Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering

DNS tunneling detection

Jan Karsch

Supervisor: Ing. Ivan Nikolaev
May 2018

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Karsch** Jméno: **Jan** Osobní číslo: **438028**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Detekce DNS tunelování

Název bakalářské práce anglicky:

DNS tunneling detection

Pokyny pro vypracování:

1. Study principles of DNS tunneling
2. Perform experiments and analysis on DNS tunneling tools and data
3. Write a research document that summarizes patterns useful for detection based on differences between experiments on NetFlow protocol data and tunneled data
4. Based on analysis and research document, design the algorithm for detection of DNS tunneling.
5. Implement the algorithm, compare it to the state-of-the-art algorithms on network traffic data.
6. Evaluate implemented algorithm to real NetFlow protocol traffic datasets and datasets containing tunneled data.

Seznam doporučené literatury:

[1] ALBITZ, Paul. a Cricket. LIU. DNS and BIND. 3rd ed. Sebastopol, CA: O'Reilly, c1998. ISBN 9781565925120.
[2] DNStunnel.de - free DNS tunneling service. DNStunnel.de - free DNS tunneling service [online]. Copyright ? 2006 [cit. 07.01.2018]. Available from: https://dnstunnel.de/
[3] Merlo A., Papaleo G., Veneziano S., Aiello M. (2011) A Comparative Performance Evaluation of DNS Tunneling Tools. In: Herrero Á., Corchado E. (eds) Computational Intelligence in Security for Information Systems. Lecture Notes in Computer Science, vol 6694. Springer, Berlin, Heidelberg

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Ivan Nikolaev, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **19.02.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Ivan Nikolaev podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.

Datum převzetí zadání Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

I would like to thank my supervisor Ing. Ivan Nikolaev from Cisco Systems, Inc. for patience, valuable guidance and advice.

Declaration

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.

In Prague, 25. May 2018

. . . .

Abstract

This thesis focuses on detecting DNS tunneling and maps the whole process of necessary steps related to it. Research describes tools that are available for realizing DNS tunnels, their differences, and effect on network traffic. These tools are also used to realize experiments and capture relevant real traffic NetFlow protocol data with related operations of converting data between PCAP and NetFlow traffic format. For detection purpose is implemented pairing algorithm that pairs data as request-response and also mixing algorithm for infecting real traffic with tunneled one. Detection itself is done with machine learning classification, where used classifier takes infected traffic data analyzed with multiple Feature functions.

Keywords: Domain Name System, tunneling, network, security, machine learning, detection

Supervisor: Ing. Ivan Nikolaev
Cisco Systems, Inc., Charles Square Center, Karlovo Naměstí 10, 12000, Praha

Abstrakt

Tato práce se soustředí na detekci DNS tunelování a obsahuje s tím spojený proces nutných kroků. Popisuje dostupné nástroje, které jsou vhodné pro realizaci DNS tunelů, jejich rozdíly a vliv na síťový provoz. Tyto nástroje jsou rovněž použity pro vytvoření experimentů, zachycení relevantních síťových dat protokolu NetFlow a s tím spojenými operacemi pro konverzi dat mezi formáty PCAP a NetFlow. Pro detekci byl implementován párovací algoritmus, který spáruje data jako dotaz-odpověď a také mixovací algoritmus pro nakažení dat z reálného provozu daty tunelovanými. Detekce je provedena pomocí klasicikace strojového učení, kde použitý klasifikátor vezme nakažená síťová data analyzována několika Feature funkcemi.

Klíčová slova: Systém Doménových Jmen, tunelování, síť, bezpečnost, Strojové učení, detekce

Překlad názvu: Detekce DNS tunelování

Contents

Introduction 1

1 DNS tunneling 3
1.1 DNS 3
1.1.1 DNS records 4
1.1.2 DNS message format 5
1.2 Tunneling 5
1.2.1 DNS tunneling 5
1.3 Detection method construction 6
1.4 State of the art 7

2 DNS tunnel tools and experiments 9
2.1 Setup 9
2.2 DNS Tunneling tools 10
2.2.1 Iodine 10
2.2.2 DNSCAT2 12
2.2.3 TCP-over-DNS 12
2.2.4 DNS2TCP 13
2.3 OzymanDNS 14
2.4 Experiments 14

3 Dataset preparation 17
3.1 NetFlow protocol 17
3.2 Traffic conversion 18
3.3 Pairing algorithm 21
3.4 Mixing algorithm 22
3.5 Creating feature matrix 23
3.5.1 Feature functions 23

4 Classification 25
4.1 Preprocessing 25
4.2 Measuring 26
4.2.1 Confusion matrix 26
4.2.2 Precision 26
4.2.3 Recall 26
4.2.4 Precision-recall curve 27
4.3 Results 28

5 Implementation 31
5.1 Languages and libraries 31
5.2 Code 32
5.2.1 Modules 32
5.2.2 Testing 33
5.3 Problems 33

Conclusion 35

A Bibliography 37

B List of abbreviations 41

C Content of attached CD 43

Figures

1.1 Tunneling model.[4][5] 3
1.2 Domain name system hierarchy. . 4
1.3 DNS tunneling model.[4][5] 6
1.4 DNS tunneling detection procedure. 6

3.1 NetFlow traffic example. 18
3.2 Pair structure. 21
3.3 Traffic infection scheme. 22

4.1 Precision-recall curve example. . 27
4.2 Naive Bayes classifier precision-recall curves. 28
4.3 Support Vector Machine classifier precision-recall curves. 29
4.4 Random Forest classifier precision-recall curves. 30

Tables

1.1 DNS message format. 5

3.1 Mixing ratios and actual used sizes. 23

Introduction

In just a few decades the Internet has experienced immense growth and has transformed in ways that were unforeseeable at the time of its creation. It started out as a network that connected several USA universities in the 70s. It was a small network where everyone knew each other and thus it was designed without security in mind. Unfortunately, this lack of security has been inherited by the modern Internet in many crucial ways. DNS system also stems from the early days of the Internet. It is a core part of the infrastructure and, unfortunately, often allows easy misuse by malevolent parties such as DDoS attacks, DNS spoofing and DNS tunneling. The focus of this thesis is on DNS tunneling and methods for detecting it using NetFlow data.

Domain Name System - DNS is a crucial tool for the Internet. It provides a way to translate domain names to IP addresses which are hard to remember. Domain Name System can be abused to create a covert and secure channel – a tunnel – which can be used to overcome network policy blocks and secretly exfiltrate data out of the network.

Chapter 1 describes how DNS infrastructure works, the tree structure of the DNS system, the format of DNS records and messages The chapter also describes how DNS system can be abused to create DNS tunnels.

Chapter 2 describes several tools which can be used to establish DNS tunnels. It describes their main characteristics, configuration parameters, as well as advantages and disadvantages. It describes the set-up in which those tools were used to perform experiments, how a DNS name server was set up in AWS, how data was captured and what experiments were performed.

The thesis continues with a description of dataset preparation in Chapter 3. There it describes the data which was used as the background traffic. It also outlines the mixing algorithm that was used to create instances of users with DNS tunnels traffic mixed in, which were used as positive instances in classification. It also describes additional algorithms that were used, such as request-response matching and DNS server filtering. The features that were used for classification are also specified in this chapter.

Chapter 4 focuses on the final part of this thesis — classification methods using different machine learning algorithms. In this chapter different classifiers are trained and tested on the dataset and the results are presented in the form of PR curves.

The final Chapter 5 describes the software engineering effort that went into the creation of the thesis. It lists the programming languages and libraries that were used, as well as the version control system and testing techniques.

The whole thesis and the outcome of the experiments is summarized in Conclusion.

Chapter 1

DNS tunneling

DNS tunneling uses DNS infrastructure to create a covert communication channel. It allows overcoming access restrictions and policies in firewalled networks. Let's have points **A**, **B**, **C** representing computers in the Internet network where A has forbidden access to C. Point B is accessible and can serve as an intermediary point between A and C.

