FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Detection of network traffic of TeamViewer
Student:	Tomáš Klatovský
Supervisor:	Ing. Tomáš Čejka, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2021/2022

Instructions

Study TeamViewer software and focus on its network traffic. Study flow-based network traffic monitoring and analysis.
Analyze the TeamViewer communication protocol using publicly available information and map the list of functionalities of the current software versions.
Create an annotated traffic dataset consisting of various captured TeamViewer sessions.
Design an algorithm for TeamViewer traffic recognition and probabilistic identification of the activities within a TeamViewer session based on packet-level characteristics.
Evaluate the performance of the designed algorithm using the created dataset.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Detection of network traffic of TeamViewer

Tomáš Klatovský

Department of Computer Systems
Supervisor: Ing. Tomáš Čejka, Ph.D.

May 13, 2021

Acknowledgements

I would like to thank Ing. Tomáš Čejka, Ph.D., and Ing. Karel Hynek for their guidance and help, and my family for their unwavering support.

Declaration

In Prague on May 13, 2021 ....

Citation of this thesis

Klatovský, Tomáš. *Detection of network traffic of TeamViewer.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Aplikace TeamViewer je jedním z nástrojů často používaných jednotlivci i organizacemi pro vzdálený přístup z důvodu správy zařízení, komplexní podpory zákazníků či spolupracovníků nebo pro zpřístupnění zdrojů uživatelům pracujícím z domova nebo jiného místa. Umožnění přístupu v takovémto rozsahu s sebou ale nese bezpečností rizika a vyžaduje pečlivé monitorování. Tato závěrečná práce se zabývá analýzou aplikace TeamViewer se zaměřením na její síťový provoz. Na základě této analýzy navrhuje postup pro detekci této komunikace a rozlišení druhů aktivity v jejím šifrovaném obsahu za užití strojového učení. Při detekci bylo dosaženo úspěšnosti 99.9 %, při pokusech o rozlišování druhů aktivity pak přinejmenším 84.9 %.

Klíčová slova TeamViewer, vzdálený přístup, IP toky, rozpoznávání aplikace, analýza provozu

Abstract

The TeamViewer application is one of the most prevalent tools for allowing individuals and organisations alike to utilize remote access to manage remote devices, provide complex support to customers or colleagues or to allow access to resources to users working from home or other remote location. Allowing this level of access, however, poses a severe threat to security and needs to be monitored closely. This thesis analyses the TeamViewer application, focusing on its network traffic. Based on this analysis, the thesis proposes a way to detect TeamViewer communication and distinguish between activities in its encrypted traffic utilizing machine learning. TeamViewer detection reached 99.9 % accuracy, while the experiments distinguishing between activities reached at least 84.9 %.

Keywords TeamViewer, remote access, IP flows, application recognition, traffic analysis

Contents

List of Figures

List of Tables

Introduction

Even before the recent pandemic-related lockdowns and the resultant rush to move various tasks online, individuals and organisations both were expanding their use of remote access functionality at a growing pace to be able to manage remote devices, provide complex support to customers and colleagues or to provide access to resources to users working from home or other remote location. As the interest in such tools increased, a number of applications emerged to serve this growing market. These include, among others, applications such as LogMeIn [1], AnyDesk [2], Splashtop [3] or, indeed, TeamViewer [4].

TeamViewer first released in 2005 and allowed users to simply share their screen with a partner, to transfer files to them or to gain direct control and interact with their desktop [5]. As development continued, more features were added over time, most notably audio/video conferencing. In 2021, TeamViewer is one of the more prevalent tools for accomplishing these tasks – the developer of the application claims to service more than 200 million users worldwide [6].

There are reasons other than TeamViewer's feature list that can also help explain its success. It offers a free version under the condition that the application is used in personal, non-commercial fashion [7] and supports a broad variety of platforms [8], allowing cross-platform use. Another important aspect is simplicity of use. Basic usage of the application requires essentially no setup after installation and no account is required either – the application automatically generates credentials which suffice for immediate use.

However, all of these conveniences coupled with the level of access granted by the application pose a severe threat to security and can be exploited in various ways. Many of the common abuses of remote access software take the form of social engineering, such as technical support scams, where the scammer impersonates a support technician and demands payment for fake services, attempts to gain access to the victim's system to steal sensible data (such as banking details) and so on. Another way remote access applications can be used in criminal activity is by incorporating them into RAT (Remote

Access Trojan) malware. As [9] notes, attackers will create special email attachments, web-links, download packages and various other means of luring the user into installing the software and allowing the attacker access. In the case of TeamViewer, malware examples such as TeamSpy or Skywyder are mentioned by [10]. This makes the ability to detect the activity of applications such as TeamViewer in a network a valuable tool for detecting suspicious behavior. That could be the application being used at all (e.g., if forbidden by policy) or being used at odd times, which might point to the presence of malware in the network or other malicious activity.

The first chapter of the thesis shall analyze the TeamViewer application (using the version available at the start of writing, 15.16.8), its usage and its network traffic. Related works will be discussed, with one investigating packet-level characteristics of TeamViewer communication and the other aiming to distinguish between different activities in TeamViewer's encrypted traffic based on its statistical features. Flow-based view of network traffic and relevant tools for its monitoring and capture will be explained.

Further chapters go into detail describing the experimental part of the thesis. This includes the creation and annotation of datasets, feature extraction and the final feature vector, machine learning models used and the design of the experiments conducted – binary and class-based classification. Binary classification explores the possibilities of TeamViewer traffic detection, whereas class-based classification attempts to distinguish between types of activity conducted in the encrypted traffic. An examination of the results follows.

Chapter 1

Background

This chapter aims to establish the required theoretical knowledge needed for the rest of the thesis. First, an introduction to flow-based network monitoring is given. Then, the TeamViewer application and its network behavior are analysed. With this having been established, two relevant works exploring the TeamViewer application are discussed.

1.1 Flow-based network traffic monitoring

Network monitoring is used for a number of purposes, ranging from troubleshooting and performance evaluation to aiding in network defence. Two general approaches to network monitoring are worth discussing in the context of this thesis: packet-based and flow-based. The basic differences between the two are the scale at which network traffic is viewed and consequently the level of detail of possible analysis.

The packet-based approach is concerned with every packet sent in the course of communication between devices, allowing for each unit of communication to be inspected. Different aspects of packet content may be inspected, ranging anywhere from simple IP header values to various payload contents. However, the packet-based approach becomes less relevant in analyzing encrypted traffic as the pertinent data becomes inaccessible. Another potential issue is the large file size of stored captures (considerably larger compared to the results of a flow-based approach) and the overall higher performance requirements. Packet-based approach will not be described further in this section as the thesis focuses primarily on flow-based monitoring. However, relevant basics shall be discussed in section 1.3.1, in which a related work makes use of Wireshark, an application using this approach.

Unlike the packet-based approach of preserving every detail, the flow-based approach instead generates a general summary of a communication, using a set of its attributes to uniquely identify it as a separate flow. The flow usually contains metadata such as timestamps or aggregated amounts of transferred data and it may be expanded to include further information (e.g., aggregated TCP flags).

1.1.1 IP flow definition

A network flow is described in RFC 3917 [11] as "*a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties.*" This set of common properties (such as the contained addressing information across various network layers) is called a flow key. A flow generally spans the timeframe between the first and the last packet with a given flow key being observed. The observation of the flow ends when certain predetermined conditions are met. Examples of such conditions include observing the natural end of a connection (e.g., TCP termination), an idle timeout (i.e., elapsing the longest allowed time between packets) or an active timeout (i.e., reaching the longest allowed duration of a flow) [12].

RFC 7011 [13] provides a common example of a flow key that may be used: source and destination IP address, source and destination port and the transport protocol used. As the combination is directional (one side is designated as the source and the other as the destination), it could be expected that a network flow specified in this manner groups packets headed in a single direction, source to destination. In RFC 5103 [14], such a network flow is described as unidirectional (uniflow). A bidirectional flow (biflow) is then comprised of packets headed in both directions – the biflow is effectively built from two uniflows, which share the same directionless attributes (e.g., protocol) and for which the source and destination attributes are the same, only opposite (e.g., the source IP address of the first is the same as the destination IP address of the second and so on). The biflow itself is assigned source and destination addressing information in the same format as a uniflow. Different methods of assigning direction to a biflow exist, however, the relevant method for this thesis does so based on the direction of the first packet observed in the biflow. For additional direction-specific data (e.g., packet count), the directionality is maintained, i.e., two separate values are kept, one for the forward direction (source to destination), and one for the reverse direction (destination to source).

1.1.2 IP flow meters and ipfixprobe

For the generation of flow data from either live or captured traffic, tools referred to as flow meters (sometimes also as flow exporters) are used. A flow meter is described in [12] as such: *"A flow meter generates flow data - which contains information about each connection observed on a network - from a stream of observed packets."*

In the course of the metering process, flow records are created for newly recognized flows, updated as more packets belonging to the flow are observed and ultimately, when the flow is considered over, its corresponding record is exported and removed from the flow record table [11]. There are various flow meters available for use, such as Cisco's Joy [15], YAF (Yet Another Flowmeter) [12], CICFlowMeter [16] or, indeed, ipfixprobe [17], which is used in the course of this thesis as the source of data for the TeamViewer analysis. Ipfixprobe is a flow meter implemented as a module in the NEMEA system, where the *"NEMEA (Network Measurements Analysis) system is a streamwise, flow-based and modular detection system for network traffic analysis. It consists of many independent modules which are interconnected via communication interfaces and each of the modules has its own task."* [18].

According to its documentation [17], ipfixprobe is designed to create biflows (identified by the source and destination IP address and port number, and the protocol used) either from input PCAP files or directly from live traffic on a network interface. If a PCAP file is used as a source, the entire file is processed and exported to provided output interface (this can mean saving to a file or passing the flow data to another module via, e.g., a Unix domain socket). If traffic is captured live on a network interface, flows are exported as they end. Command line options are available to alter the conditions for ending a flow. For example, `-c NUM` can set a limit to the number of packets to be captured before exporting and `-t NUM:NUM` can be used to change the active and inactive (idle) timeout values. By default, there is no limit to the number of packets, the standard active timeout is 5 minutes and the standard inactive timeout is 30 seconds, i.e., a flow will be ended if no activity related to it was registered in the past 30 seconds or if 5 minutes elapsed since its beginning.

A variety of parsing plugins is supported [17], allowing additional information (e.g., protocol-specific fields or flow metadata) to be extracted. Relevant plugins for this thesis are:

IDPContent The Initial Data Packets Content plugin adds to the default exported flow features two new ones: `idpcontent` and `idpcontent_rev`. These contain the first 100 bytes of the TCP/UDP payload in the first packet in the forward and reverse direction, respectively.

PSTATS The Packet Statistics plugin adds metadata about the packets observed at the beginning of the biflow – this includes lists of their TCP/UDP payload lengths, timestamps, TCP flags (0 if not applicable) and their directions in the biflow. By default, this is done for the first 30 packets.

1.2 The TeamViewer application

TeamViewer is most commonly known as a tool for the control and management of remote devices. In addition to that, it allows file transfer to and from those devices, implements audio and video conferencing in the Meetings feature and a text chat option is also offered. TeamViewer can currently be used on Windows, Mac OS, Chrome OS and various distributions of Linux and offers apps for Android and iOS [8], allowing broad cross-platform interaction. The developer lists officially supported Linux distributions, noting that the client may possibly work on others. Originally, TeamViewer on Linux would be used by running an adapted Windows version utilizing Wine, described as a compatibility layer capable of running Windows applications [19]. In August of 2018, the first native Linux client was released [20]. This allowed for the implementation of some missing features, bringing the Linux version closer to its peers on other platforms. At the time of writing, however, the Linux version still does not support the Meetings feature to allow for video or audio communication between users.

