UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Discovery of Web Attacks by Inspecting HTTPS Network Traffic with Machine Learning and Similarity Search

Nuno Pedro Godinho Cavaco Durão

Mestrado em Segurança Informática

Dissertação orientada por:
Prof.ª Doutora Ibéria Vitória de Sousa Medeiros
Prof. Doutor Vinicius Vielmo Cogo

2022

Agradecimentos

Sou muito grato à minha família pela força que me deram. Aos meus pais por estarem sempre comigo. À minha esposa Sónia, pelo apoio diário que me permitiu prosseguir a minha carreira académica. Aos meus filhos, Inês e Tomás, que me ajudaram a ultrapassar todos os momentos difíceis.

Agradeço aos meus orientadores, Ibéria Medeiros e Vinicius Cogo, pela determinação com que me ajudaram e pela liberdade que me deram, no desenvolvimento deste trabalho.

Este trabalho foi parcialmente suportado por fundos nacionais através da Fundação para a Ciência e a Tecnologia (FCT) com referência ao projeto SEAL (PTDC/CCIINF/29058/2017) e à Unidade de Investigação LASIGE (UIDB/00408/2020 e UIDP/00408/2020).

Dedicado à Inês e ao Tomás.

Resumo

A Internet disponibiliza de forma transversal o acesso a uma infinidade de serviços que permitem uma sociedade digital e progressiva. A disponibilização de aplicações web permite desde a interconexão entre pessoas, à possibilidade de realizar compras ou operações bancárias. Ainda recentemente, com a pandemia do Covid-19, a mudança significativa de recursos para ambiente cloud, permitiu a rápida adaptação ao teletrabalho. Sendo a disponibilidade de aplicações web uma necessidade emergente, a pressão para a sua produção é cada vez maior e provoca alguns constrangimentos. A democratização do desenvolvimento aplicacional veio criar uma série de programadores com pouca base técnica, que usam ferramentas low-cost e em que a segurança é vista como um custo e não como um dos pilares de uma aplicação web.

Com a utilização crescente de aplicações web, a informação transacionada pelas mesmas é cada vez maior em quantidade e em valor intrínseco. Por outro lado, o acesso a ferramentas maliciosas por parte de atores criminosos, tem feito aumentar os ataques sobre este tipo de aplicações. Tal como do lado dos programadores, também do lado dos atacantes a facilidade de acesso a ferramentas prontas a usar, sem qualquer conhecimento técnico, é cada vez maior, o que potencia o aparecimento de atores maliciosos. A informação que pode ser acedida é de grande valor, a variedade de aplicações disponíveis online de forma ininterrupta é cada vez maior e sem limites geográficos, a impunidade dos atacantes ainda é usual e os ataques sucedem-se.

Por parte da comunidade de segurança informática, existe o esforço de desenvolver e criar sistemas, protocolos e políticas para tentar proteger a informação e as aplicações em questão. Uma das soluções utilizadas é o protocolo de comunicação *Hypertext Transfer Protocol Secure* (HTTPS), protocolo que se tornou o standard para cifrar a comunicação entre cliente e servidor. HTTPS garante Integridade e Confidencialidade ao cifrar o conteúdo dos pacotes transmitidos na rede recorrendo ao protocolo *Transport Layer Security* (TLS). Outra solução largamente empregada é a inspeção de tráfego web, por recurso a técnicas de *Deep Package Inspection* (DPI). A utilização de DPI permite filtrar pacotes com recurso a técnicas de assinaturas ou por pesquisa de palavras-chave.

A adoção de DPI acarreta no entanto dois problemas distintos. O seu uso pode originar uma degradação do serviço, devido ao peso computacional da sua implementação e a sua utilização é muito limitada quando se trata de comunicação baseada em HTTPS. Para ultrapassar a carga computacional e a potencial degradação do serviço, a solução passa pela utilização de tráfego agregado por fluxo, *NetFlow*, que permite a agregação de tráfego com propriedades comuns e

assim aumentar a capacidade de monitorizar o tráfego. Relativamente à comunicação baseada em HTTPS, a utilização de DPI não é possível, visto que os pacotes em HTTPS estão cifrados e a utilização de DPI implica aceder ao conteúdo do pacote. Para ultrapassar essa limitação, uma solução que se encontra disseminada é a utilização de servidores intermédios (proxies) para decifrar os dados, inspecioná-los e voltar a cifrá-los para os enviar ao servidor. Esta solução cria um ponto de ataque adicional e enfraquece substancialmente a segurança criptográfica da comunicação entre cliente e servidor. A segurança da rede e em especial das aplicações web, no caso da comunicação por HTTPS, não fica totalmente acautelada com a utilização de *firewalls* e DPI, devido à dificuldade de aceder ao conteúdo dos pacotes.

Esta tese tem como objetivo apresentar uma solução para deteção de ataques web sobre tráfego HTTPS, com recurso a técnicas de agregação de tráfego, aprendizagem automática e descoberta de similaridades de conteúdo. Neste sentido, começámos por estudar os ataques a aplicações web, em especial SQLi e XSS. Estudámos ainda a agregação de tráfego em *NetFlows*. De seguida desenvolvemos um sistema para detetar tráfego anómalo em *NetFlow*, com base em tráfego HTTPS, através do uso de aprendizagem automática não supervisionada. Este sistema engloba a captura do tráfego HTTPS e a criação do *NetFlow* a partir desse tráfego. O *NetFlow* agrega o tráfego com propriedades em comum e que passa no ponto de observação durante um determinado período de tempo. Garantindo que as propriedades do tráfego utilizadas para construir o *NetFlow* são adequadas à criação de clusters de tráfego, é possível com recurso a um algoritmo de aprendizagem automática não supervisionada a segregação de tráfego anómalo. De forma a certificar que o tráfego anómalo inclui ataques a aplicações Web, foi desenvolvido um motor de pesquisa de similaridade, baseado em *Locality Sensitive Hashing* (LSH), que serve para pesquisar o conteúdo dos pacotes de HTTPS. Para alcançar o acesso ao conteúdo dos pacotes sem a necessidade de decifar o tráfego entre o cliente e o servidor, a solução recorre aos logs no servidor. Com vista a assegurar a utilização prática dos logs é fundamental garantir que o servidor regista em log todos os pedidos que recebe e criar um extrator de logs. A funcionalidade do extrator de logs, implica a necessidade de transformar os logs existentes no servidor num formato utilizável e que permita a análise do conteúdo dos pacotes. Este extrator de logs, engloba um filtro que permite filtrar apenas os pedidos que chegam ao servidor e que incluem o conteúdo dos pedidos. Com este extrator de logs conseguimos não só segmentar os logs do tráfego assinalado como anómalo, mas também estruturar o conteúdo dos pacotes de forma a conseguirmos analisar e garantir que o tráfego inclui ataques a aplicações web. Para validar se o tráfego e, em concreto, cada um dos pacotes analisados, inclui ataques, criámos um motor de pesquisa de similaridades que permite efetuar pesquisas com recurso a medidas de similaridade. A utilização de medidas de similaridade suporta a identificação de ataques sem a carga computacional associada a um DPI e permite a identificação de ataques semelhantes mas não identificados previamente. O motor de pesquisa de similaridades foi implementado, através do uso de algoritmos de LSH. De forma a manter o sistema atualizado e com capacidade de evolução, foi ainda desenhado um sistema de melhoria continua, que permite reforçar o motor de pesquisa de similaridades com novos ataques, de forma

a robustecer a sua eficiência ao longo da sua utilização.

Na avaliação realizada à solução foi possível a segregação de tráfego anómalo, com base num NetFlow de tráfego HTTPS. Esta segregação foi efetuada com recurso a um algoritmo de aprendizagem automática não supervisionada. A validação de que o tráfego anómalo inclui ataques a aplicações web, foi efetuado com o motor de pesquisa de similaridades. Foi possível constatar que o grau de objetividade do motor de pesquisa de similaridades, permitiu identificar com grande segurança a existência de ataques. Testando o sistema de melhoria continua, foi possível observar uma melhoria no motor de pesquisa de similaridades. Embora a melhoria identificada fosse pequena, o sistema é continuo e as melhorias são acumuladas ao longo do tempo, pelo que se apresenta como uma mais valia para o sistema total.

Algumas das contribuições deste estudo são: o estudo de ataques em HTTPS a aplicações web; o desenho e implementação de uma estrutura para detetar tráfego anómalo em HTTPS, recorrendo a tráfego agregado em NetFlows e aprendizagem automática não supervisionada; o desenvolvimento de uma ferramenta para processar e compilar logs de Apache; o estudo de pesquisas de similaridade; a elaboração de um processo de deteção de payloads com ataques, utilizando pesquisa de similaridades; a evolução da solução num processo de melhoria continua; a avaliação da solução proposta.

Palavras-chave: Detecção de ataques web, Tráfego HTTPS, NetFlows, Apendizagem automática, Local Sensitive Hashing

Abstract

Web applications are the building blocks of many services, from social networks to banks. Network security threats have remained a permanent concern since the advent of data communication. Not withstanding, security breaches are still a serious problem since web applications incorporate both company information and private client data. Traditional Intrusion Detection Systems (IDS) inspect the payload of the packets looking for known intrusion signatures or deviations from normal behavior. However, this Deep Packet Inspection (DPI) approach cannot inspect encrypted network traffic of Hypertext Transfer Protocol Secure (HTTPS), a protocol that has been widely adopted nowadays to protect data communication. We are interested in web application attacks, and to accurately detect them, we must access the payload. Network flows are able to aggregate flows of traffic with common properties, so they can be employed for inspecting large amounts of traffic.

The main objective of this thesis is to develop a system to discover anomalous HTTPS traffic and confirm that the payloads included in it contains web applications attacks. We propose a new reliable method and system to identify traffic that may include web application attacks by analysing HTTPS network flows (netflows) and discovering payload content similarities. We resort to unsupervised machine learning algorithms to cluster netflows and identify anomalous traffic and to Locality Sensitive Hashing (LSH) algorithms to create a Similarity Search Engine (SSE) capable of correctly identifying the presence of known web applications attacks over this traffic. We involve the system in a continuous improvement process to keep a reliable detection as new web applications attacks are discovered.

We evaluated the system, which showed that it could detect anomalous traffic, the SSE was able to confirm the presence of web attacks into that anomalous traffic, and the continuous improvement process was able to increase the accuracy of the SSE.

Keywords: Detection of Web attacks, HTTPS Network traffic, Netflows, Machine Learning, Local Sensitive Hashing

Contents

List of Figures

List of Tables

Acronyms

API Application Programming Interface

CIA Confidentiality, Integrity and Authentication

DBSCAN Density-Based Spatial Clustering of Applications with Noise

DDoS Distributed Denial-of-Service

DOM Document Object Model

DoS Denial-of-Service

DPI Deep Packet Inspection

DVWA Damn Vulnerable Web App

ENISA European Union Agency for Cybersecurity

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority

IDS Intrusion Detection System

IE Information Elements

IETF Internet Engineering Task Force

IP Internet Protocol

IPFIX Internet Protocol Flow Information Export

ISO International Organization for Standardization

LSH Locality-Sensitive Hashing

NVD National Vulnerability Database

OSI Open Systems Interconnection

OSVDB Open Source Vulnerability Database

OWASP Open Web Application Security Project

PCAP Packet Capture

RFC Request for Comments

SaaS Software as a Service

SHA Secure Hash Algorithm

SIP Session Initiation Protocol

SOC Security Operations Center

SQL Structured Query Language

SQLi Structured Query Language injection

SSE Similarity Search Engine

SSH Secure Shell Protocol

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

URL Uniform Resource Locator

UTC Coordinated Universal Time

WINE Worldwide Intelligence Network Environment

XSS Cross Site Scripting

ZAP Zed Attack Proxy

Chapter 1

Introduction

In a world where Covid has increased the shift to the remote paradigm, the amount of online web applications is rising and rising. On the user side, there is a quest for more and more tools to be available online. The need to keep delivering web applications is huge [6]. Due to its versatility in offering services to society, a web application is one of the most widely utilised technologies today, either in society, government, or industry. Web applications contribute to the improvement of our everyday lives since they can supply nearly any type of service, from helping people to communicate, disseminating information and relieving the strain of recurring activities, like shopping, paying bills and working.

The pressure for offering more web applications, forces the adoption of low code development and software as a service (SaaS) solutions. Either one of them could present week points as the companies must entrust the security of those solutions. There is also a tendency of low security standards as security is often considered a cost and not a major pillar of the solution. We have seen that for people without programming expertise, web application creation is becoming more accessible. The popularity of low code solutions, which anybody can use to deploy a web application is growing. Entrepreneurship in the field of SaaS with solutions based on low code, from people without a technological background is rising.

As a result, many users personal data that circulates on the Internet becomes a prime catch for attackers. In order to address this threat, experts devised a slew of security procedures, protocols and policies to protect the data in question. One of the major solution adopted was HTTPS, which has become the protocol used by most browsers. In terms of web applications, the use of Deep Package Inspection (DPI) is a major technological tool in Intrusion Detection Systems (IDS) for core services such as protocol analysis, antivirus protection, online filtering and intrusion detection. DPI, however, relies on the capacity to examine the packet content, which is not possible when dealing with HTTPS. The demand for network security are not being answer by simple firewalls, and the IDS now in use is lacking support to deal with the use of HTTPS.

1.1 Motivation

The MyFreeCams website, an adult streaming website, got breached in late 2020 [7]. In December an hacker stole data from the website, through an Structured Query Language injection (SQLi) attack. The data stolen includes the records of 2 million premium members of the website. The stolen data was later on sale, on an online hacker forum, for $ 1.500 worth of Bitcoin per 10.000 records. The post was deleted, as well as the hacker account, but it was possible to find that the hacker got 40 transactions on his bitcoin wallet, totalizing $ 22.400 in Bitcoin.

SQLi is nowadays a major Web Application Security Risk [8]. From the attacker point of view, it is not only accessible to conduct this kind of attacks, but it is also a profitable business. The ability to detect these attacks are of interest importance. The common technique used to detect SQLi, resorts to DPI. DPI is a great way to detect attacks, over inspection of network traffic but is a cumbersome technique, and creates a bottleneck in the system. More and more the use of Network flows, the aggregation of packets into flows, presents a way to monitor and analysis the traffic. The network flow tools and knowledge are sturdy. With the use of network flows and resourcing to Machine Learning techniques, we can analyse a large amount of traffic. We can detect several web attacks, but there are a few types of web attacks that the used tools are not capable of detecting them. That is the case of SQLi and Cross-site scripting (XSS). On top of these difficulties, the use of HTTPS traffic increases the difficulty to detect such attacks because the payload is encrypted and therefore, it can not be inspected. HTTPS prevents a DPI approach to be successful, as the payload is encrypted, and therefore cannot be analysed. One of the available solution to deal with this is the use of an intermediary proxy to decrypt the traffic between the client and the server. This proxy allows the inspection of the packets content, but creates a security issue [9] that we do not want.

To address this problem, we must find a resilient system, capable of dealing with the amount of data to process. The solution is to use a network flow. Using a network flow we are able to process the huge amount of data, and rapidly identify anomalous traffic with attacks. To identify anomalous traffic, in a network flow, one of the tools to use is Machine Learning, which will provide a way to identify the clusters with malicious traffic, and pinpointing the traffic to further analyse. After the identification of the anomalous traffic, we will confirm, that the traffic contains web applications attacks, without decrypting the traffic between the client and the server, but take advantage of the web server logs. Finally we propose a way to evolve the system, by growing the database of known attacks.

1.2 Objectives

The goal of this dissertation is to discover web attacks in HTTPS network traffic, without decrypting the traffic. We propose to accomplish this by the use of Unsupervised Machine Learning to detect anomalous traffic in network flows (NetFlows), and confirming that the network traffic contains web attacks by the use of a Similarity Search Engine (SSE) based on Locality Sensitive

Hashing (LSH) algorithms. The research will be focused on HTTPS, but the solution is not limited to HTTPS network traffic; it can be used with HTTP traffic. Even though we want to discover web attacks in a broad sense, SQLi and XSS are two types of web attacks with special interest. One other important constrain we want to comply with, is not decrypting the traffic between the client and the server. The main objectives of this research is:

- Study web application attacks (exploits and attack vectors) and NetFlow structure.
- Study of Locality-sensitive hashing to detect similarities in payloads.
- Define an approach to identify anomalous traffic by analysing and processing NetFlows with unsupervised machine learning algorithms and confirm the presence of web application attacks on this traffic, without decrypting the traffic between the client and the server.
- Implement the approach in order to achieve a reliable detection system.
- Maintain a robust system that can evolve as new attacks are discovered.

1.3 Contributions

The main contributions of this dissertation are:

- A study on web application attacks in HTTPS.
- A study of Locality-Sensitive Hashing (LSH), its parameters and different algorithms, to detect similarities in payloads.
- The design and implementation of a solution to detect web attacks in NetFlows, produced from HTTPS traffic, through the use of unsupervised machine learning algorithms, and their confirmation through a SSE, based on LSH algorithms, for searching their payload contents.
- A log parser tool to compile the logs produced in Apache and produce a uniform and complete output with all the payloads received in the server.
- An evaluation of the solution showing that it is able to process network flows and identify malicious ones, and then confirm such attacks trough LSH.
- A continuous improvement process that evolves and enhances the system for new web attacks and LSH searches.

1.4 Thesis Organisation

In the next chapter, we present the base knowledge of web attacks discovery, and related work of this dissertation. In Chapter 3, we present our approach for a solution to detect anomalous traffic in HTTPS, using NetFlow and machine learning, a way to confirm that the anomalous traffic

contains web application attacks and a way to improve the system as we gain more examples of attacks. We discuss the issues we had in addressing the problems we encountered, as well as the solutions we suggest to solve them. In Chapter 4, we present the environment scenario, design and implementation details. In Chapter 5 we present the evaluation of the solution. We conclude with Chapter 6 with the final conclusions and future work.

Chapter 2

Background and Related Work

The purpose of this chapter is to establish the context and background related to the topics of this thesis. With the rise of web application attacks, it is fundamental to develop strategies and frameworks to detect them. We will cover the main options available and also the latest studies related.

2.1 Network Traffic

The National Institute of Standards and Technology defines network traffic [10] as *computer network communications that are carried over wired or wireless networks between hosts.* To study network traffic, we must overview the Transmission Control Protocol/Internet Protocol (TCP/IP). The OSI model [11], specifies the functions of a networking systems and how data is passed between computers. The International Telecommunication Union in its standard x.200 [12] defines 7 layers to the OSI Model:

7 – Application

6 – Presentation

5 – Session

4 – Transport

3 – Network

2 – Data link

1 – Physical

There is however the TCP/IP model [13] [14], developed prior to the OSI Model. This TCP/IP model functionality is based on four layers, each with its own set of protocols. TCP/IP is a tiered server architecture system in which each layer is specified by the function it is to execute. These four TCP IP layers work together to transfer data from one layer to the next. The layers that constitute the TCP/IP Model are:

- Application Layer. In the OSI model is equivalent to the combination of Session, Presentation and application layers. It contains higher-level protocols such as File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP) and Hyper Text Transfer Protocol (HTTP). The data output of this layer is called a message.
- Host-to-Host Layer. This layer is in charge of end-to-end connectivity and error-free data transfer. It protects upper-layer applications from data complexity. Two protocols found in this layer are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). In this layer the message from the previous layer is added a header to produce a segment with TCP
- Internet Layer. In this layer the Internet Protocol (IP) implements a set of rules, for routing and addressing datagrams. A datagram contains a header, with source and destination address, and data.
- Network Interface. This layer corresponds to the Data Link and Physical layers of the OSI model and characterise how data is physically transported via the network, which includes how bits are electrically or optically communicated by hardware devices that interface directly with a network media.

We now need to better define some standards of the application layer and the Host-to-Host layer and that is done in the next sections.

2.1.1 HTTP and HTTPS

HTTP stands for Hypertext transfer protocol. It is accurately described in RFC 2616 [15] as a standard protocol at the application-level intended for distributed and collaborative information systems. The HTTP protocol is a client-server or request-response protocol. Requests are submitted by a user-agent (usually a browser). A client A makes a request to a server B. The server B interprets and processes the request and provides an answer that sends through a response to client A. This protocol is generic and stateless and can be utilised for many more uses than its primary use in hypertext, by extending the request methods, error codes and headers. For example, it can be utilised for name servers or distributed object management systems. On the TCP/IP model, the HTTP protocol is considered in the application layer, that works on the base of the transport protocols, the most usual is the TCP, although in can be adapted to work on UDP, as it is the case with HTTP/3 [16].

2.1.1.1 Client

The client or user-agent can represent any mechanism that represents the user and its intentions, the common use case is a web browser. There realistically are other programs used by some users as software developers, testers and also attackers that can be scarcely used instead of a web browser. Whatever the case, it is invariably the client that establishing a TCP connection, and sends an HTTP request.

2.1.1.2 Server

A Server is a computer that typically runs a software web server, whose function is to connect to the Internet and efficiently manage the network communication. But on a more broad definition the web server (the software) governs the requests and promptly sends the responses. It can typically range between a simple HTTP server that just lets the user access files (static web server) to a dynamic web server, which can perform many things, like process the request, running a program, updating a database and returning the response as a web page.

TCP connections are established via an exchange typically known as the three-way handshake, as seen in Figure 2.1. After TCP connection is established, the client sends an HTTP request. The HTTP methods can be GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS and TRACE [17] . After the server delivers the response, the TCP connection is closed.

