



**UNIVERSITY
OF TURKU**

TLS - JA3 Client Fingerprinting Comparisons for Applications running on Windows, Linux and Android Operating Systems

Cyber Security
Master's Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Ayush Gupta

Supervisors:
Antti Hakkala (University of Turku)
Jenny Heino (Forcepoint)

June 2021

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin Originality Check service.

Master of Science in Technology Thesis

TLS - JA3 Client Fingerprinting Comparisons for Applications running on Windows, Linux and Android Operating Systems

Department of Computing, Faculty of Technology

University of Turku

Subject: Cyber Security

Programme: Master's Degree Programme in Information and Communication Technology

Author: Ayush Gupta

Title: Mr.

Number of pages: xx pages, yy appendix pages

Date: March 2022

Abstract

TLS fingerprinting has evolved due to the rising concerns of cybersecurity. The need for a mechanism that can detect the malicious activity in the encrypted traffic during communication between server and client without decryption has led to the development of the JA3 methodology. Various studies in the past endeavoured to analyse network traffic using various statistical, mathematical, and intelligent computing techniques. The uniqueness of this research is that it attempts to examine the uniqueness of the fingerprints produced by a client on Android, Windows and Linux operating systems and conducts cross-comparisons of their characteristics.

The questions that are answered in the study are –

- (a) What is the probable list of the possible fingerprints generated by a particular client?
- (b) Do different clients generate similar fingerprints on same operating system?
- (c) Whether different clients generate similar fingerprints on different operating systems?
- (d) For a given client, does the Ja3 generate same or different fingerprint on different operating systems.

Different methodologies have been deployed to extract and analyse the data produced on the selected operating systems. This study analyses on the TLS Fingerprinting of Client Hello packet generated by different Clients from the selected OS.

Analysis of the results reveal that JA3 obtained from various applications for different Operating Systems exhibit differing features. While some application produced unique fingerprints while some produced common hashes for a particular operating system. Web Browsers like Chrome, Mozilla Firefox and Edge produced large number of hashes per OS. Some application like Teams and Skype were found to have common hashes. For some application like Telegram, no hashes were produced.

When the JA3 is applied in Firewall to be used with End-Point detection, unique JA3 fingerprints can be used to identify the application, while the common or similar hashes can be used to predict the application type.

Keywords: TLS, Fingerprinting, JA3, Operating System, Encryption

Table of contents

1	Introduction	7
1.1	Fingerprinting and Cyber Security	7
1.2	Research Problem	8
1.3	Research Methods and Data Collection	9
1.4	Structure of the Thesis	10
2	Concept and Background	12
2.1	Network and Network Traffic	12
2.2	History of SSL	12
2.3	Evolution of TLS	14
2.4	TLS Protocol	15
2.4.1	Working of TLS	16
2.4.2	Traffic features of TLS Protocol	20
2.5	Fingerprinting	20
2.6	Types of fingerprinting	21
2.6.1	Network based HTTPS Client fingerprinting	21
2.6.2	Operating System fingerprinting	21
2.6.3	Browser fingerprinting	22
2.6.4	Markov Chain Fingerprinting	22
2.6.5	JA3/JA3S fingerprinting	23
2.7	TLS fingerprinting	23
2.8	JA3	24
2.8.1	Benefits of JA3	27
2.8.2	Limitations of JA3	27
2.9	Attacks and Vulnerabilities of SSL/TLS	27
2.9.1	Downgrade Attacks	27
2.9.2	Cross-protocol Attack	29
2.9.3	Bleichenbacher Attack	30
2.9.4	Drop ChangeCipherSpec Attack	31
2.9.5	Timing attack	31
2.9.6	Other Attacks	32
3	Review of Literature on JA3 Fingerprinting	35

3.1	Flow Traffic Analysis Methods	35
3.2	Fingerprinting Robustness Evaluation Approaches	36
3.3	Research Gap	38
3.4	Research Questions	38
4	Data and Methods	39
4.1	Schema of the study	39
4.2	Data Generation on Operating Systems	40
4.2.1	Android	40
4.2.2	Automation in Android web browser	42
4.3	Windows Traffic Analysis	43
4.4	Linux Traffic Analysis	48
4.5	Methods	49
4.5.1	Extracting JA3 from the Capture packets	49
4.5.2	Getting all Unique JA3 from the datasets	52
5	Results and Inferences	54
5.1	Applications in consideration	54
5.2	Results from Android	54
5.3	Results from Linux	55
5.4	Results from Windows	55
5.5	Application Wise Comparison on Android, Linux and Windows	56
5.6	Inferences	62
6	Conclusion	66
6.1	Challenges and Limitations	66
6.2	Scope for further research	69
	References	70
	Appendices	74
	Appendix 1 Linux	74
	Appendix 2 Windows	75
	Appendix 3 Android	77

Appendix 4 All Operating Systems	79
Appendix 5 Android Comparision with JA3ER	82
Appendix 6 Linux Comparison with JA3ER	85
Appendix 7 Windows Comparison with JA3ER	88

1 Introduction

Since its introduction, the Internet has taken over the world by storm. Now the Internet has become everyone's everyday necessary commodity. Everything is connected to internet now and one cannot imagine his/her life without it. The researchers have realized the need of security and encryption of data required for internet to protect users from attackers. Various security protocols were designed to resolve this issue.

Secure Socket Layer(SSL) Protocol[1] was developed to encrypt and secure data so that sensitive information can be exchanged between server and client. SSL adds a layer of security over the other protocols. However, the SSL suffered from statistical biases leading to increase in brute force attacks and leakage of sensitive cipher material. In view of the shortcomings and vulnerabilities of the SSL protocol, SSL protocol was replaced by Transport Layer Security(TLS) protocol. TLS aims to provide privacy and data integrity between two more communicating parties. TLS fingerprinting helps use to identify the malicious communication without decryption and also no exploration of IP addresses[2]. TLS is agile with multifaceted quality, high latency time covering a wider range of network applications and providing critical services such as confidentiality and integrity. Ja3 leverages the attributes of TLS to uniquely identify client applications across multiple sessions. Sale force engineers provided a mechanism using Ja3 to systematically fingerprinting in order to achieve high degree of accuracy[3].

1.1 Fingerprinting and Cyber Security

Fingerprinting is a method of gathering unique information about an object. In the field of Cyber Security, fingerprinting is called as "Footprinting". Footprinting comprises of various methods that can be used to classify and identify various networks, users, IP Addresses, hardware, software etc. The information generated from these methods is obtained carefully so that it is unique in nature and can be appropriately used to analyse objects. Footprinting methodologies can be based on Network HTTPS Clients, Operating Systems, Browsers etc. Footprinting techniques are classified on the basis of analysis and data collection – (a) *Active* – involving direct interaction with the target. The analysis is done after a query is sent and response is obtained. (b) *Passive* – involving no direct interaction with the target. The traffic is captured and analysed without sending any query.

In this thesis, active fingerprinting method will be used to fingerprint TLS packets so that unique clients can be identified. JA3, a TLS fingerprinting based method has been used to generate the fingerprints from TLS Client Hello Packet.

Need for Fingerprinting

Fingerprinting obtains the unique characters or features from the target object. Fingerprinting, as a measure of security, authenticates users, provides several advantages. Organizations can address the vulnerabilities of their networks, devices and operating systems through fingerprinting. The fingerprinting enables the attackers to gather information on types and version of operating system, SNMP information, domain names, network blocks, VPN points etc. They launch custom packets to carry out attacks that are largely customised to yield a damage or undue advantage to the target system. As a preventive measure, the organisation can perform active or passive fingerprinting to replicate the actions of attackers, which can provide clues on the vulnerabilities. Thus, the critical points where the sensitive information can be leaked can be strictly monitored and leakage of data out of the organization's environment can be avoided.

Fingerprinting essentially reveals the various features of the network like open or closed ports, network configuration and distinct devices attached to the network. Fingerprinting allows us to evaluate the level of privacy that can be placed on the vulnerable target objects so that their access can be controlled in the event of attack. Fingerprinting can also classify resources which enables us to identify and block the malicious requests originating and traveling from inside or outside of the network. Since there are multiple fingerprinting techniques, different ones can be deployed to protect the critical resources of the organisation.

1.2 Research Problem

Various fingerprinting techniques have developed in the field of cybersecurity in the recent times. Of these, the TLS fingerprinting is closest to fingerprinting at client side. TLS connection initiates Client Hello packet that calls the server end and the capabilities of the client, presented in a preference order. On receipt of similar response packet the comparison is carried out for the packets and optimal cipher suites, compression algorithms etc. can be optimally determined. Lee Brotherston in 2015 has presented the mechanism of TLS fingerprinting[4]. As a step further, Security Engineers John B. Althouse, Jeff Atkinson, and Josh Atkins, security engineers developed an open source method called as "JA3" which fingerprints the TLS negotiation between client and server. It is argued that JA3 and JA3S combined fingerprinting can assist in

producing a higher fidelity identification of the encrypted communication between a specific client and its server[3]. JA3 methodology facilitates the identification of the application initiating the TLS session through a comparison of JA3 fingerprints (databases). The unique hash from the Hello Packet enables to fingerprint a particular client with certain acceptable accuracy.

The researches on TLS fingerprinting are mainly focussed on matching hashes from the repository and application of statistical procedures to determine the accuracy of hashes[5]–[8]. However, there is an ongoing debate and research and on the reliability and usability of TLS fingerprinting. Detection of malicious activity in encrypted traffic requires analysis of JA3 hashes for fingerprints generated for client applications. It can be argued that different operating systems have different internal structures. Their responses vary across time periods on identical data sets

The server TLS response is likely to be identical in majority of the cases for different applications on different operating systems and also for same application with different versions. This implies that JA3 analysis of servers is of little use. However, the client software generates TLS packets with unique contents.

The motivation of the thesis is to examine the uniqueness of fingerprints produced by a client on different operating systems and a cross comparison among them.

1.3 Research Methods and Data Collection

The operating systems Android, Linux and Windows will be used in the analysis. The applications that are available on all the platforms are being analysed in the study.

In case of Windows, firewall has been disabled and a software utility named as “Simplewall” was used. Simplewall allows us to configure the Windows Filtering Platform used to handle network activity on computers running Windows[9]. Using Simplewall, irrelevant communication can be blocked and JA3 can be generated only for the targeted application.

For Linux, it is easier to generate the JA3 information because there are no background processes running in the background to perform TLS handshake[2].

For Android, an application “Afwall plus” was used to block unnecessary background applications and then JA3 information was captured. Afwall is an advanced iptables editor for

Android and provides fine-grained control over which Android apps are allowed to access the network[10].

Some of the applications that were suspected to generate more than one Ja3 fingerprints were ran for a span of 24 hours and were used frequently so that all the unique fingerprints can be captured.

1.4 Structure of the Thesis

The thesis is divided into 5 chapters. First three chapters gives the overall idea about the thesis. They also explain the idea behind the research and how the research was made, topics focused and conceptual background.

The later part of the thesis tries to explain how the analysis was conducted, dataset was generated and gathered. The research questions are tried to answered in the last two chapters of the thesis.

Chapter 1 presents the motivation for the research and research problem in depth. It gives the brief idea about Fingerprinting and introduction of Ja3. It talks about the general idea of the thesis that is being presented.

Chapter 2 gives the conceptual background of the thesis. It describes about the Network analysis, origination of SSL and how it led to the development of TLS. It deeply explains the working of TLS protocol by demonstrating the TLS Handshake procedure. It explains about the Fingerprinting and types of Fingerprinting in usage. It also introduces the concept of TLS Fingerprinting and gives an in-depth knowledge of the working principle behind JA3. It also discusses the vulnerabilities and attacks that are prevalent against TLS protocol.

Chapter 3 presents the Literature review. It talks about the similar studies and researches that have been done related to the topic. It also talks about how different authors have used Ja3 in TLS Fingerprinting. Towards the end, the research gap has been identified.

Chapter 4 is all about the setting up the working environment and preparing resources for gathering of data. It presents how the different operating systems have been prepared to run specific applications so that JA3 can be generated from it. It also presents some other tools or resources that have been used in the thesis.

Chapter 5 presents the analysis of the findings that have been observed.

Chapter 6 especially talks about the conclusions that have been made from the observations. It discusses all the challenges and problems that were encountered during the research and analysis. It also gives the way for the future research, explains some shortcomings and presents what has been left from the overall research.

2 Concept and Background

The Internet is made up of different devices which are always communicating with each other using packets. Packet is a fundamental unit for a protocol. Different protocols have different fundamental units, or protocol data units (PDUs). Network Packet is described as a fundamental unit of communication for network. A packet contains different types of information like sender's address, receiver's address, type of packet, etc. Packet on Internet carry the data in the form of protocols used - Transmission Control Protocol/Internet Protocol (TCP/IP).

2.1 Network and Network Traffic

Network packets makeup the Network traffic. Network traffic is analogous to Vehicle traffic; there is a channel or medium where the Packets (like vehicles) travel.

Network analysis is also known as Network Traffic Analysis, is a method for analysing networks. It includes monitoring the activities occurring on the network, collecting relevant network information, and reporting it in a real-time scenario. It can be used for various purposes like studying the behaviour of the network, fault detection and correction, identifying malicious activities, etc. It also helps in the identification of the protocols that can be vulnerable or might cause some problems in the future that will affect the normal working of the network. It is extensively used to troubleshoot the slowdown of the network. Traffic identification is essential to classify legitimate activity from malicious one. It can be useless or slightly useful on a personal level, but it is extremely useful on the enterprise level, where there are thousands of machines connected with different servers and network devices, producing a vast amount of traffic. So, this serves as the most suitable choice for most attackers where they can hide their attacks in the network crowd.

2.2 History of SSL

In the year 1994, when the Internet was becoming popular, there arose a need for secure communication. At that time, the Internet was still in its early development stage. If someone is communicating with someone else, anyone can eavesdrop on the communication easily in a blink of an eye. So, this was not ideal and concerned the researchers. So, this had to be fixed at any cost. Netscape Communication was dominating the Internet world in 1994, and their Netscape Navigator, the first browser ever made, controlled about 80% share of the overall market. But this browser had a problem, and attackers can easily "listen" to any communication

over the Internet. This was a severe issue that was eroding the confidence of the general public towards the usage of Internet.

The Internet at that time consist of computers connected with cables, and every computer has the ability to hear every communication being taken place on the Internet. Even though the messaged were packed and addressed to be received by certain individuals, but it doesn't deter hackers from intercepting the messages. At that time, a person with little knowledge about the technology can download the software that can allow him/her to intercept any message sent/received on the whole Internet. The messages were passed through Hubs (now called as routers) which are further connected to different Hubs, and the message travels to reach its final destination. So, any intermediate node is able to read this information without telling anyone.

Netscape recognized the importance of having a secret message and private communication; otherwise, the trust of the general public will be broken. They knew that people would hesitate to make online eCommerce communication if their credit cards numbers are stolen easily by eavesdroppers. So, nobody will buy anything from the Internet, and faith in this new technology will be lost.

Inspired by the movie "A Christmas Story (1983)" Netscape decided to implement their own internet version of "Secret decoder ring". And this "Secret decoder ring" was implemented on individual level. So now everyone was able to send messages through the Internet so that the person intended to receive it will be able to read it (decode).

This "Secret decoder ring" was called as "Secure Socket Layer" by the Netscape.

Netscape implemented a system of security keys which will be present on both sender and receiver. Two keys were introduced, called Private and Public Key. Two or more computers on the Internet don't communicate directly, and they use Web Browsers like Chrome, Safari, Mozilla, etc. The Web Browsers communicate with Web Servers and know how to use Public and Private Keys to encrypt and decrypt messages.

They perform a "Handshake" or "SSL Handshake" to be precise, where they exchange Security Keys so that encryption and decryption can take place at both ends.

But the problem didn't end here. Netscape soon realized that the people participating on the Internet couldn't be trusted solely if they have keys, it might be possible that a user has leaked his/her keys by chance to an attacker. So, Netscape further developed a 3rd party on which sender and receiver can trust. This trusted 3rd party can vouch for another person (sender and receiver). This led to the development of an SSL Certificate[11].

The SSL first introduced by Netscape in 1994 was named SSL v1.0, but it was never made public because it had “trust” issues. Further, the first official release of SSL as SSL v2.0 was released in 1995. In November 1996, v3.0 was released as indicated in Figure 1.

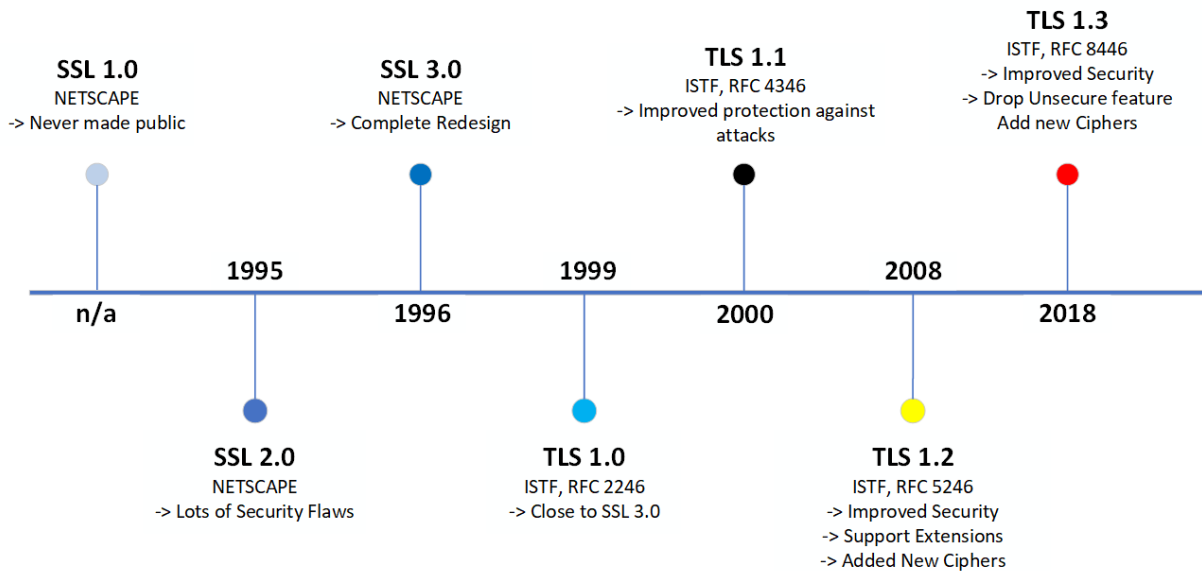


Figure 1 TLS/SSL Timeline

The above diagrammatic presentation has been derived from [12], [13].

2.3 Evolution of TLS

Secure Socket Layer(SSL) has evolved as a combination of protocols[14] that are meant to provide reliable services on the communication networks[15]. SSL has been upgraded many times from viz. SSLv1.0 to SSLv3.1, when SSLv3.1 is eventually known as TLSv1.0.

Netscape Communications developed the SSL 1.0 in middle 1994. The SSL v1 suffered from various security flaws and was not publicised. SSL 2.0 as a usable version of SSL was released in 1995. Because of the vulnerabilities another version of SSL – SSL3.0 was released. SSL 2.0 and SSL 3.0 were dropped in the year 2011 and 2014, respectively, because they had major flaws and drawbacks from a security point of view which led to the development of TLS[16].

The most prominent security flaws of SSL were the decrypting of RSA with obsolete and weakened encryption (drown) for v2 and proneness to poodle attacks in v3. These vulnerabilities limited the application of SSL v2 and v3 are both useless.

TLS 1.0 evolved as a refurbished revision of SSLv3.0 that compromises the detailed specification of a pseudo-random function utilized for key derivation. After the release of SSLv3.0, experts have identified a host of vulnerabilities, rendering the risk of data

compromises at various levels, especially in payment systems[17]. The security issues of SSL, TLS1.0, and TLS1.1 are critical and major websites, including Google, have stopped supporting these versions[18]. The TLS ecosystem has undergone tremendous changes since 2012, and the use of TLS1.2 and TLS1.3 has increased significantly, with 90% connections driven by forward-secret cipher suites[19]. It is seen that web browsers are adopting new and new cryptographic algorithms and evolving versions of the protocol. However, though they increasingly rely on TLS, the applications are still using insecure and obsolete protocol options according to Anderson and McGrew(a)[20].

In the past, tremendous attention is paid to the security of the protocol, but there is always a danger of unknown attacks as argued by Curguz[21]. Much emphasis is laid on the encryption of the data in network communication to prevent unauthorized analysis and capture external observers' data.. Latovicka *et al.* [22] have examined the various traffic patterns of the TLS1.3 protocol. Using an ML on the TSL handshake parameters, authors attempt to identify the client machine's operating system and compare their findings vis-vis other popular identification techniques. They have tested the proposed method in a large wireless network to get the proper identification of the client device connected to the network. It is also argued that data acquisition is becoming increasingly complex and the data flow needs to be improved with information from application protocols that continue to change and modify data sector specifications from previous versions.

The traditional approach to TLS fingerprinting relies on a clear text data creating a fingerprint string that helps to identify the process before exchange. Authors have termed these techniques as 'library', wherein fingerprints can be mapped to a large number of unique processes. Anderson and McGrew(b) [8] has proposed a method of generalized fingerprinting by the creation of a fingerprint knowledge base that names the most probable process name using weighted naïve Bayes classifiers.

Razaghpanah *et al.* [23] have shown that default TLS libraries are in use by most applications, making them vulnerable to attacks given outdated Android OS versions. For mobile device identification, the approach suggested by researchers is device hardware-based fingerprinting.

2.4 TLS Protocol

Every application running on any architecture or on any system uses TLS while communicating through a server on the Internet. TLS stands for Transport Layer Security. TLS is an update for

SSL v3 and was released in 1999. TLS is not different from SSL, it can be thought of as a more refined and secure version of SSL. TLS is used to encrypt data sent and received across the Internet. TLS provides a cryptographic mechanism so that the exchange of sensitive information can be made possible avoiding the problem of eavesdropping. TLS is similar to SSL, which was its predecessor, it was frequently used, but it had many flaws which paved the way for the development of TLS[23]. Just like SSL, TLS had also undergone changes in its work when it was first released in 1999. The recent TLS version is TLSv1.3 which is used to date.

2.4.1 Working of TLS

The TLS communication takes place in a similar manner as SSL, but some more things are added. The TLS communication begins with a Handshake[12], they exchange security information that includes ciphers, encryption protocols etc., to determine connection properties. The flowchart of TLS adapted from [24] is given in Figure 2.

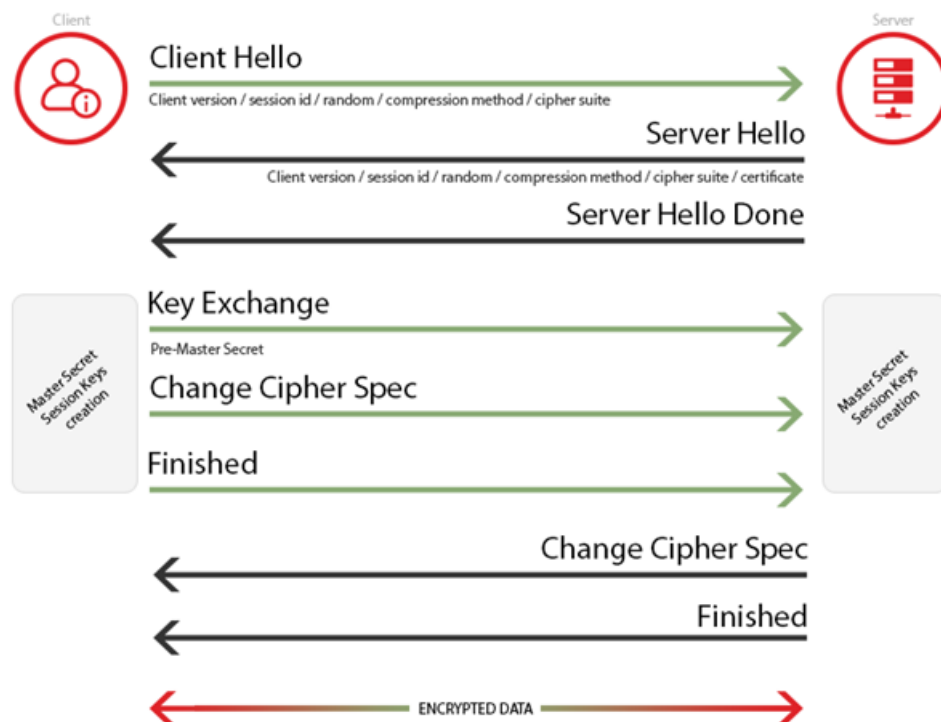


Figure 2 TLS Flowchart

The operations steps of TLS protocol are as follows.

Step 1: Client Hello

The very first step is sending a Hello packet. The client generates a Client-Hello packet that contains different parameters of the TLS packet: Client Version, Client-Random, Session ID, Compression_methods, Cipher Suites, and finally Extensions (Figure 3).

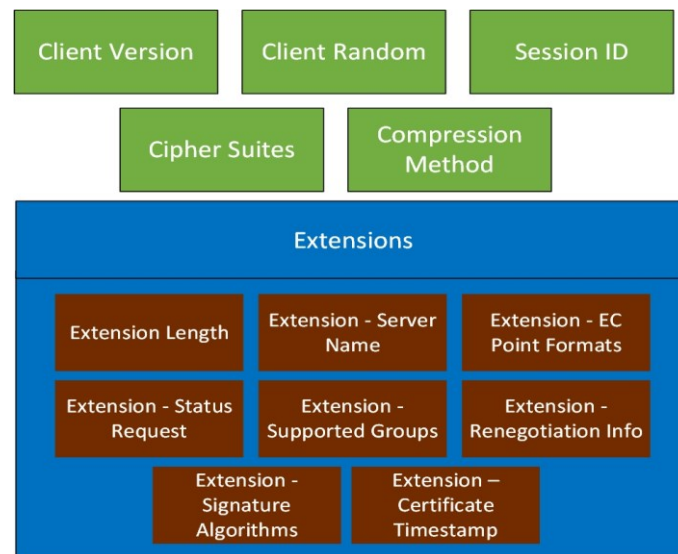


Figure 3 Client Hello Packet

Client Version: It represents the version of the TLS protocol used by the client. Since the last version of SSL was v3, so TLS v1 is represented by “3,1”, TLS v2 by “3,2” and similarly “3,3” represents TLS v3.