1.2.1 Basic usage

The following section describes the default ”out-of-the-box” behavior of the application. This can be further customized through a variety of security settings, such as user-defined passwords, allow/block lists etc.

When installed, a unique, device-specific TeamViewer ID (referred to as TID from now on) is automatically generated *”based on various hardware and software characteristics”* [21] alongside a temporary password. The TID is generally 9 or 10 digits long and is used as the identifier of the device, not the user. For audio/video conferencing, a meeting ID is generated instead at the meeting’s creation and is to be shared out to other users to allow them to join. The user can begin a session with just the destination device’s TID and its temporary password (for file transfer and remote desktop) or the meeting ID for conferencing. These features do not require the used to register an account and log in (the text chat feature does).

A unique TID will also be generated on a virtual machine, with the resulting TIDs seemingly belonging to different number ranges. Two physical devices and two virtual devices have had TeamViewer installed: a physical and a virtual Windows machine and a physical and a virtual machine with a Linux distribution (Linux Mint and CentOS 7). A 10 digit TID was generated on both physical machines, beginning in 1 48x xxx xxx. On both the virtual machines, a 9 digit TID was generated, with the CentOS device generating a TID beginning in 842 xxx xxx and the Windows device beginning in 682 xxx xxx. A deeper analysis would be required to reach conclusions with any degree of certainty, but this difference may be useful to differentiate between physical and virtual devices.

1.2.2 Security

The application is designed with ease of use in mind – once installed, there is very little required of the user to start using the application. A TeamViewer account or a user-specified password are optional as the application generates a random password alongside the TID. The standard length of this password is 4 characters, with lengths of 6, 8 and 10 also being an option. By default, this random password is kept until application restart, at which point a new password is generated. However, it can also be set to either automatically generate a new password or to offer this option after each session. The generation of such passwords can be disabled.

Assuming default configurations of the devices, the password and the device TID are sufficient credentials for starting, e.g., a remote control session. The weakness of the default password, coupled with the fact that the TIDs are visible in network traffic (see section 1.3.1), appears to be a severe security flaw. Should a device with TeamViewer in default configuration be left running for a long period of time, its TID could be eavesdropped and the password guessed. To help defend against such attacks, the developer claims in [21] that "*the application exponentially increases the latency between connection attempts*" in response to repeated failed authentication attempts. This persists until the correct password is entered. Although this may address the danger of brute-force attacks against the weak default password, it is not clear whether or not this measure might be abused to perform a denial-of-service attack of sorts.

While some information about the traffic can be gleaned from the proprietary protocol used to carry it, this relates specifically to information regarding the connection itself, as opposed to the activity contained in it. According to [21], the traffic is encrypted with AES(256-bit). The key is generated by one of the clients, then encrypted with a public key of the other client and shared. TeamViewer uses RSA public/private key pairs, with a server acting as a proxy for safely sharing public keys between clients (all clients know the server's public key).

1.2.3 Important options

While TeamViewer offers a wide variety of options to help configure and customize the user experience, two specific ones are important to discuss in the context of the application's network behavior - *Incoming LAN connections* and *Use UDP*.

LAN use

Using a TID or the meeting ID is the intended behavior for connections made over the Internet. In [22], the following is explained by the developer: it is possible to enable the application to also accept LAN connections, or to accept

LAN connections exclusively. If the LAN only option is chosen, TeamViewer disconnects from the Internet and TIDs can no longer be used. Instead, the application displays the device's local area network IP address for use as the new identifier of destination device. As this thesis focuses on connections made over the Internet, this option was disabled during the conducted experiments and was not explored further.

Allowing UDP

This option is enabled by default and recommended by the developer. Turning the option off does not disable the use of UDP entirely (it is still used especially for audio/video calls). Instead, having the option turned on allows a pair of devices to negotiate direct UDP communication. This is done via a maintained TCP connection to a server (discussed in section 1.2.4), through which IP addresses and port numbers are exchanged. This approach is discussed in RFC 5128 [23], where it's referred to as *UDP hole punching* – a method of starting communication between two devices each behind a networking device performing Network Address Translation (NAT).

Under normal circumstances, simply sending UDP messages to the other device will only result in them being discarded, as they belong to no UDP session known to the device running NAT. However, when the two devices (prompted by the server) attempt to send messages to each other at the same time, a UDP communication session will be opened for each of them at their respective NAT device. If the destination IP-address:port-number pair that caused the session to be opened matches the source IP-address:port-number pair of the incoming traffic, the traffic will be recognized as belonging to the session and allowed to pass through the NAT to the device behind it. This way, the server helps establish the communication by choosing the port numbers to use, distributing them to the devices and prompting them to start communicating at the same time. This communication seems to only use both destination and source port numbers higher than 30,000.

This is discussed in [10], where the claim is made that the connection data (IP addresses and UDP ports) the TeamViewer server distributes to the devices for setting up direct UDP communication can be observed in the content of a specific packet in the client-server TCP communication. However, this could not be replicated. Regardless, this UDP communication (or lack thereof) can still be observed. For example, the same file transferred under the same conditions with this option disabled will produce no observable UDP traffic. Instead, the transfer will be noticeable in a TCP connection to a TeamViewer server. With the option on, a direct communication between the two devices can be observed, containing roughly the same amount of data as in the previous TCP connection.

Table 1.1 demonstrates such comparison. The same file had been sent between devices in the same network, first with the option turned off, then on.

Table 1.1: Comparison of allowed and disallowed UDP

UDP setting	Protocol	Source IP	Destination IP	Source port	Dest port	Bytes	Bytes reverse
Off	TCP	192.168.1.103	188.172.246.174	1737	5938	150,830	**1,211,416**
On	TCP	192.168.1.103	188.172.246.167	1722	5938	93,975	**106,884**
	UDP	192.168.1.106	192.168.1.103	38433	60116	**1,143,074**	48,112

With the setting off, the bulk of the transfer took place through a proxy connection to a TeamViewer server (188.172.246.174) and no related UDP flow was observed. With the setting on, a UDP flow can be seen going directly between the two devices, carrying the bulk of the data. During both captures, no other flow had carried more than 42 kB. For the TCP connections, the relevant statistic for the file transfer is the number of bytes in the reverse direction as a client-initiated TCP connection to the server is used (the receiving client is thus considered the source of the flow). For the UDP communication, the sending client is considered the source in this instance.

1.2.4 Network behavior

Upon closer examination, TeamViewer network traffic exhibits certain specific characteristics which can help better recognize it. As mentioned in [24], the connection establishment is not encrypted and while some obfuscation is employed further in the payload, common values (referred to as magic numbers from now) identifying the proprietary protocol can be found in the TeamViewer protocol data unit - either 0x1724 or 0x1130. This is further discussed in section 1.3.1.

An aspect of the traffic that does not require payload inspection is mentioned in the developer's security overview [21]. Here, they go on to discuss (amongst other things) common characteristics of communication with a TeamViewer server (these do not apply to client-to-client UDP communication discussed in section 1.2.3). Namely, these include specific port numbers and IP addresses used.

Destination port numbers

When the client attempts to communicate with the server, it will do so on one of a set of predefined destination port numbers. Any UDP communication with a server will use port 5938. The primary option for TCP is also 5938, while some traffic, such as update checks, will use port 443, which is also the first port number the application will fail over to in case port 5938 is unavailable. Finally, port 80 may be used as a last resort at the cost of decreased performance (the overhead increases and automatic reconnection will not function). This behavior can vary on some platforms as is demonstrated in table 1.2 (original overview from [25]).

Destination IP addresses

TeamViewer servers use IP addresses from various IP address ranges, which are claimed to frequently change [21]. An address associated with a server is thus identified on the basis of its pointer record (PTR) in the Domain Name System (DNS). This record is used to look up the name associated with a provided IP address (this is commonly referred to as reverse DNS lookup). A TeamViewer server IP address will return a name in the format `*.teamviewer.com`. For example, at the time of writing, the reverse lookup of 213.227.186.144, a TeamViewer IP address observed in a capture, returns `ES-MAD-ANX-R013.teamviewer.com`. This more specific format, [country]-[city]-[provider]-R[xxx].teamviewer.com, can be commonly observed when inspecting IP addresses used by TeamViewer servers. Provider in this context refers to data center providers – in the example given, ANX likely refers to ANEXIA Internetdienstleistungs GmbH, a data center provider used by TeamViewer [26].

Table 1.2: Used destination port number overview

OS	**TCP/UDP Port 5938**	**TCP Port 443**	**TCP Port 80**
Windows	Yes	Yes	Yes
macOS	Yes	Yes	Yes
Linux	Yes	Yes	Yes
ChromeOS	Yes	Yes	Yes
iOS	Yes	No	Yes
Android	Yes	Yes	Yes
Windows Mobile	Yes	No	No

In addition to specific values, a pattern of behavior can also be observed. As the application starts, a DNS query is sent, requesting the IP address for `routerX.teamviewer.com`, where `X` stands for a number. Once the query is resolved, a client-initiated outbound TCP connection is established and maintained to allow communication with the server. To keep the connection open, roughly every 55 seconds, a keepalive pattern repeats. The client sends a packet with TeamViewer payload, which the server acknowledges with a TCP ACK reply. The same then happens in reverse and after about 5 seconds, the client sends another such packet and the server again acknowledges. According to information provided by the Wireshark dissector discussed in section 1.3.1, the first two TeamViewer payloads carry the `CMD_KEEPALIVEBEEP` command, whereas the last one carries TeamViewer's own `CMD_ACK` command.

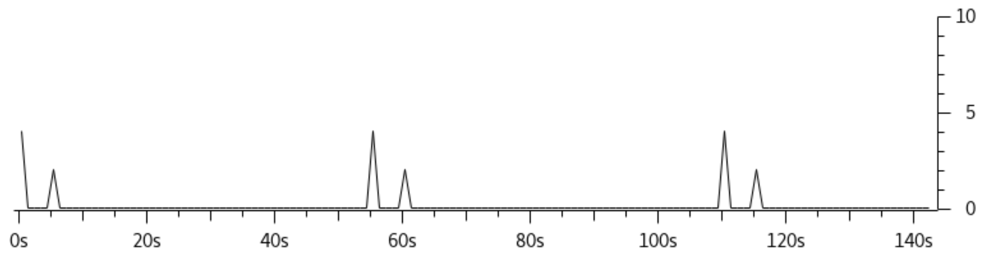


Figure 1.1: Observed keepalive packets (Wireshark graph)

1.3 Related works

1.3.1 Wireshark dissector

An analysis of the TeamViewer application had been conducted by Braden Thomas in [24] (archived version, reposted in [27]). To better understand the application's network behavior, he claims to have conducted a closer examination by partially reverse-engineering TeamViewer's Mac OS client (version 7). The information gleaned from the disassembly, alongside further information provided by the application's logs, were then used to create a set of plugins (called dissectors) for the Wireshark analyzer [28], allowing for the application's proprietary protocol to be analysed. Particularly relevant for this thesis is the central, lower-level dissector, which serves to interpret a common part of the TeamViewer payload. It is this specific dissector that shall be referred to as *the* TeamViewer dissector. Based on the value of what Thomas [24] marks as a TeamViewer command, the command-specific remainder of the data may be passed to one of the specialized higher-level dissectors. A dictionary of values and corresponding command names is provided.

