnels in smaller networks as well, e.g. a company network.

The most important to study further is how this model will work on a larger dataset. The dataset used in this study contained only 14814 entries, which is a bit small. Specifically when representing a mobile network where the number of users are in the millions, and the number of malicious users are only in the hundreds.

References

[BG10] Kenton Born and David Gustafson. Detecting dns tunnels using character frequency analysis. *arXiv preprint arXiv:1004.4358*, 2010.

[bsca] Hofmeyr, Neels from Osmocom. E-mail correspondence.

[bscb] Openbsc gprs/edgne setup page. http://openbsc.osmocom.org/trac/wiki/OpenBSC_GPRS. Accessed: 2016-06-09.

[Dim07] Christos K Dimitriadis. Improving mobile core network security with honeynets. *IEEE Security & Privacy*, (4):40–47, 2007.

[dnsa] Dns tunnelling. http://resources.infosecinstitute.com/dns-tunnelling/. Accessed: 2016-02-14.

[dnsb] Dnscat. http://tadek.pietraszek.org/projects/DNScat/. Accessed: 2016-05-21.

[Far13] Greg Farnham. Detecting dns tunneling. *InfoSec Reading Room*, 2013.

[ian] Domain Name System (DNS) Parameters. http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml. Accessed: 2016-05-30.

[iod] How to setup iodine. http://dev.kryo.se/iodine/wiki/HowtoSetup. Accessed: 2016-02-14.

[kmea] Clustering, k-means. http://scikit-learn.org/stable/modules/clustering.html#k-means. Accessed: 2016-06-09.

[kmeb] K-means. http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans. Accessed: 2016-06-09.

[MD88] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 123–133, New York, NY, USA, 1988. ACM.

[Moc83] Paul V Mockapetris. Domain names: Implementation specification. 1983.

[Moc87a] Paul Mockapetris. Rfc 1035—domain names—implementation and specification, november 1987. *URL http://www. ietf. org/rfc/rfc1035. txt*, 1987.

[Moc87b] Paul V Mockapetris. Domain names-concepts and facilities. 1987.

[MY02] Larry M Manevitz and Malik Yousef. One-class svms for document classification. *the Journal of machine Learning research*, 2:139–154, 2002.

[ope] Openggsn. http://cgit.osmocom.org/openggsn/. Accessed: 2016-02-20.

[out] Novelty and outlier detection. http://scikit-learn.org/stable/modules/outlier_detection.html. Accessed: 2016-05-21.

[pca] Pca. http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Accessed: 2016-06-09.

[PP11] David Perez and Jose Pico. A practical attack against gprs/edge/umts/hspa mobile data communications. *Black Hat DC*, 2011.

[ras] Do you know what your dns resolver is doing right now? http://www.securityweek.com/do-you-know-what-your-dns-resolver-doing-right-now. Accessed: 2016-05-21.

[RDSC+12] Daan Raman, Bjorn De Sutter, Bart Coppens, Stijn Volckaert, Koen De Bosschere, Pieter Danhieux, and Erik Van Buggenhout. Dns tunneling for network penetration. In *Information Security and Cryptology–ICISC 2012*, pages 65–77. Springer, 2012.

[run] Skow, Rune from Concept. Personal communication.

[svm] Support vector machines. http://scikit-learn.org/stable/modules/svm.html. Accessed: 2016-06-09.

[VH12] Linh Vu Hong. Dns traffic analysis for network-based malware detection. 2012.

[vLCL08] Tom van Leijenhorst, Kwan-Wu Chin, and Darryn Lowe. On the viability and performance of dns tunneling. 2008.

