Application of Browser Fingerprinting using JA3 Hashes in Digital Forensics

Mathii, Pius Muisyo

Submitted in partial fulfilment of the requirements for the Degree of Master of Science in Information System Security (MSc. ISS) at Strathmore University

Faculty of Information Technology
Strathmore University
Nairobi, Kenya

September, 2021

Declaration and Approval

Declaration

I declare that this work has not been previously submitted and approved for the award of a Master Degree by Strathmore University or any other institution. To the best of my knowledge and belief, the dissertation contains no material previously published or written by any other person except where due reference is made in the dissertation itself.

Student: Mathii Pius Muisyo

Signature 

Date 15th September, 2021

Approval

This dissertation of Mathii Pius Muisyo, was reviewed and approved by:

Dr. Humphrey Njogu,
Lecturer, Faculty of Information Technology,
Strathmore University

Dr. Julius Butime,
Dean, School of Computing and Engineering Sciences,
Strathmore University

Dr. Bernard Shibwabo,
Director of Graduate Studies,
Strathmore University

Abstract

Web-based communication has become more secure in recent years as a result of Transport Layer Security (TLS) encapsulation. TLS increases user security by encrypting transmitted data; however, it restricts network monitoring and data capturing, which is important for digital forensics. With the constant evolution of TLS protocol suites, creating unique and stable TLS fingerprints for forensic purposes is difficult. Furthermore, content advertising and tracking plugins contribute to "communication noise," limiting the use of TLS fingerprinting. This paper describes an experiment using JA3 hashes for TLS fingerprinting of network applications and focuses on fingerprinting of browsers, specifically the stability, reliability, and uniqueness of JA3 fingerprints. The study also looks at the applicability of JA3 fingerprints in digital forensics. Agile software development methodology was used to achieve the design, implementation, testing and validation aspects of the solution. The final product was an interactive shell script that examines an unknown network capture file and identifies the identity of the browser that was used based on JA3 algorithm. The performance of the tool was good overall based on extensive testing and evaluation.

Keywords: Browsers. TLS fingerprinting. Network Forensics. JA3 hash. Encrypted Communication.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	-	Application Programming Interface
AWK	-	Aho Weinberger and Kernighan
CLI	-	Command Line Interface
CPU	-	Central Processing Unit
CSS	-	Cascading Style Sheets
DB	-	Database
DDR3	-	Double Data Rate 3
DHCP	-	Dynamic Host Configuration Protocol
DNS	-	Domain Name System
FOD	-	Function Oriented Design
FTP	-	File Transfer Protocol
GPL	-	GNU General Public License
GREASE	-	Generate Random Extensions and Sustain Extensibility
HTML	-	HyperText Markup Language
HTTP	-	Hypertext Transfer Protocol
HTTPS	-	HyperText Transfer Protocol Secure
IANA	-	Internet Assigned Numbers Authority
IETF	-	Internet Engineering Task Force
IM	-	Instant Messaging
IMAP	-	Internet Message Access Protocol
IP	-	Internet Protocol
LAN	-	Local Area Network
LDAP	-	Lightweight Directory Access Protocol
LEAs	-	Law Enforcement Agents
NNTP	-	Network News Transfer Protocol
OS	-	Operating System
PCAP	-	Packet Capture
PCAPNG	-	PCAP Next Generation file format
PII	-	Personally identifiable information
PKI	-	Public Key Infrasture
POP	-	Post Office Protocol
POP3	-	Post Office Protocol 3

PRF	-	Pseudorandom Function Family
RAM	-	Random Access Memory
RFC	-	Request for Comments
SED	-	Stream Editor
SMTP	-	Simple Mail Transfer Protocol
SQL	-	Structured Query Language
SSD	-	System Sequence Diagrams
SSL	-	Secure Sockets Layer
TCP	-	Transmission Control Protocol
TDD	-	Test-Driven Development
TLS	-	Transport Layer Security
UA	-	User Agent
UML	-	Unified Modelling Language
URI	-	Uniform Resource Identifier
URL	-	Uniform Resource Locator
VPN	-	Virtual Private Network
Wi-Fi	-	Wireless Fidelity
WSGI	-	Web Server Gateway Interface
XML	-	Extensible Markup Language
XSS	-	Cross-Site Scripting
XXE	-	XML External Entities

Term Definitions

Profiling is defined as the process of determining relationships between data in databases that can be used for identifying and representing a nonhuman or human subject (group or individual), as well as the presentation of profiles not only to individuate but also to identify or represent a member of a category or group (Sapsford & Jupp, 1996). Furthermore, data mining technology is commonly thought of as methods for revealing relevant patterns and producing profiles from large amounts of data.

Fingerprinting is a method of collecting publicly available information about a distant computing scheme known as attributes or features for the purpose of documentation resolution (Nikiforakis et al., 2013). The data is used to create a digital fingerprint of the remote device. Fingerprints can be used to partially or completely identify individual devices or users. Active fingerprinting uses querying to obtain specific fingerprinting data from a remote device, such as web browser parameters or network settings. Passive fingerprinting is based on data obtained by monitoring a remote device's communication without interfering with it.

Since there is a noticeable overlap of these terms, we need to clarify how these terms are used in our research.

Acknowledgements

Much appreciation and gratitude to God Almighty for giving me the strength, health and capacity to do the dissertation. Secondly, I take this opportunity to express my profound gratitude and deep regards to my supervisor Dr. Humphrey Njogu for his exemplary guidance, monitoring and constant encouragement throughout the course of this dissertation. I also thank my parents, sisters and brothers for their constant encouragement without which this research would not be possible.

I must acknowledge the many friends, colleagues, students, teachers, archivists, and other librarians as well who assisted, advised, and supported my research and writing efforts over the years. Especially, I need to express my gratitude and deep appreciation to Lucy and Daniel whose friendship, brotherhood, hospitality, knowledge, and wisdom have supported, enlightened, and entertained me over the many years of our friendship. They have consistently helped me keep perspective on what is important in life and shown me how to deal with reality.

Dedications

This dissertation is dedicated to my great parents and siblings who saw that I never lacked when I was schooling, not forgetting all other parties involved in the project including and in particular the management of @ilabAfrica for the award of the scholarship into MSc. ISS program.

Chapter 1 : Introduction

1.1 Background of the Study

One of the major objectives of network forensics is to analyse captured network communication and look for digital artefacts related with a security incident, cybercrime as well as espionage. Investigators usually ask who was sending the data, when, what was transmitted, or who was the recipient (Velan et al., 2015). Using packet capturing, flow statistics, cache analysis and log entries they try to describe suspect's behaviour.

In case of serious crimes, law enforcement agencies may intercept network traffic in order to find evidence. The 2015 revelation of millions of Wikileaks personal documents, many users and companies improved on transmitted data security, in particular opposing interception (Laperdrix et al., 2015). Increased imposition for transmitted data security has resulted to encrypted technique adoption by various network protocols. Since that time, only encrypted communication techniques are supported by almost all services and network application by the use of Transport Layer Security (TLS).

In response to communication encryption, researcher's activities are focused on behavior analysis of communication in an encrypted way, aimed at obtaining metadata concerning encrypted protocols and services (FaizKhademi et al., 2015). Secured transmission statistical analysis was one of the researcher's interests and also obtaining various features from TSL handshake for TSL fingerprint computation. According to Althouse (2019), John Althouse, Jeff Atkinson and Josh Atkins proposed one popular fingerprint TSL technique known as JA3 in 2015. Their methodology involved intrusion system for detection such as Flowmon, Suricata or Bro used mostly for detection of malware and blacklisting, also known as network application identification and monitoring (Williams et al., 2020).

In 2017, Lee Brotherston discovered that through a Secure Sockets Layer (SSL) handshake, several client user mediators will approach a TLS handshake application in a distinctive method. This includes web browsers in different operating systems such as Linux, Mac OS and windows (Lemon, 2015). The fingerprint relies on data from ClientHello messages in the SSL handshake. Inspired by Lee's work, a team from Salesfoce (John B. Althouse, Jeff Atkinson & Josh Atkins) created JA3, a standard for generating SSL user fingerprints in a simple and shareable way (Althouse, 2019).

JA3 gathers the decimal values of the bytes for the following fields in the Client Hello packet; SSL Version, Accepted Ciphers, List of Extensions, Elliptic Curves, and Elliptic Curve

Formats. It then concatenates those values together in order, using a "," to delimit each field and a "-" to delimit each value in each field (Jeng et al., 2021). These strings are then MD5 hashed to produce an easily consumable and shareable 32-character fingerprint. This is the JA3 SSL Client Fingerprint.

Web based apps communicate over the Internet when a user activity is initiated to check updates, synchronize local data, or retrieve remote status, making it probable to recognize a browser depending on typical set of presentations and their versions that are used over the internet (Shafagh et al., 2015). Web based applications can be identified from captured TLS traffic using JA3 hashes (retrieved from client communication) or JA3S hashes (retrieved from server communication).

Nonetheless, there are significant questions associated with digital forensics. They include: Are the fingerprints consistent enough to recognize a particular application? How steady are the fingerprints? How can an exclusive fingerprint database of browser apps be created? This study aims at studying useability of browser fingerprinting using JA3 hashes on designated web-based apps, present a way how unique fingerprints might be obtained and deliberate the useability of JA3 fingerprinting for arithmetical forensics.

This project extends JA3 functionalities by making it possible to easily identify the type of web browser based on network communication (Conti et al., 2015). Three additional SSL handshake fields are introduced to make the data more informative.

1.2 Problem Statement

Traditional browser fingerprinting approaches require active interaction with the device and do not associate the fingerprints with important metadata such as timestamp, communicating parties and protocols involved. Also, some fingerprints do not represent communication of the browser and remote server. These approaches do not sanitize the traffic sources therefore fingerprints generated this way include outliers, for instance, advertisement applications, operating system apps among others. The digital evidence resulting from such techniques present forensic challenges and cannot be applied by law enforcement agents for operational purposes or as a part of lawful interception.

This research addresses these challenges by developing an enhanced JA3 based fingerprinting tool (Lee Brotherston, 2017) that analyses Domain Name System (DNS) traffic to eliminate outliers and extract additional identifying information from packets to enhance the useability of JA3 fingerprints for forensic purposes.

1.3 Research Objectives

1.3.1 General Objective

To develop an advanced JA3 based fingerprinting tool that analyses Domain Name System (DNS) traffic to eliminate outliers and extract additional identifying information from packets to enhance the useability of JA3 fingerprints for forensic purposes.

1.3.2 Specific Objectives

1. To review common vulnerabilities and threats in browser communication.
2. To review existing fingerprinting techniques by other researchers used to identify web browsers.
3. To design, develop and test an improved JA3 based fingerprinting tool for browser identification in forensic scenarios based on clean and accurate fingerprints.
4. To validate the effectiveness of the tool and its use in digital forensics.

1.4 Research Questions

1. What are the vulnerabilities and threats that are common in browser communication?
2. What fingerprinting techniques by other researchers have been used in identifying web browsers?
3. How can JA3 algorithm be used in designing, developing, and testing a fingerprinting tool that could be used by forensic investigators to identify web browsers accurately based on clean fingerprints?
4. How can the effectiveness of the tool and its use in digital forensics be validated?

1.5 Justification

The possibility to identify web browsers, together with the communicating devices is beneficial to intelligence operations and forensic analysis. They can be able to know when communication happened, where and who was communicating, and which services were exchanging information. This extra information could serve as evidence and create a significant understanding to the case under investigation.

For instance, posts printed under the unidentified social system account might be exposed by relating the public posts' time with the actions' time as concluded by the web browser practise assisting hate criminality inquiries.

1.6 Scope

This research focuses on passive identification of web browsers using JA3 fingerprints and is not meant to be used in analysing live network traffic. The fingerprinting tool has been built to analyse captured network communication from three web browsers namely Firefox, Google Chrome and Opera from four operating systems; Mac OS, Kali Linux, Windows 10, and Debian.

Chapter 2 : Literature Review

2.1 Introduction

The chapter focuses on identifying and examining the study other researchers have done in the implementation of JA3 hashes fingerprinting. In this chapter, a structure of browser traffic is described, and the protocols that can be exploited to obtain fingerprinting data are also described. Using the combination of different identification techniques, the research will create a multi-level browser profile that can be used for identification of a given browser in the network traffic. Unlike other approaches, the research restricts himself to passive data capturing so that the result can be applied by law enforcement agencies for operational purposes or as a part of lawful interception.

A detailed information of what other researchers have found out in relation to the JA3 hashes fingerprinting will avoid unnecessary and unintentional duplication of the review and show nature with other existing research works. The review will form a framework within which the study findings will be interpreted in order to overcome the limitations in previous studies.

2.2 Importance of Information Security

As technology advances, organizations become more and more reliant on their information systems. The public as well gets concerned especially about the security of their personal information. The dangers to information frameworks from hackers are increasingly expanding. Currently information security is pivotal to all firms to secure their information and the behaviours of their business. As Kim & Kankanhalli (2009) defines it, Information security is the assurance/protection of information and the framework, and hardware that utilise, store and transmit that information.

According to Kim & Kankanhalli (2009), security of information systems helps in protecting a firm's ability to operate, enabling safe operation of applications executed on the firm's IT frameworks, safeguarding the technology assets used within the organisation and protecting the data that the organisation collects and utilises.

Generally, data held on IT frameworks is very valuable and basic to any business firms. It is crucial to be careful over information security since a significant part of the business value is strenuous in the value of its information. As Gollmann (2010) says, Information is the premise of competitive advantage. What is more, in the not-for-profit driven sector, with expanded awareness of fraud and information power, it is likewise, the part of a firm that most needs

control. Without information, neither organisations nor the not-for-profit driven firms could work. Protection and value of information are pivotal errands for modern organisations (Bedford, 2020). In this sense, each and every organisation should enforce security policies for the purpose of preserving confidentiality, integrity and availability.

2.3 Structure of Web Browser Communication

There are a lot of web browsers available in the market. Table 2.1 lists the most common web browser available today:

Table 2.1 Popular Web Browsers (Nielson, et al., 2008)

Browser	**Vendor**
Netscape Navigator	Netscape Communications Corp.
K-meleon	K-meleon
Sea Monkey	Mozilla Foundation
Safari	Apple
Google Chrome	Google
Internet Explorer	Microsoft
Mozilla Firefox	Mozilla
Opera	Opera Software

The most basic factor that all web browsers must have, according to Grosskurth & Godfrey (2005), are the following: Dispatcher/controller in the Processor that acts as a control unit. It also takes input from the mouse or keyboard, interprets it, and puts other features to work based on the feedback it receives. The regulator provides information to the interpreter, who then implements the order line by line. However, some interpreters are required while others are optional, such as the Java interpreter and the HyperText Markup Language (HTML) interpreter program (Joshi et al., 2017). The Client Program specifies the protocol that will be used to obtain a service. SMTP, HTTP, POP, NNTP, and FTP are the most commonly used client applications.

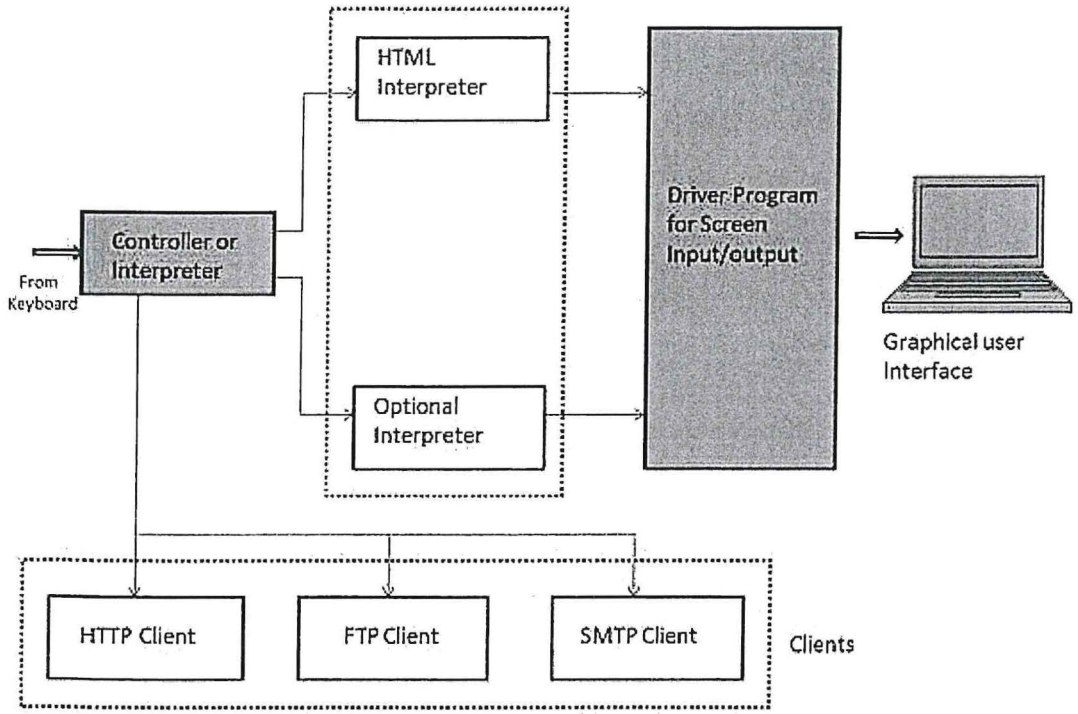


Figure 2.1 Browser Architecture (Grosskurth & Godfrey, 2005).

HTTP is used by web browsers to communicate with web servers. Web servers wait for client request messages, process them when they arrive, and send an HTTP Response message back to the browser. The HTTP Response status code in the response indicates whether or not the request was successful (e.g., "200 OK" for success, "404 Not Found" if the resource cannot be found, "403 Forbidden" if the user is not authorised to see the resource, etc) (Pohl, 2021). The requested resource will be in the body of an effective response to a GET request.

The web browser renders an HTML file when it is returned. The browser can discover links to other resources as part of the processing (for example, an HTML page can reference JavaScript and Cascading Style Sheets (CSS) pages, and will submit separate HTTP Requests to download these files.

HTTPS (HyperText Transfer Protocol Secure) is a variant of the HTTP protocol that is encrypted. All communication between a client and a server is encrypted using Secure Socket Layer (SSL) or TLS (Dastres & Soori, 2020). This secure link enables clients to share confidential data with a server in a secure manner, such as when conducting financial transactions or shopping online.

2.4 Browser Threats and Vulnerabilities

The following are the common browser threats and vulnerabilities. This study focused on the top ten.

2.4.1 Threats

Injection: Injection flaws, such as Structured Query Language (SQL), NoSQL, Operating System (OS), and Lightweight Directory Access Protocol (LDAP) injection, occur when untrusted data is sent to an interpreter as part of a command or query (Humayun et al., 2020). The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Cross-Site Scripting (XSS): XSS flaws occur when an application includes untrusted data in a new web page without sufficient validation or escaping, or when a browser API that can generate HTML or JavaScript updates an existing web page with user-supplied data (Gupta & Gupta, 2017). XSS allows attackers to run scripts in the victim's browser, allowing them to hijack user sessions, deface websites, or redirect users to malicious websites.

2.4.2 Vulnerabilities

Broken Authentication: Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently (Hassan et al., 2018).

Sensitive Data Exposure: Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and Personally Identifiable Information (PII). Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes (Yang et al., 2018). Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.

XML External Entities (XXE): Many older or poorly configured Extensible Markup Language (XML) processors evaluate external entity references within XML documents (Osincev & Laponina, 2019) External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

Broken Access Control: According to Hassan et al. 2018, restrictions on what authenticated users can do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.

Security Misconfiguration: Security misconfiguration is the most seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information (Poptani & Gatty, 2018). Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.

Insecure Deserialization: Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks (Bach-Nutman, 2020).

Using Components with Known Vulnerabilities: Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover (Kumar & Goyal, 2019) Applications and APIs using components with known vulnerabilities may undermine application defences and enable various attacks and impacts.

Insufficient Logging & Monitoring: Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data (Rivera-Ortiz & Pasquale, 2020). Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

2.5 Secure Communication for Web Browsers

The web browser is one of the most heavily used programs on a computer or mobile device today. Because of its ubiquitous nature, it is also an extremely popular target for attackers. Though the web browser was only originally intended to display text documents, it has since become the de facto tool or framework for interacting with videos, images, forms, games and all the other content the Internet has to offer (Soni & Jain, 2018). While it is very handy for the user to have a single program handle so many different types of media and functions, it also

makes the web browser more complex to secure, as it leaves many 'weak points' that attacker can leverage to their own advantage.

The most targeted aspects of a web browser include connections to online resources, plugins installed on the browser and vulnerabilities in the browser itself. To fetch content from a site for viewing, a web browser normally communicates with a DNS server that directs it to the correct site; the site then provides the desired content to the browser (Soni & Jain, 2018). There are various attacks that subvert and intercept this communication. The actual interception can happen at various points, and usually ends in redirecting the browser to a malicious site, where it (and the user) is exposed to unsolicited content, drive-by downloads, and exploit kits.

Attackers can target vulnerabilities in third-party plugins that users install on their browser to either hijack the browser's web traffic, snoop on it (particularly for sensitive finance-related data) or to perform harmful actions on the device, such as installing malware (Rezaei & Liu, 2019). Attackers often leverage flaws in the browser itself to either snoop on sensitive data transmitted via the web browser (for example, when entered in forms on a webpage) or to perform harmful actions on the device.

2.6 Secure Communication for Web-based Applications using TLS

TLS is a collection of cryptographic procedures designed to ensure the security of communications over a computer system (Dierks & Rescorla, 2008). Besides, TLS is a replacement to SSL. SSL was developed by Netscape in the 1990s to offer secure transactions between web browsers and web servers in support of commerce over the Internet. SSL became a de facto standard; however, it has since been turned outdated by TLS which is standardised by the Internet Engineering Task Force (IETF). TLS involves a handshake between the client and server before adding encoded infrastructures. This handshake is then used to choose the best equally suitable hashing algorithms, compression schemes, and cryptographic ciphers, among other things. The whole process is done in clear, since the cryptography method to apply is yet to be established.

TLS uses Public Key Infrastructure (PKI) to validate public key cryptography and peer systems to ease the interchange of period keys that are used to encrypt the SSL session (Turner, 2014). Many applications use TLS to provide authentication and encryption. The most widely used application is HTTPS (Rescorla, 2000). Other well-known applications that were using poor authentication and no encryption were modified to be transported within TLS. Examples include SMTP (Hoffman, 2002), LDAP (Hodges, Morgan & Wahl, 2000), and POP3

(Newman, 1999). Figure 2.2 depicts the steps that are taken in the negotiation of a new TLS connection between a web server and web browser.

