University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Machine-learning based Detection of Malicious DNS-over-HTTPS (DoH) Traffic Based on Packet Captures

David Stalder
Zurich, Switzerland
Student ID: 13-929-872

Supervisor: Jan von der Assen
Date of Submission: April 18, 2021

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Das Ziel dieser Bachelorarbeit ist die Implementation eines funktionierenden Prototyps zur Erkennung von bösartigem DNS-over-HTTPS (DoH) Datenverkehr in das bereits bestehende System SecGrid, einer Plattform zur Extraktion von Netzwerkdaten, deren Analyse und der Erkennung von Cyber-Attacken, welche an der Universität Zürich entwickelt wurde. Die Implementation enthält eine Feature Extraction Komponente, welche speziell für die Extraktion von DoH Datenverkehr basierend auf TCP Datenfluss entwickelt wurde und einer zweilagigen Machine Learning Pipeline für die Erkennung von bösartigem DoH Datenverkehr. Die Auswertung beweist, dass der Prototyp für einzelne Datensätze sehr genau ist, jedoch sinkt die Genauigkeit drastisch, sobald die Machine Learning Modelle mit verschiedenen Daten trainiert und anschliessend getestet werden. Das Fazit ist, dass eine Diversifizierung der Datensätze nötig ist, damit sie besser abgestimmt sind auf verschiedene Browsereinstellungen und alle verfügbaren DoH Server, zusätzlich sollten die bereits vorhandenen Datensätze qualitativ und quantitativ verbessert und erweitert werden.

The goal of this thesis is to implement a working prototype for the detection of malicious DNS-over-HTTPS (DoH) traffic into the already existing System SecGrid, a platform for the extraction of internet traffic, its analysis, and the detection of cyber-attacks developed by the CSG-Group at the University of Zurich. The implementation contains a special feature extraction for DoH traffic based on TCP-flows and a two Layered Machine Learning pipeline for the detection of malicious DoH traffic. The evaluation proves that the prototype is extremely precise for single data-sets, but as soon as the models are trained and tested with different data the accuracy of the prototype deteriorates drastically. The conclusion is the diversification of the training data-sets into data-sets that are aligned with real-world browser settings and all available DoH resolvers and especially the quantitative and qualitative extension of the state-of-the-art data.

Acknowledgments

I would like to thank everyone involved in this project, especially to my primary supervisor Jan von der Assen for his support during this thesis. His permanent advice and constructive feedback had a direct impact onto the outcome of my thesis. Further I wan to thank everyone who did the proofreading of my documentation and helped me to improve it. Last but not least I want to thank my family and friends for their support during this time.

Contents

Chapter 1

Introduction

1.1 Motivation

Conventional DNS queries entail the problem that not only the user and the resolver, but also nearly everyone, can see the content of those queries. Therefore, DNS over HTTPS (DoH) was introduced in October 2018 with the goal to improve the internet security and user privacy by sending encrypted DNS queries using the HTTPS protocol. In September 2019, Firefox announced to adopt DoH into their browser [1] (see Figure 1.1) and since then the usage of this protocol experienced a steep increase [2], although other security approaches still form the vast majority.

Figure 1.1: Tweet [1] in which the Implementation of DoH was introduced

A huge benefit of DoH is that it protects the content of the traffic from the insight of third parties. But exactly because of this point, experts have expressed concerns [3], since it is also not possible for DNS monitoring systems to have a direct insight into the traffic. This complicates the inspection, the detection, and the blocking of DoH traffic and makes it a difficult task. Therefore, DoH brings along not only desired properties, but also many undesired properties, like bypassing of DNS monitoring Systems or exploitation of upstream DNS traffic [4]. One example for this type of misusage is the Godlua Backdoor Malware [5] which uses DoH to obtain the address of the C2. A more recent example for the misuse of DoH is the Iranian hacker group Oilrig (APT34) [6], which used DoH in combination with the tool DNSExfiltrator [7] to receive data without being noticed from a hacked network.

The encryption of DoH traffic makes direct deep inspection for malware detection a nearly impossible task. Nevertheless, there exists another way to get an insight in the data traffic

of DoH: leveraging the unique traffic shape. There exist approaches where the traffic is successfully analyzed using machine learning models. The SecGrid [8] system developed at the University of Zurich by the CSG Group provides a platform for granular feature extraction. The aim of this thesis is to implement a prototypical malicious DoH traffic detection component into SecGrid.

1.2 Description of Work

This thesis is separated into three different stages: the first stage is the introduction into the problem, in the second stage a literature research is conducted and the findings of it are presented, and into the final stage a working prototype is created, evaluated, and finally the whole work is documented.

The first stage initially comprises the fundamental understanding of DoH. Therefore, a deep insight into DNS, its encryption, and the detection of encrypted DNS queries is to be achieved to understand the underlying problem of this thesis to fulfil the objectives of the thesis. Further, the insight into the baseline system with its background will be the predominant part of this stage. The platform will be analyzed to get an overview of the existing system, the feature extraction and the current machine learning based clients that are already implemented into the system. The final step of this stage will be the decomposition of the thesis into tasks, such that a timeline can be established where the systematic implementation of the work can be planned.

In the second stage, the current state of the literature is surveyed. The first step is research on the background of DoH to get a detailed insight into the objective. This includes a survey about the specific characteristics of malware that misuse the security properties of DoH for cyberattacks. The main part of this stage will be a survey about the current approaches in the detection of malicious DoH traffic, with special focus on machine learning based approaches. In the subsequent part of this stage, the findings achieved in the literature research are presented, followed by the main design decisions for the thesis. Finally, the findings are adopted into the current system architecture. Therefore, the current platform architecture is redesigned, and the findings are implemented to show the technical feasibility and the usefulness of the work.

The third and last stage contains the implementation of a working prototype, where the findings and the design decisions of the second stage are implemented into the platform. The prototype contains a working DoH traffic detection component. The final part of this stage is the evaluation and the documentation of the work.