Appendix A

Original csv file

```
"No.","Time","Source","Destination","Protocol","Length","Info"
"1","0.000000000","192.168.1.1","192.168.1.14","DNS","164","Standard query response 0xee9f NULL 039D2546851188122-52087-1-e.tg16.nf5.in"
"2","0.052876000","192.168.1.1","192.168.1.14","DNS","365","Standard query response 0xea0e NULL 156N2546851188122-191-116-MImis7wFip533GTGFqPSt8O9SJuQM6AAAAPwA.AAAD6RQAANEnIQABABmpZrBcdTqzZEGSV7gG78d9RNG7VpJ2AEAHJ9osAAAE.BCAoABhlj3N0TTgieie.tg16.po0.in"
"3","0.054648000","192.168.1.14","192.168.1.1","DNS","214","Standard query 0x4861 NULL 154N2546851188122-194-114-MAqEEXmef4hsq54msV1LOjKH7WOTAAAAQgAAA.AD6RQAANEnKQABABmpXrBcdTqzZEGSV7gG78d9R9W7VpJ2AEAHJ9MwAAAEBC.AoABhph3N0TTgieie.tg16.qv4.in"
"4","0.062105000","192.168.1.14","192.168.1.1","DNS","99","Standard query 0xdaa7 NULL 039D2546851188122-52088-1-e.tg16.g6h.in"
"5","0.070010000","192.168.1.1","192.168.1.14","DNS","242","Standard query response 0xc889 NULL 117N2546851188122-193-79-MIRD67xlksXPs3bJnT2YLipt2wlk2AAAAQQAbh.soFAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.11v.in"
"6","0.093659000","192.168.1.1","192.168.1.14","DNS","279","Standard query response 0x4861 NULL 154N2546851188122-194-114-MAqEEXmef4hsq54msV1LOjKH7WOTAAAAQgAAA.AD6RQAANEnKQABABmpXrBcdTqzZEGSV7gG78d9R9W7VpJ2AEAHJ9MwAAAEBC.AoABhph3N0TTgieie.tg16.qv4.in"
"7","0.102736000","192.168.1.1","192.168.1.14","DNS","164","Standard query response 0xdaa7 NULL 039D2546851188122-52088-1-e.tg16.g6h.in"
```

```
"8","0.138089000","192.168.1.14","192.168.1.1","DNS","176","
    Standard query 0x4d88  NULL 116N2546851188122-196-78-
    MAYouN36sEoPlm180Vm6dvKSfAr8AAAARAAbhs.4
    FAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.mm4.in"
"9","0.162943000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0xb452  NULL 039D2546851188122-52089-1-e.
    tg16.06x.in"
"10","0.176664000","192.168.1.1","192.168.1.14","DNS
    ","241","Standard query response 0x4d88  NULL 116
    N2546851188122-196-78-
    MAYouN36sEoPlm180Vm6dvKSfAr8AAAARAAbhs.4
    FAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.mm4.in"
"11","0.201363000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0xb452  NULL 039
    D2546851188122-52089-1-e.tg16.06x.in"
"12","0.264901000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0xb23a  NULL 039D2546851188122-52090-1-e.
    tg16.8uy.in"
"13","0.303419000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0xb23a  NULL 039
    D2546851188122-52090-1-e.tg16.8uy.in"
"14","0.365518000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x777e  NULL 039D2546851188122-52091-1-e.
    tg16.z84.in"
"15","0.404028000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0x777e  NULL 039
    D2546851188122-52091-1-e.tg16.z84.in"
"16","0.465739000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0xdb98  NULL 039D2546851188122-52092-1-e.
    tg16.88j.in"
"17","0.473949000","192.168.1.14","192.168.1.1","DNS
    ","177","Standard query 0x0e0d  NULL 117N2546851188122