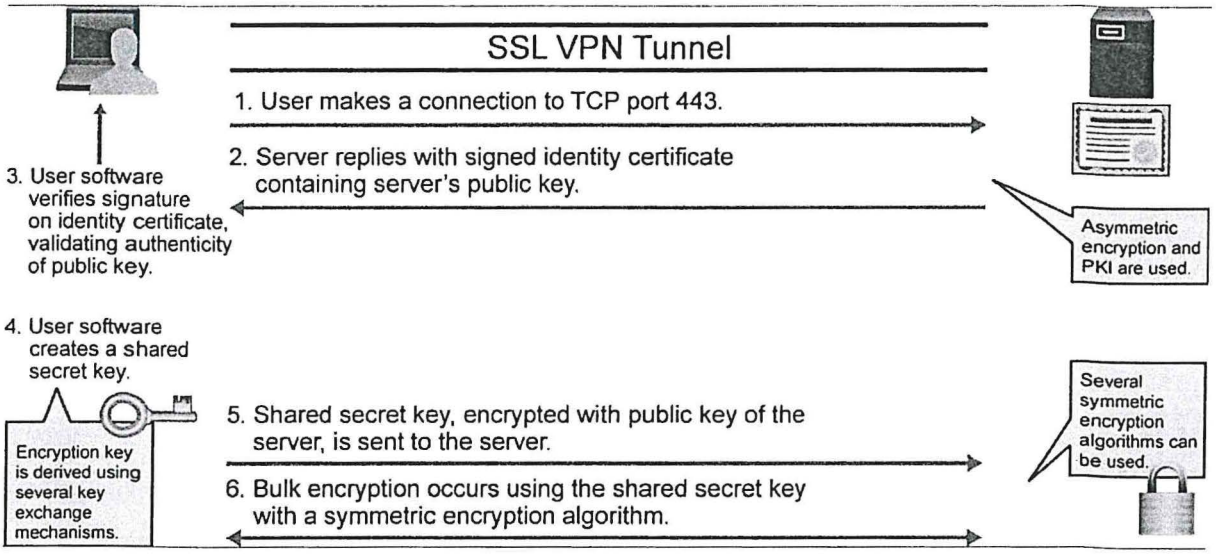


Figure 2.2 Cryptographic architecture of SSL and TLS (Rescorla, 2000)

2.6.1 Cipher Suites

A cipher suite is a set of algorithms that help secure a network connection that uses SSL/TLS (Steiner et al., 2001). An SSL/TLS cipher suite is used to define a set of cryptographic algorithms including the authentication and key exchange algorithms (such as RSA), encryption algorithm (such as Advanced Encryption Standard - AES), message authentication code algorithm (such as SHA), and the Pseudorandom Function Family (PRF). The cipher suites are described in RFC 5288 and RFC 5289.

When a TLS connection is established, a TLS handshake occurs. Within the TLS handshake, a client hello and a server hello message are passed. First, the client sends a list of the cipher suites that it supports, in order of preference. Then the server replies with the cipher suite that it has selected from the client's list (Polk, McKay & Chokhani, 2014). Figure 2.5 depicts part of a Packet Capture (PCAP) with a TLS client that is presenting the set of cipher suites that it can support to the server.

Figure 2.3 PCAP listing the cipher suites that client supports (Nath, 2015)

2.7 Analysis of Encrypted Network Traffic

The growing popularity of encrypted network traffic has a positive and negative side. On the one side, it ensures safe data transfer, prevents eavesdropping, and boosts the reliability of communicating hosts. In the other hand, it makes legal network traffic control, such as traffic classification and host recognition, more difficult. We can now track, recognise, and classify plain-text network traffic like HTTP, but analysing encrypted communication is difficult (Husák et al., 2016).

The options for establishing SSL/TLS communication have been established by researchers, and these options are used in real traffic. They employ methods from Velan et al. (2015) survey

as a foundation and real-world network data to classify these possibilities They then determined which choices vary the most and whether this variability reflects different traffic trends, such as different communication partners or types of traffic.

It is simple to decide which client is being used using TLS Fingerprinting, and then to apply this knowledge from both the attacker and the defender perspectives (Lemon, 2015). A TLS communication will always start with a ClientHello packet, which announces the client's capabilities to the server end of the connection in priority order. The server will respond with a "server hello" packet that describes the server's capabilities in priority order. The client and server will decide the best cipher suites, compression algorithms, and other features to use based on their preferences by comparing the two packets.

It is possible to create a fingerprint to recognize a client on subsequent sessions by collecting the elements of the ClientHello packet that remain unchanged from session to session for each client. TLS version, record TLS version, cipher suites, compression options, and a list of extensions are among the fields captured. In addition, data from three separate extensions (if available): signature algorithms, elliptic curves, and elliptic curve point format are captured (Delignat-Lavaud et al., 2017). This combined data is not only more accurate in terms of remaining static for every client, but it also provides more granularity than evaluating cipher suites alone, which has a much higher number of fingerprint collisions.

User recognition and browser fingerprinting play an important role in network security and the detection of malicious activities, for example, by identifying obsolete systems and detecting suspicious behavior. Commercial uses (targeting advertising, price discrimination, determining financial credibility), network accounting, and client behavior tracking all benefit from identification and fingerprinting (Bujlow et al., 2015).

Some researchers have attempted to classify clients using the HTTP User-Agent (UA) string, but this approach is unreliable. The User-Agent string is still at risk of being forged. Illegitimate web crawlers and bots, for example, often spoof the User-Agent string in order to be mistaken for legitimate ones like Googlebot (Zeifman, 2012). Another reason HTTP UA strings cannot be trusted is that web browsers have been using identifiers of each other in their User-Agent for a long time in order to address compatibility problems with those web pages.

Fingerprinting based on mobile device hardware is one of the feasible approaches for identifying mobile devices based on captured network contact that has been researched in the past. This approach assesses the device's physical characteristics, such as the image sensor,

frequency response of the speaker-microphone system, accelerometer accuracy, GPS clock skew, touch screen misalignment, and so on (Bojinov et al., 2014). We can classify a group of devices with the same or identical hardware using this method. Obtaining such a fingerprint necessitates active contact with the user, which is typically accomplished by the use of a specially designed program that collects all relevant data from the device. Hardware features for passive network management are difficult to acquire.

2.8 Passive Protocol Fingerprinting

Protocol fingerprinting techniques show variations in how different software systems use and execute protocols. As a result, fingerprinting of the most popular Internet protocols is possible. We classify candidate attributes to obtain the protocol fingerprint (Lastovicka et al., 2018). Attributes can be built from the protocol structure or they can be equivalent to protocol fields. User-Agent, for example, may be a useful protocol attribute. The following subsections describe a collection of attributes that can be used to fingerprint each protocol. The attributes were chosen based on published research and our own experiments. Additionally, the sum of information provided by each attribute can be computed to validate the attribute collection.

2.8.1 TCP/IP Communication

Based on the information in TCP and IP headers, passive TCP/IP fingerprinting will decide the device's operating system (Taleck, 2004). This is possible due to minor variations in network stack implementations that enable each operating system to be identified. Instead of aggressively probing the target structure, the passive approach is non-intrusive and relies on observation of daily traffic. The commonly used passive fingerprinting tool f0p is rule-based and relies on a manually generated signature database. Initial time to live, IPv4/IPv6 option length, maximum segment size, tcp window size, tcp window scaling factor, tcp options and order, selective recognition option, and content of SYN packet attributes are all included in the OS fingerprinting.

2.8.2 DHCP Communication

DHCP fingerprinting enables a computer and its operating system to be identified. DHCP provides a number of choices for identifying the customer (Chang et al., 2015). The client's Discover and Request packets contain DHCP header data, which is used to create a fingerprint. The options section is the most interesting because it contains information such as the operating system name, application name, vendor id, and other values. Furthermore, each DHCP client has a unique set of options that are dependent on the operating system and version.

Additional parameters include domain name server, renewal time, netmask, and DHCP server identifier, among others. These additional parameters, on the other hand, are dependent on the ISP provider, so they adjust when you connect to a different operator. The following DHCP attributes are used for DHCP fingerprinting: client identifier, host name, vendor class identifier, and parameter request list (Bai et al., 2019). These characteristics are consistent. Furthermore, these device data are created by client software and are difficult to forge by the user of a mobile phone. The disadvantage of DHCP fingerprinting, as previously mentioned, is that it is restricted to the LAN only.

2.8.3 DNS Communication

DNS data is used for the majority of Internet applications. We can determine the operating system, installed programs, and user behaviour by analysing DNS data. DNS communication is not encrypted by devices. As a result, the required information can be extracted quickly from the DNS header. We extract attribute values from regular unicast and multicast DNS requests for fingerprinting (Matsunaka et al., 2013). DNS fingerprints are built using the following information: DNS request sort, DNS server IP address, and DNS resolved name. By analysing DNS query patterns, it is possible to classify operating systems and some applications in addition to collecting DNS queries. Operating systems have a habit of sending complex DNS queries on a regular basis. It can be used as a secondary method of detecting the operating system.

DNS fingerprinting, on the other hand, is influenced by user behavior and link time. We receive hundreds of unique DNS domain names after collecting DNS data for an hour. As a result, these data must be filtered and a secure set of DNS names must be found before the fingerprint can be created (Matsunaka et al., 2013). Filtering can be done based on the number of unique domain names found, with domain names that fall below a certain threshold being removed from the fingerprint.

2.8.4 HTTPS Communication

SSL/TLS encrypts a large amount of network traffic. Based on the contact material, this decreases fingerprint capabilities. It is possible to recognize the implementation of SSL/TLS library and its version by observing information exchanged during safe link establishment (Husák et al., 2016). SSL/TLS implementations with unique identifiers and cipher suite sets are commonly found on operating systems. The following attributes from the Client Hello

opening message can be used for SSL fingerprinting: TLS/SSL Version (SSL 3.0, TLS 1.0, TLS 1.1, and TLS 1.2), cipher suite list, and supported extensions.

A cipher community is a collection of cryptographic algorithms that can be used to encrypt and secure messages (McGrew & Bailey, 2012). Each cipher suite is characterized by an IANA-defined standard value. The SSL/TLS library and its version determine the content and order of available cipher suite objects. The following is an example of a cipher suite list: 49195, 49199, 49162, 49171, 49172, 156, 47, and 53. The SSL/TLS library also includes a list of available extensions for fingerprinting, such as 0, 11, 10, 35, 13, 13172, and 16.

2.9 JA3 Algorithm

The main idea for fingerprinting TLS clients came from Lee Brotherston's 2017 research (Gancheva et al., 2020), in which he discovered that by collecting the elements of the ClientHello packet that remain unchanged from session to session for each client, it is possible to create a fingerprint to recognize a specific client on subsequent sessions. John Althouse, Jeff Atkinson, and Josh Atkins (2019) took this idea and turned it into an open-source tool called JA3.

The first packet sent by the client is used by the majority of TLS fingerprinting methods: Client Hello. The Client Hello includes an imprint of the client application's TLS configuration, which is dependent on the TLS library and operating system used. In this paper, we look at the JA3 fingerprint, which is derived from five TLS handshake fields: TLS Handshake version, Cipher suites, Extensions, Supported Groups (former Elliptic Curve), and Elliptic Curve point format (see Figure 2). Some TLS fingerprinting implementations use different TLS fields; for example, Kotzias et al. (2018) do not use the TLS version.

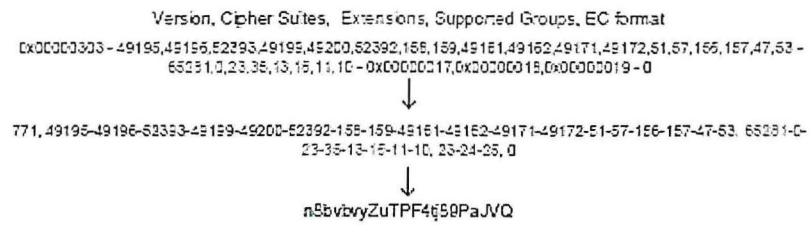


Figure 2.4 Computing JA3 Hash (Althouse, 2019)

According to Althouse (2019) the JA3 approach is used to collect the decimal values of the bytes in the Client Hello packet's following fields:

1. TLS version: The TLS protocol version in which the client wants to interact during this session. This SHOULD be the most recent (highest-valued) version that the client supports. For instance, 00000303.

2. Ciphers: This is a list of the client's cryptographic choices, ordered by the client's first choice. This vector MUST contain at least the cipher suite from that session if the session id field is not empty (indicating a session resumption request) (Althouse, 2019). For example, 4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-53-10 and 4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-53-10.

3. Extensions: Clients can send data in the extensions field to request additional functionality from servers. E.g., 0-23-65281-10-11-35-16-5-51-43-13-45-28-21

4. EllipticCurves and EllipticCurvePointFormats: These allow for the negotiation of particular curves and point formats (e.g., compressed vs. uncompressed) during the handshake that initiates a new session (Althouse, 2019). These extensions are particularly useful for clients who only accept a limited number of curves or point formats. By using the required extensions in its ClientHello message, the client enumerates the curves it supports and the point formats it can parse. By using an extension in its ServerHello post, the server also enumerates the point formats it may parse. 0000001d 00000017 00000018 00000019 00000100 00000101 0 is an illustration.

In TLS -version and elliptic-curves, hexadecimal values are translated to decimal. The values are then concatenated in sequence, using a "," to delimit each field and a "-" to delimit each value in each field (Althouse, 2019). The following is the resultant string:

771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-53-10,0-23-65281-10-11-35-16-5-51-43-13-45-28-21,29-23-24-25-256-257,0

This string is then hashed with md5 to produce a fixed length hash value. The hash below is generated by the string above:

B20b44b18b853ef29ab773e921b03422

This is one client-hello packet's JA3 fingerprint. If a similar browser is used, the client-hello message would be the same, resulting in a positive identification by matching the hashes (Althouse, 2019). Fingerprinting is the term used to describe this process.

In contrast to Firefox, the JA3 implementation for Google Chrome and Opera browsers is slightly different due to presence of some random values in their client hello packets, introduced by their creator, Google LLC. Google begun working on a project aimed to Generate Random Extensions and Sustain Extensibility (GREASE) standards to TLS. The method was accepted by IETF in January 2020 as RFC 8701 (Benjamin, 2020). They are meant to reduce extensibility challenges in the TLS ecosystem.

To keep JA3 fingerprints stable, the above-mentioned values must be removed. GREASE values are normally excluded from TLS fingerprinting in most JA3 implementations. TLS fingerprinting can also exclude extension value 65281 in addition to GREASE values. The red numbers in the list of extensions reflect a renegotiation option in the TLS handshake (Rescorla, 2010). The TLS Client Hello Padding Extension specified by RFC 7685 is the final option that can be ignored (Postel, 1980). A client adds the padding extension (value 21, shown in green in the table) to ensure that the packet is of the correct size.

2.10 Existing Solutions

2.10.1 Fast Go Implementation

Open Systems (2019) developed a Go library based on the JA3 algorithm as well as a command-line tool that reads packets from PCAP and PCAPNG files as well as an interface. The JA3 library's Go implementation was designed to test performance, robustness, and correctness. The solution uses TLS Client Hello packets to provide JA3 Client Fingerprinting for the Go language. Basic Use ja3 takes a byte of TCP payload data and generates a JA3 string and digest for it. The authors discovered that the Go implementation was hundreds of times faster than the Python implementation. The table 2.2 summarises the findings which include the execution time by the user and internal system calls. The CPU utilisation is also indicated.

Table 2.2 JA3 Tests and Benchmarks

	JA3Exporter	**Official Python Implementation**	**Dreadl0cks Go Implementation**
User	0.46s	23.10s	2.47s
System	0.05s	0.40s	0.11s
CPU	113%	98%	164%
Total	0.453	23.874	1.565

Unfortunately, the authors could not find any public available implementation that could fulfil all these requirements.

2.10.2 Detecting Malware in TLS Traffic

Roques (2019) designed and implemented an alternative to the unpractical method of decrypting TLS packets' payload before looking for signs of malware activity. The work presented a supervised classifier that can detect malicious TLS flows in a company's network traffic based on a set of features related to TLS, certificates, and flow metadata. The classifier was trained on curated datasets of benign and malware observations, which were extracted from capture files thanks to a set of tools specially developed for this purpose.

Malware often use custom parameters when communicating via TLS to their command-and-control server, which results in a unique JA3 fingerprint. Knowing that, blacklists of malware' JA3 hashes can be compiled (Roques, 2019). The researcher developed a JA3 profiling detector that extracts JA3 from TLS flows and builds a profile of typical JA3 hashes for each host in the network. Then after that training period, new JA3 hashes absent from a host's profile and its neighbours are flagged as suspicious. Combining other detectors with the classifier would serve to improve precision and lower the false positive rate.

2.10.3 On Reliability of JA3 Hashes for Fingerprinting Mobile Applications

Matoušek et al. (2020) performed various experiments on mobile apps to measure the stability, reliability, and uniqueness of JA3 hashes in identification. They concentrated their research on mobile devices, especially the identification of mobile apps in network traffic. One of the characteristics of mobile apps is that they communicate over the Internet on a daily basis without overt user intervention, such as for software updates, data synchronization, or remote status checks (Razaghpanah et al. 2017). This allows for the identification of a mobile device based on a collection of applications installed on the device (Kurtz et al., 2016). Mobile apps can be identified from the captured TLS traffic using JA3 hashes (retrieved from the client's side of communication) or JA3S hashes (the server's side of communication). The table 2.3 shows their findings for various mobile apps across android 7, 8.1 and 9.

Table 2.3 Stability of TLS Fingerprints over OS Version

Mobile App	Android 7	Android 8.1		Android 9	
Feature	present	added	missing	added	missing
CP JA3	1	1	1	0	0
CP JA3S	2	2	2	0	0
CP SNI	2	0	0	0	0
Mujvlak JA3	1	1	1	0	0
Mujvlak JA3S	1	0	0	0	0
Mujvlak SNI	1	0	0	0	0
Reddit JA3	1	2	1	0	0
Reddit JA3S	1	2	1	0	0
Reddit SNI	9	3	2	4	0
Seznam CZ JA3	3	3	3	2	1
Seznam CZ JA3S	11	0	0	2	0
Seznam CZ SNI	12	0	1	1	0

2.10.4 Other Fingerprinting Solutions

The options for establishing SSL/TLS communication have been established by researchers, and these options are used in real traffic. They employ methods from Velan et al. (2015) survey as a foundation and real-world network data to classify these possibilities They then determined which choices vary the most and whether this variability reflects different traffic trends, such as different communication partners or types of traffic.

Some researchers have attempted to classify clients using the HTTP User-Agent string, but this approach is unreliable. The User-Agent string is still at risk of being forged. Illegitimate web crawlers and bots, for example, often spoof the User-Agent string in order to be mistaken for legitimate ones like Googlebot (Zeifman, 2012). Another reason HTTP UA strings cannot be trusted is that web browsers have been using identifiers of each other in their User-Agent for a long time in order to address compatibility problems with those web pages.

TLS fingerprinting isn't a new concept, and its origins can be traced back to Ivan Ristic's security research in 2008, when he developed an Apache module that latently fingerprinted associated customers based on figure suites. Using this method, he created a mark base that differentiated a variety of programs and operating systems (Abel, 2019). This approach was

then extended to HTTP customer IDs (Husak et al., 2016) and used in the Bro and Surikata IDS systems for inactive discovery.

Blake Anderson et al. (2018) looked at a variety of TLS encoded streams and introduced a number of familiar information features from TLS consumer and worker hello messages, such as TLS rendition, TLS figures suites, and TLS augmentations, which they used to locate mal-products. They also took note of the employee's approval and the customer's public key duration, as well as the grouping of record lengths, dates, and types of TLS meetings. They were able to identify code suites and augmentations that were present in malware traffic but not in normal rush hour traffic. TLS customer setups for the 18 malevolent families were defined by the developers. Essentially, they identified TLS worker setups that were frequented by 18 nefarious families. For malware order, they use TLS along with various features (stream information, between appearance times, byte conveyance). They improved the classifier's accuracy from 96.7 percent to 98.2 percent (Blake et al., 2018). Excluding TLS features results in significantly worse execution, as shown by their research.

Anderson and McGrew (2019) propose a method that combines end-host data with network data to better understand application behavior. This process, on the other hand, necessitates access to both end hosts and linked networks. Their fingerprint database contains real traffic from 24,000 hosts, with 471 million benign connections and millions of malware connections. The authors were able to connect destination information with end point data such as timestamp, endpoint ID, operating system, and process name using end point data. They discovered that while GREASE values are produced spontaneously, their location is deterministic during fingerprint analysis.

Rather than eliminating GREASE values, they changed them to 0a0a. They also looked at TLS fingerprint similarity, which was measured using the Levenshtein distance. If the difference between two TLS fingerprints was less than or equal to 10% of the number of cipher suites, extension types, and extension values, they were considered identical. The Levenshtein distance, according to the authors, is an intuitive method for recognizing near fingerprints (Anderson & McGrew, 2019). They also discovered that certain TLS libraries alter their default parameters to best fit the platform they are running on. Another noteworthy aspect of the dataset is the prevalence of application types. They discovered that browsers account for 37.1 percent of connections, while email applications account for 19.3 percent, collaboration tools

for 17.2 percent, and the framework for 9%. Fingerprints such as device libraries, utilities like osquery and DropBox, and browsers had a 6 month or longer lifespan.

The methods described above were mostly effective with popular network traffic and network applications. Another paper that is related to our research is about the use of TLS in Android apps (Razaghpanah et al. 2017). The authors look at how TLS behaves on mobile platforms. They created the Lumen Android app, which is installed on a mobile device and intercepts TLS connections while collecting traffic statistics. The authors used the Lumen app to look at how 7.258 applications use TLS.

They looked at handshakes in relation to the app's TLS API and library. Their research was based on app protection and TLS flaws. They demonstrated that TLS libraries and OS APIs changed supported cipher suites across versions, causing TLS fingerprints to change (Razaghpanah et al. 2017). They also demonstrated that each TLS library and OS version had its own set of cipher suites. They created a database of fingerprints paired with OSes and libraries, observing the impact of major and minor OS or TLS library revisions on the fingerprint.