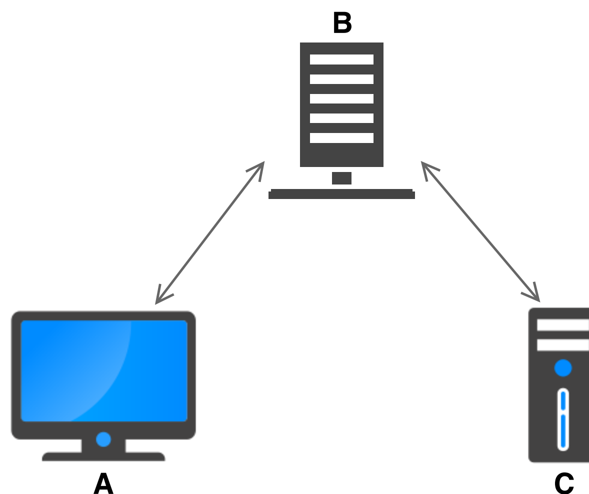

Figure 1.1: Tunneling model.[4][5]

This section gives a brief overview of DNS infrastructure. It acts as a translator between domain names and IP addresses. Besides that, Domain Name System also represents complex protocol running under an application layer of ISO/OSI model.

Following that, the mechanism of DNS tunneling is explained, as well as the state-of-the-art methods for detecting it.

1.1 DNS

DNS stands for Domain Name System. It allows translating domain names like www.google.com to IP addresses like 172.217.23.206 which are then used in IP network communication.

Domain Name System uses TCP and UDP protocols on port 53 - UDP protocol is used the most. Domain name namespace is structured hierarchically as a tree. This means that each node contains information about its subnodes.

Let's take as an example a domain name *sub.example.com*. After creating a new DNS request for this domain name, query first goes to one of 13 root DNS servers to resolve *.com* as a top-level domain. Following that, a request goes to *.com* server address, asks for *example.com* domain and in response gets its IP address. Last stop is at *example.com* server, as it asks for *sub.example.com* name which resolves in a final response of given DNS record type. In DNS protocol is the concept of so-called **NameServers** that handle DNS requests in given zone. Any IP address or domain in Internet infrastructure can be managed by any existing NameServer in the World, no matter where this server is located physically.

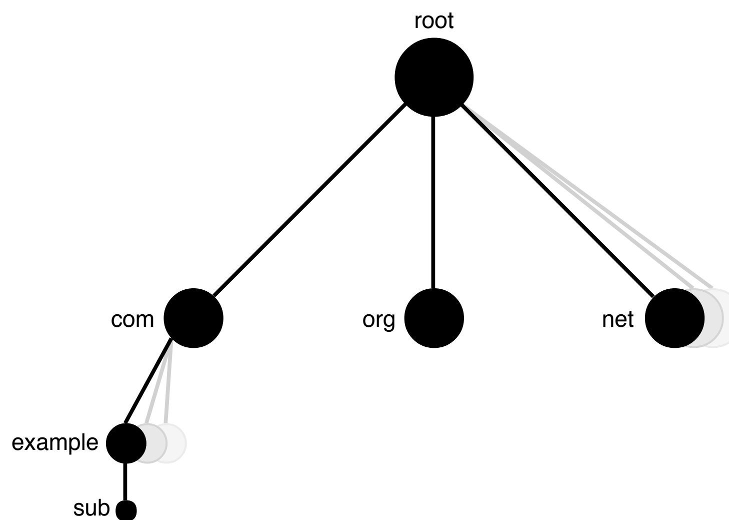

Figure 1.2: Domain name system hierarchy.

1.1.1 DNS records

DNS protocol uses multiple record types for different purposes. The most common record type is **A** record which translates domain names to IP addresses. **AAAA** records have the same function but for IPv6 protocol. **CNAME** record types are also known as aliases. This kind of record points to another defined domain name and never to an IP address, that offers a way, where one IP address can have multiple domain names (aliases) and only one record needs to be changed if the IP address changes. NS stands for **N**ame**S**erver, which means, it keeps the address of the authoritative NameServer that can resolve any subdomains for this domain. **TXT** records associate any given text with a domain name. **MX** records provide us information about mail servers.

1.1.2 DNS message format

Message transported by DNS protocol is composed by parts in following Table 1.1.

Header
Questions 1-n
Answers 0-n
Authority information
Additional section

Table 1.1: DNS message format.

1.2 Tunneling

Tunneling is transferring one network protocol inside another one — a process called encapsulation. For example, it is possible to create an SSH tunnel to a remote server and transfer all network communication securely through the SSH tunnel. To the outside, the connection will appear like an ordinary SSH connection, but on the inside, there might be HTTP, FTP, SMTP and other communication going on. Some protocols like SSH are designed specifically to support network tunneling. Other protocols, like DNS, are not designed for it but can be abused to create network tunnels and hide other communication inside it.

1.2.1 DNS tunneling

DNS tunneling provides a way to share data between two nodes by encapsulating them into DNS protocol packets (Figure 1.3). As intermediary point (**B**) is any available DNS server and node **C** is a Name Server that is under the control of the adversary. Note that primitive versions of DNS tunneling can work by directly sending DNS packets to the destination server **C** mimicking the DNS protocol. This, however, can be easily detected. Moreover, direct access might be forbidden by the firewall policy in many networks.

DNS Name Server **C** needs to be registered as a Name Server for a specific domain in the DNS system and both machines (**A** and **C**) need to run client and server side of DNS tunneling software. The DNS tunnel is created by encapsulating network communication into DNS requests. The DNS requests are addressed to subdomains of the Name Server domain and data is encoded into those subdomains, commonly using Base32 encoding.

As already mentioned, the encoded communication can be any type of network communication. For example, it allows to create SSH connections that communicate solely through DNS requests on the outside. DNS tunneling tools always have a client-server architecture where server-side listens for incoming connections from clients. Incoming information comes in the form of DNS requests and response in the form of DNS responses.

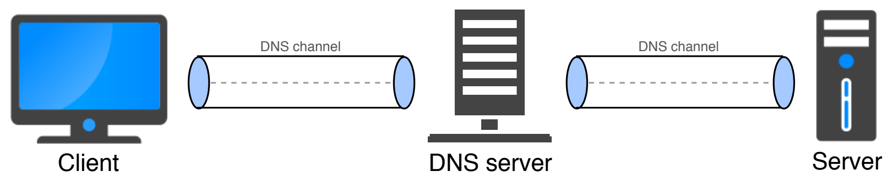

Figure 1.3: DNS tunneling model.[4][5]

1.3 Detection method construction

The main goal of this thesis is to design a detection method for DNS tunneling using NetFlow data. That means, write an algorithm that takes traffic data as input, evaluates all users and marks those that perform DNS tunnels.

The detection method in this thesis uses NetFlow data as input. NetFlow data provides aggregate statistics about IP connections in the network. A NetFlow record consists of a time stamp, duration, source and destination address and ports, protocol and transferred bytes and packets. This information may seem quite limited, but it is possible to extract a lot of knowledge from it by analyzing the problem and designing appropriate features for solving it.

To our knowledge, there is no NetFlow dataset with a lot of clean, regular users as well as users who perform DNS tunnels. This dataset is necessary for creation and testing of a classifier, so one was created synthetically. First step was performing DNS tunnel experiments, capturing PCAP data from the experiments and converting them to NetFlows. Seconds step was mixing the DNS tunnel behavior into the traffic of ordinary users from a real university network. The users with mixed in DNS tunnels were labeled as positive in the time frame where DNS tunnel was mixed in. Everything else was labeled as negative. Then a request-response pairing algorithm was run on the data. After that a feature matrix was created, where each row represented one user's behavior in a five minute time frame. The whole process is outline in Figure 1.4

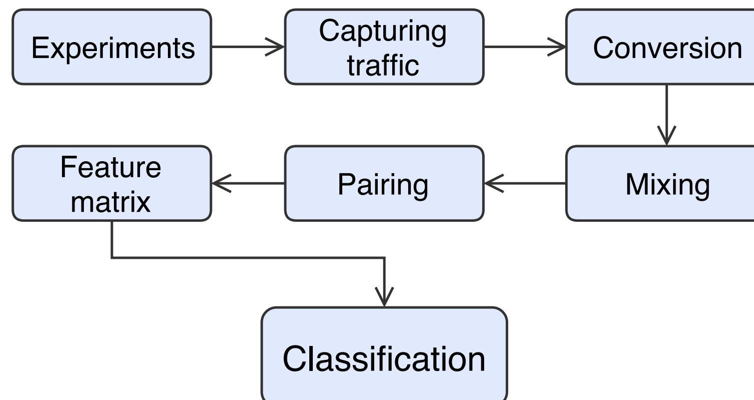

Figure 1.4: DNS tunneling detection procedure.

1.4 State of the art

Multiple approaches have already been created, this section summarizes some of them and compares with a solution of this thesis.