Further development and analysis building on the original article was conducted by [10] four years later, however, their version of the dissector was not shared. The dissector is therefore explored in its original form. It was not analyzed thoroughly or developed further in the course of writing this thesis and related findings were verified independent of it. However, despite being an old prototype (the information and the dissector had first been shared in 2013), some of the claims presented in the initial article were still successfully reproduced. This suggests that to some extent, the dissector still functions and warrants further examination.

Wireshark

An example of packet-based approach to network monitoring, Wireshark can be described as a packet (or network protocol) analyzer. It allows the user to either capture network traffic on a given interface or to import a capture file in a variety of formats (at the time of writing, Wireshark uses the PCAPNG format as default for saving captures). Wireshark then attempts to extract as much information from the provided packet data as possible, interpreting the contents based on its knowledge of networking protocols. This is done through specialized dissectors – the developer describes this process in their guide to the application [29] as such:

"Each dissector decodes its part of the protocol and then hands off decoding to subsequent dissectors for an encapsulated protocol. Every dissection starts with the Frame dissector which dissects the details of the capture file itself (e.g. timestamps). From there it passes the data on to the lowest-level data dissector, e.g. the Ethernet dissector for the Ethernet header. The payload is then passed on to the next dissector (e.g. IP) and so on. At each stage, details of the packet are decoded and displayed. Dissectors can either be built-in to Wireshark or written as a self-registering plugin (a shared library or DLL)."

How the dissector works

The dissector is written in the lua programming language as a self-registering plugin. To use the plugin, the source file simply needs to be placed in Wireshark's plugin folder. The plugin defines the protocol in question for Wireshark by creating a Proto (protocol) object, which handles the data processing. To receive data from a lower-level dissector, a registration must be made in its associated table of subdissectors. For example, in order for the TeamViewer dissector to receive data from the lower-level TCP dissector, an entry must be added to the TCP dissector's `tcp.port` table, associating a port number (e.g., 5938) with the TeamViewer protocol. In a similar fashion, the higher-level, command-specific dissectors register in the subdissector table of the general TeamViewer dissector, associating themselves with a specific command value. Once the data from a lower-level dissector had been obtained, it is processed based on the algorithm defined for the protocol in the plugin.

According to the article accompanying the dissector [24], a magic number can be found in the first two bytes of the data. Two values have been observed - 0x1130 and 0x1724 (big-endian order). Following the magic number, a value specifying the type of command being sent can be found in the 3rd byte. The magic number values distinguish between two command formats which differ in header structure and in the use of obfuscation (0x1724 uses it almost always while 0x1130 only sometimes). The obfuscation is done by bit rotation: a subsection of the data (not including the aforementioned values) is rotated to the left by one bit. Exploring and describing this further, [10] claims that after reversing this process (where necessary), a broad variety of information (e.g., TIDs, session IDs, OS type etc.) can be found in the deobfuscated contents.

Observed TeamViewer behavior

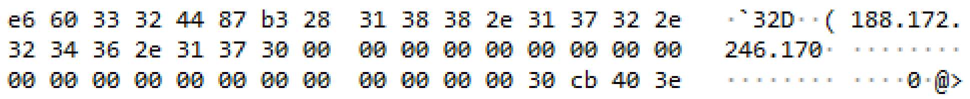
The most important aspect of TeamViewer communication gleaned from the dissector results is the presence of magic numbers in the payloads - either 0x1724 or 0x1130. Their placement is dependant on the transport protocol but mostly consistent (an exception is discussed in section 2.1.4). When TCP is used, the magic numbers are placed at the beginning of the payload, unless the payload is padded. Variable-length zero padding has been observed, pushing the values further into the payload and requiring the padding to be stripped. UDP on the other hand has been observed to consistently use the 12th and 13th byte. The first 11 bytes display further identifiable patterns. For example, a progressively incremented value can be observed in the first 4 bytes at certain times, suggesting datagram numbering (in little-endian ordering).

It is claimed by both [10] and [24] that the byte following the magic number contains a value representing the command being sent. Although this has not been thoroughly verified as it is outside the scope of this thesis, it seems to have merit. The dissector is accompanied by a dictionary mapping various command names to the values found in this specific byte. The corresponding command name of this value would on occasion match the suspected purpose of the payload it was found in.

For example, when observing the keepalive traffic described in section 1.2.4, the value contained in the first two payloads - 0x1B (27) - finds a corresponding dictionary match of `CMD_KEEPALIVEBEEP`. Similarly, when observing a file being sent, the value found in the bulk of the payloads - 0x6b (107) - corresponds to `CMD_DATASTREAM`. Further research in this area could help better understand the proprietary protocol and possibly help glean more information about the traffic from the pattern of commands being sent.

In their discussion of a particular command value, [10] mentions some identifying information of the communicating parties being clearly visible in a part of a TCP connection. When a device receives a message regarding a requested connection to it, both the source and destination TIDs can be seen (command value 0x16 (22) - `CMD_CONNECTTOWAITINGTHREAD`).

This can be replicated and is shown in the following example: the TIDs (in little-endian order) of the communicating devices can be seen, followed by the IP address (in plaintext) of the TeamViewer server acting as a proxy. The first four bytes contain the destination ID, the next four the source ID, followed by a section containing the IP address.



```
e6 60 33 32 44 87 b3 28  31 38 38 2e 31 37 32 2e   ·`32D··( 188.172.
32 34 36 2e 31 37 30 00  00 00 00 00 00 00 00 00   246.170· ········
00 00 00 00 00 00 00 00  00 00 00 00 30 cb 40 3e   ········ ····0·@>
```

Figure 1.2: TeamViewer ID values in payload (Wireshark)

This is demonstrated in figure 1.2. The little-endian value of the first 4 byte sequence, 0x323360e6 (842 227 942 in decimal), contains a CentOS virtual machine ID (connection destination). The value of the following sequence, 0x28b38744 (682 854 212 in decimal), contains a Windows10 virtual machine ID (connection source). Finally, the IP address 188.172.246.170 can be seen – this address was verified at the time of writing as a TeamViewer server address through its PTR record (`AT-VIE-ANX-R007.teamviewer.com`). In this manner, information identifying communication between specific devices can be extracted from the traffic.

1.3.2 TeamViewer encrypted traffic analysis

The only publication I am aware of that focuses on identifying the activity in encrypted TeamViewer traffic was authored by Altschaffel et al. [30] in 2013 and is based on TeamViewer version 7 (current version at the time of writing is 15). The paper seeks to introduce a method for distinguishing between classes of encrypted traffic based on statistical properties of its constituent network traffic flows. Binary classification (TeamViewer traffic or not) is not explored. The work uses the open source WEKA machine learning software [31].

Four classes of encrypted traffic are recognized: audio, video and text chat conversations and file transfer. This approach differs from the selection used later in this thesis, where instead three principal classes are recognized: file transfer, conferencing (represented by audio conversations and screen-sharing) and remote desktop control.

Network flow data is captured for each chosen class and a vector of features is generated to create a dataset (this dataset has not been publicized). In processing the data, a self-developed feature extractor is mentioned to have been used to generate a variety of information at three different levels of detail: local, window-based and global. Local features contain packet-specific information, such as packet length, capture timestamp etc. Both window-based and global features concern themselves with statistical data about the network flow at different scope. Global features take into account the entire network flow, whereas window-based features consider smaller sections, broken down either by the number of packets observed or by time elapsed. For generating these features, a sequence of packet lengths and times (SPLT) of unlimited size is used. Contrary to that approach, this thesis uses only the first 30 packets, as it might serve as a basis for a NEMEA module implementation. Using an SPLT of unlimited size is not feasible at high network speeds due to prohibitive performance cost, which is why NEMEA (and this thesis) limit the SPLT to 30 packets.

Certain features are only used to derive others and then dropped. This includes features containing total values (such as the aggregate of packet lengths across the whole flow) which are used to derive values relative to e.g., time or chosen window-size. Ultimately, a set of machine learning models is trained and tested on a 66 % and 34 % splits of the dataset, respectively. Across all the models, the class of encrypted traffic is claimed to be identified correctly in the vast majority of cases (over 99.9 %). In conclusion, the authors discuss the need to broaden the variety and size of the dataset and conduct further research into feature selection.

Chapter 2

Experiment design and dataset creation

The experimental portion of this thesis is limited to a certain degree by the availability of relevant network data. Two sources are currently available: bulk capture option made possible in the CESNET2 [32] network and data captured by hand in a local network. While it is possible to identify TeamViewer network flows in a provided bulk capture, the specific activity contained in the encrypted payload cannot be determined. As such, the data cannot be used in training or testing a machine learning model for distinguishing between contained activities.

Therefore, the decision was made to conduct two separate experiments in machine-learning classification better suited to available methods of capturing network flows. The first experiment explores binary classification for distinguishing between TeamViewer and other network traffic, utilizing the opportunity of capturing a larger amount of network flows in the CESNET infrastructure. The second experiment, a proof-of-concept for activity classification, aims to distinguish between three important classes of activity within TeamViewer: file transfer, remote control and conferencing.

To support the experiment design, a smaller flow capture had been collected locally to use in distinguishing between activity in encrypted TeamViewer traffic, and larger captures conducted at CESNET had been obtained for binary classification. Two datasets had been created for binary classification (one for training and one for testing), and one for the activity classification. For the sake of brevity, the data gathered for the binary classification experiment in the CESNET infrastructure shall be referred to as *backbone* traffic data, whereas the data gathered locally for the activity classifier shall be referred to as *activity* traffic data.

All the captured network data had been processed and exported by ipfixprobe and saved in the CSV file format. The resulting network flows were further explored and manipulated in an interactive notebook with the Python programming language using the following libraries: Numpy [33] and Pandas [34] for data analysis and manipulation, Scikit-learn [35] for machine learning and related activities (e.g., feature evaluation) and finally, the Feature Exploration Toolkit (FET) library [36]. FET was developed for the master thesis by Daniel Uhříček [37], implements further feature extraction (discussed in section 2.1.6) and allows for convenient exploration of the dataset utilizing scikit-learn functions.

2.1 Backbone traffic

The data for both experiments is processed in a similar matter with some key differences. Therefore, the backbone traffic datasets shall be described fully and the activity dataset shall be discussed in summary of relevant differences between it and the backbone ones.

2.1.1 Traffic captures and the capture environment

The environments in which the data was captured differ to a large extent. The backbone data is fed into ipfixprobe directly from a network interface at the outer perimeter of the CESNET2 public network infrastructure. This means all the potentially personal information, including the destination and source IP addresses, has to be anonymized. It is therefore impossible to use the destination IP address in labeling the flows. Two specific kinds of capture were conducted:

- TCP traffic using ports 443 and 80, limited to known TeamViewer IP addresses (found locally and verified by their PTR record)
- TCP traffic using port 5938, with no IP address limitation

For each, two separate captures were done, roughly a week apart. The port and protocol combinations used are known TeamViewer combinations (see section 1.2.4). Traffic capture of TCP on ports 443 and 80 had been limited to known TeamViewer IP addresses. This was done because TeamViewer will primarily use port 5938 for most of its traffic while the failover ports 443 and 80 are used for other very common purposes such as HTTPS and HTTP, respectively.