2.11 Conceptual Framework

The conceptual diagram, figure 2.5 defines the relevant variables for this research and maps out how they might relate to each other. JA3 fingerprint is the independent variable that affect the results of browser identification, which is the dependent variable. However, the moderating variables (captured network communication, JA3 algorithm and existing fingerprints) may alter the effect that the generated JA3 fingerprint may have on the results of browser identification. For instance, if the captured network communication does not have any browser generated packets, or it is corrupted, then browser identification will not be possible. Also, if the JA3 algorithm is not implemented correctly, or the existing fingerprints are not accurate, the browser identification will not be viable. Finally, the software environment is a mediating

variable in that it links the JA3 fingerprint generation including the comparison with existing fingerprints to successfully identify a browser, as a cause-and-effect relationship.

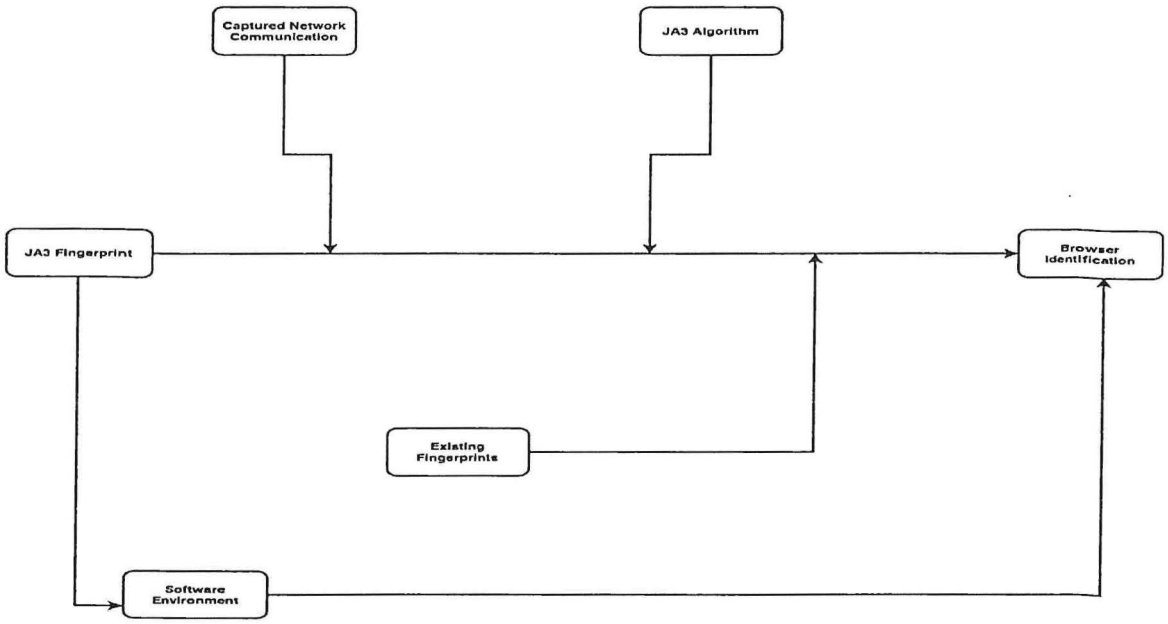


Figure 2.5 Conceptual Framework

2.12 Conclusion

TLS fingerprinting is not a new method, and its popularity is linked to the security investigations of various researchers who actively fingerprinted coupled customers based on cipher groups. Blake Anderson et al. (2018) pre-processed millions of TLS encoded flows and discovered a collection of observable data topographies from server hello and TLS client messages, such as TLS extensions, cipher suites, and TLS version, that they could use to detect malware. Since web communication is encrypted, fingerprinting techniques have been fine-tuned to depend on the unencrypted portion of network data collected during the safe channel establishment's initialization process. However, advanced techniques for extracting only the relevant portion of such data, which is sufficient for fingerprinting and forensic applications, are needed for analysis.

This literature review has established that the available solutions do not filter out packets not originating from the applications under analysis. There needs to be a way that ensures that the fingerprints are clean and reliable for purposes of identification. Also, the available JA3 implementations only give the fingerprints without indicating from which nodes it originated from, what time was the communication and which services were involved during the communication. This limits their value to forensic investigations.

Chapter 3 : Research Methodology

3.1 Introduction

The chapter discusses the research methodology that was applied. It focuses on detailing the approach taken in doing the research that influenced the development of the solution. The chapter provides a formal documentation for the phases that the system was subjected to during its development phases. It also describes the detailed objectives for every phase and the outcomes necessary from a phase before the next one can start. The chosen methodology (Agile Development Methodology) and subsequent steps were arrived at after carefully reviewing the literature and observing how other systems work.

The first two objectives (common vulnerabilities and threats in browsers, and a review existing fingerprinting techniques by other researchers used to identify web browsers) were addressed under the planning phase of the selected Agile methodology. The third objective which was to design, develop and test an improved JA3 based fingerprinting tool for browser identification in forensic scenarios based on clean and accurate fingerprints was addressed under the system design and implementation phases of the Agile Development Methodology. The final objective, which was to validate the effectiveness of the tool and its use in digital forensics was addressed under the testing phase of Development Methodology.

3.2 Research Design

The research design was partially theoretic in nature through literature review and the implementation of a tool used to validate the proposed solution. An evaluation of pertinent research materials on different browser fingerprinting techniques and existing implementations of JA3 hashes fingerprinting was adequate to answer the two first objectives of this study. The common vulnerabilities and threats in browser communication were identified and discussed in section 2.4. A review of existing fingerprinting techniques for browsers was also conducted and discussed in section 2.10. Thus, the theoretical research was envisioned to offer unfathomable theoretic understanding of this research area. This formed a foundation for the validation and development of the planned solution.

3.3 Software Development Methodology

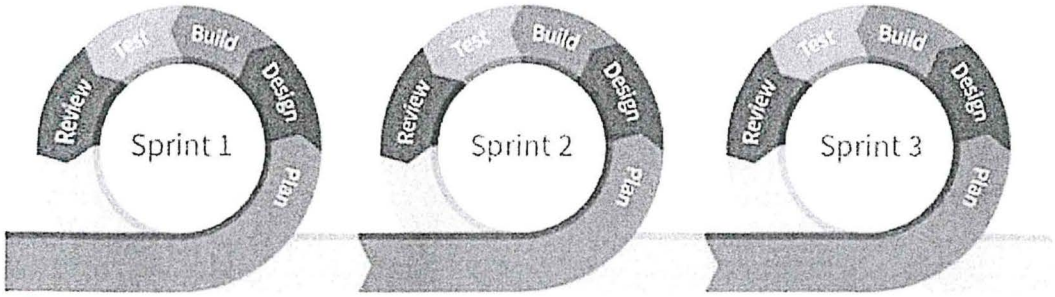


Figure 3.1 Agile Methodology (Maruping, Venkatesh & Agarwal, 2009)

Agile software development method takes into account quicker iteration and more regular delivery with resulting client input. Agile cycles permit release timetables and client input opportunities, this permits quicker and more controlled upgrades (Dybå and Dingsøyr, 2008). Figure 3.1 shows the steps that were followed in the research to achieve the set objectives for this dissertation, the initial step was planning which includes the collection of the planned project requirements and determining what it is ought to do or how it should do it. The second advance was design which incorporates defining the architecture and design of the system. These cycles addressed the third and fourth objectives of the study which were to design, develop and test a JA3 based fingerprinting tool for browsers, and to validate the effectiveness of the tool and its use in digital forensics.

System development is the third step which included execution of the system. Test and feedback were the fourth step which created space for system improvement. After feedback, the steps should be repeated if changes are being implemented on the system. The created application was tested autonomously during each advancement iteration. The data flow between the different components is also tested to ensure complete test coverage. Testing the application made sure that the needed functionalities are working as required.

The steps were divided into five distinct phases:

Planning: This step aimed to determine if there was a need to develop the system. The problem to be addressed was evaluated and ascertained if it was valid. Similar systems were examined to determine where they fall short. The primary stakeholders in the proposed system were identified. Finally, the scope of the system was defined.

Requirements Analysis: The aim of this step was to determine the feasibility of developing the system based on the data acquired in the planning phase. The system

requirements were formulated, to determine if there are adequate resources / knowhow to build the system.

Design: This phase defined the data, interfaces, architecture and modules for a system to satisfy specified requirements. Users interact with systems analysts and develop prototypes and that represent all system processes, outputs, and inputs process.

Building: This phase entailed programming and application development to convert the design models into a working application. Users continued to participate through suggesting changes or improvements as the developers work.

Testing: This was the final part of each iteration whereby the developer found out whether his code is working according to the requirements. The end goal was to determine if the developed system was ready for implementation.

3.3.1 System Planning

The study took necessary steps to ensure that the processes to be undertaken were outlined clearly to achieve the research objectives. This involved establishing how existing implementations of JA3 hash fingerprinting performed. Existing literature was reviewed where related work was examined, and valuable insights gathered on how to improve on the gaps identified.

The researcher examined various threats and vulnerabilities in web browser technologies. This led to a conclusion that largely the security threats relate to the way communication is packaged and transmitted through the network. End devices also play a big role in determining browser security.

In order determine the useability of the JA3 fingerprinting approach, the researcher reviewed existing JA3 implementation by other researchers which include Fast Go implementation, Malware Detection in encrypted browser communication, JA3 implementation in Android application among others.

The review identified various gaps which this research sought to address. In addition, it helped in coming up with a better approach in achieving the desired outcomes. The focus was on both evolving a high-level list of original necessities and setting the project's scope.

3.3.2 Requirements Analysis

In this step, data composed through scrutinizing systems in the planning stage was analysed. Gaps identified in the literature review were analysed to formulate the requirements of the

developed system. This included functional requirements, non-functional requirements and user requirements outlined in chapter 4 sub-section 4.2. The researcher figured out that the most feasible way to implement these requirements was to use bash scripting technology because it integrates and works seamlessly with network packet analysis tools and string manipulation tools. The nature of the data (network capture files) necessitated the use of packet analysis tool, whereby TSHARK is the most robust and suitable to work as a background process of the developed application. The requirement to format network fields in confoming with JA3 standard necessitated the integration of native Linux string manipulation tools such as Stream Editor.

A key requirement was how the dataset was to be accurately generated. Section 3.3.2.1 below explains how this requirement was addressed.

3.3.2.1 Data Collection

The dataset comprises TLS communication of Opera, Firefox, and Chrome web browsers. The browsers were running under four diverse operating systems including Debian, Ubuntu, Windows 10, Mac OS, and Kali Linux. These are the most common operating systems in use. The data collection process involved capturing traffic from various URLs picked randomly with the only condition being they are using TLS communication protocol, that is supporting secure communication via HTTPS. Bulk URL Opener browser extension was used to load ten multiple URLs at once. Wireshark was then opened and packet capturing through the relevant interface initiated. The URLs are listed below:

1. https://www.strathmore.edu/
2. https://www.homeworkmarket.com
3. https://www.paypal-mobilemoney.com/m-pesa
4. https://edition.cnn.com/
5. https://mail.google.com/mail/
6. https://selfservice.equitybankgroup.com/login
7. https://twitter.com/home
8. https://www.nasa.gov/
9. https://www.samsung.com/africa_en/
10. https://www.dell.com/

The captured data included irrelevant packets (noise) from communication not generated by the URLs under the scope. These were removed through DNS analysis.

3.3.2.2 Data Cleaning

Only packets to known destinations should be fingerprinted. A full packet capture entails traffic to and from a variety of destinations, including the operating system, background applications, and other running apps that communicate with remote services. Because this study is only concerned with browser TLS fingerprinting, traffic from / from other applications / services should be removed. Browser traffic, on the other hand, includes communication generated from advertisements services, browser plugins, and online services that are not started by a user. This should also be removed in order to maintain communication to user-initiated destinations. This will aid in ensuring that the fingerprints are clean and identifiable across different operating systems or browser versions. This has the added benefit of reducing the size of the dataset to be analysed, increasing tool efficiency.

This is accomplished by filtering traffic on the basis of known DNS destinations. Records are examined, and IP addresses are compared to known domain names. TLS Client Hello packets are extracted, and fingerprints are generated for the extracted IP addresses. This is achieved as explained in the steps below:

1: Get a set of URLS and open them using a browser to be fingerprinted. It would be more efficient to open all of them at once using a bulk URL opener browser plugin. Use Wireshark to capture and save the PCAP file.

2: Use TSHARK to extract DNS A records and response names. Match the two results so that each domain name is matched with the respective IP address, as shown in the commands in figure 3.2 below:

```
${TSHARK} -r "${INFILE}" -T fields -e dns.a > dns_a.csv
${TSHARK} -r "${INFILE}" -T fields -e dns.resp.name > dns_resp_name.csv
paste -d "," dns_a.csv dns_resp_name.csv | sort | uniq > dnsAllRcsUniq.csv
```

Figure 3.2 DNS Records Extraction Commands

The resulting CSV looks as shown in table 3.1 below:

Table 3.1 DNS Records

0.0.0.0	sync.search.spotxchange.com	sync.search-gtm.spotxchange.com.ak	null.spotxchange.com	
102.132.96.27	connect.facebook.net	scontent.xx.fbcdn.net		
102.132.96.35	www.facebook.com	star-mini.c10r.facebook.com		
102.132.96.8	cx.atdmt.com	atlas.c10r.facebook.com		
104.16.149.64	104.16.148.64	cdn.cookielaw.org	cdn.cookielaw.org	
104.16.15.243	104.16.14.243	104.16.12.243	104.16.11.243	104.16.1
104.16.18.94	104.16.19.94	cdnjs.cloudflare.com	cdnjs.cloudflare.com	
104.16.87.20	104.16.85.20	104.16.86.20	104.16.89.20	104.16.8
104.18.100.194	104.18.98.194	104.18.99.194	104.18.102.194	104.18.1
104.19.148.8	104.19.147.8	script.crazyegg.com	script.crazyegg.com	script.craz
104.20.43.26	104.20.44.26	www.homeworkmarket.com	www.homeworkmarket.com	
104.22.24.87	104.22.25.87	172.67.13.182	mwzeom.zeotap.co	mwzeom
104.244.42.65	twitter.com	twitter.com	twitter.com	twitter.cc
104.244.42.66	api.twitter.com	tpop-api.twitter.com	twitter.com	twitter.cc
104.244.42.69	t.co	t.co	t.co	t.co
104.75.197.180	cdn3.optimizely.com	cdn.optimizely.com.edgekey.net	e6640.x.akamaiedge.net	
104.75.203.185	stags.bluekai.com	tags.bluekai.com.edgekey.net	e9126.x.akamaiedge.net	
104.75.220.21	fcs.dellcdn.com	fcs.dellcdn.com.akadns.net	fcs.dellcdn.com-v2.	e19274.d
104.75.220.21	www.dell.com	www1.dell-cidr.akadns.net	cdn-www.dell.com-v	cdn-www
104.75.221.150	c.evidon.com	wildcard.evidon.com.edgekey.net	e12841.d.akamaiedge.net	
104.92.151.74	cdn.cnn.com	ion-ma.turner.com.edgekey.net	e12596.dscj.akamaiedge.net	
104.92.151.74	lightning.cnn.com	ion-ma.turner.com.edgekey.net	e12596.dscj.akamaiedge.net	
104.92.152.224	6852bd0b.akstat.io	wildcard46.akstat.io.edgekey.net	e4518.dscx.akamaiedge.net	

3: Copy the extracted IP addresses and insert them inside the TSHARK filter, as shown in figure 3.3:

```
${TSHARK} -r "${INFILE}" -T fields \
    -e ip.dst \
    -e tls.handshake.version \
    -e tls.handshake.ciphersuite \
    -e tls.handshake.extension.type \
    -e tls.handshake.extensions_supported_group \
    -e tls.handshake.extensions_ec_point_format \
    -R "tls.handshake.type==1 and ip.addr in {198.57.179.99 147.229.2.82 147.229.2.90
       157.240.30.35 184.51.8.147 2.18.68.206 52.30.88.10 34.253.101.66 34.249.165.210
       52.48.145.94 34.255.235.176 54.171.79.102 52.19.22.175 34.248.108.242 52.45.117.194
       52.205.228.94 34.232.204.33 34.232.6.198    52.5.59.12 54.174.68.95 52.7.248.149
       52.7.114.31 52.46.135.211 52.46.141.49 54.239.26.255 99.86.240.33 99.86.241.241
       18.205.93.1 18.205.93.2 18.205.93.0 192.124.249.12 192.124.249.5}" -2
```

Figure 3.3 TSHARK Fields and Filter

Table 3.2 describes the data that was captured in detail:

Table 3.2 Browser Communication Dataset Parameters

			All IP Addresses			Filtered IP Addresses				
Browser	**Operating System**	**BroswerVersion**	**Packets #**	**tls.hello Packets**	**Unique Fingerprints**	**Packets #**	**tls.hello Packets**	**Unique Fingerprints**	**File Size (Bytes)**	**Timestamp**
Chrome	Debian	90.0.4430.93	9803	169	263	2190	41	41	31980396	4/27/21 13:34
Chrome	Kali Linux	88.0.4324.96	40395	301	265	10997	54	54	25349708	4/26/21 7:14
Chrome	Mac OS	90.0.4430.85	12541	227	145	1758	29	29	21230200	4/27/21 12:13
Chrome	windows	90.0.4430.85	52495	442	295	7861	28	28	28107884	4/27/21 11:26
Firefox2	Kali Linux	78.3.0esr	33171	303	3	7252	31	3	9705480	4/26/21 3:39
Firefox	Debian	78.8.0esr	55270	478	4	10932	48	3	25697996	4/27/21 13:29
Firefox	Kali Linux	78.3.0esr	38039	322	4	7682	21	3	8336400	4/27/21 10:19
Firefox	Mac OS	85	35542	313	5	7935	36	3	35870748	4/26/21 3:20
Firefox	Mac OS	88	43243	370	5	8922	54	2	40290856	4/26/21 3:28
Firefox	windows	88.0	19157	296	2	3816	68	2	32599468	4/27/21 11:21
Opera	Debian	75.0.3969.218	34200	335	314	4814	36	36	31437964	4/27/21 13:44
Opera	Kali Linux	75.3969.218	47431	79	176	2937	40	40	26912500	4/27/21 11:11
Opera	Mac OS	75.0	31665	353	324	7548	38	38	31410060	4/27/21 12:16
Opera	windows	75.0.3969.243	19947	276	76	3926	15	15	47930456	4/27/21 11:05

3.3.3 System Design

The design step involved coming up with the application designs based on the requirements gathered. Functional Oriented Design (FOD) techniques was used to refine the functional requirements identified during system analysis and to decompose the design into sets of interacting units where each unit has clearly defined functions (Gancheva et al., 2020). Unified Modelling Language (UML) was used to specify, visualise, construct, and document the artifacts of the system.

The stage included the creation of the system architecture which illustrated the input, process, and output of the application. The input to the system is captured PCAP network files that are read to extract relevant TLS fields. The process shows how the extracted fields are processed and fingerprints generated. The output is what the user gets after running the application.

Also, sequence diagram was designed to illustrate the major interactions taking place between the various subsystems and actors in the developed prototype. Use case diagram was used to show required usages of a system under design or analysis and to capture what the system is supposed to do. It was used to illustrates the major interactions taking place between the various subsystems and actors in the developed prototype. An entity relationship diagram was designed to show the blueprint of the database that was implemented for the browser fingerprinting tool.

These artifacts will be developed by use of the Draw.io tool.

3.3.4 System Building

The development tools chosen for this study have different components which help in attaining certain functionalities. **Shell scripting** was mainly used to develop the tool. It encompassed Unix string manipulation tools which helps in formatting extracted packet fields into strings and parse them to prepare for fingerprint generation / hashing. These Unix utilities include SED and AWK. TSHARK commands were used to extract the relevant fields from the Client Hello packets. MySQL is the relational database management system that was used to store fingerprints and for runtime querying during browser identification. MySQL was preferred because it is open source and cross-platform.

Due to the technical nature of shell scripts, an approach to create a user-friendly interface was conceptualised and implemented using Werkzeug, a library that provides a simple web interface to receive user requests and display results of the script's analysis in a nice interface. The library is implemented using Flask, which is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has

no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

3.3.5 System Testing

Unit testing was used to test the functionality of the system. Unit testing is described as a component of Test-Driven Development (TDD), or a practical methodology that adopts a fastidious strategy to building a product by method for consistent testing and adjustment. In line with test driven development, fizzling unit tests were written first (Moe, 2019). At that point the researcher composed a code and refactor the application until the test passes. TDD brought about an unambiguous and predictable code base.

Secure coding test was conducted to ensure that the code follows the recommended coding principles to avoid vulnerabilities. ShellCheck framework was used to point out and clarify typical beginner's syntax issues that cause a shell to give cryptic error messages, point out and clarify typical intermediate level semantic problems that cause a shell to behave strangely and counter-intuitively, and to point out subtle caveats, corner cases and pitfalls that may cause an advanced user's otherwise working script to fail under future circumstances.

3.3.6 System Review

The objective of system review was to determine whether the system was built in conformity with professional standards and to ensure the user complying with the system usage. In this case peer review was used. The researcher approached professionals who are in the information security industry to review the system.

3.4 Research Quality Aspects

Research quality aspects describes the extent to which the research will be conducted correctly. Validity and reliability were utilized to check the quality features.

3.4.1 Validity

Kimberlin & Winterstein (2008) argues validity determines if a study is accurately and truthfully conducted, and if what it was intended to measure was achieved. Repeatability validity was carried out to ensure that similar results are produced on identical test items in the same environment by the same operator using the same equipment within short intervals of time. Also, reproducibility validity was carried out to ensure that similar results are obtained when using the same method on identical test items in different laboratories with different operators utilizing different equipment.

3.4.2 Reliability

Reliability describes the degree to which there is consistency in results over a given time and precise illustration of the collective population under study. If the results of a research may be simulated in a similar methodology, at that point the research method is considered reliable (Kimberlin & Winterstein, 2008).

3.5 Conclusion

The chapter highlights the methodology that was adopted in conducting the study. The data collection methods and analysis were suggested to guide the researcher in generating inferences from the evidence collected and hence answer the research questions. The tools employed were also highlighted.

Chapter 4 : System Design and Architecture

4.1 Overview

This chapter presents a detailed view of how the browser fingerprinting tool was designed. The researcher used the use case diagram, data flow diagrams, system sequence diagram and data modelling to illustrate different aspects of the browser fingerprinting tool.

4.2 System Requirements and Analysis

For the system to conduct browser fingerprinting, it is mandatory to develop the structure considering the malicious techniques present in phishing emails. It is thus from the requirements analysis that one identifies the appropriate resources that will satisfy the needs and requirements based on the objectives set. According to Enfocus (Solutions, 2017), system requirements is an important process as it enables the proposed system to capture the existing gaps. It also enables easy management of the end solution as well as ensuring that the product fits the organization structure.