- **Flow-Based Detection of DNS Tunnels** [1]

  This research performs tunneling experiments with the same approach as this thesis do. It uses Iodine DNS tunneling tool for multiple experiments including tunneled HTTP traffic. Detection is based on statistical analysis of captured NetFlow-based data, defining multiple meaningful metrics (similar to our features in Section 3.5) for traffic congestion and density, such as *bytes per flow*, *packets per flow* etc. Following that, the paper describes 3 methods for anomaly detections and 5 detectors using them.

- **DNS Tunneling Detection Method Based on Multilabel Support Vector Machine** [2]

  Research focusing on detecting DNS tunnels with multilabel approach instead of binary (tunneled vs non-tunneled). The paper tries to detect multiple types of DNS tunnels (FTP, POP3, HTTP, HTTPS) separately, also defines multiple features similar to ours. The authors use a different type of data — they analyze domain names. Finally, they use Support Vector Machine classifier and Multilabel Bayes classifier for detection.

- **A Bigram Based Real Time DNS Tunnel Detection Approach** [3]

  Paper that defines experimental real-time scoring mechanism based on evaluation of domain names. Names are decomposed to *bigrams* and afterwards evaluated using frequency analysis, scoring mechanism and following classification.

Existing papers above have in some aspects similar approach. Naturally, perform DNS tunneling experiments with captured traffic data and some also create feature matrices by analyzing users traffic.

Nevertheless, two of mentioned solutions do not use NetFlow-based dataset and analyze data such as request domain names by performing deep packet inspection. This might be considered as an attack on users' privacy.

The approach in this thesis is based on using NetFlow data that does not give any information containing transmitted data or domain names. Also, an advantage of the NetFlow-based solution is in performance cost as it does not have to unpack each request separately which takes a lot of processing power. NetFlow gives summarized requests in one IP flow (more in section 3.1).

The third solution operates with the NetFlow-based data and tests different detection techniques. However, it only tests on DNS tunneling tool, whereas this thesis use four different tools.

Even though mentioned researchers work with tunneling tools, they do not perform multiple experiments with different software and focus on one or two

of those. This thesis contains four described tools that were made to make work and also a variety of use cases for a bigger diversity of the dataset.

Chapter 2

DNS tunnel tools and experiments

The previous section was mostly theoretical. Main concepts and techniques in tunneling protocols have been characterized and also was described the main purpose of Domain Name System.

We are now capable of running first DNS tunneling experiments resulting in captured traffic logs for multiple different tunneling attempts.

First sections take a look at creating the right environment for running tunneling tools, because, except infected computer, tunneling needs a server that listens for client requests and DNS server as a forwarder.

In next sections will be finally performed DNS tunneling experiments, described step-by-step for each used malware.

2.1 Setup

Before running tunneling tools and creating experiments firstly has to be created a working technical background. Previous sections talked about client-server architecture, meaning, need of at least two different working stations with Linux operation system. The server needs to be on the same local network as a client or, in the best case, have a public IP address.

Amazon AWS[6] has been chosen as it allows to create virtual machines with the selected operation system (offers multiple choices) and also gives us 750 working hours per month. Amazon instances are not by default able to receive traffic except incoming ssh connections (port 22). To permit all traffic, new security group, that does not drop inbound or outbound data, has to be configured.

The last required node is an intermediary point between client and server, which, as was already mentioned, has to be some DNS server. Luckily, there are free providers that allow users to create multiple DNS record types - FreeDNS[7]. Client side will send DNS packets to DNS server, that will forward all incoming traffic for given domain name to another Internet point, which is in this case, AWS server. As this thesis describes in Section 1.1, forwarding to a different zone or domain name can be done by defining NS record type. The incoming request asking for domain name X will be redirected to different nameserver having domain name Y. However, how does the DNS server knows, where is the Y zone? The Solution is the so-called

glue record. Right after reading NS record type, DNS server asks itself what is the IP address of Y zone, meaning, A or AAAA record for destination zone has to be specified. There is no explicit rule for domain names, only that both names used in NS record have to be different from each other.

```
tunit.mooo.com. IN NS dnstctu.mooo.com
dnstctu.mooo.com. IN A 35.158.155.65
```

Listing 2.1: DNS server configuration records.

Final DNS server setup, where *tunit.mooo.com* is domain name client sends requests to. Because there is defined NS record, all requests are about to be forwarded to *dnstctu.mooo.com* which IP address is in next-defined A record - *35.158.155.65*, leading to transfer of DNS requests from a client to the server. As client always sends data to tunit.mooo.com, DNS server does not change request in any particular way and sends it to destination AWS server. At this moment, the server-side of tunneling tool has to work as DNS server itself to process all incoming DNS requests for tunit.mooo.com.

2.2 DNS Tunneling tools

DNS tunneling software, as an instrument for possible hackers, is mostly for Linux operation system distributions. Even though most of the nowadays attacks happen on Microsoft Windows systems because of a larger user base, only few tunneling tools have a version for this system.

We have tried to make work as many tools as was possible, but since first DNS malware was written in early years of the previous decade, most of them are now obsolete and sometimes not compatible. Unfortunately, any concrete reason for defects was not found but it might be caused by obsoleteness of code, non-stable connection or inefficient implementation.

Finally was managed to make work 4 software, Iodine[8], DNSCAT2[9], TCP-over-DNS[10] and DNS2TCP[11]. Other tested tools were OzymanDNS[12], Heyoka[13], DNSCAT[14], TUNS[15], DNScapy[16], DeNiSe[17]. All tested tools are client-server architecture where server-side listens for incoming DNS requests.

2.2.1 Iodine

Iodine tunneling tool has got GitHub project repository with a detailed description of features and functionalities, it also offers a complete tutorial for DNS setup as was described in previous sections.

Iodine is by default part of Kali Linux distribution and also of popular Advanced Packaging Tools (APT) system. For installing Iodine on custom distribution, execute the following command to the terminal.

```
$ sudo apt-get install iodine
```

The Previous command installs two separate tools - Iodined, for the server instance and Iodine for client one. Except for GitHub documentation, Iodine

offers -h execute parameter for displaying implemented command line switches and settings.

```
Server
$ sudo iodined [options] tunnel_ip topdomain

Client
$ sudo iodine topdomain

-f to keep running in the foreground
-r to skip RAW UDP mode attempt
-P password used for authentication
-T force DNS record type
-O force downstream encoding
-c to disabled check of client IP/port on each request
-D debug level
```

All mentioned parameters have not been used, because Iodine has autodetect as a default option for all of them. It also might be kind of contra-productive. For example, forcing one exact DNS record type escalated in the totally non-stable experiment. The goal was to create experiments as elementary as was possible since we expected a high amount of data coming through DNS protocol. Following commands were used to run both instances.

```
$ sudo iodined -f -c -P password 10.0.0.1 sub.tunit.mooo.com

$ sudo iodine -r -f -P password sub.tunit.mooo.com
```

Experimenting with Iodine revealed one interesting thing. Used topdomain *sub.tunit.mooo.com* in commands on both sides is little different from the one used in Section 2.1 - whole topdomain was enhanced with one more subdomain *sub*. Multiple user experiments and tests showed that this case was much faster and stable than non-enhanced one.

After executing the second command, for client instance, Iodine takes few seconds to create the tunnel through DNS. As Iodine completes tunnel, it also creates a **TUN0** virtual interface on both sides. These interfaces represent created connection between both nodes where the server has chosen IP address 10.0.0.1 and client following one - 10.0.0.2. This finally offers us to create tunneled TCP connection through DNS.

```
Server
$ ssh user@10.0.0.2

Client
$ ssh user@10.0.0.1
```

Also, by creating tunnel is created a private network where both nodes can visit each other even though client instance does not have a public IP address. This has to mean, the client side of Iodine periodically sends packets to a server and asks for updates.

2.2.2 DNSCAT2

DNSCAT 2 has its own ruby implementation of a terminal interface, does not force to use ssh and uses own commands. It offers functionalities such as transferring files between endpoints or creating shell session from a server to client.

```
Server
$ sudo ruby dnscat2.rb tunit.mooo.com -c password

-c for session password

Client
$ ./dnscat --secret=key tunit.mooo.com
```

After creating a tunnel, server-side has access to the client. Following commands open command line from the server to client.

```
window -i 1
shell
# Shell session created
sessions # Lists all sessions for
          obtaining the number of a created shell session
session -i 2

-i sets which session or window to open
```

2.2.3 TCP-over-DNS

TCP-over-DNS is DNS tunneling tools written in JAVA language. It is one of classic tunneling tools that force us to use ssh connection.

```
Server
$ sudo java -jar tcp-over-dns-server.jar
--domain tunit.mooo.com --forward-port 22

Client
```

```
$ java -jar tcp-over-dns-client.jar --domain tunit.mooo.com
--listen-port 3333 --interval 100 --dns-server 8.8.8.8
```

After establishing a connection, can be started ssh session from the client.

```
$ ssh user@localhost -p 3333
```

As was chosen port 3333 to listen on for a client, client side is listening on the same exact port for incoming data to tunnel through DNS. Also, without used *–dns-server* parameter, tool gave us Java exceptions that prevented us from creating a tunnel.