An exploratory UDP capture on destination port 5938 had also been conducted to examine the traffic. However, far fewer flows compared to the TCP captures had been found, the vast majority of which had been identified as TeamViewer traffic (roughly a 93:7 split). Furthermore, unlike with TCP, no existing datasets containing similar traffic and captured in the CESNET2 network are available. The gathering of sufficiently large dataset of UDP traffic similar to TeamViewer's would thus be considerably more difficult and require further analysis of other similar applications, expanding the scope of the thesis. Under the supervisor's advice, the decision had been made to limit the binary classification experiment to TCP traffic.

2.1.2 Live data capture and flow export

To begin with, live network traffic was sourced for ipfixprobe to process. For the backbone data, ipfixprobe captured live traffic directly from a network interface and processed it with the IDPContent and PSTATS plugins activated. Each plugin would generate a separate result, which would then be exported to its own separate output, containing plugin-specific flow features (the outputs were later combined for further processing). These flow data outputs were exported into an intermediary product, which was then processed by the NEMEA module *logger* [38]. This module allows for transforming the results into a desirable format, in this case a CSV file.

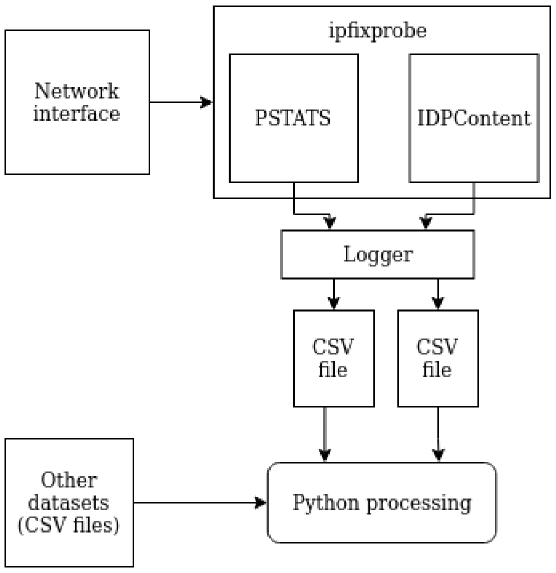


Figure 2.1: Sources of network flow data

2.1.3 Flow aggregation

When capturing the backbone data, ipfixprobe was running under default settings, including a 30 second idle timeout timer. This caused an issue with TeamViewer's connection-keepalive mechanism. As described in section 1.2.4, TeamViewer maintains a TCP connection to a server (unless set to LAN only). A small exchange of packets takes place between the client and the server roughly every 55 seconds, with the exchange lasting about 5 seconds (first packet to last packet). This means that the time period between the last packet of the previous set and the first packet of the next set is about 50 seconds. As the idle timeout value effectively specifies the largest time period of inactivity before a flow is considered over, an inactive connection to a TeamViewer server only sending keepalive messages will continuously generate new flows until some activity is initiated.

This can be avoided by setting the idle timeout value higher (using the `-t active_timeout_value:idle_timeout_value` option). However, changing these settings in the CESNET2 monitoring setup is not feasible as it would cause a disruption in monitoring operations. Therefore, a post-capture aggregation of the data was done locally to reduce the degree to which the flows are broken up. A timeframe of 5 minutes was chosen, same as the ipfixprobe default for active timeout (i.e., the longest possible duration before exporting). The 5 minute value is established in ipfixprobe as a reasonable compromise between measuring accuracy and latency.

For illustration, TeamViewer was left running idle and its traffic had been captured via Wireshark. The resulting PCAPNG file was processed by ipfixprobe, first using the default idle timeout value of 30 seconds, then using an increased timeout value of 5 minutes. In table 2.1, this is shown using five separate flows. All of these flows shared the same addressing information (destination and source IP-address:port pairs and protocol). The first four flows are taken from the 30-second idle timeout export, the fifth one is their aggregation in the 5-minute export.

It can be seen that during the first flow, another activity had taken place as the last packet was seen roughly 39 seconds after the first one. In the following three flows, however, each time roughly 5 seconds would elapse between the first packet and the last packet being observed. Following that, it would take further 50 seconds for the next flow to begin.

Table 2.1: Four separate flows aggregated into one

Time first	**Time last**
15:10:35.351	15:11:14.412
15:12:04.382	15:12:09.475
15:12:59.408	15:13:04.511
15:13:54.442	15:13:59.544
15:10:35.351	15:13:59.544

2.1.4 Flow filtering and labeling

Before proceeding further, the obtained flow captures are filtered to create a more representative set of the intended flows. The backbone flow captures are first aggregated, then filtered to only include flows with a basic minimum number of packets sent in either direction and at least a basic minimum duration (8 packets and 1 second of duration, specifically). This is done to reduce the effects of an uncontrollable capture and of the limitations of the capture environment so as to produce a set of flows more representative of both TeamViewer's traffic and of other traffic likely to be found alongside TeamViewer.

Another issue addressed by further filtering of the captured flows is concerned with the directionality of TeamViewer traffic. The direction of a biflow (i.e., which side is considered the source of the biflow) is based on the first packet observed in that biflow. As the observed flows could not be controlled, some of the TeamViewer flows would be recorded as going in the opposite direction. So as to maintain the directionality of flows (client-initiated connections towards the server), TeamViewer flows found to be headed in the opposite direction were filtered out.

After the aggregation and filtering are done, the flows are labeled. Two basic methods for recognizing TeamViewer flows had been established: searching for the magic numbers 0x1130 and 0x1724 (discussed in greater detail in section 1.3.1) in the TeamViewer protocol data unit, and by seeking a developer-defined naming pattern in the PTR record of a given destination IP address (discussed in section 1.2.4), respectively. As the IP addresses are anonymized in the case of the backbone data, only magic numbers are used to distinguish between TeamViewer (”TV”) and other (”OT”) traffic. For TCP, an attempt is first made to remove initial zero padding, then the first two bytes are checked. If a magic number is found in one of the directional IDPContent values, the flow is labeled as TeamViewer. Traffic which had been labeled as non-TeamViewer had been retained as a source of other (”OT”) traffic for the datasets.

Absent magic numbers

During exploration of captured backbone data, it was found that rarely, a TCP flow would contain a magic number in one directional IDPContent feature and a different value in the other. In some cases, the absence of a magic number could be explained by a padding so long the magic number would be, entirely or in part, pushed past the first 100 bytes. Because the IDPContent plugin cannot be easily reconfigured to obtain more bytes in the CESNET2 capture environment, should two such packets ever be the first directional packets to be observed in a flow, the flow will be mislabeled.

A subset of these flows did contain actual values different from known magic numbers. Further examination of locally captured traffic in Wireshark had eventually revealed packets belonging to known TeamViewer connections in which TeamViewer payloads would not contain a magic number. However, no shared values could be found.

This was examined closer using the backbone data captures eventually used to create the training dataset. Two captures had been conducted, one on

Table 2.2: Labeling TCP captures using magic numbers

	Total	**Bidir**	**Unidir 1 payload**	**Unidir 2 payloads**	**No magic, has payload(s)**	**No magic, no payloads**
5938	5,603,258	3,868,053	1,030,174	176	6,663	698,192
443 & 80	3,481,268	1,338,877	351,829	1,203	214,597	1,574,762
Sum	9,084,526	5,206,930	1,382,003	1,379	221,260	2,272,954
Sum/Total	1	0.57316	0.15212	0.00015	0.02435	0.25020

port 5938, one on ports 443 and 80 (see section 2.1.1 for more detail). Across these two captures, a little over 9 million flows had been captured. Among those, 1379 flows had been found that had contained a magic number in one of their IDPContent values and something else in the other (this still includes cases of the aforementioned extensive padding).

In table 2.2, the captured flows are classified based on observed magic number content. Magic numbers could be found in both directions (Bidir), in one direction (Unidir) or in neither direction (No magic). For unidirectional flows, a distinction is made between flows in which only one payload had been provided by IDPContent, and those where payloads in both direction had been available. Similarly, for flows where no magic was found, a distinction is made between flows where at least one payload had been available, and those where no payload was available. Given the minimal representation of potentially problematic flows in the data, the decision had been made not to explore them further in the course of this thesis.

2.1.5 Blending with other known captures

Finally, the flows obtained from the backbone data (both TeamViewer and other) had been blended with other flow data to further expand the variety of traffic. The added flow data was obtained from various other available datasets which had been captured under the same conditions and contain known flows of certain protocols. The only exception is the PSTATS plugin configuration – in some captures, the plugin had considered the default 30 packets, in others, different values were used. Therefore, only the first 30 values in each PSTATS generated list (lengths, directions, times and TCP flags) were kept for all data. Otherwise, the flows had been processed in the same manner as in the TeamViewer captures and contain the same information except for the IDPContent values. This is not an issue as those are only used for labeling the flows, which in this case can be done simply based on the origin of the flows. To better challenge the classifier, certain datasets had been sampled for their potential similarity while others were sampled to broaden the variety of contained traffic. The following types of traffic had been added to the dataset:

TLS TLS is similar in the sense that it carries encrypted traffic containing a variety of content, including activities similar to TeamViewer's. This includes video streams (from streaming services such as Twitch or Netflix) and file-transfer traffic (e.g., FileSender, OneDrive). Datasets known to contain this specific traffic had been sampled.

HTTP(S) As TeamViewer uses ports 443 and 80, HTTPS and HTTP traffic had also been sampled to include more traffic likely to be encountered alongside TeamViewer in real networks.

Other Samples of other protocols (IMAP, POP3, SIP, SMTP, SSH) had been included to broaden the variety of traffic.

TLS traffic had been sampled from data used in writing [39], SSH traffic from both [40] and [41] and HTTP, HTTPS and other traffic from [41]. Different capture files were sampled for the two datasets. Where only one file was available, it was split before sampling so as to prevent the same flow from entering both datasets. The `DataFrame.sample(numberOfSamples)` Pandas method was used for the sampling. The method allows the user to specify whether or not a row can be sampled more than once. This was disallowed.

2.1.6 Full feature set

At this stage, a basic set of flow features had been generated by ipfixprobe and the plugins, and flow duration had been calculated to aid in filtering. However, while some of these features are retained in the full feature set, others are either not used at all (source and destination MAC addresses, `link_bit_field` and `dir_bit_field`), or are used only for deriving further features by utilizing elements of the FET library. Both unused and intermediary features are then removed before the final feature set is finished. Notable intermediary or auxiliary features include:

Source and destination IP address Irrelevant in backbone data (as addresses are anonymized), but used to help identify TeamViewer flows in the activity data capture.

Time first, time last The timestamps of the first and the last packet. Used to calculate flow duration.

PSTATS-generated features These are related to the first 30 packets seen in the biflow and include payload lengths, timestamps, directions and contained TCP flags. Each of these is kept as a list of values generated in the order of packet arrival. These are used to derive more specific features later in the processing. If the flow is not using TCP, the TCP flag values will default to 0.

IDPContent and IDPContent reverse The first 100 bytes of the TCP or UDP payload. Taken from the first packet in the forward (source-to-destination) and reverse (destination-to-source direction) direction.

Used for labeling the flows based on the presence of a TeamViewer magic number in the payload.

Once the flow content of the datasets was finalized, further features were generated and added to the dataset using the FET library. The previously calculated flow duration was used to derive relative packet and byte rates based on known ipfixprobe counts of packets and bytes sent. This includes forward, reverse and total rates. Duration is kept as a standalone feature in the dataset. Ultimately, after all unused and intermediary features are removed, the full feature set consisting of a total of 57 features kept (58 including the label) is finalized.