To arrive at the requirements, the researcher applied two techniques, namely, documents analysis and prototyping. In document analysis, documentation of existing systems was reviewed. This helped in conducting gap analysis and scoping of the project. Bits of information often buried in existing documents assisted the researcher to ask questions as part of validating requirement completeness. Finally, prototyping was used to gather preliminary requirements used to build an initial version of the solution. It was then shown to the supervisor, who then gave the researcher additional requirements who changed the application and cycle around with the supervisor again. This repetitive process continued until the software met the needs.

The system requirements stem from the user requirements that outline what the users expect to accomplish with the system.

4.2.1 User Requirements

The user requirements include:

1. Inputting an unknown network capture file to the system for analysis.
2. Choosing how to analyse the network capture file withing the application, for instance, determine the browser identity, to check the extended fields only, or even to compute fingerprints only.
3. To review the results of the analysis in a friendly interface / data representation such as a table.

However, system requirements requires that each need is broken down and defined clearly. A review of the flow of all events as per the interaction of each requirement is then performed, so as that decision making on whether the requirement is needed, becomes easier. System requirements analysis enables creation of a development framework thus providing a good basis for all future works.

The research categorises system requirements into two, namely: Functional and Non-Functional requirements (Solutions, 2017). Functional requirements entail the functions that the system must do or deliver. They are the desired functionality that the client expects from the proposed system. A functional requirement also describes the interaction between the system and its environment.

4.2.2 Functional Requirements

The functional requirements of the proposed algorithm are as follows:

1. Load network capture file and pass it to bash variables for feature extraction and analysis.
2. Read and extract the relevant fields (such as timestamp, source IP addresses, and source and destination ports) from a previously saved capture file.
3. Manipulate and format extracted fields to satisfy JA3 requirements.
4. Hash the formatted strings using MD5 message-digest algorithm, producing a 128-bit hash value.
5. Match each fingerprint to a pre-calculated set and fetch the corresponding application name.
6. Display packet fields with associated applications and identifying metadata.

4.2.3 Non-Functional Requirements

The Non-Functional requirements of the proposed algorithm are as follows:

1. The system should be as accurate as possible to increase reliability.
2. The system should have access to relevant external software that it relies on to ensure effectiveness in its performance.
3. The system should not require a lot of maintenance.
4. The system should take very little time possible to perform fingerprinting.
5. The system should be accurate, readily available while not interfering with the overall computer performance experience to increase the user level of trust.

6. The system should be able to perform and achieve its requirements based on the variety of checks implemented therein.
7. In terms of the user's perspective, the system should work seamlessly. The user should not have a clue of any undergoing fingerprinting computations.

4.3 System Architecture

The shell script implementation includes a number of steps aimed at extending JA3 functionality by allowing it to identify the web browsers used in a given communication. PCAP files are read and analysed using a variety of command-line tools in order to determine the browser's identity. Figure 4.1 illustrates the various components of the system.

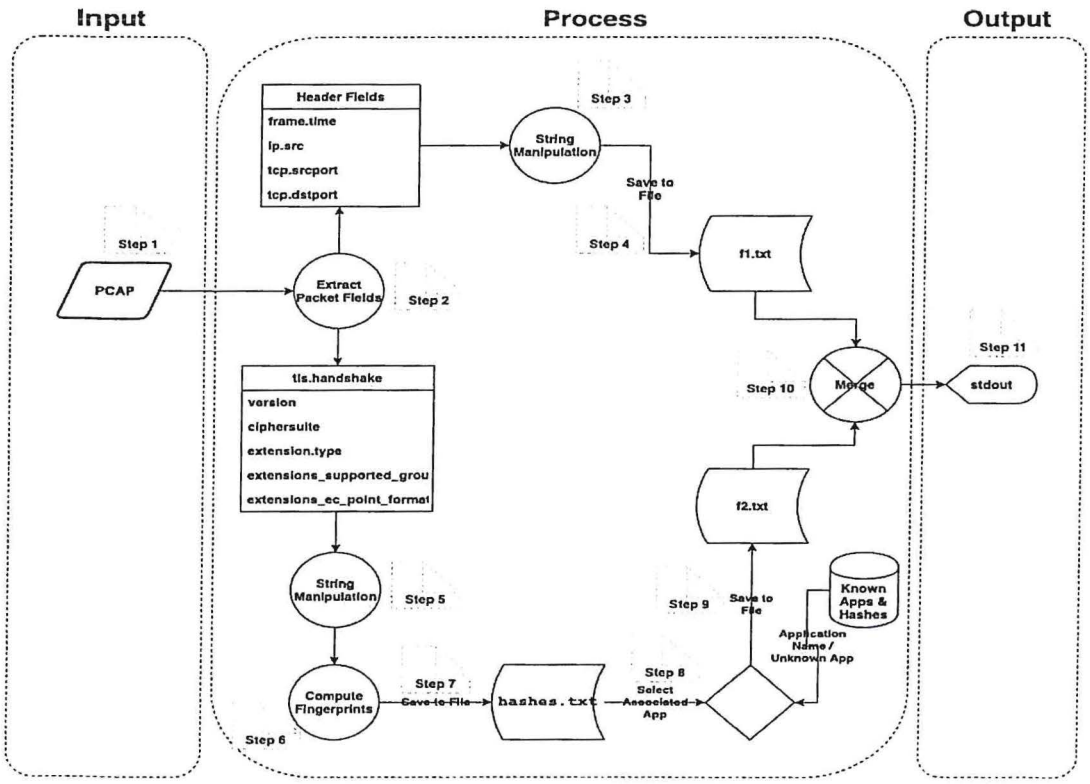


Figure 4.1 ClassifyJA3 System Architecture

Input

Step 1: Load PCAP file from file system to the application for processing.

Process

Step 2: Extract Packet Fields

This level makes use of TSHARK tool in extracting packet fields from network PCAP file. The fields include:

A. frame.time - Represents frame arrival time

B. ip.src - Represents IPv4 source address

C. tcp.srcport – Represents TCP Source Port

D. tcp.dstport - Represents TCP Destination Port

E. tls.handshake.version

F. tls.handshake.ciphersuite

G. tls.handshake.extension.type

H. tls.handshake.extensions_supported_group

I. tls.handshake.extensions_ec_point_format

Steps 3 - 5: String Manipulation

The extracted fields are then processed using awk tool and saved into a text file for subsequent processing.

Step 6 - 7: Compute Fingerprints

The following step is computing the JA3 fingerprints, which are evaluated from the following tls.handshake fields:

A. version – Represents TLS version,

B. ciphersuite - Represents supported TLS version cipher suite,

C. extension.type - Represents TLS extension type,

D. extensions_supported_group - Represents TLS supported group, and

E. extensions_ec_point_format - Represents TLS elliptic curve point format.

These packet fields are extracted from unencrypted Client Hello messages. According to research, different applications generate these fields in different ways. The fields are processed using string manipulation tools such as the AWK tool and stream editor (SED) to generate a JA3 equivalent fingerprint. This includes converting hexadecimal fields to decimals and replacing all commas with dashes. Each line is then MD5 hashed, and the output of fixed length hashes is saved to a text file called hashes.txt. The main reason for using MD5 is to keep fingerprints short. Furthermore, due to the small size of the data set, hash collisions are not an issue.

Step 8 - 9: Retrieve Identified App

The fingerprints computed are then compared with already known fingerprints stored in a database. The script compares each generated fingerprint from unknown PCAP with the know database entries via a recursive SQL query. The result is the browser name if match is found, otherwise the query returns "UknownApp" value and save the output in a text file.

Step 10: Merge

This level joins the contents of the text files generated above to show which we browsers were used to generate each packet and output is displayed to the user.

Output

Step 11: Display Results

The last step is to show the results of the identified application with its associated identifying information.

4.4 System Design

This section discusses the various design artifacts based on Unified Modelling Language (UML). These include a sequence diagram, use case with its description, entity relationship diagram and a database schema.

4.4.1 Use Case

Figure 4.2 is a use case diagram that illustrates the major interactions taking place between the various subsystems and actors in the developed prototype.

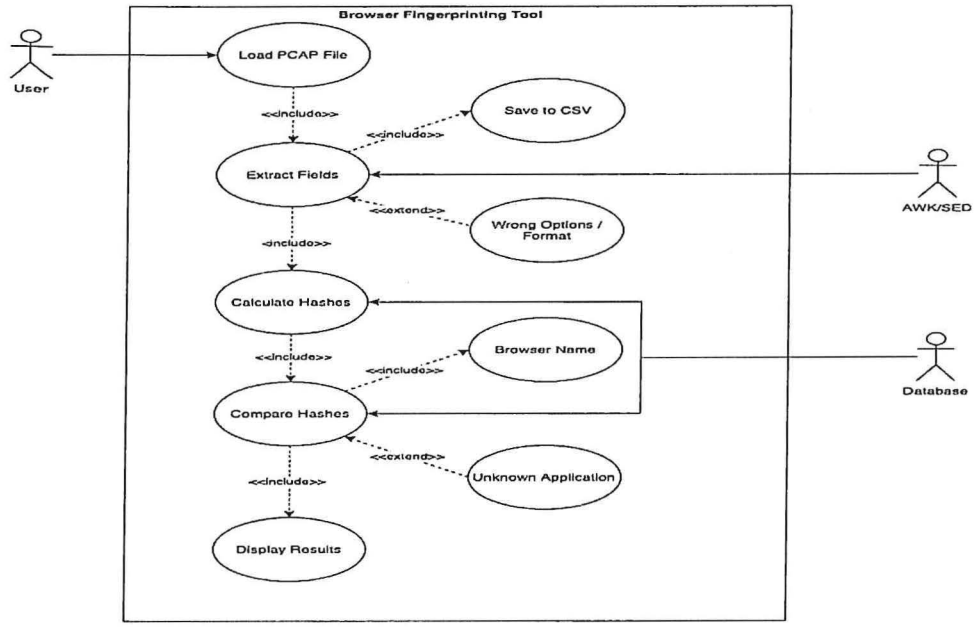


Figure 4.2 Use case diagram

Table 4.1: Load PCAP File Use Case

Use Case Name:	**Load PCAP File**
Description:	In this use case, the user runs the shell script with a PCAP file as a parameter, which is then loaded into a bash variable $1
Primary Actor:	User

Secondary Actor:	None
Include Use Cases:	Extract fields Generate hashes Compare hashes
Extend Use Cases:	None
Preconditions:	The input should be a PCAP network capture file
Post Conditions:	The tool identifies the browser that has been used during the communication
Main Flow:	Run the shell script with two variables – switch *e* and PCAP file The system includes identification information for each fingerprint
Alternative Flows:	Run the shell script with two variables – switch *f* and PCAP file The system excludes the additional identification information and displays only browser name and its associated fingerprint

Appendix A shows the use cases for extract fields, calculate hashes, compare hashes, and display results.

4.4.2 Sequence Diagram

Figure 4.3 is a use sequence diagram that illustrates the major interactions taking place between the various subsystems and actors in the developed prototype.

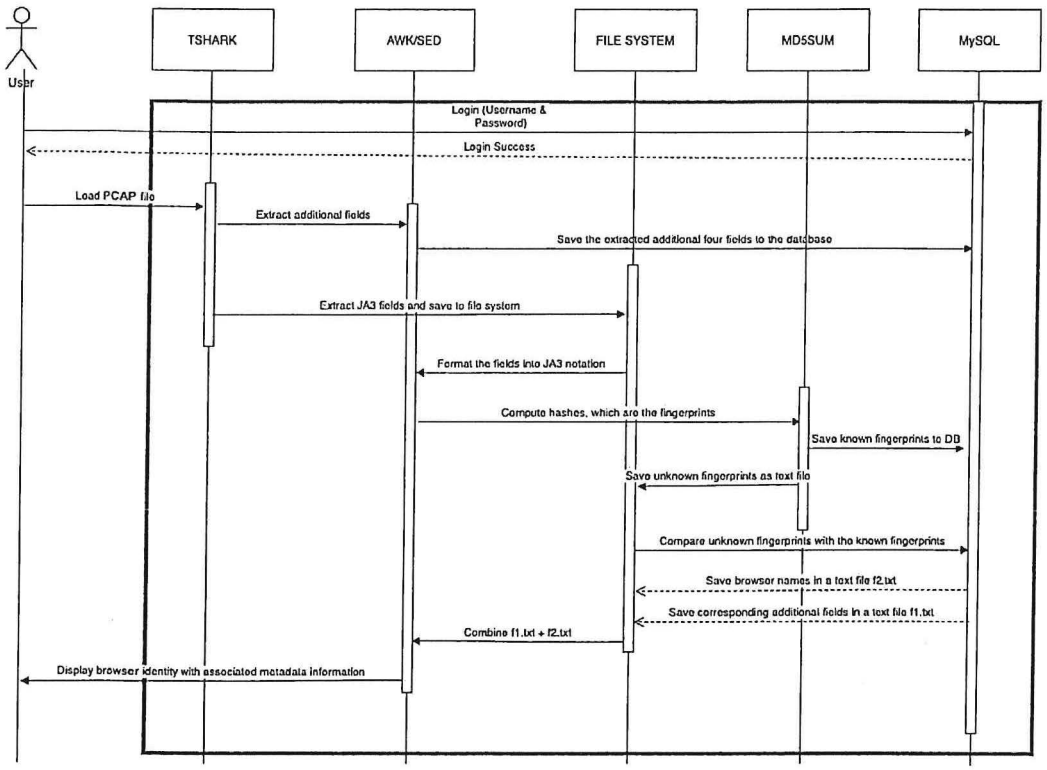


Figure 4.3 Sequence Diagram

Login: User authenticates via the web interface to start using the tool. A registered username and password are required for login.

Input network capture file: A user executes the application which requires a PCAP file as a parameter. The file is loaded and initialised to bash variable $1. File read error is checked before any further processing to ensure that it is a valid network capture file.

Extract JA3 client hello fields: The application uses TSHARK internally to process PCAP files. Packet fields relating to time, source IP address source and destination ports are extracted.

Save text file one: The application also integrates string manipulation tools that are used to format the extracted fields in the previous step. These are then saved into a text file for further analysis.

Extract JA3 fields: Now the fields required for fingerprint generation using JA3 algorithm are extracted. These includes client hello header fields identified in the earlier section.

Generate fingerprints: Hash the saved formatted fields texts line by line using MD5 message digest to produce a fixed length 128-bit string and save into a database.

Compare generated fingerprints with the saved known fingerprints: A database that already has verified fingerprints for specific browsers is used to compare the new generated fingerprints to match identity of browser names, If the fingerprint down not match any database

entry, the SQL query returns "UnkownApp" results. The query results are saved into another temporary file.

Combine saved text files containing provisional results: The text file that contains the four packet fields from the communication, is programmatically combined with the one with database query results.

Display fields with associated browser name: A combination of the text files from the previous step is displayed in the terminal to reveal the identity of the web browser that was used in the communication.

4.4.3 Entity Relationship Diagram

Figure 4.4 is an entity relationship diagram that shows the blueprint of the database that was implemented for the browser fingerprinting tool. This simple schema shows the relationship between Ja3 table header fields table. The primary key for both tables is frame_no, which helps to link the JA3 hashes with their respective descriptive information such as timestamp, source IP address and port numbers.

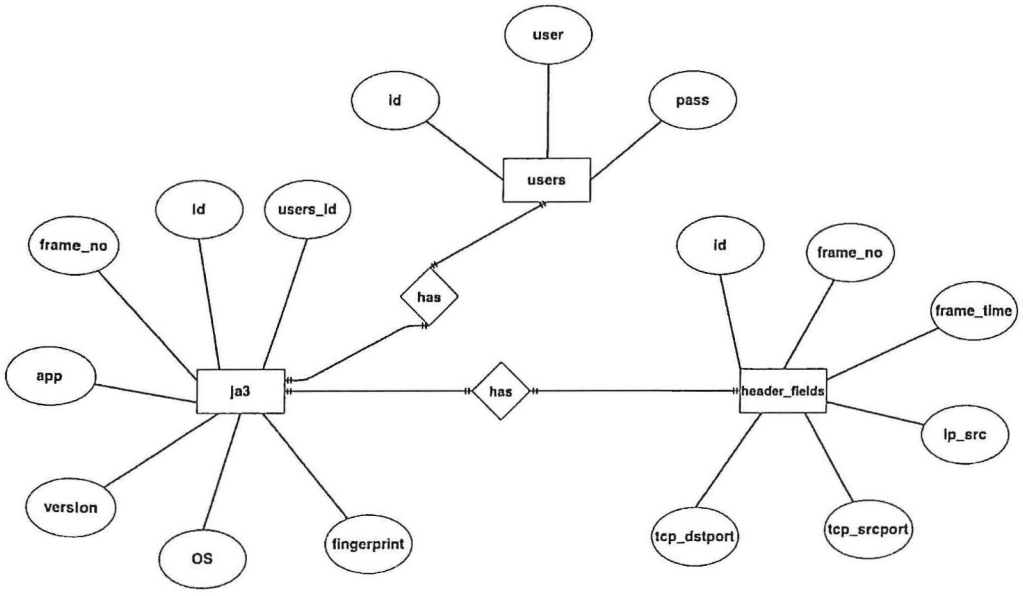


Figure 4.4 Entity Relationship Diagram

4.4.4 Database Schema

Figure 4.5 is a database schema describing two entities (tables) and their relationship. The relationship between ja3 table and header_fields table is one to one as a single JA3 fingerprint can only have one frame number, timestamp, source IP address, tcp source port and tcp destination port.

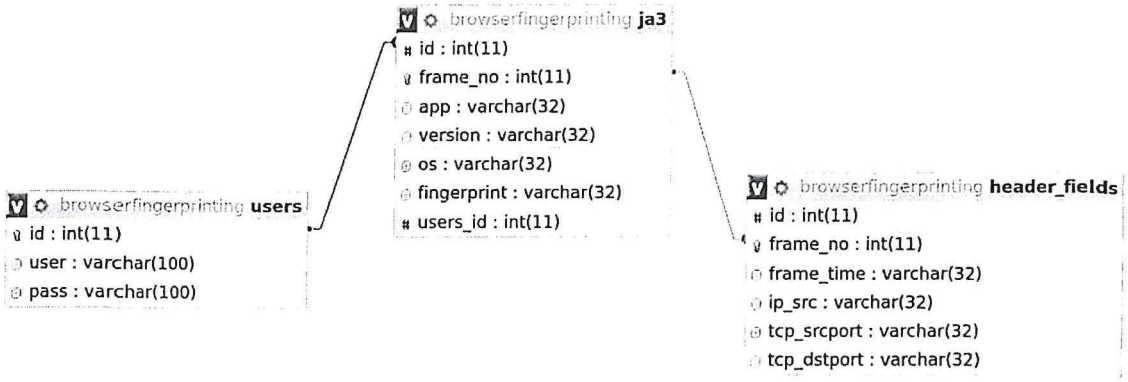


Figure 4.5 Database Schema

4.5 Security Design of The Tool

Secure coding practices were applied during the implementation of the system. Several measures were taken into consideration to ensure that the tool does not have bugs. First, passing filenames and other positional arguments to commands was done with standard user privileges, which can be escalated using sudo command. This avoids executing commands as root, which may be misused by an adversary.

Temporary files are used for runtime execution to ensure fast and efficient storage usage. This avoids overreliance on the database system and saves on storage. Systems that require a lot of storage usually crash in environments with limited resources. This could be viewed as a potential denial of service (DoS) issue. Finally, the researcher used ShellCheck to check for bugs. The ShellCheck linter automatically catches several common coding mistakes. It was run it regularly, ideally with integration into the editor and the test suite and address all its diagnostics. Because this is an open-source project, the fingerprints will be shared with the public online community for research purposes. This way, other researchers and cybersecurity partitioners can learn and test the tool.

4.6 Command Line Interface (CLI) User Interaction and Graphical User Interface (GUI)

The system allows users to interact with the system using both CLI and GUI. The command line interface enables a user to execute the browser fingerprinting tool with more command and control than the use of a graphical user interface, but this requires some level of experience and technical knowledge in the use of command line-based systems. Not all users will be comfortable using the CLI thus the need of a GUI.

4.6.1 CLI

The Terminal is the most common CLI for Unix environment. Since the tool is based on bash script, the user executes it via a terminal, passing relevant commands and options which can be viewed from the help menu. The syntax is as follows: #./brwFing.sh <PCAP file> [--full | --comp | --ext] where –full option generates fingerprints using JA3 fields, --ext option generates fingerprints with additional extensions and –comp compares new fingerprints with known fingerprints and display comprehensive information. The <PCAP file> is the network capture file whose browser identity is being sought.

4.6.2 GUI Wireframes

The following are wireframes of a web-based application which allows the user to interact with the scripts in a more user-friendly manner.

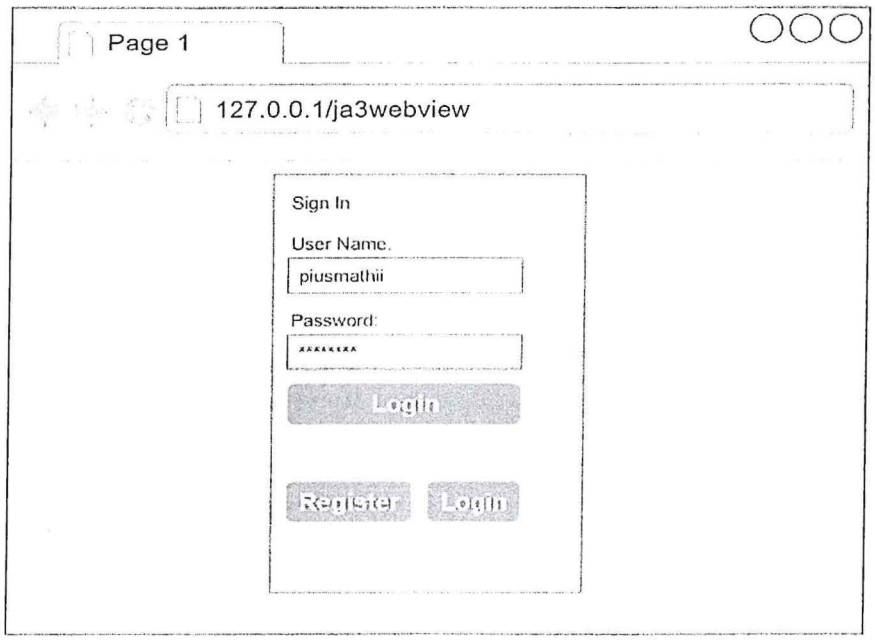


Figure 4.6 Login page wireframe

Figure 4.6 shows a login page wireframe. If the user is new to the system, he or she is required to register else she should login.