1.3 Thesis Outline

This report is structured as follows. Based on the formulation of the thesis goal in this Chapter 1, Chapter 2 delivers the background knowledge on which this thesis is built. The focus there is laid to the DNS-over-HTTPS protocol, on its basics, strengths, vulnerabilities, and malware

which is already abusing it. Furthermore, SecGrid is presented in more detail, including the file-format of the Packet-Capture. Chapter 3 is dedicated to the findings of the literature research. Thus, the focus there is laid on current solutions for the detection of malicious DNS-over-HTTPS traffic, with special interest on Machine Learning based approaches. In Chapter 4, the design of the architecture of the prototype for malicious DNS-over-HTTPS traffic detection, which is implemented into SecGrid is presented. Chapter 5 treats the implementation of the prototype into SecGrid step by step. Foremost, the Feature Extraction is described, followed by the extraction of the data-set which was used for the training data of the Machine Learning model and ultimately the Machine Learning model implementation is described. In Chapter 6 the evaluation of the prototype is conducted, whereas the Feature Extraction and the ML Model are focused. Finally, Chapter 7 recapitulates the work and the findings of this thesis, the conclusions and limitations are presented, and future work is stated.

Chapter 2

Background

The purpose of this chapter is to establish the background knowledge for this thesis. First, an introduction into the Domain Name System and the Hyptertext Transfer Protocol Secure including their vulnerabilities which are found up to date builds the basis of the chapter. Then, the functionality of the Transmission Control Protocol is introduced. The core of this chapter is the Section DNS-over-HTTPS, where the functionality, the strengths, and the vulnerabilities including a survey of the malware abusing the protocol are presented. In the end of the chapter, the platform SecGrid including a definition of PCAP-Files is shown.

2.1 Domain Name System

The Domain Name System (DNS) is a protocol that is used to process naming resources in such a way that different users, networks etc. are all able to understand and identify the respective naming resource in their own language. An example for it is a user who types a URL into a web-browser. Since this URL is only understandable by human beings, it has to be translated into a numerical IP-address and made understandable for the internet. This part is taken by the DNS, which is a directory that administers the namespace of the whole internet. DNS has been introduced in 1993 and has since then become a crucial part of the internet [9].

However, in this early time of the internet nearly no one thought about internet security and therefore lone DNS-queries are highly endangered to suffer cyberattacks. [10] found three different ways to view the vulnerabilities of DNS: the conceptual view, the structural view and the communication view. In the conceptual view, they applied a model for testing the information security called CIA Triad, which is structured into three parts: the confidentiality, the integrity and the availability. In all three parts, this model showed that DNS is not secure and prone to attacks or failures of the servers due to the fact that DNS is not encrypted and has a hierarchical structure, such that anyone can tamper it. In the structural view, they found that the hierarchical tree structure of DNS servers makes it easier for attackers to attack several services used by a lot of users at the time. Another finding in this view was the exposure of DNS server information, which is caused by insufficient security configurations of the DNS servers. This allows an attacker to send malicious data to the user and the user still

believes that this data is secure. In the third view, the communication view, they found that the packets are not secured through the usage of UDP, caching problems caused by Cache Poisoning can occur and there is insufficient protection against Distributed Denial of Service attacks. In summary, those three views show clearly that DNS is vulnerable in many aspects, which is the reason that in the evolution of the internet security many more security protocols have been developed [10].

2.2 Hypertext Transfer Protocol Secure

The Hypertext Transfer Protocol (HTTP) [11] was introduced in the early 1990s and has since then become a crucial part of the internet. It defines the communication of the web-server and the web-browser, communicates by using the Transmission Control Protocol (TCP) [12] and works on the Application Layer of the OSI-Model [13]. The Hypertext Transfer Protocol Secure (HTTPS) [14] extends the HTTP, whereas the major difference between HTTPS and HTTP is that HTTPS additionally is encrypted by using Secure Sockets Layer (SSL) [15] or its successor, the Transport Layer Security (TLS) protocol [16]. HTTPS uses port 443 by default, and its URLs start with "*https://*". The common start of a TLS connection [17] is a *handshake* between the resolver and the client, where the Public Key Infrastructure (PKI) is used to first authenticate the resolver, afterwards a symmetric key for the session is generated by asymmetric encryption [18].

[19] listed a compilation of possible attacks on HTTPS. *Protocol Version Downgrade* happens when a Man-In-The-Middle attacker sends `ClientHello` messages until the client's protocol version is reduced to an older one, which is more prone to attacks. *RSA Decryption oracles* happen in HTTPS since the padding scheme of TLS is assailable to padding oracle attacks, which means that the attacker is able to decrypt a cryptographic message. *Heartbleed* means that an attack could uncover the long-ranging private keys of the resolver by exploiting memory management problems in resolver implementations. They organized insecure channels into three different categories: leaky, tainted, and partially leaky channels. Leaky channels arise when a connection to a vulnerable channel is classified as confidential. The attacker can here try to get the Premaster Secret and with it the possibility to decrypt and read all the recorded network traffic. Tainted channels are prone to Man-In-The-Middle-Attacks which enable the attacker to decrypt, read and modify the communication between the client and the server. Partially leaky channels enable the attacker to find out sporadic small secrets in the long run. By abusing the secret repetition assumption, the attacker can find out those small secrets by capturing the exchange of repeated messages. Summarizing they found that although HTTPS nowadays is necessary, it is not entirely safe due to weaknesses of the underlying TCP implementation, but with the advancing improvement of TLS the internet security also rises [19].

2.3 Transmission Control Protocol

The Transmission Control Protocol (TCP) [12] defines how data is exchanged between the different network components. It was first introduced in 1981 and its last extension was

published in RFC7323 in 2014 and works on the transport layer of the OSI-Model. A TCP connection between two endpoints [18] can be clearly identified by the IP-address and the port of the client, and the IP-address and the port of the server and work by exchanging so-called packets. When a client wants to establish a connection to the server, it has to send a synchronization (SYN) packet to the server. If the server cannot establish this connection, it answers with a reset (RST) packet to show the client that it is (currently) unavailable. But if the server is available, it answers with a synchronization acknowledgement (SYN/ACK) packet. The client on the other hand sends an acknowledgement (ACK) answer to say that it received the SYN/ACK packet. After this sequence, which is illustrated in Figure 2.1 on the left, the connection is established and the server and the client are able to exchange data.