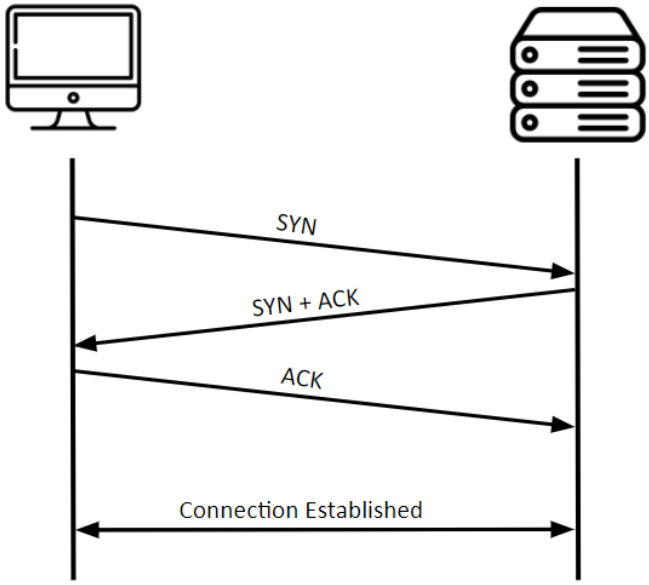


Figure 2.1: TCP Three-way handshake

HTTP messages can be requests when the communication goes from client to server and responses when it goes from server to client. HTTP messages are composed of:

- Start-line

  Either a request line, describe the requests, or a response line in case of HTTP response with its status, or whether successful or failure. It is always a single line and must end with a carriage return followed by a line feed (<CR><LF>).

- HTTP header

  The HTTP headers are colon separated key-value pairs in clear-text, and must end with a carriage return followed by a line feed (<CR><LF>). Although the HTTP methods are case-sensitive, the header fields are case-insensitive. There is no limitation on the number

of fields a header has or the size of each field, but most servers and clients impose some sort of limits for security reasons. The Internet Assigned Numbers Authority (IANA) maintains a permanent registry of header fields [18] that was last updated in 2020-09-01.

- Blank line

  The empty line must consist of only `(<CR><LF>)` and no other white space, and indicates the end of the HTTP header.

- Body

  The Body is not mandatory. It is usually used in POST or PUT requests, and is used to carry the entity-body associated with the request or response.

In Figure 2.2 we can see an HTTP message. It is a request message with a GET method.

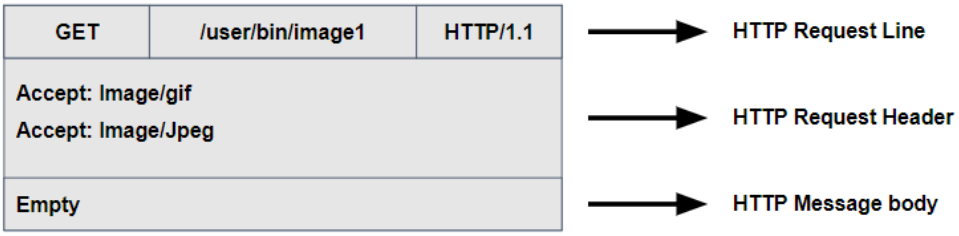


Figure 2.2: HTTP message [1]

HTTPS is the protocol identifier for HTTP over TLS, which means that the HTTP is enclosed in a cryptographic protocol, to provide security to the communication over the network, enforcing Confidentiality, Integrity and Authentication (CIA). HTTPS is defined in RFC 2818 [19] and is often described as establishing a secure channel over an insecure network. In fact, establishing a secure channel over an insecure network is achieved by the Transport Layer Security (TLS) which implement privacy and data integrity, among the two communicating applications [20].

The connection is made private and secure with the use of a symmetric-key algorithm, used to encrypt the data. The algorithm keys are generated for each connection, based on a shared secret negotiated during the TLS handshake, as seen in Figure 2.3. It is during this handshake that the client indicates to the server the setup of the TLS connection.

Because HTTPS is HTTP on top of TLS, all of the underlying HTTP protocol can be encrypted. This includes all HTTP message contents, such as the request URL, query parameters, headers, cookies and body. The most an attacker can discover efficiently is that a TLS connection is undergoing, and the IP addresses and domain names of the active participants.

The use of HTTPS adequately protects against man-in-the-middle attacks, eavesdropping and tampering. HTTPS URLs starts with “HTTPS://” using by default port 443, and HTTP URLs starts with “HTTP://” and use port 80 by default. As the privacy concerns grow and network security naturally becomes a more relevant principle, the extensive use of HTTPS is becoming a modern standard.

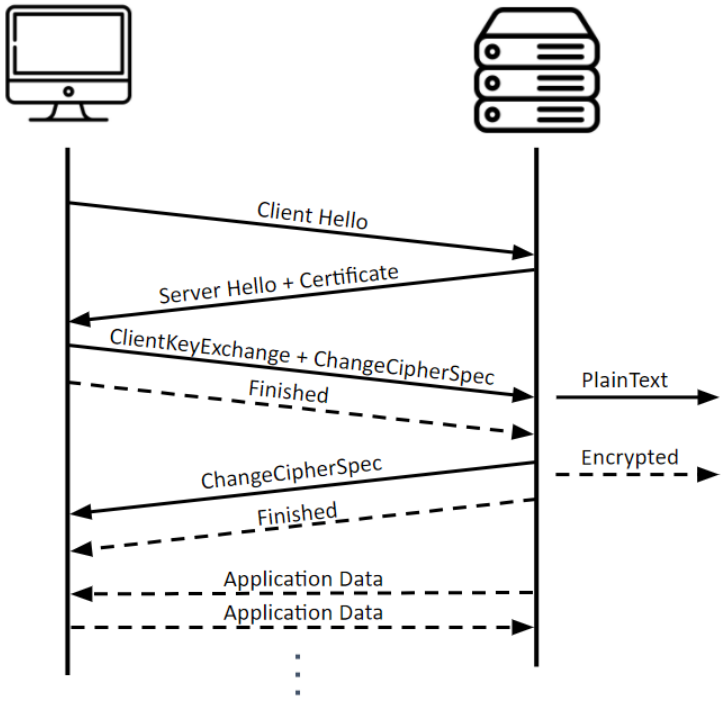


Figure 2.3: TLS handshake

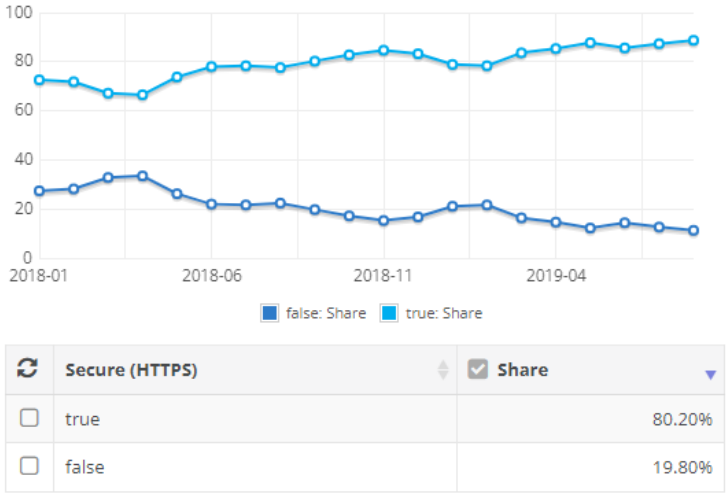
Using net marketshare [21] for market share statistics, we can conclude that from January 2018 to August 2019, 80,20% of the traffic from desktops, laptops and mobile was on HTTPS and as observed in Figure 2.4 the tendency is to rise. Because the packet content is encrypted, HTTPS traffic is not prone to accurately detect certain kinds of web attacks, using traditional methods, such as Deep Packet Inspection (DPI).

For example Pramod *et al.* [22] used DPI to detect SQLi attacks but just on HTTP, because on HTTPS the payload is encrypted. There are additionally some direct and noticeable protocol-related performance costs related to HTTPS, such as the increasing latency. Naylor *et al.* [23] identified some indirect consequences in in-network services like loss of caching and on some services such as parental control or virus scanning. One major result from their published study was that although HTTPS messages are bigger in size, the vital concern is with the TLS handshake because in 50% of TLS communication they study, the hand-shake represents more than 42% of the total data exchanged.

2.1.2 NetFlow

2.1.2.1 Basic Description

NetFlow represents an aggregated flow of traffic obtained through a system of tools that collect, process and export a flow. The flow is aggregated from a group of packets that goes through an observation point during a certain time interval. It is a passive approach to network monitoring, works well in high-speed and high-volume networks. Although the time interval represents a major part in defining the NetFlow, it is equally important to remember that the packets that constitute the



Secure (HTTPS)	Share
true	80.20%
false	19.80%

Figure 2.4: Netshare statistics for HTTPS traffic done by desktops, laptops and mobile from January 2018 to August 2019

flow share some common properties. These common properties are usually the header fields, as for example destination IP, destination port and application header fields. Another part of the NetFlow are some characteristics about the packets, for example the length, and characteristics derived from the packet treatment. The use of flows started in 1991 with *meter*, a process developed to inspect a stream of packets from a communication medium or enclosed by a pair of media [24]. The produced meter aggregated packets belonging to Flows among communicating systems.

2.1.2.2 The Importance of WebFlow

DPI is a widely used technique that got mainstream with the use by all major firewall manufacturing. DPI inspects all individual packets, including header and payload, allowing the analysis and filtering of the packet. In case of a firewall, it can typically allow the packet to pass or discard the packet. Because DPI checks all packets it causes a bottleneck on the network and with the gradual growth of network traffic, it gets increasingly disadvantageous over time.

The practical use of DPI in intrusion detection systems (IDS) can equally create several problems. As the firewall must read the packet, it can also be attacked. For example a specially crafted Session Initiation Protocol (SIP) traffic can trigger a denial-of-service on some Cisco devices [25].

The significant approach to mitigate these problems is the use of some kind of NetFlow. Using NetFlow it is possible to obtain an overview of the packets, in a format that can be rapidly analysed. For this work we intentionally choose to work with flows as the main input, instead of DPI, as DPI presents the drawbacks mentioned before. Hofstede *et al.* [2] mention 5 key benefits of NetFlow:

- Suitable for high-speed networks
- Widely deployed and available. Integration with all major routers, switches and firewalls.
- Widely studied, and used in security analysis, capacity planning, accounting, profiling, for data retention laws, among others.

- Major data reduction can be achieved. As good as 1/2000 of the original volume.
- Flow export is usually less privacy-sensitive than packet export.

A typical flow monitoring setup, as explained by Hofstede *et al.* [2] is represented in Figure 2.5

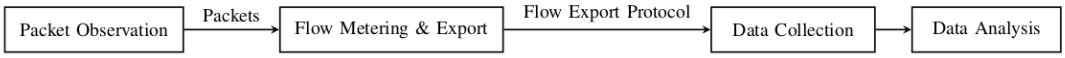


Figure 2.5: Flow monitoring setup [2]

The first step is the packet observation where the packets are processed. In this stage the packets are read, timestamped and truncated. Depending on the capturing tool, there can exist packet sampling and filtering. Packet sampling and filtering objective is to forward only certain packets to the Flow Metering & Export in order to reduce consumption of bandwidth, memory and computation cycles. The second step is Flow Metering & Export where the packets are combined in flows. The packets that constitute the flow, possess some common properties, like time interval, header fields, as for example destination IP or destination port, application header fields, the flow compile length, and characteristics derived from the packet treatment. The combined packets wait passively in a flow cache, until it is considered finished (usually time related), when the flow is exported. The third step is Data Collection which includes aggregation, filtering and data compression. The fourth and ultimate step is Data analysis for traffic profiling, classification, attack detection, anomaly detection, intrusion detection or research among other uses.

The result is a flow record that accurately represents the information related to the traffic represented in a given flow. The fields included in a flow record vary from the technique used in step two, but usually include fields as IPs, ports, packet counts, timestamps and so on.

2.1.2.3 NetFlow V9

The major standard in NetFlow is dictated by CISCO. In the current NetFlow version (Version 9 [3]), the record format is build with a packet header and a minimal of one FlowSet or template, as can be seen in Figure 2.6

Figure 2.6: Fields of the record format in NetFlow version 9 [3]

The fields of the record format are :

- Packet header
  The packet header is composed of:
  Version - The Version of NetFlow records exported in the current packet
  Count - Number of FlowSet records (template + data) in the current packet
  System Up time - Milliseconds since first boot

Unix Seconds - Seconds since 0000 Coordinated Universal Time (UTC) 1970
Sequence Number - Incremental sequence counter of all export packets sent by this export device; this value is cumulative, and it can be used to identify whether any export packets have been missed
Source ID - 32-bit value guarantees uniqueness for all flows exported from a particular device.

- Template FlowSet
  The objective of the template FlowSet is to describe the fields of the data FlowSets, and both can be intermingled within an export packet. Cisco considers that template FlowSet implements a description of the fields that will be present in future data FlowSets [3]. The Template FlowSet format is composed of:

  FlowSet ID - used to distinguish template records from data records.
  Length - total length of this FlowSet.
  Template ID - each template is given a unique ID by the rooter.
  Field Count - number of fields in this template record.
  Field 1 Type - numeric value represents the type of the field.
  Field 1 Length - The length in bytes of the field.
  ... Field n Type
  Field n Length
  In NetFlow version 9 there are 104 fields definitions and 23 fields reserved for future use. The defined fields varied from IP related fields, packet associated fields, time related fields and many more.

- DataFlow
  The DataFlow format is composed of:
  FlowSet ID = Template ID - The FlowSet ID maps to a (previously received) template ID
  Length - Length of the DataFlow set.
  Record N ... Field N - Previously defined in the template record referenced by the FlowSet ID/template ID.
  Padding - 32 bit boundary

2.1.2.4 IPFIX

IPFIX stands for Internet Protocol Flow Information Export, it is a protocol defined by IETF (Internet Engineering Task Force) first documented on RFC 3917 as the preliminary way of exporting IP flow information, obtained in routers, traffic measurement probes, and middleboxes [26].

The objective was to define a standard for collecting and analysing network data [26]. IPFIX was created as a modernised NetFlow v9 protocol, with some extensions and requirements, such as

transport, string variable length encoding, security and template withdrawal message [27]. IPFIX is more comprehensive than NetFlow, as it works with Cisco, but also with all the range of brands and devices. Nowadays it is difficult for any organisation to intentionally keep a vendor specific solution, so IPFIX is a more used protocol for NetFlow collection and analysis. Much like a typical NetFlow setup, IPFIX is composed of four steps.

In Figure 2.7 we can see Packet Observation, Metering Process, Exporting Process and Collecting Process. The Packet Observation step is where the packets are captured and pre-processed.

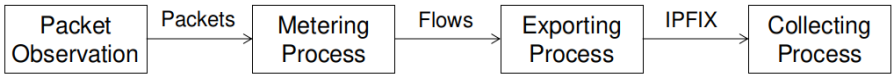


Figure 2.7: The four stages of IPFIX

At this stage, the packets go through capturing, time stamping, truncation, sampling and filtering. The sampling and filtering are set up with rules that determine which packets are processed. The Metering Process step comprehends the aggregation of packets in flows and the exporting of flows. It is in this step that the flow record is defined, containing the measured properties of the flow, like number of bytes of the packets in the flow and other properties like IPs, ports and so on. The fields that can be exported in IPFIX flow records are called Information Elements (IEs). There are 491 IE [28] more than double of the equivalent field definition in NetFlow Version 9. All these fields allow IPFIX to incorporate Internet Assigned Numbers Authority (IANA) -standard list of IEs for multi-vendor interoperability as well as proprietary vendor/enterprise-specific IEs. There is additionally the use of flow caches, as in NetFlow V9. The exporting process can incorporate sampling and filtering. The difference with the sampling and filtering done in the Metering Process step is that in this step, the sampling and filtering are done on the flow record and before, it was done on the packets. The Collecting process is where the flows are stored and can also be subject to further processes, such as data compression, aggregation and summary generation.

The broad range of IPFIX makes it useful for many types of traffic related analysis. For example Matoušek *et al.* [29] have used IPFIX for identification of operating systems from Internet traffic. Hofstede *et al.* [30] presented an extension for the flow exporters that was able to accurately detect flooding attacks in brief instances. They were capable of effectively surmounting the detection delays provoked by flow-based monitoring systems as DPI. Muñoz *et al.* [31] implemented a system using machine learning models, to detect cryptocurrency miners, through the analysis of NetFlow/IPFIX. Toorn *et al.* [32] have investigated the patterns of HTTP(S) dictionary attacks at the flow-level, using NetFlow/IPFIX. All these works used IPFIX, because of its capability of creating a personalised NetFlow according to the study at hand.

2.2 Web Attacks

As the Internet grows, every day there are more and more organisations present on the web. Those organisations work, collect and process large amounts of data, composed of critical and sensitive

personal data, which constitute a point of concern, security wise. Just this last year, because of the COVID-19 pandemic and the trend of working from home, the attack surface grew exponentially.

As stated by ISO 27001 [33] one of the security trinity parts are people. In terms of security concerns, attackers are driven by capacity, opportunity and will. Capacity can be further explained as knowledge and tools. Any attacker from any place in the world, has nowadays unlimited knowledge and tools. It is possible to aggressively target any service present on the web. The necessary tools are free and vast, making it possible for an attacker to intentionally target several entities at the same time. There has been, until now, relative impunity for attackers. All this naturally leads to a growth in attacks, on the web available devices or services.

Opportunity can be further explained as time and vulnerabilities. The vulnerabilities are mostly publicised and public. Lin *et al.* [34] list the most used vulnerability databases as the United States National Vulnerability Database (NVD), the Open Source Vulnerability Database (OSVDB), Carnegie Mellon's CERT and Symantec's Worldwide Intelligence Network Environment(WINE).

The will of the attacker is what escalates the destructive potential of the threat. Whether it is money, ideology or any other motivation, the results are the same. The relentless attacks are more targeted, more advanced and more destructive than ever before.

The European Union Agency for Cybersecurity (ENISA), in its threat landscape report [35] from January 2019 to April 2020, enlist Web-based attacks as the number two cyber threat. There is much work done to detect web attacks, using NetFlow or IPFIX. Zhenqi *et al.* [36] discuss the use of NetFlow to detect DoS, DDoS and Network worm virus traffic. Sperotto *et al.* [37] discuss the state of the art solutions in the use of flow based IDS to detect DoS, Scans, Worms and Botnets. Najafabadi *et al.* [38] discuss the use of NetFlow to detect SSH Brute force attacks. Toorn *et al.* [32] discuss the use of NetFlow/IPFIX to detect brute force attacks in both HTTP and HTTPS.

To the best of our knowledge there is no work on detecting SQLi and XSS attacks using NetFlow or IPFIX for HTTPS traffic. The studies for this kind of attacks are usually done using DPI. For example Pramod *et al.* [22] propose a system using DPI to detect SQLi. For the purpose of this study, we will focus on SQLi and XSS, two of the most used web application attacks.

2.2.1 SQLi

SQLi attacks, were first documented in 1998 [39]. It was considered in the OWASP Top 10 Web Application Security Risks in 2010, 2013 and 2017, as the first security risk experienced by Web applications [8] and the third in 2021. SQL Injection represent a type of injection attack on a web application, where the attacker crafts SQL code into a user input field. Web site features, mostly accessible in the user interface, are affected by SQL injection attack. These features include input fields such as those presented in forms, text fields, password fields, check boxes, radio buttons and sliders, support requests, search functions, feedback fields, shopping carts and even the functions that creates the dynamic web page content. These user input fields are passed into the SQL query,

granting the attacker unauthorised and unlimited access to the database. If the attacker gains access to the database, it undermines the confidentiality, integrity and authority of the data. The results can be devastating. SQL Injection comprehend several types of attacks, as detailed in Table 2.1.

Table 2.1: SQLi attack types [5]

Types of Attack	Working Method
Tautologies	SQL injection queries are injected into one or more conditional statements so that they are always evaluated to be true.
Logically Incorrect Queries	Using error messages rejected by the database to find useful data facilitating injection of the backend database.
Union Query	Injected query is joined with a safe query using the keyword UNION in order to get information related to other tables from the application.
Stored Procedure	Many databases have built-in stored procedures. The attacker executes these built-in functions using malicious SQL Injection codes.
Piggy-Backed Queries	Additional malicious queries are inserted into. an original into query.
Inference	An attacker derives logical conclusions from the answer to a true/false question concerning the database.
Blind Injection	Information is collected by inferring from the replies of the page after questioning the server true/false questions.
Timing Attacks	An attacker collects information by observing the response time (behaviour) of the database.
Alternate Encodings	It aims to avoid being identified by secure defensive coding and automated prevention mechanisms. Hence, it helps the attackers to evade detection. It is usually combined with other attack techniques.

The major types of mitigation are user input sanitisation, use of parameterised queries, use of stored procedures [40] and parameter tampering [41]. As for detecting there must exist some kind of packet inspection. For example, Pramod *et al.* [22] propose a signature based approach to detect SQL injection, by using a pattern matching algorithm to detect an attack. One of the major concerns nowadays is that the use of HTTPS is not compatible with DPI. HTTPS prevents the use of many tools available to detect SQLi as they depend on reading the packet content.