Table 2.3: Full feature set

Feature category	**Specific features**
Connection-related	src_port, dst_port, tcp_flags, tcp_flags_rev
Flow-related, absolute	packets, packets_rev, bytes, bytes_rev, duration
Flow-related, relative to flow duration	bytes_rate, bytes_rev_rate, bytes_total_rate, packets_rate, packets_rev_rate, packets_total_rate
TCP-flag counts (first 30 packets)	fin_count, syn_count, rst_count, psh_count, ack_count, urg_count
TCP-flag ratios (first 30 packets)	fin_ratio, syn_ratio, rst_ratio, psh_ratio, ack_ratio, urg_ratio
Packet-size statistics (total, forward, reverse)	lengths_min, lengths_max, lengths_mean, lengths_std, fwd_lengths_min, fwd_lengths_max, fwd_lengths_mean, fwd_lengths_std, bwd_lengths_min, bwd_lengths_max, bwd_lengths_mean, bwd_lengths_std
Inter-arrival time statistics (IAT) (total, forward, reverse)	pkt_iat_min, pkt_iat_max, pkt_iat_mean, pkt_iat_std, fwd_pkt_iat_min, fwd_pkt_iat_max, fwd_pkt_iat_mean, fwd_pkt_iat_std, bwd_pkt_iat_min, bwd_pkt_iat_max, bwd_pkt_iat_mean, bwd_pkt_iat_std
Normalized IAT statistics (total, forward, reverse)	norm_fwd_pkt_iat_mean, norm_fwd_pkt_iat_std, norm_bwd_pkt_iat_mean, norm_bwd_pkt_iat_std, norm_pkt_iat_mean, norm_pkt_iat_std
Labeling	label

Connection-related Source and destination port numbers allow the direction of flows to be identified in backbone data and are used for preliminary filtering of activity data as non-TeamViewer flows are dropped. Port-based features are further discussed in section 2.1.6. The `tcp_flags` and `tcp_flags_rev` features contain aggregates of TCP flags observed in the course of a flow. As the presence of a flag is represented by a bit value in a specific position, the aggregation can be achieved using a bitwise OR.

Flow-related Byte and packet related features are used both as-is and for calculating their respective rates relative to flow duration. The duration of the flow is calculated as the time elapsed between the first and the last packet observed. Duration is used for three different purposes: to aid in basic flow filtering, to derive features relative to time and as a standalone feature.

TCP flag counts and ratios Not to be confused with the actual `tcp_flags` features, rather than aggregate all flags observed, these features count the number of occurences of individual flags. The following flags are tracked: ACK, FIN, RST, PSH, SYN and URG. The PSTATS plugin generates a list of TCP flag values contained in the first (by default) 30 packets of the biflow. FET then adds the counts of each flag and the ratio of a specific flag found to total number of observed packets as features.

Packet-size statistics The minimum, maximum, mean and the standard deviation of payload lengths are calculated based on the PSTATS-generated list of packet payload lengths. Both these and the following inter-arrival features are calculated both in separate flow directions (forward/reverse) and as a total.

Inter-arrival time statistics (IAT) The minimum, maximum, mean and the standard deviation of packet inter-arrival times (i.e., times elapsed between packet arrivals) are based on the PSTATS list of packet arrival timestamps. Also included are the mean and standard deviation values of normalized inter-arrival times. The normalization is done as a binary choice where times shorter than a given length are evaluated as 0 and longer ones as 1. The default breakpoint is 5 seconds.

Labeling The label is generated using the methods discussed in 2.1.4.

Port-based features

An inherent characteristic of TeamViewer traffic should be discussed in this context, specifically the issue of used port numbers. As mentioned in section 1.2.4, TeamViewer uses destination TCP ports 5938, 443 and 80 for communication with its servers. While non-TeamViewer flows destined for ports 443 and 80 are easy to capture as these ports are used for HTTP and HTTPS traffic, non-TeamViewer traffic headed for port 5938 is considerably more difficult to find. This leads to a heavy imbalance in the dataset – TeamViewer traffic is largely destined for port 5938, whereas the majority of the other traffic is not. For example, in the training set, roughly 85,000 of TeamViewer flows (about 74.7 % of the total) have a destination port number 5938. Only 845 non-TeamViewer flows share the same destination port number, a negligible amount. Port numbers are therefore a potent tool for identifying TeamViewer traffic but may possibly diminish the impact of other features.

2.1.7 The final outcome

Overall, two datasets have been created – a training and a testing one. The datasets were saved and manipulated in the form of Pandas DataFrames using the Python module Pickle [42]. Pickle is used for object (de)serialization, allowing a Python object (such as a Pandas DataFrame) to be saved to and loaded from a file. To that end, the *.pickle file format is used. However, their final form is saved in the CSV format so as to provide a generic, easily accessible format.

The creation of both the datasets follows the same process outlined in this chapter. Two pairs of network captures, separated by over a week of time, had been conducted to gather the initial data (section 2.1.1). Each pair was then merged into one, had its flows aggregated (section 2.1.3) and was filtered and labeled (section 2.1.4). Both pairs had then also been expanded with other relevant traffic (section 2.1.5), however, a different ratio of TeamViewer traffic to other traffic had been chosen.

The training set was made to contain roughly the same amount of TeamViewer and other traffic to allow the models to learn using a larger amount and variety of required flows. The testing set, on the other hand, aimed to use a ratio that would somewhat better reflect the amount of TeamViewer flows in captured traffic. To this end, the ratio of traffic in this set is roughly 1 to 10 (TeamViewer to other traffic). This means that in the training set, 113,602 TeamViewer and 109,781 non-TeamViewer flows are contained. In the testing set, a total of 9,745 TeamViewer and 93,603 other flows are contained.

Table 2.4: Class distribution in binary classification datasets

	Training dataset	**Testing dataset**
TeamViewer flows	113,602	9,745
Other flows	109,781	93,603

2.2 Local traffic

As the available network traffic data is difficult to obtain in large amount by hand, only one smaller dataset had been assembled for the activity classification experiment as a proof-of-concept for a broader investigation. The activity data was captured in the Wireshark application (ran on one of the communicating devices), saved as a PCAPNG file and then passed to ipfixprobe. The central device on which captures had been conducted was a Windows desktop, communicating with a Linux Mint laptop (for remote control and file transfer), a virtual Windows machine (for one-sided audio traffic) and another desktop computer on the Internet (when capturing two-sided audio traffic). The *Use UDP* option, discussed in section 1.2.3, had been turned on during the captures. This was done because the conferencing feature uses UDP regardless of the setting. With the setting on, the other classes also use UDP in their traffic. Finally, the PSTATS plugin had been set to use the default number of packets, which is 30.

A set of captures had been conducted for each of the classes. Between 90 to 100 captures had been conducted for each class, leading to roughly the same number of flows per class remaining after processing and filtering (about 250, including TCP and UDP flows).

Table 2.5: Class distribution in activity classification datasets

	CO	**FT**	**RC**
TCP flows	171	156	134
UDP flows	70	115	97

Some of the activity was chosen because of possible similarity to other classes, so as to better test the machine learning models. For example, the screen sharing aspect of conferencing potentially being somewhat similar to the desktop-sharing during a remote control session, or unidirectional audio traffic being more similar to a file transfer than a call in which both sides talk. The specifics of the captures are as follows:

Conferencing The conferencing class contains traffic in which screen sharing and audio calls were conducted, both separately and at the same time. The traffic had been captured on a device connecting to a meeting hosted by another device. The audio calls could be bidirectional (i.e., both sides speak at some point) or unidirectional (only one side talks during the call). Only unidirectional calls were combined with screen-sharing. For unidirectional activity (screen-sharing or one-sided audio calls), the directionality of traffic (i.e., which side of the communication speaks or shares the screen) had been varied. Both conferencing and remote transfer captures would last roughly between 30 seconds to 3 minutes.

Remote control Traffic was captured both when connecting to a remote device and when being connected to by a remote device. To further vary the traffic, different activity had taken place in these remote sessions, ranging from more intensive (playing video or aggressively moving a window across the screen) to less intensive (typing or motioning with the cursor) to simple inactivity.

File transfer Transfers of files (both sending and receiving) had been captured, using mostly file sizes between 3–5 MB, although in a smaller subset of captures, a 50 MB file had been sent. A variety of formats have been used, including a ZIP archive, a BMP image or a simple TXT file. In most captures, a single file had been sent, but in a smaller set of captures, multiple files had been sent in a row with varying intervals between them.

Because the conferencing feature was observed to use UDP even with the *Use UDP* option off and because the option is on by default, the decision had been made to turn the option on so as to allow file transfer and remote control to also use UDP.

2.2.1 Filtering and aggregation

As the flow exporter is run locally, it is possible to increase the idle timeout value and consequently remove the need for post-capture flow aggregation. As for filtering, activity data is approached somewhat differently as only TeamViewer traffic is to be kept. Therefore, the traffic is first filtered to remove non-TeamViewer traffic. This is done using the protocol value (only TCP(6) and UDP(17) are kept) and the destination port value (for TCP flows only). After the dataset had been narrowed down to flows which can contain TeamViewer traffic, the IP addresses are examined to determine which ones belong to TeamViewer servers (see section 1.2.4). This information, coupled with the magic numbers contained in the IDPContent features, is then used to both label and consequently filter the traffic to only contain TeamViewer flows.

Flows in which only one or two packets had been sent were then filtered out as they do not carry the conducted activity and would only confuse the classifiers. Most of these flows share a common set of characteristics, including having a destination IP address of a TeamViewer server, the destination port number 5938 and a source port number over 40000. A difference can be observed between such flows based on the activity being conducted. Whereas during conferencing captures, these flows only ever carry a single payload of a specific length to the server, during file transfer and remote control captures, the server and client can exchange one or two packets and at least two different payload lengths can be observed. The purpose of this traffic is unknown, although in the case of remote control and file transfer it may be the establishment of direct UDP communication as described in section 1.2.3.

A subset of these flows differ in that they can observed travelling between the local private and public IP addresses, using exclusively both destination and source port numbers over 30000. This is likely a part of the aforementioned direct UDP communication set up, with the capture of such flows being a result of capturing the network traffic on the same device that is being used to generate it. Regardless, these are also filtered out.

2.2.2 Labeling

Flow labeling had been taken into consideration when choosing the approach to capturing activity data and thus only a single activity was conducted in any given capture. Afterwards, the captures had been grouped by activity and filtered down to only contain TeamViewer flows identified by magic numbers or known TeamViewer IP address (only magic numbers could be used for direct UDP communication). UDP traffic differs from TCP in terms of identification in that the magic numbers are not placed at the beginning of the data, but rather in the 12^{th} and 13^{th} bytes. Once the binary classification is used to remove non-TeamViewer flows, each group of flows had been labeled based on the known activity conducted during its capture as Conference ("CO"), File Transfer ("FT") or Remote Control ("RC").

2.2.3 Activity features

As the activity dataset contains both TCP and UDP traffic, the original full feature set is used and expanded to include the transport protocol used. As UDP is added, it is also worth discussing the port numbers used in the activity dataset. As only TeamViewer traffic is included, the TCP destination port is not quite as influential as it was in binary classification (see 2.1.6). However, a similar issue arises with UDP port numbers.

Conferencing still uses a TeamViewer server as a proxy, communicating with it on UDP port 5938. The other two use cases, however, use their TCP connections to a server to negotiate direct UDP communication to each other. This direct communication does not use port 5938, instead it has been observed to use a wide variety of port numbers, ranging roughly between 30000–65535 (for both source and destination port number). The port number disparity distinguishes the conferencing UDP traffic from the others.