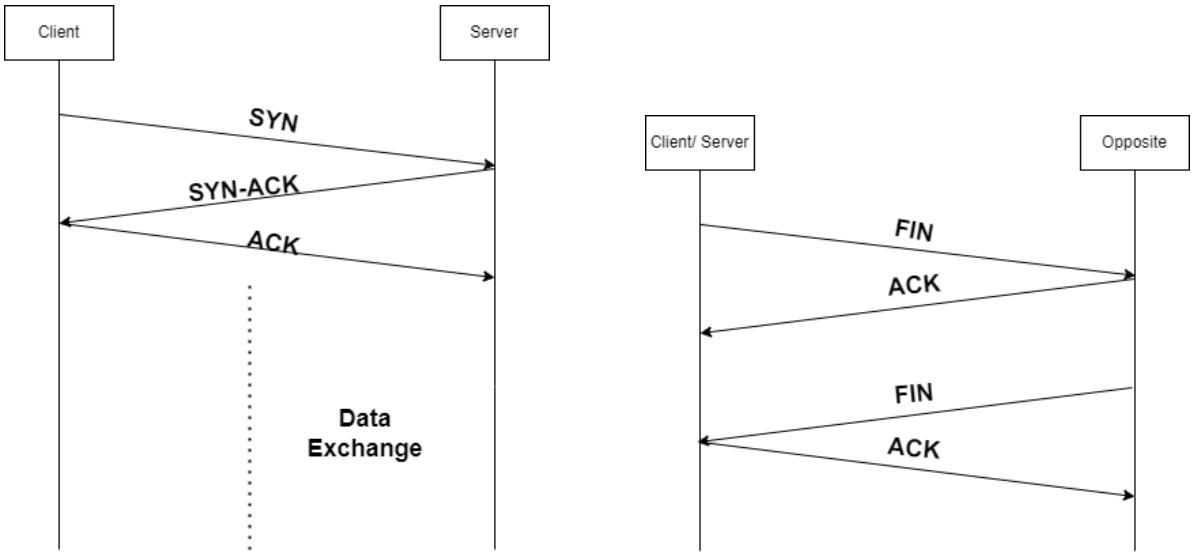


Figure 2.1: Start (left) and End (right) of a TCP connection

If one side (the server or the client) wants to finish the connection [20] and sends no more data, it sends a finish packet (FIN) to the opposite. The opposite confirms that it received the FIN packet by sending an ACK packet. Now the opposite sends a FIN packet back to the side which sent the first FIN packet, and this side confirms the receipt of this FIN packet with an ACK packet as well. After this closing process, which is illustrated in Figure 2.1 on the right, the opposite can close the connection. Another possibility to close a TCP connection is simply by sending an RST package to the opposite side and then the connection is closed. [20] stated that during this process, many things can go wrong, which he called *abnormal terminations*. Those abnormal terminations could be either an interrupted setup or a disconnection, and they could occur due to the following reasons: There could be an insufficient amount of resources or a network disrupt could occur, the session could crash due to a bug in the implementation, one side has already closed the connection and the other side continues sending data, or the resolver declines to establish a connection with the client [20].

2.4 DNS-over-HTTPS

DNS-over-HTTPS (DoH) is a protocol used for the traffic of DNS queries using the HTTPS protocol, which was first introduced in 2018. There are two different approaches of DoH, the

first one uses DNS in wire-format enclosed in HTTPS [21], the other one uses DNS represented in JSON format [22]. Both approaches are supported by the majority of all implementations, but the more commonly used approach is DNS in wire-format [23], therefore the main focus of this Bachelor Thesis will be laid on the DNS in wire format approach. There are several public DoH servers [24] available, inter alia *Cloudflare*, *Google*, *Quad9*, and *AdGuard*.

2.4.1 Strengths

[25] conducted a large-scale measurement of DNS-over-Encryption, inter alia, of DoH. They stated that the mixture of DoH queries with other HTTPS traffic effectively resists traffic analysis that only targets DNS queries. This improves the protection of the content of DNS queries and with it the privacy of the traffic significantly, in other words it becomes difficult for an attacker to get an insight of the DNS queries. Another point is that DoH demands both encryption and authentication of servers. If a server is unable to provide these two security measures, the DoH query will fail and no data exchange will take place between the respective server and the client. Due to this fact, already the detection of DoH servers becomes very challenging [25]. [4] stated further that the authenticated and encrypted responses from the servers are immune from unauthorized modification by attackers.

2.4.2 Vulnerabilities

DoH entails the benefit that the DNS traffic between client and resolver is protected from unauthorized access, but this fact also leads to issues. [4] presented the following issues: DoH conveys a false sense of security, DNS monitoring can be bypassed, the DNS upstream can be exploited, and there are concerns for internal network configurations and information. DoH has a lot of properties that promise being absolutely protected against cyberattacks. But in return, there are possibilities for an attacker to gain information about the connection the client made to the server. This leads to a false sense of security. DNS monitoring has the function to filter and stop traffic from known malicious domains. Using DoH makes it impossible for these monitoring services to have a direct insight into the plaintext DNS traffic. If now external DoH servers are used, attackers get the possibility to get insight into the encyrpted DNS traffic. If an organization uses a device or an application that is configured to use an external DoH server, it ignores internal security measures and connects directly to that external resolver. This fact is a huge concern for internal network configurations and information because if now the client wants to connect to an internal domain, the device or the application sends the query first to the preferred and external resolver before at worst failing the internal resolver. Thus, it becomes possible that internal network information is revealed to unauthorized persons. DoH takes place only on the "last mile" between the client who starts the DNS request and the DoH server which is addressed by the client. If the DoH server transmits the traffic to top-level root DNS servers, it could occur that the traffic from this point is not encrypted anymore. Thus, an attacker who can exploit the upstream DNS traffic, can here have an insight on the plaintext DNS traffic, and can try to divert the traffic to a malicious server [4].

2.4.3 Malware

As seen in the previous Section, DoH is not fully secure. On this account, there have recently already been cyberattacks that abuse the security properties of DoH, in particular the fact that there is no possibility for insight into DoH traffic [26]. This Section creates an overview over the most common malware that abused the properties of DoH in the recent past, namely the Godlua Backdoor, PsiXBot, and Oilrig's APT34.