2.2.4 DNS2TCP

DNS2TCP is part of popular Kali Linux distribution. The easiest way to get this tool is to download it from Kali Linux repositories as it does not have any official documentation. For running DNS2TCP, it is necessary to edit the configuration file on server-side located in /etc/dns2tcpd.conf.

```
listen = 0.0.0.0
port = 53
\# If you change this value, also
\# change the USER variable in /etc/default/dns2tcpd
user = nobody
chroot = /tmp
domain = tunit.mooo.com
resources = ssh:127.0.0.1:22
```

```
Server
$ sudo dns2tcpd -F -f /etc/dns2tcpd.conf

Client
$ sudo dns2tcpc -c -z tunit.mooo.com -r ssh 8.8.8.8 -l 3333

-F for running in the foreground
-f pidfile location
-c compression
-z domain name
-r resource to use
-l port to listen on
```

Connecting to the server is the same as in previous cases - use SSH command on the port that was set for the client side to listen on.

```
ssh user@localhost -p 3333
```

2.3 OzymanDNS

As was specified before, OzymanDNS is a tool that was not successfully executed. Anyway, since this software has a lot of written documentation and is quite popular in used sources, we consider it as a tool that should be at least documented for possible future experiments. Also, a lot of time was spent with experimenting and trying to make this tunneling software work. OzymanDNS is Perl written and again has client and server instances to run. The first encountered problem was missing libraries, which was quite a common obstacle for most of the tested tools, but usually, used language exception told us which library was exactly missing. OzymanDNS was different as it kept referencing to very small libraries and even older versions. Whole library process was quite unnecessary as the final required command is very simple.

```
$ apt-get install screen libnet-dns-perl libmime-base32-perl
```

```
Server
$ sudo ./nomde.pl -i 0.0.0.0 -p tunit.mooo.com

Client
$ ssh -C -o ProxyCommand="./droute.pl sshdns.tunit.mooo.com"
        user@localhost
```

From client-side command, it was expected to create secure shell session, but unfortunately, nothing happened on both sides. Analysis with TCPDUMP told us, that DNS requests from the client to the server are going through, but server-side of OzymanDNS was not capturing any of them. Based on that, we have not invested more time in making this tool work.

2.4 Experiments

The result of tunneling experiments has to be captured network traffic. Following chapter specify NetFlow protocol data which is this thesis goal to detect DNS tunneling with and as we will get to know, for local Linux experiments, network traffic cannot be capped right into NetFlow protocol format. The solution is to capture traffic into common PCAP (packet capture) format and afterward convert to NetFlow.

To have same experiments for each tool, as a mechanism for experiments that generates traffic data is used basic Bash file with multiple Linux terminal commands containing *sleeps* to approach effect of the real user.

```
whoami
sleep 0.2
date
sleep 0.4
mkdir dir
```

```
sleep 1
ls -la
sleep 0.5
ifconfig
...
```

Listing 2.2: Fakeuser script snippet.

Next experiment was to upload a file from client to the server. Multiple files sizes were tested but even for 1KB file, the successful transfer was not guaranteed and resulted in errors. Final file size was 500 bytes. In following written Bash script, we can also notice tool sshpass that allows us to write ssh password in the command line as it speeds up experiments and doesn't need any additional user interaction.

Following bash script takes two arguments, first string parameter that represents used tunneling tool and second representing a path to file that is sent through the tunnel.

```
if [$1 == "iodine"]
then
  for i in {1..20}
  do
    sshpass -p "password" rsync -v -e ssh $2
        user@10.0.0.1:/home/user/
  done
elif [$1 == 'dns2tcp'] || [$1 == 'tcpoverdns']
then
  for i in {1..20}
  do
    sshpass -p "password" rsync -v -e "ssh␣-p␣3333" $2
        user@localhost:/home/user/
  done
fi
then
```

Listing 2.3: File upload Bash script.

Some tools created the possibility to initiate ssh connection to the client. This means the client has to send periodical requests to a server that ask for updates. The last experiment was to capture only those periodical data.

The only operation left is to perform experiments and capture data. There are multiple Linux tools for traffic monitoring and capturing. As any special functions are not necessary, we ended with **Tshark**[18] which is a terminal version of the well-known tool for traffic analysis - **Wireshark**, common **TCPDUMP** packet analyzer would serve the purpose as well. Tshark is into 5-minute intervals due to detection reasons described in the following chapter.

```
sudo tshark -a duration:300 -w traffic.pcap

-a for capture autostop condition
-w setting traffic output file
```

Each experiment was repeated 20 times for used tool which overall gave us 160 experiments. 60 for Iodine, 20 for DNSCAT2 (Since it has its own implementation and experiments with written scripts were not possible) and each TCP-over-DNS, as well as DNS2TCP, took 40.

Chapter 3
Dataset preparation

Detection process continues and finally gets to the point where we can manipulate data, analyze it and use it in other experiments. However, captured data are quite ambiguous and is hard to to use for detection as it is. Also, all detection experiments have to be done on NetFlow protocol data, which means, we have to first come up with a method to convert captured PCAP data to NetFlow.

As will get to know, this protocol does not give us that many information as PCAP. Because detection technique is a use of machine learning classifiers, part of the procedure is infecting real traffic with tunneled one which simulates traffic where actual users perform DNS tunneling.

The background traffic data originates from an academical network. The background dataset covers several days of network usage and contain data from thousands of actual users.

Section 3.1 explains NetFlow format and describes what kind of data it provides. Followed by very important traffic conversion between PCAP and NetFlow in Section 3.2.

Section 3.3 describes a solution for gathering more information - pairing algorithm that matches flows in the traffic as request-response. For traffic infection, we have hundreds of real traffic NetFlow users and also performed experiments that serve as input to mixing algorithm.

Final Section 3.5 is dedicated to a critical part which is an algorithm that takes infected data, applies multiple feature functions and returns a matrix of features for every single IP address from network traffic.

3.1 NetFlow protocol

Dataset preparation Chapter describes, in full, all the steps that were taken to prepare the dataset that was used for training and testing the classifiers. This section offers a detailed description of the desired dataset we would like to detect DNS tunneling with.

NetFlow[22] is by origin one of protocols implemented by Cisco company. The main purpose is to monitor network by IP traffic flows which brings a detailed view of networks to administrators and owners.

IP flows are leading technique used in this protocol. It is defined as a sequence of packets with same attributes such as *Source IP address*, *Destination IP address*, and *Protocol*. In common Wireshark or TCPDUMP analysis, it is common to have as many traffic rows as was sent packets. This NetFlow technique takes packets of same sources, destinations, protocols and merges them in one single flow. Each flow contains below-listed information.

- Date
- Flow start
- Duration
- Protocol
- Source IP address + Source port
- Destination IP address + Destination port
- Flags (For TCP protocol)
- Type of service (From IP header)
- Packets transmitted
- Bytes transmitted
- Number of flows

Date flow start	Duration	Proto	Src IP Addr:Port		Dst IP Addr:Port	Packets	Bytes	Flows
2018-10-01 00:00:00.459	0.000	UDP	107.0.0.1:2333	->	192.168.1.1:22126	1	32	1
2018-10-01 00:00:00.363	0.000	UDP	192.168.1.1:22126	->	107.0.0.1:2333	1	98	1

Figure 3.1: NetFlow traffic example.

3.2 Traffic conversion

As NetFlow is by origin Cisco standard, it is widely supported by routers and other network devices. **nfdump**[23] toolset was used for converting PCAP data to NetFlow.

nfcapd Listens for incoming netflow data on given local port, collects and stores into flow records that are usually rotated in 5-minute intervals.

nfdump Designed for reading files collected by nfcapd.

nfpcapd Collector for creating data from traffic or PCAP files.

Mentioned nfcapd hasn't, by default, allowed function for reading PCAP files. It can be activated by compiling nfdump tool with additional arguments. Default nfdump version in APT repositories also does not include this functionality.

```
./configure --enable-readpcap --enable-nfpcapd
make
sudo make install
```

Unfortunately, as the nfdump documentation says, *readpcap* is experimental functionality that is not fully developed yet and hasn't worked for our captured data. The same result happened in case of a *nfpcapd* parameter, which installed separate tool as was mentioned above.