Client-Random: It represents the 32 bytes of data that is generated randomly by client.

Session ID: The client can provide the Session ID of the previous TLS session so that the communication can be resumed. For client to resume the previous session, client should load the key information from the memory if it remembers. This is advantageous because it will save a lot of computation time and network round trip. So, it is preferred whenever possible. If a new session is created, then the length of the bit identifier is set to “00”

Compression_methods: The Client will provide the list of compression methods that it supports in the order of preference. The compression will be used before encryption. But this feature can weaken the security of the Encrypted data so this has been removed from future TLS protocols

Cipher Suites: The client will provide the list of cipher suites that will be used for the exchange of information, encryption of data, and message authentication. The list is ordered in a manner of first most preferred to least preferred.

Extensions: The client also provides the list of optional extensions to the server that can be used to enable optional features.

Step 2: Server Hello

A Server-Hello packet is a reply to the Client-Hello packet to the client by the server. It may either contain the selected options from the Client-Hello packet proposed by the client or the “Handshake-Failure” message. So, either it can accept or reject the communication. Server TLS packet contains Server Version, Server-random, Session-ID, Cipher-Suites, Compression-Methods(Figure 4).

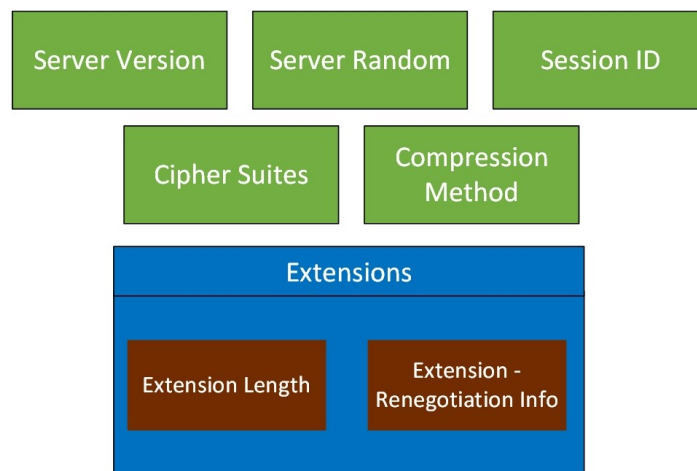


Figure 4 TLS Server Hello

Step 3: Server Certificate

After the server sends its Server-Hello packet, it also sends its Server Certificate. The Server Certificate is proof that the identity of the server is verified. Server Certificate contains the Signing Authority, Data of Issue, and Server’s Public Key.

Step 4: Client Certificate

Sometimes, the server can also ask the client to send its Client-Certificate to authenticate itself. This is an optional case.

Step 5: Server Key Exchange

The server sends its Server-Key-Exchange if the Server-Certificate provided to the client is not enough to generate and exchange a Pre-Master secret. This usually happens when specific encryption protocols like DHE_RSA, DHE_DSS and DH_anon are used.

Step 6: Server Hello Done

Server-Hello Packets makes the confirmation of finishing of Server-Hello message to the client and all the information required for TLS communication is sent by the server.

Step 7: Client Key Exchange

After the Server-Hello is done the server sends this message and while this message is received, the client creates a Pre-Master-Key. The pre-Master key is generated by the client and encrypted using the Server-Public key received in the Server-Certificate packet. The server accepts the Pre-Master-Key and decrypts it using its own Private Key. Now both Server and Client have pre-master-key, and they calculate Master-Key using a PRF (Psuedo Random Function).

```
master_secret = PRF(pre_master_secret, "master secret", ClientHello.random + ServerHello.random)
[0..47];
```

Now, this Master Key will be used by both Client and Server to generate Session-Keys so that data can be encrypted and decrypted on both ends.

Step 8: Client Change Cipher Spec

Now, the client is ready to communicate in an encrypted environment. This step shows that there is a change in encryption. So from now onwards, any data sent by the client is subject to encryption by a symmetric shared key.

Step 9: Finishing of Client Handshake

In the handshake process, this is the last message. It signifies the success of handshake is successful, and becomes the first encrypted message signifying secure communication between client and server.

Step10: Server Change Cipher Spec

Now the server is also ready for performing communication in an encrypted environment. Similar to the client, Symmetric shared key encrypts the data sent by the server.

Step11: Finishing of Server Handshake

This message symbolizes that it was the last message of the handshake process from the server-side. The Handshake from the server-side is finished.

A client initiates TLS communication using a TCP connection which is a 3-way handshake. The client sends a Hello packet that contains different types of Encryption mechanisms supported by it, and then the server replies back, and both mutually negotiate on common grounds for encryption to establish communication. So, this TLS exchange information can be used to Fingerprint any application that uses TLS.

2.4.2 Traffic features of TLS Protocol

In context of information security, TLS protocol provides three key concepts.

Authentication - in the TLS is achieved through the public-key cryptography. Public key cryptography server is authenticated through a certificate issued by authorized certification authority. The authentication is optional but recommended for all users.

Confidentiality - Symmetric cryptography ensures the confidentiality of the protocol. The Cryptographic keys that are generated in every session are subject to negotiation of specific symmetric algorithm.

Integrity - In every transmitted packet, Hashed Message Authentication Code is embedded. This ensures integrity and makes computationally infeasible to perform attacks since in TLS1.3 hashing MAC algorithms are SHA-256, SHA-384, and SHA-512.

2.5 Fingerprinting

Fingerprinting helps us to obtain unique information from an object, which is tabulated and for larger set of objects. Fingerprinting comprises of a variety of methods that use a broad spectrum of technologies to identify the unauthorized attack and detect malware. Fingerprinting has also established itself into the Cyber Security domain and is known by the term “Footprinting”. Cyber Security Fingerprinting can be used to identify and classify Networks, Users, Ip addresses, hardware devices, Software etc., on the Internet uniquely[25]. Many different tools and techniques have been developed and designed especially for these tasks. This assists in the forensics and examination of the Network, so as a result behaviour of the Network can be better understood.

A fingerprint must essentially possess the following characteristics.

Ability to Discriminate – Fingerprint should have discrimination power in order that it can produce different fingerprints for different targets so that a target can be uniquely identified among the options and create a homogenous demarcation.

Stability – The fingerprinting technique must produce the same fingerprint for the same target over multiple measurements and remain stable over time.

Efficiency – Fingerprint must generate the fingerprints in defined time and evaluate against a database of previous candidates.

Resilience to Evasion – Various methods are in vogue to avoid fingerprints or reduce their consequences. The fingerprinting technique must be robust enough to known or possible evasions.

Fingerprinting can be performed in an active or passive way. The active interaction with the target entity(active fingerprinting) helps to determine the adequacy and effectiveness of the defensive processes. Since this type of fingerprinted can be detected easily, the attackers use an alternative approach(passive fingerprinting). In this approach, the attackers silent observe the network traffic and do not interfere with the sources. The traffic is not sent to the server , rather collected by the attackers. However, passive fingerprinting yields lesser or insufficient information and sometimes may not be of much use.

2.6 Types of fingerprinting

2.6.1 Network based HTTPS Client fingerprinting

In Network-based HTTPS Client identification method, proper client identification is achieved by creating a dictionary, where the Cipher suite list of the client is paired with their respective User-Agent. From the TLS handshake, the elements chosen are list of Cipher suites. The global network address and IP-based geographical location of the user are analysed from incoming HTTP requests and proxy server can be detected using an explicit request. This methodology comprises of two approaches – (a) host based – server monitoring and (b) flow based – network monitoring[26].

2.6.2 Operating System fingerprinting

This type of fingerprinting involves detecting end-host operating systems, through analysis of packets originating from the system. This method of fingerprinting is limited to packets contain a full-fledged TCP connection that must have SYN, SYN/ACK, and ACK connection. The

methodology of OS fingerprinting is based on the premise that default values are set up different operating system which can be analysed to differentiate among the OS.

OS fingerprinting can be (a) Active(intrusive) – sending carefully designed packets to the target system to determine the target OS and examining the TCP/IP behaviour of received responses or (b) Passive(non-intrusive) – collecting samples passively from a host. Passive method of fingerprinting is more effective than active method since firewalls can be easily be breached.

For manual modes, tools like FTP, Telnet, and NetCat are used. Automated techniques are popularly conducted through NMAP and XProbe2. In TCP/ICMP Fingerprinting, the analyst attempts to identify the target host from the headers in a TCP packet/ICMP packet.

2.6.3 Browser fingerprinting

In a browser fingerprinting, the information collected through an interaction with the web browser of the analysed device based on its specific and unique configuration. The user information revealed from the browser and computer includes various parameters like version of operating system, browser user agent, network IPs, device IDs, batter information etc. These fingerprints include the IP address, screen parameters of JavaScript , browser headers, cookies' information, plug-ins, updates etc.

Various techniques have evolved to fingerprint browsers for we tracking. A widely used digital fingerprinting is use of canvas through HTML5 canvas element allows the browser API to draw invisible images or text. In WebGL the browser is forced to render images off-screen. The images can be analysed to extract information about device's hardware and graphics system. Fingerprinting via audio is based on analysing the manner the sound is played by the device.

2.6.4 Markov Chain Fingerprinting

In a Markov chain fingerprinting method, the first-order or second-order homogeneous Markov chains are used to classify the traffic, conducted at the side of the server. The analysis is carried out through a message type sequence emerging from a single-direction flow form the server to the client[27]. This method can be used for modelling the statistical fingerprints for various applications. Supervise machine learning in the Markov model has produced high accuracy rates[28] .It is established that incorrect implementation practice, misuse of SSL/TLS protocol, multiple server configurations are responsible for application discrimination[27].

2.6.5 JA3/JA3S fingerprinting

This technique is proposed by Salesforce[29], wherein both the ClientHello and the ServerHello are utilized to fingerprint the negotiation between the client and server. It uses MD5 hash to produce a 32-character fingerprint. In initial form, JA3 used only the client side of the TLS session establishing messages. JA3S overcomes the problem of JA3 in a situation where client applications use the same OS sockets or common libraries resulting in identical JA3 fingerprints[29]. The thesis has specific reference to JA3, client-side analysis of fingerprints.

JA3S is not relevant since the same server will formulate its Server Hello message differently depending on the Client Hello message and its contents. It is not possible to fingerprint a server just based on its Hello message like we could with clients and JA3. The way that the server responds is always the same for the same client, though different for a different client.

2.7 TLS fingerprinting

TLS fingerprinting is a method of extracting a certain parameter from a TLS client request (ClientHello) and comparing with a predefined or customized set of fingerprints to identify attack patterns. The concept of TLS fingerprinting was demonstrated by Lee Brotherston in 2015 at his Derby Con talk[26]. According to him, the TLS packets can be considered as objects, and information obtained from them can be used to generate Fingerprints. Once it is obtained, it can be used to identify any other similar TLS communication taking place on the Network. Now only the client is taken into consideration but not the server. This has mainly two reasons, *first* is that server can be probed easily that is information about any server can be gathered easily. They are always in the listening state; hence it can be determined what the server is, but this can't be done with a client because it is not always in the Listening state. The *second* reason is that the server always announces what it is if someone is connected to it. For example, for an SMTP server, it just announces that it is an SMTP server itself, and in a similar way, this is true for any other Server as well. So, there is no real need to fingerprint servers because they will always be the same once a person connects to it like. For example, if you connect to Apache box and the next time you connect to it will still an Apache box. So, we don't really need to determine things on a per-connection basis. Fingerprinting a client is important because we don't know what that client is. We have no idea what type of TCP connection it is

going to send as it can have different applications like different web explorers, which generate TCP connections differently.

It can be seen in figure 5 that we don't need the Length identifiers because they are present to navigate the packet so that different fields can be identified correctly, as the client Hello packet can be of different lengths. We can also drop the Session-Id because it is unique to the session, so it will keep on changing with every new session. Random can also be dropped because it is randomly generated by the client so it is not fixed.

Content Type	Version	Length
Handshake Type	Length	Version
Random	Session ID Length	Session ID
Cipher Suites Length	Cipher Suites	Compression Methods Length
Compression Methods	Extension	

Figure 5 Fields from Client Hello packet for Fingerprinting

In the end, we are left with the fields: Content-Type, Version, Handshake Type, Version, Cipher Suites, Compression Methods, and Extensions, so this information can be used to obtain a fingerprint.

2.8 JA3

This idea of TLS Fingerprinting proposed by Lee Brotherston was further researched by Engineers working at Salesforce John B. Althouse, Jeff Atkinson, and Josh Atkins, and hence JA3 was made. The underlying principle behind this research is that contents of the TLS Packet send by the client remains the same even in the different sessions, so Fingerprint can be made to identify different Clients in the same session. So, we can differentiate and mark each client uniquely, which is running on the same session. The concept of JA3, which was used in this thesis, generates the fingerprint in the form of a “hash,” which can be used to identify and classify different or similar TLS communications.

These engineers at salesforce identified that in some cases for the Cipher suites in the Client Hello Packet appear more secure and go to less secure. Still, in some other cases, this was not the case i.e., the order was random. So, this gave them an idea that this can be used as a criterion to make fingerprints so that clients can be fingerprinted based on SSL/TLS version. So, they decided to take this criterion along with a list of extensions in the order of their appearance, Elliptic Curve, and Elliptic curve formats to further added some more complexity so to make Fingerprint more unique to a client.



Figure 6 Generating Ja3 Fingerprint

Therefore, a Fingerprint is the resultant of

$$JA3 = MD5(\text{Version}, \text{Ciphers}, \text{Extension}, \text{EC}, \text{EC_Point_Formats})$$

So, the client version is taken, then cipher suites are the order in which they are seen, then extension in the order in which they are seen, EC that are supported in the order in which they are seen, and finally EC_Formats and then finally all are hashed together(Figure 6). In client identification, the five fields are altogether sufficient for any analysis[30].

Now it is extremely that this is strictly followed because this is how they are ordered and implemented. This information makes it unique for any client application. Now 300 different ciphers are supported, so the string can get very large. So, the MD5 hash was chosen because it generates only 32 characters no matter how long the input string is.

There may be some cases when there are no extensions. In those cases, the extensions field is left behind and followed by a comma. For example, in the case of Microsoft Edge, the fingerprint from the client hello packet assumes the form in the given table.

Table 1 Typical Fingerprint in Microsoft Edge

Parameters	Description
Version	771
Ciphers	4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53
Extension	0-23-65281-10-11-35-16-5-13-18-51-45-43-27-21
EC	29-23-24
EC_Point_Formats	0

For the above information , MD5 hash is generated as follows.

MD5(771,4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-156-157-47-53,0-23-65281-10-11-35-16-5-13-18-51-45-43-27-21,29-23-24,0)

So, Fingerprint = **b32309a26951912be7dba376398abc3b** is obtained for **Microsoft Edge**

Similarly, **f15797a734d0b4f171a86fd35c9a5e43** is the JA3 fingerprint for **Firefox on windows**.

Table 2 Partial List of common applications with their JA3 fingerprints

Application Name	JA3
Chrome (Windows)	b32309a26951912be7dba376398abc3b
Firefox (Windows)	f15797a734d0b4f171a86fd35c9a5e43
Tor Browser	e7d705a3286e19ea42f587b344ee6865
Trickbot Malware	6734f37431670b3ab4292b8f60f29984

Source : Althouse [3]

The analysis was made on commonly used user applications on different operating systems: Android, Linux, and Windows. The analysis was made on live capture of packets originating from these operating systems. A “JA3-Hash” database was created where the application name is listed and the JA3 fingerprint hash value tagged is tagged along with it. The thesis presents the analysis and the results obtained on the TLS Fingerprinting on various applications, so the reader will have overall knowledge about how the different applications behave in the different environments and which applications use the same or different TLS libraries on the same or other architecture.

2.8.1 Benefits of JA3

Compared to domain names or IP addresses, Ja3 offers several advantages. The benefit of utilizing JA3 is that it has many advantages in detecting breaches and compromises (Indicator of compromise). In various datasets investigations, Ja3 can be used to detect identical hashes in other traffic communications. It is certainly an improvement over the conventional approach to cybersecurity like backlisting, data breach detections, whitelisting, etc. [31]. Interestingly, malware detection is based on the communication (transport layer) rather than the connection sought by the client (internet layer) [29]. JA3 hashes as input signature make it one of the most specific and reliable security solutions for malware analysis and management. The integration of Version, Ciphers, Extension, EC, EC_Point_Formats provides a robust mechanism to identify clients.

2.8.2 Limitations of JA3

JA3 may be the same for two client applications. Sometimes, the OS sockets or libraries are common in a client application resulting in a shared JA3 hash. In that case, the same Ja3S will be reflected for an application communicating with the server. Malware may use a common library in C&C server communication, making JA3 indistinguishable from a genuine one. However, the response of the C&C server is unique, thus detecting the malware. The number of fields used to generate fingerprints makes the possible Ja3 and Ja3S limited. Modification of ClientHello's values by the attacker is not impossible, though rare. The blacklisting performed by Ja3 has similar limitations as in other mechanisms.

2.9 Attacks and Vulnerabilities of SSL/TLS

Though data protection through SSL/TLS protocol is extensively exercised, yet there are various flaws in the security mechanism that distort the integrity of data usage. Following types of known attacks can be executed on the SSL/TLS Handshake protocol.

2.9.1 Downgrade Attacks

FREAK – Discovered in 2015, it manipulates the implementation weakness of SSL/TLS. Typically, these protocols brace “export-grade cryptography”. Immediately upon connecting to the webserver by the client, MITM downgrade TLS connections. to that uses keys of 512 bits or even less. The special format handshake messages in the RSA_EXPORT cipher suite are

accepted without any request being made(Figure 7). This bug was due to US government earlier regulations, and the solution is disabling support for export cipher suites.

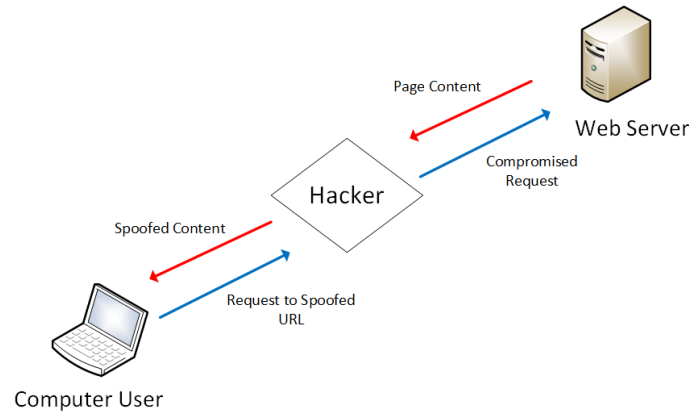


Figure 7 Freak Attack

LOGJAM - Similar to FREAK, this attack was directed on Diffie-Hellman key exchange (published in 2015), which is a popular method of exchanging cryptographic keys in a secured manner. The fault in the TLS protocol is that the server signature in a Diffie-Hellman key exchange (DHE) handshake coverage is limited to DH parameters(Figure 8). It is said that the best way for performing the attack on DHE is to compromising one of the private exponents that compute the discreet log. If the attacker finds the discrete log, he can calculate the private key.

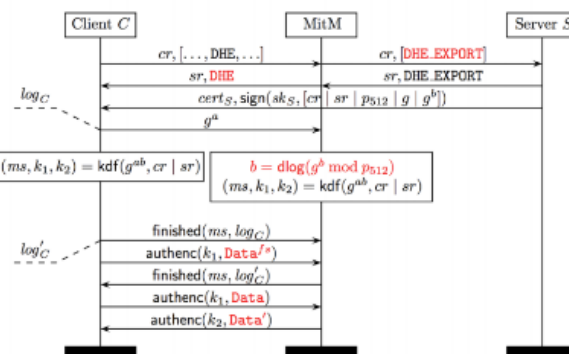


Figure 8 Logjam Attack

Thus, clients and servers are required to disable export cipher suites to tackle this vulnerability.

POODLE - Padding Oracle on Downgraded Legacy Encryption builds on the vulnerability in the SSL 3.0, though it is not present in newer versions like TLS. For this type of attack, MITM, the attacker has first to understand the communication between the client and server and then must convince the server to use SSL3.0 protocol, thus defying the client that the client cannot use TLS1.2.

Published in 2014, the only solution to respond to Poodle is disabling previous SSL versions and supporting TLS downgrade protection.

VERSION ROLLBACK - Since SSL3.0 supports SSL, a version rollback is a possible vulnerability. The attacker leads the server and client-side in the communication forcing them to use lower versions of SSL2.0. To prevent this attack, the TLS 1.3 mechanism ensures that the client and server send a finished message containing MAC over the previous handshake, thus ensuring that negotiated parameters are not modified by MITM. In SSL2.0, if the server supports non-RSA key exchange, this attack becomes redundant.

CIPHERSUITE ROLLBACK - During the handshake phase, the cipher suite lists cryptographic algorithms that travel in a clean-text format during the initial ClientHello messages between the client and server(Figure 9).

MITM attackers can intercept and imitate the valid user, thus obtaining credentials. This vulnerability has been handled by authenticating the handshake message in the Finished message containing MAC. Changing cipher specifications in the finished message's or a warning message for error are the solution that has been proposed[32].

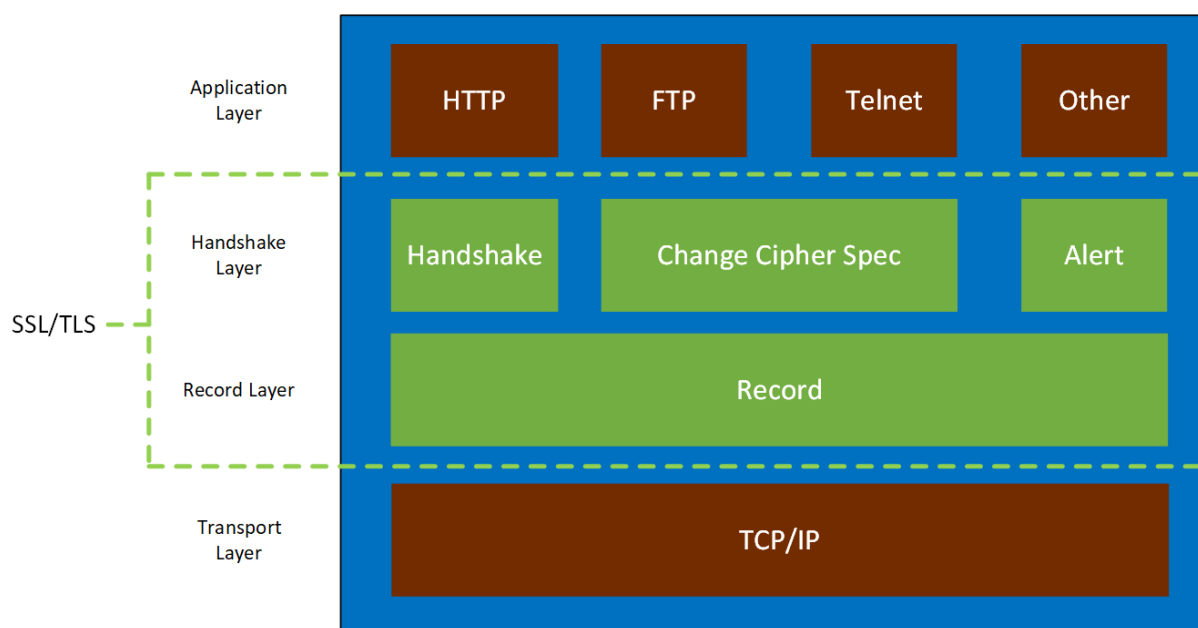


Figure 9 CipherSuite Rollback

2.9.2 Cross-protocol Attack

Cross-protocol attack, also known as Key Exchange Algorithm confusion making SSL3.0 vulnerable. The server sends temporary key parameters as per the long-term certified signing key to the client wherein the type of key is not specified in the parameter creating confusion.

The client is forced to use the RSA key exchange, and the server is forced to use the Diffie-Hellman key exchange (Figure 10). Confusion arises when the client interprets Diffie-Hellman parameters as a constituent of the RSA key. Creating a new format of ServerKeyExchange message is a solution to tackle this attack.

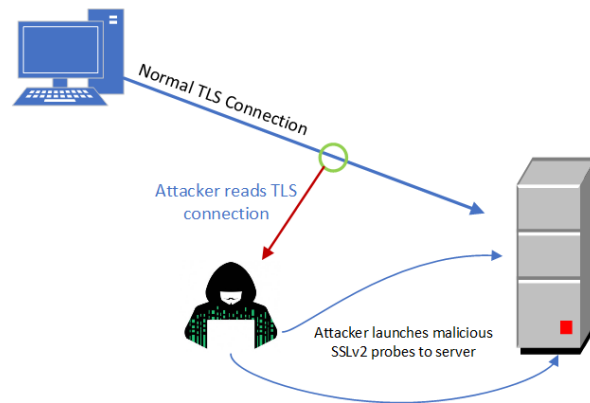
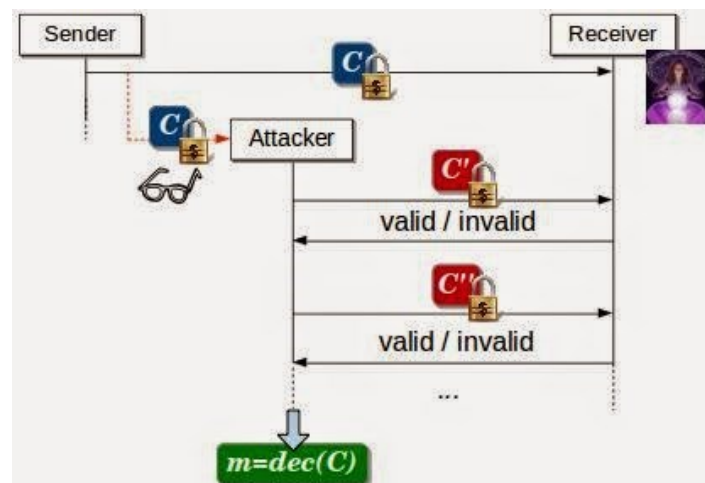


Figure 10 Cross-Protocol Attack

2.9.3 Bleichenbacher Attack

Presented by Daniel Bleichenbacher in 1999, this attack happens when the exchange of keys takes place using RSA algorithm and PKCS#1 v1.5 padding relying on oracle. Using the RSA algorithm, pre-master secret is encrypted via PKCS v1.5 within the ClientKeyExchange messages.