In April 2019, [5] found a Lua-based Backdoor-file. Lua [27] is a programming language which is according to [28] a pretty popular and upcoming choice for web applications. A Lua-based Backdoor-file is a Lua Code Injection, which leaves a "backdoor" for an attacker to inject malicious queries into the affected web-application. For C2 connections, Godlua Backdoor employs a redundant communication technique, where attackers use a combination of several ways to store a C2 address, and at the same time they use HTTPS and DoH to ensure the secure communications for their bots [5].

In early 2019, PsiXBot [29], which uses Google's DoH service to get the domain name of Google's C2, was observed for the first time. With the knowledge of the domain name of the C2, PsiXBot can encrypt its DNS traffic and insert it masked into the normal HTTPS traffic and therefore circumvent the DNS firewalls. It also uses the method *Fast Flux* [30], which is a method to rapidly change the DNS entries by the usage of a Botnet to avoid being detected.

Oilrig, also known as APT34, is an Iranian hacker group which became in 2020 the first of their kind to incorporate the DoH protocol into its attacks [6], using their until then newest tool, the DNSExfiltrator [7]. This tool is capable of transferring data between two points using the DoH protocol. Oilrig used these abilities to laterally shift data across an internal network to exfiltrate the channel, where they could take data undetected from monitoring systems, using exactly the security property of DoH that does not allow having an insight into the traffic.

2.5 SecGrid

SecGrid [8] is a platform that facilitates Machine Learning for the Analysis, Classification and Visualization of cyber-attacks. It is written in JavaScript (JS) and implements a set of miners, which are able to analyze network traffic files (i.e. PCAP-files [31]) and gain an insight into possible attacking patterns and classify them. Furthermore, the suggested miners are extensible.

Figure 2.3 shows the current architecture of SecGrid. In the *User Layer*, the user can choose already available data-sets or upload new ones to detect in the web-based interface. There, the data is forwarded to the RESTFUl-API, which takes the part of the data manager in the *Data Layer*. If a new data-set is uploaded, the data-manager forwards it to the data extraction pipeline, where it gets analyzed and forwarded back to the *User Layer*. In the *User Layer*, the data is visualized in the *Vizualization Module*, such that they deliver meaningful information to the user.

2.6 PCAP-Files

Packet Capture (PCAP) [31] files are used to capture live packet traffic data. The files can be generated with network analyzers like *Wireshark* [32]. The packets in turn can be used to analyze the traffic which is done by the client. The features that are recorded in PCAP files are listed in Table 2.1. Most of the features are self-explanatory, like *No.* (Number of the packet), *Time*, *Source*, *Destination*, *Protocol*, and *Length*. The feature *Info* contains the most important information about the packet, since here the user can see which type of message this packet is, e.g. in terms of the TPC protocol the PCAP file can capture if it was a SYN packet, or an ACK packet, etc. Figure 2.2 shows the summary of the features of an example PCAP file parsed in *Wireshark*. Other PCAP-parsers may extract slightly different features.

Feature	**Explanation**
No.	Number of the packet
Time	Time the packet was captured
Source	Source IP of the packet
Destination	Destination IP of the packet
Protocol	Protocol that was used for sending this packet
Length	Length of the packet (in bytes)
Info	Info about the packet