    -198-79-MMXStyCGLvtxKdj5juASaRoZiiuVaAAAARgAbh.
    tIFAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.g6h.in"
"18","0.504310000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0xdb98  NULL 039
    D2546851188122-52092-1-e.tg16.88j.in"
"19","0.512515000","192.168.1.1","192.168.1.14","DNS
    ","242","Standard query response 0x0e0d  NULL 117
    N2546851188122-198-79-
    MMXStyCGLvtxKdj5juASaRoZiiuVaAAAARgAbh.
    tIFAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.g6h.in"
"20","0.566344000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x2288  NULL 039D2546851188122-52093-1-e.
    tg16.bn3.in"
"21","0.604844000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0x2288  NULL 039
    D2546851188122-52093-1-e.tg16.bn3.in"
"22","0.678283000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0xf902  NULL 039D2546851188122-52094-1-e.
    tg16.qv4.in"
"23","0.716718000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0xf902  NULL 039
    D2546851188122-52094-1-e.tg16.qv4.in"
"24","0.765526000","192.168.1.14","192.168.1.1","DNS","75","
    Standard query 0xbc4c  A mail.google.com"
"25","0.789699000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x3a7a  NULL 039D2546851188122-52095-1-e.
    tg16.na2.in"
"26","0.799816000","192.168.1.1","192.168.1.14","DNS
    ","254","Standard query response 0xbc4c  CNAME googlemail
    .l.google.com A 216.58.209.101"
"31","0.828771000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0x3a7a  NULL 039
    D2546851188122-52095-1-e.tg16.na2.in"
"46","0.901398000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0xded7  NULL 039D2546851188122-52096-1-e.
    tg16.mm4.in"
"47","0.939638000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0xded7  NULL 039
    D2546851188122-52096-1-e.tg16.mm4.in"
"48","1.012849000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x9b25  NULL 039D2546851188122-52097-1-e.
    tg16.nf5.in"
"49","1.051220000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0x9b25  NULL 039
    D2546851188122-52097-1-e.tg16.nf5.in"
"60","1.124293000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x61cd  NULL 039D2546851188122-52098-1-e.
    tg16.g6h.in"
"61","1.141586000","192.168.1.14","192.168.1.1","DNS
    ","178","Standard query 0x6efe  NULL 118N2546851188122
    -200-80-MGMqCPzHzH3MisoaUCldDiisXAusxaAAAASAAb.
    htYFAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.8uy.in"
"63","1.162566000","192.168.1.1","192.168.1.14","DNS
    ","164","Standard query response 0x61cd  NULL 039
    D2546851188122-52098-1-e.tg16.g6h.in"
"64","1.179858000","192.168.1.1","192.168.1.14","DNS
    ","243","Standard query response 0x6efe  NULL 118
    N2546851188122-200-80-
    MGMqCPzHzH3MisoaUCldDiisXAusxaAAAASAAb.
    htYFAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.8uy.in"
"65","1.235769000","192.168.1.14","192.168.1.1","DNS","99","
    Standard query 0x82b5  NULL 039D2546851188122-52099-1-e.
    tg16.06x.in"
```