Source: <https://web-in-security.blogspot.com/2014/08/old-attacks-on-new-tls-implementations.html>

Figure 11 Bleichenbacher Attack

The solution to this vulnerability is to treat the incorrectly formatted messages in an indistinguishable manner vis-à-vis the RSA blocks that are correctly formatted.

RSA-Based Sessions in SSL/TLS presented by Klima, Pokorny, and Rosa in 2003 is based on a random pre-master secret. The pre-master secret value, if recovered by the attacker, results in decryption of the whole SSL/TLS session captured[33].

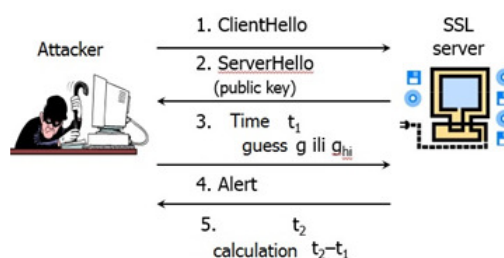
Some new Bleichenbacher attacks, like internal exception error messages, timing differences in JSSE, etc., have also been published in 2014[34].

2.9.4 Drop ChangeCipherSpec Attack

David Wagner and Bruce Schneier indicated Drop ChangeCipherSpec attack vulnerability in SSL2.0. In the handshake phase, both the parties in the communication are notified by ChangeCipherSpec message, and a signal is sent via ChangeCipherSpec message that the communication will be performed with the agreed parameters. In the process, MITM sends the finished message before the ChangeCipherSpec message is sent, resulting in the start of false communication or deleting the ChangeCipherSpec, implying failure of communication. Thus, receipt of the ChangeCipherSpec message by both parties is a must before accepting the Finished message to avoid this vulnerability.

2.9.5 Timing attack

In this attack, timing variants, for different input data values of the cryptographic operations are exploited. On the server, the computation of the private key is performed by computing the time difference that lapses between sending a tailored ClientKeyExchange message and receipt of Alert message that triggers the irregular pre-master secret[35] illustrated in the following figure as demonstrated by [36].



Source: J. Ćurguz - Vulnerabilities of the SSL/TLS Protocol

Figure 12 Timing Attack

RSA blinding can provide an effective defence against this attack.

2.9.6 Other Attacks

SKIP-TLS - It was found in 2015 that in open-source TLS implementations, many of these implementations can incorrectly allow skipping of some messages in spite of being essential for the selected cipher suite.

The attacker on the network can send an arbitrary website's certificate, skipping the rest of the message and the client accept the certificate, thus exposing unencrypted application data.

BEAST - Browser Exploit Against SSL/TLS was performed by Thai Duong and Juliano Rizzo in 2011. In a BEAST attack, the man-in-the-middle attacker decrypts the data flowing between a web server and browser at the end-user side for the Cipher block chaining suites in TLSv1.0 and previous versions (Figure 13). Manipulating the block boundaries cleverly allows a MITM to sniff the encrypted traffic in order to find out tiny pieces of information without performing decryption that is available for manipulation irrespective of type/strength of block cipher. The theoretical vulnerability of BEAST can be combined with other vulnerabilities to perform a major attack.

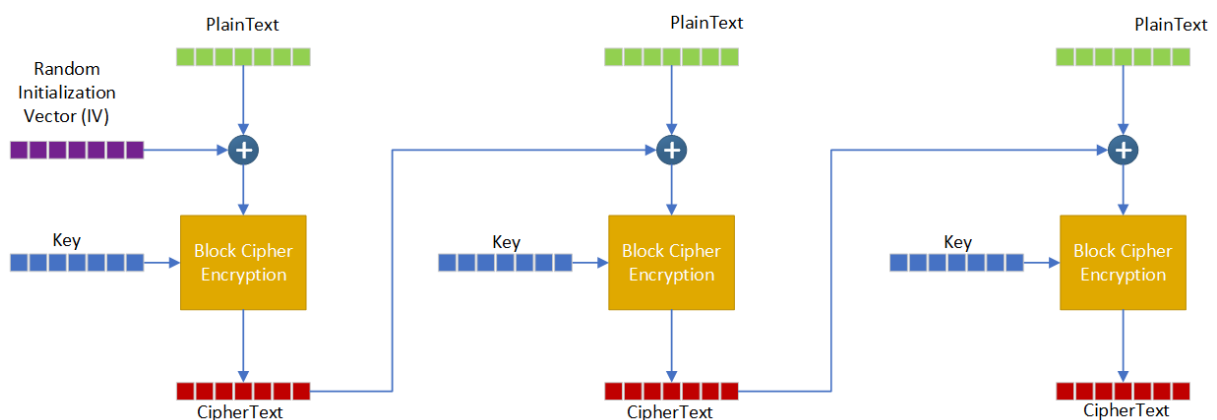
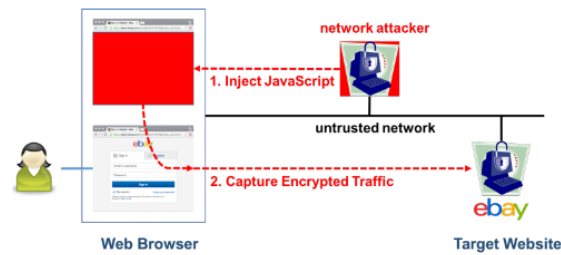


Figure 13 BEAST Attack

Various methods like switching to stream cipher, changing block cipher mode, insertion of empty packets for consuming unsafe initialization vectors, using 1/n-1 packet splitting, etc., have been deployed[37]. Presently, switching to TLS 1.1 and TLS 1.2 can eradicate its vulnerability.

SWEET 32- Published in 2016 by research team of French National Research Institute for Computer Science. The motivation for the attack is drawn from the design flaws in some ciphers viz. DES and Triple-DES (3DES) ciphers that are used in TLS, OpenVPN, SSH etc. [38].

MITM can recover some plaintext data from long-duration connections and steal a large quantum of encrypted traffic.



Source: <https://www.tomshardware.com/news/sweet32-64-bit-cipher-collision-attack,32550.html>

Figure 14 Sweet 32 Attack

LUCKY 13 - It is a cryptographic timing attack that can be performed against TLS protocol implementations using the cipher block chaining mode of operation. Nadhem J. AlFardan and Kenny Paterson in 2013 demonstrated this attack on Free Software implementations of TLS and shown that all products are vulnerable. All the versions of TLS that use the cipher block chaining mode of encryption are vulnerable to this type of attack. In this attack, the plaintext recovery for OpenSSL is possible that allows the attacker to read the plaintext(Figure 15).

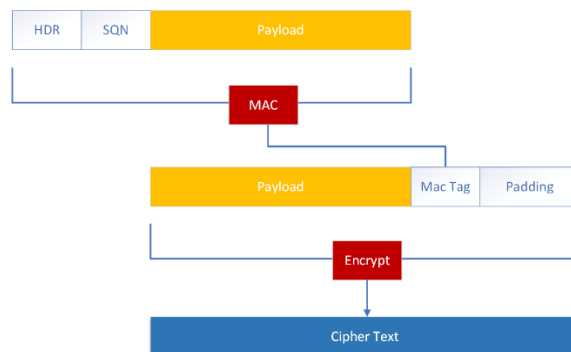


Figure 15 Lucky 13 Attack

The attack utilizes the different calculation times based on plaintext padded[39]. A suggested method to prevent the attack is Authenticated encryption with associated data cipher suites. However, during the disclosure time of the attack, implementations did not support TLS1.2 Version. In the recent past, various implementations have tried to limit the timing leaks.

HEARTBLEED – It is a buffer over-read OpenSSL bug of the cryptographic library that allows the unauthorized user to extract data from software over and above what is allowed. In Heartbleed, a malicious heartbeat request usually with a large length field and a small payload is sent to the target to bring out victim's response that allows for reading of the victim's

memory(up to 64KB), as in OpenSSL(Figure 16). Publicly disclosed in 2014, [40]inform that this bug was present on thousands of web servers and major websites.

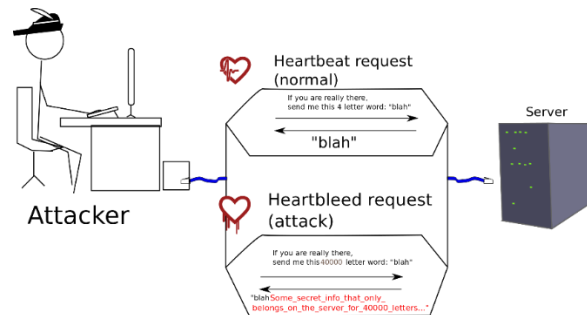


Figure 16 Heartbleed Attack

Source: <https://thecustomizewindows.com/2014/04/openssl-and-the-heartbleed-bug/>

We can see that various attacks have been discovered in the evolving versions of SSL/TLS. The change of configurations of the client or server has been an acceptable solution in many cases. However, many invasive changes have been made to the implementation to handle the vulnerability in some cases. Vulnerabilities are still evolving with the complexity of applications and communication modes between the devices and network systems.

3 Review of Literature on JA3 Fingerprinting

Transport Layer Security Protocol(TLS) endeavours to provide the security and integrity of data between the communicating networks. The two main components of TLS are Record Protocol and Handshake Protocol. Record Protocol is responsible for the connection security of the channel during the data delivery process. Handshake Protocol handles the authentication and establishment of the key used to encrypt the application as demonstrated by Krawczyk et al. [41]. They have extracted a key encapsulation mechanism (KEM) from the TLS Handshake Protocol to illustrate the security of the TLS protocol using a secure authenticated encryption scheme in the Record Protocol. TLS fingerprinting has evolved as a method of detecting malicious activity on a real-time basis. Bujlow *et al.* [42] argue that various new fingerprinting techniques have emerged to identify the user without reliance on client-based storage and enabling private browsing mode. The data security approaches have emerged from a host of privacy friendly techniques to highly invasive techniques like fingerprinting. From a user perspective, a script execution blocking browser extension can be combined to render browser fingerprinting impossible coupled with the malfunctioning risks of significant portion of websites. A significant number of researches on TLS fingerprinting analysis have been attempted on various mobile and android applications because these applications communicate on the Internet regularly with lesser active user interactions for features like updates, remote sensing, and multiple kinds of synchronizations according to Razaghpanah *et al.* [43]. We find very little researches on fingerprint analysis of applications on operating systems. One impactful research work is conducted by Latovicka *et al.* [22] for TLS identification of OS on the encrypted traffic. On analysing the available literature, we find three main perspectives – (a) Evolution of TLS versions and their applications, (b) Methods of analyzing flow traffic, and (c) Techniques for improving the robustness of fingerprinting mechanisms. These have been summarised in the following sections.

3.1 Flow Traffic Analysis Methods

The classification of network traffic can be performed using a variety of approaches. For payload-based classifiers, we need knowledge of a packets' structure and feature-based methods that utilize the specific characteristics to analyse the protocol flow. Veelan *et al.* [44] authors have shown that deciding the best approach is inconclusive because of the primarily sole dependence on the data sets used and configurations applied for analysis. In an analysis of high-speed networks' traffic, the monitoring of flow is a preferred method compared to the

traditional approach of analysing packets. TLS handshake information is incorporated in the network flow using IPFIX as an export mechanism in the work of Hofstede et al. [45].

Various network management tasks like dynamic access control, quality of service, intrusion detection, etc., are based on real-time traffic classification proposed by Paxson [46]. Flow analysis is required to recognize the application protocols for dynamic correspondence. Of the various methods for protocol recognition - port-based or payload based, payload based methods cannot deal with the applications running on the dynamic ports. Also, if the packet payloads are encrypted, it becomes difficult for the payload-based classifier to compare packet payload with known signatures.

The HTTP User-Agent string used to identify the clients is unreliable, and the reciprocal use of identifiers by web browsers to resolve compatibility issues has been in practice for a long time as argued by Husak *et al.* [47]. Pioneering work in fingerprinting is the release of an Apache module by Ivan Ristic that monitors SSL handshakes and correlates the offered cipher suite lists with HTTP User-Agent strings in the work of Husak *et al.* [47]. The authors established that the User-Agent of a client in HTTPS could be estimated using SSL/TLS handshake analysis. The fingerprints differ among clients and correlate to User-Agent values from an HTTP headers[48]. Also, the TLS/SSL fingerprints of the users trying to hack/misuse the application were different and can be detected and monitored.

Levillain *et al.* [49] have conducted a long period enumeration of the HTTPS servers on the Internet and find that a significant number of servers are intolerant to some cipher suite lists and detected malicious certificate chains.

Using the previously existing techniques that enable a TLS Fingerprinting Ja3 iRule, specifying rate limits and blocking based on TLS signature, authors have created a proc iRule named fingerprintTLSproc shown by ArvinF [6]. The authors monitor TLS signatures and the corresponding IP addresses and filters using iRule and generate logs sent to a high-speed logging server. Based on the rule, the traffic can be dropped, and malicious TLS can be detected.

3.2 Fingerprinting Robustness Evaluation Approaches

Various classification approaches based on machine learning are also evolving. There is wide use of flow statistics, classification models based on naïve bias, support vector machine, etc., for flow classification Erman *et al.* [50], Zander *et al.*[51].

Metadata analysis of the encrypted traffic is the focus of researchers. They are using two types of research philosophies. One focuses on the statistical analysis of the encrypted transmission data. Other focuses on the encryption features resulting from TLS handshake, also called fingerprinting, which is the essence of this project. The major contributions to the work on TLS fingerprinting include the solutions proposed by Anderson *et al.* [20], Husak *et al.* [8] and Anderson [52]. The most popular implementation methodology of fingerprinting, called JA3 fingerprinting, is proposed by Althouse [29] to serve multiple purposes of (a) identification of network applications, (b) malware detection, and (c) blacklisting.

To prevent the extensibility failures in the TLS ecosystem GREASE (Generate Random Extensions and Sustain Extensibility) mechanism has been proposed by Benjamin [53]. Effectively implemented servers and software can fetch the “GEASE” values from the client without failures or connection termination.

Kotzias *et al.* [19] have conducted passive monitoring of the SSL/TLS connections to analyze the changes in TLS cipher suites and the extensions offered by clients and accepted by servers by omitting GREASE values from TLS handshakes. Fingerprint mappings to a library, program etc., throws light on the stability and collisions of TLS fingerprints.

Some authors have conducted statistical analysis to monitor encrypted network traffic. Deri *et al.* [54] have demonstrated how their method, combined with novel techniques, can characterize and fingerprint encrypted traffic effectively. Network analysis algorithms have been improved so that a DPI engine can be constructed to do more than just identify network protocols and decode few packets. This helps us to check traffic against the compliance model. Since multiple fingerprints exist by same applications, JA3 can also be used as an indicator of change.

To analyze the application behavior, Anderson *et al.* [20] have combined the end host data with the network data. They have observed the deterministic pattern of GREASE values and linked them to a fixed string 0a0a. Using Levenshtein distance, they determined the similarity of TLS fingerprints. They have observed the changes in default parameters by some TLS libraries to suit their running platform better.

Machine learning can be used with Ja3 to identify malicious connections requests has been demonstrated by Heinemeyer *et al.* [31]. The almost unique fingerprints can be used for whitelisting, blacklisting connections, and also searching for an indication of compromise. The author has developed an AI algorithm that autonomously detects unique Ja3's analogous for a

network and can also classify JA3 originating from unusual devices. A particular Ja3 that has never been seen in the network can be identified and flagged for suspicious activity.

3.3 Research Gap

It is observed from the literature that many studies have been performed that refer to Ja3, TLS fingerprinting, and Network Analysis. While some studies have tried to highlight the evolution of TLS versions and their benefit and limitations over their previous versions, the other set of researchers have attempted to conduct a statistics-based network traffic analysis. Machine learning models, Markov chain applications have also been attempted for TLS analysis. Some authors have also worked on Ja3, focusing on mobile device fingerprinting. However, there is no impactful research that emphasizes TLS fingerprinting based on Ja3 that does a comparative analysis of different operating systems. The server TLS response is likely to be identical in majority of the cases for different applications on different operating systems and also for same application with different versions. This implies that JA3 analysis of servers is of little use. However, the client software generates TLS packets with unique contents. **The motivation of the thesis is to examine the uniqueness of fingerprints produced by a client on different operating systems and perform a cross comparison among them.**

3.4 Research Questions

The research questions answered in the study are as follows.

RQ 1: What is the list of the possible fingerprints generated by a particular client?

RQ 2: Do different clients generate similar fingerprints on the same operating system?

RQ 3: Whether different clients generate similar fingerprints on different operating systems?

RQ 4: For a given client, is the JA3 fingerprint same on different operating systems.

4 Data and Methods

The objective of the study is to analyse the fingerprints generated by the clients on selected operating systems. Various methodologies have been used to gather the required datasets on various applications on Linux, Windows and Android. The most important part of the data capture was to isolate the network activity so that network activity can be captured for a particular application. So, it was important to either isolate the application activity or isolate their network activity by any means. Different methods were used for Windows, Linux and Android. Obtaining data on Android is relatively easier compared to Windows and Linux since in these operating systems background communication poses difficulties.

4.1 Schema of the study

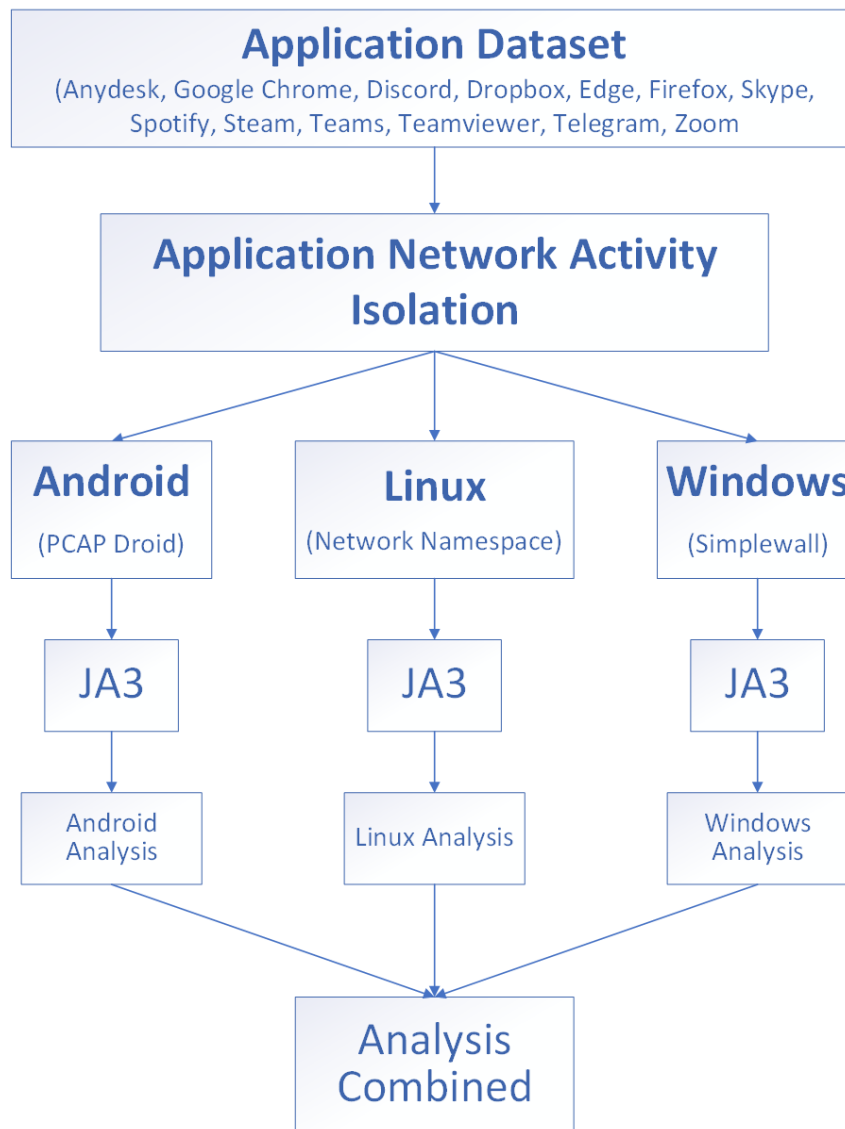


Figure 17 Schema of the Study

The following sections present in detail the methods of data generation on respective operating systems.

4.2 Data Generation on Operating Systems

4.2.1 Android

The android device used to generate dataset is One Plus 7 with Android 11 wherein the required applications have been installed. With android devices, a common seamlessness problem is encountered when an application's background process pops up a dialog in response to some event[55]. There is no direct way to isolate the application network activity on android since there is a host of background process by google that are constantly communicating with google servers. Therefore, an application called PCAP Remote has been installed. PCAP Remote is a network sniffer application which does not require root permission[56]. It allows us to debug and analyse Android traffic for all the application installed on the smart phone. The application used the built-in SSH server. The traffic is captured using an Android built-in service called VpnService. When VpnService is started as shown, a new Tunnel virtual network card is generated. All traffic on the phone passes through this virtual network card. Earlier versions of Android supported only built-in VPN connections, and VPN servers had to be standard PPTP or L2TP/IPSec implementations, but starting with Android 4.0, provides the VpnService which is family of interfaces and now it can allow developers to implement their own custom-built VPN protocols. This API provides the ability to intercept IP packets. In general, it creates a virtual network interface, configures addresses and routing rules, and returns a file descriptor to the application. Each read from the descriptor retrieves an outgoing packet which was routed to the interface. Each write to the descriptor injects an incoming packet just like it was received from the interface. The interface is running on Internet Protocol (IP), so packets are always started with IP headers. The application then completes a VPN connection by processing and exchanging packets with the remote server over a tunnel. After the VPN service is started, the applications installed starts to communicate with the System Network. Instead of communicating with external directly, System Network has a Local Tunnel Interface which communicates with the VPN App which has invoked the VPN Service and then packets from the device are sent to external network.

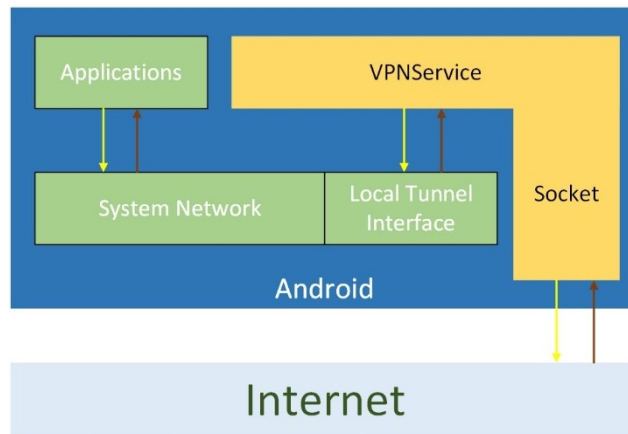


Figure 18 Working of VPN service inside Android

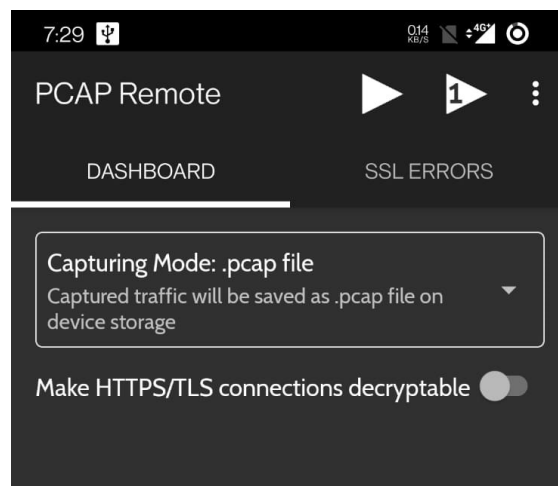


Figure 19 Dashboard of PCAP Remote android app

Figure 18 shows the dashboard of the application. It gives two option to capture either in “.pcap” format or by SSH server. Pcap format was selected in order to reduce complexity. We have an option to either capture for whole device or target a specific application, so we selected for the specific application because analysis is to be made on a specific application separately. After we hit the stop button, it will ask us to give a location in storage where we want to save the file.