Table 2.1: Features of a PCAP File in Wireshark

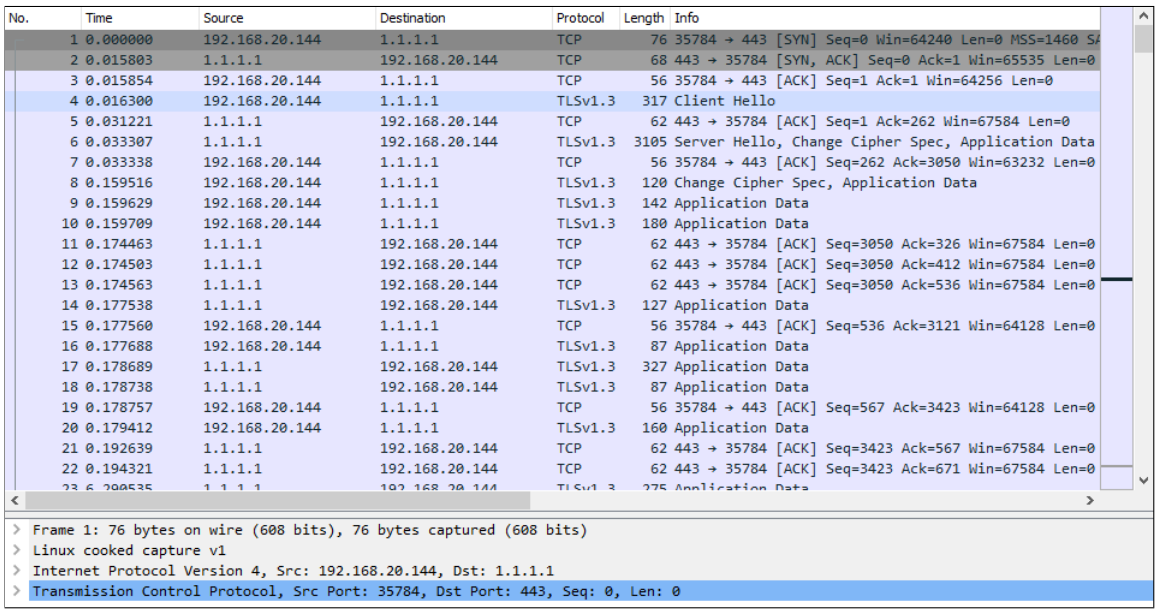


Figure 2.2: Example PCAP File in Wireshark

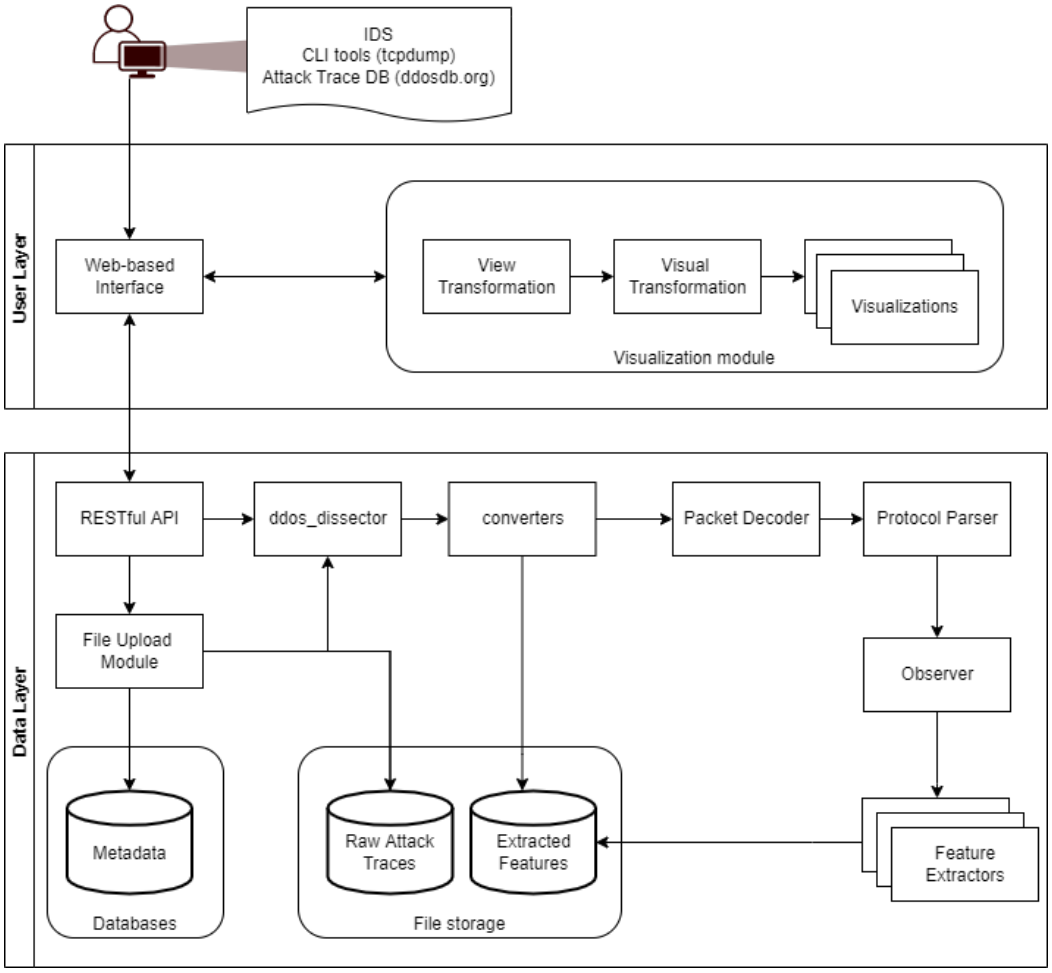


Figure 2.3: Current Architecture of SecGrid [8]

Chapter 3

Related Work

Although DoH is relatively new compared to other security protocols, there is already a contemplative amount of work done in relation to it. The main focus of this chapter is to create an overview of the work which was already conducted concerning DoH. The first reference shows a study where downgrading attacks on DoH connections were conducted in four different ways, followed by the second reference which presents ways to detect and analyze DoH traffic using existing tools.

All the following references show how Machine Learning (ML) or Deep Learning (DL) models can be used to detect malicious DoH traffic. The third reference shows that it is possible to separate DoH traffic from other traffic by using Machine Learning models by first conducting a detailed feature analysis of DoH traffic and later testing different Machine Learning models. The fourth reference sets the cornerstone for all the upcoming references by creating a dataset containing malicious and benign DoH traffic and non-DoH traffic, including a feature extraction module. The next reference tried to separate malicious DoH traffic from benign traffic by using Machine Learning models. The sixth reference did essentially the same task but this time by using a layer-approach where he separated DoH traffic from non-DoH traffic in the first Layer and then in the second Layer separated benign DoH traffic from malicious traffic. He also tested several Machine Learning models. The seventh reference took the work of the sixth reference and improved it by conducting statistical tests for a more efficient feature extraction and Machine Learning model performance. The eighth reference and the last one in the Machine Learning Section designed a three-layered approach, where they took the idea with the two-layers of the sixth reference and tried to find out which DoH tunnel tool was causing the malicious traffic in the third Layer. The last reference in this chapter used Deep Learning approaches for the detection of malicious DoH traffic. In Table 3.7, the key dimensions of every reference are summarized.

3.1 Downgrading Attacks on DoH

[33] conducted a study where they carried out DoH downgrade attacks on six different browsers (Chrome, Firefox, Edge, Brave, Opera, and Vivaldi). A DoH downgrade attack translates DoH

traffic into plain text DNS or in other words, the encrypted DoH connection is turned into an unencrypted DNS connection. DoH communication has two phases: in the first phase the client connects for the first time with the resolver using an unencrypted DNS request and in the second phase the reliable connection to the DoH server is established and from this moment client and server only communicate with encrypted HTTPS GET and POST methods. In both phases, attackers who are able to intervene can keep the communication on the lower security level. [33] proposed four different attack vectors during the two mentioned phases: DNS Cache Poisoning, DNS Traffic Interception, TCP Reset Injection, and TCP Traffic Interception. They tested every combination of browser and attack vector and found that every single test lead to a successful attack. As countermeasures, they proposed revisions of DoH implementations or DoH protocols [33].

3.2 Capturing and Analyzing DoH Traffic

[34] presented a study where he first conducted several tests to detect encrypted DNS traffic, especially DoH traffic. His goal was to show how attackers can use DoH to bypass the existing monitoring of organizations. Furthermore, he proposed a system that is able to detect and restrain unintentional DoH traffic. He also presented Zeek, which can be used to observe the DNS traffic and record the traffic in Zeek logs, which can be passed to another tool called RITA. RITA can be used to interpret the Zeek logs statistically to identify malicious traffic such as beaconing. His findings are that companies can protect themselves by using an adequate analysis and monitoring setup [34].

3.3 Machine Learning Based Approaches

The references in this Section used ML based approaches. The first reference did an expanded feature analysis of DoH traffic, followed by the trial of several ML models to detect DoH traffic. In the second reference, the basis was built for all the upcoming references by creating a data-set and an associated feature extraction. The further references evaluated different ways to detect malicious DoH traffic using ML approaches.