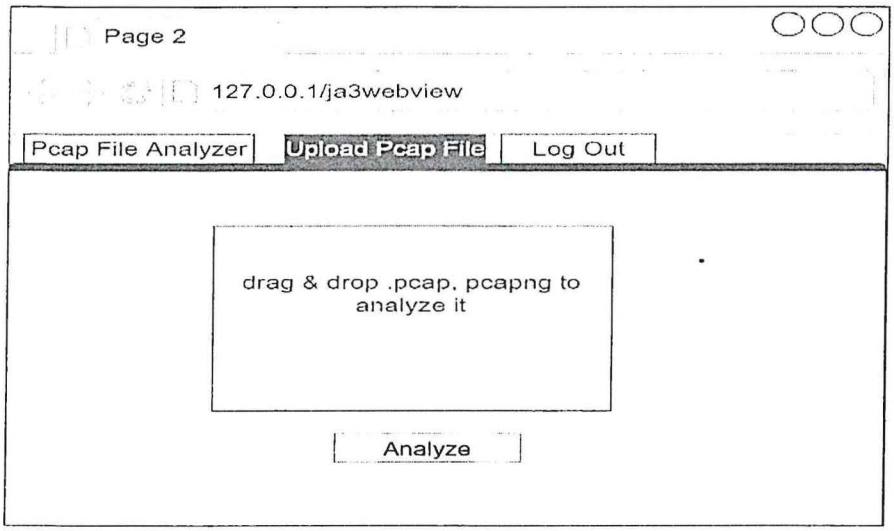


Figure 4.7 Homepage/ Pcap file upload page wireframe

Figure 4.8 shows the landing page wireframe. It allows the user to upload or drag and drop pcap files for analyses. It also enables the user to log out.

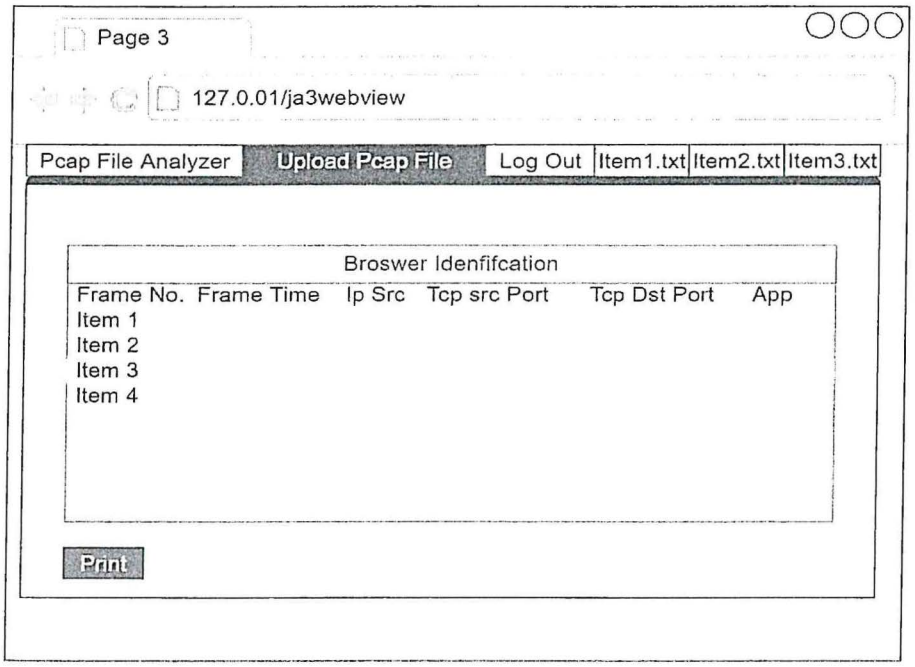


Figure 4.8 Results Page Wireframe

Figure 4.9 shows the results page wireframe. The results are displayed after the bash scripts are run by the python tool. The browser will be identified as a result of comparison between known and unknown fingerprints.

Chapter 5 : System Implementation and Testing

5.1 Overview

This chapter discusses the system implementation procedures. This includes the implementation environment, data preparation, main functions of the prototype and system testing which describes the kind of tests carried out and the intended results. The relevant screenshots of the prototype are provided.

5.2 Software Environment

The system is a shell script application that runs on the client computer. It is implemented on Debian which is a Linux distribution. The system was built in different phases using several programming / scripting languages and software that include:

5.2.1 Bash Programming

Bash Programming is the art of creating shell scripts designed to be run by the Unix shell, a command-line interpreter. The system uses scripts to set up the environment, run processes, and audit the system through log maintenance.

5.2.2 Python

Python3 has been used to build an application that reads and interprets the bash scripts to allow an interface with the graphical user interface via the browser. The python app uses the Werkzeug server and Flask to deploy the web-based application.

5.2.3 Stream Editor and AWK

Stream Editor (SED) is a stream editor used to perform basic text transformations on an input stream (a file or input from a pipeline). Using this tool allows to Can do insertion, deletion, search and replace(substitution). SED command in Unix supports regular expression which allows it to perform complex pattern matching. The Aho, Weinberger & Kernighan (AWK) language is a data-driven scripting language consisting of a set of actions to be taken against streams of textual data – either run directly on files or used as part of a pipeline – for purposes of extracting or transforming text, such as producing formatted reports. The language extensively uses the string datatype, associative arrays (that is, arrays indexed by key strings), and regular expressions.

5.2.4 TShark

TShark is a terminal oriented version of Wireshark designed for capturing and displaying packets when an interactive user interface isn't necessary or available. This tool was used TShark to read packets from a previously saved capture file, either printing a decoded form of those packets to the standard output or writing the packets to a file. TShark's native capture file format is **pcapng** format, which is also the format used by wireshark and various other tools.

5.2.5 ShellCheck

ShellCheck is a GPLv3 static analysis tool that gives warnings and suggestions for bash/sh shell scripts. In this research, it is used for code analysis to point out and clarify typical syntax issues that cause a shell to give cryptic error messages, to point out and clarify typical intermediate level semantic problems that cause a shell to behave strangely and counter intuitively and to point out subtle caveats, corner cases and pitfalls that may cause an advanced user's otherwise working script to fail under future circumstances.

5.2.5 HTML/CSS

HTML (Hypertext Markup Language) have been used to provide the structure of the web interface pages and CSS (Cascading Style Sheets) has been used to enhance the visual layout accordingly. The HTML and CSS are used to render the interface of the web app since the app is built using Python.

5.2.6 Xampp

It is a stack of software, which includes Apache distributions, MySQL database, and interpreters for scripts written in the PHP and Perl programming languages. In this reaserch, Xampp has been used to develop and test web interface locally.

5.2.7 Werkzeug Server

Werkzeug is a collection of libraries that can be used to create a WSGI (Web Server Gateway Interface) compatible web application in Python. A WSGI server is necessary for Python web applications since a web server cannot communicate directly with Python. WSGI is an interface between a web server and a Python-based web application.

5.3.8 Flask

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form

validation, or any other components where pre-existing third-party libraries provide common functions. Flask wraps Werkzeug, using it to handle the details of WSGI while providing more structure and patterns for defining powerful applications.

5.3 Hardware Environment

The number of resources this tool needs depends on your environment and on the size of the capture file you are analysing. Larger capture files will require more memory and disk space. The values below should be fine for small to medium-sized capture files no more than a few hundred MB:

1. Central Processing Unit: Any modern 64-bit AMD64/x86-64 or 32-bit x86 processor
2. Random Access Memory: 500 MB available RAM. Larger capture files require more RAM. +

   Disk Space: 500 MB available disk space. Capture files require additional disk space.
3. Network Interface: Any standard ethernet network card or 802.11 interface for Wi-Fi.

Network environment during PCAP capturing depend on the device that the browser is installed. Laptops using wireless network connection will capture packets via the Wi-Fi (en1) network interface while desktop computers without wireless connectivity will use Ethernet (en0) network interface. Hosts running on virtual hosting environment such as VirtualBox or VMWare will often capture via Ethernet (en0) since that is the default mapped interface irrespective of the interface used by the host operating system.

5.4 Set-up and Configuration

The tool runs in a Linux environment. The steps below show how to install dependencies and run the tool.

5.4.1 Installing Dependencies

Execute the commands below step by step. You can simply copy the commands and paste into the command line interface.

sudo apt update
sudo apt-get install tshark
sudo apt-get install -y gawk

pip install -r requirements.txt (The requirements.txt is provided with rest of development files)

Check the system logs to confirm that there are no related errors. Finally install an SQL database (such as Maria DB), create a database named browsenoh d*ingerprinting with three tabls named users, ja3 and header_fields.

5.4.2 Data Cleaning and Preparation

The dataset comprises TLS communication of Opera, Firefox, and Chrome web browsers. The browsers were running under four diverse operating systems including Debian, Ubuntu, Windows 10, Mac OS, and Kali Linux. Only packets to known destinations should be fingerprinted. This is accomplished by filtering traffic based on known DNS destinations. Records are examined, and IP addresses are compared to known domain names. TLS Client Hello packets are extracted, and fingerprints are generated for the extracted IP addresses. This is achieved as explained in the steps below:

1: Get a set of URLS and open them using a browser to be fingerprinted. It would be more efficient to open all of them at once using a bulk URL opener browser plugin. Use Wireshark to capture and save the PCAP file.

2: Use TSHARK to extract DNS A records and response names. Match the two results so that each domain name is matched with the respective IP address, as shown in the commands in figure 5.1 below:

```
${TSHARK} -r "${INFILE}" -T fields -e dns.a > dns_a.csv
${TSHARK} -r "${INFILE}" -T fields -e dns.resp.name > dns_resp_name.csv
paste -d "," dns_a.csv dns_resp_name.csv | sort | uniq > dnsAllRcsUniq.csv
```

Figure 5.1 DNS Records Extraction Commands

The resulting CSV looks as shown in table 5.1 below:

Table 5.1 DNS Records

104.20.43.26	104.20.44.26	www.homeworkmarket.com	www.homeworkmarket.com
104.22.24.87	104.22.25.87	172.67.13.182	mwzeom.zeotap.co mwzeom
104.244.42.65	twitter.com	twitter.com	twitter.com twitter.cc
104.244.42.66	api.twitter.com	tpop-api.twitter.com	twitter.com twitter.cc
104.244.42.69	t.co	t.co	t.co t.co
104.75.197.180	cdn3.optimizely.com	cdn.optimizely.com.edgekey.net	e6640.x.akamaiedge.net
104.75.203.185	stags.bluekai.com	tags.bluekai.com.edgekey.net	e9126.x.akamaiedge.net
104.75.220.21	fcs.dellcdn.com	fcs.dellcdn.com.akadns.net	fcs.dellcdn.com-v2. e19274.d
104.75.220.21	www.dell.com	www1.dell-cidr.akadns.net	cdn-www.dell.com-\ cdn-www
104.75.221.150	c.evidon.com	wildcard.evidon.com.edgekey.net	e12841.d.akamaiedge.net
104.92.151.74	cdn.cnn.com	ion-ma.turner.com.edgekey.net	e12596.dscj.akamaiedge.net
104.92.151.74	lightning.cnn.com	ion-ma.turner.com.edgekey.net	e12596.dscj.akamaiedge.net
104.92.152.224	6852bd0b.akstat.io	wildcard46.akstat.io.edgekey.net	e4518.dscx.akamaiedge.net
104.92.152.224	s.go-mpulse.net	ip46.go-mpulse.net.edgekey.net	e4518.dscx.akamaiedge.net

3: Copy the extracted IP addresses and insert them inside the TSHARK filter, as shown in figure 5.2:

```
${TSHARK} -r "${INFILE}" -T fields \
    -e ip.dst \
    -e tls.handshake.version \
    -e tls.handshake.ciphersuite \
    -e tls.handshake.extension.type \
    -e tls.handshake.extensions_supported_group \
    -e tls.handshake.extensions_ec_point_format \
    -R "tls.handshake.type==1 and ip.addr in {198.57.179.99 147.229.2.82 147.229.2.90
        157.240.30.35 184.51.8.147 2.18.68.206 52.30.88.10 34.253.101.66 34.249.165.210
        52.48.145.94 34.255.235.176 54.171.79.102 52.19.22.175 34.248.108.242 52.45.117.194
        52.205.228.94 34.232.204.33 34.232.6.198    52.5.59.12 54.174.68.95 52.7.248.149
        52.7.114.31 52.46.135.211 52.46.141.49 54.239.26.255 99.86.240.33 99.86.241.241
        18.205.93.1 18.205.93.2 18.205.93.0 192.124.249.12 192.124.249.5}" -2
```

Figure 5.2 TSHARK Fields and Filter

5.4.3 Running the Tool

The tool can be run in two ways, executing scripts via command line or using a browser to run the scripts. The both options cater for the needs of diverse user preferences based on technical knowhow and the level of control required.

The native way to run the tool is by the user of command line interpreter. To do this, first change directory <cd> to the directory containing the source code and type sudo chmod +x <appName.sh> to make the script executable. Create a directory named *results* using the command mkdir results. This is where computed data and runtime files will be stored. Execute the script as in ./appName.sh to display help information with various options that you can use to run the tool. See figure 5.3.

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~/Documents]
└─$ ./brwFing.sh
Synopsis: This tool processes a PCAP file with TLS communication
 from web browsers and creates fingerprints for classification
Author: (c) 2021, Pius Muisyo Mathii, Strathmore University
Syntax: ./brwFing.sh <PCAP file> [--full | --ext | --comp]

Options:
 -f | --full, generate fingerprints using JA3 fields
 -x | --ext, generate fingerptints with additional extentsions
 -c | --comp, compare new fingerprints with known fingerprints a
nd display comprehensive information
```

Figure 5.3 Help Function Display

To run the tool via the web browser, run the python script to create a web server instance that can be accessed via a web browser. Follow the steps below:

1. Navigate to the main tool folder

2. Open your terminal
3. Run python app.py

Sample output

** Serving Flask app 'app' (lazy loading)*

** Environment: production*

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

** Debug mode: on*

** Running on http://127.0.0.1:5300/ (Press CTRL+C to quit)*

** Restarting with stat*

** Debugger is active!*

** Debugger PIN: 505-595-599*

Open the app link (*http://127.0.0.1:5300)* on your browser to use the app. A user will be prompted to sign in or register to use the tool, as shown in figure 5.4 below:

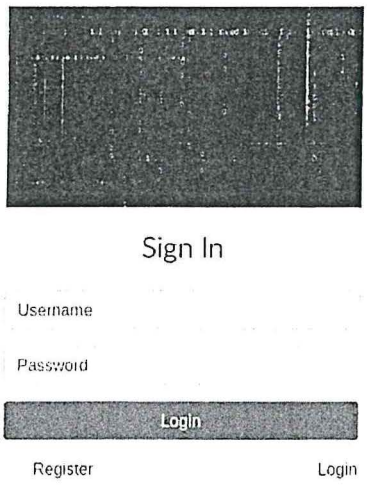


Figure 5.4 Login Screen

5.5 Functions of the Prototype

5.5.1 Use of the Graphical User Interface

This is a very user-friendly way of using the tool in that the user will not need to interact with the shell scripts. All operations are accomplished via a simple web browser interface. The figure 5.5 below shows the home interface of the tool via a web browser:

Figure 5.5 Homepage/ Pcap file upload page

As annotated in figure 5.5 a user just drag drops a PCAP file into the tool input area and clicks on the "Analyze" button to to compute the fingerpints and run a comparison against the database to identify the type of the browser as shown in the figure 5.6 below:

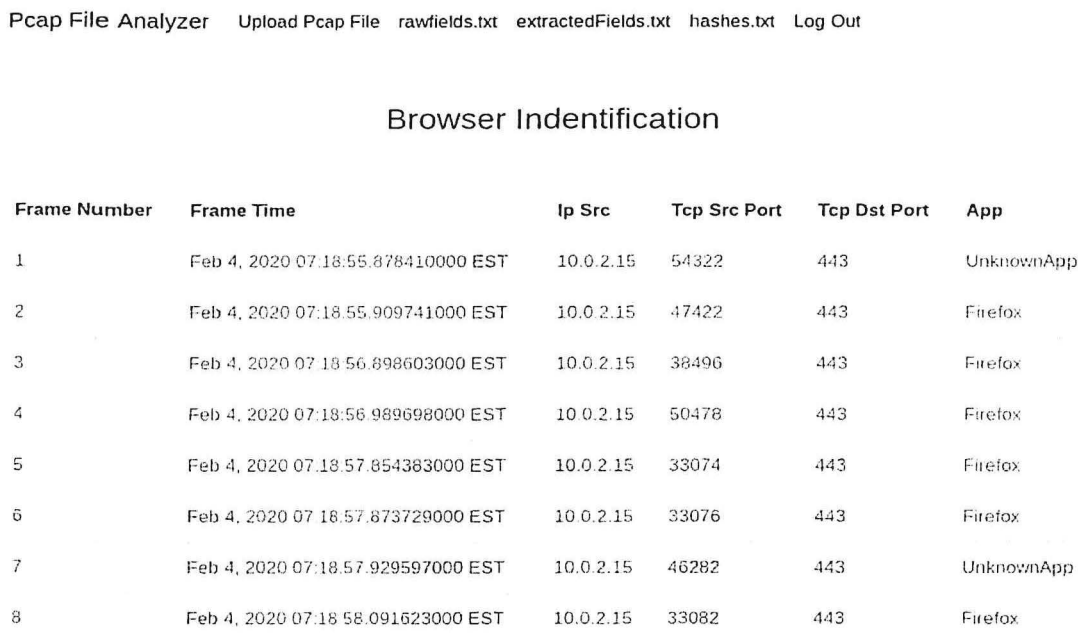


Figure 5.6 Results page

5.5.2 Command Line Interface

5.5.2.1 Extracting Network Packet Fields

The first function of the tool is to extract various fields from PCAP files. The fields are extracted from tls handshake client hello packets. Figure 5.7 below shows a screenshot of the unformatted JA3 fields that have been extracted form a PCAP file. The code snippets below show the JA3 fields and explainer fields that are extracted respectively.

```
#Extract JA3 fields and filter by client hello packets
${TSHARK} -r "${INFILE}" -T fields \
-e tls.handshake.version \
-e tls.handshake.ciphersuite \
-e tls.handshake.extension.type \
-e tls.handshake.extensions_supported_group \
-e tls.handshake.extensions_ec_point_format \
-R tls.handshake.type==1 -2
```

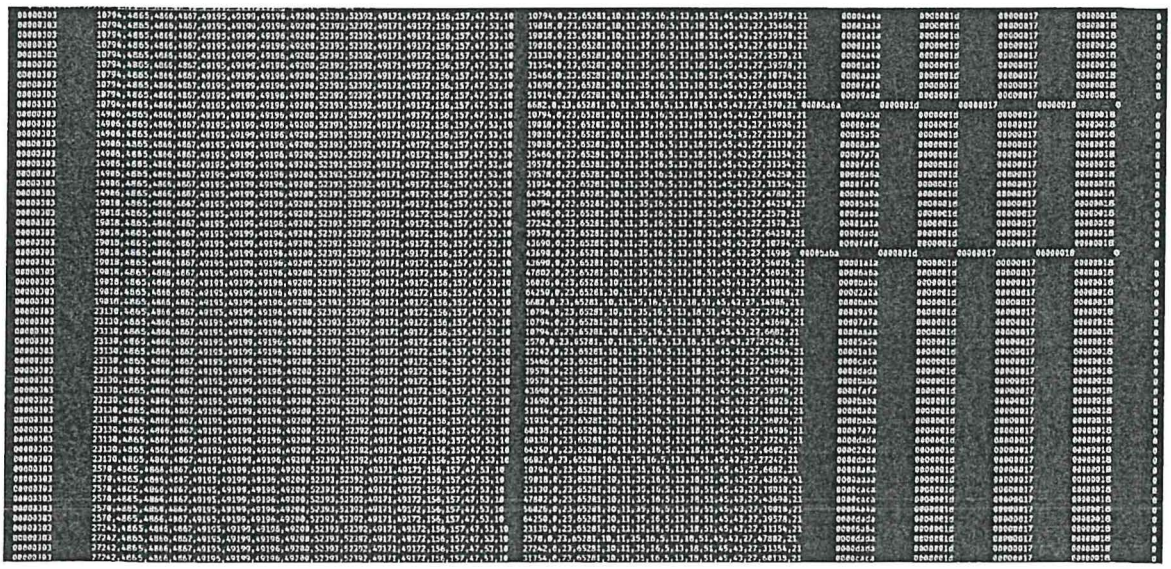


Figure 5.7 Unformatted JA3 Fields

```
#Extract explainer fields and filter by client hello packets
${TSHARK} -r "${INFILE}" -T fields \
-e frame.number \
-e frame.time \
-e ip.src \
-e tcp.srcport \
-e tcp.dstport \
-R tls.handshake.type==1 -2
```

5.5.2.2 Packet Fields Formatting and Manipulation

The extracted fields are raw and in different number formats such as decimal and hexadecimal. JA3 method follows formatting guidelines that necessitate formatting and manipulation of the extracted fields. AWK and SED tools are used primarily to format the fields and change number formats into the desired string pattern, ready for hashing, as displayed in figure 5.8. The code snippet below shows how this is technically achieved.