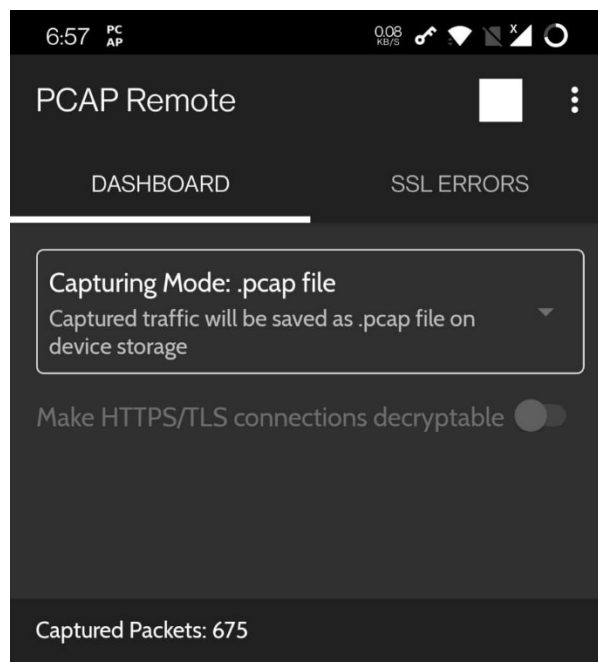
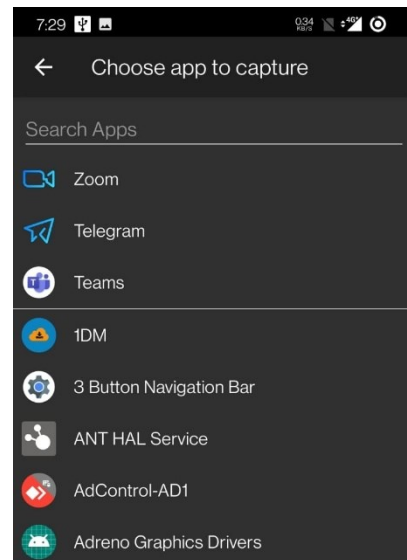
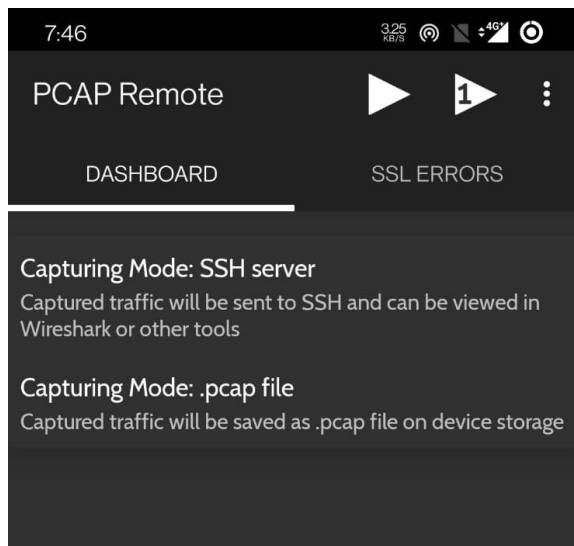


Figure 20 Working of PCAP Droid

4.2.2 Automation in Android web browser

Android device applications generate a a lot of different Ja3 hashes from the web browser. In computing

Ja3 fingerprint, the data is in clear text and the fingerprint is obtained by concatenating and hashing[57]. However, it was decided to automate the process so that random websites will be opened and corresponding data traffic is generated. A separate ADB shell script has been

written that uses Android Debugging Bridge to execute shell script in android. This script opens different web pages on the browser for a set time interval. After opening a specific page, the tab is destroyed and another website is opened on a different tab. This was made in a way so that memory consumption can be avoided by limiting the opening of different tabs. List of websites was added in a CSV file, the script will load the website from the list sequentially. The website list was made to have domains from different countries and different continents. The list of websites has been obtained from Netcraft. This list contains the set of most popular websites from a country and region, all mixed together in one CSV file. The main idea was to visit different random websites so that different TLS packets (1.1 or 1.2 or 1.3) can be obtained.

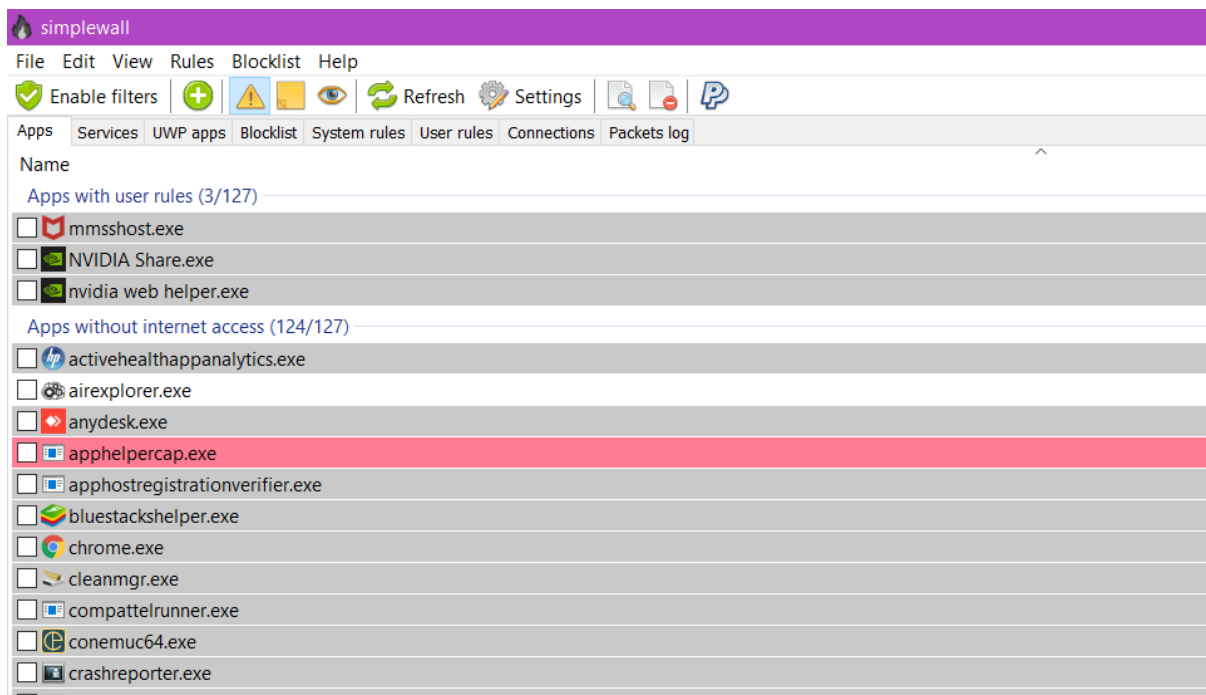
4.3 Windows Traffic Analysis

The analysis of fingerprints is challenging compared to other operating systems. Windows generate more background network traffic compared to Linux and Android. Windows contain many services and Universal Windows Platform (UWP) apps, on top of this application imported custom rules as made by Windows firewall. There is a constant communication from SSDP and local host. To stop this communication, a Firewall management software for Windows called as Simplewall has been used.

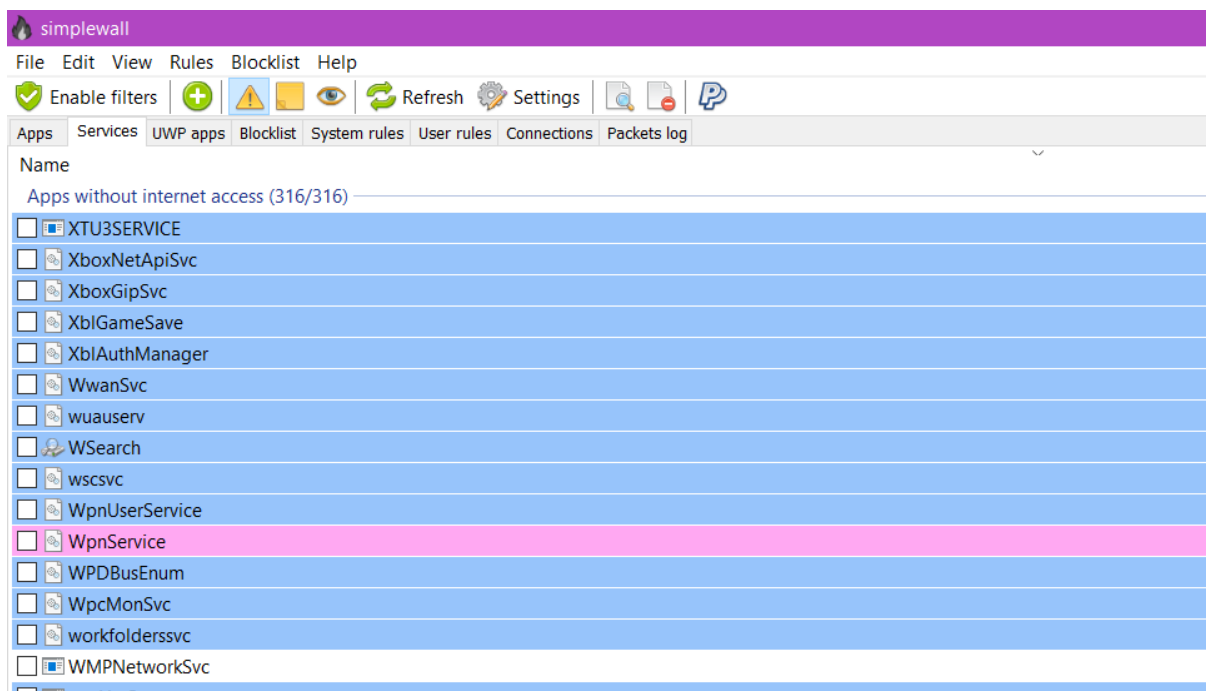
Simplewall helps us to apply filters to Wfp, that aid in exercising control on services and protocols that are allowed or blocked[58]. This application provides a simple interface that is user friendly and can be used to control network activity on Windows to large extent. One simpler tool called as TCP LogView has been used to monitor the active applications, it maps the Source Address, Source Port with Destination Address, Destination Port with Application name. It thus facilitates in spotting an application which is using which IP and port to communicate with outside network.

Following procedure has been implemented to Windows.

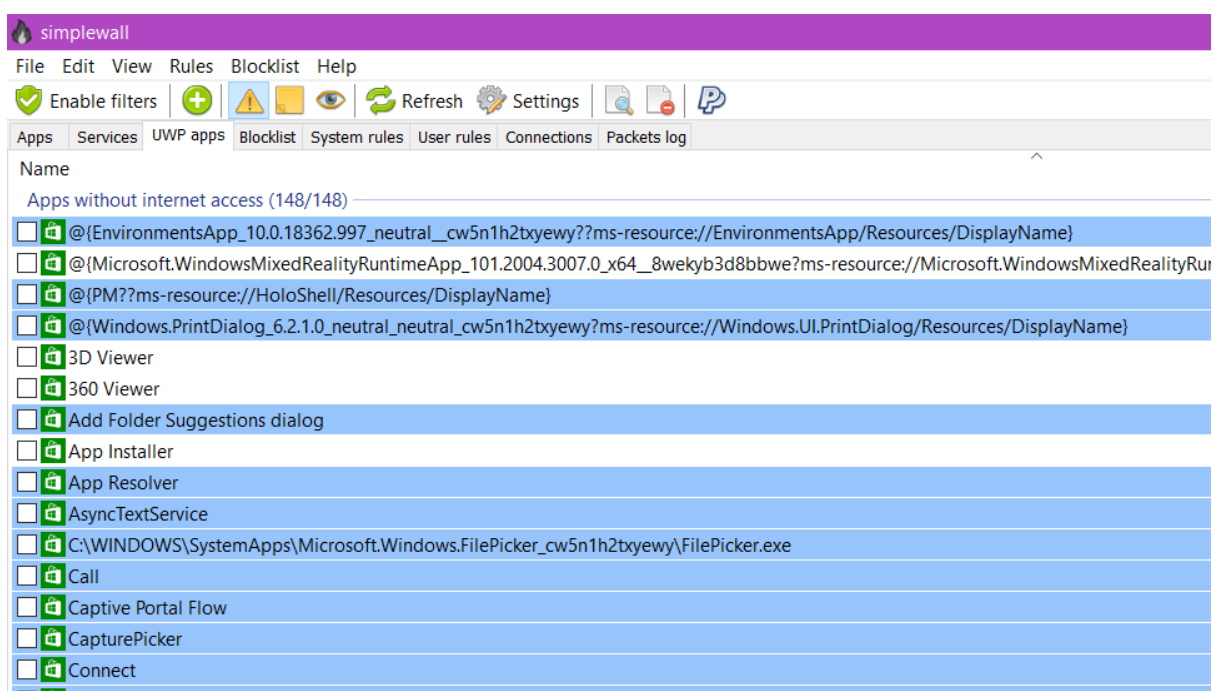
STEP 1: Block all Applications Installed



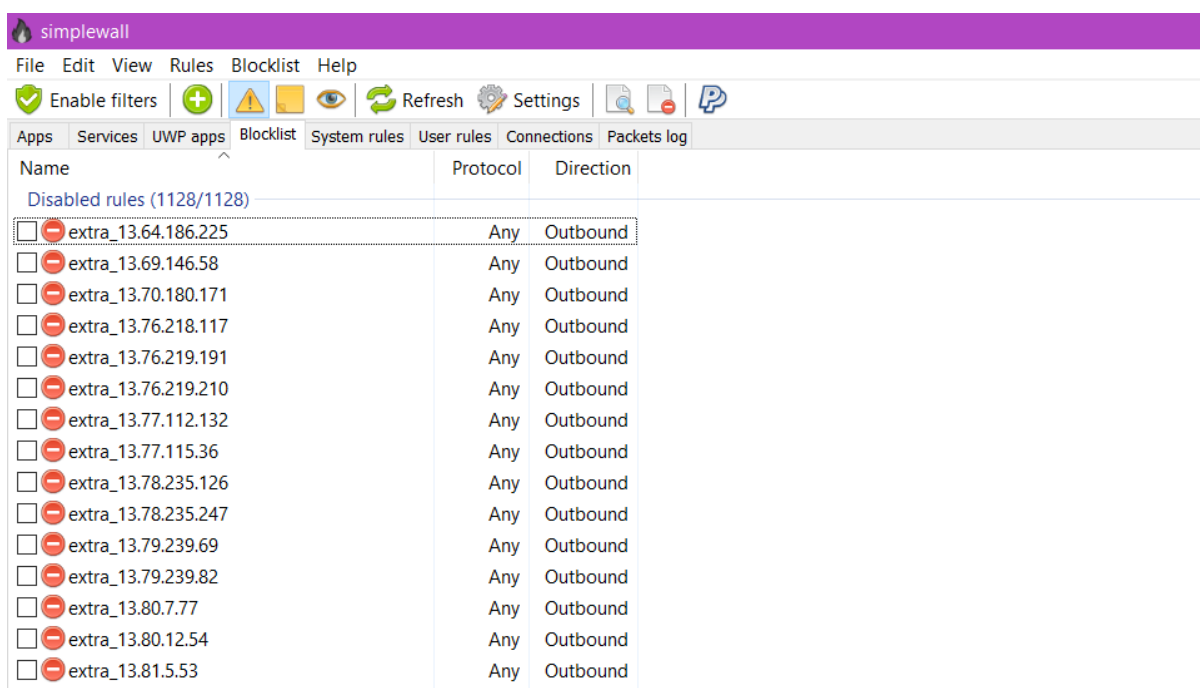
STEP 2: Block all Application Services



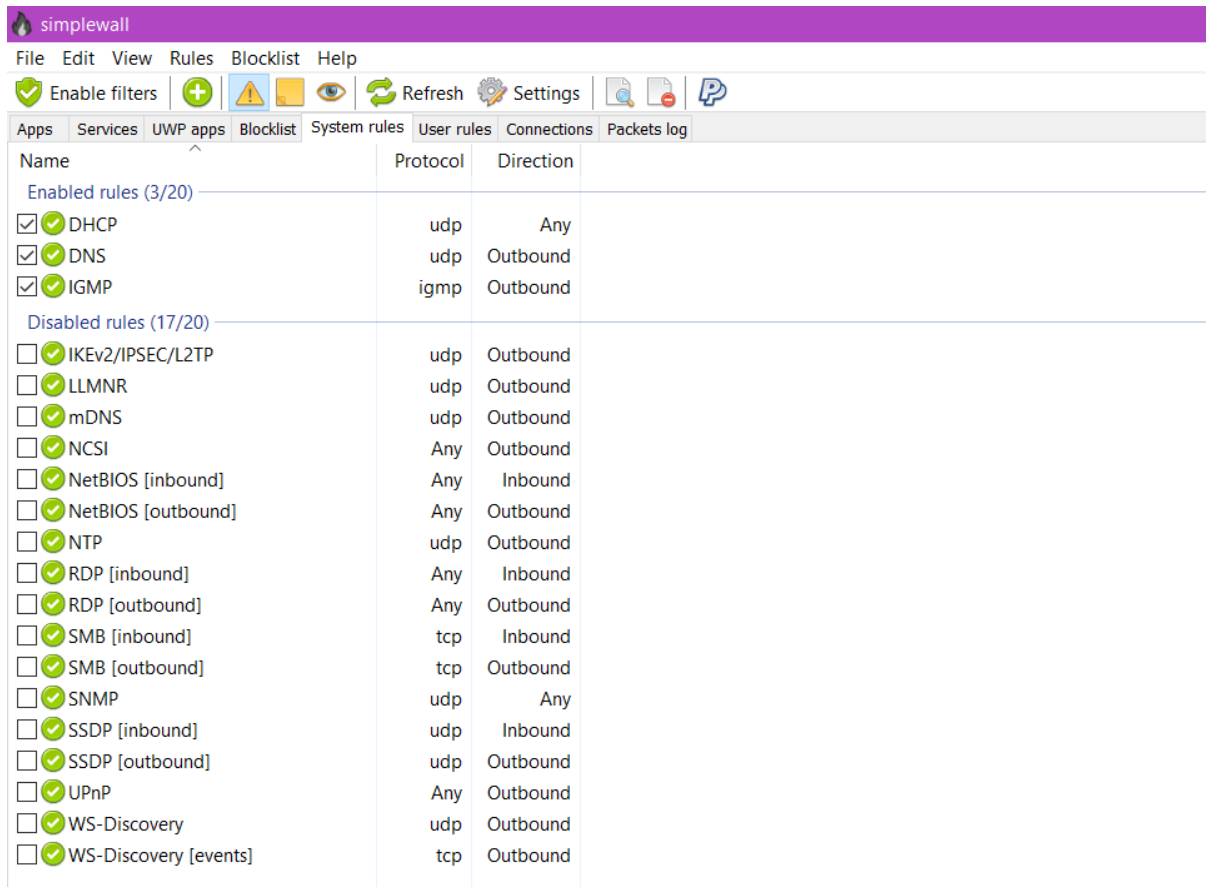
STEP 3: Block all Windows services and Universal Windows Platform Apps



STEP 4: Disable all custom IP rules imported from Windows Firewall



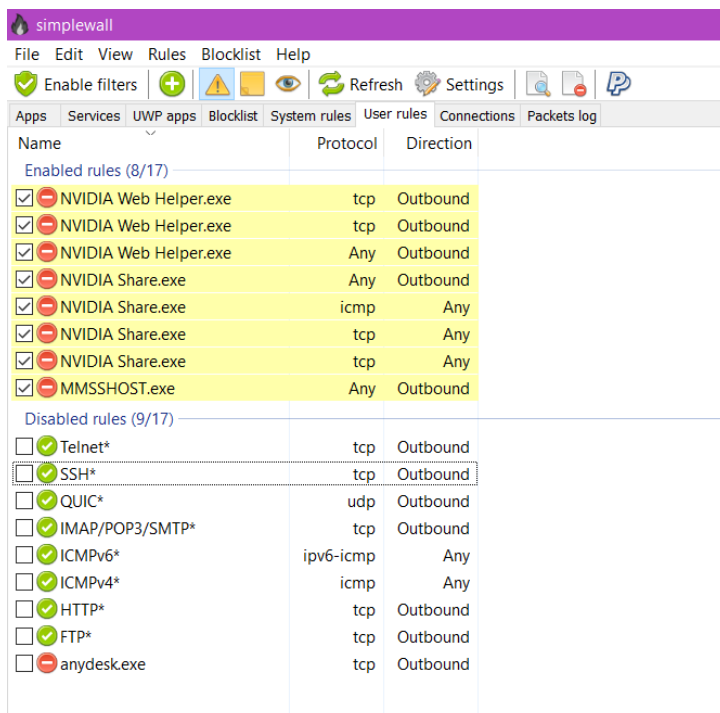
STEP 5: Disable all system and enable only DHCP (for host discovery), DNS (for DNS queries) and IGMP (To check reachability to a server).



The screenshot shows the SimpleWall application window. The 'System rules' tab is selected. The interface displays a list of rules categorized into 'Enabled rules (3/20)' and 'Disabled rules (17/20)'. The 'Enabled rules' list includes DHCP, DNS, and IGMP. The 'Disabled rules' list includes various protocols like IKEv2/IPSEC/L2TP, LLMNR, mDNS, NCSI, NetBIOS, NTP, RDP, SMB, SNMP, SSDP, UPnP, WS-Discovery, and WS-Discovery [events].

Name	Protocol	Direction
Enabled rules (3/20)		
<input checked="" type="checkbox"/> DHCP	udp	Any
<input checked="" type="checkbox"/> DNS	udp	Outbound
<input checked="" type="checkbox"/> IGMP	igmp	Outbound
Disabled rules (17/20)		
<input type="checkbox"/> IKEv2/IPSEC/L2TP	udp	Outbound
<input type="checkbox"/> LLMNR	udp	Outbound
<input type="checkbox"/> mDNS	udp	Outbound
<input type="checkbox"/> NCSI	Any	Outbound
<input type="checkbox"/> NetBIOS [inbound]	Any	Inbound
<input type="checkbox"/> NetBIOS [outbound]	Any	Outbound
<input type="checkbox"/> NTP	udp	Outbound
<input type="checkbox"/> RDP [inbound]	Any	Inbound
<input type="checkbox"/> RDP [outbound]	Any	Outbound
<input type="checkbox"/> SMB [inbound]	tcp	Inbound
<input type="checkbox"/> SMB [outbound]	tcp	Outbound
<input type="checkbox"/> SNMP	udp	Any
<input type="checkbox"/> SSDP [inbound]	udp	Inbound
<input type="checkbox"/> SSDP [outbound]	udp	Outbound
<input type="checkbox"/> UPnP	Any	Outbound
<input type="checkbox"/> WS-Discovery	udp	Outbound
<input type="checkbox"/> WS-Discovery [events]	tcp	Outbound






STEP 6: Create custom rules for some applications and also disable some other User based services.



The screenshot shows the SimpleWall application window with the 'User rules' tab selected. The interface displays a list of rules categorized into 'Enabled rules (8/17)' and 'Disabled rules (9/17)'. The 'Enabled rules' list includes several rules for NVIDIA Web Helper.exe, NVIDIA Share.exe, and MMSSHOST.exe. The 'Disabled rules' list includes Telnet*, SSH*, QUIC*, IMAP/POP3/SMTP*, ICMPv6*, ICMPv4*, HTTP*, FTP*, and anydesk.exe.

Name	Protocol	Direction
Enabled rules (8/17)		
<input checked="" type="checkbox"/> NVIDIA Web Helper.exe	tcp	Outbound
<input checked="" type="checkbox"/> NVIDIA Web Helper.exe	tcp	Outbound
<input checked="" type="checkbox"/> NVIDIA Web Helper.exe	Any	Outbound
<input checked="" type="checkbox"/> NVIDIA Share.exe	Any	Outbound
<input checked="" type="checkbox"/> NVIDIA Share.exe	icmp	Any
<input checked="" type="checkbox"/> NVIDIA Share.exe	tcp	Any
<input checked="" type="checkbox"/> NVIDIA Share.exe	tcp	Any
<input checked="" type="checkbox"/> MMSSHOST.exe	Any	Outbound
Disabled rules (9/17)		
<input type="checkbox"/> Telnet*	tcp	Outbound
<input type="checkbox"/> SSH*	tcp	Outbound
<input type="checkbox"/> QUIC*	udp	Outbound
<input type="checkbox"/> IMAP/POP3/SMTP*	tcp	Outbound
<input type="checkbox"/> ICMPv6*	ipv6-icmp	Any
<input type="checkbox"/> ICMPv4*	icmp	Any
<input type="checkbox"/> HTTP*	tcp	Outbound
<input type="checkbox"/> FTP*	tcp	Outbound
<input type="checkbox"/> anydesk.exe	tcp	Outbound

STEP 7: Open Wireshark to check if any other network activity is present or not. Open TCP LogView for better idea.

TcpLogView						
File Edit View Options Help						
    						
Local Address	Local Port	Remote Address...	Remote Host Name	Remote Port	Process Name	
192.168.116.101	63851	204.79.197.203		443	msedge.exe	
192.168.116.101	49603	204.79.197.203		443	msedge.exe	
192.168.116.101	52765	216.58.210.142		443	msedge.exe	
192.168.116.101	53339	193.229.113.42		443	msedge.exe	
192.168.116.101	65359	13.89.178.26		443	msedge.exe	
192.168.116.101	49391	130.232.246.1		443	msedge.exe	
192.168.116.101	53123	130.232.246.1		443	msedge.exe	
192.168.116.101	49571	130.232.246.1		443	msedge.exe	
192.168.116.101	53123	130.232.246.1		443	msedge.exe	
192.168.116.101	62231	13.89.178.26		443	msedge.exe	
192.168.116.101	63360	130.232.246.1		443	msedge.exe	
192.168.116.101	64425	13.89.178.26		443	msedge.exe	
192.168.116.101	50294	130.232.246.1		443	msedge.exe	
192.168.116.101	61051	130.232.246.1		443	msedge.exe	
192.168.116.101	63743	130.232.246.1		443	msedge.exe	
192.168.116.101	55850	13.89.178.26		443	msedge.exe	
192.168.116.101	62231	13.89.178.26		443	msedge.exe	
192.168.116.101	50061	40.101.50.2		443	WINWORD.EXE	
192.168.116.101	60628	185.86.139.115		443	msedge.exe	
192.168.116.101	53945	52.178.182.73		443	smartscreen.exe	
192.168.116.101	59194	216.58.209.163		443	chrome.exe	
192.168.116.101	53945	52.178.182.73		443	smartscreen.exe	
192.168.116.101	50060	216.58.209.173		443	chrome.exe	
192.168.116.101	53542	216.58.210.164		443	chrome.exe	
192.168.116.101	60707	216.58.209.161		443	chrome.exe	
192.168.116.101	64463	13.33.242.54		443	chrome.exe	
192.168.116.101	53339	193.229.113.42		443	msedge.exe	

4.4 Linux Traffic Analysis

Analysis on Linux is significantly different from Android and Windows OS. Isolating applications on Linux is a difficult task. To overcome this issue, Network Namespace has been used. It is a logical copy of the network stack from the host system possessing its own IP addresses, network interfaces, routing tables etc. Host system physical interfaces exist in global namespace[59]. It allows to run in a separate isolated space with their own memory and network stack. Since Network Namespace in Linux has its own network driver which is binded to active network driver on Linux (Wifi or LAN) so it is easy to capture packets using Wireshark or Tcpdump from Terminal.