Chapter 3

Experiment realisation

Both kinds of experiments to be conducted seek to classify captured network flows. In the first, the classification is binary – either a flow contains TeamViewer traffic or it does not. In the second one, the provided dataset is known to contain TeamViewer traffic and a classification of the content of the traffic is the goal. The content can belong to one of three classes: conferencing, file transfer or remote control. In either case, a set of machine learning models are trained to perform the aforementioned classification using the prepared datasets. The tasks performed in this chapter have used the scikit-learn implementations of various machine learning models and associated functions.

3.1 Machine learning classifiers

Having prepared suitable datasets with a broad feature set to work with, machine learning classifiers were used to classify the traffic. The learning process of a model can either be supervised (i.e., the provided datasets are labeled) or not. For this thesis, the datasets had been labeled and as such, the learning was supervised. A selection of machine learning classifiers had been chosen for the task, consisting of a decision tree, a K-Nearest-Neighbors classifier and three ensemble approaches: RandomForest, AdaBoost and Gradient Tree Boosting.

The decision tree machine-learning method derives a set of rules from provided data features. Based on these rules, a branching set of questions is asked about the data, at the end of which the specified subset of data can be classified a certain way. While it remains a popular choice, especially because its methodology (i.e., asking a set of questions to reach a conclusion) is easy to understand and visualize, it may be susceptible to certain issues. Examples discussed in [43] include difficulty in generalising data (overfitting on the training data) or possible instability, where small variations in data lead to large differences in the generated trees. These issues can be addressed by adjusting the characteristics of the tree (e.g., maximum depth) or by using an ensemble approach to refine the outcome.

K-Nearest-Neighbors takes a different approach to classification in the sense that it does not create a model based on the input training data and instead stores it. When prompted to classify a sample, it then finds the most-alike known samples in the training data and reads their labels. The classification is then a matter of simple majority vote between the found labels.

The ensemble approaches are specific in that they utilize multiple simpler models, deriving the ultimate outcome from the results of the constituent models. The random forest approach employs multiple decision trees which are trained on different samples of the provided training data. The outcomes of the decision trees are then averaged. AdaBoost, while by default also using decision trees, takes a different approach. Instead of averaging the outcomes of the individual trees, it produces intermediary results, based on which new classification attempts are made, this time with adjusted weighting of misclassified elements. This way, difficult-to-classify problems are given more prominence. GradientBoosting is similar to AdaBoost, but rather then relying on weight adjustments in its learning, it attempts to minimize the results of a loss function used to quantify the error between the expected and the predicted values.

Table 3.1: Hyperparameters used for machine-learning classifiers

Classifier	**Hyperparameters used**
DecisionTree	random_state = 42, max_depth = 30, min_sample_split = 2
RandomForest	random_state = 42, n_estimators = 100, max_depth = None, min_sample_split = 2
K-Neighbors	n_neighbors = 5, leaf_size = 30
AdaBoost	estimator = DecisionTreeClassifier(random_state=42, max_depth=30), n_estimators = 50, learning_rate = 1
GradientBoosting	random_state = 42, n_estimators = 100, learning_rate = 1

Table 3.1 shows the classifier input parameters (they can also be referred to as hyperparameters) used during the experimental portion of this thesis. If

a hyperparemeter is not mentioned, the default value provided by its scikit-learn implementation was used. Decision trees have had their maximum depth reduced as a basic precaution to help reduce the risk of overfitting. Where applicable, a specific `random_state` value is supplied to ensure deterministic behavior.

3.2 Further feature vector exploration

The relative importances of features in classification can be observed by visualizing the importance values assigned to them by a capable machine learning classifier. In this case, the FET library function `plot_feature_importances`, based on the ExtraTreesClassifier, was used. Furthermore, the baseline feature vector can be reduced to explore the effect of various features on the classification process or to set up further experiments based around specific feature subsets.

A total of 57 features (58 including labels) was established in the binary datasets, with the activity dataset also containing the transport protocol used as a feature. In both cases, port numbers are included in the baseline feature vector. Two possible reductions of the feature vector have emerged in the course of writing this thesis: the removal of port-number features, and the removal of absolute-value features (e.g., overall packets sent) akin to that done in [30]. On one hand, the removal of absolute-value features removes the influence of unpredictable aspects of traffic from the data (e.g., the size of the file sent, or the length of a conference). On the other hand, relative features, such as the rate of packets sent over a unit of time, can be heavily influenced by the capture environment performance. As such, this approach is better suited to controlled capture environments as opposed to public networks.

Further, the feature vector can be reduced using the scikit-learn function RFECV [44] (Recursive Feature Elimination and Cross Validation). The function is given a set of data, an estimator and possible further optional parameters. The goal of the function is to recursively prune features based on their importances (those are calculated by the estimator trained on provided data). A minimum number of features may be specified, but a higher-than-minimum number may ultimately be kept, as the function runs a cross-validation loop and chooses the best-scoring outcome. For this thesis, scoring was based on the accuracy of the outcome (see section 4.1).

3.3 Training and testing the models

To prepare the training data for the actual classifier training, it is first normalized, i.e., the values are transformed in such a way as to fit within a specific number range while still retaining the differences between individual samples. In the course of the conducted experiments, a scikit-learn-implemented

`MinMaxScaler` had been employed to normalize the feature values to fit within the range between 0 and 1. This is especially important for the k-Nearest-Neighbors classifier as it removes the difference in scale between various features while retaining the relative differences between individual values of a feature. This helps to prevent the scale of feature values from influencing the neighbor-distance calculations. After the data is prepared as described, the classifiers can be trained and used to predict testing data labels.

This process is streamlined using the scikit-learn `Pipeline` class. An instance of this class allows the user to create a convenient wrapper around a sequence of steps that would normally be taken in the course of data preprocessing and the learning of a model. In this case, the pipeline is initialized with a scaler (for data normalization) and the classifier in question. The method `pipeline.fit(trainingData, trainingLabels)` is then called, which combines the fitting of the scaler, the consequent transformation of the data and the fitting (or learning) of the classifier. The pipeline then retains the fitted scaler and classifier, allowing the method `pipeline.predict(testData)` to transform the data using the scaler before passing it to the classifier for classification.

Chapter 4

Evaluation

4.1 Performance metrics

In the case of simple binary classification, four different outcomes can be found: true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Assuming two classes - 1) belongs to TeamViewer, and 2) does not belong to TeamViewer - we refer to the first one as the positive class and the second one as the negative class. A true positive is a sample correctly classified as belonging to TeamViewer, a false positive is a sample incorrectly classified as belonging to TeamViewer etc. When the classification contains more than two labels, it is treated as a set of such binary problems [45], one for each class (the other classes collectively play the role of the negative class). The results of each individual problem are then averaged to reach the final metric. In this chapter, the average is calculated as a mean of the contributing values and not weighted in any way.

The classification results can either be viewed as a matrix (called a confusion matrix) including all of the above or a single value quantifying a certain quality of the classification can be calculated to score the performance. The methods of scoring used here are accuracy, precision, recall and the F1 score as implemented in scikit-learn. The F1 score can also be referred to as balanced F-score, as it balances the contributions made by precision and recall.

Accuracy is concerned with the number of all correct predictions relative to the total number of predictions: $Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$.

Precision evaluates how likely it is for a positive prediction to be correct, with lesser precision rating meaning more false positive predictions: $Precision = \frac{TP}{TP+FP}$.

Recall seeks to quantify how likely it is for a positive sample to be correctly predicted: $Recall = \frac{TP}{TP+FN}$.

F1 score is a metric combining precision and recall as equally important: $F1 = 2 * \frac{precision*recall}{precision+recall}$.

Table 4.1 shows an example of a confusion matrix. Confusion matrices going forward will be formatted so that each row contains the overall number of true samples of a given class, whereas the column contains the overall number of samples predicted to be of that class. The matrix can be further expanded to include metrics (e.g., recall) calculated separately for each class.

Table 4.1: Confusion matrix example

	Predicted TeamViewer	Predicted Other
True TeamViewer	TP	FN
True Other	FP	TN

4.2 Binary classification

The models would first be trained on the prepared training dataset with equal proportions of TeamViewer and other traffic, then used to predict the labels of the testing set. The goal in this phase was to both evaluate the performance of the classifiers on the data as-is, and to experiment with feature vector reduction, namely by removing port-number features and utilizing the RFECV function to attempt to find a well-performing subset of the feature vector.

First, before any feature vector manipulation, the baseline feature vector had been explored. Using the FET function `plot_feature_importances`, an evaluation of the features was visualized. This evaluation was generated by an ExtraTreesClassifier trained on the prepared training set. Figure 4.1 shows the 15 most important features from the evaluation. While the destination port ranks reasonably high as a feature, its importance is comparable to others and in some cases it is overshadowed by them. Specifically, statistical features based on the ACK and PSH flag counts, and those based on packet lengths do very well.

Two separate experiments had been conducted to explore the impact port-based features have on classification – the first using the baseline feature vector and the second removing the destination and source port features. The difference between the two was found to be minimal with the exception of a minor increase in false positive predictions in the AdaBoost and DecisionTree classifier results. As such, only the results of the second experiment are shown in figure 4.2, accompanied by the original AdaBoost results from before the

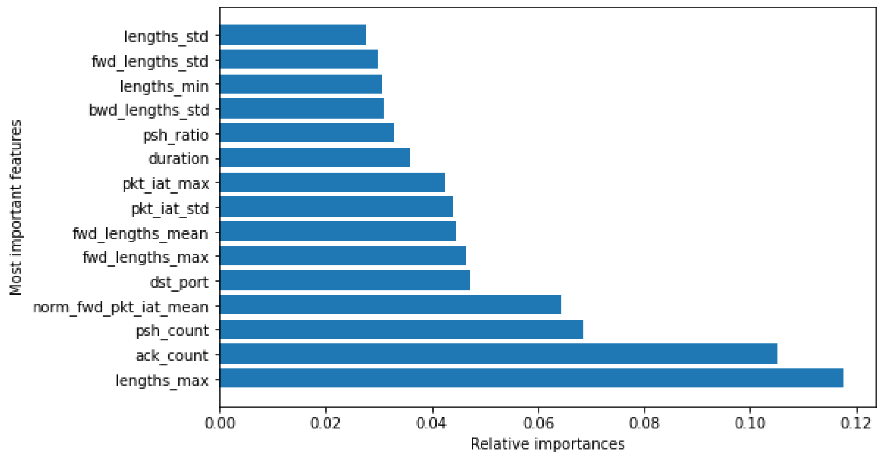


Figure 4.1: Backbone training dataset - most important features

removal for comparison. The performance is roughly even across the board, with the DecisionTree and AdaBoost classifiers suffering a decrease in precision. The best performers overall are the RandomForest and GradientBoosting classifiers.

Table 4.2: Results after port numbers have been removed

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 score**
AdaBoost (before)	0.99639	0.96386	0.99917	0.98960
AdaBoost (after)	0.99064	0.91080	0.99856	0.97373
DecisionTree	0.99090	0.91287	0.99887	0.97444
GradientBoosting	0.99827	0.98391	0.99805	0.99499
K-Neighbors	0.99652	0.96650	0.99774	0.98997
RandomForest	0.99939	0.99428	0.99928	0.99821

Given the performance of other features, port-number based features had been removed from the feature vector for the rest of the binary classification experiments. This was done to see how well the classifiers manage to predict the results relying more on varied statistical properties of captured traffic as opposed to static features such as port numbers. However, port-number based features are kept in the original feature vector of the datasets because they remain a relevant aspect of the traffic flows and can be used to differentiate TeamViewer flows from others.