appendixCSV/Unformatted.csv

Appendix B Reformatted csv file

```
Time,Source,Destination,Protocol,LengthUp,LengthDown,Info,Label
0.03972300000000175,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x82b5 NULL 039D2546851188122-52099-1-e.tg16.06x.in,1
0.6289750000000003,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0xeade NULL 039D2546851188122-52105-1-e.tg16.mm4.in,1
0.19516800000000023,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x0ac4 NULL 039D2546851188122-52109-1-e.tg16.8uy.in,1
0.038536999999999982,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x1891 NULL 039D2546851188122-52113-1-e.tg16.na2.in,1
0.04226600000000058,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0xdd58 NULL 039D2546851188122-52129-1-e.tg16.z84.in,1
0.03849400000000003,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x4c0a NULL 039D2546851188122-52136-1-e.tg16.g6h.in,1
0.045115999999999927,192.168.1.14,192.168.1.1,DNS,176,241,Standard query 0xd1c2 NULL 116N2546851188122-208-78-MFaWhRUf5xooQ4W4gjn5YulOu6h1AAAAUAAbht.4FAwQBBAIEAwMBAwIDAwIBAgICAwEBM3QAAAieie.tg16.jk0.in,1
0.15343900000000055,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0xaf46 NULL 039D2546851188122-52147-1-e.tg16.06x.in,1
0.10595200000000027,192.168.1.60,192.168.1.1,DNS,76,290,Standard query 0xb0d0 A start.ubuntu.com,0
```

```
0.0652040000000000137,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x609e  NULL 039D2546851188122-52226-1-e.tg16.nf5.in,1
0.0721220000000000024,192.168.1.60,192.168.1.1,DNS,75,136,Standard query 0xb947  AAAA help.ubuntu.com,0
0.1067440000000000261,192.168.1.60,192.168.1.1,DNS,74,135,Standard query 0xcc73  AAAA www.ubuntu.com,0
0.0387170000000000189,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0xdeb5  NULL 039D2546851188122-52255-1-e.tg16.g6h.in,1
1.181477000000001,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x4aea  NULL 039D2546851188122-52259-1-e.tg16.88j.in,1
1.0046719999999993,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x472e  NULL 039D2546851188122-52261-1-e.tg16.qv4.in,1
0.0250419999999999912,192.168.1.60,192.168.1.1,DNS,89,288,Standard query 0x0107  AAAA safebrowsing-cache.google.com,0
0.5029949999999985,192.168.1.60,192.168.1.1,DNS,92,216,Standard query 0x9a19  AAAA tiles-cloudfront.cdn.mozilla.net,0
0.7623779999999982,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x520c  NULL 039D2546851188122-52281-1-e.tg16.na2.in,1
0.6643119999999989,192.168.1.60,192.168.1.1,DNS,89,288,Standard query 0x5340  AAAA safebrowsing-cache.google.com,0
0.0385810000000000064,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x4375  NULL 039D2546851188122-52290-1-e.tg16.na2.in,1
0.4454049999999974,192.168.1.60,192.168.1.1,DNS,65,302,Standard query 0x7d48  A vg.no,0
0.2326769999999989,192.168.1.60,192.168.1.1,DNS,83,535,Standard query 0xb2c6  A self-repair.mozilla.org,0
0.0441809999999999825,192.168.1.14,192.168.1.1,DNS,99,164,Standard query 0x5546  NULL 039D2546851188122-52314-1-e.tg16.nf5.in,1
0.0426859999999999978,192.168.1.14,192.168.1.1,DNS,225,290,Standard query 0xf7aa  NULL 165N2546851188122-267-125-MPisaTIis47isxrkWipyJcWipyRiiMfCXAAAA.
```

```
    AiiwAAAAD6RQAANND3QABABiiHIrBcdTpBM7lLULgBQOwipnfbpHYU6AEAHJ
    .isr8AAAEBCAoABnfoXViiGRwieie.tg16.7vv.in,1
0.03876799999999747,192.168.1.14,192.168.1.1,DNS,212,277,
    Standard query 0xd5bc  NULL 152N2546851188122-279-112-
    MPLiiwqer31FJLk4HFVWGaD1ocoQZAAAAlwAb.
    hwIveG1sO3E9MC45LGltYWdlL3dlYnAsKii8qO3E9MC44DQpVc2VyLUFnZW5
    .0OiiBNb3ppbGwie.tg16.vb0.in,1
0.32990399999999991,192.168.1.60,192.168.1.1,DNS,87,417,
    Standard query 0xbcc9  A shavar.services.mozilla.com,0
0.35129199999999973,192.168.1.60,192.168.1.1,DNS,87,205,
    Standard query 0x04b8  AAAA shavar.services.mozilla.com,0
0.03485900000000086,192.168.1.60,192.168.1.1,DNS,74,309,
    Standard query 0x21f2  A acdn.adnxs.com,0
0.11150499999999752,192.168.1.60,192.168.1.1,DNS,74,267,
    Standard query 0x4536  A pixel.glimr.io,0
0.15580900000000142,192.168.1.60,192.168.1.1,DNS,69,302,
    Standard query 0x8baa  AAAA api.vg.no,0
0.18039399999999972,192.168.1.60,192.168.1.1,DNS,72,305,
    Standard query 0x0dab  AAAA static.vg.no,0
0.20447300000000013,192.168.1.60,192.168.1.1,DNS,68,315,
    Standard query 0x2d23  AAAA 1.vgc.no,0
0.20224599999999882,192.168.1.60,192.168.1.1,DNS,75,296,
    Standard query 0x83a4  A click.vgnett.no,0
0.85812100000000006,192.168.1.60,192.168.1.1,DNS,72,186,
    Standard query 0xc70e  AAAA ib.adnxs.com,0
0.77425999999999982,192.168.1.60,192.168.1.1,DNS,76,138,
    Standard query 0x62e7  AAAA logc189.xiti.com,0
0.28958199999999934,192.168.1.14,192.168.1.1,DNS,99,164,
    Standard query 0xff11  NULL 039D2546851188122-52392-1-e.
    tg16.z84.in,1
```