We were successful with combining nfdump, nfcapd, and tool named **softflowd**[24]. Softflowd has the same function nfpcapd should have, it creates an exporter, that takes input data as an argument and sends to defined IP address and port.

First has to be executed nfcapd as it listens on given port and afterward softflowd with pcap file as input. Result NetFlow traffic can be read by running nfdump.

```
sudo nfcapd -p 12345 -l ./path

-p port to listen on
-l path to save NetFlow traffic to

softflowd -n 127.0.0.1:12345 -r ./file.pcap

-n IP address with the port to listen on
-r input pcap file

sudo nfdump -o long -r ./nfcapd.file

-o output data format
-r input NetFlow file
```

Nfdump offers a few data output formats and the *long* one used corresponds with real traffic data, that are prepared for infecting.

We tried to automatize conversion process with bash scripts but the problem occurred since nfcapd names output files with a timestamp in YYYYMMDDHHMM format and does not have a parameter for additional indexing or renaming. The workaround was found with custom setting of rotation argument in nfcapd for 1 minute. That means each file will be created minute after minute. The only thing left is to make softflowd send data with approx 1-minute delay. To ensure that, basic sleep bash command was used.

```
sudo nfcapd -p 12345 -l ./path -t 60

Softflowd_script.sh
#!/bin/bash
for file in ./path/*.pcap
do
  softflowd -n 127.0.0.1:12345 -r $file
```

```
  sleep 61.03
done
```

By using delays, processing obviously take some time for hundreds of files but is, at least, automated.

The only thing left is to process NetFlow data through nfdump and make them readable. However, to match our real traffic caps, some additional operations had to be done as nfdump creates some unnecessary data at the bottom of the file. Following script creates raw data with nfdump, insert string to the first line and removes last 4 rows.

```
Nfdump_script.sh
#!/bin/bash
rm ./readable/*
for file in ./nfcapd.*;
do
  fname = ${file##*/}
  touch ./readable/$fname_raw
  nfdump -o long -r $file >> ./readable/$fname_raw
  sed -i '1s/^/<tunneled>\r\n/' ./readable/$fname_raw
  touch ./readable/$fname
  head -n -4 ./readable/$fname_raw >> ./readable/$fname
  rm ./readable/$fname_raw
done
```

3.3 Pairing algorithm

To get as many relevant information as possible was implemented pairing algorithm that pairs input NetFlow data as request-response. To store data is used using Pandas library with so-called DataFrames (more in Section 5) allowing us to efficiently work with given information.

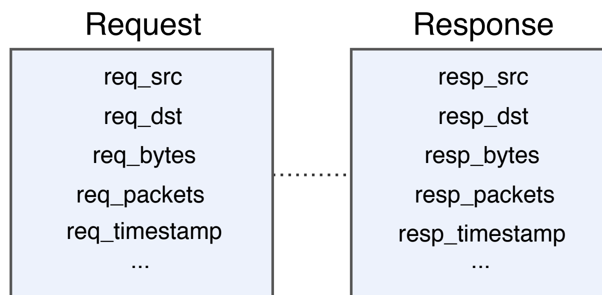

Figure 3.2: Pair structure.

Algorithm iterates over input data record by record and inserts or updates them by key, composed of source and destination, to Python dictionary. Each key, in this data structure, points to list of pairs algorithm will try to update (if it is relevant) or append a new one that might be updated by a different record in following iterations.

```
for row in data:
  if create_map_key(row.src, row.dst) in dict:
    update_existing_record(row, row.src, row.dst)
  elif create_map_key(row.dst, row.src) in dict:
    update_existing_record(row, row.dst, row.src)
  else:
    insert_new_record(row)
```

Listing 3.1: Algorithm for record distribution to the dictionary.

Since all obtained real traffic data are from a single network, we know IP addresses and ranges that belong to local users - which is our only concern, users from different networks are not relevant. While creating a pair, the algorithm asks if the current processed source IP address belongs to the local network and if so, insert record to pair as a request, in opposite case, it would insert as a response.

```
new_pair = None
  if is_address_local(row.src_ip):
    new_pair = create_request(row)
  elif not is_address_local(row.src_ip):
    new_pair = create_response(row)
return new_pair
```

Listing 3.2: Creating a new pair.

In case of existing key in the dictionary, the algorithm does not immediately create new pair but iterates over list the key points to, find the most relevant pair and try to update it. Function for checking relevancy compares currently processed record with the existing pair. It checks IP addresses because for creating pairs, both have to be switched - X['src'] == Y['dst'] and X['dst'] == Y['src']. As well as protocols that have to be same in both records.

Good idea is to check timestamp relevancy, but delays and latency reasons cause that using time information is not absolutely trustworthy. After all, the algorithm has timestamp comparisons but with additional 1 second to both, past and future, directions.

```
relevant = ex.timestamp <= new.timestamp <= (
           ex.timestamp + time(seconds=1))

if new.src == ex.dst and new.dst == ex.src
              and relevant:
    return True
```

Listing 3.3: Checking relevancy if existing record is a request

3.4 Mixing algorithm

Our real traffic background data contains hundreds of NetFlow protocol traffic files in the interval of 5 minutes and as mentioned in previous section 2.4, tunneled experiments were also captured into 5-minute intervals as it needs to have data of the same time-relevancy. Mixing algorithm is another step for creating relevant data as input for the machine-learning classifiers. The main requirement for the algorithm is infecting real traffic caps with our tunneling experiments for simulation of the real case when a user performs DNS tunneling in actual traffic.

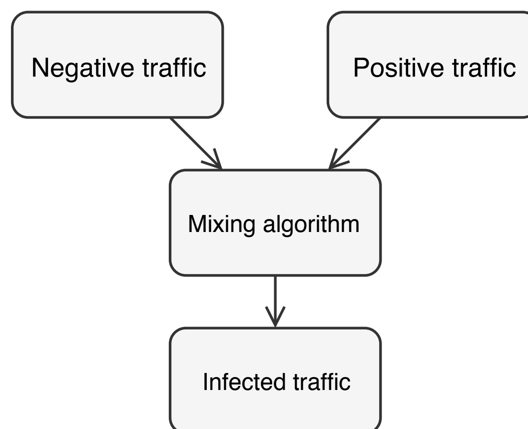

Figure 3.3: Traffic infection scheme.

Technically, we have to pick one existing user from real traffic data and insert new records. The data for picking users go through multiple filters

because it has to be IP address from the local network, also cannot be for example local DNS server and for correctness, this data cannot contain one user multiple times.

When algorithm picks one user, it also receives tunneled data as input and substitutes existing IP addresses for users one. Note that, as was used one user for one infection, it can not be used in different one at given 5-minute interval.

We have also managed to create multiple mixed data based on the ratio of tunneled and non-tunneled users - 1:10, 1:100, 1:1000. This ratio says how many tunneled users are in infected traffic, for example, ratio 1:10 says that every tenth user has performed DNS tunneling. Actual used tunneled and non-tunneled sizes are in following Table 3.1. Used ratios are compared to different classifiers and evaluated in following Section 4.

Ratio	Actual users
1:10	10 000:100 000
1:100	1 000:100 000
1:1000	100:100 000

Table 3.1: Mixing ratios and actual used sizes.

3.5 Creating feature matrix

So far, have been implemented crucial parts for preprocessing data and next goal is to create feature matrix by using them. Feature matrix represents input mixed data analyzed by multiple functions. Each matrix row will represent one user from traffic and row columns will be different features. Every feature needs to serve as an attribute that will differ real traffic from tunneled. Set of feature functions is applied to whole paired traffic from each user.

3.5.1 Feature functions

Following listing describes feature functions used for data analysis and creating result matrix. Functions are applied on the structure of requests from one user.

- **Bytes median, Packets median**

  As tunneling experiments showed, encapsulated TCP connection creates a big amount of data over a 5-minute interval. Median of bytes and median of packets should differ this aspect from other traffic.

- **Number of DNS requests**

  Feature evaluating sum of sent DNS requests.

- **Packets/Flows**

  Calculates the sum of sent packets per sum of flows.

- **Packets/Unique destinations**

  As mentioned, traffic gets denser with a created tunnel. Sum of sent packets per sum of unique destinations.

- **DNS requests/Unique DNS destination**

  Very similar feature as previous one, but also very important to have. Same traffic "congestion" happens after creating common TCP connection. Calculating the same feature only for DNS requests can differ Tunneled traffic from non-tunneled.

- **Paired/flows**

  Sum of paired per sum of all flows.