2.2.2 XSS

Cross Site Scripting (XSS) attack is a specific kind of attack which allows a malicious user to inject a malicious script in a benign or trusted website. The attacker is then able to deliver a malicious script to an unsuspecting user, gaining access to cookies, session tokens, sensitive information or tampering with the information provided. XSS was considered the third security risk experienced by Web applications in OWASP Top 10 of 2010, 2013 and 2021, and the seventh in 2017. It is furthermore the second most prevalent issue in the OWASP Top 10 list, and is found within around two-thirds of all applications. There are three types of XSS:

- Dom Based XSS

  Domed Based XSS, is also known as type-0 XSS. It is a client-side attack. With the imperative need for a more dynamic experience, the use of applications that use the client side to perform most of the presentation actions, and just pulling data on demand from the server, lead to the exploding popularity of Document Object Model (DOM) based vulnerabilities.

The attacker code is parsed in the client side and is written to the DOM by the web application. This specific type of attack is presented on pages using Javascript, AJAX, VBScript and other client-side languages.

- Stored XSS
  Stored XSS is also known as type-1 XSS or persisted XSS. It is undoubtedly considered a high or critical risk. In this peculiar type of attack, the malicious code is efficiently stored on the server-side and then is viewed or parsed at a later time by an unsuspecting user or even by a site administrator.

- Reflected XSS
  Reflected XSS is also known as type-2 XSS or non persistent XSS. It is considered the most basic web vulnerability attack. On a reflected XSS, the application or API, seamlessly incorporates unsanitised or unescaped user input as part of the output. The malicious code is included in the response and is promptly executed. It is a server-side vulnerability, as the malicious code is parsed at the server-side and not on the client-side. The most typical form of attack is through email or managing a neutral site, producing an unsuspecting URL, pointing to a trusted site or an attacker controlled page, containing the XSS vector, that is executed inside the users browser.

Dayal *et al.* [42] produced a comprehensive inspection of XSS attack and discussed a few tools utilised for detecting XSS and some rules for preventing XSS. Liu *et al.* [43] produced a survey of Exploitation and Detection Methods of XSS Vulnerabilities, and discussed alternative methods for detecting this kind of vulnerabilities. In terms of detection, all of the studies analysed used some kind of DPI [44] [45] [46]. As with SQLi, one of the major concerns nowadays is that the use of HTTPS is not compatible with DPI. That prevents the use of many tools to convenient detect XSS as they depend on reading the packet content.

2.3 Machine Learning

Machine learning is a set of algorithms that provides the ability to automatically learn and improve from experience or previous events, without being explicitly programmed. Although an algorithm, in this context, represents just a sequence of statistical processing steps, they are capable of recognising patterns and features in huge datasets in such a way that supports decision taking and more accurate predictions. The distinctive methods of machine learning are Supervised machine learning, Unsupervised machine learning, Semi-supervised machine learning and Reinforcement machine learning. Rincy *et al.* [4], compiled the classification of machine learning algorithms, as shown in Figure 2.8

Buczak *et al.* [47] conducted a survey of machine learning methods for cyber security intrusion detection. They tried developing a map of the type of method best indicated for each type of attack. What they discovered was that the richness and complexity of the methods made it

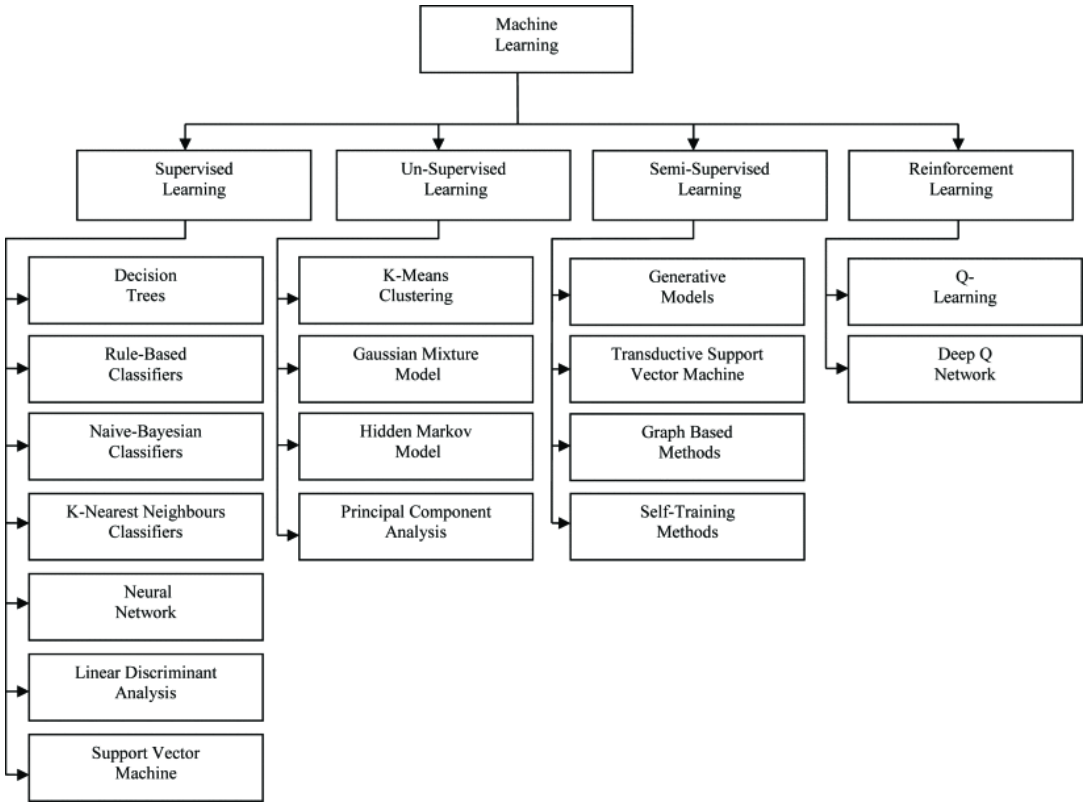


Figure 2.8: Classification of Machine Learning Algorithms [4]

impossible to construct such a framework of correspondences. Nevertheless, for the type of study, this work focuses, with copious amount of data, structured flows, but with unstructured content, and many types of web attacks implementations, the most promised system of machine learning, is unsupervised machine learning

2.3.0.1 Supervised Machine Learning

Supervised machine learning, is a machine learning technique that starts by building a dataset and knowing how that data is correctly classified. This labelled dataset is the training data, used to train the model. This training entails an external entity that indicates when the model is producing the desired result or not. The objective is to identify patterns in data, in a way that helps the analysis of the data. A typical supervised learning algorithm is represented in Figure 2.9.

In supervised machine learning, starting with an input variable P and an output variable Q, the target of the algorithm is to study the mapping function to the output variable $Q = f(P)$. There are different algorithms to achieve this, such as:

- Decision trees: Is a predictive algorithm where the input data or observations are represented on the branches, and the conclusions are represented on the leaves.
- Rule based Classifiers: Is a descriptive model, based on rules. The classic if / then / else, where there is antecedent and consequent.

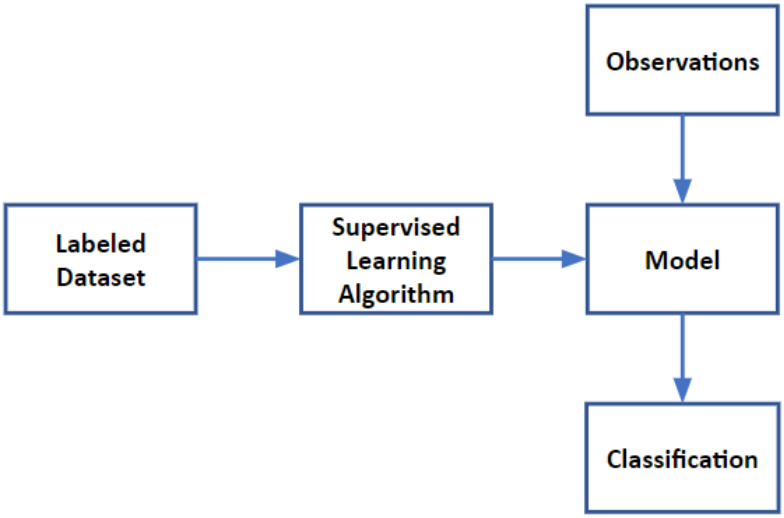


Figure 2.9: Supervised learning algorithm

- Naive-Bayesian classifier: Based on Baye's Theorem, it is a group of algorithms that work with the motto that each pair of features to be classified are independent of one another.
- k- Nearest Neighbour classifiers: In this algorithm, all cases are classified, and new cases are classified based on some similarity measure.
- Neural Network: Inspired by Brain neurons, a neural network algorithm works in several layers of analysing and learning data, in an incremental way.
- Support Vector Machines: Using a hyperplane with the dimensional space of the problem features, to separate two data classes and classify the data points

2.3.0.2 Unsupervised Machine Learning

Unsupervised machine learning technique is based on a large dataset, and through the use of algorithms recognises a structure and extracts features, sorts and classifies data, all without any external validation. This technique is extremely effective for discovering unseen structures in the data and for tasks like anomaly detection. A typical unsupervised learning algorithm works as described in Figure 2.10.

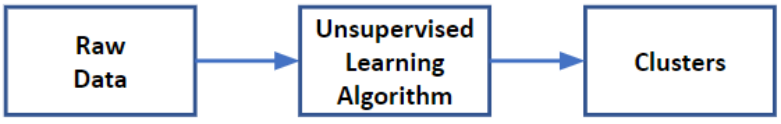


Figure 2.10: Unsupervised learning algorithm

In unsupervised machine learning there is also an input variable P, but there is no output variable. There are different algorithms to achieve this, such as:

- Clustering: More than an algorithm, clustering is a set of techniques that detects patterns in high-dimensional unlabelled data. There are multiple ways of clustering. In hierarchical clustering, data points are grouped by the distance among them. In centroid models like K-means, individual clusters are represented by their mean vector. In distribution models like Expectation Maximisation algorithm, the groups go along to a statistical distribution. In Density models the data points are grouped in dense and connected regions, as for example the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm. In graph models each cluster is defined as a set of connected nodes where each node includes an edge to at least one other node in the set.

- Gaussian Mixture Model: This is a probabilistic model, working on the basis that all data points originate from a mixture of a finite number of Gaussian distributions.

- Hidden Markov Model: This model addresses the problem as a Markov process, divided in two components, one being an observable component and the other an unobservable.

- Principal Component Analysis: This approach uses a dimensionality-reduction method to reduce the dimensions of the data, creating a smaller set that contains most of the relevant information of the original data.

2.4 Similarity Search

Similarity search is the most generic name for a variety of processes that all work on the same idea of exploring (usually extremely huge) areas of items with the sole accessible comparator being the similarity between any two objects. This is becoming more relevant in an age of massive information repositories with items that have no natural order, such as large collections of photographs, music, and other complex digital objects. Finding *nearest neighbours* is a relatively common assignment. Similarity search consider uses, such as detecting duplicate or similar documents, as well as audio/video search. Although employing brute force to search for all possible combinations, it will get the precise nearest neighbour; but this method is not scalable. This problem has been the subject of considerable research into approximate algorithms. Although these algorithms cannot ensure that we will get a precise answer, they will almost always produce a good estimate. These algorithms are more scalable and quicker. To work with payloads constituted by keys and values, a token-based algorithm is the best option. The most common algorithms are the Jaccard Distance, the Cosine Distance and the Euclidean Distance. Both the Cosine Distance and the Euclidean Distance measure the distance between sets, based on each representation in space. Euclidean distance represent the actual difference of individual numerical characteristics and the Cosine Distance is used to discern between directions. The Jaccard Distance is the level of discrimination between two sets, measured by the ratio of distinct elements to all elements in those sets.

2.4.1 Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a set of functions allowing hashing data points into buckets, so that data points close to each other are likely to be in the same bucket, while data points further apart are likely to be in distinct buckets. It identifies observations with varying degrees of resemblance much easier. Examining data for similar objects is a fundamental data-mining challenge. When looking for pairs of comparable objects, the simplest strategy is to look at every pair of items. Even with abundant hardware capabilities, looking at all pairings of items when working with a huge dataset is computationally expensive. In LSH, we can use several hash functions that are not the usual kind of hash functions. Rather, they are deliberately built to have the property that if two objects are similar, they are far more likely to be placed in the same buckets of a hash function than if they are not similar. The candidate pairs, which are pairs of items that end up in the same bucket for at least one of the hash algorithms, may next be examined. To begin, we must transform the documents we wish to analyse into sets, allowing us to examine textual similarity of documents as sets with a substantial overlap. The Jaccard similarity of sets, which is the ratio of the sizes of their intersection and union, is used to quantify their resemblance.

2.4.1.1 Jaccard Similarity

Finding textually comparable documents is one of the issues that Jaccard similarity solves. This type of resemblance is referred to as character-level similarity, as opposed to similar meaning. The Jaccard Similarity of two sets S and T is given by $|S \cap T|/|S \cup T|$. In Figure 2.11 we can see an example that there are three elements at the intersection of the sets and a total of eight elements, which make a Jaccard Similarity of 3/8 or 37%.

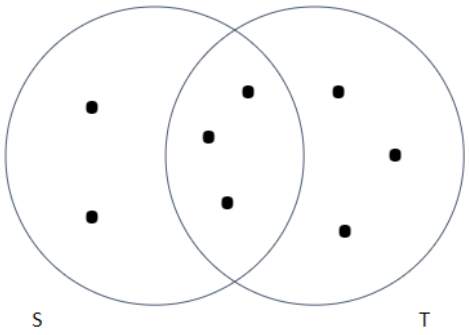


Figure 2.11: Jaccard similarity of sets S and T

2.4.1.2 Shingling

A k-shingling is a set of distinct shingles (consequently n-grams) in natural language processing, each of which is made up of continuous subsequences of tokens inside a document and may be used to determine document similarity. The symbol k stands for the number of tokens in each shingle that has been chosen or solved for. By using shingles, documents that have the same short pieces in them will have many common elements in their sets. This also occur if those elements

appear in different orders or in different keys or values of a dictionary. We may associate each value with the set of k-shingles that appear one or more times inside that value by defining a k-shingle as any substring of a k length present in the value. Leskovec *et al.* [48] presents as an example a document *D* as the string `abcdabd`, that with a K=2, present a set of 2-shingles as `{ab,bc,cd,da,bd}`. We must note that within *D*, the substring `ab` appears twice, but as a shingle appears only once. That is the intentionally behaviour as we want a *set* of shingles, that is a collection without repetitions.

2.4.1.3 Hashing

Rather than utilising substrings as shingles, we use a hash function to map strings of length k to a number of buckets and consider the bucket number as the shingle. The set of integers that constitute a document is thus the set of bucket numbers of one or more k-shingles that appear in the document. The hashing is done to create a signature that represents the set in a smaller version. The hashing also makes the signature size uniform, as it is always the same size. This can be done if the hashing function can guarantee that the Jaccard similarity of the signature is the same as the Jaccard similarity of the sets. After hashing the probability of two hashing being the same is directly connected to the similarity of the original sets. Considering two Payloads, `P1` and `P2` and denoting `H` as the function that hashes a set, then:

- `similarity(P1,P2)` is high $\implies$ `Probability(H(P1) == H(P2))` is high
- `similarity(P1,P2)` is low $\implies$ `Probability(H(P1) == H(P2))` is low

We call this probabilistic data structure for computing Jaccard similarity between sets of MinHash.

Permutations

Even after creating and hashing a set of shingles, the storage space required is still significant. A set must be reduced to a smaller representation, i.e. a hash, known as signature. We recall that for a signature to maintain its utility, one must be able to compare the signatures of two sets and estimate the Jaccard similarity of the underlying sets only based on these signatures. The signatures are unlikely to match the precise similarity of the sets represented, but the offered approximation is precise enough. The signatures we want to make for sets are the result of multiple calculations, each of which is a MinHash of the whole group of sets. If a collection of sets is represented as a boolean matrix, with the columns corresponding to the sets and the rows corresponding to the elements, the matrix values represents when an element is present in a set.

For example, Table 2.2 represents a collection of sets. When element `a` is present in each of the Sets `S` it has a value of `1`. In the example element `a` is present in Set `S1` and as so the value in position `(a,S1)` is `1`.

To produce a MinHash of a set, is to perform a random permutation of the rows. Expected similarity of two signatures is equal to the Jaccard similarity of the columns. The signature is the result of numerous permutations and as so, the longer the signatures, *i.e.* the more permutations

Table 2.2: Boolean matrix representing a collection of sets

Elements	S1	S2	S3	S4
a	0	0	1	0
b	0	1	1	0
c	0	0	0	1
d	1	0	0	0
e	1	1	0	0

done, the lower the error. The Jaccard similarity is given by the rows with 1. In Table 2.3, we can see a permutation of the matrix in Table 2.2.

Table 2.3: Permutation of the Matrix in Table 2.2

Elements	S1	S2	S3	S4
a	1	1	0	0
b	0	0	0	1
c	1	0	0	0
d	0	1	1	1
e	0	0	1	0

This theoretical approach to MinHash is not feasible in large sets, however it is possible to simulate the effect of a random permutation by a random hash function that maps row numbers to as many buckets as there are rows.

Threshold

The core concept behind LSH is to discover an algorithm that indicates if two signatures constitute a candidate pair or not, based on whether their similarity is larger than a threshold t. Dividing the signature matrix into b bands of r rows each is an effective approach to pick the hashings. There is a hash function for each band that takes r integer vectors (the fraction of one set within that band) and hashes them to a vast number of buckets. We may use the same hash algorithm for all bands, but each band has its own bucket array, thus columns with the same vector but on different bands will not hash to the same bucket. When using MinHash signatures for the items, dividing the signature matrix into b bands of r rows each is an effective technique to pick the hashings. If we employ b bands with r rows and a pair of documents has a Jaccard similarity of s. The threshold, or the value of similarity s at which the chance of becoming a candidate equals the chance of not becoming one, is a function of b and r. Pairs with similarity above the threshold are extremely likely to become candidates for a big b and r, whereas pairs with similarity below the threshold are unlikely to become candidates.

2.5 Related Work

In order to perform anomalous traffic detection, we have two kinds of approaches presented in the literature. The more traditional one employs Deep Packet Inspection (DPI) and the second uses NetFlows.

DPI

The core pillars of current online security protocols, security functions, user service, and network management are all integrated in DPI. DPI is a widely used technique that usually relies on pattern matching algorithms for classification, but which are computational complex. DPI analyses the contents of data packets using specified criteria that the system administrator has preprogrammed. It then determines how to respond to the risks it has discovered. DPI can not only detect threats, but it can also determine where they came from based on the contents of the packet and its header, i.e., it identify the application or service that initiated the attack. DPI may also be configured to use filters to detect and reroute network traffic that originates from a certain online service or IP address. DPI examines the contents of packets travelling through a certain location and makes real-time judgements based on network administrator-defined criteria. DPI may inspect the contents of communications and determine which service they originate from. The analyser in this method thoroughly examines the contents of each package. There are many commercial products based on DPI, and there is many literature about it. Rescio *et.al.* [49] presents a Benchmark and Comparison of the major commercial products and lists several studies. Cheng *et.al.* [50] goes even further by developing effective algorithms for analysing the most used information protocols and to create a software system for collecting statistical data.

For safe transactions, HTTPS is becoming increasingly popular. The majority of popular websites have made HTTPS the default option. With the rise of encrypted communication, new issues in network security monitoring and analysis have arisen. One of solution often presented is the use of a proxy server. Jarmoc [51] discusses how a proxy server works and the risks associated with it. Interception proxies inject themselves into the traffic flow and terminate the clients request in order to inspect plain-text contents of conversations via SSL. The intercepting proxy sends a second request to the server on behalf of the client. As a result of this behaviour, one end-to-end session becomes two discrete but connected point-to-point sessions. While transferring between the two encrypted sessions, the purpose is to provide access to the plain-text session data.

This form of interception has a price tag attached to it. Intercepting SSL-encrypted connections loses some privacy and integrity in exchange for content examination, but the cost is the threat of endpoint validation and authenticity. SSL interception proxy implementers and designers should think about these hazards and how their solutions work in uncommon situations. Jarmoc [51] presents several risks introduced by this approach, which are:

- Legal exposure: As a result of reviewing communications intended to be encrypted and secret, a business may practice an illegal activity by using an SSL interception proxy. Legal

hazards should be properly considered by implementing organisations.

- Increased threat surface: Interception proxies are used to examine the plain-text contents of otherwise encrypted transmission. As a result, making a single location where all encrypted sessions can be read in plain text is a high-value target for an attacker
- Decreased cipher strength: When an intercepting proxy is used, two point-to-point encrypted connections are created. Since these sessions negotiate their cipher suites separately, either the client-side or server-side sessions may utilise a weaker encryption than a session between the client endpoint and the server endpoint.
- Transitive trust: Basically transitive trust occurs when *the client trusts the proxy and the proxy trusts the server, then the client trusts the server.* This defect might present itself in a variety of ways, passing along some problems as Self-signed, Expired or Revoked certificates.

Sherry *et.al.* [52] propose a solution to DPI over TLS by the use of BlindBox. BlindBox is a middlebox implementation, with some privacy models, preventing the middlebox from accessing traffic that does not identifies as an attack by the rule generator. However, this approach is supported on the adoption of a new end-to-end encryption protocol to replace HTTPS altogether.

NetFlows

The use of NetFlows presented in some literature evolve around HTTP traffic. The few times where authors work with HTTPS traffic, they keep the work to attacks different from the ones based on payload text, as SQLi and XSS. Hou *et.al.* [53] presents a solution for identifying DDoS traffic with NetFlow feature selection and machine learning. Najafabadi *et.al.* [54] put forward a scheme to use machine learning approach for the detection of SSH brute force attacks using NetFlow. Bakhshandeh *et.al.* [55] propose an efficient user identification approach based on NetFlow analysis. We already stated about some of these works in Section 2.2 and the lack of literature using HTTPS based NetFlow to detect attacks. One of the few exceptions is the work done by Toorn [32] where they investigate the patterns of HTTPS dictionary attacks in NetFlow. Their approach lacks the analysis of web attacks like SQLi and XSS.