3.3.1 Separation of DoH Traffic

[35] executed a detailed feature analysis and evaluated the five popular Machine Learning classifiers K-Nearest Neighbours (5-NN), C4.5 Decision Tree, Random Forest, Naïve Bayes, and Ada-Boosted Decision Tree to differentiate DoH traffic from classic DNS traffic. Due to a lack of sources, they created their own data set using on the one hand Google Chrome and Mozilla Firefox in two separate Virtual Machines and controlled them both with the Selenium Framework to create browser DoH traffic and on the other hand they collected the traffic by using a DoH proxy, i.e. Cloudflare. They wrapped up all the gathered data in PCAP-files. The feature analysis (listed in Table 3.1) revealed that there is an essential difference between

the duration of a DoH connection and the connection of a single HTTPS connection, since the DoH connection is in the most cases established once and then lasts until the end of the secure connection, whereas the single HTTPS connection lasts for a shorter time (Feature 1). If the HTTPS connection lasts longer there might be file downloading or video streaming etc. involved, but then this kind of HTTPS connection tends to exchange a much higher amount of data in shorter time than a DoH connection, thus the amount of data transmitted can also be a distinctive feature (Feature 2). The size of the transmitted packets (Feature 3) is another indication for a DoH connection: while the size of transmitted packets in normal HTTP packets is very big, the size of DoH packets is much smaller. But this feature is considered as less significant, since HTTP packets can also have same sizes like DoH packets, depending on the purpose of the connection. A further feature (Feature 4) is the symmetry of the amount of incoming and outgoing data of a DNS query compared to HTTPS traffic. While in common DNS the amount and size of requests and responses is nearly identical, it is also similar in the beginning of an HTTPS connection, but the longer the HTTPS connection lasts, the more asymmetric the ratio becomes.

Considering the results, the authors found that in the two cases DoH Client evaluation and DoH recognition Naïve Bayes delivered the worst results, whereas the Ada-Boosted Decision Tree delivered the best results in both cases, namely 99.6% accuracy and 0.996 F1 score in the DoH recognition and 99.9% accuracy and 0.999 F1-score in DoH client classification. Although Naïve Bayes delivered the worst results in this study, it was nevertheless comparatively good with a high precision, which points out that the feature vector they chose is very stable. However, they limited their work by saying that the DoH detection and the client recognition are only possible when multiple DNS queries are connected, single connections are not possible to detect with their approach [35].

No.	Feature
1	Duration of the Flow
2	Amount of Data sent in the Flow
3	Size of the transmitted Packets
4	Symmetry of the amount of incoming and outgoing data

Table 3.1: Most Important Features found by [35]

3.3.2 Feature Extraction

[36] created a framework called DoHlyzer [37] that is able to extract necessary features for the classification and characterization out of DoH traffic. The framework uses the scapy [38] library written in Python to detect PCAP files which contain the network traffic and which are created with tools such as *Wireshark* or *tcpdump*. The module DoHMeter which he implemented into DoHlyzer is capable to extract a set of statistical features. Additionally, he added the header information Source IP, Destination IP, Source Port and Destination Port to be able to identify the flow which are listed in Table 3.2.

He introduced a clumping process in which he clumped the flows to reduce the size of the flows, whereas each of the features is extracted from a clump. To avoid that different flows are aggregated in the same clump, he used the header information and a timeout value. He separated the statistical features into information about the amount of bytes sent and received listed in Table 3.3, statistical information about the packet length in one flow listed in Table 3.4, statistical information about the packet time in one flow listed in Table 3.5, and statistical information about the packet response time between an outgoing query and the following response in one flow listed in Table 3.6.

Feature	**Explanation**
Source IP	The IP from which the query was sent
Destination IP	The IP which received the query
Source Port	The Port from which the query was sent
Destination Port	The Port which received the query

Table 3.2: Header Information [36]

Another part of his work was the creation of a data-set named *CIRA-CIC-DoHBrw-2020*. This data-set contains HTTPS traffic flows which are in one level separated into non-DoH traffic and DoH traffic, in the second level the DoH traffic is separated into benign and malicious traffic, and in the third level the malicious DoH traffic is separated into the traffic of three different tunneling tools (*iodine*. *DNS2TCP*, and *DNScat2*). He used four public DoH providers in his work, namely *AdGuard*, *Cloudflare*, *Google DNS*, and *Quad9*. Finally, he made his data-set publicly available under [39] in PCAP-format as well as in CSV-format.

Parameter	**Feature**	**Explanation**
F1	Number of flow bytes sent	The amount of bytes sent in the current flow in bytes
F2	Rate of flow bytes sent	The rate of bytes sent in the current flow in bytes/second
F3	Number of flow bytes received	The amount of bytes received in the current flow in bytes
F4	Rate of flow bytes received	The rate of bytes received in the current flow in bytes/second

Table 3.3: Information about the Amount of Bytes sent and received [36]

Parameter	**Feature**	**Explanation**
F5	Mean Packet Length	The mean of all packet lenghts in one flow in bytes

F6	Median Packet Length	The median of all packet lenghts in one flow in bytes
F7	Mode Packet Length	The mode of all packet lenghts in one flow in bytes
F8	Variance of Packet Length	The variance of all packet lenghts in one flow in bytes
F9	Standard Deviation of Packet Length	The standard deviation of all packet lenghts in one flow in bytes
F10	Coefficient of Variation of Packet Length	The coefficient of variation of of all packet lenghts in one flow in bytes
F11	Skew from Median Packet Length	The skew of each packet compared to the median packet length of the flow in bytes
F12	Skew from Mode Packet Length	The skew of each packet compared to the mode packet length of the flow in bytes

Table 3.4: Statistical Information about the Packet Length [36]

Parameter	**Feature**	**Explanation**
F13	Mean Packet Time	The mean of all packet durations in one flow in seconds
F14	Median Packet Time	The median of all packet durations in one flow in seconds
F15	Mode Packet Time	The mode of all packet durations in one flow in seconds
F16	Variance of Packet Time	The variance of all packet durations in one flow in seconds
F17	Standard Deviation of Packet Time	The standard deviation of all packet durations in one flow in seconds
F18	Coefficient of Variation Packet Time	The mean of all packet durations in one flow in seconds
F19	Skew from Median Packet Time	The skew of each packet compared to the median packet duration of the flow in seconds
F20	Skew from Mode Packet Time	The skew of each packet compared to the median packet duration of the flow in seconds