appendixCSV/Formatted.csv

Appendix C
Reformatting program

```python
import csv
import numpy as np


def openfile(filename):
    all = []
    containsPacketNumber = False
    with open(filename, 'rb') as readfile:
        spamreader = csv.reader(readfile, delimiter=",", quotechar='\"')
        for line in spamreader:
            if line[0] == "No.":
                containsPacketNumber = True
            if containsPacketNumber == True:
                line.pop(0)
                if line[3] == "DNS":
                    all.append(line)
            else:
                if line[3] == "DNS":
                    all.append(line)
    return all


def reformatCsv(filename):
    """
    Reformat the csv file by combining the query and the response
    and finding the time between them.
    """
    all = openfile(filename)
    queryID = ""
    counter = 0
    with open('csv/reformatDNS_3vm_XL.csv', 'wb') as writefile:
        spamwriter = csv.writer(writefile, delimiter=',', quotechar='\"')
        for line in all:
            if ".14" in line[1] or ".58" in line[1] or ".59" in line[1] or ".60" in line[1]:
                info = line[5]
                infoArr = info.split()
                queryID = infoArr[2]
                for line2 in all[counter:]:
                    if queryID in line2[5] and line[1] == line2[2]:
                        line[0] = float(line2[0]) - float(line[0])
                        line.insert(5, line2[4])
                        if ".14" in line[1] and "NULL" not in line[-1]:
                            pass

                        else:
                            spamwriter.writerow(line)
                        break
            counter += 1

def reduceMalDns(filename):
    """
    Removes, at random, half of the malicious DNS requests and responses.
    Takes in a reformatted csv file
    """
    all = openfile(filename)
    queryID = ""
    counter = 0
    with open('csv/reformatDNS_3vm_small_WithLables.csv', 'wb') as writefile:
        spamwriter = csv.writer(writefile, delimiter=',', quotechar='\"')
        for line in all:
            if ".14" in line[1]:
                if np.random.randint(0,2) == 1:
                    spamwriter.writerow(line)
                else:
                    pass
            else:
                spamwriter.writerow(line)




def addLabel(filename):
    """
    Adds a lable, 1, to the malicious DNS traffic, and 0 to
   regular traffic
    Takes in a reformatted csv file
    """
    all = openfile(filename)
    with open('csv/reformatDNS_3vm_XL_WithLables.csv', 'wb')
    as writefile:
        spamwriter = csv.writer(writefile, delimiter=',',
   quotechar='\"')
        for line in all:
            if ".14" in line[1]:
                line.append(-1)
                spamwriter.writerow(line)
            else:
                line.append(1)
                spamwriter.writerow(line)


def trainAndTestSet(filename):
    """
    Divides the csv file in two, one training set and one
   test set.
    """
    all = openfile(filename)
    testarray = []
    trainarray = []
    for line in all:
        if ".14" in line [1]:
            if np.random.randint(0,4) == 0:
                testarray.append(line)
            else:
                trainarray.append(line)
        else:
            if np.random.randint(0, 4) == 0:
                testarray.append(line)
            else:
                trainarray.append(line)
    with open('csv/reformatDNS_3vm_small_WithLables_addedRegDNS_testSet.csv', 'wb') as writetest:
        testwriter = csv.writer(writetest, delimiter=',', quotechar='\"')
        for line in testarray:
            testwriter.writerow(line)

    with open('csv/reformatDNS_3vm_small_WithLables_addedRegDNS_trainSet.csv', 'wb') as writetrain:
        trainwriter = csv.writer(writetrain, delimiter=',', quotechar='\"')
        for line in trainarray:
            trainwriter.writerow(line)



#addLabel('csv/reformatDNS_LargeV6.csv')
#trainAndTestSet('csv/reformatDNS_LargeV6_WithLables.csv')
#reformatCsv('csv/3VM_SlowDNS_xxlarge.csv')
#addLabel('csv/reformatDNS_3vm_XL.csv')
trainAndTestSet('csv/reformatDNS_3vm_small_WithLables_addedRegDNS_fix.csv')
```