Chapter 3

Proposed Approach

This chapter presents the challenges faced to solve the problem of detecting web attacks over HTTPS network traffic, as well as the solution we propose for this. Section 3.1 states the challenges in managing HTTPS network traffic, Section 3.2 presents an overview of the proposed solution and Section 3.3 details its main modules.

3.1 Challenges

3.1.1 Capture Suspicious HTTPS Network Traffic Related to Web Application Attacks

HTTPS stands for HTTP over TLS, which encapsulates the HTTP protocol with cryptographic primitives, to provide security to the communication over the network, enforcing confidentiality, integrity and authentication. As a practical consequence, this security enhancement prevents one from having access to the plain-text payload during network traffic analysis. Decrypting the HTTPS packets enable one to access the payloads in plaintext. This can be achieved by the use of a proxy server that captures all the traffic, decrypt and access it, and finally re-encrypt it. In practice, as seen in Figure 3.1, the proxy does not decrypt the direct connection between the client and the server, but establishes two different connections, acting as a client to the server and as a server to the client, and so sharing an encryption key with each other.

Having access to the payloads means that it is possible to employ Deep Packet Inspection (DPI) techniques to inspect their content. The problem with this approach is that it tends to create a bottleneck in the network and decreases the overall security. The bottleneck is caused by the fact that decrypting data and evaluating it in real time is a processor-intensive task that many hardware-based security devices cannot handle. Moreover, the fact that there is an additional component where the payload resides decrypted, between the client and the final server, increases the attack surface, decreasing the encryption strength and threatening the overall data security of HTTPS traffic.

This thesis proposes a solution to deal with HTTPS traffic that resorts to NetFlows. NetFlow is a network protocol for monitoring network flows and collecting IP traffic statistics. It provides snapshots of the network traffic flow and hints about its volume and communication channels.

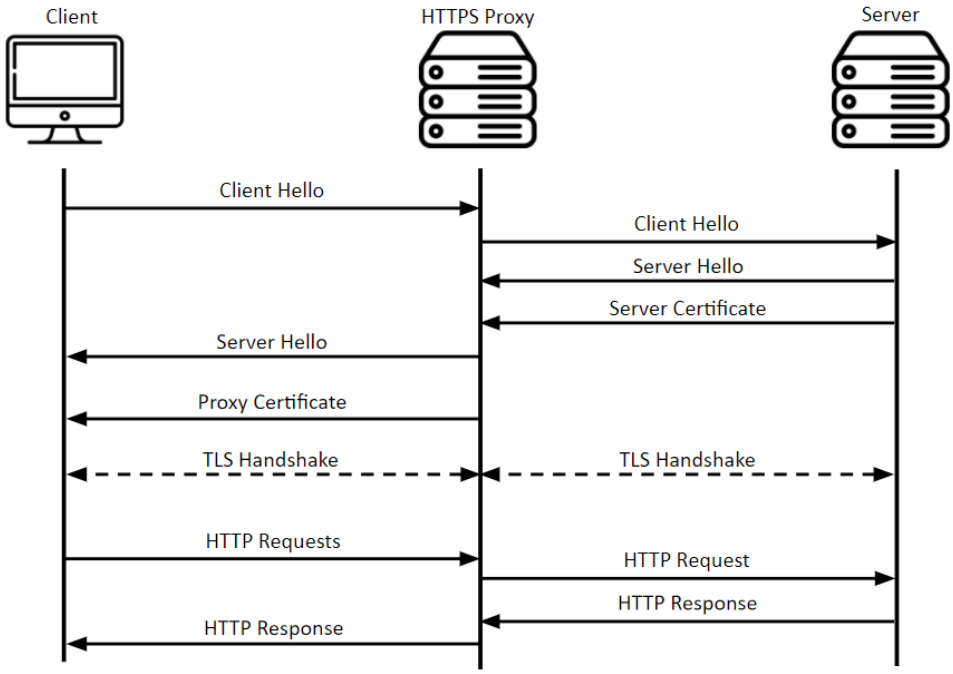


Figure 3.1: Intermediary HTTPS Proxy

Using a NetFlow monitoring system enables more efficient and effective monitoring and analysis of these flow records about network traffic. More specifically, throughout this thesis we use the term *WebFlows* to clarify when it deals with web network traffic.

There are two premises to identify anomalous traffic through WebFlows:

1. Any classification system requires a large enough dataset. In this case, there must exist a large amount of collected traffic, to enable the comparison of WebFlows and a segregation between the normal and the anomalous traffic.

2. The collected fields in the WebFlow must be useful for the overall solution.

As pointed out in Section 2.5, some works already try to detect anomalous traffic using WebFlows (e.g., [53], [54], [55], [32]). However, these studies are based on HTTP traffic and none of them investigated web application attacks (e.g., SQLi and XSS) over HTTPS, as we intend to conduct in this thesis. The few existing studies for HTTPS (e.g., [32]) try to detect only deviations to the normal behaviour network traffic, such as DoS and brute force attacks.

WebFlow by itself will not result in a binary truth. It will produce a condensed version of the traffic with many important information, but we have to find out a way to interpret the data and extract useful information. One of the most usual use cases in web attacks is the use of fuzzers and scanners by attackers for detecting and exploiting vulnerabilities. These automated tools can perform vulnerability searches in web applications, and their noticeable consequence is the large volume of network traffic they produce. This is the first premise for a useful WebFlow.

We propose the use of unsupervised machine learning to process WebFlows which will allow us to analyse a WebFlow without previous knowledge of its content. We intend to collect the right

fields, as we previously stated for the second premise of a useful WebFlow, so we can use a clustering algorithm over WebFlows to aggregate them into clusters by similarities existing between them, and identify the traffic of a scanner.

3.1.2 Web Application Attacks Constraints

In terms of web attack detection some attacks are detectable through the observations of a WebFlow. That is the case of brute force attacks, for example HTTP(S) dictionary attacks.

On the other spectrum, there are attacks where we need to analyse the content of the payload, to detect the attack. SQLi and XSS are examples of these types of attacks. However, we cannot directly identify these in a WebFlow because we do not have access to the payload.

At the first instance, we will conduct anomalous traffic identification with the use of a clustering algorithm over WebFlows, but we need to develop some more resources to completely guarantee that we identified traffic from a scanner. We want to have access to the payload but without any additional decrypting mechanism of the traffic. Since HTTPS is an application protocol, and the web application is at the end-point of the communication, the payload included on WebFlows will be decrypted by the web application, i.e., the web server that stores the web application, and it will be logged by the web server. Therefore, we propose a solution that captures these logs and analyses them to discover web application attacks provided from suspicious WebFlows. However, analysing these logs is not straightforward and we identified some problems:

1. POST requests are not logged by the web server, in which they are the most interesting request we want to analyse as we have guarantee they contain data send by users / scanners.

2. Logs register all communication steps which hardens the task of discovering attacks in a fast and efficient way.

3. The log format is not be the best to conduct such inspections.

To cope with these problems, the solution from this thesis will also include a module to register and capture POST requests, and a filter and a parser to, respectively, obtain only those logs containing user data from GET and POST requests and transform them into a format that will be easier to process.

3.1.3 How to Find Web Application Attacks from a Scanner?

Observing the network traffic generated by the scanner, one may observe some characteristics, such as the use of certain words and the use of variations of strings. For example, in Listing 3.1 four variations of the same payload are presented. If one uses a DPI method with a pattern matching solution, not only will the database need to have all variations, but the search will also be computationally intensive, since it has to search entries on a huge database in a timely manner.

The solution from this thesis applies a Similarity Search approach, namely the use of LSH, an efficient algorithm for searches over large datasets. This algorithm family focuses on constructing

```
bug=31&form_bug=submit
bug=50&form_bug=submit
bug=129&form_bug=submit
bug=28&form_bug=submit
```

Listing 3.1: Example of payloads from a scanner

condensed representations of a given input data that may be compared afterwards. In reality, applying LSH algorithms on similar payloads produces nearly identical hashes, in contrast to cryptographic hashes (e.g., SHA256), where hashing two similar strings yields two significantly different hashes.

3.2 Solution Overview

This section presents an overview of the solution we propose to detect web attacks over HTTPS network traffic without decrypting the payload on the communication. For the solution to detect an attack that resides in the payload, it has to access and analyse its content.

Analysing the payload path, one can observe all points where it passes, i.e., from its construction in the client browser, passing through the network and the web server, until it arrives at the web application. From this analysis it is observed that in the first section of the path one can collect the network traffic between the client and the web server, with this part it is viable to construct the WebFlow, although the payload is not accessible. On the last part of the path, the payload enters in the web Server, is decrypted and logged by the web server, and thus allowing access to it.

The overall idea is:

1. Detecting anomalous traffic with high chances of including web attacks like SQLi or XSS.
   Any traffic that strays from the normal traffic conducted by humans, including scanners and fuzzers, will have some different characteristics. With the use of WebFlow analysis we could identify these traffic.

2. Analyse the payload content of the traffic previously detected.
   It is not feasible to read and analyse all the payloads content, but, after identifying abnormal traffic, we can determine a subset of the network traffic where to look, and with the use of various tools detecting the presence of scanners and fuzzers that performed SQLi or XSS attacks.

3. Continuous improvement process
   Independent of the tool used to analyse the payload content, it must be kept up to date, in order to adjust them to the repeated evolution of attack tools.

The Architecture of the proposed solution passes for correctly identifying anomalous traffic, grabbing that traffic from the server log and identifying attacks. Figure 3.2 shows the architecture

of the solution, which comprises six entities; *WebFlow, WebApp, Log Extractor, Similarity Search Engine, String Analyser* and *SOC*.

For identifying anomalous traffic, like those from scanners, we resort to WebFlows. For grabbing the logs from the server associated with the monitoring WebApp the Log Extractor filters and parses the logs; next for identifying attacks we use a two step approach: first we use similarity search algorithms in the Similarity Search Engine, and second if we still cannot detect an attack in a safety threshold of correctness, we use several string analysers in the String Analyser. At the end, the SOC will receive an alarm if an attack is identified.

3.3 Main Modules

3.3.1 Detecting Anomalous Traffic

To detect anomalous traffic, like those originated from a scanner or a fuzzer, we resort to the use of WebFlow, as there are numerous options to use. In order to create a WebFlow, we capture the network traffic in the form of PCAP and convert it into a WebFlow.

Our objective at this point is to segregate anomalous traffic. We must accommodate for the high volume of traffic and correctly segregate it without the need of any previous work like the use of training data. Our approach uses unsupervised machine learning, *i.e.* clustering algorithms, to find natural groups in the WebFlow. This technique is extremely effective for discovering unseen groups in the data. Tasks like anomaly detection can be resolved with this kind of algorithms, as data points that are in the same group have similar features and data points in different groups have different features. We propose the use of clustering to detect patterns in high-dimensional unlabelled data. The result of this stage is the creation of several clusters of WebFlows.

As the traffic from a scanner considerably differs from the normal traffic, the clusters formed by the algorithm will group traffic with similar features, like source and volume, two of the major characteristics that identifies a scanner. This will allow the correct identification of the scanner.

In this last step the collector also converts the IP format to decimal numbers and the TCP flags to integer. An example of the final result of this process is displayed in Table 3.1

Table 3.1: Example of WebFlow

sTime	sIP	dIP	sPort	dPort	protocol	packets	bytes	flags	duration	eTime
2021/03/29T18:12:06.121	3232249961	3232249960	42610	443	6	7	919	27	0.008	2021/03/29T18:12:06.129
2021/03/29T18:12:06.122	3232249960	3232249961	443	42610	6	4	1766	27	0.007	2021/03/29T18:12:06.129
2021/03/29T18:12:06.380	3232249961	3232249960	42612	443	6	7	919	27	0.005	2021/03/29T18:12:06.385
2021/03/29T18:12:06.381	3232249960	3232249961	443	42612	6	4	1766	27	0.004	2021/03/29T18:12:06.385
2021/03/29T18:12:06.788	3232249961	3232249960	42614	443	6	7	919	27	0.004	2021/03/29T18:12:06.792
2021/03/29T18:12:06.788	3232249960	3232249961	443	42614	6	4	1766	27	0.004	2021/03/29T18:12:06.792
2021/03/29T18:12:08.983	3232249961	3232249960	42616	443	6	7	919	27	0.007	2021/03/29T18:12:08.990
2021/03/29T18:12:08.983	3232249960	3232249961	443	42616	6	4	1766	27	0.007	2021/03/29T18:12:08.990
2021/03/29T18:12:09.165	3232249961	3232249960	42618	443	6	7	919	27	0.005	2021/03/29T18:12:09.170
2021/03/29T18:12:09.165	3232249960	3232249961	443	42618	6	4	1766	27	0.005	2021/03/29T18:12:09.170
2021/03/29T18:12:09.621	3232249961	3232249960	42620	443	6	7	919	27	0.005	2021/03/29T18:12:09.626
2021/03/29T18:12:09.622	3232249960	3232249961	443	42620	6	4	1766	27	0.004	2021/03/29T18:12:09.626
2021/03/29T18:07:04.716	3232249961	4026531834	47768	1900	17	4	780	0	3.015	2021/03/29T18:07:07.731
2021/03/29T18:12:11.870	3232249961	3232249960	42622	443	6	7	919	27	0.006	2021/03/29T18:12:11.876
2021/03/29T18:12:11.870	3232249960	3232249961	443	42622	6	4	1766	27	0.006	2021/03/29T18:12:11.876
2021/03/29T18:12:12.079	3232249961	3232249960	42624	443	6	7	919	27	0.003	2021/03/29T18:12:12.082
2021/03/29T18:12:12.079	3232249960	3232249961	443	42624	6	4	1766	27	0.003	2021/03/29T18:12:12.082

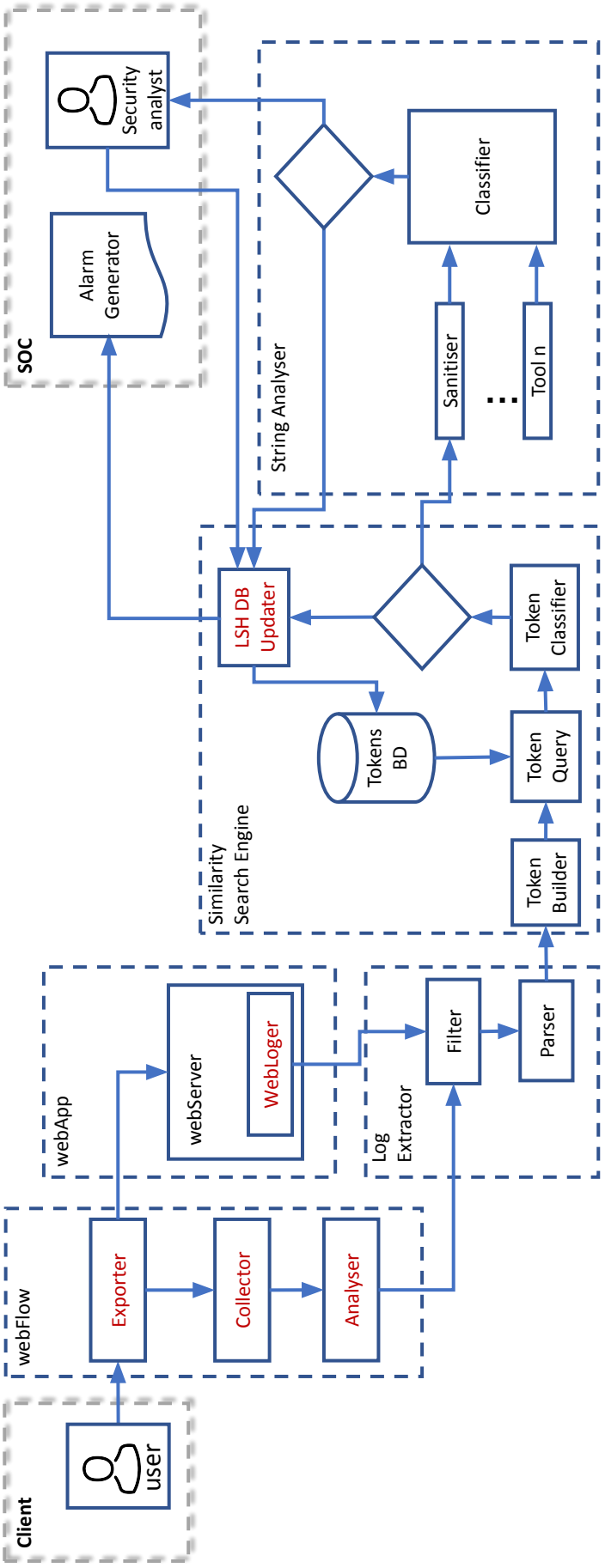


Figure 3.2: Architecture of the proposed solution to detect web attacks over HTTPS network traffic.

3.3.1.1 Analyser

With a WebFlow available, a tool was needed to analyse it, and pinpoint the network traffic done by the scanners. The major constraints of this tool was the need to keep it fully autonomous and capable of analysing a WebFlow without any previous knowledge of its content. We resort to the use of unsupervised machine learning. To understand which algorithm is the best to process the WebFlows, we conducted a set of experiments, which we present next.

DBSCAN

First, we tried the analyser with the DBSCAN algorithm. This was not a good solution as the clusters had different densities. Although DBSCAN identifies outliers and clearly finds arbitrarily sized and shaped clusters, it has problems setting the distance thresholds because of cluster density varieties, and, therefore, it presented too many clusters with no clear distinction between them.

Mean Shift

As a second experiment we used the Mean Shift algorithm, but with the same result: too many clusters with no clear distinction between them. In this case, the problem remain the uneven density of the WebFlow dataset. This is a centroid based algorithm, that determines centroid candidates to be the mean of the points within a given region. In our case, it produces a large set of centroids which does not allow us to identify anomalous traffic.

K-Means

At last we used K-Means. K-Means is a flat clustering algorithm, which means that the number of clusters are already known. It tries to segregate the dataset into K clusters, in a way that each element of the dataset belongs to the cluster with the nearest centroid. One of its advantages is the speed, because of its linear complexity of $O(n)$, as all it is doing is calculating the distances between points and group centres.

Because this algorithm relies upon the distance between points, and we have features with different measures, we need to standardise all the features into the same scale. In standardisation, each feature is scaled by subtracting the mean and dividing by the standard deviation. This shifts the distribution to a mean of zero and a standard deviation on one. We tried different scalers, but got the best results with StandardScaler and MixMaxScaler.

In StandardScaler the mean is removed and the data is scaled to unit variance. This maintains some influence of the outliers point in the calculation of the mean and of the standard deviation. The StandardScaler value z is calculated by Formula 3.1.

$$z = (x - u)/s \tag{3.1}$$

where x is the value being normalised, u is the mean of the training samples, and s is the standard deviation of the training samples. The mean is calculated by Formula 3.2

$$u = sum(x)/count(x) \tag{3.2}$$

and the standard deviation by Formula 3.3

$$s = \sqrt{(sum((x-u)^2)/count(x))} \tag{3.3}$$

The K-means algorithm relies on a pre-knowledge of how many groups will be created. The selection of K should be computed and not guessed. This is often classified as an objection in the use of K-Means. The best way to circumvent this problem is by the use of the Elbow method. In the Elbow method the number of clusters varies and for each value of K we measure the explained variation. This will show at what value of K, the distance between the mean of a cluster and the rest of the points of the cluster is the lowest. In order to measure these, we use inertia. Inertia is the sum of squared distances of points to their closest cluster centre. The optimal K is found where the elbow is created. In Figure 3.3, we can see an example where $K = 3$

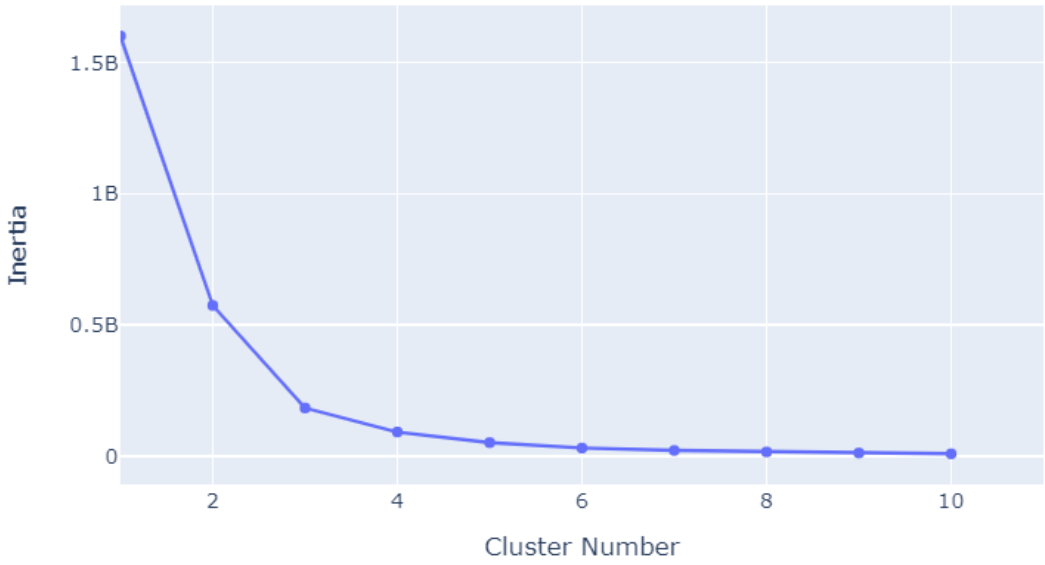


Figure 3.3: Example of elbow method for K=3

3.3.2 Analyse Payload

To analyse the payload we need to access the original payload, and then find a way to analyse it. We do not want to add any more points of weakness to the system in order to get access to the payload. Therefore our goal is to work with what a normal system has, and eventually extending some features but not implementing new ones. To accomplish this we resort to the web server logs.