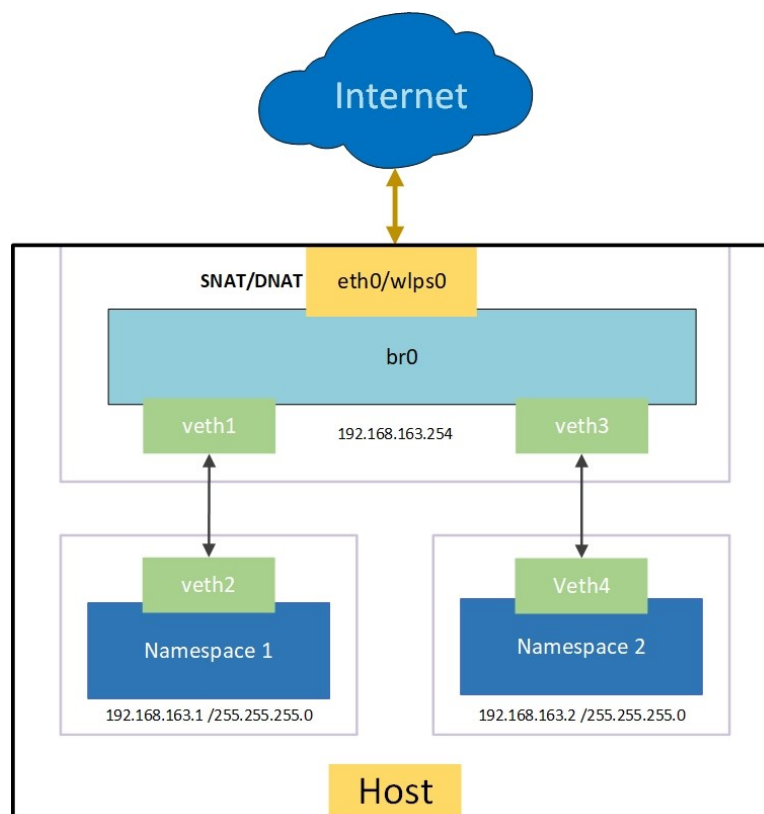


Figure 21 : Working of Network Namespace on Linux

A Linux machine can have one or multiple network namespaces. Network Namespace have their own virtual interface. This interface is connected to another Virtual interface created by Linux host. If there are multiple networks then different virtual interfaces created by the Linux Host form together a bridge. And finally, this bridge or single virtual interface created by Linux host is connected to physical active network interface which can be Wi-Fi or Lan interface. Applications are made to run in the Namespace. There they use the default active network interface which was virtual interface of the namespace. In the namespace a when a particular application was running, it has its own network stack, memory and storage which is different

than the system. Thus, we can establish that application isolation has been achieved. In Figure 21 veth1-veth2 form one “veth” pair, similarly veth3-veth4 form another “veth” pair. It is also important that every veth pair should be in same IP subnet. It is possible that different veth pairs can belong to different IP subnets.

4.5.1 Extracting JA3 from the Capture packets

```

PS M:\Thesis-work\Temp> j3a3 "Small Capture.pcap"
[
  {
    "destination_ip": "204.79.197.203",
    "destination_port": 443,
    "ja3": "771,4865-4866-4867-49195-49199-49196-49200-52393-52392-49171-49172-49173-49174-49175-49176-49177-49178-49179-49180-49181-49182-49183-49184-49185-49186-49187-49188-49189-49190-49191-49192-49193-49194-49195-49196-49197-49198-49199-49200-49201-49202-49203-49204-49205-49206-49207-49208-49209-49210-49211-49212-49213-49214-49215-49216-49217-49218-49219-49220-49221-49222-49223-49224-49225-49226-49227-49228-49229-49230-49231-49232-49233-49234-49235-49236-49237-49238-49239-49240-49241-49242-49243-49244-49245-49246-49247-49248-49249-49250-49251-49252-49253-49254-49255-49256-49257-49258-49259-49260-49261-49262-49263-49264-49265-49266-49267-49268-49269-49270-49271-49272-49273-49274-49275-49276-49277-49278-49279-49280-49281-49282-49283-49284-49285-49286-49287-49288-49289-49290-49291-49292-49293-49294-49295-49296-49297-49298-49299-49300-49301-49302-49303-49304-49305-49306-49307-49308-49309-49310-49311-49312-49313-49314-49315-49316-49317-49318-49319-49320-49321-49322-49323-49324-49325-49326-49327-49328-49329-49330-49331-49332-49333-49334-49335-49336-49337-49338-49339-49340-49341-49342-49343-49344-49345-49346-49347-49348-49349-49350-49351-49352-49353-49354-49355-49356-49357-49358-49359-49360-49361-49362-49363-49364-49365-49366-49367-49368-49369-49370-49371-49372-49373-49374-49375-49376-49377-49378-49379-49380-49381-49382-49383-49384-49385-49386-49387-49388-49389-49390-49391-49392-49393-49394-49395-49396-49397-49398-49399-49400-49401-49402-49403-49404-49405-49406-49407-49408-49409-49410-49411-49412-49413-49414-49415-49416-49417-49418-49419-49420-49421-49422-49423-49424-49425-49426-49427-49428-49429-49430-49431-49432-49433-49434-49435-49436-49437-49438-49439-49440-49441-49442-49443-49444-49445-49446-49447-49448-49449-49450-49451-49452-49453-49454-49455-49456-49457-49458-49459-49460-49461-49462-49463-49464-49465-49466-49467-49468-49469-49470-49471-49472-49473-49474-49475-49476-49477-49478-49479-49480-49481-49482-49483-49484-49485-49486-49487-49488-49489-49490-49491-49492-49493-49494-49495-49496-49497-49498-49499-49500-49501-49502-49503-49504-49505-49506-49507-49508-49509-49510-49511-49512-49513-49514-49515-49516-49517-49518-49519-49520-49521-49522-49523-49524-49525-49526-49527-49528-49529-49530-49531-49532-49533-49534-49535-49536-49537-49538-49539-49540-49541-49542-49543-49544-49545-49546-49547-49548-49549-49550-49551-49552-49553-49554-49555-49556-49557-49558-49559-49560-49561-49562-49563-49564-49565-49566-49567-49568-49569-49570-49571-49572-49573-49574-49575-49576-49577-49578-49579-49580-49581-49582-49583-49584-49585-49586-49587-49588-49589-49590-49591-49592-49593-49594-49595-49596-49597-49598-49599-49600-49601-49602-49603-49604-49605-49606-49607-49608-49609-49610-49611-49612-49613-49614-49615-49616-49617-49618-49619-49620-49621-49622-49623-49624-49625-49626-49627-49628-49629-49630-49631-49632-49633-49634-49635-49636-49637-49638-49639-49640-49641-49642-49643-49644-49645-49646-49647-49648-49649-49650-49651-49652-49653-49654-49655-49656-49657-49658-49659-49660-49661-49662-49663-49664-49665-49666-49667-49668-49669-49670-49671-49672-49673-49674-49675-49676-49677-49678-49679-49680-49681-49682-49683-49684-49685-49686-49687-49688-49689-49690-49691-49692-49693-49694-49695-49696-49697-49698-49699-49700-49701-49702-49703-49704-49705-49706-49707-49708-49709-49710-49711-49712-49713-49714-49715-49716-49717-49718-49719-49720-49721-49722-49723-49724-49725-49726-49727-49728-49729-49730-49731-49732-49733-49734-49735-49736-49737-49738-49739-49740-49741-49742-49743-49744-49745-49746-49747-49748-49749-49750-49751-49752-49753-49754-49755-49756-49757-49758-49759-49760-49761-49762-49763-49764-49765-49766-49767-49768-49769-49770-49771-49772-49773-49774-49775-49776-49777-49778-49779-49780-49781-49782-49783-49784-49785-49786-49787-49788-49789-49790-49791-49792-49793-49794-49795-49796-49797-49798-49799-49800-49801-49802-49803-49804-49805-49806-49807-49808-49809-49810-49811-49812-49813-49814-49815-49816-49817-49818-49819-49820-49821-49822-49823-49824-49825-49826-49827-49828-49829-49830-49831-49832-4
```

Instead of generating results like this, a better approach is to utilize the “<” operator of PS or Terminal. This is called as “stdout” operator. It directs the output of any command to any file. This has been combined with the original command to output the result produced by the command into a file.

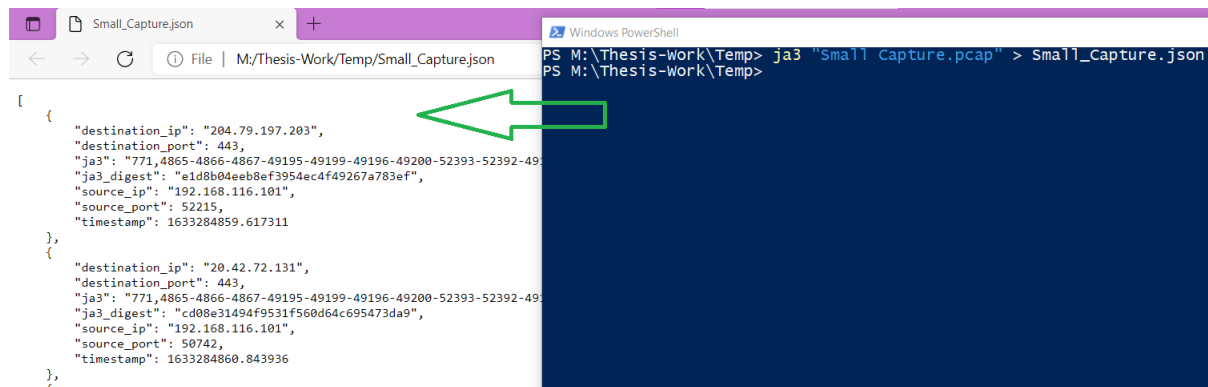


Figure 23 Appending JA3 output to json format

The output can then be stored directly into a file and then can be visualized appropriately. Since Python produces JA3 in a json format so the file can then be opened using a web browser easily. For raw packets, JA3 for python failed and resulted in empty string. So, JA3 plugin for wireshark was used. This plugin displays the JA3 for every Hello Cline packet from that packet capture File. “export as CSV” option for Wireshark was used.

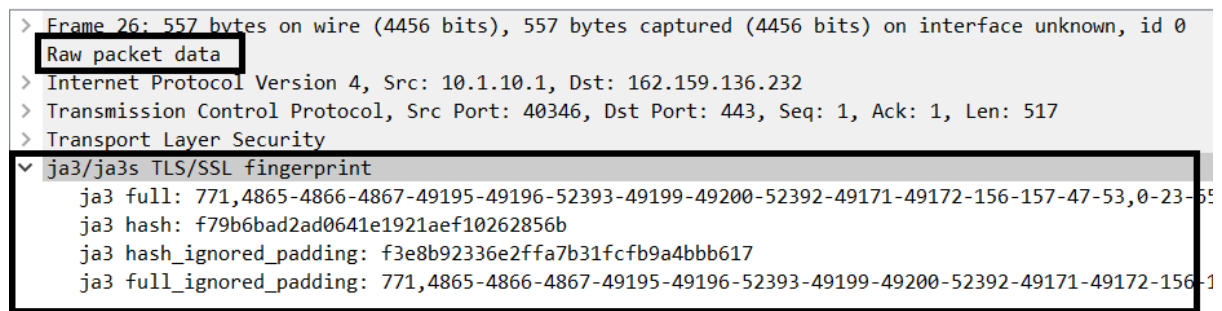


Figure 24 JA3 Plugin in Wireshark

After getting these values from the plugin, this can then be applied as a Column to display for other Hello Packets as well.

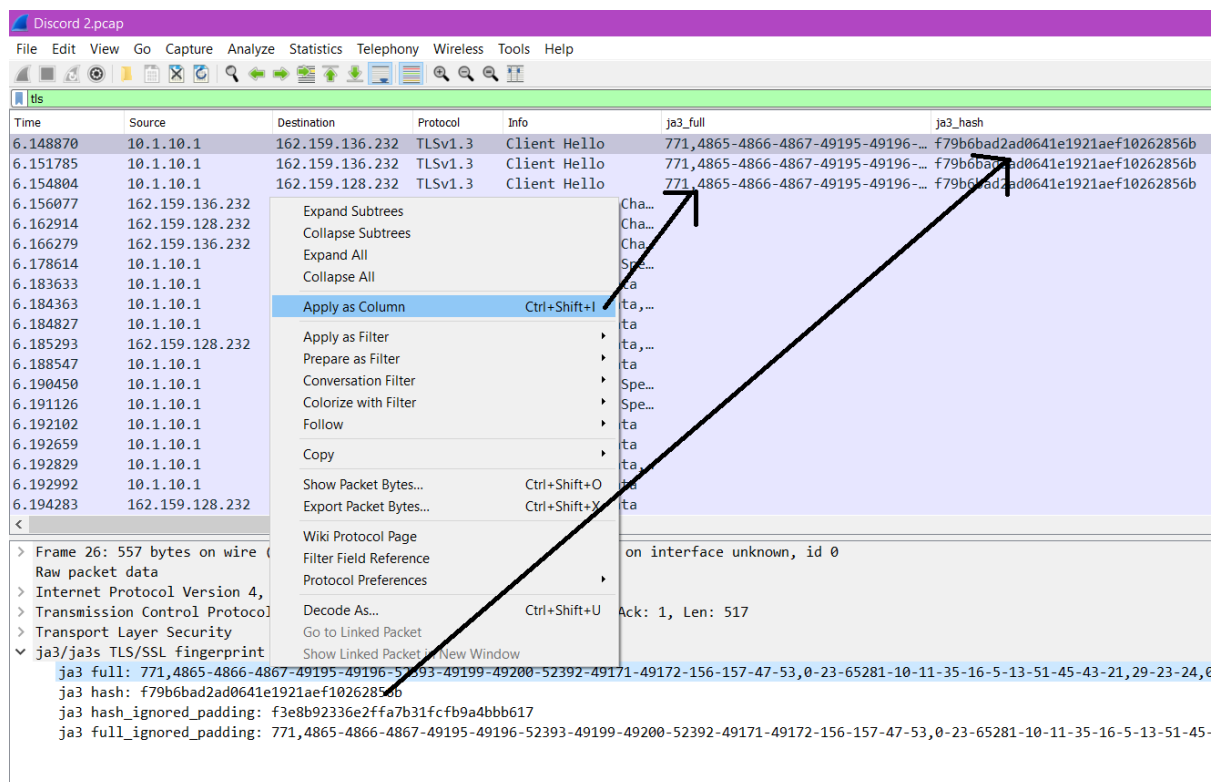


Figure 25 Applying JA3 plugin details as column in Wireshark

Accordingly, JA3 and JA3 hash values will pop up as a column values for all other Hello Packets as well. Finally this can be exported as Packet dissection in the form of CSV.

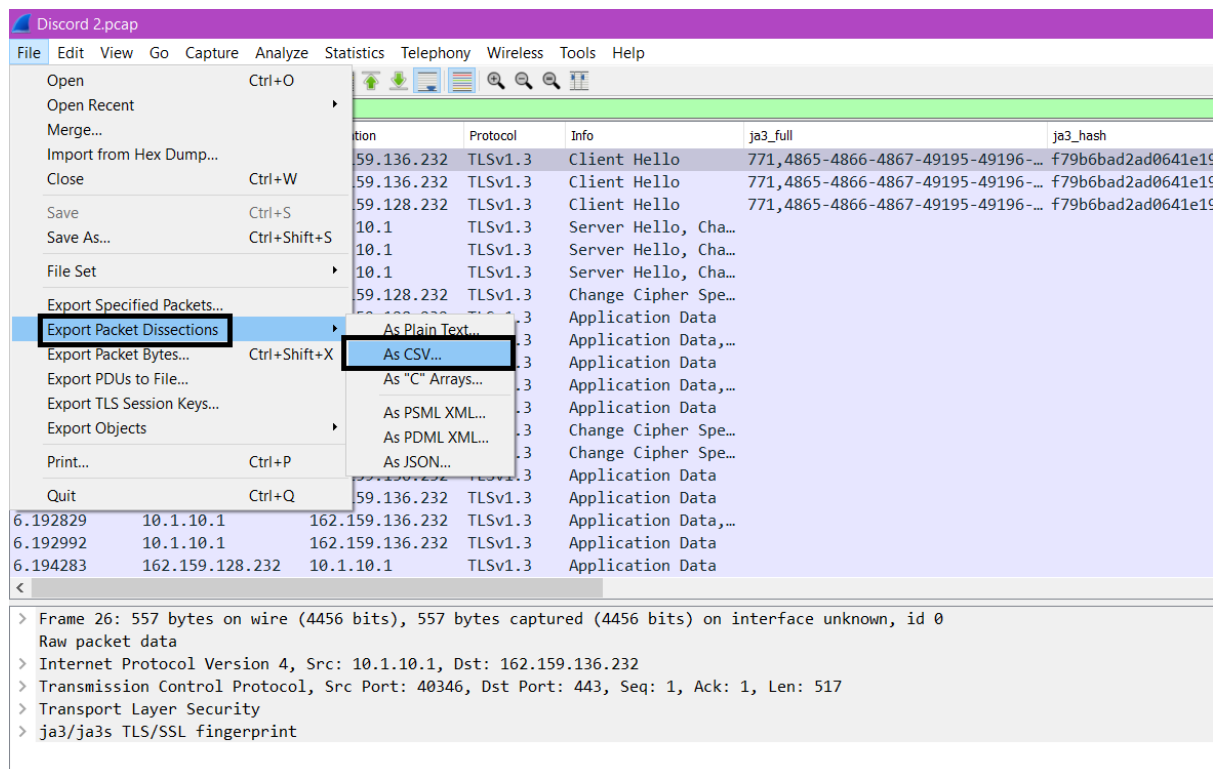


Figure 26 Exporting packets in CSV file from Wireshark

Lastly, the final output can be viewed in the Excel or any other spread sheet viewer.

	A	B	C	D	E	
	Source	Destination	Protocol	Info	ja3_hash	ja3_full
1	10.1.10.1	162.159.136.232	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
2	10.1.10.1	162.159.136.232	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
3	10.1.10.1	162.159.128.232	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
4	10.1.10.1	162.159.136.234	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
5	10.1.10.1	162.159.138.234	TLSv1.2	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
6	10.1.10.1	162.159.128.235	TLSv1.2	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
7	10.1.10.1	162.159.134.232	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
8	10.1.10.1	162.159.137.232	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
9	10.1.10.1	162.159.133.234	TLSv1.3	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
10	10.1.10.1	162.159.130.235	TLSv1.2	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
11	10.1.10.1	162.159.128.235	TLSv1.2	Client Hello	f79b6bad2ad0641e1921aef10262856b	771,4865-4866-4867-49195-49196-52393-
12	10.1.10.1	162.159.134.232	TLSv1.3	Client Hello	c67e9dc27d283f1f89b4ebb4b4670c21	771,4865-4866-4867-49195-49196-52393-
13	10.1.10.1	162.159.135.233	TLSv1.3	Client Hello	c67e9dc27d283f1f89b4ebb4b4670c21	771,4865-4866-4867-49195-49196-52393-
14	10.1.10.1	162.159.134.232	TLSv1.3	Client Hello	c67e9dc27d283f1f89b4ebb4b4670c21	771,4865-4866-4867-49195-49196-52393-
15	10.1.10.1	162.159.134.232	TLSv1.3	Client Hello	c67e9dc27d283f1f89b4ebb4b4670c21	771,4865-4866-4867-49195-49196-52393-
16	10.1.10.1	162.159.134.232	TLSv1.3	Client Hello	c67e9dc27d283f1f89b4ebb4b4670c21	771,4865-4866-4867-49195-49196-52393-

Figure 27 Viewing packets dissection exported from Wireshark in CSV file

4.5.2 Getting all Unique JA3 from the datasets

After the datasets have been obtained from the various captures either in the CSV or JSON format, they can be imported into python. Inside python, they can be imported as “pandas dataframe” so that data analysis operation can be performed on them. For Android and Linux, in case of Discord, the values are imported as shown:-

```

▼ Importing required Libraries

[1] import pandas as pd
import numpy as np

▼ Android :-

[2] data_android1 = pd.read_csv("/content/drive/MyDrive/Android Capture 21-09-21/Discord1.csv")
data_android2 = pd.read_csv("/content/drive/MyDrive/Android Capture 21-09-21/Discord2.csv")
frames = [data_android1, data_android2]
data_android = pd.concat(frames)
Android = data_android[["ja3_hash", "ja3_full"]]
Android = Android.drop_duplicates(subset = ["ja3_hash"])
Android = Android.dropna()
Android_Hash = Android['ja3_hash'].to_list()
Android_String = Android['ja3_full'].to_list()
data_android['ja3_hash'].value_counts()

c67e9dc27d283f1f89b4ebb4b4670c21    45
9b02ebd3a43b62d825e1ac605b621dc8    43
f79b6bad2ad0641e1921aef10262856b    15
84cd49a759d15947552e61b46fa5f2ec     9
Name: ja3_hash, dtype: int64

```

```

Linux :-

[ ] data_linux = pd.read_json("/content/drive/MyDrive/Linux Capture 21-09-21/Discord.json",encoding='utf-16')
Linux = data_linux[["ja3_digest","ja3"]]
Linux = Linux.drop_duplicates(subset = ["ja3_digest"])
Linux = Linux.dropna()
Linux_Hash = Linux['ja3_digest'].to_list()
Linux_String = Linux['ja3'].to_list()
data_linux['ja3_digest'].value_counts()

7094b0d5b83dec3f4072bbf49e4b480a    5
27c1c07256b84edb63e0fba28778d5a1    1
Name: ja3_digest, dtype: int64

```

Figure 28 Extracting unique JA3 using pandas

The data was imported differently in case for CSV and JSON format but the steps to refine and produce the results were similar. First the dataset was imported as a “dataframe” variable. Then only the interested columns can be exported to another variable. Since we are only concerned with the JA3 string and JA3 hash, only these have been imported. Now from the imported columns, duplicate values were removed and empty rows were removed so that only unique (not Null) data sets are available. These are then further stored in the different variables respectively. Figure represents the unique data obtained from the dataset along with the count of that data. So JA3 strings extracted from different datasets can be merged together and exported together in any form like txt or spreadsheet format.

Google-Collaboratory makes work very simpler because it can think of like an e-notebook with python code and output displayed, so it can be shared with everyone and anyone. So, it helps to visualize the output properly with ease and complexity. Also, it works on Python with Linux kernel so it gives power of both. We can easily run Linux and complex Python commands easily. Secondly it can also read the data from the google-drive very easily. So once the data is uploaded to the Google-Drive, it can then be directly accessed inside the Google-Collaboratory without any hassle. Third and most importantly it can provide CPU or GPU computations as well so that speeds up the work very well. And lastly it is “Free to use” so anyone with a simple google account can go and use it.

5 Results and Inferences

The results obtained from the JA3 obtained from various applications running on Linux, Windows and Android are presented in the following sections.

5.1 Applications in consideration

Different applications were considered for the analysis. It was important to note that only those applications were considered that are available for all three operating systems : Android, Linux and Windows. Applications considered are as follows.

- | | |
|------------------|----------------|
| 1. Anydesk | 8. Spotify |
| 2. Google Chrome | 9. Steam |
| 3. Discord | 10. Teams |
| 4. Dropbox | 11. Teamviewer |
| 5. Edge | 12. Telegram |
| 6. Firefox | 13. Zoom |
| 7. Skype | |

The applications selected for analysis as indicated above are commonly used in office environments. From the perspective of use, these applications are most popular in their category and hence have been selected for analysis.

5.2 Results from Android¹

The following observations can be made from the results. 42 unique hashes have been obtained from the applications selected for analysis. Many applications generated more than three hashes while Anydesk, Teamviewer and Telegram generated one or no hash. Skype in total generated 11 unique hashes, Chrome generated 9 while Edge generated 8 unique hashes respectively. Some hashes are found to be common across applications. The hash “9b02ebd3a43b62d825e1ac605b621dc8” is found to be generated by all the applications except Anydesk, Steam and Zoom. Also the hash “cd08e31494f9531f560d64c695473da9” is found to be generated by five applications.

Skype and Teams generated some hashes which were common among them. Since these applications have been developed by Microsoft in the first place, it can be inferred that

¹ The full results are listed in Appendix 3 due to their large size.

Microsoft might have used the same TLS library for generating hashes which are subsequently used by Skype and Teams. It was a running hypothesis that Chrome, Edge and Firefox may generate common Ja3 hashes but the results reject the notion. Out of all the hashes produced by Chrome(9), Edge(8) and Firefox(5), only one hash is found to be common i.e “9b02ebd3a43b62d825e1ac605b621dc8”.