/home/tk/PycharmProjects/MachineLearning/reduceDNS2.py

Appendix D

Prediction program

```
from sklearn import svm
import csv
from sklearn import metrics
import numpy as np
from sklearn.preprocessing import scale

with open('csv/
    reformatDNS_3vm_small_WithLables_addedRegDNS_mallast.csv'
    , 'rb') as csvfile:
     spamreader = csv.reader(csvfile, delimiter=',',
    quotechar='|')
     temp = []
     templabel = []
     for row in spamreader:
         one = float(row[4])
         two = float(row[5])
         temp.append([one, two])
         templabel.append(int(row[-1]))

X2 = np.array(temp)
X = scale(X2)
outlier = []
inlier = []
temptrue = []
for i, line in enumerate(X):
    if templabel[i] == 1:
        outlier.append(line)
        temptrue.append(-1)
    else:
        if np.random.randint(0, 4) == 0:
            outlier.append(line)
            temptrue.append(1)
        else:
            inlier.append(line)

X_train = np.array(inlier)

X_test = np.array(outlier)
y_true = np.array(temptrue)

clf = svm.OneClassSVM(kernel="rbf")
clf.fit(X_train)
y_pred_test = clf.predict(X_test)
print "Kernel = " + clf.kernel, metrics.
    classification_report(y_true, y_pred_test)

clf = svm.OneClassSVM(kernel="sigmoid")
clf.fit(X_train)
y_pred_test = clf.predict(X_test)
print "Kernel = " + clf.kernel, metrics.
    classification_report(y_true, y_pred_test)

clf = svm.OneClassSVM(kernel="linear")
clf.fit(X_train)
y_pred_test = clf.predict(X_test)
print "Kernel = " + clf.kernel, metrics.
    classification_report(y_true, y_pred_test)

clf = svm.OneClassSVM(kernel="poly")
clf.fit(X_train)
y_pred_test = clf.predict(X_test)
print "Kernel = " + clf.kernel, metrics.
    classification_report(y_true, y_pred_test)
```

/home/tk/PycharmProjects/MachineLearning/predictionV2.py

Appendix E
Test results

		precision	recall	f1-score	support
Kernel = rbf	Outlier	0.40	1.00	0.57	1124
	Inlier	1.00	0.51	0.67	3466
	avg / total	0.85	0.63	0.65	4590
Kernel = sigmoid	Outlier	0.25	1.00	0.40	1124
	Inlier	0.00	0.00	0.00	3466
	avg / total	0.06	0.25	0.10	4590
Kernel = linear	Outlier	0.41	0.20	0.27	1124
	Inlier	0.77	0.91	0.84	3466
	avg / total	0.69	0.73	0.70	4590
Kernel = poly	Outlier	0.03	0.07	0.05	1124
	Inlier	0.55	0.37	0.44	3466
	avg / total	0.42	0.30	0.35	4590

Table E.1: Classification report for OCSVM with different kernels and default parameters using the features LengthUp, LengthDown and Time from the dataset.

		precision	recall	f1-score	support
Kernel = rbf	Outlier	0.76	1.00	0.86	1124
	Inlier	1.00	0.90	0.95	3466
	avg / total	0.94	0.92	0.93	4590
Kernel = sigmoid	Outlier	0.24	1.00	0.39	1124
	Inlier	0.0	0.00	0.00	3466
	avg / total	0.06	0.24	0.10	4590
Kernel = linear	Outlier	0.01	0.01	0.01	1124
	Inlier	0.57	0.43	0.49	3466
	avg / total	0.43	0.33	0.37	4590
Kernel = poly	Outlier	0.04	0.08	0.05	1124
	Inlier	0.57	0.40	0.47	3466
	avg / total	0.44	0.32	0.37	4590

Table E.2: Classification report for OCSVM with different kernels and default parameters except `nu` which was set to 0.1, using the features LengthUp, LengthDown and Time from the dataset.

		precision	recall	f1-score	support
Kernel = rbf	Outlier	0.77	1.00	0.87	1124
	Inlier	1.00	0.90	0.95	3466
	avg / total	0.94	0.93	0.93	4590
Kernel = sigmoid	Outlier	0.25	1.00	0.40	1124
	Inlier	0.0	0.00	0.00	3466
	avg / total	0.06	0.25	0.10	4590
Kernel = linear	Outlier	0.32	1.00	0.48	1124
	Inlier	1.00	0.30	0.46	3466
	avg / total	0.83	0.47	0.46	4590
Kernel = poly	Outlier	0.24	0.94	0.38	1124
	Inlier	0.62	0.03	0.06	3466
	avg / total	0.53	0.26	0.14	4590