3.3.2.1 Web Application (WebApp)

To get access to the payload without any additional resource from the ones used in a normal setup, we must capture it when the traffic arrives at the web server that contains the web application.

When the web server receives the traffic, it decrypts it in order to the web application process the requests, and logs the interactions derived from the requests.

We must ensure that all the traffic is logged by the server since not all web servers do this by default, where for that in some cases we must use or build some add-ons. For example, the Apache server default logs incorporates all content from GET requests, but does not log the POST requests. This is a problem because we want to analyse the content of all types of payloads and the POST requests, like the GET requests, are the ones that contain user input data and so the most interesting to access.

WebLogger

The web server creates two major log files. The first is a file containing all information about requests coming into the web server. The second is an error log file containing information about errors occurring in the web server while processing the requests and is also where any used add-on will output its data. We must have access to all of the payloads but that is not straightforward. By default the web server does not log the payload of a POST request. To surpass this, the log extractor resorts to an additional module that extends the logs to register a full POST request, including its payload. This add-on will output its data to the error log file.

3.3.2.2 Log Extractor

Before we start identifying attacks we must get the subset of logs corresponding to the WebFlow we identified as anomalous and prepare those logs for being analysed. The WebFlow will pinpoint some subset of traffic. This traffic must be collected from the servers logs. To correctly match the traffic with the logs we must have some identical features. After collecting the interested subset we must prepare the logs to be analysed. The filter and the parser are the identities that are responsible for this.

Filter

First, all the logs go through a log filter. In the Filter, we select the subset of logs that correspond to the traffic identified in the WebFlow. In this case the matching keys are datetime, source and destiny. We have to guarantee that both WebFlow and logs have these fields to make a correct match. However, the datetime is not the same because the traffic of the WebFlow is captured a moment before it arrives at the server, but it is close enough. The filter takes this deviation into account and manages it through a threshold to assure it gets all the logs related to the previously identified traffic. In practice the use of the datetime field is based on the minutes precision, which is precise enough for this work. The filter also discards the logs that do not have a payload included. This means logs from establishing the connection or from the SSL handshake are discarded, leaving just the logs with a GET or a POST request to go through.

Parser

The logs data are produced in chunks. To correctly work with the payload, we needed to rebuild the logs. For that we need to build a log parser as the only tools existing in the literature [56] [57] for the best of our knowledge, to parse the logs are old and do not have any updates in several years.

The log parser parses the chunked logs, identifies the GET and POST requests, joins all the parts of each GET or POST request and rebuilds the payloads. The final result is a steady stream of logs, with all the requests and its payloads. This allowed a lean down set of payloads, without all the extra information that is not needed in the next steps.

3.3.2.3 Similarity Search Engine

At this point, we have a dataset consisting of payloads. It is a subset of all the traffic, but nonetheless it can be a huge dataset. The extensive amount of payloads and the fact that it is a semi complex object, as it has keys and values of different kinds and sizes, means that it is not practical to do a standard transversal search for attacks. The traditional use of pattern matching algorithms used by a DPI is not feasible. Instead, we propose a new approach of similarity based package inspection.

In this section, we opted by a solution that is capable of indexing high-dimensional data for answering similarity-search queries searching a large dataset. From the realm of similarity search algorithms, the Locality-Sensitive Hashing (LSH) is the one we propose to use. With this algorithm, like the solution used in the WebFlow, we cluster the logs in buckets of similar data. In this probabilistic data structure it is possible to search a huge dataset in a reasonable time. The LSH algorithm utilised was min-wise independent permutations or *MinHash*. This LSH algorithm is based on the Jaccard similarity measure to determine which instances of the dataset better suit with the data we want to search.

In order to search for similar items we need to construct a database of items in which to search. The adopted solution must take to account two constraints:

- Query speed.
  We need to query the database often and the time it takes to get a result, must not be a bottleneck.

- Speed to update the database.
  Each index of the database has parameters that are dependent on the number of tokens in the database. As we will see further on, we will need to update the database with new payloads.

In order to create an LSH index that complies with those two constraints we tested the use of the algorithms MinHash LSH, MinHash LSH Ensemble and MinHash LSH Forest. The MinHash LSH Forest had one major drawback for this study: the results are fixed sized. This means that we must indicate the number of items we want as a result, meaning that to comply with the number of items for the result, the algorithm return items with very little similarity. That is useful in some

use cases, but not in ours, as we need to study the number of results based on different thresholds and permutations, to find the correct adjustment.

The LSH index is also affected by the need of tuning and re-tuning. In the case of the MinHash LSH Ensemble, each time we need to update the database, we need to rebuild it from the scratch to update the index. Compared to the other algorithms this is a major drawback.

The MinHash LSH has a different approach, and it is simpler to update its index. This update does not force a total rebuild of the index, being a much faster solution than the MinHash LSH Ensemble. For this reason our final selection was the MinHash LSH.

To tune the parameters of the algorithm we tested MinHash LSH with several values for its parameters. We were able to test different hash functions, different thresholds and different permutations. For the hash functions we did not find significant differences between the tested ones, both in performance as well in accuracy. For the thresholds, we tried a significant range, with the goal of finding the best value. In conjunction with the previously, we needed to test between 128 and 256 permutations. The higher permutation value was expected to produce better accuracy but with slower speed and higher memory usage. In this section every tool must work with the same threshold and the same permutation value, in order to work consistently.

Token builder

The objective of the Token builder is to standardised the payloads in tokens. Although in the search engine the objects to search are payloads, the LSH hash is being represented as a sequence of integers and because of that, the input data must be standardise in tokens. By creating tokens, payloads that have the same short pieces in them will have many common elements. As seen in Section 2.4.1.2 we may associate each payload with a set of tokens that appear one or more time inside that payload by defining a k-shingle as any substring of k length presented in the payload. In order to find the best k-shingle, we need to define which part of the payload we must search. We started a comparison between a token with key and value and a token with just the value. We observed that the keys produced too much noise and being a repetitive part in the token does not offer any accuracy in the results, so we decided to remove the keys. We also observed some other repetitive substrings, such as the *Submit* value from forms, that represented non dangerous values and we decided to remove them as well. After this first step in the preparation of the tokens, we realised that scanners tend to work with variations of strings.

```
Token 1: [4, ..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F, ..%2
   ↪ F..%2F..%2F..%2F..%2F..%2F..%2FWindows%2Fsystem., in]
Token 2: [4, ..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F, ..%2
   ↪ F..%2F..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd]
Token 3: [4, ..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F..%2F, ..%2
   ↪ F..%2F..%2F..%2F..%2F..%2F..%2F]
```

Listing 3.2: Example of tokens

In the three examples of tokens presented in Listing 3.2, the difference between them is minimal. We needed to find a way of maximising the differences and producing more accuracy to

detect differences.

To find related payloads, the most effective technique is to define the payloads as sets, which will produce short strings from the payloads. Each of these short strings is called a shingle. By segregating the payloads in shingles, payloads with short sections in common, will have more similarity. On the other hand, longer payloads with repetitive characters which are used to camouflage parts of the payloads, will have less similarity.

To implement the k-shingle, we used *n-grams*, an approach most known in the field of computational linguistics. This approach transforms our token into a continuous sequence of unique n items. The value of n should be chosen so that the likelihood of any given shingle occurring in any given payload is minimal.

For example, a n-gram with n=3, for the string `"..%2F..%2F"` is

`[..%, .%2, %2F, 2F., F.., ..%, .%2, %2F]`

These still have some repetitive shingles (e.g., `..%`), so the token builder removes any duplicates, and the final result is:

`[..\%, .\%2, \%2F, 2F., F..]`

With this approach the three tokens from Listing 3.2 will be transformed into the tokens presented in Listing 3.3

```
Token 1: [4, ..%, .%2, %2F, 2F., F.., 2FW, FWi, Win, ind, ndd,
    ↪ ddo, dow, ows, ws%, s%2, %Fs, Fsy, sys, yst, ste, tem, em.,
    ↪  m.i, .in]
Token 2: [4, ..%, .%2, %2F, 2F., F.., 2Fe, Fet, etc, tc%, c%2, 2
    ↪ Fp, Fpa, pas, ass, ssw, swd]
Token 3: [4, ..%, .%2, %2F, 2F., F..]
```

Listing 3.3: Example of transformed tokens of Listing 3.2

As shown in Listing 3.3 we passed from three very similar tokens to three very distinct tokens. We were able to minimise redundant parts of the payloads and produce meaningful tokens.

Evaluating the pairs of tokens within each Listing, it was visible that their Jaccard Similarity which represent their similarity distance is lower, meaning that this is a good approach for maximising the differences in order to gain more accuracy. Table 3.2 shows this evaluation conducted in Listings 3.2 and 3.3.

Table 3.2: Jaccard similarity of test case

	Before			After		
Tokens	Equal Tokens	Total Tokens	Jaccard Similarity	Equal Tokens	Total Tokens	Jaccard Similarity
<1,2>	2	5	0,4	6	42	0,14
<1,3>	2	5	0,4	6	31	0,19
<2,3>	2	4	0,5	6	23	0,26

The final objective of the Token Builder is to get the payload logs as input and produce a set of tokens. First the Token Builder removes everything but the meaningfully values, from each

payload. Then, each set of values is transformed into a token without repetitive values, and each value is transformed into n-gram values and encoded into the LSH object. A complete example of this processing, and so resulting in an LSH object, is depicted in the payload of Listing 3.4 that is transformed in the token in Listing 3.5

```
login=ZAP&email=ZAP&password=ZAP&password_conf=ZAP&secret=ZAP+OR
   ↪ +1\%3D1+--+&mail_activation=&action=create
```

Listing 3.4: An example of a complete payload

```
[ZAP,AP+,P+O,+OR,OR+,R+1,+1\,1\%,\%3,%3D,3D1,D1+,1+_,+--,--+,cre
   ↪ ,rea,eat,ate]
```

Listing 3.5: The resulting token of the payload from Listing 3.4

Tokens Database

This section presents a token database, constructed beforehand. We produced a controlled dataset of traffic generated with scanners, retrieved the logs, passed them through the log extractor, created tokens with the token builder, and updated the tokens in the database.

The database is an LSH object, and as such it has a threshold value and a permutation value. These values are determined in the Token Builder.

With the MinHash LSH, the LSH DB Updater can simply append a new token into the DB, which makes having more than one DB with different thresholds and permutation a possibility that we used.

Token Query

The token query searches for buckets with a similarity above a certain threshold. This component works with a large dataset, and we need to make many queries. To do this in a capable manner we can work with the hash function, the threshold and the permutations value. We tried different hash functions as SHA1 and MurmurHash3, but without visible impact on the final result, so we opted to work with the default hash function of SHA1. Bigger permutation values give higher accuracy, but it reduces the speed and increases memory usage. These occurs because a higher permutation value means more CPU instructions for every token and more tokens to be stored. From the initial tests it was not obvious what the correct values of threshold and permutation to choose, so the best option was to create several DB for the different values of threshold and permutations, and perform queries to each database.

Token Classifier

The token classifier is a simple decision tool, easy to adjust by the Security Operation Center (SOC) analyst, according to the feedback of previous decisions. Each query done in the Token query will have between zero and several results, and the quantity of the results will let us choose

what to do. Basically we will have to decide between three options. The token is benign, it needs further analysis, or it is an attack.

3.3.3 Continuous Improvement

The continuous improvement is an important stage of the overall process, as it will allow to update the Tokens database with new entries. To achieve this improvement system, there is a need to use additional tools and even the help of the SOC.

3.3.3.1 LSH DB Updater

The token query will return tokens based on its similarity. If the query returns an exact match then the token is already present in the database and we do not need to update it. There are however two situations where we can choose to update the DB:

- If the token is similar but not the same.
  In this event, the query gives as a result similar tokens, but not an equal token. We can update the database with this new token. This approach will increase the database overtime. We can tweak this approach and we only update the database if the tokens in the database are different but with a similarity between certain thresholds. For example a token with similarity 0.99 is not updated to the DB, but a token with 0.8 is. In our case we opted to update the database with all the tokens that are different, independent of its similarity, as we did not face any problems with the database size.

- If the token classifier decides that the token is an attack.
  In this event, we insert the new token into the LSH Index. Later through the analysis, every time that it is decided that a new token is an attack and the LSH index does not yet have that token, we use the LSH DB Updater to append the new token to the LSH DB and improve the system.

Alarm Generator

Every time a new payload is added to the database it means that it is a new payload observed by the system, and as so there is an alarm generator implemented with the purpose of alerting the SOC of the new payload.

3.3.3.2 String Analyser

To this point, from every token analysed we have come to three different conclusions:

- It is an attack
- It is probably not an attack
- It is not an attack

As we do not want uncertainty, the second conclusion is not good, and so in that case we need to further investigate the token. In this area, the quantity of payloads is significantly lower, and we can analyse the values of the payload in a more traditional way, with different string analysers. The design of the solution is based on a sequential use of tools that filters the values. The number of tools can vary and can be changed accordingly to the needs of the user of the tool. This modular approach allows for the scalability of the system.

Sanitiser

The Sanitiser is a module that receives a payload, and uses a sanitisation function to compare the input of that function with its output. First we must address the fact that an Uniform Resource Identifier (URI) must be encoded identically for global compatibility. A two-step technique is used to map the huge range of characters used throughout the world into the limited allowable characters in a URI. This is achieved with the use of UTF-8 encoding to convert the character string into a sequence of bytes and then each byte that is not a letter or digit is converted into *%HH*, being *HH* the hexadecimal value of the byte. In order to use a sanitisation function, the Sanitiser starts with a URL encode parsing, to prepare the string.

The sanitisation function removes from the string any characters that could represent a threat. Like the entity where it resides, the Sanitiser can be one or more functions, according to the intentions of the system owner. For example if the system owner wants to catch blank or empty strings and consider those as an attack, there could be a custom Sanitiser. The overall behaviour of the Sanitiser is observed in Figure 3.4

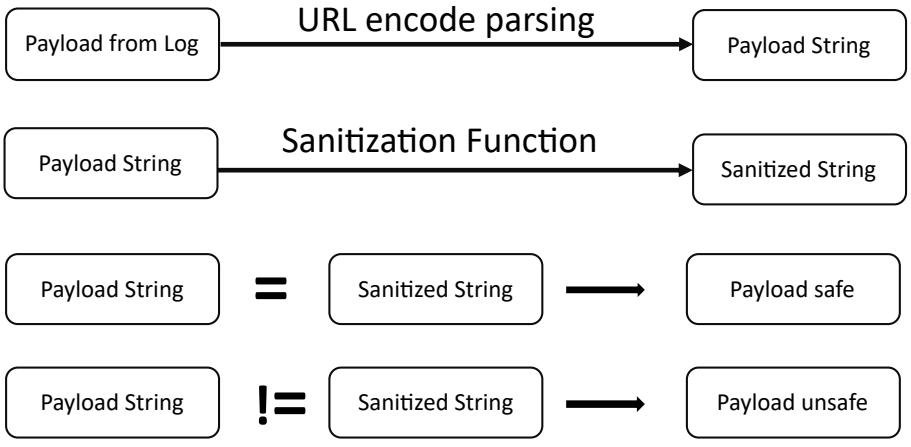


Figure 3.4: Sanitisation

String Classifier

The classifier will decide based on the result of each tool, if the string is an attack or not. It can make a decision for each tool separately. It is not necessary to use every tool. As soon as the

classifier decides that the string is an attack or not, there is no need to use additional tools. If the classifier cannot decide with accuracy, then we use another tool until we have no more tools, and in that case the string must be analysed by the next area. The Classifier will receive the payloads with the results from each tool used. It is the Classifier that defines what to do next.

If the result is that a payload is an attack, then that payload must be added to the Similarity Search Engine DB, so that next time it is earlier identifiable by the Similarity Search Engine. Otherwise the payload can go to the next tool of the String analyser.

Going through the tools, there can exist three possible outcomes:

- A tool considers that payload as an attack and the String Classifier sends it to the LSH DB Updater, to update the Similarity Search Engine.
- All the tools consider the payload safe.
- Some or at least one of the tools does not provide a result with a high certainty. In that case, the classifier must send the payload to the SOC analyst.

3.3.3.3 SOC

In the SOC, the security analyst receives all the payloads that the previous modules were not able to correctly identify. The security analyst has two choices, it can consider the payload benign or it can consider the payload an attack. There is a concern related to the number of payloads that the security analyst must analyse. To reduce this quantity, it is necessary that all previous modules correctly identify all the payloads. It is fundamental that the system maximises its accuracy. If the security analyst marks the payload as an attack, the payload must be included on the Tokens DB, in the similarity search engine. This is necessary, so that the payload is identified earlier next time, in the similarity search engine, preventing from going to the String Analyser and the SOC.

It is also the security analyst job to select the parameters of the overall solution, which include for example, the LSH parameters such as threshold, permutations, the number of databases to keep (he may opt to keep only one database of a specific threshold and permutation, or he may keep several databases with different parameters).

Chapter 4

Implementation

This chapter describes the implementation of the system proposed on this thesis. The overall system was implemented with the use of a controlled network that is discussed in Section 4.1. It allowed the study of the network traffic from beginning to end. Starting with a client, we observe the Network Traffic in Section 4.2, which includes the explanation where the logs are generated. Section 4.3 explains how the logs are extracted and prepared. Section 4.4 describes the Similarity Search Engine and its components. Finally, in Section 4.5, we address the continuous improvement approach in the String analyser entity. For the implementation of the system, we used several different tools, some of which needed adjustments to be adapted to our goals and others were specifically developed in this work.

4.1 Development Scenario

To test and build the tools needed for our implementation, there was a need to create a controlled network to test different scenarios and collect data that allowed us to make implementation decisions. This controlled network makes possible the use of vulnerable applications without endangering live systems, the use of scanners without creating bottlenecks in real networks and the creation of controlled network traffic with known traffic, giving accurate results in our testing.

The final network is observed in Figure 4.1. Client A was used for running specialised bots, scanners and fuzzers. Clients B and C were used for running different clients for manual navigation and testing. The webserver ran an Apache HTTP Server to offer a web application and storage. All machines were Virtual Machines, implemented with Oracle VM VirtualBox version 6.1 [58]. Clients A and B ran a Debian based distro, Client C ran a Windows OS (all the OSes were 64 bit versions). All machines were implemented with the specifications presented in Table 4.1.

Table 4.1: Basic specification of the virtual machines

Characteristic	Value
Base Memory	4 GB
Processors	2
Acceleration	VT - x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization
Network	VirtualBox Host-Only Ethernet Adapter

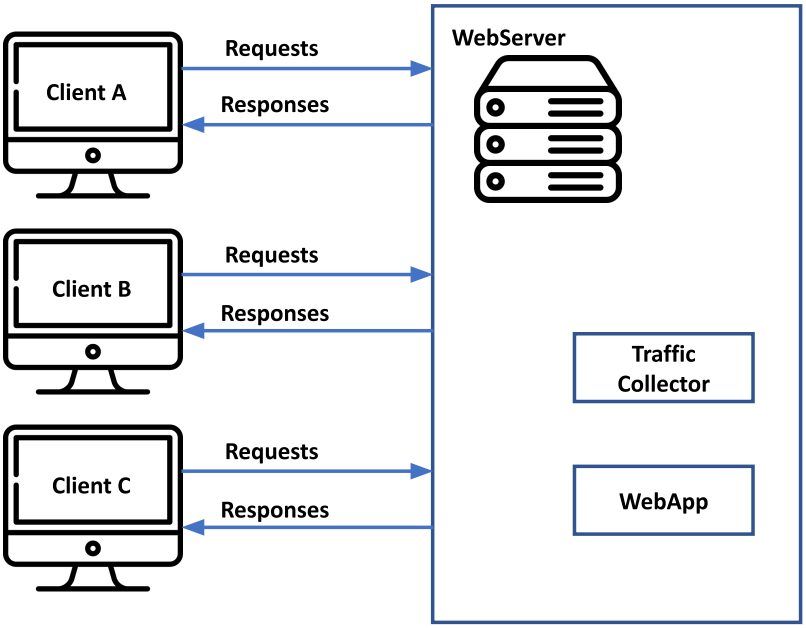


Figure 4.1: Controlled network build for testing

The network was a Host-only adapter of VirtualBox. This kind of network was needed as we were testing vulnerable applications, which are not supposed to be installed in Internet facing servers. With this network all the virtual machines can communicate to each other and with the host as if they were connected through a physical Ethernet switch. This network is not connected to the world outside the host since they are not connected to a physical networking interface. The host was a personal computer, running a windows 64-bit OS, with the specifications of Table 4.2.

Table 4.2: Specification of the host machine

Characteristic	Value
Processor	Intel(R)w Core (™) i7-6600U CPU @ 2.60GHz
Installed memory (RAM)	20 GB
Network	VirtualBox Host-Only Ethernet Adapter

The web application used for the testing was *DVWA - Damn Vulnerable Web Application* [59] which is a PHP/MySQL web application with known severe vulnerabilities. It was created with the purpose of facilitating the construction of a legal and safe environment for testing.