- **Bytes up/Packets up**

  Sum of uploaded DNS bytes per sum of DNS request packets.

- **Bytes down/Packets down**

  Sum of downloaded DNS bytes per sum of DNS response packets.

- **Bytes up/Bytes down**

  Sum of uploaded DNS bytes per sum of DNS downloaded bytes.

- **Requests/Unique source port**

  Each tunneling software always creates a tunnel and binds it to source port which does not change over the created connection. It was surprising to us, but our opinion is, there is a possibility to write a tool which picks different source port over time, even for each request. This feature can significantly influence results and multiple comparison classifier experiments are done to test it. This feature calculates the sum of requests per sum of unique source ports.

As can be noticed, we are creating multiple features for the division of values. But what would happen if, for example, analyzed user has no DNS records? Obviously, division by zero error. This cases and other similar ones are handled by putting temporary 'NaN' values that are managed in following classifier experiments.

Previously described algorithms also provide functions for creating labels that tell if currently manipulated traffic is tunneled or non-tunneled, it is very important part of classifier experiments.

Chapter 4

Classification

Classification is a statistical process of categorizing new data to existing classes based on its features. For example computer recognition of handwritten text from images by analyzing pixels and their coordinates on the input image.

In the context of this project, we need to recognize which of given users from network traffic perform DNS tunneling. As our data have only two resulting values (True/False or Tunneled/Non-tunneled) classifier can assign, it is called *Binary classification*.

Used classification algorithms, known as *classifiers*, are a concrete implementation of mathematical operations. The last section performs and describes results of classification experiments using **Gaussian Naive Bayes**[25], **Support Vector Machines**[26] and **Random Forest**[27] classifiers.

4.1 Preprocessing

Just before putting data to actual classification algorithm, there has to be done some preprocessing on our feature matrix.

Missing values

The previous section mentioned what happens in the case of division by zero error while computing feature function. The algorithm, instead of calculating, inserts 'NaN' value representing a missing value. Machine learning algorithms need correct data and missing values has to be replaced with relevant ones. These values can be replaced with median or mean operations, but in some cases, it also might be a good idea to use large negative values. Missing values in this thesis are replaced with default mean strategy.

Feature scaling

Machine learning algorithms often calculate *distance* between points by *Euclidean distance*. If one feature would have a broad interval of values, it would also negatively influence this distance calculation. For this purpose is realized scaling of features by *standardization* method.

$$x_{stand} = \frac{x - \text{mean}(x)}{\text{standard deviation } (x)} \tag{4.1}$$

4.2 Measuring

There are multiple existing measurement techniques for model evaluation performance. Observing only error rate or accuracy might be a little obscure as it depends on the ratio of positive and negative cases. Our experiments were evaluated using *precision* and *recall* in which we need to know what is *confusion matrix*.

4.2.1 Confusion matrix

The confusion matrix is a table used for describing classification model performance. It identifies values that were predicted wrong across available labels. In our case, Binary classification, confusion matrix does not get too confusing as we have only two values to predict and there are no multiple relations between classes. Matrix divides into categories below.

TP: True positive - Number of objects predicted positive and are actually positive.

FP: False positive - Number of objects predicted positive and are actually negative.

TN: True negative - Number of objects predicted as negative and are actually negative.

FN: False negative - Number of objects predicted as negative and are actually positive.

4.2.2 Precision

Precision, also known as *positive predictive value* is a fraction of positive objects among all objects classifier marked as positive. In other words, it represents the probability that positively labeled object is truly positive.

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

4.2.3 Recall

Recall, also known as *sensitivity* is a fraction of correctly labeled positive objects among all truly positive objects. In other words, it represents probability that actual positive object will be labeled as positive.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

4.2.4 Precision-recall curve

The following section contains concrete model evaluation with use of previously described precision and recall as it can be used for visualization by plotting a *precision-recall curve*. The problem, machine-learning classifiers solve, is finding of the optimal *threshold* that differs existing labeled points in the space. It might also be interpreted as a hyperplane cutting through space and dividing it, based on that, classifier makes final label predictions. Optimal threshold also sets the best trade-off between precision and recall values because usually with growing recall value, precision gets lower. Then, moving with threshold hyperplane in space would also have an influence on predicted labels and certainly on the precision with the recall. Final precision-recall curve uses multiple threshold positions with calculating precision and recall for each. The best case is to have 100% precision for every recall value.

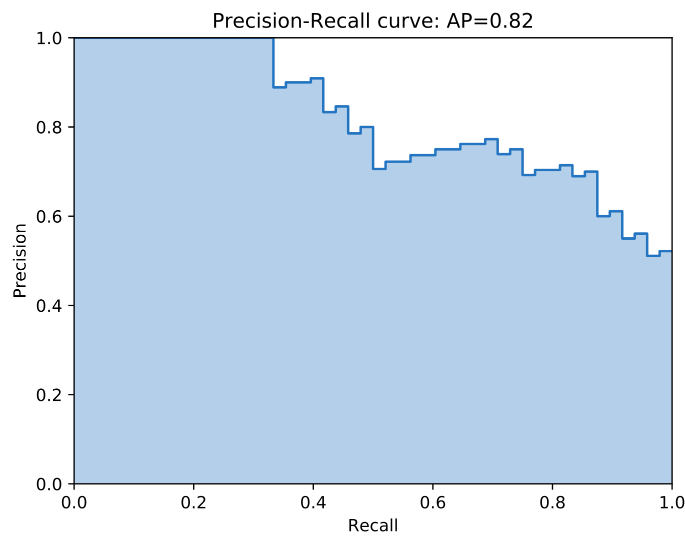

Figure 4.1: Precision-recall curve example.

4.3 Results

This section contains a final summary of classifier experiments. As it is usual in machine-learning, input data are divided into training and testing, in this case with 2:1 training:testing ratio. Model is fit on the training data and following predictions are afterward evaluated for testing data.

Each classifier was tested on multiple input data based on the ratio of positive and negative users, starting with 1:10, 1:100 and 1:1000. Also, as was mentioned in Section 3.5, classifiers are tested on input data enhanced with the port feature, the last experiment was to evaluate classifiers against feature matrix without this exact feature.

Gaussian Naive Bayes classifier

Naive Bayes classifier is one of the basic classifiers and as was expected, with growing ratio of positive and negative users its average precision gets lower. Classifier has promising results for 1:10 and 1:100 ratios, but having these ratios in real traffic is highly unlikely. For the 1:1000 ratio, its average precision is only 31%. The last figure shows a ratio of 1:1000 without the port feature. Even though the model has very similar performance, removing a port feature slightly increased average precision to 33% which is exactly opposite than was expected.

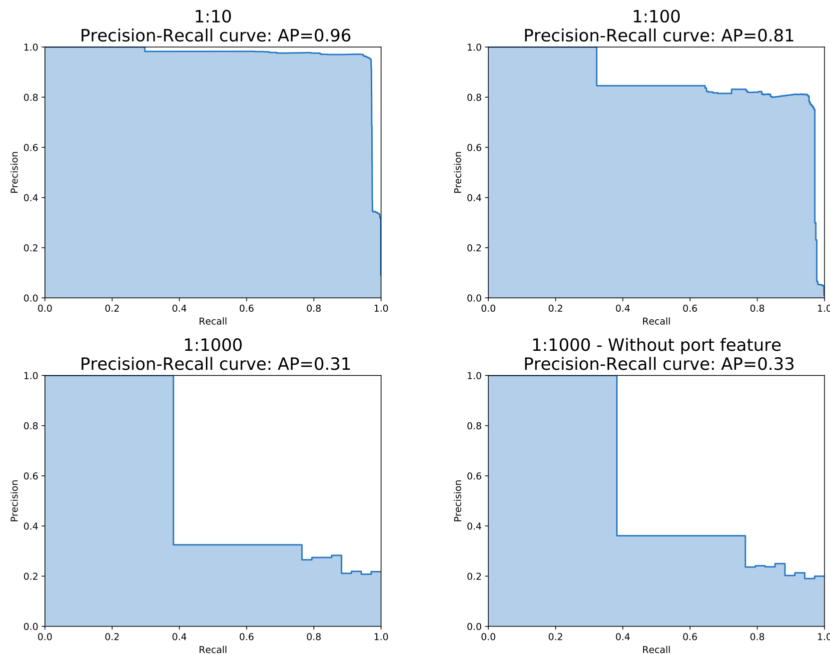

Figure 4.2: Naive Bayes classifier precision-recall curves.