5.3 Results from Linux²

For Linux, 38 unique Hashes were obtained for all the applications. Out of the set of applications analysed, Firefox produced maximum of 10 Unique hashes while Edge produced 9 and Chrome produced 8 unique hashes. Anydesk and TeamViewer didn't produce any hashes while surprisingly Zoom and Discord produced only 2 hashes. The hash “7094b0d5b83dec3f4072bbf49e4b480a” has been found to be in common with at in five applications - Chrome, Discord, Skype, Spotify and Teams. Out of all the unique hashes, 26 hashes have been uniquely produced by one application and are not common between any of the applications. Skype and Teams have yielded common hashes. Chrome and Edge have two common hashes while Firefox didn't have any common hash with either of them or with other applications as well.

5.4 Results from Windows³

Analysis of windows operating system reveal 37 unique hashes in total. Out of all the applications, Edge generated 8 unique hashes, Discord and Dropbox produced 3, Steam generated 4 hashes. Anydesk and Telegram didn't produce any hash while TeamViewer produced just one hash. The hash “74ad8ec6876e2e3366bfd566581ca7e8” was shared by maximum of 4 applications: Chrome, Discord, Edge, Spotify.

Out of all the hashes, 23 hashes have been uniquely generated by the applications and are not common with any other application. Skype has generated 4 and Teams generated 6 hashes. Both of these applications have just one hash in common. This implies that Microsoft probably uses different TLS libraries to implement these Software. Chrome and Edge have almost generated same hashes but some hashes were uncommon. On the other hand, Firefox didn't generate any hash which is common with either of two.

² The full results are listed in Appendix 1 due to their large size.

³ The full results are listed in Appendix 2 due to their large size.

5.5 Application Wise Comparison on Android, Linux and Windows

A comparison of the results obtained from the applications running on the three selected operating systems is presented as follows.

1. Anydesk.

JA3	Android	Linux	Windows
29b5a018fa5992fe23560c16af0dc9fc	X		
201999283915cc31cee6b15472ef3332		X	
3f2fba0262b1a22b739126dfb2fe7a7d			X

Anydesk only generated unique hashes for Android, Linux and Windows. There is no overlap between any operating system.

2. Google chrome

JA3	Android	Linux	Windows
7283d4318a3295c0b7d74a7eb02ce29f		X	
0d69ff451640d67ee8b5122752834766	X		X
e1d8b04eeb8ef3954ec4f49267a783ef	X		X
74ad8ec6876e2e3366bfd566581ca7e8	X		X
8979ea6385c2505e8c6d4086a8a39be3	X		X
598872011444709307b861ae817a4b60	X		X
9b02ebd3a43b62d825e1ac605b621dc8	X		
a6d03565735ac2f6f810b11d037e08ed		X	
2eacfe5e29a49ebf4e7875340c67b41d	X		
ee7181db614cb63ec32c96863ead1b1c			X
cd08e31494f9531f560d64c695473da9	X		X
3d90a696919abaf92ba5e076e4421a6a	X		
7094b0d5b83dec3f4072bbf49e4b480a		X	
c9539121a34c8104133553aa3b211b61		X	
cb2a20fe5f35977440057f8bd1980fba		X	
27c1c07256b84edb63e0fba28778d5a1		X	
283e90b31bad9ee68a89ea1ad528e168		X	
ab890ae7a7163df667ef665287073a15		X	

Chrome generated in all 13 Unique Hashes for all three operating systems. Android and Windows have a lot in common hashes. While for Linux there is no overlap with Android and Windows. This implies that Chrome uses same TLS libraries for Android and Windows, but different Library for Linux.

3. Discord

JA3	Android	Linux	Windows
84cd49a759d15947552e61b46fa5f2ec	X		
74ad8ec6876e2e3366bfd566581ca7e8			X

c67e9dc27d283f1f89b4ebb4b4670c21	X		
7094b0d5b83dec3f4072bbf49e4b480a		X	
f79b6bad2ad0641e1921aef10262856b	X		
cd08e31494f9531f560d64c695473da9			X
9b02ebd3a43b62d825e1ac605b621dc8	X		
3b5074b1b5d032e5620f69f9f700ff0e			X
27c1c07256b84edb63e0fba28778d5a1		X	

For Discord, 9 unique hashes have been generated in total. There is no overlap between any operating system for Discord. It can be inferred that Discord uses entirely different TLS Libraries for Android, Linux and Windows. 4 hashes were produced in Android while only 2 were seen in Linux, Windows had 3 hashes.

4. Dropbox

JA3	Android	Linux	Windows
9b02ebd3a43b62d825e1ac605b621dc8	X		
66918128f1b9b03303d77c6f2eefd128			X
533b47a1acfb9b244447765a2ec2a4d	X		
59fe159ce36975a15410412d25c2e114	X		
a0e9f5d64349fb13191bc781f81f42e1			X
effe9b59e99e730d14f23d971080682b			X
f79b6bad2ad0641e1921aef10262856b	X		
b075db672378ecc07a6e964ebf923de6		X	

Dropbox produced total 8 unique hashes in aggregate. There is no overlap between any operating system for Dropbox. So, similarly Discord, Dropbox as well uses entirely different TLS Libraries for Android, Linux and Windows. Only one hash was seen in Linux.

5. Edge

JA3	Android	Linux	Windows
598872011444709307b861ae817a4b60			X
fd0c0a519648e519f40f7fed962076fc		X	
26e28949bdfc88c8769c99b2f20ad18d		X	
fedca33016b974c390faa610378b5a62		X	
66918128f1b9b03303d77c6f2eefd128	X		
cd08e31494f9531f560d64c695473da9			X
554719594ba90b02ae410c297c6e50ad	X		
3f27d27a5f1348476cac6ec873e505cf	X		

27c1c07256b84edb63e0fba28778d5a1		X	
cb2a20fe5f35977440057f8bd1980fba		X	
8f41a697eff27e008f969cf7b5ba4117	X		
663d4f09c017c69df656f4372b7ee7ca		X	
0de4f16d9e23b224ed22c7eda958cede			X
e1d8b04eeb8ef3954ec4f49267a783ef			X
8979ea6385c2505e8c6d4086a8a39be3			X
74ad8ec6876e2e3366bfd566581ca7e8			X
1b73862eae8f1711440a446b1ef357fd		X	
eb725168365ef8cf4aabc309ce94b9e	X		
0d69ff451640d67ee8b5122752834766			X
0ae18052c288c1bd39910255598ed827	X		
aa50c12a5dfa717d9d6ab34e97de79d5		X	
9b02ebd3a43b62d825e1ac605b621dc8	X		
a0e9f5d64349fb13191bc781f81f42e1			X
f58f4c92d50fe2785d35d6e2eb8756f2	X		
dd6aa87fa4b2b05a72ab994663ad94c5		X	

Edge generated 25 unique hashes in total. There is no overlap between any operating system for Edge. Equal number of hashes have been observed in Android, Linux and Windows

6. Firefox

JA3	Android	Linux	Windows
aa7744226c695c0b2e440419848cf700			X
3cf05a4527fe8c00f228443524e3d76e			X
7d2fafa1dbc311cf6b2ed9ea1580bf7f		X	
029f230ecf557eb57747b5debc47512c			X
f14ec85ee5580a29f6523e24e5d3d527			X
df208241e7f3897d4ca38cfe68eabb21			X
c834494f5948ae026d160656c93c8871	X		
6b5e0cfe988c723ee71faf54f8460684	X	X	
0b18afc3387a48ed8e75634f89b9bd07		X	
5942576315948177ddbdb7f97eecbc23		X	
ff0f32710f49eabdc2d802d0be5b5695		X	
9b02ebd3a43b62d825e1ac605b621dc8	X		
4d0dbc200ffee1085949cea3410a5547		X	
a72f351cf3c3cd1edb345f7dc071d813		X	X
a0262d81f08838bbb1877a10e3fd70f1	X	X	
766543824ed9f5d5adeec04a50c21904		X	
c67e9dc27d283f1f89b4ebb4b4670c21	X		
1af5d1fe5c1c9bdba4ec723ac5cab44f		X	
af99c5134dc32b4f2e289e01ac7fe8b1			X

Total 19 unique hashes were generated by Firefox. Only 2 overlap are visualised in Android and Linux while there was no overlap with Windows. It can be derived that Edge might use same TLS libraries for Linux and Android but altogether different Library in Windows.

7. Skype

JA3	Android	Linux	Windows
f79b6bad2ad0641e1921aef10262856b	X		
dda262729e5413660ec0e6a8d4279860			X
eaabed81520b23ea8a800b36bd7e359e	X		
cd08e31494f9531f560d64c695473da9	X		
8979ea6385c2505e8c6d4086a8a39be3	X		
ca450f8afb5c4c9c678d1c667c7482f2		X	
7f805430de1e7d98b1de033adb58cf46			X
7094b0d5b83dec3f4072bbf49e4b480a		X	
20dff5d81f228563c0318b235baafae9	X	X	
a1c672bda2bda1a05bdca801144b2760	X		
3b5074b1b5d032e5620f69f9f700ff0e			X
b32309a26951912be7dba376398abc3b			X
7d52c9129b8b07502d1471697c2982dd		X	
3fc9b7563d8f20616129ddf848a5fdde	X		
6ec2896feff5746955f700c0023f5804	X		
9b02ebd3a43b62d825e1ac605b621dc8	X		
e1d8b04eeb8ef3954ec4f49267a783ef	X		
3b92fc00fff571f3cd2ad12f32856bac	X		

Skype produced 18 unique hashes. Android generated a large number of hashes while Windows generated only 4. There is no overlap between any operating systems.

8. Spotify

JA3	Android	Linux	Windows
b32309a26951912be7dba376398abc3b			X
9b02ebd3a43b62d825e1ac605b621dc8	X		
27c1c07256b84edb63e0fba28778d5a1		X	
7094b0d5b83dec3f4072bbf49e4b480a		X	
74ad8ec6876e2e3366bfd566581ca7e8			X
ee7181db614cb63ec32c96863ead1b1c			X
f79b6bad2ad0641e1921aef10262856b	X		
af0b643ba6479160d5fdb42c09c32b9a	X	X	X
c67e9dc27d283f1f89b4ebb4b4670c21	X		
7f805430de1e7d98b1de033adb58cf46			X

cd08e31494f9531f560d64c695473da9	X		
a0e9f5d64349fb13191bc781f81f42e1			X
d75e7289c86c15b305ac36097bfa0487	X		

Spotify generated 13 unique hashes in all. Surprisingly there is one hash which is common for Android, Linux and Windows - “af0b643ba6479160d5fdb42c09c32b9a”. Except the common hash, there is no overlap between any operating system.

9. Steam

JA3	Android	Linux	Windows
598872011444709307b861ae817a4b60	X		
cd08e31494f9531f560d64c695473da9	X		
e9a3c539cf07e9734e684bddd67e3847		X	
e5b607b5862a46cab44d7bacd582b3cd		X	
66918128f1b9b03303d77c6f2eefd128			X
554719594ba90b02ae410c297c6e50ad			X
a2de60d8d8cd26f3f1131952a5adcba7			X
ef5b9ad415172970f6e8ee7c87745c14			X
ee26b1f1aec16d6098768e2c67388ace	X		
3fc9b7563d8f20616129ddf848a5fdde	X		
d5a82b7e86091b9fc3bd24e2c9ae2919		X	
bbcc15c33ab7b9c6e467584d9aeb3020		X	
6ec2896feff5746955f700c0023f5804	X		

Steam generated 13 unique hashes. There is no overlap between any operating systems.

10. Teams

JA3	Android	Linux	Windows
a6d03565735ac2f6f810b11d037e08ed		X	
554719594ba90b02ae410c297c6e50ad			X
7e8a75b0be593e6c7eb5992dd1de084d	X		
98d82c43de4e22a055da96bceflca9b9		X	
f3e8b92336e2ffa7b31fcfb9a4bbb617	X		
28a2c9bd18a11de089ef85a160da29e4			X
54328bd36c14bd82ddaa0c04b25ed9ad			X
66918128f1b9b03303d77c6f2eefd128			X
f79b6bad2ad0641e1921aef10262856b	X		
7283d4318a3295c0b7d74a7eb02ce29f		X	
7094b0d5b83dec3f4072bbf49e4b480a		X	
e5b607b5862a46cab44d7bacd582b3cd		X	
080027e640f8ec6d966f79fc2f7ca551			X
cd08e31494f9531f560d64c695473da9	X		
20dff5d81f228563c0318b235baafae9	X	X	
9b02ebd3a43b62d825e1ac605b621dc8	X		

0acf40799ecbd83506d6caeb38514671	X		
7d52c9129b8b07502d1471697c2982dd		X	
3b5074b1b5d032e5620f69f9f700ff0e			X

Teams generated 19 Unique hashes. Only one overlap has been observed in Android and Linux.

11. TeamViewer

JA3	Android	Linux	Windows
9b02ebd3a43b62d825e1ac605b621dc8	X		
74954a0c86284d0d6e1c4efefe92b521			X

Only 2 hashes were generated for TeamViewer. There was no overlap for Android and Windows. Linux didn't generate any hash.

12. Telegram

Telegram didn't generate any hash on Android, Linux or Windows.

13. Zoom

JA3	Android	Linux	Windows
cd08e31494f9531f560d64c695473da9	X		
10492f76a2b8f67ac1672f76bbf7aa0e	X		
672d76590b3edaa92dfbcab7c963ed12	X		
76e38c7e3a1f21d42c77d3e3db538c8e			X
10f1ebcdc5bf282bdf4aec39fb4cfc8b	X		X
50a7456e0d02951e510e4ace60fa3313			X
832952db10f1453442636675bed2702b			X
3fc9b7563d8f20616129ddf848a5fdde	X		
48d7add7fc9c54f66ec4ab9165f46e4b	X	X	X
259dd0f97c6df37f3c55a730eb0d9238		X	X

Zoom generated 10 unique hashes in all. There is one overlap on all the operating systems "48d7add7fc9c54f66ec4ab9165f46e4b". There was also one overlap observed for Linux and Windows "259dd0f97c6df37f3c55a730eb0d9238".

5.6 Inferences

Analysis of the final result for all the application(Appendix 4) is follows. Total of 99 Hashes were observed for all the application for all the operating systems. Edge produced highest number of hashes: 25. Following Firefox and Teams produced 19 hashes, Skype and Chrome produced 18 hashes. Teamviewer(2) and Telegram(0) were among the applications with least number of hashes. “9b02ebd3a43b62d825e1ac605b621dc8” was produced by most of the applications about 9 out of 13 applications. “cd08e31494f9531f560d64c695473da9” was produced by 8 applications. Some overlap is observed in Skype and Spotify, Skype and Teams. There is some overlap between Chrome and Edge while there is hardly any overlap between Firefox and Edge, Firefox and Chrome except for “9b02ebd3a43b62d825e1ac605b621dc8”.

Some of the hashes for **Android** like :-

- cd08e31494f9531f560d64c695473da9
- 598872011444709307b861ae817a4b60
- e1d8b04eeb8ef3954ec4f49267a783ef
- 0ae18052c288c1bd39910255598ed827
- c67e9dc27d283f1f89b4ebb4b4670c21
- 48d7add7fc9c54f66ec4ab9165f46e4b
- 10f1ebcdc5bf282bdf4aec39fb4cfc8b
- 20dff5d81f228563c0318b235baafae9
- a0262d81f08838bbb1877a10e3fd70f1
- 3fc9b7563d8f20616129ddf848a5fdde
- 533b47a1acfb9b244447765a2ec2a4d
- 59fe159ce36975a15410412d25c2e114
- 8979ea6385c2505e8c6d4086a8a39be3
- a1c672bda2bda1a05bdca801144b2760
- 29b5a018fa5992fe23560c16af0dc9fc
- ee26b1f1aec16d6098768e2c67388ace
- 672d76590b3edaa92dfbcab7c963ed12
- 10492f76a2b8f67ac1672f76bbf7aa0e
- 3f27d27a5f1348476cac6ec873e505cf
- af0b643ba6479160d5fdb42c09c32b9a
- 2eacfe5e29a49ebf4e7875340c67b41d

- 7e8a75b0be593e6c7eb5992dd1de084d
- 3b92fc00fff571f3cd2ad12f32856bac

were generated in the result-set obtained from the analysis, but they were not present in the JA3er.com database.

The hash “cd08e31494f9531f560d64c695473da9” was produced in highest frequency, it was generated by Chrome, Skype, Spotify, Steam, Teams and Zoom but no information has been reported by anyone on the ja3er.com’s database. On further analysis, it was found out that on the website “joesandbox.com”, this hash is also generated by a malware known as “Wave Browser_6xz4wdjd_.exe” on the Windows Operating system. Analysis comments says “hijacking vulnerabilities found, Creates COM task schedule object, Stores files to the Windows start menu directory, evasive API chain checking for user administrative privileges”. The hash “66918128f1b9b03303d77c6f2eefd128” when searched in the online Ja3er’s database, reported that this was used as in “Brute Force Attack” and also used in “Poison Attack”. This hash was only generated by “Edge” in the Android results.

The hash “8f41a697eff27e008f969cf7b5ba4117” was also produced in big frequency as well. The JA3er’s database says that this hash is also produced by some website “winworkers.club”. On further analysis, this hash is also generated by another Android application “LanguageLine InSight_v2.0.1_apkpure.com.apk”. It falls in the “Suspicious” behavior category. The comments were “Delete Device Data, Device Lockout, Manipulate App Store Rankings or Ratings”

The hash “d75e7289c86c15b305ac36097bfa0487” has been reported as “wsTls/1.2” in the Ja3er database. Further analysis, ws-tls is used in RPC Server that uses TLS as well.

Just like Android dataset comparison, some of the **Linux** hashes like :-

- aa50c12a5dfa717d9d6ab34e97de79d5
- 4d0dbc200ffee1085949cea3410a5547
- 1af5d1fe5c1c9bdba4ec723ac5cab44f
- a0262d81f08838bbb1877a10e3fd70f1
- 1b73862eae8f1711440a446b1ef357fd
- dd6aa87fa4b2b05a72ab994663ad94c5
- 0b18afc3387a48ed8e75634f89b9bd07
- 7d2fafa1dbc311cf6b2ed9ea1580bf7f

- e9a3c539cf07e9734e684bddd67e3847
- fedca33016b974c390faa610378b5a62
- 27c1c07256b84edb63e0fba28778d5a1
- 20dff5d81f228563c0318b235baafae9
- af0b643ba6479160d5fdb42c09c32b9a
- bbcc15c33ab7b9c6e467584d9aeb3020
- 663d4f09c017c69df656f4372b7ee7ca
- b075db672378eec07a6e964ebf923de6
- 48d7add7fc9c54f66ec4ab9165f46e4b
- d5a82b7e86091b9fc3bd24e2c9ae2919
- 98d82c43de4e22a055da96bcef1ca9b9
- 766543824ed9f5d5adeec04a50c21904
- cb2a20fe5f35977440057f8bd1980fba
- 283e90b31bad9ee68a89ea1ad528e168
- 26e28949bdfc88c8769c99b2f20ad18d
- 5942576315948177ddbdb7f97eebc23
- ab890ae7a7163df667ef665287073a15
- 201999283915cc31cee6b15472ef3332
- ff0f32710f49eabdc2d802d0be5b5695

were not present in the JA3er.com database but generated in the Linux analysis.

The hash “e5b607b5862a46cab44d7bacd582b3cd” was also produced (around 70 times). But from the JA3er’s Database, one of the remarks says that “yaroslavl” which doesn't make any sense because it is name of a Russian city.

The hash “259dd0f97c6df37f3c55a730eb0d9238” has remarks attached which says “Possible Zoom Client”. Surprisingly from our analysis it turned to be actual “Zoom” Client.

The hash “201999283915cc31cee6b15472ef3332” was generated only once by Anydesk and that too on Linux. This hash was not found on the JA3er’s database. By further analysis, one website (How to Spot Unsafe Communications using nDPI Flow Risk Score – ntop) says that this hash belongs to ANYDESK service, and it has FLOW Score Total of 80, which means the session made by the ANYDESK application at that time was unsafe because it has high FLOW Score Value.

Also some of the hashes from applications in Windows like :-

- cd08e31494f9531f560d64c695473da9
- 598872011444709307b861ae817a4b60
- e1d8b04eeb8ef3954ec4f49267a783ef
- f14ec85ee5580a29f6523e24e5d3d527
- 0d69ff451640d67ee8b5122752834766
- 50a7456e0d02951e510e4ace60fa3313
- 8979ea6385c2505e8c6d4086a8a39be3
- effe9b59e99e730d14f23d971080682b
- 832952db10f1453442636675bed2702b
- af0b643ba6479160d5fdb42c09c32b9a
- 76e38c7e3a1f21d42c77d3e3db538c8e
- ee7181db614cb63ec32c96863ead1b1c
- 0de4f16d9e23b224ed22c7eda958cede
- ef5b9ad415172970f6e8ee7c87745c14

were not present in the JA3er.com database but generated as well.

The hash “029f230ecf557eb57747b5debc47512c” was produced (around 6200 times). But from the JA3er’s Database, one of the remarks says it belongs to User Agent : “X”. It even has a count of 3809 which was surprising. The hash “af99c5134dc32b4f2e289e01ac7fe8b1” has User Agent tagged as : “attack using Brute force”, but was generated by Firefox.

The hash “3b5074b1b5d032e5620f69f9f700ff0e” was generated by Discord on windows. But on the JA3er’s database, it has been tagged with User Agents like : “Poison attack” or “Suspect some firewall-appliance that opens TLS Browser”

The hash “a2de60d8d8cd26f3f1131952a5adcba7” has been generated by Steam in the Windows. This one is correctly classified by User Agent as “Valve/Steam HTTP Client 1.0” in the JA3er’s database.

The hash “74954a0c86284d0d6e1c4efefe92b521” is generated by Teamviewer in Windows with count of 24. From the JA3er’s database, it was tagged with User Agents like : “Fiddler Everywhere”, “Curl” and “Retard”.

The hash “3f2fba0262b1a22b739126dfb2fe7a7d” was generated by Anydesk in Windows. Also from the JA3er’s Database, it was tagged with User Agents : “AnyDesk Relay”, “HTTPS/80 Anydesk”, “Anydesk Relay”. This means that ANYdesk was correctly classified although its count in the database was only 1.

6 Conclusion

The main objective of this thesis was to analyse the TLS packets of various applications using JA3 method. It was found out that many applications use same TLS libraries which results in same JA3 hashes, while some applications use different libraries even on different Operating systems. Since, web browsers generate more hashes per operating system it is necessary to consider other famous browsers as well like Opera or Safari. Since Apple devices are also used widely so, the “macOS” on Apple PC and “Ios” on Apple Smartphones should also be considered in future research work.

6.1 Challenges and Limitations

The research study has encountered various challenges and limitations during the data collection and analysis. These are summarised as follows.

1. The first challenge was to isolate the network activity of every application on every Operating System
2. For the Android, we used the target device as One Plus 7 with Android 11. To capture the network data of every application, it has to be manually stopped from the application settings before it is relaunched after the VPN service is started.
3. Sometimes the raw packets are captured instead of proper normal packets. So, the JA3 on the python cannot parse the pcap file which has packets in raw format as it gives error “Improper TCPDUMP header”.

```
> Frame 21: 557 bytes on wire (4456 bits), 557 bytes captured (4456 bits) on interface unknown, id 0
Raw packet data
> Internet Protocol Version 4, Src: 10.1.10.1, Dst: 162.159.133.232
> Transmission Control Protocol, Src Port: 42454, Dst Port: 443, Seq: 1, Ack: 1, Len: 517
> Transport Layer Security
▼ ja3/ja3s TLS/SSL fingerprint
  ja3 full: 771,4865-4866-4867-49195-49196-52393-49199-49200-52392-49171-49172-156-157-47-53,0-23-
  ja3 hash: f79b6bad2ad0641e1921aef10262856b
  ja3 hash_ignored_padding: f3e8b92336e2ffa7b31fcfb9a4bbb617
  ja3 full_ignored_padding: 771,4865-4866-4867-49195-49196-52393-49199-49200-52392-49171-49172-156

0000 45 00 02 2d b9 1b 40 00 40 06 43 26 0a 01 0a 01  E----@. @.C&---
0010 a2 9f 85 e8 a5 d6 01 bb 9f 53 23 1b 4d 85 5d 9c  -.....S#.M.-]
0020 50 18 00 9c 50 b9 00 00 16 03 01 02 00 01 00 01  P...P...
0030 fc 03 03 aa 93 f9 ad 57 3f 4c 26 39 a6 9f 10 48  -...W ?L&9---H
0040 1b 16 a4 7d 5e e0 6a e5 20 b1 22 25 39 08 6b bc  -...}^j.  "%9.k
0050 e4 b1 d7 20 54 bd 5c db 0d 88 de 4d 79 3f 2d ee  -..T.\. ...My?--
0060 ae 43 49 5a ac 6b 11 8a 04 26 35 3b 42 e4 a2 4f  -CIZ.k.  &5;B--0
0070 19 03 94 06 00 1e 13 01 13 02 13 03 c0 2b c0 2c  -.....+.,
0080 cc a9 c0 2f c0 30 cc a8 c0 13 c0 14 00 9c 00 9d  -.../-0. ....
0090 00 2f 00 35 01 00 01 95 00 00 00 16 00 14 00 00  -/-5....
00a0 11 64 6c 2e 64 69 73 63 6f 72 64 61 70 70 2e 6e  -dl.disc ordapp.n
00b0 65 74 00 17 00 00 ff 01 00 01 00 00 0a 00 08 00  et-----
00c0 06 00 1d 00 17 00 10 00 06 00 02 01 00 00 22 00  #
```

```

Windows PowerShell
PS M:\Thesis-Work\Android 26-09-21> ja3 -a Discord.pcap
Traceback (most recent call last):
  File "F:\Program Files\Python39\lib\site-packages\ja3\ja3.py", line 242, in main
    capture = dpkt.pcap.Reader(fp)
  File "F:\Program Files\Python39\lib\site-packages\dpkt\pcap.py", line 251, in __init__
    raise ValueError('invalid tcpdump header')
ValueError: invalid tcpdump header

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "F:\Program Files\Python39\Scripts\ja3-script.py", line 33, in <module>
    sys.exit(load_entry_point('pyja3==1.0.0', 'console_scripts', 'ja3')())
  File "F:\Program Files\Python39\lib\site-packages\ja3\ja3.py", line 244, in main
    raise Exception("File doesn't appear to be a PCAP: %s" % e)
Exception: File doesn't appear to be a PCAP: invalid tcpdump header

```

Figure 29 Invalid TCPDUMP header error

4. On Windows, it had a lot of background communication. SSDP or Simple Service Discovery Protocol was running constantly in the background. So, in order to limit the background communications, this ssdp service had to be manually stopped from the service settings in windows 10. SSDP causes a lot of miscellaneous traffic in the background, we also saw miscellaneous TLS packets generated because of this service.
5. UpnP Device Host also works with SSDP. UpnP stands for Universal Plug and Play, it is used to automatically open the ports on the NAT by removing the need for manually port configuration on the port forward rules. So, this service was disabled as well.