4.2 Network Traffic

The final implementation uses the WebFlow to construct a dataset. Flows originated on the App Server are discarded as these are the responses, which was identified by the protocol, Destination IP and Port features of the WebFlow. The available features in the WebFlow can be consulted in Table 4.3, and for the machine learning analysis, the selected features were Source IP, Source port, Number of packets and Duration. Source IP was selected because all the traffic from the scanner will have the same origin and will help identify the source. Source port, Number of packets and

duration are related to each request. Source port is changed between requests and as such is easy to count requests.

4.2.1 Exporter

For capturing the Network Traffic in a PCAP format, we use *tcpdump* [60]. The *tcpdump* is a powerful command-line tool that is capable of capturing network traffic. This tool also allows the analysis of network traffic, but in our implementation we only needed it for capturing the traffic. We configured this tool to produce a PCAP file with UTC Timestamp, Source ID, Destination ID, Protocol, Packet Length and the rest of the packet, including payload. In case of HTTPS traffic, the packets of the TLS protocol are captured but their payload is encrypted.

4.2.2 Collector

After a few experiments in capturing, exporting and analysing the packets, we chose the *SiLK* [61] framework, as it allows us to personalise the fields to work with, and it provides a full tool suite that supports collection, storage and analysis. The major features we used from the SiLK tool set [62] were:

- For filtering, displaying and sorting

  rwfilter - Select SiLK Flow records form the data repository and partition the records into one or more "pass" and/or "fail" output streams.

  rwcut - Print the attributes of SiLK Flow records in a delimited, columnar, human-readable format.

  rwsort - Sort SiLK Flow records using a user-specified key comprised of record attributes, and write the records to the named output path or to the standard output.

- For Counting and Grouping

  rwcount - Summarise (i.e., group or bin) SiLK Flow records across time, producing textual output with counts of bytes, packets, and flow records for each time bin.

  rwuniq - Summarise SiLK Flow records by a user-specified key comprised of record attributes and print columns for the total byte, packet, and/or flow counts for each bin. It can also count the number of distinct values for a field.

  rwstats - Summarise SiLK Flow records just like *rwuniq*, but sort the results by a value field to generate the Top-N or Bottom-N list and print the results.

- For packet and IPFIX processing

  rwp2yaf2silk - Convert a packet capture (PCAP) file - such as a file produced by *tcpdump* - to a single file of SiLK Flow records.

  rwpcut - Read a packet capture file and print its contents in a textual form similar to that produced by *rwcut*.

This set of tools allowed us to experiment and test different options. First the collector transforms the PCAP file into a WebFlow with *rwp2yaf2silk*. This step produced a file (rw extension) with a full WebFlow. Also, in the collector, the WebFlow is transformed into a CSV readable file with the *rwcut* tool. With *rwcut* we transform a WebFlow file into a CSV file with all the fields needed to use in the next steps. The chosen fields are presented in Table 4.3

Table 4.3: Fields used for the WebFlow

Field name	Description
sIP	source IP address
dIP	destination IP address
sPort	source port for TCP and UDP, or equivalent
dPort	destination port for TCP and UDP, or equivalent
protocol	IP protocol
packets	packet count
bytes	byte count
flags	bit-wise OR of TCP flags over all packets
sTime	starting time of flow in millisecond resolution
duration	duration of flow in millisecond resolution
eTime	end time of flow in millisecond resolution

4.2.3 WebApp

The WebLoger was constructed on the base of an Apache server. The Apache server logs all the GET requests including their payloads, but it does not log the POST request payloads. To obtain those payloads, the WebLoger uses the *mod_dumpio* add-on. This Apache module logs all input received by Apache and all output sent by Apache. Logging is done right after SSL decrypting (for input) and right before SSL encrypting (for output). This means that we are able to access the original payload, without being encrypted and without creating additional components where data resides in plain text.

To make the module available, it must be loaded in the Apache configuration, and the Apache LogLevel must be configured to *trace7*.

The Apache configuration file is presented in Listing 4.1.

```
# Enabling Dumpio
LogLevel dumpio:trace7
DumpIOInput On
DumpIOOutput On
```

Listing 4.1: Apache configuration

`LogLevel dumpio` is used to set the log level for trace messages to the value 7, meaning that the logs produced by *mod_dumpio* are being registered in a full verbose way. `DumpIOInput` indicates for *mod_dumpio* to log all inbound traffic to the server and `DumpIOOutput` indicates to log all outbound traffic.

Since *mod_dumpio* logs to the error logs file, we must set up its error log format. Apache makes available a large set of options to configure the log format [63]. In our case, the chosen log format was with the following options:

```
ErrorLogFormat "[%{u}t] [Req_ID: %{c}L] [R: %L] [%l] [%F] [%E]
    ↪ [%m] [client %a] [Local %A] | %M"
```

`"["` , `"]"` and `"|"` were just placeholders to facilitate the parsing to be done afterwards and the other parameters are described in Table 4.4

Table 4.4: Format String in Apache ErrorLogFormat Directive

Format String	Description
`%{u}t`	The current time including micro-seconds
`%{c}L`	Log ID of the connection if used in connection scope, empty otherwise
`%L`	Log ID of the request
`%l`	Loglevel of the message
`%F`	Source file name and line number of the log
`%E`	APR/OS error status code and string
`%m`	Name of the module logging the message
`%a`	Client IP address and port of the request
`%A`	Local IP-address and port
`%M`	The actual log message

An example excerpt of *mod_dumpio* logs is presented in Listing 4.2.

```
[Sat Apr 17 08:39:18.979837 2021] [Req_ID: a163MYZd+CE][R: [
   ↪ trace7] [mod_dumpio.c(151)] [dumpio] [client
   ↪ 192.168.56.105:48094] [Local 192.168.56.104:443]|
   ↪ mod_dumpio: dumpio_in - 103
[Sat Apr 17 08:39:18.979846 2021] [Req_ID: a163MYZd+CE][R: [
   ↪ trace7] [mod_dumpio.c(164)] [dumpio] [client
   ↪ 192.168.56.105:48094] [Local 192.168.56.104:443]|
   ↪ mod_dumpio: dumpio_out
[Sat Apr 17 08:39:18.979851 2021] [Req_ID: a163MYZd+CE][R: [
   ↪ trace7] [mod_dumpio.c(63)] [dumpio] [client
   ↪ 192.168.56.105:48094] [Local 192.168.56.104:443]|
   ↪ mod_dumpio: dumpio_out (metadata-FLUSH): 0 bytes
[Sat Apr 17 08:39:18.979856 2021] [Req_ID: a163MYZd+CE][R: [
   ↪ trace7] [mod_dumpio.c(63)] [dumpio] [client
   ↪ 192.168.56.105:48094] [Local 192.168.56.104:443]|
   ↪ mod_dumpio: dumpio_out (metadata-EOC): 0 bytes
[Sat Apr 17 08:39:18.980113 2021] [Req_ID: 0zm3MYZe+CE][R: [info
   ↪ ] [ssl_engine_io.c(1381)] [ssl] [client
   ↪ 192.168.56.105:48092] [Local 192.168.56.104:443]| AH02008:
   ↪ SSL library error 1 in handshake (server
   ↪ 192.168.56.102:443)
[Sat Apr 17 08:39:18.980142 2021] [Req_ID: [info] [ssl_engine_io
   ↪ .c(1382)] [ssl] [client [Local SSL Library Error: error
   ↪ :14094416:SSL routines:ssl3_read_bytes:sslv3 alert
   ↪ certificate unknown (SSL alert number 46)
[Sat Apr 17 08:39:18.980151 2021] [Req_ID: 0zm3MYZe+CE][R: [info
   ↪ ] [ssl_engine_io.c(1106)] [ssl] [client
   ↪ 192.168.56.105:48092] [Local 192.168.56.104:443]| AH01998:
   ↪ Connection closed to child 1 with abortive shutdown (server
   ↪  192.168.56.102:443)
[Sat Apr 17 08:39:18.980183 2021] [Req_ID: 0zm3MYZe+CE][R: [
   ↪ trace7] [mod_dumpio.c(151)] [dumpio] [client
   ↪ 192.168.56.105:48092] [Local 192.168.56.104:443]|
   ↪ mod_dumpio: dumpio_in - 20014
```

Listing 4.2: Example of an ErrorLog produced by *mod_dumpio* module.

4.3 Log Extractor

The LogExtractor works with the error log file from the Apache server, which was produced by the WebLoger (as described in Section 4.2.3) and retrieves the logs associated with the result from the WebFlow Analyser. The implementation used Python V3.8 as the programming language.

4.3.1 Filter

The WebFlow Analyser identifies the portion of traffic where there is evidence of a scanner. The source ID and the datetime stamp of the Apache log allows the Filter to match the logs with the

subset of network traffic. This filter was also implemented in Python.

4.3.2 Parser

After identifying the part of the logs needed, there must be some more work in the *error.log* file and that work is done by the Parser. From the format string options presented in the log file, as mentioned in Table 4.4, the Parser uses the options `%F` and `%M`. Option `%F` allows to filter for the log file produced by the line code # 103 as this is the code that outputs the logs with the payloads. In Listing 4.3 one can see an example of this case in line 2.

```
1 [mod_dumpio.c(63)]
2 [mod_dumpio.c(103)]
3 [mod_dumpio.c(140)]
```

Listing 4.3: Examples of option `%F` in log file

The log message produced by format string option `%M` starts with a preposition that allows one to infer the traffic direction. We want the inbound traffic, in the example in Listing 4.4 are the lines 1 and 2.

```
1 mod_dumpio: dumpio_in (data-TRANSIENT): POST /bWAPP/sqli_12.
    ↪ php HTTP/1.1\r\n
2 mod_dumpio: dumpio_in (data-TRANSIENT): GET /bWAPP/iframei.
    ↪ php?ParamUrl=robots.txt&ParamWidth=250&ParamHeight=250
    ↪ HTTP/1.1\r\n
3 mod_dumpio: dumpio_out (data-HEAP): HTTP/1.1 200 OK\r\nDate:
    ↪ Sat, 17 Jul 2021 21:00:07 GMT\r\nServer: Apache/2.4.38 (
    ↪ Debian)\r\nLast-Modified: Sat, 17 Jul 2021 16:50:36 GMT\
    ↪ r\nETag: "a7-5c75480d56028"\r\nAccept-Ranges: bytes\r\
    ↪ nContent-Length: 167\r\nVary: Accept-Encoding\r\nKeep-
    ↪ Alive: timeout=5, max=100\r\nConnection: Keep-Alive\r\
    ↪ nContent-Type: text/plain\r\n\r\n
```

Listing 4.4: Examples of option `%M` in log file

One important aspect to consider was that the traffic is logged in chunked parts and not as a whole. This fact means that, each log line is part of a request and since the lines are intermingled, the POST and GET requests logs are scattered throughout the file. An example can be seen in Listing 4.5, where there is an example with only datetime, Source file name and line number of the log call and message.

```
1  [Sat Jan 02 18:03:30.394295 2021] [mod_dumpio.c(164)]
   ↪ mod_dumpio: dumpio_out
2  [Sat Jan 02 18:03:30.394586 2021] [mod_dumpio.c(63)]
   ↪ mod_dumpio: dumpio_out (metadata-FLUSH): 0 bytes
3  [Sat Jan 02 18:03:30.394703 2021] [mod_dumpio.c(140)]
   ↪ mod_dumpio: dumpio_in [getline-blocking] 0 readbytes
4  [Sat Jan 02 18:03:31.005101 2021] [mod_dumpio.c(63)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): 1 bytes
5  [Sat Jan 02 18:03:31.005159 2021] [mod_dumpio.c(103)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): G
6  [Sat Jan 02 18:03:31.005169 2021] [mod_dumpio.c(63)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): 55 bytes
7  [Sat Jan 02 18:03:31.005195 2021] [mod_dumpio.c(103)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): ET /vulnerabilities/
   ↪ sqli/?id=1&Submit=Submit HTTP/1.1\r\n
8  [Sat Jan 02 18:03:31.005218 2021] [mod_dumpio.c(140)]
   ↪ mod_dumpio: dumpio_in [getline-blocking] 0 readbytes
9  [Sat Jan 02 18:03:31.005226 2021] [mod_dumpio.c(63)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): 22 bytes
10 [Sat Jan 02 18:03:31.005232 2021] [mod_dumpio.c(103)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): Host: 192.168.56.104\
   ↪ r\n
11 [Sat Jan 02 18:03:31.005239 2021] [mod_dumpio.c(140)]
   ↪ mod_dumpio: dumpio_in [getline-blocking] 0 readbytes
12 [Sat Jan 02 18:03:31.005245 2021] [mod_dumpio.c(63)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): 24 bytes
13 [Sat Jan 02 18:03:31.005251 2021] [mod_dumpio.c(103)]
   ↪ mod_dumpio: dumpio_in (data-HEAP): Connection: keep-
   ↪ alive\r\n
```

Listing 4.5: log file parsed example

From this example, we start by filtering the logs made by code line #103, with a mode configuration of $dump\_io$. In Listing 4.5, that result is given by lines 5, 7, 10 and 13. After filtering these lines, the request is complete, as presented in Listing 4.6.

The implementation of the Parser was done in Python, by traversing the log file, keeping the lines that we were interested in and discarding the others. From the result in Listing 4.6 we get the final payload in the first line $id = 1\&Submit = Submit$. In case of a POST request, the payload is in the message at the last line. Those are the final values passed to the next module of our system, the Similarity Search Engine.

```
GET /vulnerabilities/sqli/?id=1&Submit=Submit HTTP/1.1
Host: 192.168.56.104
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
   ↪ /537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari
   ↪ /537.36
Accept: text/html,application/xhtml+xml,application/xml;q
   ↪ =0.9,image/webp,image/apng,*/*;q=0.8,application/signed-
   ↪ exchange;v=b3;q=0.9
Referer: http://192.168.56.104/vulnerabilities/sqli/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: PHPSESSID=vjdso0ulg77g2ennu1r18skq5k; security=low
```

Listing 4.6: Example of complete request

4.4 Similarity Search Engine

The similarity search engine works with tokens obtained from the payloads. This module needs to transform the payloads into tokens, construct a database of tokens and create a tool to query that database. We chose to continue with Python and used the datasketch [64] library to estimate the Jaccard similarity between payloads.

The main objective was building a LSH object with tokens built from the payloads of the anomalous traffic. After that, the similarity search engine can query the LSH object and find similar tokens (and therefore similar payloads). When a new payload is found, this module can also transform the new payload into a token and update the LSH database with it.

4.4.1 Token Builder

The first step is to retrieve all the payloads values. To do so, the token builder uses basic string manipulation as the payloads are always in the format: $key1 = value1\&key2 = value2\&key3 = value3$ and so on. As already stated in Section 3.3.2.1 the token builder must:

- Use only the values of the payload
- Transform the values into n-grams of n=3
- Remove duplicates

All of these requirements are easily accomplished in Python, using string manipulation, and Python data structures, as for example the *set*. In Python, a *set* is a data type used to store collections of data, which are unordered, unchangeable (items are not changeable), and do not allow duplicate values.

4.4.2 Token Database

The tokens database was previously constructed by creating a dataset. In order to create a database we used the network built in Section 4.1, and set up a Client A with a scanner. We used *OWASP Zed Attack Proxy (ZAP)* [65].

First, we had to create a *ZAP* script that allowed for the scanner to login into *DVWA* and perform a scan. *DVWA* has several security levels, and it was fundamental that the *ZAP* scanner did not change the security level from low. A low security level means that the application has several active vulnerabilities and the scanner will try to detect and exploit them. The *ZAP* script was a JavaScript/ECMAScript. It started by creating a context that had a username and password necessary to create the requests for a login, verifying if the authentication is working, and creating a seed for the spider. In order to get the maximum examples of payloads created by the scanner, all the scan tests were chosen. These included tests for the following classes of web vulnerabilities:

- Directory Browsing
- Source Code Disclosure - /WEB-INF folder
- Buffer Overflow
- CRLF Injection
- Cross Site Scripting (Persistent)
- Cross Site Scripting (Persistent) - Prime
- Cross Site Scripting (Persistent) - Spider
- Cross Site Scripting (Reflected)
- Format String Error
- Parameter Tampering
- Remote OS Command Injection
- Server Side Code Injection
- Server Side Include
- SQL Injection
- External Redirect
- Script Active Scan Rules
- Path Traversal
- Remote File Inclusion

After making the client script work with ZAP, we captured the logs at the Apache server. From there, we created the tokens, and with these tokens we built the LSH database, as illustrated in Figure 4.2.

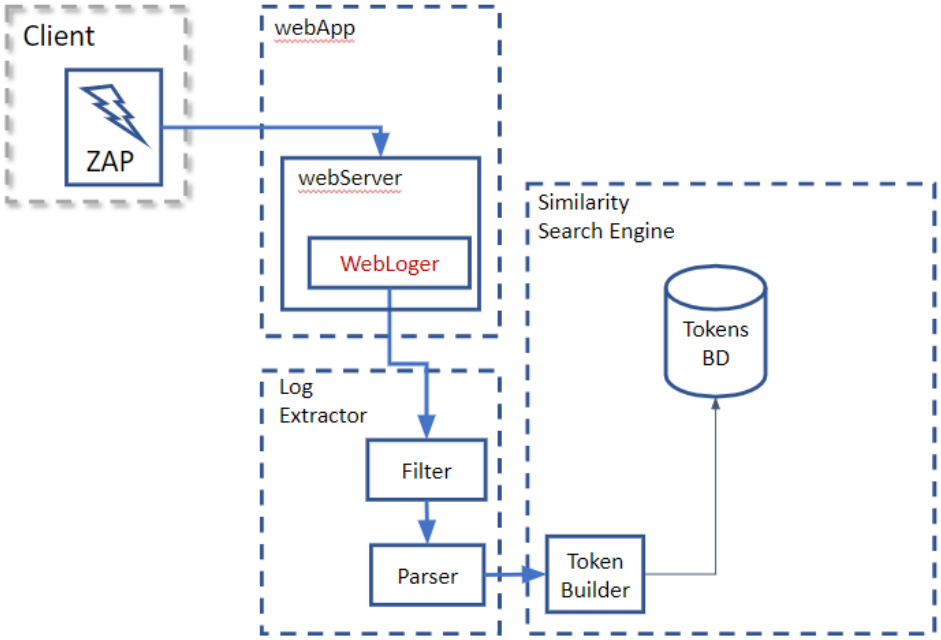


Figure 4.2: DB creation

There was no need to pinpoint the logs related to the traffic from the ZAP scanner, because it was a session without any other traffic. The final log file had 1,8 GB in size, corresponding to 373k lines, which included:

- 170k - Replies
- 164k - Get requests
- 19k - Post requests

As explained in Section 3.3.2.3 the algorithm used for LSH was the MinHash LSH. We needed several Databases with different parameters, a set with 128 permutations and a different set with 256 permutations. In each of these sets, we need a DB for each similarity threshold we chose. The option was to construct one Database for 0.60, 0.70, 0.80, 0.90, 0.95, 0.98 and 0.99. With this many Databases we can twitch the token classifier as we want. If the token query returns too many false results, we must use a higher threshold, if the token query is too slow then we can use a Database from the 128 permutations.

This many Databases comes with a price. Of course the used space is a point of interest, and can be a problem, but it is easily addressed if needed. The biggest problem is the time needed to construct all of these Databases. However the more prepared we get, the better answers we are able to present later on. As the implementation of this system was made in Python, the Tokens Database is an LSH object that is serialised to the disk drive using the pickle library [66].

4.4.2.1 Pickling

We wanted to find out the behaviour of the LSH object during pickling and unpickling. For that analysis, we used the log file mentioned above and created the same set of Database of tokens 100 times each. The time for pickling and unpickling the LSH object to and from a file was measured in microseconds within the testing environment described in Section 4.1. The measured values depend on several parameters such as the CPU speed, RAM and hardware disk among others, and it will vary on different environments. The behaviour however will be the same. We are interested in the evolution between thresholds for each permutation.

Figures 4.3 and 4.4 presents the results of these experiments, from which we can derive some conclusions:

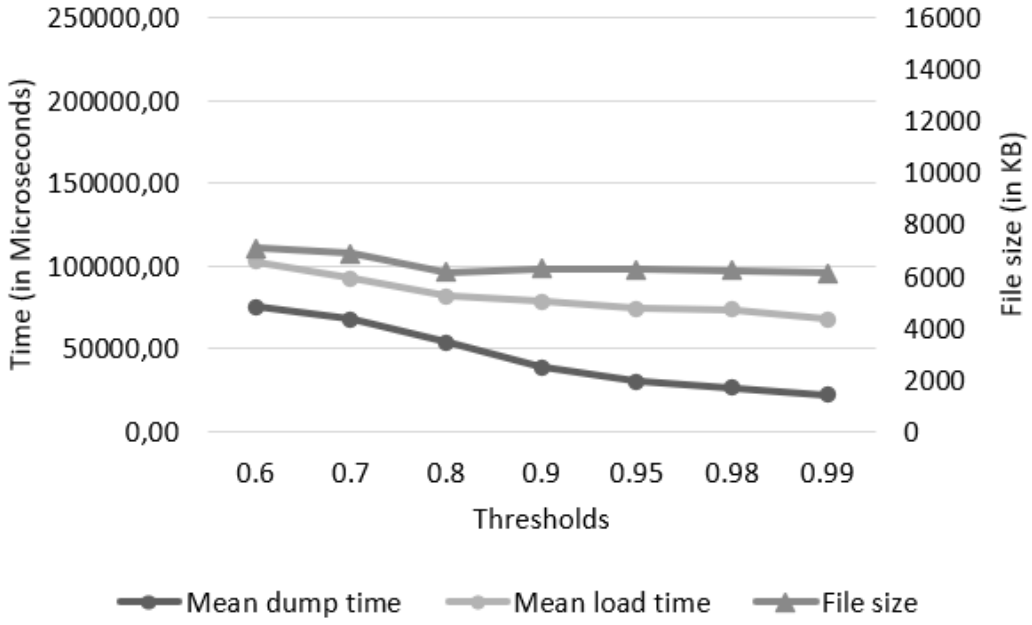


Figure 4.3: Mean dump and load time, and file size of the LSH object with 128 Permutations

- More permutations imply more calculations and, therefore, the time to create the LSH object and to serialise it to a file is bigger.
- More permutations implies a bigger LSH object and, therefore, the file representation of the object takes more space.
- The smaller the threshold, more pairs of similar tokens and less buckets will exist, which means more time to serialise the LSH object to a file and more time deserialise that file, because it is a bigger file.