Table E.3: Classification report for OCSVM with different kernels and default parameters except `nu` which was set to 0.1, using the features LengthUp and Length-Down from the dataset.

		precision	recall	f1-score	support
gamma = 0.001	Outlier	0.77	1.00	0.87	1124
	Inlier	1.00	0.90	0.95	3466
	avg / total	0.94	0.92	0.93	4590
gamma = 0.01	Outlier	0.77	1.00	0.87	1124
	Inlier	1.00	0.90	0.95	3466
	avg / total	0.94	0.92	0.93	4590
gamma = 0.1	Outlier	0.76	1.00	0.87	1124
	Inlier	1.00	0.90	0.95	3466
	avg / total	0.94	0.92	0.93	4590
gamma = 1	Outlier	0.74	1.00	0.85	1124
	Inlier	1.00	0.88	0.94	3466
	avg / total	0.93	0.91	0.92	4590

Table E.4: Classification report for OCSVM with different kernels and default parameters except `nu` which was set to 0.1 and `gamma` which change from 0.001 to 1, using the features LengthUp and LengthDown from the dataset.

		precision	recall	f1-score	support
nu = 0.001	Outlier	0.93	0.06	0.11	1124
	Inlier	0.76	1.00	0.86	3466
	avg / total	0.80	0.76	0.68	4590
nu = 0.01	Outlier	0.72	0.06	0.11	1124
	Inlier	0.76	0.99	0.86	3466
	avg / total	0.75	0.76	0.67	4590
nu = 0.1	Outlier	0.78	1.00	0.8	1124
	Inlier	1.00	0.91	0.95	3466
	avg / total	0.95	0.93	0.93	4590
nu = 0.99	Outlier	0.25	1.00	0.40	1124
	Inlier	1.00	0.01	0.02	3466
	avg / total	0.81	0.26	0.12	4590

Table E.5: Classification report for OCSVM with different kernels and default parameters except `nu` which change from 0.001 to 0.99 and `gamma` which was set to 0.01, using the features LengthUp and LengthDown from the dataset.

		precision	recall	f1-score	support
init = ndarray	Outlier	0.00	0.01	0.00	286
	Inlier	0.86	0.50	0.64	3441
	avg / total	0.93	0.52	0.62	3727
init = k-means++	Outlier	0.00	0.00	0.00	286
	Inlier	0.18	0.02	0.03	3441
	avg / total	0.17	0.02	0.03	3727
init = random	Outlier	0.00	0.00	0.00	286
	Inlier	0.18	0.02	0.03	3441
	avg / total	0.17	0.02	0.03	3727

Table E.6: Classification report for K-means models with different init values, using the features LengthUp, LengthDown, Time and length of domain name in Info from the dataset. The data is scaled.

		precision	recall	f1-score	support
init = ndarray	Outlier	0.13	0.96	0.22	286
	Inlier	0.99	0.45	0.62	3441
	avg / total	0.93	0.49	0.59	3727
init = k-means++	Outlier	0.01	0.05	0.01	286
	Inlier	0.87	0.52	0.65	3441
	avg / total	0.81	0.49	0.61	3727
init = random	Outlier	0.13	0.95	0.23	286
	Inlier	0.99	0.48	0.64	3441
	avg / total	0.93	0.51	0.61	3727

Table E.7: Classification report for K-means models with different init values, using the features LengthUp, LengthDown, Time and length of domain name in Info from the dataset. The data is not scaled.

		precision	recall	f1-score	support
init = ndarray	Outlier	0.12	0.94	0.22	286
	Inlier	0.99	0.45	0.62	3441
	avg / total	0.92	0.49	0.59	3727
init = k-means++	Outlier	0.01	0.06	0.02	286
	Inlier	0.87	0.53	0.66	3441
	avg / total	0.81	0.49	0.61	3727
init = random	Outlier	0.13	0.94	0.23	286
	Inlier	0.99	0.47	0.64	3441
	avg / total	0.92	0.51	0.61	3727

Table E.8: Classification report for K-means models with different init values, using the features LengthDown, Time and length of domain name in Info from the dataset. The data is not scaled.