9.721849	8.8.8.8	192.168.1.103	DNS	Standard query response 0xe80b No such name ...
9.784555	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
9.889136	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
9.992902	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.096557	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.201049	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.304959	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.408550	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.513033	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.616211	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.720592	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.824316	192.168.1.1	239.255.255.250	SSDP	NOTIFY * HTTP/1.1
10.880321	52.97.250.210	192.168.1.103	TCP	443 → 64977 [RST, ACK] Seq=1 Ack=1 Win=0 Len=4
12.096019	192.168.1.1	255.255.255.255	UDP	45779 → 7437 Len=173
12.927490	192.168.1.103	8.8.8.8	DNS	Standard query 0x8ca6 A jmdhpkswdbneamn.utu...

> Frame 14: 315 bytes on wire (2520 bits), 315 bytes captured (2520 bits)				
> Ethernet II, Src: Tp-LinkT_d5:4f:5e (c0:25:e9:d5:4f:5e), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)				
> Internet Protocol Version 4, Src: 192.168.1.1, Dst: 239.255.255.250				
> User Datagram Protocol, Src Port: 49463, Dst Port: 1900				
> Simple Service Discovery Protocol				

0000	01 00 5e 7f ff fa c0 25 e9 d5 4f 5e 08 00 45 00	..^...% ..0^..E-
0010	01 2d 00 00 40 00 04 11 c4 1c c0 a8 01 01 ef ff	...@...
0020	ff fa c1 37 07 6c 01 19 84 96 4e 4f 54 49 46 59	...7.l...NOTIFY
0030	20 2a 20 08 54 54 50 2f 31 2e 31 0d 0a 48 4f 53	* HTTP/ 1.1..HOS
0040	54 3a 20 32 33 39 2e 32 35 35 2e 32 35 35 2e 32	T: 239.2 55.255.2
0050	35 30 3a 31 39 30 30 0d 0a 43 41 43 48 45 2d 43	50:1900..CACHE-C
0060	4f 4e 54 52 4f 4c 3a 20 6d 61 78 2d 61 67 65 3d	ONTROL: max-age=
0070	31 30 30 0d 0a 4c 4f 43 41 54 49 4f 4e 3a 20 68	100..LOC ATION: h
0080	74 74 70 3a 2f 2f 31 39 32 2e 31 36 38 2e 31 2e	ttp://19 2.168.1.
0090	31 3a 31 39 30 30 2f 69 67 64 2e 78 6d 6c 0d 0a	1:1900/i gd.xml..
00a0	4e 54 3a 20 75 75 69 64 3a 32 35 34 65 39 39 37	NT: uuid :254e997

6. On Linux, the automation script for the network namespace can only be used to create one network name space at one time. Multiple network namespace cannot be created from the script provided. Even if multiple network namespace is created, only the first one created will work and rest will not. After the first one, the rest will try to create the same veth pair as the first one with same IP.
7. The analysis was also tried on a rooted device but the results obtained were incorrect. The device used for the analysis was Samsung Galaxy J7 (rooted) with Android 6. It was observed that “cad0d99275c692e82c0ac8d74cb16db9” was generated by every application. It is believed that because of root access, this was made by some malicious process running in the background.
8. On windows, another tool called as “ForceBindIP” which was can be used to forcefully bind the application to a particular IP address or Network Interface present on the device. But this tool worked in the earlier version of windows ie Windows Vista, XP, 98, 95 etc.

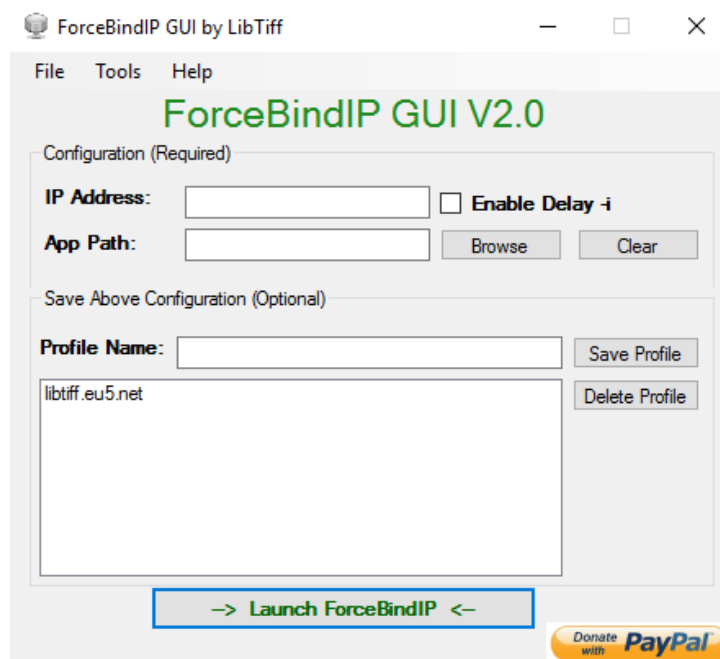


Figure 30 ForceBind GUI

9. In the android, one hash was generated “9b02ebd3a43b62d825e1ac605b621dc8” by almost all the applications. After further researching about this string on ja3er.com it was found out that this is related to “Dalvik/2.1.0 (Linux; U; Android 11; sdk_gphone_x86 Build/RSR1.201013.001)”. After reading more about this on the web, one result gave the information that this is a Dalvik User Agent. According to user-

agents.net, “This user agent string belongs to Dalvik browser running on Android OS. The browser is developed by Google Inc and renders web pages using the WebKit engine.” This user agent "cannot" be avoided because it is a system service so this specific user agent must be used to establish communication on the server. So, it is again very difficult to isolate the network activity of every application on the Android.

6.2 Scope for further research

This research work can be extended to bigger data set of applications running on same or different operating systems. A more efficient method or approach is needed to isolate the application network activity on Windows. Also, on Android, there is a need to avoid invoking of Dalvik User Agent which resulted in same JA3 hashes being produced by many applications.

References

- [1] U. Hiwarale, “A brief overview of the TCP/IP model, SSL/TLS/HTTPS protocols and SSL certificates,” IT Next, 2020. [Online]. Available: <https://medium.com/jspoint/a-brief-overview-of-the-tcp-ip-model-ssl-tls-https-protocols-and-ssl-certificates-d5a6269fe29e>
- [2] G. Perez, “JARM: A Solid Fingerprinting Tool for Detecting Malicious Servers,” securitytrails.com, 2020. <https://securitytrails.com/blog/jarm-fingerprinting-tool>
- [3] J. Althouse, “TLS Fingerprinting with JA3 and JA3S - Salesforce Engineering,” 2019. <https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967>
- [4] Lee Brotherston, “<https://github.com/LeeBrotherston/tls-fingerprinting>,” 20, 2015. <https://github.com/LeeBrotherston/tls-fingerprinting>
- [5] P. Matoušek, I. Burgetová, O. Ryšavý, and M. Victor, “On Reliability of JA3 Hashes for Fingerprinting Mobile Applications,” Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST, vol. 351, pp. 1–22, 2021, doi: 10.1007/978-3-030-68734-2_1.
- [6] ArvinF, “TLS Fingerprinting JA3 iRule Application: Rate limit and block malicious traffic based on TLS signature,” August 2020, 2020. <https://devcentral.f5.com/s/articles/TLS-Fingerprinting-JA3-iRule-Application-Rate-limit-and-block-malicious-traffic>
- [7] P. K. Gupta et al., “Machine learning interpretability meets TLS fingerprinting,” in Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, Apr. 2020, vol. 10, no. 1, pp. 415–428. doi: 10.23919/ICITST51030.2020.9351330.
- [8] B. Anderson and D. McGrew, “Accurate TLS fingerprinting using destination context and knowledge bases,” arXiv, 2020.
- [9] Henry Henry, “Simplewall Henry++,” henrypp.org, 2021. <https://www.henrypp.org/product/simplewall>
- [10] Github, “<https://github.com/ukanth/afwall>,” 2020. <https://github.com/ukanth/afwall>
- [11] A. Prodromou, “TLS Security 2: A Brief History of SSL/TLS | Acunetix,” 2019. <https://www.acunetix.com/blog/articles/history-of-tls-ssl-part-2/>
- [12] A. Prodromou, “establishing-tls-ssl-connection-part-5,” Acunetix.com, 2019. <https://www.acunetix.com/blog/articles/establishing-tls-ssl-connection-part-5/>
- [13] Digicert, “THE EVOLUTION OF SSL AND TLS,” 2015. <https://www.digicert.com/blog/evolution-of-ssl>
- [14] A. Satapathy and J. Livingston, “A Comprehensive Survey on SSL/ TLS and their Vulnerabilities,” International Journal of Computer Applications, vol. 153, no. 5, pp. 31–38, 2016, doi: 10.5120/ijca2016912063.
- [15] W. Stallings, “Cryptography and Network Security: Principles and Practice, 2nd ed.,” 1999.

- [16] T. P. N. Turner, S., IECA, “Prohibiting Secure Sockets Layer (SSL) Version 2.0,” *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2011.
- [17] S. Ganesh Kumar and E. Govindaraju, “An approach to implement cryptographic protocol version downgrade within a secure internal network: TLS 1.x to SSL,” *International Journal of Interactive Mobile Technologies*, vol. 13, no. 10, pp. 179–187, 2019, doi: 10.3991/ijim.v13i10.11308.
- [18] Packetlabs, “TLS 1.0 and TLS 1.1 Are No Longer Secure,” 2020.
- [19] P. Kotzias, K. G. Paterson, A. Razaghpanah, N. Vallina-Rodriguez, J. Amann, and J. Caballero, “Coming of age: A longitudinal study of TLS deployment,” in *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, 2018, pp. 415–428. doi: 10.1145/3278532.3278568.
- [20] B. Anderson and D. McGrew, “TLS beyond the browser: Combining end host and network data to understand application behavior,” *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC*, pp. 379–392, 2019, doi: 10.1145/3355369.3355601.
- [21] J. Čurguz, “Vulnerabilities of the SSL/TLS Protocol,” pp. 245–256, 2016, doi: 10.5121/csit.2016.60620.
- [22] M. Latovicka, S. Spacek, P. Velan, and P. Celeda, “Using TLS Fingerprints for OS Identification in Encrypted Traffic,” *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2020: Management in the Age of Softwarization and Artificial Intelligence, NOMS 2020*, 2020, doi: 10.1109/NOMS47738.2020.9110319.
- [23] N. C. A. S. Alerts, “<https://us-cert.cisa.gov/ncas/alerts>,” *Cybersecurity & Infrastructure Security Agency -Alerts*.
- [24] “TLS Security 5: Establishing a TLS Connection”.
- [25] CywareSocial, “What Is Cybersecurity Fingerprinting?,” CywareSocial, 2019.
- [26] P. S. Zlatina Gancheva and Lars Wüstrich, “TLS Fingerprinting Techniques.” https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2020-04-1/NET-2020-04-1_04.pdf
- [27] M. Korczyński and A. Duda, “Markov chain fingerprinting to classify encrypted traffic,” *Proceedings - IEEE INFOCOM*, pp. 781–789, 2014, doi: 10.1109/INFOCOM.2014.6848005.
- [28] J. Kim, J. Hwang, and K. Kim, “High-Performance Internet Traffic Classification Using a Markov Model and Kullback-Leibler Divergence,” *Mobile Information Systems*, vol. 2016, 2016, doi: 10.1155/2016/6180527.
- [29] J. Althouse, “TLS Fingerprinting with JA3 and JA3S - Salesforce Engineering,” 2019. <https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967> (accessed May 24, 2021).
- [30] R. Luks, “Nail These 6 Encrypted Traffic Cases with Flowmon | Flowmon”.

- [31] M. Heinemeyer, “Beyond the hash: How unsupervised machine learning unlocks the true power of JA3,” 2018. <https://www.darktrace.com/en/blog/beyond-the-hash-how-unsupervised-machine-learning-unlocks-the-true-power-of-ja-3/> (accessed May 30, 2021).
- [32] A. Alkazimi and E. B. Fernandez, “Cipher Suite Rollback : A Misuse Pattern for the SSL / TLS Client / Server Authentication Handshake Protocol,” p. 10, 2014.
- [33] V. Klíma, O. Pokorný, and T. Rosa, “Attacking RSA-based sessions in SSL/TLS,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 2779, pp. 426–440, 2003, doi: 10.1007/978-3-540-45238-6_33.
- [34] C. Meyer, J. Somorovsky, E. Weiss, J. Schwenk, S. Schinzel, and E. Tews, “Revisiting SSL/TLS implementations: New bleichenbacher side channels and attacks,” *Proceedings of the 23rd USENIX Security Symposium*, pp. 733–748, 2014.
- [35] C. Meyer and J. Schwenk, “SoK: Lessons learned from SSL/TLS attacks,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8267 LNCS, pp. 189–209, 2014, doi: 10.1007/978-3-319-05149-9_12.
- [36] J. Čurguz, “Vulnerabilities of the SSL/TLS Protocol,” pp. 245–256, 2016, doi: 10.5121/csit.2016.60620.
- [37] Z. Banach, “How the BEAST Attack Works | Netsparker”.
- [38] beaglesecurity.com, “SWEET32 Attack,” 18 June 2018. <https://beaglesecurity.com/blog/vulnerability/sweet32-attack.html>
- [39] M. Mimoso, “Theoretical Lucky Thirteen TLS Attacks Could Turn Practical”.
- [40] J. Fruhlinger, “What is the Heartbleed bug, how does it work and how was it fixed? | CSO Online,” CSO United Kindom, 2017.
- [41] H. Krawczyk, K. G. Paterson, and H. Wee, “On the security of the TLS protocol: A systematic analysis,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8042 LNCS, no. PART 1, pp. 429–448, 2013, doi: 10.1007/978-3-642-40041-4_24.
- [42] T. Bujlow, V. Carela-Español, J. Solé-Pareta, and P. Barlet-Ros, “Web Tracking: Mechanisms, Implications, and Defenses,” *arXiv*, pp. 1–29, 2015.
- [43] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, “Studying TLS Usage in Android Apps,” pp. 5–5, 2018, doi: 10.1145/3232755.3232779.
- [44] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, “A survey of methods for encrypted traffic classification and analysis,” *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015, doi: 10.1002/nem.1901.
- [45] R. Hofstede et al., “Flow monitoring explained: From packet capture to data analysis with NetFlow and IPFIX,” *IEEE Communications Surveys and Tutorials*, vol. 16, no. 4, pp. 2037–2064, 2014, doi: 10.1109/COMST.2014.2321898.

- [46] V. Paxson, "Bro: A system for detecting network intruders in real-time," Proceedings of the 7th USENIX Security Symposium, 1998, doi: 10.14445/23488387/ijcse-v3i5p106.
- [47] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," Eurasip Journal on Information Security, vol. 2016, no. 1, pp. 1–14, 2016, doi: 10.1186/s13635-016-0030-7.
- [48] I. Ristic, "Apache Security," Apache Security, vol. 2015, no. build 197, pp. 52–68, 2005.
- [49] O. Levillain, A. Ébalard, B. Morin, and H. Debar, "One year of SSL internet measurement," ACM International Conference Proceeding Series, pp. 11–20, 2012, doi: 10.1145/2420950.2420953.
- [50] J. Eрман, A. Mahanti, and M. Arlitt, "Internet traffic identification using machine learning," GLOBECOM - IEEE Global Telecommunications Conference, 2006, doi: 10.1109/GLOCOM.2006.443.
- [51] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," Proceedings - Conference on Local Computer Networks, LCN, vol. 2005, pp. 250–257, 2005, doi: 10.1109/LCN.2005.35.
- [52] B. Anderson, "TLS Fingerprinting in the Real World," Cisco Blogs, 2019. <https://blogs.cisco.com/security/tls-fingerprinting-in-the-real-world>
- [53] D. Benjamin, "Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility," no. 8701, 2020.
- [54] L. Deri and D. Sartiano, "Monitoring IoT Encrypted Traffic with Deep Packet Inspection and Statistical Analysis," 2020. doi: 10.23919/ICITST51030.2020.9351330.
- [55] developer.android.com, "Design for seamlessness," 2019. <https://developer.android.com/guide/practices/app-design/seamlessness> (accessed Oct. 10, 2021).
- [56] Andrey Egorov, "Receiving captured packets in Wireshark from PCAP Remote running on Android emulator," <https://egorovandreyrm.com/>, 2019. <https://egorovandreyrm.com/> (accessed Oct. 10, 2021).
- [57] G. Ross, "Osquery and JA3: Detecting malicious encrypted connections locally," www.uptycs.com, 2020. <https://www.uptycs.com/blog/osquery-and-ja3-detecting-malicious-encrypted-connections-locally> (accessed Oct. 10, 2021).
- [58] Filecroco.com, "The simplest way to protect your small business network," filecroco.com, 2021. <https://www.filecroco.com/download-simplewall/> (accessed Oct. 11, 2021).
- [59] S. Weblog, "----Introducing Linux Network Namespaces," no. September, pp. 1–7, 2013.

Appendices

The main heading of the appendices is not numbered. The same styles are used in the appendices as in the text chapters.

Appendix 1 Linux

Each appendix is numbered and given a heading.

JA3	Chrome	Edge	Firefox
48d7add7fc9c54f66ec4ab9165f46e4b			
e5b607b5862a46cab44d7bacd582b3cd			
7d52c9129b8b07502d1471697c2982dd			
c9539121a34c8104133553aa3b211b61	X		
1b73862eae8f1711440a446b1ef357fd		X	
dd6aa87fa4b2b05a72ab994663ad94c5		X	
fd0c0a519648e519f40f7fed962076fc		X	
20dff5d81f228563c0318b235baafae9			
a72f351cf3c3cd1edb345f7dc071d813			X
4d0dbc200ffec1085949cea3410a5547			X
aa50c12a5dfa717d9d6ab34e97de79d5		X	
bbcc15c33ab7b9c6e467584d9aeb3020			
7283d4318a3295c0b7d74a7eb02ce29f	X		
ff0f32710f49eabdc2d802d0be5b5695			X
e9a3c539cf07e9734e684bdd67e3847			
a6d03565735ac2f6f810b11d037e08ed	X		
fedca33016b974c390faa610378b5a62		X	
7d2fafa1dbc311cf6b2ed9ea1580bf7f			X
cb2a20fe5f35977440057f8bd1980fba	X	X	
ab890ae7a7163df667ef665287073a15	X		
af0b643ba6479160d5fdb42c09c32b9a			
ca450f8afb5c4c9c678d1c667c7482f2			
1af5d1fe5c1c9bdba4ec723ac5cab44f			X
7094b0d5b83dec3f4072bbf49e4b480a	X		
26e28949bdfc88c8769c99b2f20ad18d		X	
6b5e0cfe988c723ec71faf54f8460684			X
766543824ed9f5d5adeec04a50c21904			X
d5a82b7e86091b9fc3bd24e2c9ae2919			
5942576315948177ddb7f97eeebc23			X
27c1c07256b84edb63e0fba28778d5a1	X	X	
98d82c43de4e22a055da96bcef1ca9b9			
a0262d81f08838bbb1877a10e3fd70f1			X

b075db672378ecc07a6e964ebf923de6			
283e90b31bad9ee68a89ea1ad528e168	X		
0b18afc3387a48ed8e75634f89b9bd07			X
663d4f09c017c69df656f4372b7ee7ca		X	
259dd0f97c6df37f3c55a730eb0d9238			

Appendix 2 Windows

JA3	Anydesk	Chrome	Discord	Dropbox	Edge	Firefox	Skype	Spotify	Steam	Teams	Telegram	Teamview	Zoom
10f1ebcdc5bf282bdf4a ec39fb4cfc8b													X
a2de60d8d8cd26f3f11 31952a5adcba7									X				
7f805430de1e7d98b1d e033adb58cf46							X	X					
df208241e7f3897d4ca 38cfe68eabb21						X							
74954a0c86284d0d6e1 c4efefe92b521												X	
3cf05a4527fe8c00f228 443524e3d76e						X							
3b5074b1b5d032e562 0f69f9f700ff0e			X				X			X			
74ad8ec6876e2e3366b fd566581ca7e8		X	X		X			X					
af99c5134dc32b4f2e2 89e01ac7fe8b1						X							
66918128f1b9b03303d 77c6f2eefd128				X					X	X			
b32309a26951912be7d ba376398abc3b							X	X					
259dd0f97c6df37f3c55 a730eb0d9238													X
76e38c7e3a1f21d42c7 7d3e3db538c8e													X
aa7744226c695c0b2e4 40419848cf700						X							
48d7add7fc9c54f66ec4 ab9165f46e4b													X

8979ea6385c2505e8c6d4086a8a39be3		X			X								
54328bd36c14bd82dda a0c04b25ed9ad										X			
554719594ba90b02ae4 10c297c6e50ad									X	X			
effe9b59e99e730d14f2 3d971080682b				X									
e1d8b04eeb8ef3954ec 4f49267a783ef		X			X								
50a7456e0d02951e510 e4ace60fa3313													X
a72f351cf3c3cd1edb34 5f7dc071d813						X							
598872011444709307 b861ae817a4b60		X			X								
0d69ff451640d67ee8b 5122752834766		X			X								
0de4f16d9e23b224ed2 2c7eda958cedede					X								
029f230ecf557eb5774 7b5debc47512c						X							
080027e640f8ec6d966 f79fc2f7ca551										X			
ee7181db614cb63ec32 c96863ead1b1c		X						X					
a0e9f5d64349fb13191 bc781f81f42e1				X	X			X					
ef5b9ad415172970f6e 8ee7c87745c14									X				
832952db10f14534426 36675bed2702b													X
28a2c9bd18a11de089e f85a160da29e4										X			
af0b643ba6479160d5f db42c09c32b9a								X					
dda262729e5413660ec 0e6a8d4279860							X						
f14ec85ee5580a29f652 3e24e5d3d527						X							
cd08e31494f9531f560 d64c695473da9		X	X		X								

Appendix 3 Android

JA3	Anydesk	Chrome	Discord	Dropbox	Edge	Firefox	Skype	Spotify	Steam	Teams	Teamviewer	Telegram	Zoom
ee26b1f1aec16d6098768e2c67388ace									X				
8f41a697eff27e008f969cf7b5ba4117					X								
3fc9b7563d8f20616129ddf848a5fdde							X		X				X
84cd49a759d15947552e61b46fa5f2ec			X										
533b47a1acfb9b244447765a2ec2a4d				X									
3f27d27a5f1348476cac6ec873e505cf					X								
66918128f1b9b03303d77c6f2eefd128					X								
c834494f5948ae026d160656c93c8871						X							
af0b643ba6479160d5fdb42c09c32b9a								X					
3b92fc00fff571f3cd2ad12f32856bac							X						
0d69ff451640d67ee8b5122752834766		X											
cd08e31494f9531f560d64c695473da9		X					X	X	X	X			X
554719594ba90b02ae410c297c6e50ad					X								
6ec2896feff5746955f700c0023f5804							X		X				
a1c672bda2bda1a05bdca801144b2760							X						
8979ea6385c2505e8c6d4086a8a39be3		X					X						
7e8a75b0be593e6c7eb5992dd1de084d										X			
d75e7289c86c15b305ac36097bfa0487								X					
0ae18052c288c1bd39910255598ed827					X								

f3e8b92336e2ffa7b3 1fcfb9a4bbb617										X			
10f1ebcdc5bf282bdf 4aec39fb4cfc8b													X
c67e9dc27d283f1f8 9b4ebb4b4670c21			X			X		X					
3d90a696919abaf92 ba5e076e4421a6a		X											
48d7add7fc9c54f66 ec4ab9165f46e4b													X
10492f76a2b8f67ac 1672f76bbf7aa0e													X
2eacfe5e29a49ebf4e 7875340c67b41d		X											
5988720114447093 07b861ae817a4b60		X							X				
20dff5d81f228563c 0318b235baafae9							X			X			
e1d8b04eeb8ef3954 ec4f49267a783ef		X					X						
59fe159ce36975a15 410412d25c2e114				X									
29b5a018fa5992fe2 3560c16af0dc9fc	X												
eb725168365ef8cf4 aabcb309ce94b9e					X								
9b02ebd3a43b62d82 5e1ac605b621dc8		X	X	X	X	X	X	X		X	X		
f58f4c92d50fe2785d 35d6e2eb8756f2					X								
6b5e0cfe988c723ee 71faf54f8460684						X							
0acf40799ecbd8350 6d6caeb38514671										X			
74ad8ec6876e2e336 6bfd566581ca7e8		X											
a0262d81f08838bbb 1877a10e3fd70f1						X							
672d76590b3edaa92 dfbcab7c963ed12													X
eaabed81520b23ea8 a800b36bd7e359e							X						
f79b6bad2ad0641e1 921aef10262856b			X	X			X	X		X			