4.4.3 Token Query

To perform a query, the query token must be hashed to an existing bucket in each multimap of the LSH object. Tokens which hash to the same bucket are returned as similar. For this test we did

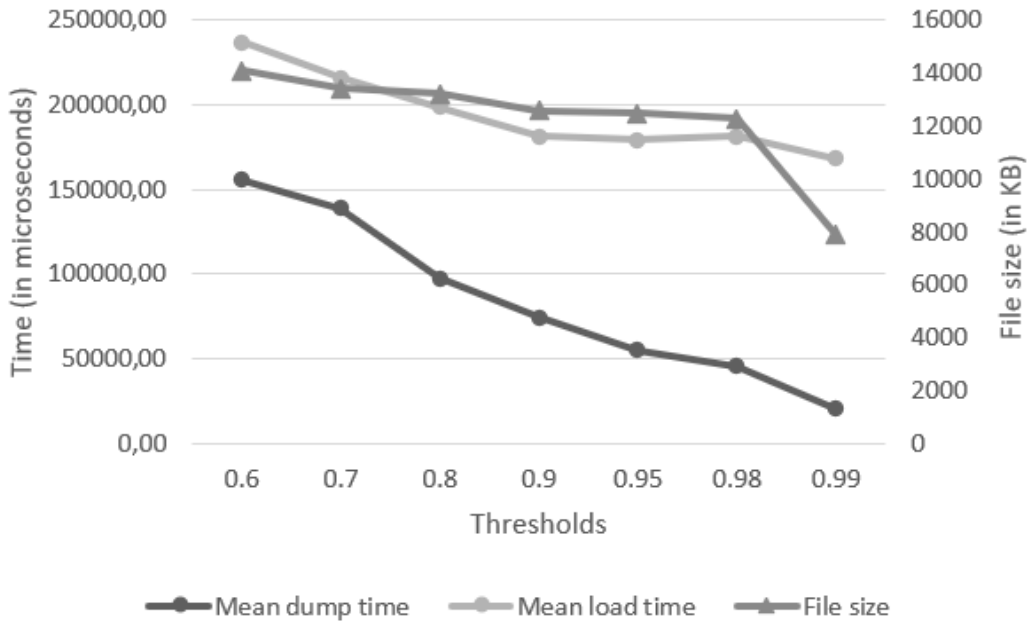


Figure 4.4: Mean dump and load time, and file size of the LSH object with 256 Permutations

1000 queries of 3 different tokens, in each combination of permutations with threshold. The tokens were one with many results, one with fewer results and one with no results, the permutations values were 128 and 256, and threshold values were 0.6, 0.7, 0.8, 0.9, 0.95, 0.98 and 0.99. In total, we executed 42 thousand queries.

The query time was measured and its average is presented in Figure 4.5. It was only possible to measure the time in microseconds, due to the fact that it was such a small timeframe, that a more detailed precision would cause an overflow in the time object and sometimes produced a negative time difference. These values must take into consideration that the variation is high, as most of the values are zero. However, it is safe to conclude that, the bigger the threshold, the less time it takes for the query to return a result.

4.4.4 Token Classifier

If a query is made to a high threshold LSH object database, and it returns one or more results, then as we have several similar payloads in the database, it is very likely that the payload we are analysing should also be a member of the DB. In this case, we update the DB and send a message to the SOC.

If a query is made to a low threshold LSH object, and returns few or any results, means that the token has no similarity with the other tokens in the DB. It implies that it is not a payload from a known scanner. We must be careful here, because we previously identified this payload, as part of a WebFlow from anomalous traffic. Because of this uncertainty, created by different indicators, we must further analyse the token.

In our tests we found that a threshold of 0.8 and considering a query with one or more results were sufficient to identify the use of a scanner, but the system is easily adjusted as needed. For

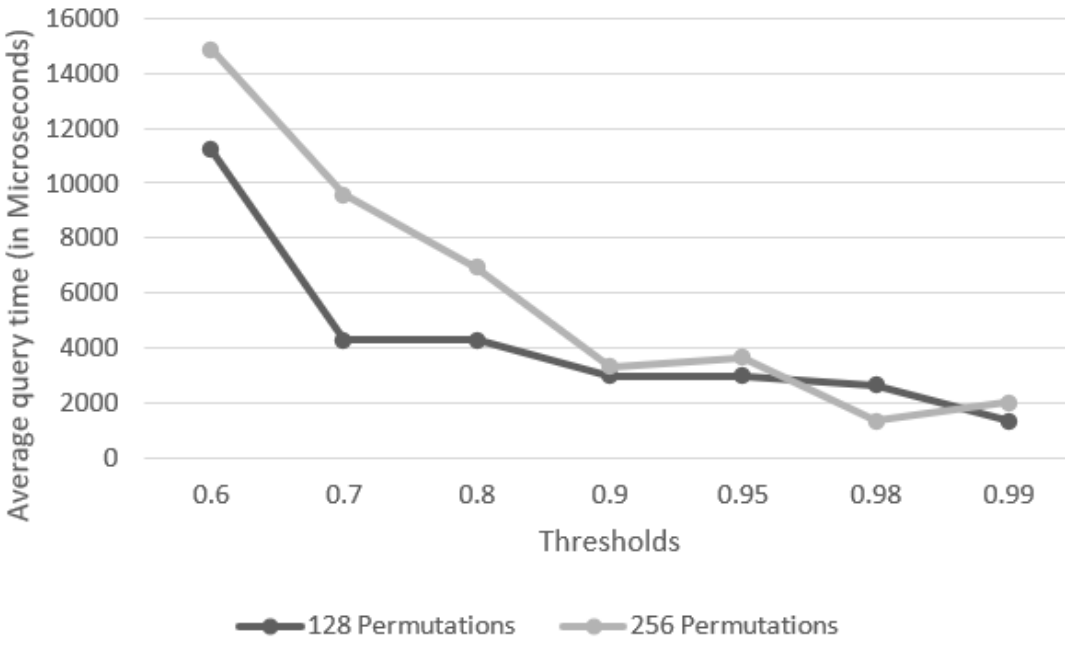


Figure 4.5: Average query time (in Microseconds)

example, we can use an LSH object with a 0.7 threshold and only choose to consider it an attack if the query returns 3 or more tokens or we can choose a 0.9 threshold and consider it an attack if the query results in 1 or more tokens.

Whichever the adjustments used, the actions of the classifier are always the same: either considering the token an attack, updating the DB and alerting the SOC or in face of uncertainty continuing to analyse the token.

4.4.5 LSH DB Updater

The LSH DB updater is used to include a new payload as a token in the DB. This process occurs when a payload is identified as an attack or as belonging to a scanner. This identification can be performed by the SOC or via the Classifier presented in the String Analyser module. The objective is to make the Similarity Search Engine more precise in future analyses. In terms of implementation, the LSH DB is an LSH object, and the way to update is simple by loading it in memory and using an insert function to simply append the new token. In Figure 4.6, one can observe a time analysis of the insert function for the different thresholds and permutations. It was only possible to measure the time in microseconds, because a more detailed precision would cause an overflow in the time object and sometimes produced a negative time difference. Because most of the values are zero, these analysis must account for the high variance, but it is reasonable to deduce that the higher the threshold, the less time it takes for the update function to execute.

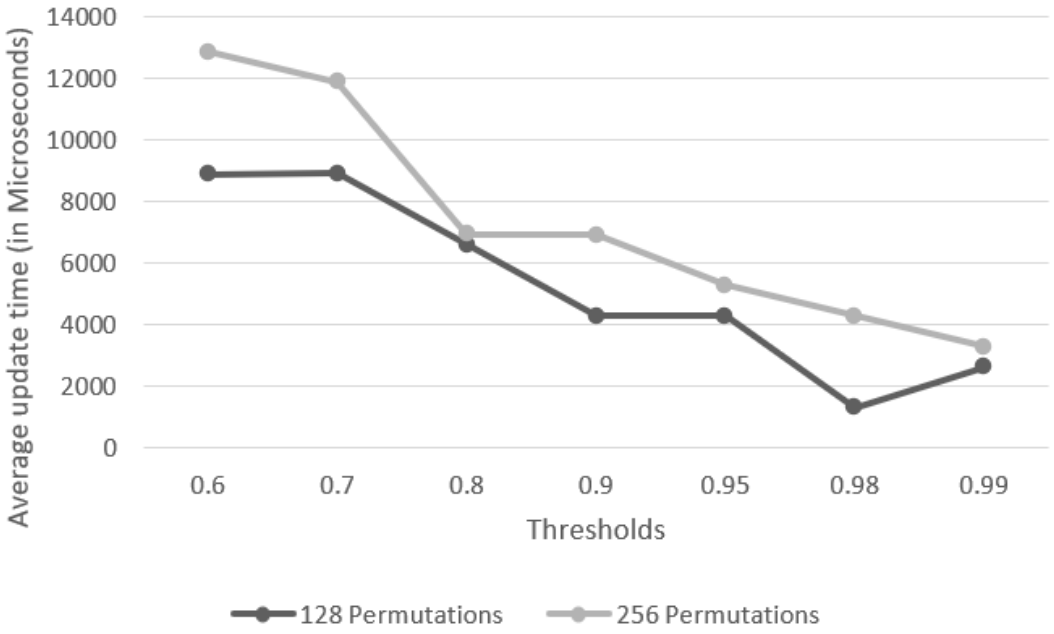


Figure 4.6: Average time to update DB

4.5 String Analyser

All the payloads that the Similarity Search Engine is not certain to be an attack, must go to the next entity, the String Analyser. In this module, we have the same objective, to analyse a payload and detect if it is an attack, but with a different approach. In this set of tools, we analyse each string, to classify it. We work with the payload and its original values as strings. The way this entity is constructed allows us to use more tools if we need it. When a payload arrives at this point, it has gone through a similarity search engine, as there is no similar payload in the DB. This indicates that the payload being examined is being analysed by our solution, for the first time. So our approach at this time is more conventional, and we focus on the payload content. The objective here is to identify a payload with an attack, so it can be added to the DB of the similarity search engine.

4.5.1 Sanitiser

In our implementation in Python we used *MySQLdb* [67], which is an interface to the popular MySQL database server that provides the Python database API. By accessing the MySQL C API, the Sanitiser can use the $mysql\_real\_escape\_string\_quote()$ function that *creates a legal SQL string for use in an SQL statement* [68].

4.5.2 Classifier

The classifier implemented a simple rule. If the Sanitiser discovers an unsafe payload then the classifier send the token to the database by the LSH DB Updater, if the Sanitiser result is that the payload is safe, the classifier discards that payload.

4.6 Final Considerations

The amount of data to analyse in each module is smaller in each step. We start with all the traffic captured in the network, we identify a subset of that traffic as suspicious with the use of WebFlow and ML, we then get the corresponding logs, and filter them to obtain the payloads. After the use of similarity search engine, we are left with a smaller subset of payloads that did not had a match. These residual payloads must be analysed in the string classifier, and there will be a marginal subset of payloads for the SOC analysis. Additionally, the String Analyser can be populated with more than one tool. Our case used a Sanitiser, but the overall system can have more string analyser tools.

Chapter 5

Evaluation

This chapter contains the experimental evaluation of the system proposed in this thesis. The experiments will be conducted on a controlled network to create normal traffic and traffic originated by a scanner. The chapter describes the experiment environment in Section 5.1 and in Section 5.2 we discuss the results. The evaluation aims to answer the following questions:

Q1 - Can the system detect anomalous traffic with high chances of including web attacks like SQLi or XSS?

Q2 - Can the system analyse the payload content of the anomalous traffic previously detected and confirm the payloads are from attackers?

Q3 - Can we improve the process?

5.1 Experimental Environment

This section describes the experimental environment we set up to conduct the experiments. The conceptual architecture follows the diagram presented in Figure 4.1 and uses the same development scenario described in Section 4.1. More specifically, we describe the concrete tools employed in each component as well as the remaining software stack when appropriate.

WebServer

We started by deploying a webserver running the web application DVWA [59], v1.10, for monitoring it. This machine also runs *tcpdump* [69] version 4.9.3 which depends on *libpcap* version 1.8.1 and *OpenSSL* version 1.1.1d 10 Sep 2019. The tcpdump tool was used to collect all the network traffic. Before starting the test we prepared the Apache server [70] version 2.4.38 (Debian), by installing the add-on *mod_dumpio* and created a configuration file to create the necessary logs, as described in Section 4.2.3. This machine runs a Debian GNU/Linux 10 (buster), with the configurations as described in Table 4.1.

Bot Traffic

To simulate normal human traffic, we created a Bot in Python [71] version 3.8. . To achieve

this, the bot will go to different pages of the application and perform different actions. Some of the actions are normal and some are attacks. As we wanted the traffic to remain similar to a human originated traffic, we created some randomness to the navigation. The time between each action was randomised and also the actions were randomised. In Figure 4.1 this was implemented in Client A, using a machine with the configurations as described in Table 4.1 and running Kali GNU/Linux Rolling, 2020.3 release.

Scanner

The scanner utilised was *OWASP ZAP* [65] Version 2.9.0, we used the same script as in Section 4.4.2. This script allowed for the scanner to login into *DVWA* and perform a scan. In the Figure 4.1 the scanner was implemented in Client B, using a machine with the configurations as described in Table 4.1 and running Kali GNU/Linux Rolling, 2020.3 release.

Traffic generation

To perform the evaluation of the traffic, we ran the bot described in Section 5.1 two times as this will create some random traffic. We opted to run the bot at two distinct times in order to create two different navigation events. Each of those times, the bot spent at least 2 hours navigating the app and performing actions. We also ran the Scanner, which took 17 minutes to perform the full scan. The traffic generated by the Scanner included traffic generated by a spider and traffic with attacks. A spider is a passive scanner tool that collects new URLs on the site being scanned. The spider starts with a list of known URLs, navigate through those URLs and retrieve all the hyperlinks in the page. It then adds the hyperlinks to the list of URLs, and keep doing so until it captures all the URLs of the website. The active scan uses the URLs captured by the spider, and attempt to discover vulnerabilities by using known attacks on those URLs. All of this traffic was captured using *tcpdump*.

5.1.1 Attack Detection

To analyse the network traffic we followed the proposed architecture and as so, we pass the traffic throughout the several entities described in Figure 3.2. The objective was to perform all three phases of the proposed approach:

1. Detecting anomalous traffic with high chances of including web attacks like SQLi or XSS.
2. Analyse the payload content of the previously selected traffic.
3. Continuous improvement process.

5.1.1.1 Detecting Anomalous Traffic

The traffic to analyse was all the traffic generated in Section 5.1, which included the (normal) traffic from the bot and the traffic from the scanner. The first step is to produce a WebFlow and then analyse the payload content.

WebFlow

The Exporter tool used was *tcpdump*, and this tool produced a PCAP file. In the Collector, we used the *SiLK* [61] Version 3.19.1, framework to transform the traffic from the PCAP file, into a WebFlow. The result observed in Table 5.1 had more than 16k lines.

Table 5.1: WebFlow before filter

sIP	dIP	sPort	dPort	Proto	Packets	Bytes	Flags	Duration
3232249957	3232249958	37152	443	6	6	867	27	0.045
3232249958	3232249957	443	37152	6	5	1818	27	0.045
3232249957	3232249958	37154	443	6	7	919	27	0.043
3232249958	3232249957	443	37154	6	4	1766	27	0.043
...	...	...	...	...	...	...	...	...

The Analyser uses this WebFlow to find the traffic of the scanner. It starts by filtering the WebFlow to keep just the traffic from the TCP protocol, easily identified by the value 6 in the *Proto* field of the WebFlow, it then filters just the inbound traffic, identified by the value of server IP in the *dIP* field. It is also safe to discard the *dPort* field, as it will always have the value 443, the port of HTTPS connections.

Table 5.2: WebFlow after filter

sIP	sPort	Packets	Bytes	Flags	Duration
3232249957	42516	18	4518	30	30.686
3232249957	37478	18	4584	30	27.89
3232249957	40634	18	4688	30	24.99
3232249957	38214	16	3932	30	20.277
...	...	...	...	...	...

This filter result in a subset as observed in Table 5.2, with 7 728 lines. We then perform the analyses by using the *K-means* algorithm with the MixMax scaler. The features analysed were sIP, sPort, packets and duration. By using the Elbow curve approach to detect the amount of clusters we discover the optimal k to be 3, where the inertia value is 86 and after which point the changes in inertia value are insignificant. This elbow curve can be observed in Figure 5.1.

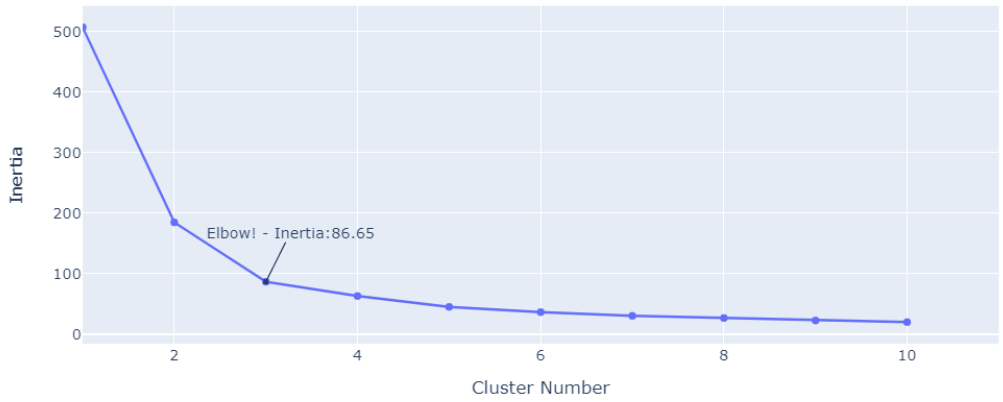


Figure 5.1: Elbow calculation

The K-means algorithm identifies three clusters with different sizes, described in Table 5.3. During the cluster formation, it is expected to group in the smaller cluster, the traffic from the

Table 5.3: Clusters sizes

# Cluster	size
0	3746
1	332
2	3650

scanner, as this traffic is in smaller quantity than the normal traffic. This is a typical scenario in real life, were there is more normal traffic than anomalous. Analysing cluster # 1 we can compare it to the original traffic and got a match for the traffic of the scanner.

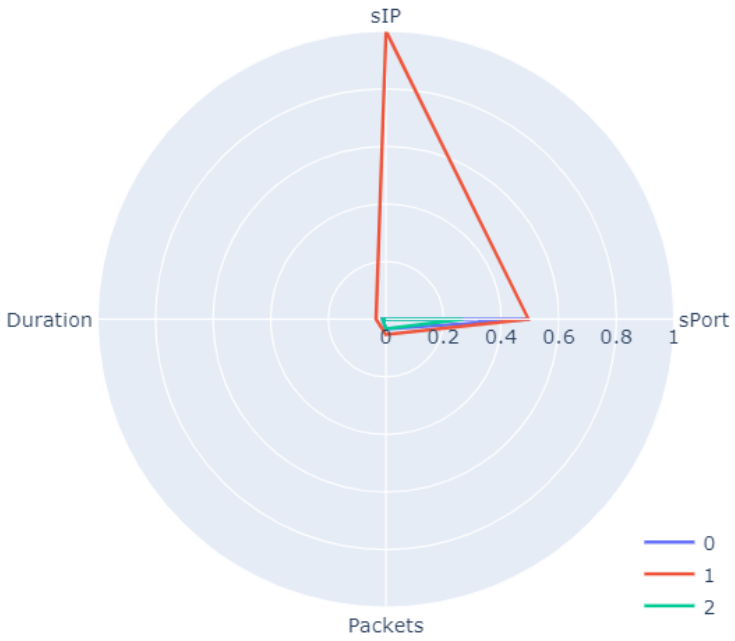


Figure 5.2: Clusters features analyse

We can observe in Figure 5.2 that the major difference is the sIP. This difference is expected in a real world scenario, but there are others differences. A closer look, in Figure 5.3, will show that there is a clear difference in cluster # 1 from the other two clusters on the duration and packets features. This cluster # 1 includes only traffic from the scanner and no traffic from the bot. Clusters # 0 and # 2 include traffic from the bot, but no traffic from the scanner.

5.1.1.2 Analyse the Payload Content

As described in Section 5.1, the webserver was previously prepared, and there was no need to do anything else at this stage, as the log error file was already created.