Support Vector Machine

Support Vector Machine classifier, in contrast with Naive Bayes one, shows great performance not only for 1:10 and 1:100 ratios, where its average precision is around 100% but also for ratio 1:1000 has average precision almost 90%. In this case, removing port feature slightly decreased an average precision for 1:1000 user ratio.

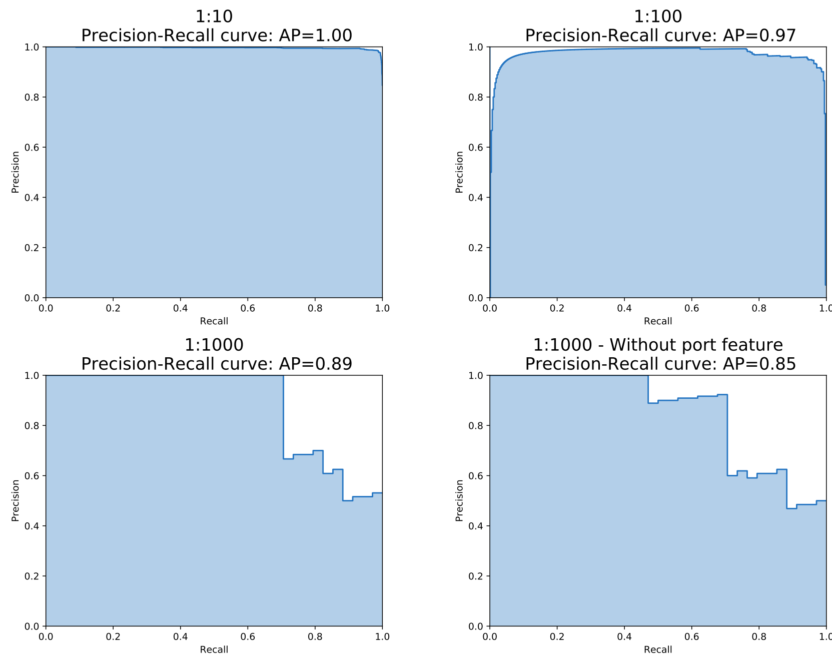

Figure 4.3: Support Vector Machine classifier precision-recall curves.

Random Forest classifier

The last used classifier was Random Forest. It results in the best precision-recall curves for all three used user ratios as its average precision is 100% for all of them. Random Forest, same as the Support Vector machine classifier, resulted in a decreased precision of negligible 1%. That means a Random Forest classifier looks very promising even if DNS tunneling software potentially changes source port for each DNS request.

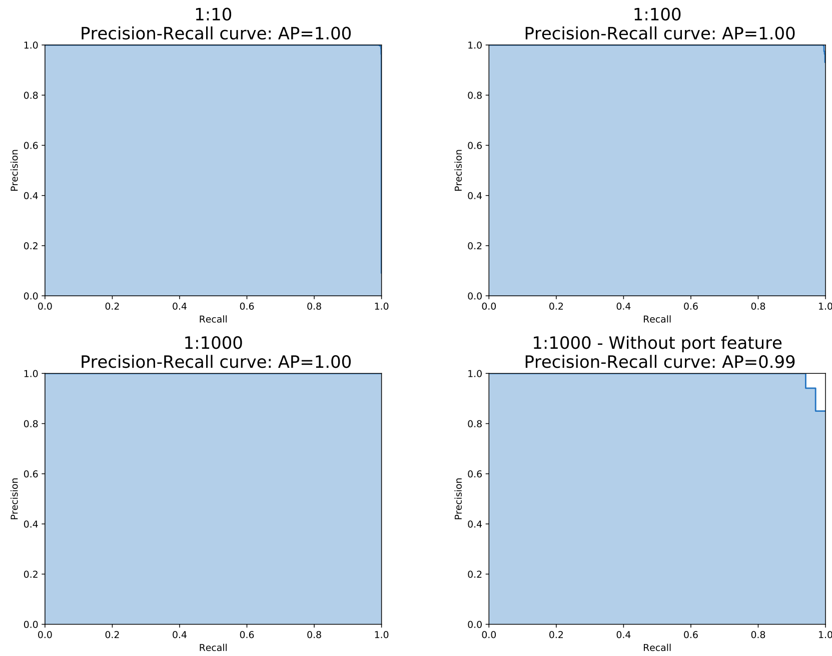

Figure 4.4: Random Forest classifier precision-recall curves.

Chapter 5
Implementation

Our detection process ended and this chapter shortly describes what programming or scripting languages were used. Also specify libraries and briefly describes their function.

Implementation process went through multiple stages and as our code is mostly experimental, it did not strictly follow conventional software engineering rules. This chapter also describes code-base (including testing) and as the last part, problems that occurred during implementation.

5.1 Languages and libraries

As the main language, have been chosen popular **Python**[28] scripting programming language in version 3.6. This language offers efficient libraries for manipulating with big data. In previous sections can also be noticed multiple scripts written in **Bash** command language.

- **Pandas**[29]

  The Package that provides fast and efficient manipulation of data with defining primary data structure so-called **Dataframe**.

- **Numpy**[30]

  The Standard library for data scientists adding support for big data and n-dimensional arrays or matrices. It offers multiple easy-to-use mathematical functions operating with these arrays. Pandas Dataframes use Numpy arrays for data handling.

- **Matplotlib**[31]

  Library designed for plotting data to 2D figures. Offers charts, histograms, scatter-plots, etc. with a goal of making them user-friendly.

- **Scikit**[32]

  Scikit-learn toolset offers user-friendly functions for data mining and analysis. Contains classification algorithms, data preprocessing, regression or clustering.

5.2 Code

Previous sections and chapters have mentioned two essential implemented algorithms. Overall, our code-base contains 7 files and 6 Python classes that are interlaced together. The whole implementation process was from the beginning managed by web-based **GitHub**[34] service for version control.

5.2.1 Modules

This section describes each modules purpose and its functions.

DataLoader

Defines methods for loading data to Pandas Dataframe structure. It also does some preprocessing like creating new matrix columns and dropping unnecessary ones. Defines functions for loading data from list structure which is used by our testing module.

PairManager

Contains already described Pairing algorithm (Section 3.3). It contains multiple functions used by the algorithm, main of them are:

- *pair_records* - Redistribute data into dictionary data structure.
- *insert_new_record* - Inserts new record based on *is_address_local* function.
- *update_existing_record* - Updates existing record.
- *is_packet_relevant* - Checks overall relevancy of new record with existing one.

DataMixer

Previously described module containing Mixing function(Section 3.4) for infecting real traffic with tunneled data. Also separates users and their request to dictionary for future analysis.

DataAnalyser

Analyses users with feature functions as described in Section 3.5 and results in feature matrix.

RecordManager

Record manager module is used mostly by PairManager, it offers functions for creating and updating records or pairs represented by Python dictionary.

AddressManager

Class containing crucial and decision-making part of Pairing algorithm. Function *is_my_address_local* checks if given IP is in range of local addresses.

5.2.2 Testing

As testing framework was used PyTest[33]. Multiple test cases were implemented for Pairing algorithm as it was the most crucial part and other parts often perform operations provided by other libraries.

The most important part is *pair_records* function as it performs whole data pairing operation. For this purpose was created loading data function from list structure because it was the easiest way to perform the testing comparison. Test list structure contains simulated records with different values to perform maximal available coverage.

The next tested is *is_packet_relevant* function that decides if the processed record will be paired with existing one. This function was tested on multiple cases with timestamps values, protocols, ports and IP address information.

5.3 Problems

Multiple problems were encountered during the implementation, most of them were caused by lack of experience in given field and were easily fixable. However, one problem remained.

Loading big data to DataFrame causes an increase of RAM usage and with multiple files is practically impossible to handle. The workaround was found in saving processed data into files, which is, from a production perspective, absolutely unacceptable. Problem is caused by a type of used data. Since it is completely common to work with files having millions of rows and multiple columns, they usually contain only numeric values. Used dataset contains a large number of string information and is considered as very memory-expensive.

Vectorization of input data should fix our problem. This process substitutes existing string values with numbers.

Conclusion

The focus of this thesis was to detect network users performing DNS tunneling with use of NetFlow protocol traffic data and machine-learning techniques. Thesis firstly described necessary theoretical background and described three existing solutions outlining their advantages and disadvantages. Following chapter took a practical approach and except technical background for realization of described four DNS tunneling tools that were made to make work to perform different experiments, such as creating tunneled SSH session or file upload.

Tunneling experiments were captured, converted to the NetFlow data and afterwards served as a data for infecting real traffic data with different user ratios of 1:10, 1:100 and 1:1000.