Table 3.5: Statistical Information about the Packet Time [36]

Parameter	Feature	Explanation
F21	Mean Request/ response time difference	The mean of the duration difference of an outgoing packet and the following response of all packet durations in one flow in seconds
F22	Median Request/ response time difference	The median of the duration difference of an outgoing packet and the following response of all packet durations in one flow in seconds
F23	Mode Request/ response time difference	The mode of the duration difference of all an outgoing packet and the following response of all packet durations in one flow in seconds
F24	Variance of Request/ response time difference	The variance of the duration difference of an outgoing packet and the following response of all packet durations in one flow in seconds
F25	Standard Deviation of Request/ response time difference	The standard deviation of the duration difference of an outgoing packet and the following response of all packet durations in one flow in seconds
F26	Coefficient of Variation of Request/ response time difference	The coefficient of variation of of the duration difference of an outgoing packet and the following response of all packet durations in one flow in seconds
F27	Skew from Median Request/ response time difference	The skew of each packet compared to the median of an outgoing packet and the following response of all packet durations in one flow in seconds
F28	Skew from Mode Request/ response time difference	The skew of each packet compared to the mode of an outgoing packet and the following response of all packet durations in one flow in seconds

Table 3.6: Statistical Information about the Packet Response time between an outgoing Query and the following Response in one Flow [36]

3.3.3 Detecting Malicous DoH Traffic

[40] used five different Machine Learning classifiers (Naïve Bayes, Logistic Regression, Random Forest, K-Nearest Neighbour, and Gradient Boosting) to detect malicious DoH traffic. They used the data-set of [36], which they combined in a new CSV file. After revising the file and removing all the rows containing null values, they ended up with a file with about 250'000 samples. In the next step they selected 31 features from the combined data set, but unfortunately due to page limitation they did not discuss them any further. Finally, the best result was delivered by the Random Forest classifier with an accuracy of 99.99% and an F1 score of 1.0 and the Gradient Boosting classifier with an accuracy of 99.97% and an F1 score of 1.0, all the other classifiers delivered non-satisfying results.

3.3.4 Two-Layered Approach

[41] used six different Machine Learning classifiers (Decision Tree, Extra Tree, Gradient Boosting, Light Gradient Boosting Machine, XGBoost, and Random Forest) in a two Layer approach to detect DoH traffic and separate it from DNS traffic in Layer 1 and separate the data into benign and malicious DoH traffic in Layer 2 (see Layer 1 and Layer 2 in Figure 3.2). He used the data of [36] as PCAP files and filtered 20'000 samples of each kind (benign and malicious), of which he used 90% as training set and 10% as testing set. He extracted 34 features which are contained in the data set and compared them finally statistically. His findings are that the most important features per Layer are *DestinationIP* and *ScourceIP* for Layer 1, for Layer 2 the most important feature is *tan(PacketLengthMode)*. In sum, the most important features across the two Layers are *PacketLengthMedian*, *PacketLengthMode*, and *PacketLengthSkewFromMode*. Concerning the Machine Learning classifiers, he found that in both Layers Light Gradient Boosting Machine and XGBoost delivered the best results with a maximum accuracy of 100%.

3.3.5 Feature Analysis

[42] rebuilt and tried to improve [41]'s work. They tested 10 different Machine Learning classifiers (Decision Tree, Random Forest, Light Gradient Boost Machine, XGBost, Linear Discriminant Analysis, K-Nearest Neighbours, Gaussian Naïve Bayes, AdaBoost, Gradient Boosting, and Extra Trees) and determined the most effective and efficient amongst them for the usage of a two layered approach like in [41]. Additionally they introduced feature selection methods to improve the performance of the Machine Learning classifiers and the usability in commercial usage. They used the data-set of [36] and like [41] they extracted 34 features.

The problem of such an enormous amount of features is that the Machine Learning model can easily be overfit which leads to poor results. Within [41]'s work, they found that the most important features (*DestinationIP* and*SourceIP*) had only about 30 different IP addresses in a data-set of 40'000 data points, which could have led to an overfitting problem of [41]'s model. They found similar problems with the features *DestinationPort*, *SourcePort*, and *Timestamp*. Therefore, they removed those five features from the data-set and ended up with a list of 29 features.

For the remaining features, they introduced and performed two different statistical tests for an improved feature selection: the Chi-Squared Test and Pearson Correlation Coefficient Test. With those two methods, they were able to detect features with statistical significance and ended up with a list of decreasing importance of the features for each Layer. The application of Feature Distribution Graphs finally revealed that the *Duration* and *ResponseTime-TimeSkewFromMedian* were the most important features for Layer 1, *PacketLengthStandard-Deviation* and *PacketLengthCoefficientofVariation* were the most important features for Layer 2. In Figure 3.1 the features are sorted by importance and Layer (Layer 1 on the left and Layer 2 on the right) including their *p-values*. The features marked red were the features that were classified as negligible after the two statistical tests.

After they found out the statistical significance of the different features, they tested the ten different Machine Learning models with the remaining 29 features to find out which model

performs the best. They chose the three best performing models: Random Forest, LGBM, and Decision Tree for Layer 1 and Random Forest, LGBM and XGBoost for Layer 2). With those three models for each Layer, they executed a Sequential Forward Selection (SFS), where they ran tests with an increasing number of features starting at one feature and ending up with 29 features. For both Layers, Random Forest performed the best due to its high accuracy, in both Layers followed by Light Gradient Boost Machine due to its high accuracy and low training time.

They unveiled that the Random Forest model is the preferred model for both Layers if it is trained only once, but if the task requires continuous training of the model, Light Gradient Boost Machine can be the preferred model due to very low training times. Besides that, they also found that with 21 features (according to the sequence of the feature-importance for the respective Layer in Figure 3.1) the models reached the optimal precision and accuracy, such that it is not necessary to have more features in the data-set. They limited their work with the statement that they used the default models of scikit-learn with no further use of hyperparameter tuning.