```
  | awk '{print substr($1,3) "\t"$2"\t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' \
    | awk -F '0x' -F ',0x' '{print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5 "\t" $6 "\t" $7 "\t" $8 "\t"$9
"\t" $10}' \
      | awk '{print $1 "\t"$2"\t"$3"\t" substr($4,3)"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' \
        > extractedFields.txt

# Replace all commas with dashes
sed -i 's/,/-/g' extractedFields.txt
```

```
#Convert Hexadecimal fields to Decimal
awk '{printf "%d,", strtonum( "0x"$1 );
   {print $2"," $3","};
   printf "%d-", strtonum( "0x"$4 );
   printf "%d-", strtonum( "0x"$5 );
   printf "%d-", strtonum( "0x"$6 );
   printf "%d-", strtonum( "0x"$7 );
   printf "%d-", strtonum( "0x"$8 );
   printf "%d,", strtonum( "0x"$9 );
   print $10 }' < extractedFields.txt \
     |    awk '{printf (NR%2==0) ? $0 "\n" : $0}' \
       > pktFields.txt #Save updated fields to text file for hashing
```

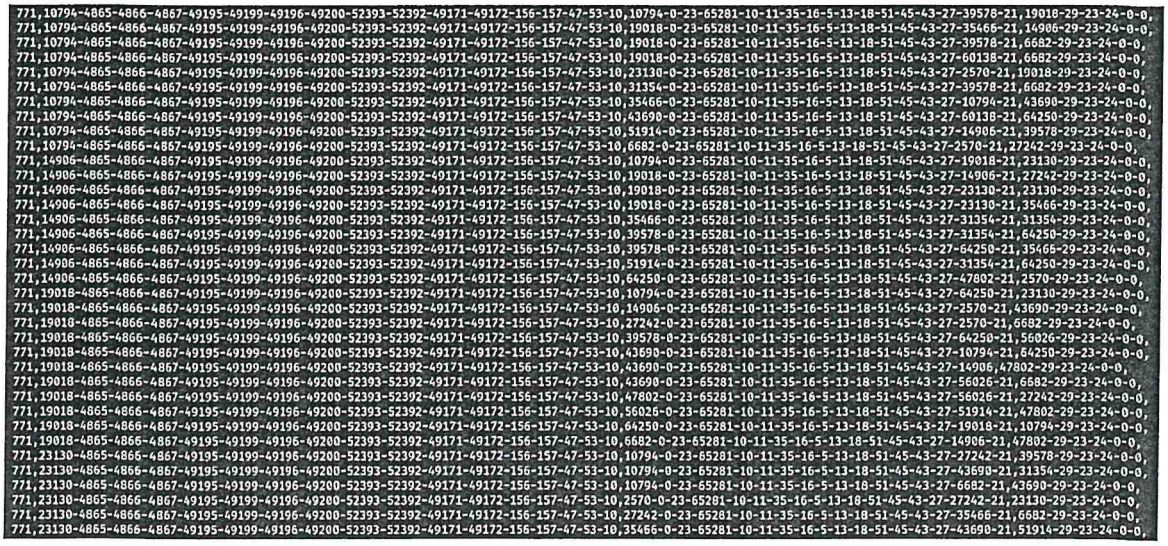


Figure 5.8 Formatted Fields

5.5.2.3 Hashing Packet Fields: Fingerprint Generation

The fields that have been formatted into a uniform string are hashed using an MD5 hash function, shown in the code snippet below. MD5 has been used because it is fast and produces short string hash values of the same length, which are easy to work with. Figure 5.9 shows some of the fingerprints. The calculated 128-bit cryptographic hashes are used to identify specific network communication belonging to a web browser.

```
# Calculate Hashes (MD5) for each line
while read -r line; do
```

```
    printf %s "$line" | md5sum | cut -f1 -d' '
done < pktFields.txt \
&& rm extractedFields.txt pktFields.txt
```

```
0048e1deb86e17404383da241b02899c
015c1ceb739b7486f286b56430d765b4
03395c9e58dbd498056ccbfe8e4572aa
03eed71af93802afa8bd28a9375ffab4
043702ef4aeb461619abff9e84a2bfc8
04ed45ec99d614d3b5b850306cfdbf01
06efb4bf9e29099d9e6b7e0f8e59ba26
086f52cc2927c385dd4f7de43673d5ab
0a03c12389ff6004a7b0cabc4c1a6eb1
0a9b2a6288b8f15b91a44fd7ba1cccf7
0adab9e41677cbefb14420115b274c12
0ce44f54f3092df8ae7b4fa15f33e95a
```

Figure 5.9 JA3 Fingerprints

5.5.2.4 Fingerprint Analysis and Comparison

One of the main functions of the system is to identify web browsers. This is achieved by comparing unknown fingerprints with the known fingerprints saved in the database. The function classifyApp() invokes the ja3_hash() function that computes JA3 fingerprints for unknown network communication and saves the output into a temporary text file. The parent function then queries the database comprising the known fingerprints and compares these fingerprints with the computed ones to find a match, whose results are displayed to the user CLI as shown in figure 5.10. The code snippets below show how this is programmatically implemented.

```
function classifyApp (){

  # md5_hash | sort | uniq > hashes.txt
  ja3_hash > hashes.txt
```

```
    while  IFS= read -r line
    do

      sudo mysql browserfingerprinting -e "SELECT IFNULL((SELECT app FROM ja3
WHERE fingerprint='$line' LIMIT 1),'UnknownApp');" | awk 'FNR == 2 {print}'