Appendix 4 All Operating Systems

JA3	Anydesk	Chrome	Discord	Dropbox	Edge	Firefox	Skype	Spotify	Steam	Teams	Telegram	Teamviewer	Zoom
66918128f1b9b03303d77c6f2ee fd128				X	X				X	X			
080027e640f8ec6d966f79fc2f7 ca551										X			
3cf05a4527fe8c00f228443524e 3d76e						X							
598872011444709307b861ae81 7a4b60		X			X				X				
29b5a018fa5992fe23560c16af0 dc9fc	X												
554719594ba90b02ae410c297c 6e50ad					X				X	X			
f3e8b92336e2ffa7b31fcfb9a4bb b617										X			
aa50c12a5dfa717d9d6ab34e97d e79d5					X								
84cd49a759d15947552e61b46f a5f2ec			X										
0de4f16d9e23b224ed22c7eda95 8cede					X								
029f230ecf557eb57747b5debc4 7512c						X							
df208241e7f3897d4ca38cfe68e abb21						X							
ff0f32710f49eabdc2d802d0be5 b5695						X							
dd6aa87fa4b2b05a72ab994663a d94c5					X								
e1d8b04eeb8ef3954ec4f49267a 783ef		X			X		X						
26e28949bdfc88c8769c99b2f20 ad18d					X								
4d0dbc200ffec1085949cea3410 a5547						X							
cb2a20fe5f35977440057f8bd19 80fba		X			X								
c9539121a34c8104133553aa3b 211b61		X											
283e90b31bad9ee68a89ea1ad5 28e168		X											
af0b643ba6479160d5fdb42c09c 32b9a								X					
af99c5134dc32b4f2e289e01ac7 fe8b1						X							
0acf40799ecbd83506d6caeb385 14671										X			
f14ec85ee5580a29f6523e24e5d 3d527						X							
663d4f09c017c69df656f4372b7 ee7ca					X								
7d2fafa1dbc311cf6b2ed9ea158 0bf7f						X							
fd0c0a519648e519f40f7fed962 076fc					X								
eb725168365ef8cf4aabcb309ce 94b9e					X								

f58f4c92d50fe2785d35d6e2eb8756f2					X								
10492f76a2b8f67ac1672f76bbf7aa0e													X
a0e9f5d64349fb13191bc781f81f42e1				X	X			X					
3b92fc00fff571f3cd2ad12f32856bac							X						
533b47a1acfb9b244447765a2ec2a4d				X									
10f1ebcdc5bf282bdf4aec39fb4cfc8b													X
20dff5d81f228563c0318b235baafae9							X			X			
7d52c9129b8b07502d1471697c2982dd							X			X			
a2de60d8d8cd26f3f1131952a5adcb7									X				
28a2c9bd18a11de089ef85a160da29e4										X			
27c1c07256b84edb63e0fba28778d5a1		X	X		X			X					
50a7456e0d02951e510e4ace60fa3313													X
eaabed81520b23ea8a800b36bd7e359e							X						
7283d4318a3295c0b7d74a7eb02ce29f		X								X			
ee26b1f1aacc16d6098768e2c67388ace									X				
54328bd36c14bd82ddaa0c04b25ed9ad										X			
832952db10f1453442636675bed2702b													X
0ae18052c288c1bd39910255598ed827					X								
74954a0c86284d0d6e1c4efefe92b521												X	
3b5074b1b5d032e5620f69f9f700ff0e			X				X			X			
98d82c43de4e22a055da96bcef1ca9b9										X			
effe9b59e99e730d14f23d971080682b				X									
d5a82b7e86091b9fc3bd24e2c9ae2919									X				
fedca33016b974c390faa610378b5a62					X								
9b02ebd3a43b62d825e1ac605b621dc8		X	X	X	X	X	X	X		X		X	
7e8a75b0be593e6c7eb5992dd1de084d										X			
a6d03565735ac2f6f810b11d037e08ed		X								X			
2eacfe5e29a49ebf4c7875340c67b41d		X											
48d7add7fc9c54f66ec4ab9165f46e4b													X
ee7181db614cb63ec32c96863ead1b1c		X						X					
76e38c7e3a1f21d42c77d3e3db538c8e													X
3f27d27a5f1348476cac6ec873e505cf					X								
e5b607b5862a46cab44d7bacd582b3cd									X	X			

b32309a26951912be7dba376398abc3b							X	X					
aa7744226c695c0b2e440419848cf700						X							
766543824ed9f5d5adeec04a50c21904						X							
1af5d1fe5c1c9bdba4ec723ac5cab44f						X							
e9a3c539cf07e9734e684bddd67e3847									X				
dda262729e5413660ec0e6a8d4279860							X						
7f805430de1e7d98b1de033adb58cf46							X	X					
ef5b9ad415172970f6e8ee7c87745c14									X				
f79b6bad2ad0641e1921aef10262856b			X	X			X	X		X			
259dd0f97c6df37f3c55a730eb0d9238													X
ab890ae7a7163df667ef665287073a15		X											
7094b0d5b83dec3f4072bbf49e4b480a		X	X				X	X		X			
c834494f5948ae026d160656c93c8871						X							
a1c672bda2bda1a05bdca801144b2760							X						
3d90a696919abaf92ba5e076e4421a6a		X											
c67e9dc27d283f1f89b4ebb4b4670c21			X			X		X					
1b73862eae8f1711440a446b1ef357fd					X								
8f41a697eff27e008f969cf7b5ba4117					X								
a72f351cf3c3cd1edb345f7dc071d813						X							
cd08e31494f9531f560d64c695473da9		X	X		X		X	X	X	X			X
d75e7289c86c15b305ac36097bfa0487								X					
b075db672378ecc07a6e964ebf923de6				X									
ca450f8afb5c4c9c678d1c667c7482f2							X						
3fc9b7563d8f20616129ddf848a5fdde							X		X				X
74ad8ec6876e2e3366bfd566581ca7e8		X	X		X			X					
5942576315948177ddb7f97e9ecbc23						X							
8979ea6385c2505e8c6d4086a8a39be3		X			X		X						
a0262d81f08838bbb1877a10e3fd70f1						X							
59fe159ce36975a15410412d25c2e114				X									
0d69ff451640d67ee8b5122752834766		X			X								
672d76590b3edaa92dfbcab7c963ed12													X
6b5e0cfe988c723ee71faf54f8460684						X							
bbcc15c33ab7b9c6e467584d9aeb3020									X				

0b18afc3387a48ed8e75634f89b9bd07						X							
6ec2896feff5746955f700c0023f5804							X		X				
Count	1	18	9	8	25	19	18	13	13	19	0	2	10

Appendix 5 Android Comparison with JA3ER

Linux_JA3_Unique	Count_from_Analysis	User-Agent	Count	Last_seen	Application_names
6b5e0cfe988c723ec71faf54f8460684	24137	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0	1	2020-07-16 13:07:51	Firefox
aa50c12a5dfa717d9d6ab34e97de79d5	11621	-	-	-	Edge
7094b0d5b83dec3f4072bbf49e4b480a	6378	chrome 98	-	2022-02-20 21:53:08	Chrome, Discord, Skype, Spotify, Teams
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36	2856	2022-01-19 2:08:17	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36	2941	2022-03-05 0:04:40	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	3348	2022-03-03 17:10:22	
4d0dbc200ffee1085949cea3410a5547	5340	-	-	-	Firefox
1af5d1fe5c1c9bdba4ec723ac5cab44f	3967	-	-	-	Firefox
a0262d81f08838bbb1877a10e3fd70f1	2048	-	-	-	Firefox
1b73862eae8f1711440a446b1ef357fd	1097	-	-	-	Edge
fd0c0a519648e519f40f7fed962076fc	929	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36	1	2021-12-01 19:00:34	Edge

dd6aa87fa4b2b05a72ab994663ad94c5	823	-	-	-	Edge
a6d03565735ac2f6f810b11d037e08ed	807	go lang useragent	1	2021-04-20 13:11:56	Chrome, Teams
0b18afc3387a48ed8e75634f89b9bd07	791	-	-	-	Firefox
7d2fafa1dbc311cf6b2ed9ea1580bf7f	466	-	-	-	Firefox
7283d4318a3295c0b7d74a7eb02ce29f	294	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0	38	2021-10-31 8:07:18	Chrome, Teams
		Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2715.91 Safari/537.36	38	2021-10-31 17:46:25	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2790.25 Safari/537.36	38	2021-10-31 17:49:31	
e9a3c539cf07e9734e684bddd67e3847	185	-	-	-	Steam
fedca33016b974c390faa610378b5a62	122	-	-	-	Edge
c9539121a34c8104133553aa3b211b61	78	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36 Edg/90.0.818.66	7	2021-10-31 17:57:40	Chrome
e5b607b5862a46cab44d7bacd582b3cd	77	yaroslav1	-	2019-08-05 10:58:20	Steam, Teams
		Mobile app YouCam Makeup	-	2020-11-30 16:44:23	
		Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148	347	2021-12-30 9:51:30	
		Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148	377	2021-12-30 9:49:38	
		Mozilla/5.0 (iPhone; CPU iPhone OS 15_1	477	2021-12-30 9:40:31	

		like Mac OS X) AppleWebKit/605 .1.15 (KHTML, like Gecko) Mobile/15E148			
7d52c9129b8b07502d1471697c 2982dd	69	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537 .36 (KHTML, like Gecko) Chrome/97.0.4692 .99 Safari/537.36 Edg/97.0.1072.69	1	2022-01-31 9:39:18	Skype, Teams
		Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0	4	2021-12-22 7:50:19	
27c1c07256b84edb63e0fba287 78d5a1	53	-	-	-	Chrome, Discord, Edge, Spotify
20dff5d81f228563c0318b235ba afae9	41	-	-	-	Skype, Teams
af0b643ba6479160d5fdb42c09 c32b9a	27	-	-	-	Spotify
bbcc15c33ab7b9c6e467584d9a eb3020	22	-	-	-	Steam
663d4f09c017c69df656f4372b7 ee7ca	19	-	-	-	Edge
b075db672378e0c07a6e964ebf 923de6	15	-	-	-	Dropbox
ca450f8afb5c4c9c678d1c667c7 482f2	14	axios/0.21.4	12	2022-01-23 12:06:21	Skype
		Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537 .36 (KHTML, like Gecko) Chrome/87.0.4280 .60 Safari/537.36	19	2021-10-31 17:44:06	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0	30	2021-10-31 18:11:57	Skype
48d7add7fc9c54f66ec4ab9165f 46e4b	11	-	-	-	Zoom
d5a82b7e86091b9fc3bd24e2c9 ae2919	4	-	-	-	Steam
98d82c43de4e22a055da96bcef 1ca9b9	4	-	-	-	Teams
766543824ed9f5d5adeec04a50 c21904	4	-	-	-	Firefox
a72f351cf3c3cd1edb345f7dc07 1d813	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0	30	2021-10-31 18:05:43	Firefox
	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64;	35	2021-10-31 18:11:49	Firefox

		rv:88.0) Gecko/20100101 Firefox/88.0			
	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0	42	2021-10-31 18:10:49	Firefox
cb2a20fe5f35977440057f8bd19 80fba	3	-	-	-	Chrome, Edge
283e90b31bad9ee68a89ea1ad5 28e168	3	-	-	-	Chrome
26e28949bdfc88c8769c99b2f2 0ad18d	3	-	-	-	Edge
5942576315948177ddb7f97e ecbc23	2	-	-	-	Firefox
ab890ae7a7163df667ef6652870 73a15	1	-	-	-	Chrome
259dd0f97c6df37f3c55a730eb0 d9238	1	Possible Zoom client	-	2020-08-04 0:18:19	Zoom
	1	Possible Zoom client	1	2020-08-07 14:54:21	Zoom
201999283915cc31cee6b15472 ef3332	1	-	-	-	Anydesk
ff0f32710f49eabdc2d802d0be5 b5695	1	-	-	-	Firefox

Appendix 6 Linux Comparison with JA3ER

Linux_JA3_Unique	Count from Analysis	User-Agent	Count	Last seen	Application names
6b5e0cfe988c723ee71faf54f8460 684	24137	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0	1	2020-07- 16 13:07:51	Firefox
aa50c12a5dfa717d9d6ab34e97de 79d5	11621	-	-	-	Edge
7094b0d5b83dec3f4072bbf49e4b 480a	6378	chrome 98	-	2022-02- 20 21:53:08	Chrome, Discord, Skype, Spotify, Teams
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/93.0.4577.82 Safari/537.36	2856	2022-01- 19 2:08:17	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36	2941	2022-03- 05 0:04:40	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36	3348	2022-03- 03 17:10:22	
4d0dbc200ffec1085949cea3410a5 547	5340	-	-	-	Firefox

1af5d1fe5c1c9bdba4ec723ac5cab44f	3967	-	-	-	Firefox
a0262d81f08838bbb1877a10e3fd70f1	2048	-	-	-	Firefox
1b73862eac8f1711440a446b1ef357fd	1097	-	-	-	Edge
fd0c0a519648e519f40f7fed962076fc	929	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36	1	2021-12-01 19:00:34	Edge
dd6aa87fa4b2b05a72ab994663ad94c5	823	-	-	-	Edge
a6d03565735ac2f6f810b11d037e08ed	807	go lang useragent	1	2021-04-20 13:11:56	Chrome, Teams
0b18afc3387a48ed8e75634f89b9bd07	791	-	-	-	Firefox
7d2fafa1dbc311cf6b2ed9ea1580bf7f	466	-	-	-	Firefox
7283d4318a3295c0b7d74a7eb02ce29f	294	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:49.0) Gecko/20100101 Firefox/49.0	38	2021-10-31 8:07:18	Chrome, Teams
		Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2715.91 Safari/537.36	38	2021-10-31 17:46:25	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/53.0.2790.25 Safari/537.36	38	2021-10-31 17:49:31	
e9a3c539cf07e9734e684bddd67e3847	185	-	-	-	Steam
fedca33016b974c390faa610378b5a62	122	-	-	-	Edge
c9539121a34c8104133553aa3b211b61	78	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.212 Safari/537.36 Edg/90.0.818.66	7	2021-10-31 17:57:40	Chrome
e5b607b5862a46cab44d7bacd582b3cd	77	yaroslavl	-	2019-08-05 10:58:20	Steam, Teams
		Mobile app YouCam Makeup	-	2020-11-30 16:44:23	
		Mozilla/5.0 (iPhone; CPU iPhone OS 15_1_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148	347	2021-12-30 9:51:30	
		Mozilla/5.0 (iPhone; CPU iPhone OS 14_8_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148	377	2021-12-30 9:49:38	

		Mozilla/5.0 (iPhone; CPU iPhone OS 15_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Mobile/15E148	477	2021-12-30 9:40:31	
7d52c9129b8b07502d1471697c2982dd	69	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36 Edg/97.0.1072.69	1	2022-01-31 9:39:18	Skype, Teams
		Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0	4	2021-12-22 7:50:19	
27c1c07256b84edb63e0fba28778d5a1	53	-	-	-	Chrome, Discord, Edge, Spotify
20dff5d81f228563c0318b235baafae9	41	-	-	-	Skype, Teams
af0b643ba6479160d5fdb42c09c32b9a	27	-	-	-	Spotify
bbcc15c33ab7b9c6e467584d9aeb3020	22	-	-	-	Steam
663d4f09c017c69df656f4372b7ee7ca	19	-	-	-	Edge
b075db672378ecc07a6e964ebf923de6	15	-	-	-	Dropbox
ca450f8afb5c4c9c678d1c667c7482f2	14	axios/0.21.4	12	2022-01-23 12:06:21	Skype
		Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.60 Safari/537.36	19	2021-10-31 17:44:06	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0	30	2021-10-31 18:11:57	Skype
48d7add7fc9c54f66ec4ab9165f46e4b	11	-	-	-	Zoom
d5a82b7e86091b9fc3bd24e2c9ae2919	4	-	-	-	Steam
98d82c43de4e22a055da96bcef1ca9b9	4	-	-	-	Teams
766543824ed9f5d5adeec04a50c21904	4	-	-	-	Firefox
a72f351cf3c3cd1edb345f7dc071d813	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0	30	2021-10-31 18:05:43	Firefox
	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0	35	2021-10-31 18:11:49	Firefox
	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0	42	2021-10-31 18:10:49	Firefox
cb2a20fe5f35977440057f8bd1980fba	3	-	-	-	Chrome, Edge
283e90b31bad9ee68a89ea1ad528e168	3	-	-	-	Chrome

26e28949bdfc88c8769c99b2f20ad18d	3	-	-	-	Edge
5942576315948177ddb7f97eecbc23	2	-	-	-	Firefox
ab890ae7a7163df667ef665287073a15	1	-	-	-	Chrome
259dd0f97c6df37f3c55a730eb0d9238	1	Possible Zoom client	-	2020-08-04 0:18:19	Zoom
	1	Possible Zoom client	1	2020-08-07 14:54:21	Zoom
201999283915cc31cee6b15472ef3332	1	-	-	-	Anydesk
ff0f32710f49eabdc2d802d0be5b5695	1	-	-	-	Firefox

Appendix 7 Windows Comparison with JA3ER

Windows_JA3_Unique	Count_from Analysis	User-Agent	Count	Last seen	Application names
aa7744226c695c0b2e440419848cf700	23473	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:92.0) Gecko/20100101 Firefox/92.0	-	2021-09-17 7:43:13	Firefox
		sneed	-	2022-02-06 6:09:38	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92.0	6699	2022-03-05 0:01:45	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:89.0) Gecko/20100101 Firefox/89.0	6748	2022-03-01 13:31:21	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0	6754	2022-03-03 4:46:31	
cd08e31494f9531f560d64c695473da9	14177	-	-	-	Chrome, Discord, Edge
df208241e7f3897d4ca38cfe68eabb21	9616	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:90.0) Gecko/20100101 Firefox/90.0	2	2021-10-31 19:14:14	Firefox
029f230ecf557eb57747b5debc47512c	6243	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:87.0) Gecko/20100101 Firefox/87.0	49	2021-10-31 19:10:30	Firefox
		x	3809	2021-06-11 22:29:12	
		Go-http-client/1.1	3856	2022-01-27 20:18:13	
598872011444709307b861ae817a4b60	3233	-	-	-	Chrome, Edge

e1d8b04eeb8ef3954ec4f49267a783ef	1126	-	-	-	Chrome, Edge
f14ec85ee5580a29f6523e24e5d3d527	977	-	-	-	Firefox
3cf05a4527fe8c00f228443524e3d76e	881	Go-http-client/1.1	1	2021-08-09 3:52:26	Firefox
b32309a26951912be7dba376398abc3b	447	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36	-	2022-02-22 11:33:14	Skype, Spotify
		Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.87 Safari/537.36	-	2022-02-25 20:09:54	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.159 Safari/537.36	47634	2022-03-04 23:46:07	
		Go-http-client/1.1	78998	2022-03-05 0:18:50	
		Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	156626	2022-02-28 11:35:33	
0d69ff451640d67ee8b5122752834766	240	-	-	-	Chrome, Edge
a0e9f5d64349fb13191bc781f81f42e1	217	Windows 10 socket initiating a TLS communication when going to a domain	-	2019-07-18 20:46:50	Dropbox, Edge, Spotify
		Windows 10 socket initiating a TLS communication when going to a domain	-	2022-01-21 6:50:39	
		WebexTeams	38	2021-06-30 16:17:14	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; Trident/7.0; rv:11.0) like Gecko	40	2022-02-24 19:09:03	
		Excel/16.0	375	2021-02-26 7:26:44	
66918128f1b9b03303d77c6f2eefd128	164	Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.0 Safari/537.36	26199	2022-01-14 12:20:04	Dropbox, Steam, Teams
10f1ebcdc5bf282bdf4aec39fb4cfc8b	164	Mozilla/5.0(X11;Ubuntu;Linuxx86_64;rv:36.0)Gecko/20100101Firefox/36.0	9476	2021-12-30 4:00:40	Zoom
48d7add7fc9c54f66ec4ab9165f46e4b	160	Mozilla/5.0 (iPhone; CPU iPhone OS 10_3_1 like Mac OS X) AppleWebKit/603.1.30 (KHTML, like Gecko) Version/10.0 Mobile/14E304 Safari/602.1	9210	2022-03-04 19:00:12	Zoom
af99c5134dc32b4f2e289e01ac7fe8b1	126	attack using brute force	-	2022-02-17 14:56:11	Firefox

3b5074b1b5d032e5620f69f9f700ff0e	71	poison attack	-	2022-01-04 14:03:08	Discord, Skype, Teams
		suspect some firewall-appliance that opens TLS (Browser: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36)	-	2022-01-19 11:53:35	
		Mozilla/5.0 (Windows NT; Windows NT 10.0; en-US) WindowsPowerShell/5.1.19041.610	462	2021-03-20 21:53:23	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.102 Safari/537.36	2149	2022-02-20 19:34:37	
		Go-http-client/1.1	8869	2022-03-03 23:00:53	
50a7456e0d02951e510e4ace60fa3313	44	-	-	-	Zoom
080027e640f8ec6d966f79fc2f7ca551	43	PostmanRuntime/7.28.1	1	2021-10-07 18:49:47	Teams
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.107 Safari/537.36	4	2021-07-29 0:50:33	
8979ea6385c2505e8c6d4086a8a39be3	43	-	-	-	Chrome, Edge
dda262729e5413660ec0e6a8d4279860	42	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.60 Safari/537.36	46	2021-10-31 17:44:06	Skype
		PostmanRuntime/7.28.4	58	2022-02-15 7:33:11	
		PostmanRuntime/7.26.8	67	2022-02-28 15:20:53	
effe9b59e99e730d14f23d971080682b	40	-	-	-	Dropbox
74ad8ec6876e2e3366bfd566581ca7e8	39	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/92.0.4515.131 Safari/537.36	-	2021-10-13 8:26:35	Chrome, Discord, Edge, Spotify
		Go-http-client/1.1	3	2021-04-20 11:47:23	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36	32	2022-02-26 18:44:40	
		Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) browser/0.0 Chrome/91.0.4472.164 Electron/13.2.2 Safari/537.36\}	36	2022-02-09 2:28:24	
a2de60d8d8cd26f3f1131952a5adcba7	39	Valve/Steam HTTP Client 1.0	4	2021-12-08	Steam

				9:58:38	
832952db10f1453442636675bed2702b	34	-	-	-	Zoom
af0b643ba6479160d5fdb42c09c32b9a	29	-	-	-	Spotify
74954a0c86284d0d6e1c4efefe92b521	24	Fiddler Everywhere	6	2022-01-01 16:13:09	Teamviewer
		curl/7.79.1	113	2022-03-05 2:16:55	
		retard	252	2021-07-22 17:22:32	
554719594ba90b02ae410c297c6e50ad	22	Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.108 Safari/537.36	-	2019-04-30 15:00:20	Steam, Teams
		Mozilla/5.0 (Linux; Android 9; SM-A530F) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/95.0.4623.1 Mobile Safari/537.36	1	2022-01-08 20:20:13	
		Mozilla/5.0 (Linux; Android 10; HRY-LX1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/84.0.4147.125 Mobile Safari/537.36	1	2021-02-12 12:00:08	
		-	2	2020-03-05 6:22:52	
76e38c7e3a1f21d42c77d3e3db538c8e	14	-	-	-	Zoom
a72f351cf3c3cd1edb345f7dc071d813	11	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0	30	2021-10-31 18:05:43	Firefox
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0	35	2021-10-31 18:11:49	
		Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:87.0) Gecko/20100101 Firefox/87.0	42	2021-10-31 18:10:49	
ee7181db614cb63ec32c96863ead1b1c	11	-	-	-	Chrome, Spotify
54328bd36c14bd82dda0c04b25ed9ad	8	Intel Graphics Executable Download Wrapper - gfxdownloadwrapper.exe	-	2021-09-20 8:38:45	Teams
		5e4e5596180ebd0ac0317125ee490707	-	2021-10-13 5:15:55	
		Lenovo computer updater	-	2022-02-14 8:41:08	
		Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36	250	2022-03-04 7:44:40	

		Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:56.0) Gecko/20100101 Firefox/56.0	251	2021-02-27 0:22:54	
		C#-http-client/4.0.30319.42000 (Microsoft Windows NT 6.2.9200.0; x64)	311	2022-01-11 9:23:34	
259dd0f97c6df37f3c55a730eb0d9238	6	Possible Zoom client	-	2020-08-04 0:18:19	Zoom
	6	Possible Zoom client	1	2020-08-07 14:54:21	
28a2c9bd18a11de089ef85a160da29e4	5	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18362	977	2022-03-04 3:23:03	Teams
	5	Mozilla/5.0 (Windows NT 10.0; Win64; x64; Xbox; Xbox One) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.19041	1506	2021-10-31 18:15:12	
	5	Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko	3331	2022-03-05 2:56:50	
7f805430de1e7d98b1de033adb58cf46	3	Mozilla/5.0 (Macintosh; Intel Mac OS X 11_0_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4328.0 Safari/537.36	181	2021-10-31 17:45:49	Skype, Spotify
	3	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/85.0.4183.83 Safari/537.36	290	2021-10-31 17:54:05	
	3	Go-http-client/1.1	2550	2022-02-23 9:18:42	
0de4f16d9e23b224ed22c7eda958cedc	2	-	-	-	Edge
3f2fba0262b1a22b739126dfb2fe7a7d	1	AnyDesk Relay	-	2020-12-15 23:04:19	Anydesk
	1	HTTPS/80 Anydesk	-	2021-02-08 8:49:56	
	1	AnyDesk Relay	1	2020-12-17 18:20:22	
ef5b9ad415172970f6e8ee7c87745c14	1	-	-	-	Steam