While the differences between the various classifiers are quite slim, the best performance based on the F1 score was achieved by the RandomForest classifier (0.998). It was therefore selected as the classifier for further evaluation, specifically for an attempted reduction of the feature vector using the RFECV function. At this point, the feature vector had contained a grand total of 55 features after the removal of destination and source port. The RFECV function had reduced this number down to 34. The most important 15 according to the RandomForest classifier used are shown in figure 4.2.

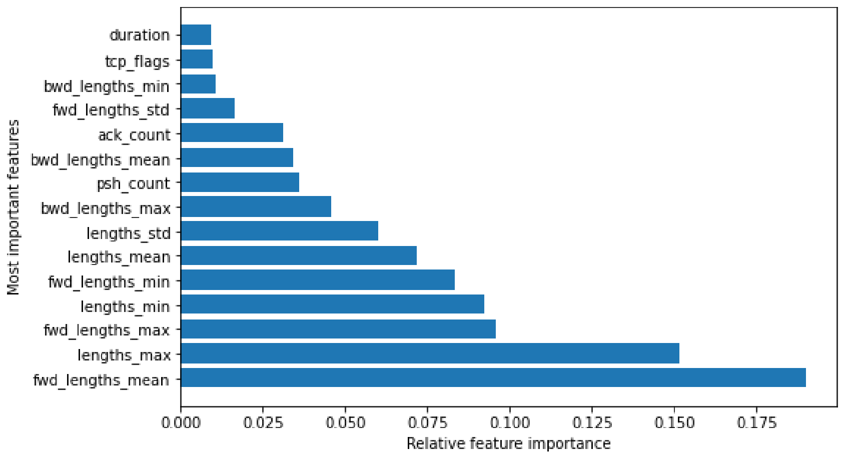


Figure 4.2: Feature importances of RandomForest after RFECV

Similar to the initial importance evaluation, features related to payload-length statistics and TCP-flag counts remain highly influential. Payload-length statistics help distinguish flows based on their beginnings using the lengths of the TCP/UDP payloads of the first 30 observed packets. Looking at the data in the training set, TeamViewer flows very commonly maintain a low maximum payload length (24 B) in these first 30 packets, while occasionally reaching as high as 1460 B.

On the other hand, other traffic is more likely to have a higher maximum payload length. A wide variety of lengths can be seen, the most common being 1460 B (likely the effect of the included TLS sample). This may point to a longer initial communication setup when using a TeamViewer server proxy in which shorter payloads are used.

Table 4.3: RandomForest results after RFECV reduction

	TV	OT
TV	9,738	7
OT	48	93,555
CP	0.99510	0.99993
CR	0.99928	0.99949

The ACK (acknowledgement) and PSH (push) TCP flags are used less in the observed initial packet payloads of TeamViewer traffic – ACK is shown to have a mean of roughly 4.4 occurences while PSH of 4.3. In other traffic, ACK and PSH display means of roughly 20.9 and 15.5, respectively. Table 4.3 shows the final results of the RandomForest classifier trained using this reduced feature set.

4.3 Activity classification

In the case of activity classification where only one set is available, the scikit-learn `train_test_split` function is used to create a training and testing subset (using a 60 % training to 40 % testing split). In order to retain the original relative representation of labels in these subsets, the function offers the `stratify` parameter. When supplied the training data labels, the function will stratify its sampling so as to maintain the original ratio of different labels. Experiments with feature selection in activity classification include three separate ideas – removing port-number features, removing absolute-value features from the feature vector (akin to [30]) and finally, utilizing the RFECV function to attempt to reduce the feature vector for the classifier found to be performing the best. Before discussing the results of the experiments, it has to be said that the experiments conducted in this section had used a limited size dataset that may not be fully representative of the traffic discussed. Consequently, the results presented are a proof-of-concept for classification of TeamViewer activities rather than a final solution.

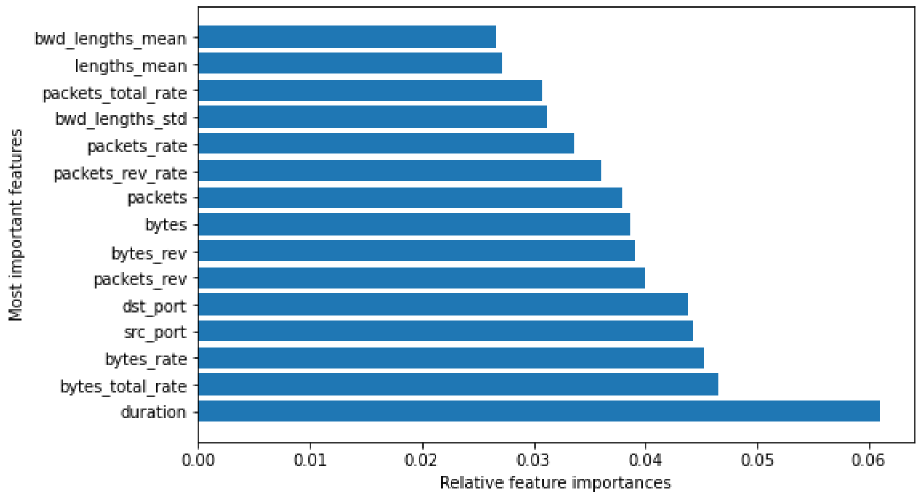


Figure 4.3: Activity dataset - most important features

Looking at the feature importances of the vector as-is (figure 4.3), the most influential feature is flow duration. While captured conferencing calls and remote control sessions were conducted in similar length, file transfer sessions, although varied, generally lasted a shorter time. It cannot be determined at the time of writing if such a difference also occurs in live traffic captured in a public network. Regardless, it merits exploring how well the traffic can be classified with absolute values such as duration removed from the feature vector (the approach of [30]).

Similarly to the binary classification experiments, the destination port number in the activity dataset is one of the most important features, but not overwhelmingly so. However, for the activity dataset, the source port number also ranks highly. The difference between TCP and UDP flows may help explain why – UDP flows uniformly use source port numbers over 30000, whereas TCP flows use numbers from a range roughly between 1300 and 5000. First, the influence of port-based features on the classification was investigated (originally discussed in section 2.2.3). The removal of port-based features does lead to a modest increase in remote control (RC) misclassifications. However, the overall results are not considerably worse, suggesting the feature vector is robust enough to perform well without having to rely on these features.

Table 4.4: DecisionTreeClassifier results - before (left) and after (right) port-features removal

	CO	FT	RC
CO	92	2	3
FT	1	92	16
RC	5	13	74
CP	0.93878	0.85981	0.79570
CR	0.94845	0.84404	0.80435

	CO	FT	RC
CO	94	2	1
FT	3	94	12
RC	8	17	67
CP	0.89524	0.83186	0.83750
CR	0.96907	0.86239	0.72826

The difference is shown in table 4.4 using two confusion matrices on the example of the most affected classifier – DecisionTree. Appended below the matrix are class-specific values for precision (CP) and recall (CR), corresponding to the class represented in the given column. In the core of the matrix, the three designated classes are represented: conferencing (CO), file transfer (FT) and remote control (RC).

Table 4.5: Overall results after removing port-based features

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 score**
AdaBoost	0.89933	0.89903	0.89789	0.89801
DecisionTree	0.85570	0.85487	0.85324	0.85220
GradientBoosting	0.89597	0.89680	0.89634	0.89656
K-Neighbors	0.79530	0.79327	0.79405	0.79126
RandomForest	0.90604	0.90561	0.90552	0.90534

Looking at the overall results of the classifiers (shown in figure 4.5), the RandomForest classifier had performed the best, with AdaBoost and GradientBoosting classifiers performing nearly as well with a marginally lower F1 score (0.905 versus 0.898 and 896, respectively). The DecisionTree classifier was more impacted by the loss of port-based features but still performed reasonably well, reaching an F1 score of 0.852. Finally, the K-Neighbors classifier performed the worst with an F1 score of 0.791.

Given these results, the RandomForest classifier had been chosen for further evaluation. The RFECV function discussed in section 3.2 has been used to reduce the feature vector and observe the effect on the classifier. The

Table 4.6: RandomForest results before (left) and after feature vector reduction

	CO	FT	RC
CO	96	1	0
FT	1	97	11
RC	3	12	77
CP	0.96000	0.88182	0.87500
CR	0.98969	0.88991	0.83696

	CO	FT	RC
CO	95	1	1
FT	1	94	14
RC	2	12	78
CP	0.96939	0.87850	0.83871
CR	0.97938	0.86239	0.84783

outcome was that 8 features would be removed. This included the counts and ratios of the FIN, RST and URG flags (i.e., `fin_count, fin_ratio` and so on) and the overall and forward normalized IAT standard deviation (i.e., `norm_pkt_iat_std` and `norm_fwd_pkt_iat_std`). The pruning of these features caused no major change to the produced results, as shown in table 4.6.

Lastly, given the somewhat arbitrary nature of the locally captured flows, an attempt was made to evaluate the performance of classifiers trained on a feature set from which absolute-value features had been removed (akin to [30]). This way, the classification is not influenced by the arbitrary choices of session length or file size.

Table 4.7: Overall results after removal of ports and absolute-value features

Classifier	**Accuracy**	**Precision**	**Recall**	**F1 score**
AdaBoost	0.81879	0.81868	0.81791	0.81818
DecisionTree	0.78523	0.78550	0.78355	0.78295
GradientBoosting	0.83893	0.84411	0.83512	0.83656
K-Neighbors	0.74497	0.73842	0.74686	0.74118
RandomForest	0.84899	0.85112	0.84637	0.84720

This removed features expressed the number of bytes or packets observed in any direction and the flow duration (i.e., `bytes, bytes_rev, packets, packets_ rev` and `duration`). The counts of observed specific TCP flags (e.g., `psh_count`) were kept, as they relate to the payloads of the first 30 packets rather than the entire flow. This feature vector reduction influenced the results more noticeably, but with the sole exception of the K-Neighbors classifier, never reduced the F1 score of any classifier below the 0.75 mark. The results are presented in table 4.7. As with previous experiments, the best performing classifier had been RandomForest with an F1 score of 0.847.

Table 4.8: RandomForest performance after absolute-value feature removal

	CO	FT	RC
CO	95	1	1
FT	1	93	15
RC	2	25	65
CP	0.96939	0.78151	0.80247
CR	0.97938	0.85321	0.70652

The performance remains comparable with previous experiments with the exception of an increase in remote control (RC) flows being misclassified as file transfer (FT). This can be seen in the decrease of FT precision and RC recall, shown in table 4.8.

Conclusion

TeamViewer is one of the more prevalent and commonly used remote access applications, allowing users to manage remote devices or to interact with colleagues and customers. Part of the appeal of the application is its ease of use, making it a viable option for most users. However, this ease of use, coupled with the level of access granted by the application, poses a severe threat to security as the application can be exploited in various ways. These could be social engineering (such as technical support scams) or the abuse of the application by malware, allowing the attacker remote access to a victim's system. Consequently, the presence of TeamViewer traffic in a network infrastructure can be an indicator of a security threat – either a company policy violation, or a malicious attack. Therefore, the main motivation of this thesis was to analyse the communication of TeamViewer on the network level (considering packets and IP flows) so as to design and develop a detection algorithm.