    done < hashes.txt

}

while [ -n "$1" ]; do
  case "$1" in
    --comp|-c)
       shift
       extAll > f1.txt
       classifyApp > f2.txt
       paste f1.txt f2.txt | awk 'BEGIN{print "frame.number \t frame.time \t ip.src \t tcp.srcport
\t tcp.dstport \t app"} {print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5 "\t" $6}'
       ;;
    *)
  esac
shift
done
```

frame.number	frame.time	ip.src		tcp.srcport	tcp.dstport		app
1	Feb 17, 2020 14:23:56.934308097 EST		EST	192.168.10.163	60744	443	Firefox
2	Feb 17, 2020 14:24:01.263443731 EST		EST	192.168.10.163	46036	443	Firefox
3	Feb 17, 2020 14:24:01.498696868 EST		EST	192.168.10.163	33658	443	Firefox
4	Feb 17, 2020 14:24:01.567972473 EST		EST	192.168.10.163	42994	443	Firefox
5	Feb 17, 2020 14:24:01.574338473 EST		EST	192.168.10.163	39164	443	Firefox
6	Feb 17, 2020 14:24:01.594882619 EST		EST	192.168.10.163	60034	443	Firefox
7	Feb 17, 2020 14:24:01.803014345 EST		EST	192.168.10.163	57876	443	Firefox
8	Feb 17, 2020 14:24:01.847133479 EST		EST	192.168.10.163	59732	443	Firefox
9	Feb 17, 2020 14:24:01.894772529 EST		EST	192.168.10.163	56806	443	Firefox
10	Feb 17, 2020 14:24:01.934966529 EST		EST	192.168.10.163	37142	443	Firefox
11	Feb 17, 2020 14:24:01.937754740 EST		EST	192.168.10.163	40274	443	Firefox
12	Feb 17, 2020 14:24:02.336280479 EST		EST	192.168.10.163	58952	443	Firefox
13	Feb 17, 2020 14:24:02.398571050 EST		EST	192.168.10.163	46068	443	Firefox
14	Feb 17, 2020 14:24:02.677703954 EST		EST	192.168.10.163	33296	443	Firefox
15	Feb 17, 2020 14:24:02.679515348 EST		EST	192.168.10.163	53070	443	Firefox
16	Feb 17, 2020 14:24:02.681086491 EST		EST	192.168.10.163	33304	443	Firefox

Figure 5.10 Browser Identification Results

5.6 System Testing

Several tests were done on the browser fingerprinting tool to ensure that it is meets its functional requirements and is secure. Due the headless nature of the browser fingerprinting tool, some vulnerabilities such as those relating to user interface and stored data are not applicable. The tool processes sensitive network data that could be detrimental when accessed by adversaries. Despite the general operating system security safeguards, an unforeseen vulnerability or human limitations could lead to exposure. The testing procedures focused on system functionalities and data handling.

5.6.1 Test Plan

Table 5.2 Test Plan

Item	Value
Objectives	To define the tools to be used throughout the testing process. To communicate to the responsible parties the items to be tested, set expectations around schedule. To define environmental needs. To define how the tests will be conducted.
Test Items	Functional Testing. Structural Testing.
Features to be Tested	Source code.
Approach	Manual tests will be carried out on all the modules of the application by different users.

	Test Cases were the developed with step-by-step procedures for testing the features. The sample target application to be updated by the system will be availed to the testers via a public GitHub account. Users participating in the testing will fill in prepared manual test cases with the results of the testing. Each user is required to repeat testing each module at least three times using the same datasets to ensure the results are repeatable.
Item Pass/Fail Criteria	All core functionality of the systems should function as expected and outlined in the individual test cases. There must be no critical defects found and an end user must be able to complete a purchase cycle successfully and initiate a refund without any errors. 95% of all test cases should pass and no failed cases should be crucial to the end user's ability to use the application.
Suspension Criteria	Testing should be paused immediately if either part of the system is found vulnerable to a security breach.
Test Deliverables	Test Plan (this document itself). Test Cases. Test Scripts. Defect/Enhancement Logs. Test Reports.
Test Environment	Virtual machine running Kali Linux version 2020.4 ShellCheck

5.6.1 Functional Testing

Functional testing checks if core requirements and functionalities are met. Table 5.3 shows the test use case summary results filled in by the users testing the tool. From the actual results obtained, it is evident that all test cases succeeded and that the tool is functioning properly as intended

Table 5.3 Testing Summary Results

ID	Test Case and Procedure	Expected Results	Actual Results	Pass /Fail
1	Test Name: Packet fields extraction Test Procedure: Open the CLI Type command sudo tshark -r firefox-klinux-68_2_0esr.pcapng -T fields -e tls.handshake.version -e tls.handshake.ciphersuite -e tls.handshake.extension.type -e tls.handshake.extensions_supported_group -e tls.handshake.extensions_ec_point_format -R tls.handshake.type==1 -2 to extract the specified fields from the capture file	0x00000303 4865,4867,4866,49195,49199,52393,52392,49196,49200,49162,49161,49171,49172,51,57,47,53,10 0,23,65281,10,11,35,16,5,51,43,13,45,28,21 0x0000001d,0x00000017,0x00000018,0x00000019,0x00000100,0x00000101 0 Exit status: 0	0x00000303 4865,4867,4866,49195,49199,52393,52392,49196,49200,49162,49161,49171,49172,51,57,47,53,10 0,23,65281,10,11,35,16,5,51,43,13,45,28,21 0x0000001d,0x00000017,0x00000018,0x00000019,0x00000100,0x00000101 0 Exit status: 0	Pass

2	Test Name: Packet Fields Formatting and Manipulation
	Test Procedure: Open the CLI
	Type command | awk '{print substr($1,3) "\t"$2"\t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' \
	| awk -F '0x' -F ',0x' '{print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5 "\t" $6 "\t" $7 "\t" $8 "\t"$9 "\t" $10}' \
	| awk '{print $1 "\t"$2"\t"$3"\t" substr($4,3)"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' >
	extractedFields.txt
	
	awk '{printf "%d,", strtonum("0x"$1);
	{print $2"," $3",");
	printf "%d-", strtonum("0x"$4);
	printf "%d-", strtonum("0x"$5);
	printf "%d-", strtonum("0x"$6);
	printf "%d-", strtonum("0x"$7);
	printf "%d-", strtonum("0x"$8);
	printf "%d,", strtonum("0x"$9);
	print $10 }' < extractedFields.txt \
	| awk '{printf (NR%2==0) ? $0 "\n" : $0}'
		771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-53-10,0-23-65281-10-11-35-16-5-51-43-13-45-28-21,29-23-24-25-256-257,0
		
		Exit status: 0
			771,4865-4867-4866-49195-49199-52393-52392-49196-49200-49162-49161-49171-49172-51-57-47-53-10,0-23-65281-10-11-35-16-5-51-43-13-45-28-21,29-23-24-25-256-257,0
			
			Exit status: 0
				Pass

3	Test Name: Hashing Packet Fields Test Procedure: Open the CLI Run the following command while read -r line; do l × printf %s "$line" | md5sum | cut -f1 -d' ' done < extractedFields.txt	a67028f7c1638543990cce852c05ee36 Exit status: 0	a67028f7c1638543990cce852c05ee36 Exit status: 0	Pass
4	Test Name: Fingerprint Analysis and Comparison Test Procedure: open the CLI Ru n the following command sudo tshark -r firefox-klinux-68_2_0esr.pcapng -c	frame.number frame.time ip.src tcp.srcport tcp.dstport app 1Feb17,202014:23:56.934308097EST EST 192.168.10.163 60744 443 Firefox Exit Status: 0	frame.number frame.time ip.src tcp.srcport tcp.dstport app 1Feb17,202014:23:56.934308097EST EST 192.168.10.163 60744 443 Firefox Exit Status: 0	Pass

5.6.2 Usability Testing

In usability testing, several aspects were examined by a representative of users to identify any usability problems, collect qualitative and quantitative data, and determine the user's satisfaction with the JA3 fingerprinting system. The users examined they systems' efficiency, responsiveness, and ease of use in general. This ensures that the system is easy to deploy and practically use. Table 5.4 below shows the useability testing framework results compiled by the researcher based on users' feedback.

Table 5.4 Usability Testing

Test Case Name: Application Usability **Date Tested**: 3rd May 2021. **Tested By**: Pius Muisyo **Test Description**: Step by step test for application usability.				
TEST STEPS				
Pre-Condition: All shell scripts must have executed successfully.				
Post-Condition: User can seamlessly run the application.				
Step	**Action**	**Expected Response**	**Pass/Fail**	**Comments**
1	User can execute all the python scripts.	All the python scripts are executable.	Pass	None
2	The script terminal commands are simple.	Terminal commands are simple to understand and easy to memorise.	Pass	None

3	The user can interpret the results.	The results generated are in tabular form and easier for the user to interpret.	Pass	Save the results in CSV for reporting

Figure 5.11 illustrates the respondents' answers to the useability questionnaire. Out of the 10 respondent who participated in application usability, five rated "Easy to use" and "efficiency" as good, and four rated the same as very good. As to "responsiveness" and "speed", six users rated good and four users for each rated as very good. Six users rated the system's "usefulness" as very good. Overall, the feedback from the users was good. All the respondents' have IT background with 5 in software development, 3 in software testing, and 2 information security individuals.

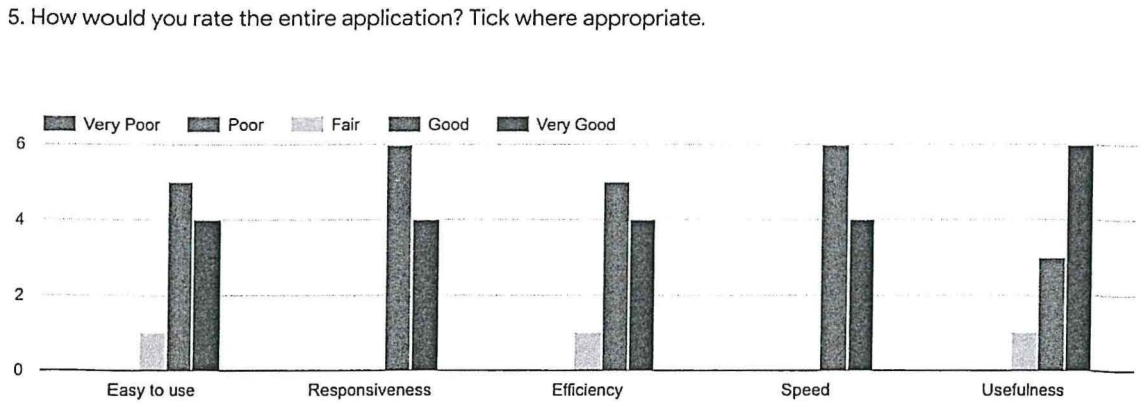


Figure 5.11 Usability Results

5.6.3 Structural Testing

In structural testing, tests are derived from the knowledge of the software's structure or internal implementation. Structural testing is critical because it is meant to provide forensic evidence to investigative officers. The use of an existing network protocol analysers and stream editors' frameworks helped in reducing the number of errors because the bugs had been identified and improved on over time by a large community of developers. The development process involved constant peer review to counter check the logic of the code.

5.6.3.1 Evaluation Environment

Operating System: Debian, Kali Linux 2020.4 running in a VirtualBox based virtual machine.

Random Access Memory: 4096MB 1600 MHz DDR3

Central Processing Unit: 2 cores, @ 3.5 GHz Quad-Core Intel Core i5

5.6.3.2 Evaluation Results

Table 5.5 shows time-based figures of the application for both user and system internal procedures, and CPU utilisation for several PCAP files which were used as the test dataset. The **dataset** column shows the PCAP files that were used during the testing (more details in table 3.2). The **user** column lists the amount of CPU time spent in user-mode code (outside the kernel) within the process. This is only actual CPU time used in executing the process. Other processes and time the process spends blocked do not count towards this figure. The **system** column lists the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls within the kernel, as opposed to library code, which is still running in user-space. The **total** column lists the wall clock time - time from start to finish of the call. This is all elapsed time including time slices used by other processes and time the process spends blocked (for example if it is waiting for I/O to complete). Finally, the **CPU** column indicates the percentage of the CPU's time that the process used while it ran.

Table 5.5 Performance Evaluation Results

Dataset	**User**	**System**	**CPU**	**Total**
firefox_klinux_78_3_0esr.pcapng	1.95s	1.36s	110%	2.98
firefox_windows_88_0.pcapng	3.86s	2.04s	110%	5.329
firefox2_klinux_78_3_0esr.pcapng	2.67s	1.48s	108%	3.811
firefox_mac_85_0.pcapng	5.30s	3.07s	109%	7.611
firefox_debian_78.8.0esr.pcapng	3.86s	2.19s	110%	5.492
firefox_mac_88_0.pcapng	5.91s	3.29s	110%	8.33
chrome_debian_90_0_4430_93.pcapng	4.30s	2.09s	108%	5.888
chrome_windows_90_0_4430_85.pcapng	4.11s	2.25s	107%	5.905

opera_mac_75_0_3969_243.pcapng	4.54s	2.55s	109%	6.491
chrome_klinux_88_0_4324_96.pcapng	3.46s	2.02s	111%	4.906
opera_debian_75_0_3969_218.pcapng	4.30s	2.20s	108%	5.963
opera_windows_75_0_3969_243.pcapng	1.90s	0.71s	104%	2.502
chrome_mac_90_0_4430_85.pcapng	4.25s	2.44s	109%	6.103
opera_klinux_75_3969_218.pcapng	3.30s	1.98s	109%	4.808

5.7 System Validation

In this section, several processes and activities were carried out to verify that the system's performance meets expectations and set objectives. The main objective of this study was to design, develop and test a JA3 based fingerprinting tool for browsers. The implemented application was able to satisfy this objective by being able to extract JA3 fields from PCAP files (including explainer fields to describe the fingerprints), compute JA3 fingerprints, compare with stored / verified fingerprints in a database and present a detailed report of the identity of the browser based on the comparison. During the system validation, the following concepts were ensured:

1. **Repeatability**: Tests were performed ten times using a similar PCAP file and similar results were observed. This demonstrated that the system produces similar results using the same method on identical inputs in the same environment by one user using the same equipment within short intervals of time.
2. **Reproducibility**: The application was installed in Ubuntu, Kali Linux, Debian and Mac OS operating systems and peer reviewed using the same packet capture files. It produced the same results for all instances. This demonstrated that the system produces similar results using the same method on identical test items in different laboratories with different operators utilizing different equipment.

Chapter 6 : Discussion of Results

6.1 Overview

Findings obtained during the study formed the basis on which the browser fingerprinting prototype was developed. The prototype was tested to ascertain that it met all its requirements. This chapter analyses the findings in relation to the research objectives and extent to which the findings agree with the literature review.

6.2 Vulnerabilities and Threats in Browser Communication

The first objective was to review common vulnerabilities and threats in browser communication. Browser based threats include a range of malicious software programs that are designed to infect victims' computers. The threats are made possible because of various vulnerabilities in the browser and the host system at large. The various vulnerabilities identified in section 2.4.2 are the most exploited by adversaries to compromise web browsers' security.

An important finding was that the most targeted aspects of a web browser include connections to online resources, plugins installed on the browser and vulnerabilities in the browser itself. Some of these plugins have access and can manage data saved in the file system. A vulnerability in these plugins would risk the sensitive data that this system is analysing and working on. This realisation informed the design and development of the browser fingerprinting tool in the way it handles and saves sensitive data. Various interventions were implemented that include the use of secure databases and deletion of runtime temporary data.

6.3 Fingerprinting Techniques for Browsers

The second objective was to review existing fingerprinting techniques by other researchers used to identify web browsers. Section 2.10 discuses some existing solutions that implement JA3 in different domains. The solutions had strengths and weaknesses that were considered in the design and implementation of the browser fingerprinting system. These were highlighted in the respective sections of each existing solution. Encrypted network traffic usage has grown and has good and bad effects. The good side is that it ensures secure data transmission, prevents eavesdropping, and increases the trustworthiness of communicating hosts. However, it complicates legitimate network traffic monitoring, such as traffic classification and host identification. We can now monitor,

identify, and classify plain-text network traffic like HTTP, but it is difficult to analyse encrypted communication.

6.4 JA3 Browser Based Fingerprinting

The third objective was to design, develop and test an improved JA3 based fingerprinting tool for browser identification in forensic scenarios based on clean and accurate fingerprints. The tool was successfully developed in line with the functional requirements. The tool runs in a Linux environment, executed by a user who wants to identify the type of browser that was used in the generated network communication. The tool works in tandem with other building utilities to extract and manipulate data. Extracted JA3 fingerprints are compared with known fingerprints saved in the database to see if there is a match with a particular browser.

6.5 System Validation

The fourth objective was to validate the effectiveness of the tool and its use in digital forensics. Section 5.6 addressed the system security testing to ensure that it is secure. To ensure that required fields from a PCAP file are extracted effectively, TSHARK was used in the background to filter out fields from other protocols that are not needed but included in the capture file. Inbuilt string manipulation tools were used to format the fields into JA3 format ready for hashing using MD5. Despite the known MD5 hash collision possibility, it is the most suitable cryptographic hashing algorithm in this project because it produces short, fixed strings of hashes which are easy to store, analyse and retrieve. MySQL database was used to store and query fingerprints for comparison with unknown fingerprints. All these aspects were tested and successfully achieved the required results without any error.

6.6 Advantages of the Developed Solution Compared to Existing Tools

The browser fingerprinting tool is an open-source software that can be downloaded via *https://github.com/Piusyo/Browser-Fingerpring-Using-JA3-Hashing-Algorithm-.git.* It is distributed under the MIT License whose conditions only require preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Not only web browsers, but also communicating devices, can be identified by the tool. At the very least, this applies to intelligence operations and forensic analysis. Creating a profile of the suspect

and correlating the identified activities with information from other sources can provide valuable insight into the case under investigation. Posts published under an anonymous social network account, for example, can be revealed by comparing the time of the public posts with the time of the actions as inferred by web browser usage, which can aid in hate crime investigations.

Regarding performance, the browser fingerprinting tool is quite fast. According to the performance evaluation analysis in table 5.3, the tool extracts relevant fields from PCAP files, formats them, hashes, compares generated fingerprints with the saved ones in the database and displays results in an average time of approximately 5.4 seconds.

Chapter 7 : Conclusions, Recommendations and Future Work

7.1 Conclusions

Browser fingerprinting can be considered as a practical approach that could be used by law enforcement officers to ascertain the identity of browsers used in communication in some unidentified network capture files. The benefit includes the identity of communicating parties (source and destination) and the services involved. This is a unique feature that related systems do not address. The researcher presented a study on the usability of JA3-based methods for browser identification in this dissertation. The method has the advantage of relying solely on TLS handshake information obtained during the secure channel establishment's initialization phase.

7.2 Recommendations

The bowser fingerprinting tool is of great importance in forensic applications, such as law enforcement agents and forensic investigators. The tool will help them gather first-hand information and provide important leads as to the trace of the communication link. It also saves them a lot of time that would otherwise be used to separately identify the communicating parties who have used the identified browser. Cybersecurity researchers and professionals are encouraged to use this tool in order to leverage on its extra available features as opposed to related JA3 tools.

7.3 Suggestions for Future Research

Current TLS versions that have been used in this research to provide information necessary for computing the fingerprints do not encrypt the three-way handshake information. However, ongoing work on the TLS protocol suggests that user privacy be strengthened by hiding more fields or even encrypting the TLS ClientHello message. Addressing these emerging challenges could be a future research topic.

In addition to knowing the browser identity and associated network information that the system reveals, it would be beneficial to the forensic fraternity to also know the operating system that was used. This may be possible through the principle of remote OS detection using TCP/IP stack fingerprinting. Integrating this feature to the tool is another topic for future work.

Finally, it would be more convenient and objective to use random URLs during the data capture stage. The application could be extended to initialise IP addresses as variables. Integrating this feature to the tool is another topic for future work.

References

Abel, R. (2019). SSL/TLS fingerprint tampering jumps from thousands to billions. SC Magazine.

Althouse, J. (2019, January 15). *TLS Fingerprinting with JA3 and JA3S - Salesforce Engineering.* Retrieved from *https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967* on 15 March 2021

Anderson, B., McGrew, D. (2019). TLS Beyond the Browser: Combining End Host and Network Data to Understand Application Behavior. In: Proceedings of the Internet Measurement Conference. pp. 379–392

Anderson, B., Paul, S., McGrew, D. (2018). Deciphering malware's use of TLS (without decryption). Journal of Computer Virology and Hacking Techniques pp. 195–211

Bach-Nutman, M. (2020). Understanding The Top 10 OWASP Vulnerabilities. *arXiv preprint arXiv:2012.09960.*

Bai, S., Kim, H., & Rexford, J. (2019). *Passive OS Fingerprinting on Commodity Switches.* Tech. Rep. TR-010-19.

Bedford, D. (2020). 3 The Changing Security Environment. In *Securing Freedom in the Global Commons* (pp. 34-48). Stanford University Press.

Benjamin, D. (2020). RFC 8701 Applying Generate Random Extensions and Sustain Extensibility (GREASE) to TLS Extensibility.

Bojinov, H., Michalevsky, Y., Nakibly, G., & Boneh, D. (2014). Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416.*

Brotherston, L., & Berlin, A. (2017). *Defensive security handbook: best practices for securing infrastructure.* " O'Reilly Media, Inc.".

Bujlow, T., Carela-Español, V., Solé-Pareta, J., & Barlet-Ros, P. (2015). Web tracking: Mechanisms, implications, and defenses. arXiv preprint arXiv:1507.07872.

Chang, D., Zhang, Q., & Li, X. (2015). Study on os fingerprinting and nat/tethering based on dns log analysis. In IRTF & ISOC Workshop on Research and Applications of Internet Measurements (RAIM).

Conti, M., Mancini, L. V., Spolaor, R., & Verde, N. V. (2015). Analyzing android encrypted network traffic to identify user actions. *IEEE Transactions on Information Forensics and Security*, *11*(1), 114-125.

Dastres, R., & Soori, M. (2020). Secure Socket Layer (SSL) in the Network and Web Security. *International Journal of Computer and Information Engineering*, *14*(10), 330-333.

Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Protzenko, J., Rastogi, A., Swamy, N., ... & Zinzindohoue, J. K. (2017, May). Implementing and proving the TLS 1.3 record layer. In *2017 IEEE Symposium on Security and Privacy (SP)* (pp. 463-482). IEEE.

Dierks, T., & Rescorla, E. (2008). The transport layer security (TLS) protocol version 1.2.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology*, *50*(9-10), 833-859.

FaizKhademi, A., Zulkernine, M., & Weldemariam, K. (2015, July). FPGuard: Detection and prevention of browser fingerprinting. In *IFIP Annual Conference on Data and Applications Security and Privacy* (pp. 293-308). Springer, Cham.

Gancheva, Z., Sattler, P., & Wüstrich, L. (2020). TLS Fingerprinting Techniques. *Network*, *15*.
Mozuni, M., & Jonas, W. (2017). An introduction to the morphological Delphi Method for design: A tool for future-oriented design research. *She Ji: The Journal of Design, Economics, and Innovation*, *3*(4), 303-318.

Gollmann, D. (2010). Computer security. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(5), 544-554.

Grosskurth, A., & Godfrey, M. W. (2005). A reference architecture for web browsers. In 21st IEEE International Conference on Software Maintenance (ICSM'05) (pp. 661-664). IEEE.

Gupta, S., & Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, *8*(1), 512-530.

Hassan, M. M., Ali, M. A., Bhuiyan, T., Sharif, M. H., & Biswas, S. Quantitative Assessment on Broken Access Control Vulnerability in Web Applications. In *International Conference on Cyber Security and Computer Science 2018.*

Hassan, M. M., Nipa, S. S., Akter, M., Haque, R., Deepa, F. N., Rahman, M., ... & Sharif, M. H. (2018). Broken authentication and session management vulnerability: a case study of web application. *International Journal of Simulation Systems, Science & Technology*, *19*(2), 6-1.

Hodges, J., Morgan, R., & Wahl, M. (2000). Lightweight Directory Access Protocol (v3): Extension for Transport Layer Security (No. RFC 2830).

Hoffman, P. (2002). SMTP service extension for secure SMTP over transport layer security.

Holz, R., & Saint-Andre, P. (2015). Recommendations for secure use of transport layer security (tls) and datagram transport layer security (dtls).

Humayun, M., Niazi, M., Jhanjhi, N. Z., Alshayeb, M., & Mahmood, S. (2020). Cyber security threats and vulnerabilities: a systematic mapping study. *Arabian Journal for Science and Engineering*, *45*(4), 3171-3189.

Husák, M., Čermák, M., Jirsík, T., & Čeleda, P. (2016). HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. *EURASIP Journal on Information Security*, *2016*(1), 1-14.

Ilangakoon, S. D., & Jayakody, J. A. (2016, December). Awareness of Sri Lankan internet users on web browsing related threats and vulnerabilities. In *2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)* (pp. 1-6). IEEE.

Jeng, T. H., Luo, W. Y., Huang, C. C., Chen, C. C., Chang, K. H., & Chen, Y. M. (2021). Cloud computing for malicious encrypted traffic analysis and collaboration. *International Journal of Grid and High-Performance Computing (IJGHPC)*, *13*(3), 12-29.

Joshi, P. N., Ravishankar, N., Raju, M. B., & Ravi, N. C. (2017, December). Contemplating Security of Http from SQL Injection and Cross Script. In *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)* (pp. 1-5). IEEE.

Khademi, A. F., Zulkernine, M., & Weldemariam, K. (2015). An empirical evaluation of web-based fingerprinting. *Ieee Software*, *32*(4), 46-52.

Kim, H. W., & Kankanhalli, A. (2009). Investigating user resistance to information systems implementation: A status quo bias perspective. *MIS quarterly*, 567-582.

Kimberlin, C. L., & Winterstein, A. G. (2008). Validity and reliability of measurement instruments used in research. *American journal of health-system pharmacy*, *65*(23), 2276-2284.

Kotzias, P., Razaghpanah, A., Amann, J., Paterson, K.G., Vallina-Rodriguez, N., Caballero, J. (2018). Coming of age: A longitudinal study of TLS deployment. In: Proceedings of the Internet Measurement Conference 2018. pp. 415–428

Kumar, R., & Goyal, R. (2019). On cloud security requirements, threats, vulnerabilities and countermeasures: A survey. *Computer Science Review*, *33*, 1-48.

Kurtz, A., Gascon, H., Becker, T., Rieck, K., Freiling, F.: Fingerprinting mobile devices using personalized configurations. Proc. Privacy Enhancing Technol. 1, 4–19 (2016)

Laperdrix, P., Rudametkin, W., & Baudry, B. (2015, May). Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification. In *2015 IEEE/ACM 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 98-108). IEEE.

Lastovicka, M., Jirsik, T., Celeda, P., Spacek, S., & Filakovsky, D. (2018). Passive os fingerprinting methods in the jungle of wireless networks. In NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium (pp. 1-9). IEEE.

Lemon, S. (2015). TLS Fingerprinting Smarter Defending & Stealthier Attacking. Online], Sep, 25.

Liu, X., Liu, Q., Wang, X., & Jia, Z. (2016, June). Fingerprinting web browser for tracing anonymous web attackers. In *2016 IEEE First International Conference on Data Science in Cyberspace (DSC)* (pp. 222-229). IEEE.

Maruping, L. M., Venkatesh, V., & Agarwal, R. (2009). A control theory perspective on agile methodology uses and changing user requirements. *Information Systems Research*, *20*(3), 377-399.

Matoušek, P., Burgetová, I., Ryšavý, O., & Victor, M. (2020, October). On Reliability of JA3 Hashes for Fingerprinting Mobile Applications. In International Conference on Digital Forensics and Cyber Crime (pp. 1-22). Springer, Cham.

Matsunaka, T., Yamada, A., & Kubota, A. (2013). Passive OS fingerprinting by DNS traffic analysis. In 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA) (pp. 243-250). IEEE.

McGrew, D., & Anderson, B. (2016, November). Enhanced telemetry for encrypted threat analytics. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)* (pp. 1-6). IEEE.

McGrew, D., & Bailey, D. (2012). Aes-ccm cipher suites for transport layer security (tls) (pp. 1-6). RFC 6655, july.

Moe, M. M. (2019). Unit Test using Test-Driven Development Approach to Support Reusability. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, *3*(3).

Nielson, J., Williamson, C., & Arlitt, M. (2008). Benchmarking modern web browsers. In 2nd IEEE Workshop on Hot Topics in Web Systems and Technologies.

Nikiforakis, N., Kapravelos, A., Joosen, W., Kruegel, C., Piessens, F., & Vigna, G. (2013). Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In 2013 IEEE Symposium on Security and Privacy (pp. 541-555). IEEE.

Okon, B. E., & Umunnah, R. A. Identification of Threats to Communication Security, Effects On Victims and Ways of Mitigating the Threats.

Open Systems, (2019): JA3 - High Performance Go Implementation. Retrieved from *https://pkg.go.dev/github.com/open-ch/ja3* on 24 March 2021.

Osincev, A. A., & Laponina, O. R. (2019). Vulnerability Testing in Web Applications External Entities XML. *International Journal of Open Information Technologies*, *7*(10), 71-79.

Pohl, T. (2021). *Secure infrastructure for exchanging rules in static code analysis tools* (Bachelor's thesis).

Polk, T., McKay, K., & Chokhani, S. (2014). Guidelines for the selection, configuration, and use of transport layer security (TLS) implementations. NIST special publication, 800(52), 32.

Poptani, R., & Gatty, M. V. (2018). Security Misconfiguration. *Security Misconfiguration*, *7*(1), 3-3.

Razaghpanah, A., Niaki, A.A., Vallina-Rodriguez, N., Sundaresan, S., Amann, J., Gill, P. (2017). Studying TLS Usage in Android Apps. In: Proceedings of the 13th Inter- national Conference on Emerging Networking Experiments and Technologies. p.350–362. ACM, New York, NY, USA

Rescorla, E. (2000). SSL and TLS: designing and building secure systems (Vol. 1). Reading: Addison-Wesley.

Rescorla, E. (2010). *Keying material exporters for transport layer security (tls)*. RFC 5705, March.

Rezaei, S., & Liu, X. (2019). A target-agnostic attack on deep models: Exploiting security vulnerabilities of transfer learning. *arXiv preprint arXiv:1904.04334*.

Rivera-Ortiz, F., & Pasquale, L. (2020, August). Automated modelling of security incidents to represent logging requirements in software systems. In *Proceedings of the 15th International Conference on Availability, Reliability and Security* (pp. 1-8).

Roques, O. (2019). Detecting Malware in TLS Traffic (Doctoral dissertation, Imperial College London).

Sapsford, R., & Jupp, V. (Eds.). (1996). Data collection and analysis. Sage.

Shafagh, H., Hithnawi, A., Dröscher, A., Duquennoy, S., & Hu, W. (2015, November). Talos: Encrypted query processing for the internet of things. In *Proceedings of the 13th ACM conference on embedded networked sensor systems* (pp. 197-210).

Solutions, E. (2017). Why requirements are important.

Soni, M., & Jain, A. (2018, February). Secure Communication and Implementation Technique for Sybil Attack in Vehicular Ad-Hoc Networks. In *2018 Second International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 539-543). IEEE.

Steiner, M., Buhler, P., Eirich, T., & Waidner, M. (2001). Secure password-based cipher suite for TLS. ACM Transactions on Information and System Security, 4(2), 134- 157.

Taleck, G. (2004). Synscan: Towards complete tcp/ip fingerprinting. CanSecWest, Vancouver BC, Canada, 1-12.

Turner, S. (2014). Transport layer security. IEEE Internet Computing, 18(6), 60-63.

Velan, P., Čermák, M., Čeleda, P., & Drašar, M. (2015). A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, *25*(5), 355-374.

Williams, B., Dong, X., & Qian, L. (2020, December). Data Driven Network Monitoring and Intrusion Detection using Machine Learning. In *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)* (pp. 1-7). IEEE.

Yang, W., Xu, K., Lian, J., Bin, L., & Ma, C. (2018). Multiple flood vulnerability assessment approach based on fuzzy comprehensive evaluation method and coordinated development degree model. *Journal of environmental management*, *213*, 440-450

Zeifman, I. (2012). Was that really a Google bot crawling my site? Retrieved from https://www.incapsula.com/blog/was-that-really-a-google-bot-crawling-my-site.html on 2 July 2020.

Appendices

Appendix A: Use Cases

Table A.1: Extract Fields Use Case

Use Case Name:	**Extract Fields**
Description:	Read and format packets from a previously saved capture file (PCAP) using TSHARK and SED/AWK
Primary Actor:	TSHARK
Secondary Actor:	SED/AWK
Include Use Cases:	Save to CSV Generate hashes Compare hashes
Extend Use Cases:	Wrong options / format
Preconditions:	The PCAP network file must be valid and have been initialised to variable $1
Post Conditions:	Format extracted fields into JA3 acceptable string for hashing
Main Flow:	Tshark extracts the JA3 fields from the capture file. Process the fields using string manipulation tools such as AWK and SED by replacing all commas with dashes and converting hexadecimal fields to decimals.
Alternative Flows:	Extract other explainer fields (such as source IP address, timestamp, source and destination port) and match with generated fingerprints to offer more explanation.

Table A.2 Calculate Hashes Use case

Use Case Name:	**Calculate Hashes**
Description:	Use MD5 message-digest algorithm to hash each string of lines, producing a 128-bit hash value.
Primary Actor:	Md5sSum
Secondary Actor:	Database (MySQL)
Include Use Cases:	Compare hashes
Extend Use Cases:	None
Preconditions:	The fields must be formatted into a predefined JA3 format using the string manipulation tools.
Post Conditions:	Generate fixed length MD5 hashes and save / compare with database.
Main Flow:	Hash each line of formatted fields to a fixed length hashes saved into a text file. Compare with known JA3 fingerprints in the database for browser identification.
Alternative Flows:	Hash each line of formatted fields belonging to a known browser into a fixed length hashes saved into a text file. Save the fingerprints in a database to be used in future comparison with unknown fingerprints.

Table A.3 Compare Hashes Use Case

Use Case Name:	**Compare Hashes**
Description:	Use an SQL query is to match each fingerprint to a pre-calculated set and fetch the corresponding application name.
Primary Actor:	Database
Secondary Actor:	None
Include Use Cases:	Display Results
Extend Use Cases:	None
Preconditions:	Existing JA3 fingerprints for known browsers in the database. Calculated JA3 fingerprints for unknown browsers.
Post Conditions:	Display a table of the browser identity together with associated identity information.
Main Flow:	An SQL query is used to match each fingerprint to a pre-calculated set and fetch the corresponding application name. If a match is not found, the query returns the "Unknown App" string.
Alternative Flows:	None

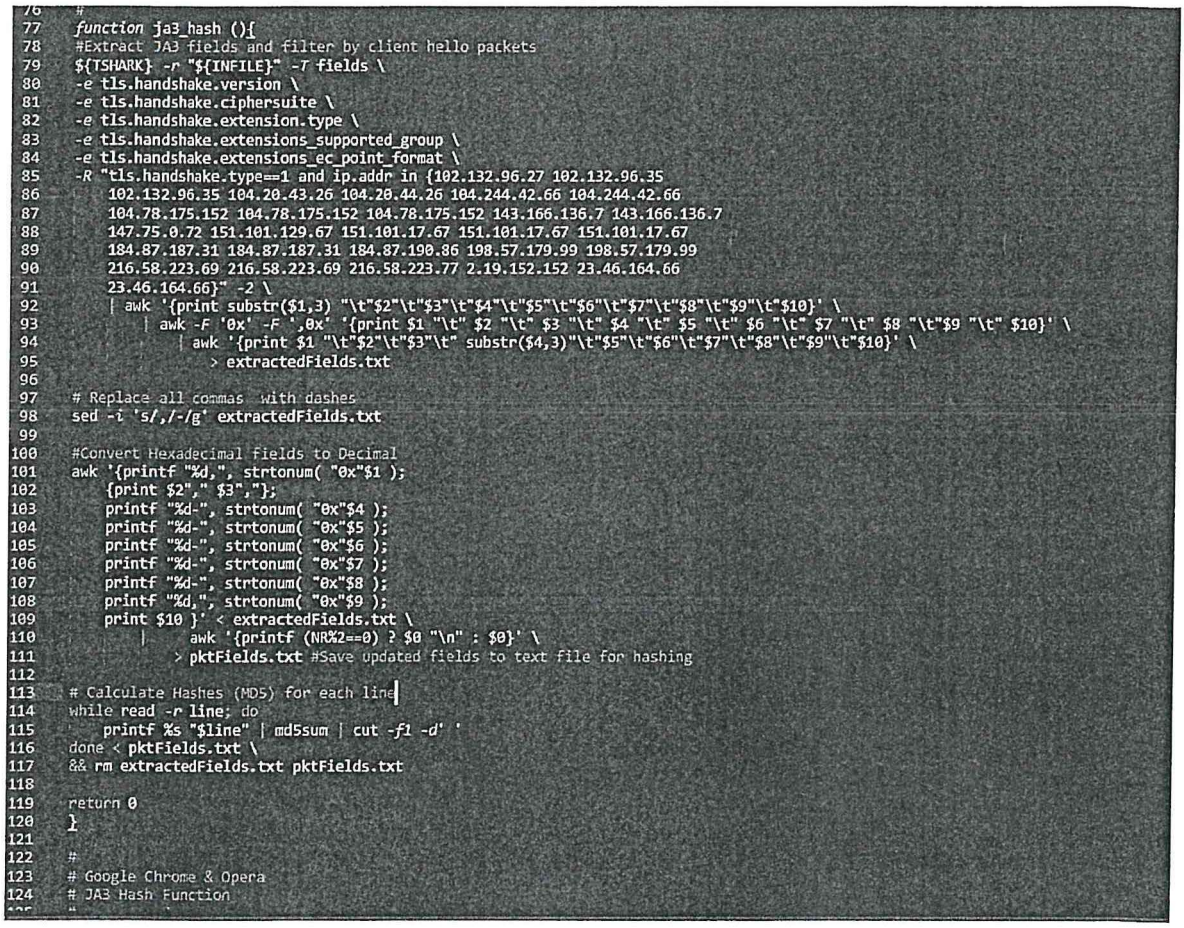
Appendix B: PCAP files Dataset

Figure B.1 PCAP files dataset

Appendix C: Shell scripting Code Snippet

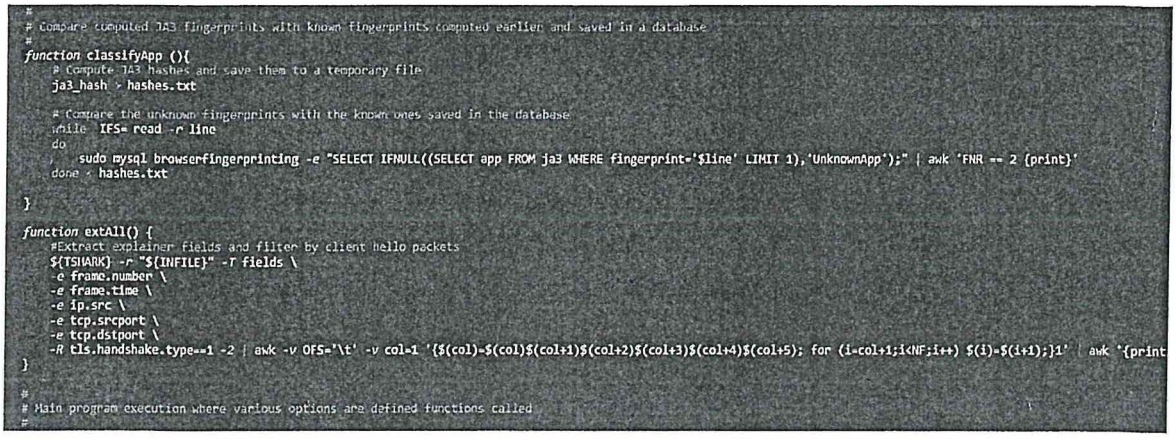
```
#!/bin/bash

TSHARK="/usr/bin/tshark"

#
# File read error
#
function filerror(){
    echo "Cannot read file \"$1\""
    exit 1
}

#
# check input PCAP file
#
if [ ! -r $1 ]; then
    filerror $1
else
    INFILE=$1
fi

filePath="${INFILE##*/}"
csvName="${filePath%.*}"