Infected data was analyzed by multiple feature functions resulting in a feature matrix where each row represented analyzed 5-minute traffic window for one single user. Matrix was used as an input to the machine-learning classifiers. Used ones were Gaussian Naive Bayes, Support Vector Machines and Random Forest.

Gaussian Naive Bayes (GNB) classifier showed good performance on ratios of 1:10 and even 1:100 with the minimal average precision of 80%. However, in real traffic, these ratios are highly unlikely to occur. Average precision for the ratio of 1:1000 resulted in 30%. This classifier does not have detection potential against created features.

Support Vector Machines (SVM) had almost 100% average precision for ratios of 1:10 and 1:100. Also for 1:1000 ratio had 90% average precision which is very promising.

Random Forest (RF) classifier, on the contrary with previous algorithms, resulted in the best results as it shows 100% average precision for each used ratio.

All three classifiers were also tested on matrix without feature analyzing use of source ports for 1:1000 ratio. GNB classifier resulted in an insignificant increase of average precision to 33% but for SVM and RF, both classifiers average precision got slightly decreased by units of percents. That means, using of port feature does not significantly affect detection performance for all three classifiers.

Multiple implementation problems were encountered but were not fixed

yet because of time reasons and are planned to be patched in future progress. First was low memory-performance as used values are mostly in string data type and can be converted to numeric values that are very memory-friendly. Except that, as all values will be numeric, pairing algorithm can use, instead of Python dictionaries, multidimensional Numpy arrays. Last planned step of future progress is to perform additional different DNS tunneling experiments to have bigger diversity in captured data for infecting real traffic dataset.

Appendix A

Bibliography

[1] Ellens W., Żuraniewski P., Sperotto A., Schotanus H., Mandjes M., Meeuwissen E. (2013) Flow-Based Detection of DNS Tunnels. In: Doyen G., Waldburger M., Čeleda P., Sperotto A., Stiller B. (eds) Emerging Management Mechanisms for the Future Internet. AIMS 2013. Lecture Notes in Computer Science, vol 7943. Springer, Berlin, Heidelberg

[2] Ahmed Almusawi and Haleh Amintoosi, "DNS Tunneling Detection Method Based on Multilabel Support Vector Machine," Security and Communication Networks, vol. 2018, Article ID 6137098, 9 pages, 2018. https://doi.org/10.1155/2018/6137098.

[3] Qi, C., Chen, X., Xu, C., Shi, J., & Liu, P. (2013). A Bigram based Real Time DNS Tunnel Detection Approach. ITQM.

[4] Server icon | Icon search engine. 2,425,000+ free and premium vector icons. [online]. Available from: https://www.iconfinder.com/icons/80980/server_icon

[5] Computer icon | Icon search engine. 2,425,000+ free and premium vector icons. [online]. Available from: https://www.iconfinder.com/icons/173187/computer_icon

[6] Amazon Web Services (AWS) - Cloud Computing Services. Amazon Web Services (AWS) - Cloud Computing Services [online]. Copyright © 2017, Amazon Web Services, Inc. or its affiliates. [cit. 18.05.2018]. Available from: https://aws.amazon.com/

[7] Joshua Anderson - FreeDNS [online]. Copyright © 2001 [cit. 18.05.2018]. Available from: http://freedns.afraid.org

[8] kryo.se: iodine (IP-over-DNS, IPv4 over DNS tunnel). kryo.se: code [online] [cit. 18.05.2018]. Available from: http://code.kryo.se/iodine/

[9] GitHub - Bowes, R. (2017, November 07). iagox86/Dnscat2 - [online] [cit. 18.05.2018]. Available from https://github.com/iagox86/dnscat2

[10] tcp-over-dns. AnalogBit [online] [cit. 18.05.2018]. Available from: http://analogbit.com/software/tcp-over-dns/

[11] dns2tcp | Penetration Testing Tools. Penetration Testing Tools - Kali Linux [online]. Copyright © 2018 [cit. 18.05.2018]. Available from: https://tools.kali.org/maintaining-access/dns2tcp

[12] OzymanDNS - Tunneling SSH over DNS · Rob 'mubix' Fuller. [online]. Copyright © 2017 [cit. 19.05.2018]. Available from: https://room362.com/post/2009/2009310ozymandns-tunneling-ssh-over-dns-html/

[13] Heyoka: your fast&spoofed DNS tunnel [online] [cit. 18.05.2018]. Available from: http://heyoka.sourceforge.net/

[14] Dnscat - SkullSecurity. [online]. [cit. 19.05.2018]. Available from: https://wiki.skullsecurity.org/Dnscat

[15] Lucas Nussbaum - TUNS. [online] [cit. 18.05.2018]. Available from: https://members.loria.fr/LNussbaum/tuns.html

[16] GitHub - FedericoCeratto/dnscapy - Pierre Bienaimé - [online]. Copyright © 2018 [cit. 18.05.2018]. Available from: https://github.com/FedericoCeratto/dnscapy

[17] GitHub - mdornseif/DeNiSe: DeNiSe is a proof of concept for tunneling TCP over DNS in Python. [online]. Copyright © 2018 [cit. 19.05.2018]. Available from: https://github.com/mdornseif/DeNiSe

[18] tshark - The Wireshark Network Analyzer 2.4.4. [online] [cit. 18.05.2018]. Available from: https://www.wireshark.org/docs/man-pages/tshark.html

[19] DNStunnel.de - free DNS tunneling service.DNStunnel.de - free DNS tunneling service [online]. Copyright © 2006 [cit. 18.05.2018]. Available from: https://dnstunnel.de/

[20] ALBITZ, Paul. a Cricket. LIU. DNS and BIND. 3rd ed. Sebastopol, CA: O'Reilly, c1998. ISBN 9781565925120.

[21] Merlo A., Papaleo G., Veneziano S., Aiello M. (2011) A Comparative Performance Evaluation of DNS Tunneling Tools. In: Herrero Á., Corchado E. (eds) Computational Intelligence in Security for Information Systems. Lecture Notes in Computer Science, vol 6694. Springer, Berlin, Heidelberg

[22] Introduction to Cisco IOS NetFlow - A Technical Overview - Cisco [online] [cit. 18.05.2018]. Available from: https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html

[23] GitHub - phaag/nfdump: Netflow processing tools.[online]. Copyright © 2018 [cit. 18.05.2018]. Available from: https://github.com/phaag/nfdump

[24] softflowd - fast software NetFlow probe. mindrot.org [online]. Available from: https://www.mindrot.org/projects/softflowd/

[25] Naive Bayes — scikit-learn 0.19.1 documentation. [online]. Copyright © 2007 [cit. 24.05.2018]. Available from: http://scikit-learn.org/stable/modules/naive_bayes.html

[26] Support Vector Machines — scikit-learn 0.19.1 documentation. [online]. Copyright © 2007 [cit. 24.05.2018]. Available from: http://scikit-learn.org/stable/modules/svm.html

[27] Random Forest Classifier — scikit-learn 0.19.1 documentation. [online]. Copyright © 2007 [cit. 24.05.2018]. Available from: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[28] Python [online]. Copyright © 2001 [cit. 19.05.2018]. Available from: https://www.python.org/

[29] Python Data Analysis Library — pandas [online]. [cit. 19.05.2018]. Available from: https://pandas.pydata.org/

[30] NumPy [online]. Copyright © 2018. [cit. 19.05.2018]. Available from: http://www.numpy.org/

[31] Matplotlib: Python plotting — Matplotlib 2.2.2 documentation [online]. Copyright © 2002 [cit. 19.05.2018]. Available from: https://matplotlib.org/

[32] Pedregosa, F., Varoquaux, G., Gramfort, A. & Michel, V. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

[33] pytest: helps you write better programs — pytest documentation. [online]. Copyright © 2015 [cit. 23.05.2018]. Available from: https://docs.pytest.org/

[34] The world's leading software development platform · GitHub. The world's leading software development platform [online]. Copyright © 2018 [cit. 23.05.2018]. Available from: https://github.com/

Appendix B

List of abbreviations

Abbreviation	Meaning
AWS	Amazon Web Services
DNS	Domain Name System
PCAP	Packet capture
SSH	Secure shell
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
RAM	Random Access Memory

Appendix C

Content of attached CD

The content of CD is divided into following files:

- **dns_tunneling_detection.pdf** - the bachelor thesis in PDF format
- **thesis_sources.zip** - TeX sources of bachelor thesis
- **dns_tunneling_detection.zip** - Python source files, Bash scripts and captured DNS tunneling experiments in NetFlow format