Log Extractor

In the Log Extractor we ran the Filter and the Parser, as stated in the Section 4.3. The Filter uses

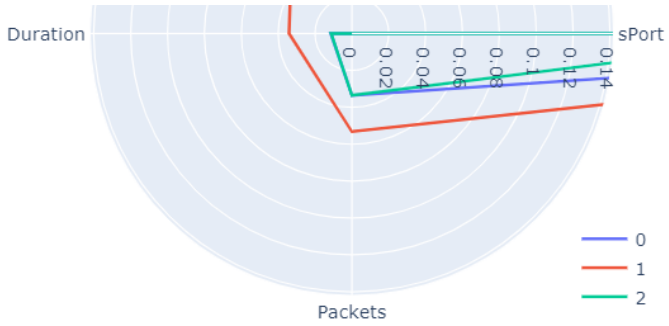


Figure 5.3: Clusters features analyse detail

the traffic identified in cluster #1, and created a subset of those logs, from the error log file, and this subset was transformed in the Parser.

Similarity Search Engine

The similarity search engine (SSE) is a major point in the proposed solution of this work. To properly evaluate it, we opted to do a Cross-validation. Cross validation is a resampling strategy that tests a model using various chunks of the data on successive rounds. It is most commonly employed in situations when the aim is the prediction and the user wants to know how well a predictive model will perform in practice. Our evaluation is based on a ratio of 80/20 and 20 tests. This is 80% of tokens are used to create the LSH DB, 20% of tokens are used to query the DB and this is done 20 times, being that each time the tokens are randomly selected. We also wanted to evaluate the permutation and the threshold values, and to accomplish that, we build a separated DB for each pair of permutation-threshold in each of the 20 tests.

Token Builder

The 332 lines of WebFlow traffic presented in cluster #1 belonged to a subset of logs, that produced 5951 tokens.

Tokens BD

Based on the cross validation approach, the DB was constructed each time with 4760 randomly selected tokens.

Token query

The remaining 1191 tokens were queried in the DB. What we want is to find if these searched tokens match to tokens that were previously identified as belonging to a scanner.

After finalising all the tests, we evaluate the percentage of tokens with one or more results in the DB. The results of this evaluation are in Table 5.4, which represents the percentage of tokens with one or more results from the query. As we can observe with a permutation of 128 and a threshold of 0.6, 94,55% of the 1191 searched tokens had one or more positive results. Since we know that all the tokens belongs to traffic originated by a scanner, we can say that in this example

we have 94,55% positive rate of identified tokens.

Table 5.4: Token query evaluation

Threshold	128	256
0.6	94,55%	93,97%
0.7	93,14%	93,12%
0.8	89,36%	89,67%
0.9	67,61%	72,06%
0.95	37,72%	38,62%
0.98	14,41%	12,27%
0.99	2,61%	2,27%

5.1.1.3 Continuous Improvement Process

The continuous improvement process is intended to be continuous, and as such this preliminary evaluation will just indicate a tendency, being necessary more prolonged tests in real life situations to correctly determine its usefulness. The major objective with this process is to increase the robustness of the tools used in Section 5.1.1.2. If the detection done in Section 5.1.1.2 was successful, then all the payloads from that traffic should be included in the DB. The DB already includes some of those payloads, and we just update the new ones. This means that the next time this anomalous traffic is detected, our DB already includes those payloads.

To test this improvement in the DB, we will perform one more test of queries. After updating the DB, we now have all of the 5951 tokens in it. From the 5951 tokens we randomly select 20% to query the DB and again this is done 20 times. After finalising all the tests, we evaluate the percentage of tokens with two or more results in the DB. This time the results must be two or more, because each searched token was itself already in the DB.
The results of this evaluation are in Table 5.5 which represents the percentage of tokens with two or more results from the query. The Table also presents the change (Δ) in growth of accuracy in comparison with the test done before the improvement, which results are in Table 5.4. As observed this approach produces an increased accuracy, and as expected by continuing to update the DB with new tokens, its accuracy will continue to improve over time.

Table 5.5: Token query evaluation after continuous improvement and Δ growth of accuracy

Threshold	128	256	Δ 128	Δ 256
0.6	94,75%	94,32%	0,20%	0,35%
0.7	93,63%	93,39%	0,49%	0,27%
0.8	90,09%	90,46%	0,73%	0,79%
0.9	69,03%	73,68%	1,42%	1,62%
0.95	38,73%	39,87%	1,01%	1,25%
0.98	15,02%	12,73%	0,61%	0,46%
0.99	2,84%	2,45%	0,23%	0,18%

String Analyser

The string analyser is a set of tools that can validate if a string contains an attack or not. All the tools in this module will help to classify a string as an attack. Each tool is a function that receives as an input a payload and returns a value that states if that payload is an attack or not. For example the sanitiser we used in Section 4.5 has 100% accuracy in the character escaping for which it was designed. If we want to escape $spaces$ then we must include another sanitiser. Each implementation of these tools is done according to the SOC Analyst decision. These sanitisation checks were study by Shar *et.al.* [72] [73]. The Software Engineering Institute at the Carnegie Mellon University have extensive code standards accordingly to the programming language, like Java [74], Pearl [75] and C++ [76]. The PHP Group have a list of filters for sanitisation [77]. For Python there are several packages available like the Schema package [78] or the HTML package [79]. We did not test any individual tool, as the objective of this module for our work, is to improve the attack detection.

5.2 Evaluation Conclusion

In the beginning of this chapter we presented three questions. This evaluation was able to respond all of the question.

Q1 - Can the system detect anomalous traffic with high chances of including web attacks like SQLi or XSS?
Yes. In the Attack Detection, section 5.1.1 we were able to perform a cluster analysis on the NetFlow and detect a cluster with the anomalous traffic.

Q2 - Can the system analyse the payload content of the anomalous traffic previously detected and confirm the payloads are from attackers?
Yes. We were able to analyse the payload and with the use of LSH we confirmed that the payloads with origin in the scanner were web application attacks. As long as the Token DB in our SSE is prepared, the SSE can make an accurate evaluation of the tokens. For example using the SSE with 256 permutations and a similarity threshold of 0.8 we concluded with 89% positive rate, that the traffic was from a scanner.

Q3 - Can we improve the process?
Yes. We observed an improvement in the SSE, because of the updates to the Token DB. The improvement was between 0.18% and 1.62%. Although it is not much, it shows a tendency of improvement, and this compound improvement will help the accuracy of SSE as it keeps evolving.

In summary it was possible to detect the anomalous traffic originated by the scanner and segregate that traffic to analyse it with a SSE. Using the SSE with 256 permutations and a threshold of 0.8 we concluded with 89% positive rate, that the traffic was from a scanner, and that with a continuous improvement approach the accuracy of the SSE improved from said 89% to 90%.

Chapter 6

Conclusion

6.1 Conclusion

The use of web applications is a major part of the Internet success. The web applications are the ones that allow the remote work, sharing economy, education democratisation and many others paradigms. Because of the importance of web applications, they are the focus of many attacks. We have analysed the common approaches, to detect attacks in network traffic, and the most established tools to prevent and detect such attacks tend to loose performance when inspecting HTTPS traffic, the web protocol widely adopted nowadays. The use of DPI is not appropriate in the presence of HTTPS traffic, and the mainstream solution of using an intermediary proxy server to decrypt the traffic, introduces a new weak point susceptible of more attacks. We propose an approach to inspect HTTPS traffic in a light way for detection of web attacks, namely SQLi and XSS. Our approach was capable of identifying anomalous traffic with the use of Machine Learning. By using an unsupervised machine learning algorithm we were able to cluster the anomalous traffic in a NetFlow created from HTTPS traffic. We also develop and implemented a Similarity Search Engine capable of confirming that the anomalous traffic contains web application attacks, without the need to decrypt the traffic between the client and the server. In order to keep our solution viable, we develop a continuous improvement process, capable of extending the Similarity Search Engine accuracy as new attacks are observed. All of our solution was develop in a way that a SOC analyst can perform adjustments on any module, accordingly to his intents.

6.2 Future Work

In future work, we plan to use the continuous improvement process, to lengthen the Similarity Search Engine database, by including traffic from more web scanners. By growing our database, one important study to follow is the ability of the system to surpass scanners traffic and perform more broad analysis.

Bibliography

[1] TechnoHub. Difference Between HTTP and HTTPS Protocols for SEO. https://www.technohub.org/difference-between-http-and-https-seo/. Accessed: 2022-03-15.

[2] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, and A. Pras. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys Tutorials*, 16(4):2037–2064, 2014.

[3] CISCO. NetFlow Version 9 Flow-Record Format. https://www.cisco.com/en/US/technologies/tk648/tk362/technologies_white_paper09186a00800a3db9.html. Accessed: 2022-03-15.

[4] T. N. Rincy and R. Gupta. A Survey on Machine Learning Approaches and Its Techniques. In *Proc. of the IEEE International Students' Conference on Electrical,Electronics and Computer Science (SCEECS)*, pages 1–6, 2020.

[5] D. A. Kindy and A. K. Pathan. A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In *Proc. of the 15th IEEE International Symposium on Consumer Electronics (ISCE)*, pages 468–471, 2011.

[6] Aplextor Laboratory. Web Statistics. https://medium.com/@aplextorlab/web-statistics-69493eebbd01. Accessed: 2022-02-21.

[7] Debah Ahmed. Massive privacy risk as hacker sold 2 million MyFreeCams user records. *https://www.hackread.com*, 2021. Accessed: 2022-03-15.

[8] Open Web Application Security Project (OWASP). OWASP Top Ten. https://owasp.org/www-project-top-ten/. Accessed: 2022-03-15.

[9] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. Tls proxies: Friend or foe? 07 2014.

[10] K. Kent, S. Chevalier, T. Grance, and H. Dang. Recommendations of the National Institute of Standards and Technology. Technical report, 2006.

[11] Yadong Li, Danlan Li, Wenqiang Cui, and Rui Zhang. Research based on OSI model. In *Proc. of the 3rd IEEE International Conference on Communication Software and Networks*, pages 554–557, 2011.

[12] ISO/IEC JTC 1 Information technology - Technical Committee. Information technology - Open System Interconnection - Basic reference model - The basic model. Standard, Telecomunication standardization sector of Internacional Telecomunication Union, 07 1994.

[13] The Internet Society. Requirements for Internet Hosts – Communication Layers. `https://datatracker.ietf.org/doc/html/rfc1122`. Accessed: 2022-03-15.

[14] The Internet Society. Requirements for Internet Hosts – Application and Support. `https://datatracker.ietf.org/doc/html/rfc1123`. Accessed: 2022-03-15.

[15] The Internet Society. Hypertext Transfer Protocol – HTTP/1.1. `https://tools.ietf.org/html/rfc2616`. Accessed: 2022-03-15.

[16] The Internet Society. Hypertext Transfer Protocol Version 3 (HTTP/3) draft-ietf-quic-http-33. `https://tools.ietf.org/html/draft-ietf-quic-http-33#section-2`. Accessed: 2022-03-15.

[17] The Internet Society. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. `https://tools.ietf.org/html/rfc7231#section-4.1`. Accessed: 2022-03-15.

[18] Internet Assigned Numbers Authority (IANA). Permanent Message Header Field Names. `https://www.iana.org/assignments/message-headers/message-headers.xml`. Accessed: 2022-03-15.

[19] The Internet Society. HTTP Over TLS. `https://tools.ietf.org/html/rfc2818`. Accessed: 2022-03-15.

[20] The Internet Society. The TLS Protocol Version 1.0. `https://tools.ietf.org/html/rfc2246`. Accessed: 2022-03-15.

[21] NetApplications.com. Market Share Statistics for Internet Technologies. `https://netmarketshare.com/`. Accessed: 2022-03-15.

[22] A. Pramod, A. Ghosh, A. Mohan, M. Shrivastava, and R. Shettar. SQLI detection system for a safer web application. In *Proc. of the IEEE International Advance Computing Conference (IACC)*, pages 237–240, 2015.

[23] D. Naylor, A. Finamore, I. Leontiadois, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste. The Cost of the "S" in HTTPS. In *Proc. of the CoNEXT '14*, pages 1–7, 2014.

[24] The Internet Society. INTERNET ACCOUNTING: BACKGROUND. https://tools.ietf.org/html/rfc1272. Accessed: 2022-03-15.

[25] Carnegie Mellon University Software Engineering Institute . Cisco ASA and FTD SIP Inspection denial-of-service vulnerability. https://www.kb.cert.org/vuls/id/339704. Accessed: 2022-03-15.

[26] The Internet Society. Requirements for IP Flow Information Export (IPFIX). https://tools.ietf.org/html/rfc3917. Accessed: 2022-03-15.

[27] B.Claise. From NetFlow to IPFIX via PSAMP: 13 years of Standardization Explained. https://www.ietf.org/blog/netflow-ipfix-psamp-13-years-standardization-explained/. Accessed: 2022-03-15.

[28] Internet Assigned Numbers Authority (IANA). IPFIX Information Elements. https://www.iana.org/assignments/ipfix/ipfix.xhtml. Accessed: 2022-03-15.

[29] P. Matoušek, O. Ryšavý, M. Grégr, and M. Vymlátil. Towards identification of operating systems from the internet traffic: IPFIX monitoring with fingerprinting and clustering. In *Proc. of the 5th International Conference on Data Communication Networking (DCNET)*, pages 1–7, 2014.

[30] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras. Towards real-time intrusion detection for NetFlow and IPFIX. In *Proc. of the 9th International Conference on Network and Service Management (CNSM 2013)*, pages 227–234, 2013.

[31] Jordi Zayuelas I Muñoz. Detection of Bitcoin miners from network measurements. Master's thesis, Universitat Politècnica de Catalunya, 4 2019.

[32] Olivier van der Toorn, Rick Hofstede, Mattijs Jonker, and Anna Sperotto. A first look at HTTP(S) intrusion detection using NetFlow/IPFIX. In *Proc. of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 862–865, 2015.

[33] ISO/IEC JTC 1/SC 27 Information security, cybersecurity and privacy protection - Technical Committee. ISO/IEC 27001:2013. Standard, International Organization for Standardization, 10 2013.

[34] Z. Lin, X. Li, and X. Kuang. Machine Learning in Vulnerability Databases. In *Proc. of the 10th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 108–113, 2017.

[35] European Union Agency for Cybersecurity. List of top 15 threats. https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends/etl-review-folder/etl-2020-enisas-list-of-top-15-threats, 2020. Accessed: 2022-03-15.

[36] W. Zhenqi and W. Xinyu. NetFlow Based Intrusion Detection System. In *Proc. of the International Conference on MultiMedia and Information Technology*, pages 825–828, 2008.

[37] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys Tutorials*, 12(3):343–356, 2010.

[38] Maryam M. Najafabadi, Taghi M. Khoshgoftaar, Chad Calvert, and Clifford Kemp. Detection of SSH Brute Force Attacks Using Aggregated Netflow Data. In *Proc. of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 283–288, 2015.

[39] Sean Michael Kerner. How Was SQL Injection Discovered? `https://www.esecurityplanet.com/networks/how-was-sql-injection-discovered/`. Accessed: 2022-03-15.

[40] D. Patel, N. Dhamdhere, P. Choudhary, and M. Pawar. A System for Prevention of SQLi Attacks. In *Proc. of the International Conference on Smart Electronics and Communication (ICOSEC)*, pages 750–753, 2020.

[41] Girdhar Gopal Sonakshi*, Rakesh Kumar. Case study of sql injection attacks. *INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY*, 5(7):176–189, July 2016.

[42] M. Dayal Ambedkar, N. S. Ambedkar, and R. S. Raw. A comprehensive inspection of cross site scripting attack. In *Proc. of the International Conference on Computing, Communication and Automation (ICCCA)*, pages 497–502, 2016.

[43] M. Liu, B. Zhang, W. Chen, and X. Zhang. A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE Access*, 7:182004–182016, 2019.

[44] K. Gupta, R. Ranjan Singh, and M. Dixit. Cross site scripting (XSS) attack detection using intrustion detection system. In *Proc. of the International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 199–203, 2017.

[45] G. Habibi and N. Surantha. XSS Attack Detection With Machine Learning and n-Gram Methods. In *Proc. of the International Conference on Information Management and Technology (ICIMTech)*, pages 516–520, 2020.

[46] L. Lei, M. Chen, C. He, and D. Li. XSS Detection Technology Based on LSTM-Attention. In *Proc. of the 5th International Conference on Control, Robotics and Cybernetics (CRC)*, pages 175–180, 2020.

[47] A. L. Buczak and E. Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys Tutorials*, 18(2):1153–1176, 2016.

[48] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of massive datasets*. Cambridge University Press, 2020.

[49] Tommaso Rescio, Thomas Favale, Francesca Soro, Marco Mellia, and Idilio Drago. DPI Solutions in Practice: Benchmark and Comparison. In *Proc. of the IEEE Security and Privacy Workshops (SPW)*, pages 37–42, 2021.

[50] Zhihui Cheng, Mykola Beshley, Halyna Beshley, Orest Kochan, and Oksana Urikova. Development of Deep Packet Inspection System for Network Traffic Analysis and Intrusion Detection. In *Proc. of the 15th IEEE International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*, pages 877–881, 2020.

[51] Jeff Jarmoc. Transitive trust: Ssl/tls interception proxies, 3 2012.

[52] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. BlindBox: Deep Packet Inspection over Encrypted Traffic. *SIGCOMM Comput. Commun. Rev.*, 45(4):213–226, aug 2015.

[53] Jiangpan Hou, Peipei Fu, Zigang Cao, and Anlin Xu. Machine Learning Based DDos Detection Through NetFlow Analysis. In *Proc. of the IEEE Military Communications Conference (MILCOM)*, pages 1–6, 2018.

[54] Maryam M. Najafabadi, Taghi M. Khoshgoftaar, Chad Calvert, and Clifford Kemp. Detection of SSH Brute Force Attacks Using Aggregated Netflow Data. In *Proc. of the 14th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 283–288, 2015.

[55] Atieh Bakhshandeh and Zahra Eskandari. An efficient user identification approach based on Netflow analysis. In *Proc. of the 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)*, pages 1–5, 2018.

[56] David Tulloh. dumpio2curl.pl - Extracts dumpio output from Apache logs for debugging and replaying. `https://github.com/lod/dumpio2curl`. Accessed: 2022-03-15.

[57] Geoffrey Simmons. dumpio_parse.pl - Parse Apache error_logs. `https://uplex.de/replay/dumpio_parse.pl`. Accessed: 2021-04-14.

[58] Oracle. Oracle VM VirtualBox. `https://www.virtualbox.org/`. Accessed: 2022-03-15.

[59] DVWA team. Damn Vulnerable Web Application (DVWA). `https://dvwa.co.uk/`. Accessed: 2022-03-15.

[60] Wireshark Team. tshark - Dump and analyze network traffic. `https://www.wireshark.org/docs/man-pages/tshark.html`. Accessed: 2022-03-15.

[61] CERT Network Situational Awareness Team. SiLK, the System for Internet-Level Knowledge. https://tools.netsa.cert.org/silk/index.html. Accessed: 2022-03-15.

[62] CERT Network Situational Awareness Team. SiLK Documentation. https://tools.netsa.cert.org/silk/docs.html. Accessed: 2022-03-15.

[63] Apache. Apache ErrorLogFormat Directive. https://httpd.apache.org/docs/2.4/mod/core.html#errorlogformat. Accessed: 2022-03-15.

[64] Eric Zhu. datasketch: Big Data Looks Small. https://github.com/ekzhu/datasketch. Accessed: 2022-03-15.

[65] ZAP Dev Team. OWASP Zed Attack Proxy. https://www.zaproxy.org/. Accessed: 2022-03-15.

[66] Python Software Foundation. Python object serialization. https://docs.python.org/3/library/pickle.html. Accessed: 2022-03-15.

[67] Andy Dustman. MySQLdb User's Guide. https://mysqlclient.readthedocs.io/user_guide.html. Accessed: 2022-03-15.

[68] Oracle Corporation and/or its affiliates. MySQL Documentation. https://dev.mysql.com/doc/c-api/8.0/en/mysql-real-escape-string-quote.html. Accessed: 2022-03-15.

[69] The Tcpdump Group. tcpdump, a powerful command-line packet analyzer. https://www.tcpdump.org/index.html. Accessed: 2022-03-15.

[70] The Apache Software Foundation. The Apache HTTP Server Project. https://httpd.apache.org/. Accessed: 2022-03-15.

[71] Python Software Foundation. Python.org. https://python.org. Accessed: 2022-03-15.

[72] Lwin Khin Shar and Hee Beng Kuan Tan. Predicting common web application vulnerabilities from input validation and sanitization code patterns. In *Proc. of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 310–313, 2012.

[73] Lwin Khin Shar and Hee Beng Kuan Tan. Mining input sanitization patterns for predicting SQL injection and cross site scripting vulnerabilities. In *Proc. of the 34th International Conference on Software Engineering (ICSE)*, pages 1293–1296, 2012.

[74] Carnegie Mellon University. Input Validation and Data Sanitization. https://wiki.sei.cmu.edu/confluence/display/java/Input+Validation+and+Data+Sanitization. Accessed: 2022-03-15.

[75] Carnegie Mellon University. SEI CERT Perl Coding Standard. https://www.securecoding.cert.org/confluence/display/perl/CERT+Perl+Secure+Coding+Standard. Accessed: 2022-03-15.

[76] Carnegie Mellon University. CERT C++ Secure Coding Guidelines. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682. Accessed: 2022-03-15.

[77] The PHP Group. List of filters for sanitization. https://www.php.net/manual/en/filter.filters.sanitize.php. Accessed: 2022-03-15.

[78] Vladimir Keleshev. Python Schema Package. https://github.com/keleshev/schema. Accessed: 2022-03-15.

[79] Richard Jones. Python HTML Package. https://pypi.org/project/html/. Accessed: 2022-03-15.