${TSHARK} -r "${INFILE}" -T fields -e dns.a > dns_data/dns_a_"$csvName".csv
${TSHARK} -r "${INFILE}" -T fields -e dns.resp.name > dns_data/dns_resp_name_"$csvName".csv
paste -d "," dns_data/dns_a_"$csvName".csv dns_data/dns_resp_name_"$csvName".csv | sort | uniq > dns_data/dnsAllRcsUniq_"$csvName".csv

function ja3_hash(){
${TSHARK} -r "${INFILE}" -T fields \
-e ip.dst \
-e tls.handshake.version \
-e tls.handshake.ciphersuite \
-e tls.handshake.extension.type \
-e tls.handshake.extensions_supported_group \
-e tls.handshake.extensions_ec_point_format \
-R "tls.handshake.type==1 and ip.addr in {198.57.179.99 147.229.2.82 147.229.2.90
157.240.30.35 184.51.8.147 2.18.68.206 52.30.88.10 34.253.101.66 34.249.165.210
52.48.145.94 34.255.235.176 54.171.79.102 52.19.22.175 34.248.188.242 52.45.117.194
52.205.228.94 34.232.204.33 34.232.6.198   52.5.59.12 54.174.68.95 52.7.248.149
52.7.114.31 52.46.135.211 52.46.141.49 54.239.26.255 99.86.240.33 99.86.241.241
18.205.93.1 18.205.93.2 18.205.93.0 192.124.249.12 192.124.249.5}" -2
}
```

Figure C.1 DNS_Filter.sh script for performing DNS filter



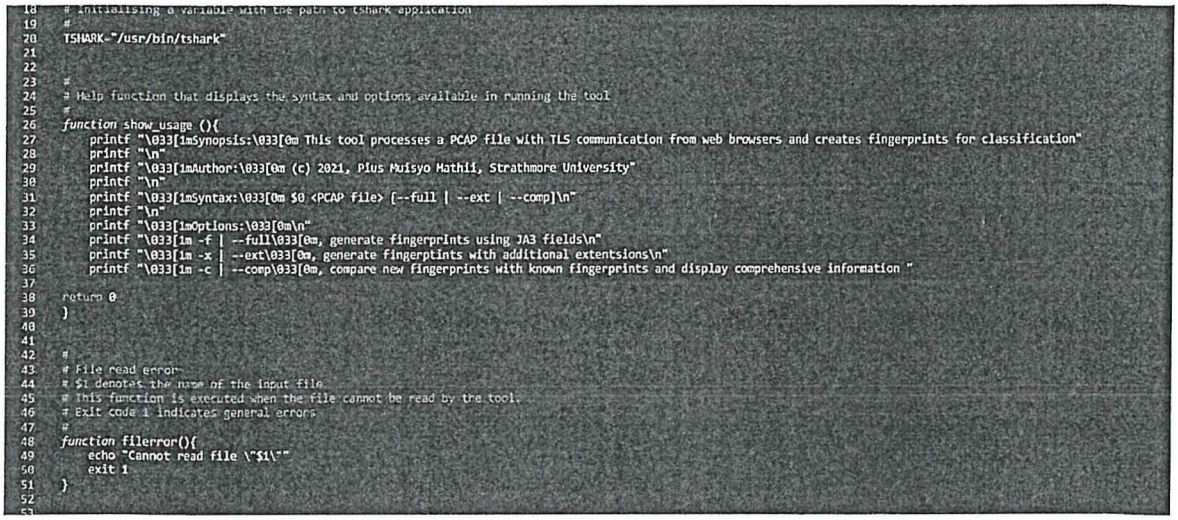
```
#
function ja3_hash (){
#Extract JA3 fields and filter by client hello packets
${TSHARK} -r "${INFILE}" -T fields \
-e tls.handshake.version \
-e tls.handshake.ciphersuite \
-e tls.handshake.extension.type \
-e tls.handshake.extensions_supported_group \
-e tls.handshake.extensions_ec_point_format \
-R "tls.handshake.type==1 and ip.addr in {102.132.96.27 102.132.96.35
    102.132.96.35 104.20.43.26 104.20.44.26 104.244.42.66 104.244.42.66
    104.78.175.152 104.78.175.152 104.78.175.152 143.166.136.7 143.166.136.7
    147.75.0.72 151.101.129.67 151.101.17.67 151.101.17.67 151.101.17.67
    184.87.187.31 184.87.187.31 184.87.190.86 198.57.179.99 198.57.179.99
    216.58.223.69 216.58.223.69 216.58.223.77 2.19.152.152 23.46.164.66
    23.46.164.66}" -2 \
    | awk '{print substr($1,3) "\t"$2"\t"$3"\t"$4"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' \
        | awk -F '0x' -F ',0x' '{print $1 "\t" $2 "\t" $3 "\t" $4 "\t" $5 "\t" $6 "\t" $7 "\t" $8 "\t"$9 "\t" $10}' \
            | awk '{print $1 "\t"$2"\t"$3"\t" substr($4,3)"\t"$5"\t"$6"\t"$7"\t"$8"\t"$9"\t"$10}' \
                > extractedFields.txt

# Replace all commas  with dashes
sed -i 's/,/-/g' extractedFields.txt

#Convert Hexadecimal fields to Decimal
awk '{printf "%d,", strtonum( "0x"$1 );
    {print $2"," $3",",};
    printf "%d-", strtonum( "0x"$4 );
    printf "%d-", strtonum( "0x"$5 );
    printf "%d-", strtonum( "0x"$6 );
    printf "%d-", strtonum( "0x"$7 );
    printf "%d-", strtonum( "0x"$8 );
    printf "%d,", strtonum( "0x"$9 );
    print $10 }' < extractedFields.txt \
        |     awk '{printf (NR%2==0) ? $0 "\n" : $0}' \
            > pktFields.txt #Save updated fields to text file for hashing

# Calculate Hashes (MD5) for each line
while read -r line; do
    printf %s "$line" | md5sum | cut -f1 -d' '
done < pktFields.txt \
&& rm extractedFields.txt pktFields.txt

return 0
}

#
# Google Chrome & Opera
# JA3 Hash Function
```

Figure C.2 brwFing_filteredIPS.sh script for Creating Fingerprints for known browsers



```
#
# Compare computed JA3 fingerprints with known fingerprints computed earlier and saved in a database
#
function classifyApp (){
    # Compute JA3 hashes and save them to a temporary file
    ja3_hash > hashes.txt

    # Compare the unknown fingerprints with the known ones saved in the database
    while IFS= read -r line
    do
        sudo mysql browserfingerprinting -e "SELECT IFNULL((SELECT app FROM ja3 WHERE fingerprint='$line' LIMIT 1),'UnknownApp');" | awk 'FNR == 2 {print}'
    done < hashes.txt

}

function extAll() {
    #Extract explainer fields and filter by client hello packets
    ${TSHARK} -r "${INFILE}" -T fields \
    -e frame.number \
    -e frame.time \
    -e ip.src \
    -e tcp.srcport \
    -e tcp.dstport \
    -R tls.handshake.type==1 -2 | awk -v OFS='\t' -v col=1 '{$(col)=$(col)$(col+1)$(col+2)$(col+3)$(col+4)$(col+5); for (i=col+1;i<NF;i++) $(i)=$(i+1);}1' | awk '{print
}

#
# Main program execution where various options are defined functions called
#
```

Figure C.3 brwFing_filteredIPS.sh script for Creating Fingerprints for known browsers



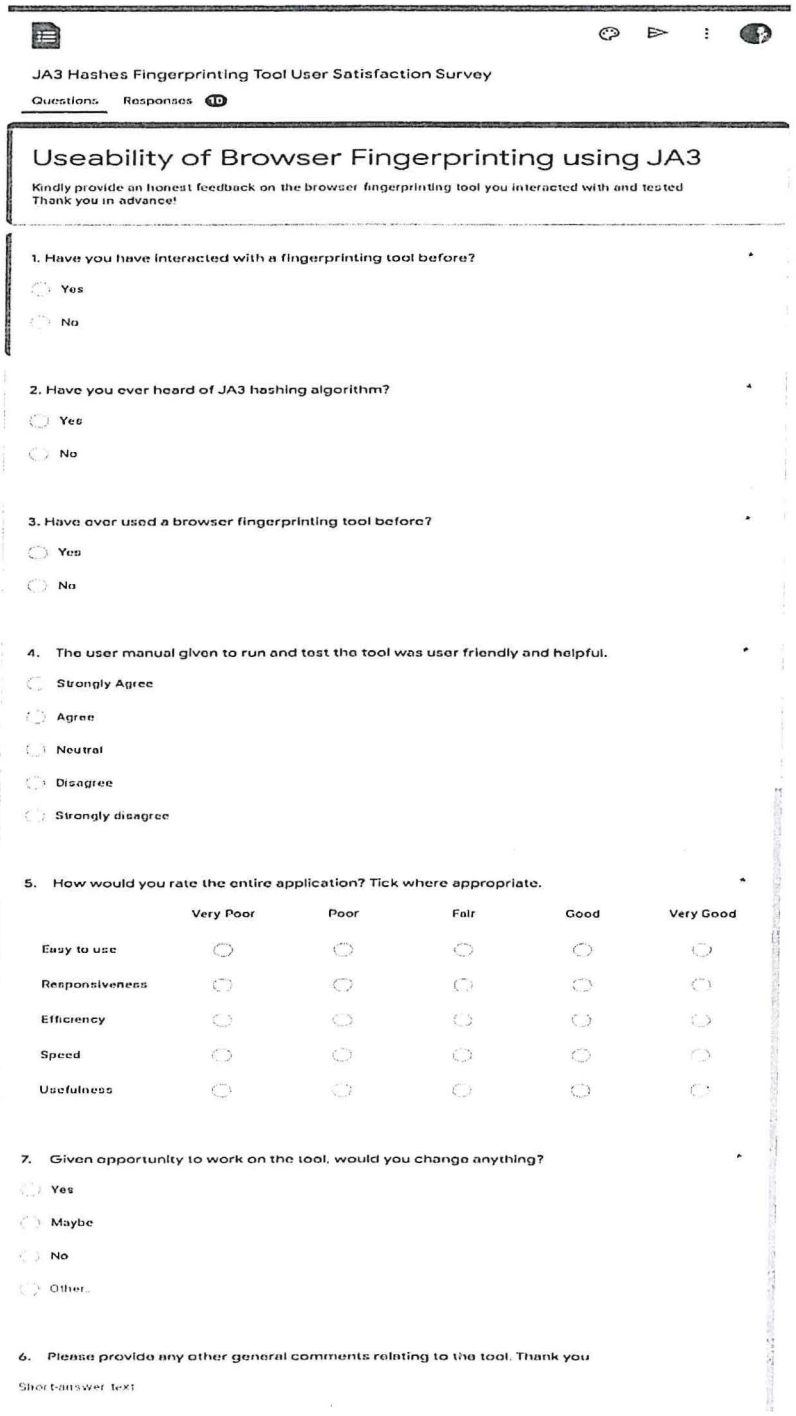
```
# Initialising a variable with the path to tshark application
#
TSHARK="/usr/bin/tshark"


#
# Help function that displays the syntax and options available in running the tool
#
function show_usage (){
    printf "\033[1mSynopsis:\033[0m This tool processes a PCAP file with TLS communication from web browsers and creates fingerprints for classification"
    printf "\n"
    printf "\033[1mAuthor:\033[0m (c) 2021, Pius Muisyo Mathii, Strathmore University"
    printf "\n"
    printf "\033[1mSyntax:\033[0m $0 <PCAP file> [--full | --ext | --comp]\n"
    printf "\n"
    printf "\033[1mOptions:\033[0m\n"
    printf "\033[1m -f | --full\033[0m, generate fingerprints using JA3 fields\n"
    printf "\033[1m -x | --ext\033[0m, generate fingerptints with additional extentsions\n"
    printf "\033[1m -c | --comp\033[0m, compare new fingerprints with known fingerprints and display comprehensive information "

return 0
}


#
# File read error
# $1 denotes the name of the input file
# This function is executed when the file cannot be read by the tool.
# Exit code 1 indicates general errors
#
function filerror(){
    echo "Cannot read file \"$1\""
    exit 1
}
```

Figure C.4 brwFing.sh script for comparison between known and unknown browser fingerprints

```
# Main program execution where various options are defined functions called
#
while [ -n "$1" ]; do
  case "$1" in
    --full|-f)      ## processing input data with --full option
        shift
        filePath="${INFILE##*/}"
        csvName="${filePath%.*}"
        ja3_hash | sort | uniq > fingerprints/uniq_"$csvName".csv



        ;;

    --ext|-x)       ## processing input data with --ext option
        shift
        ja3_hash > ja3_fingerprints.txt

        ${TSHARK} -r "${INFILE}" -T fields \
            -e frame.number \
            -e frame.time_relative \
            -e ip.src \
            -e ip.dst \
            -R tls.handshake.type==1 -2 \
                | awk '{print $1 "\t" $2" \t" $3 "\t" $4}' \
                    > extractedxFields.txt


        # Combine extensions fields with the JA3 fingeprints, and save output into a comma separated file
        paste extractedxFields.txt ja3_fingerprints.txt | awk  'BEGIN{print "frame.number','frame.time_relative','ip.src','ip.dst','ja3_fingerprints"}
                                                    {print $1 "," $2 "," $3 "," $4 "," $5}'

        ;;

    --comp|-c)
        shift
        extAll > f1.txt
        classifyApp > f2.txt
        paste f1.txt f2.txt | awk 'BEGIN{print "frame.number \t frame.time \t ip.src \t tcp.srcport \t tcp.dstport \t app"}{print $1 "\t" $2 "\t" $3 "\t" $4 "\t

        ;;

    --ch_op|-o)
        shift
        filePath="${INFILE##*/}"
        csvName="${filePath%.*}"
        ja3_hash_chrome_opera | sort | uniq > fingerprints/uniq_"$csvName".csv
        ;;
    *)
  esac
shift
done
```

Figure C.5 brwFing.sh script for comparison between known and unknown browser fingerprints

Appendix D: Usability Questionnaire and Responses



JA3 Hashes Fingerprinting Tool User Satisfaction Survey

Questions Responses 10

Useability of Browser Fingerprinting using JA3

Kindly provide an honest feedback on the browser fingerprinting tool you interacted with and tested
Thank you in advance!

1. Have you have interacted with a fingerprinting tool before? *
- Yes
- No

2. Have you ever heard of JA3 hashing algorithm? *
- Yes
- No

3. Have ever used a browser fingerprinting tool before? *
- Yes
- No

4. The user manual given to run and test the tool was user friendly and helpful. *
- Strongly Agree
- Agree
- Neutral
- Disagree
- Strongly disagree

5. How would you rate the entire application? Tick where appropriate. *

	Very Poor	Poor	Fair	Good	Very Good
Easy to use					
Responsiveness					
Efficiency					
Speed					
Usefulness					

7. Given opportunity to work on the tool, would you change anything? *
- Yes
- Maybe
- No
- Other...

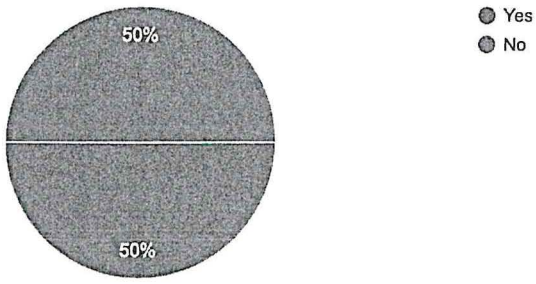
6. Please provide any other general comments relating to the tool. Thank you

Short-answer text

Figure D.1 Usability Survey Questionnaire

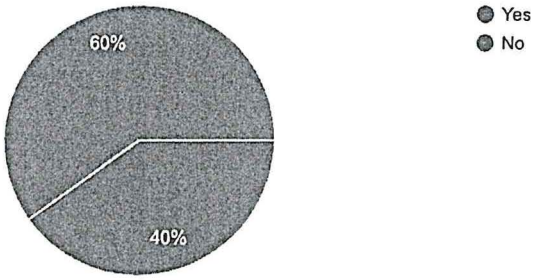
1. Have you have interacted with a fingerprinting tool before?

10 responses



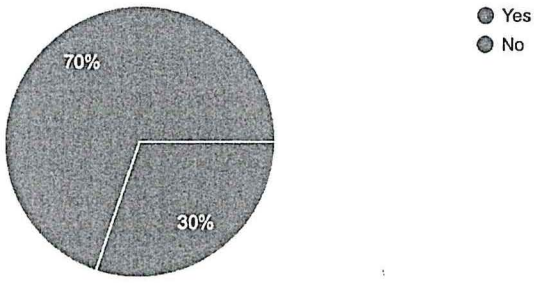
2. Have you ever heard of JA3 hashing algorithm?

10 responses



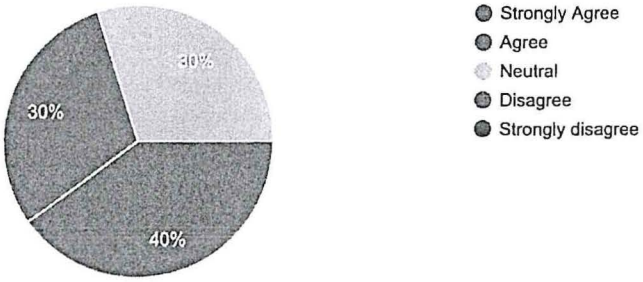
3. Have ever used a browser fingerprinting tool before?

10 responses



4. The user manual given to run and test the tool was user friendly and helpful.

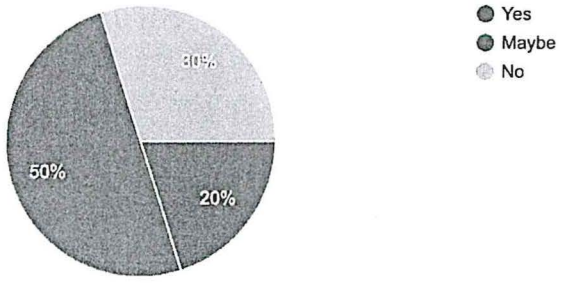
10 responses



5. How would you rate the entire application? Tick where appropriate.

7. Given opportunity to work on the tool, would you change anything?

10 responses



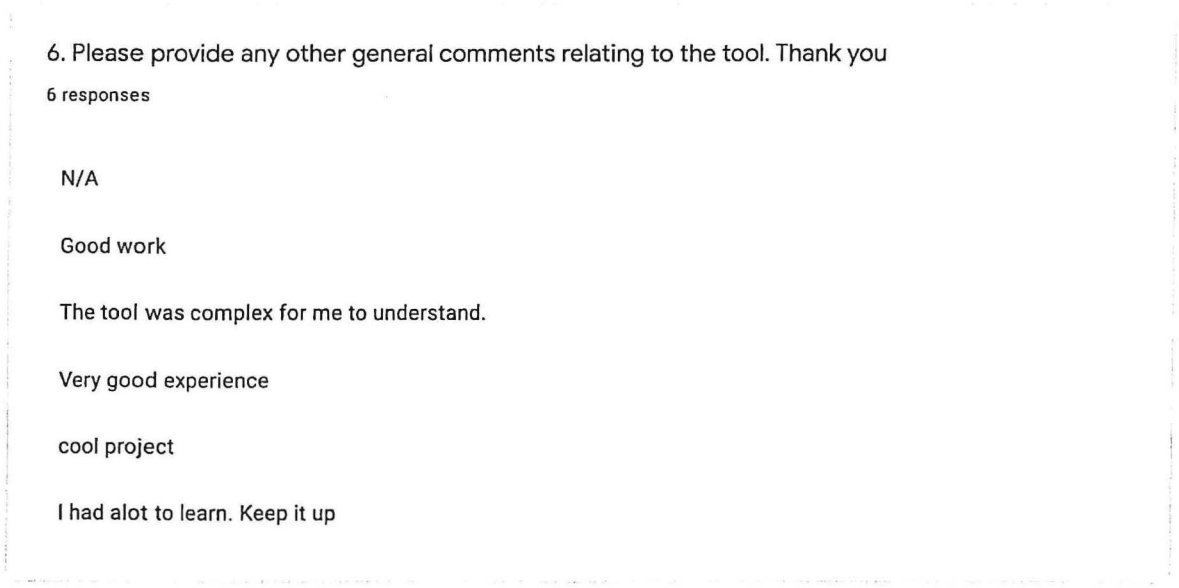


Figure D.2 Usability Survey Responses

Appendix E: Plagiarism Check Report

Graduate Theses/Dissertations Similarity Checker (2021)

Similarity Checker for **BOTH** *pre-defense* and *final* submissions. The latest submission made at the time of closure shall be assumed as final.

Submission status

Attempt number	This is attempt 1.
Submission status	Submitted for grading
Grading status	Not graded
Due date	Wednesday, 30 June 2021, 12:00 AM
Time remaining	53 days 20 hours
Last modified	Friday, 7 May 2021, 3:41 AM
File submissions	Application of Browser Fingerprinting using JA3 Hashes in Digital Forensics.docx Ouriginal: **17%** 7 May 2021, 3:41 AM
Submission comments	▸ Comments (0)

Strathmore UNIVERSITY

1st November 2021

Mr Mathii Pius,
pmuisyo@strathmore.edu

Dear Mr Mathii,

RE: Application of Browser Fingerprinting using JA3 Hashes in Digital Forensics

This is to inform you that SU-IERC has reviewed and **approved** your above **SU-master's** research proposal. Your application reference number is **SU-IERC1075/21.** The approval period is **1st November 2021 to 31st October 2022.**

This approval is subject to compliance with the following requirements:

i. Only approved documents including (informed consents, study instruments, MTA) will be used
ii. All changes including (amendments, deviations, and violations) are submitted for review and approval by SU-IERC.
iii. Death and life-threatening problems and serious adverse events or unexpected adverse events whether related or unrelated to the study must be reported to SU-IERC within 48 hours of notification
iv. Any changes, anticipated or otherwise that may increase the risks or affected safety or welfare of study participants and others or affect the integrity of the research must be reported to SU-IERC within 48 hours
v. Clearance for export of biological specimens must be obtained from relevant institutions.
vi. Submission of a request for renewal of approval at least 60 days prior to expiry of the approval period. Attach a comprehensive progress report to support the renewal.
vii. Submission of an executive summary report within 90 days upon completion of the study to SU-IERC.

Prior to commencing your study, you will be expected to obtain a research license from National Commission for Science, Technology, and Innovation (NACOSTI) https://research-portal.nacosti.go.ke/ and obtain other clearances needed.

Yours sincerely,

Prof Fred Were,
Chairperson; SU-IERC

STRATHMORE UNIVERSITY INSTITUTIONAL ETHICS REVIEW COMMITTEE
(SU-IERC)
01 Nov 2021
TEL: +254 (0)70303400
P.O BOX 59857-00200
NAIROBI-KENYA