This thesis had dealt with the traffic recognition of TeamViewer using machine learning models that were comprehensively evaluated in the designed experiments. As no publicized datasets of TeamViewer traffic could be found, one of the primary contributions of this thesis is the creation of datasets of traffic samples for the training and testing of machine learning algorithms. The chief goal of this effort was to invent a feasible detection algorithm to reliably recognize TeamViewer connections among other traffic. Additionally, the experiments continued on to distinguish between different contents of encrypted traffic, i.e., remote control, file transfer and conferencing. The results of the experiments are promising, with the detection of TeamViewer traffic reaching 99.9 % accuracy and distinguishing between activities reaching at least 84.9 %. The best classification models that were found are based on DecisionTrees/Random Forests and as such can be easily deployed to analyse high-speed network traffic.

Future work

Future work could include the expansion of the binary datasets, as they primarily focused on TCP communication, with UDP traffic. The scope of the activity dataset could be further expanded. Finally, the Wireshark dissector could help further explore the workings of the application. Unfortunately, further analysis and development of this tool were out of the scope of this thesis and remain a topic for future work.

Bibliography

[1] *LogMeIn.* [online] [visited 2021-04-26]. Available from: https://www.logmein.com/

[2] *The Fast Remote Desktop Application - AnyDesk.* [online] [visited 2021-04-26]. Available from: https://anydesk.com/en

[3] *Splashtop: Remote Access & Remote Support Software* [online]. Apr 2021, [visited 2021-04-30]. Available from: https://www.splashtop.com/

[4] TeamViewer AG. *TeamViewer 15.16.8* [software]. Mar 2021, [visited 2021-04-10]. Available from: https://www.teamviewer.com/en/download/windows/

[5] *TeamViewer Benefits* [online]. Oct 2005, [visited 2021-04-10]. Available from: https://web.archive.org/web/20051001021009/http://teamviewer.com/benefits.aspx

[6] *Growth* [online]. Apr 2021, [visited 2021-04-10]. Available from: https://www.teamviewer.com/en/company/growth/

[7] *Why pay for TeamViewer* [online]. [visited 2021-04-10]. Available from: https://www.teamviewer.com/en/why-pay/

[8] *Supported operating systems* [online]. [visited 2021-04-10]. Available from: https://community.teamviewer.com/English/kb/articles/24141-which-operating-systems-are-supported

[9] *Remote Access Trojan (RAT)* [online]. Mar 2021, [visited 2021-04-26]. Available from: https://blog.malwarebytes.com/threats/remote-access-trojan-rat/

[10] PEARSON, D. Security Analysis: TeamViewer. In: *Awake Security Blog* [online]. Aug 2019, [visited 2021-04-10]. Available from: https://awakesecurity.com/blog/analyzing-teamviewer/

[11] ZSEBY, T.; CLAISE, B.; et al. *Requirements for IP Flow Information Export (IPFIX)* [online]. RFC 3917, Oct. 2004, doi:10.17487/RFC3917. Available from: `https://rfc-editor.org/rfc/rfc3917.txt`

[12] INACIO, C. M.; TRAMMELL, B. *YAF: Yet Another Flowmeter.* [online]. Aug 2010. [visited 2021-04-20]. Available from: `https://resources.sei.cmu.edu/asset_files/WhitePaper/2010_019_001_57050.pdf`

[13] AITKEN, P.; CLAISE, B.; et al. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information.* [online] [visited 2021-04-30]. RFC 7011, Sept. 2013, doi:10.17487/RFC7011. Available from: `https://rfc-editor.org/rfc/rfc7011.txt`

[14] TRAMMELL, B.; BOSCHI, E. *Bidirectional Flow Export Using IP Flow Information Export (IPFIX)* [online] [visited 2021-04-30]. RFC 5103, Jan. 2008, doi:10.17487/RFC5103. Available from: `https://rfc-editor.org/rfc/rfc5103.txt`

[15] *Cisco/Joy: A package for capturing and analyzing network flow data and intraflow data, for network research, forensics and security monitoring* [online] [visited 2021-04-29]. Available from: `https://developer.cisco.com/codeexchange/github/repo/cisco/joy/`

[16] HABIBI LASHKARI, A.; DRAPER GIL, G.; et al. *CICFlowmeter-V4.0* [online]. 2020, [visited 2021-04-30]. Available from: `https://github.com/CanadianInstituteForCybersecurity/CICFlowMeter`

[17] *CESNET/ipfixprobe* [online]. 2021, [visited 2021-04-30]. Available from: `https://github.com/CESNET/ipfixprobe`

[18] *CESNET/Nemea* [online]. 2021, [visited 2021-04-30]. Available from: `https://github.com/CESNET/NEMEA#nemea-system`

[19] *What is Wine?* [online]. 2021, [visited 2021-04-26]. Available from: `https://www.winehq.org/`

[20] *The Wait is Over: Presenting TeamViewer's Native Linux Client.* [online]. Aug 2018, [visited 2021-04-10]. Available from: `https://community.teamviewer.com/English/discussion/36765/the-wait-is-over-presenting-teamviewer-s-native-linux-client`

[21] *Security Overview* [online]. [visited 2021-04-10]. Available from: `https://www.teamviewer.com/en/trust-center/security/`

[22] *Can TeamViewer be used within a local network (LAN) only?* [online]. [visited 2021-04-10]. Available from: `https:`

`//community.teamviewer.com/English/kb/articles/4618-can-teamviewer-be-used-within-a-local-network-lan-only`

[23] FORD, B.; KEGEL, D.; et al. *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)* [online] [visited 2021-04-30]. RFC 5128, Mar. 2008, doi:10.17487/RFC5128. Available from: `https://rfc-editor.org/rfc/rfc5128.txt`

[24] THOMAS, B. *TeamViewer authentication protocol (part 1 of 3)* [online]. Jan 2013, [visited 2021-04-10]. Available from: `https://web.archive.org/web/20190411063636/https://www.optiv.com/blog/teamviewer-authentication-protocol-part-1-of-3`

[25] *Which ports are used by TeamViewer?* [online]. [visited 2021-04-10]. Available from: `https://community.teamviewer.com/English/kb/articles/4139-which-ports-are-used-by-teamviewer`

[26] *TeamViewer Trust Center - FAQs* [online]. [visited 2021-04-10]. Available from: `https://www.teamviewer.com/en/trust-center/faq/`

[27] THOMAS, B. *TeamViewer authentication protocol (part 1 of 3)* [online]. Feb 2014, [visited 2021-04-10]. Available from: `http://bennysecurity.blogspot.com/2014/02/rawtech-blog-teamviewer-authentication.html`

[28] Wireshark Foundation. *Wireshark 3.0.5* [software]. [visited 2021-05-10]. Available from: `https://www.wireshark.org/#download`

[29] Chapter 9. Packet Dissection. In: *Wireshark Developer's Guide* [online]. Version 3.5.0. [visited 2021-04-26]. Available from: `https://www.wireshark.org/docs/wsdg_html_chunked/ChapterDissection.html`

[30] ALTSCHAFFEL, R.; CLAUSING, R.; et al. Statistical Pattern Recognition Based Content Analysis on Encrypted Network: Traffic for the TeamViewer Application. In: *2013 Seventh International Conference on IT Security Incident Management and IT Forensics* [online]. Institute of Electrical and Electronics Engineers, 2013. [visited 2021-04-03]. doi: 10.1109/IMF.2013.19. Available from: `https://ieeexplore.ieee.org/abstract/document/6568559`

[31] *Weka 3 - Data Mining with Open Source Machine Learning Software in Java.* [online] [visited 2021-04-25]. Available from: `https://www.cs.waikato.ac.nz/ml/weka/`

[32] CESNET z.s.p.o. *CESNET2 network* [online]. Aug 2015, [visited 2021-04-30]. Available from: `https://www.cesnet.cz/services/ip-connectivity-ip/cesnet2-network/?lang=en`

[33] *Numpy* [online]. 2021, [visited 2021-04-30]. Available from: `https://numpy.org/`

[34] *Pandas - Python Data Analysis Library.* [online] Apr 2021, [visited 2021-04-30]. Available from: `https://pandas.pydata.org/`

[35] *Scikit-Learn: Machine Learning in Python.* [online] Apr 2021, [visited 2021-04-24]. Available from: `https://scikit-learn.org/stable/`

[36] UHŘÍČEK, D. *Feature Exploration Toolkit* [online]. [visited 2021-04-10]. Available from: `https://gitlab.fit.cvut.cz/uhricdan/fet`

[37] UHŘÍČEK, D. *Detection of IoT Malware in Computer Networks.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

[38] *CESNET/Nemea-Modules/logger* [online]. Dec 2020 [visited 2021-04-25]. Available from: `https://github.com/CESNET/Nemea-Modules/tree/master/logger`

[39] TROPKOVÁ, Z. *Klasifikace akcí přenášených skrz šifrované TLS spojení.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

[40] SMEJKAL, R. *Klasifikace komunikace SSH protokolu.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

[41] HULÁK, M. *Flow-Based Traffic and Devices Classification in Computer Networks.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020. Available from: `https://dspace.cvut.cz/handle/10467/90394`

[42] *Pickle - Python Object Serialization.* [online] Apr 2021, [visited 2021-04-25]. Available from: `https://docs.python.org/3/library/pickle.html`

[43] 1.10. Decision Trees. In: *Scikit-learn 0.24.2 documentation* [online]. 2020, [visited 2021-04-29]. Available from: `https://scikit-learn.org/stable/modules/tree.html#tree`

[44] 1.13. Feature selection. In: *Scikit-learn 0.24.2 documentation* [online]. 2020, [visited 2021-04-30]. Available from: `https://scikit-learn.org/stable/modules/feature_selection.html#rfe`

[45] 3.3. Metrics and scoring: quantifying the quality of predictions. [visited 2021-04-30]. Available from: `https://scikit-learn.org/stable/modules/model_evaluation.html#from-binary-to-multiclass-and-multilabel`

Appendix A

Acronyms

ACK	Acknowledgement
AES	Advanced Encryption Standard
BMP	Bitmap Image File
CESNET	Czech Education and Scientific NETwork
CO	Conferencing
CP	Class precision
CR	Class recall
CSV	Comma-separated values
DLL	Dynamic-link library
DNS	Domain Name System
FET	Feature Exploration Toolkit
FIN	Finished
FT	File transfer
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IAT	Inter-arrival time
IDPContent	Initial Data Packets Content
IMAP	Internet Message Access Protocol
LAN	Local Area Network

NAT	Network Address Translation
NEMEA	Network Measurements Analysis
OT	Other
PCAP	Packet Capture
PCAPNG	Packet Capture Next Generation
POP3	Post Office Protocol 3
PSH	Push
PSTATS	Packet Statistics
PTR	Pointer
RAT	Remote Access Trojan
RC	Remote Control
RFC	Request For Comments
RFECV	Recursive Feature Elimination Cross Validation
RSA	Rivest, Shamir, Adleman
RST	Reset
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SPLT	Sequence of Packet Lengths and Times
SYN	Synchronisation
TCP	Transmission Control Protocol
TID	TeamViewer ID
TLS	Transport Layer Security
TV	TeamViewer
UDP	User Datagram Protocol
URG	Urgent
YAF	Yet Another Flowmeter

Appendix **B**

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
datasets ................................... the directory of datasets
notebooks ......................... the directory of Jupyter notebooks
  ipynb ........................ the directory of the original notebooks
  pdf ............................. the directory of their pdf versions
  shared.py ...................... Python module of shared functions
BP-2021-Klatovsky-Tomas.pdf .......... the thesis text in PDF format
thesisSource.zip ................ the LaTeX source codes of the thesis
```