	Layer 1		Layer 2	
#	Feature	p-value	Feature	p-value
1	Duration	0.00	PacketLengthStandardDeviation	0.00
2	ResponseTimeTimeSkewFromMedian	0.00	PacketLengthCoefficientofVariation	0.00
3	ResponseTimeTimeMode	0.00	FlowReceivedRate	$1.45e^{-244}$
4	ResponseTimeTimeMedian	0.00	PacketLengthMean	$7.98e^{-217}$
5	ResponseTimeTimeMean	0.00	Duration	$2.51e^{-216}$
6	PacketTimeSkewFromMedian	0.00	PacketTimeSkewFromMedian	$4.56e^{-188}$
7	PacketTimeMode	0.00	FlowSentRate	$1.86e^{-176}$
8	PacketTimeMedian	0.00	PacketLengthVariance	$5.35e^{-147}$
9	PacketTimeMean	0.00	PacketTimeMean	$1.86e^{-131}$
10	ResponseTimeTimeSkewFromMode	0.00	PacketTimeStandardDeviation	$3.62e^{-129}$
11	PacketTimeVariance	0.00	ResponseTimeTimeMedian	$3.36e^{-115}$
12	PacketLengthCoefficientofVariation	0.00	PacketTimeMedian	$5.14e^{-95}$
13	PacketTimeStandardDeviation	0.00	ResponseTimeTimeSkewFromMode	$9.87e^{-91}$
14	PacketLengthMode	0.00	ResponseTimeTimeMean	$1.58e^{-62}$
15	PacketLengthMedian	0.00	ResponseTimeTimeMode	$4.34e^{-61}$
16	PacketLengthMean	0.00	PacketTimeCoefficientofVariation	$6.08e^{-59}$
17	FlowBytesSent	0.00	ResponseTimeTimeSkewFromMedian	$1.38e^{-49}$
18	ResponseTimeTimeCoefficientofVariation	0.00	PacketTimeMode	$8.24e^{-34}$
19	PacketLengthStandardDeviation	$2.46e^{-133}$	FlowBytesSent	$3.98e^{-29}$
20	PacketLengthVariance	$2.84e^{-130}$	FlowBytesReceived	$4.97e^{-28}$
21	PacketTimeCoefficientofVariation	$2.44e^{-48}$	PacketLengthMode	$3.10e^{-26}$
22	FlowReceivedRate	$3.03e^{-41}$	ResponseTimeTimeCoefficientofVariation	$1.83e^{-14}$
23	ResponseTimeTimeStandardDeviation	$1.89e^{-36}$	PacketLengthSkewFromMedian	$8.14e^{-11}$
24	PacketLengthSkewFromMode	$1.36e^{-11}$	PacketTimeVariance	$8.36e^{-6}$
25	FlowBytesReceived	$2.80e^{-9}$	PacketTimeSkewFromMode	$6.50e^{-5}$
26	PacketLengthSkewFromMedian	$1.87e^{-3}$	PacketLengthMedian	$5.99e^{-5}$
27	FlowSentRate	0.51	ResponseTimeTimeStandardDeviation	0.01
28	ResponseTimeTimeVariance	0.55	ResponseTimeTimeVariance	0.03
29	PacketTimeSkewFromMode	0.64	PacketLengthSkewFromMode	0.99

Figure 3.1: Features listed by Importance, Layer 1 on the Left, Layer 2 on the Right [42]

3.3.6 Three-Layered Approach

[43] developed a system that is able to identify the DNS tunnel tool that is causing the malicious DNS traffic. The basis of this system is shown in the previous works [40], [41], and [42], where the first step is the separation of DoH traffic and normal DNS traffic, and the second step is to detect malicious DoH traffic. [43] added a third step to that system which is the identification of the malicious DNS tunnel tool, i.e. dns2tcp, dnscat2, and iodine (see Figure 3.2). They used the data-set of [36] for their evaluation where they extracted 28 statistical traffic features and chose the Machine Learning models XGBoost, Light Gradient Boost Machine, and CatBoost to test for the best performing model in each step. The comparison of the models in each of the three phases resulted in the selection of XGBoost for the first phase, Light Gradient Boost for the second Layer and CatBoost for the third Layer due to their performance and accuracy in the respective phase. Moreover, they found that *Mode Packet Length* and *Mean Packet Time* were the most important features for Layer one, *Mode Packet Length* and *Median Packet Length* were the most important features for Layer two. In the third Layer, the most important features followed the pattern *Request/response time difference*, and according to that pattern the most important features were *Median Request/response time difference*, *Skew from median Request/response time difference*, *Skew from median Request/response time difference*, and *Skew from mode Request/response time difference*. XGBoost in the first Layer had the accuracy of 99.81%, the precision of 99.81%, and the F-score of 99.87%, CatBoost in the second Layer had the accuracy of 99.99%, the precision of 100%, and the F-score of 99.99%, and Light Gradient Boost Machine in the third Layer had the accuracy of 97.22%, the precision of 94.97%, and the F-score of 95.19%.

3.4 Deep Learning Based Approach

[44] used the Deep Learning algorithm Bi-Directional Recurrent Neural Network (BRNN) to detect malicious DNS queries that are sent using encrypted tunnels. They used the data-set of [36] for their evaluation, which they pre-processed by deleting redundant or zero rows, then they normalized all the values between a specific range, labelled the values into malicious and benign, and split them into a training and a testing set with the ratio of 80% to 20%. From the data set they extracted 34 features, but they used only statistical features, therefore they deleted the 6 non-statistical features. In the training phase, they trained their model to identify benign and malicious DoH query patterns, and in the following testing phase they tested if the model was able to identify the trained patterns. They achieved an accuracy of 100% with no false positives and false negatives rates and concluded that their model is an effective way for organizations to monitor and protect their environment.

Reference	Key Dimensions
[33]	Downgrading Attacks on DoH connections
[34]	Caption of DoH traffic
[35]	Separation of DoH traffic from DNS traffic using Machine Learning Models Feature analysis
[36]	Feature extraction Creation of a DoH data-set
[40]	Tests of MLmodels to detect malicious traffic
[41]	Two layered approach for detection of malicious DoH traffic Tests of ML models
[42]	Improvement of [41]'s work Statistical tests for feature analysis Sequential Forward Selection for amount of feature Optimization
[43]	Three layered approach to find tunnel tool causing malicious DoH traffic
[44]	Use Deep Learning models to detect malicious DoH traffic

Table 3.7: Summary of the References presented in this Chapter

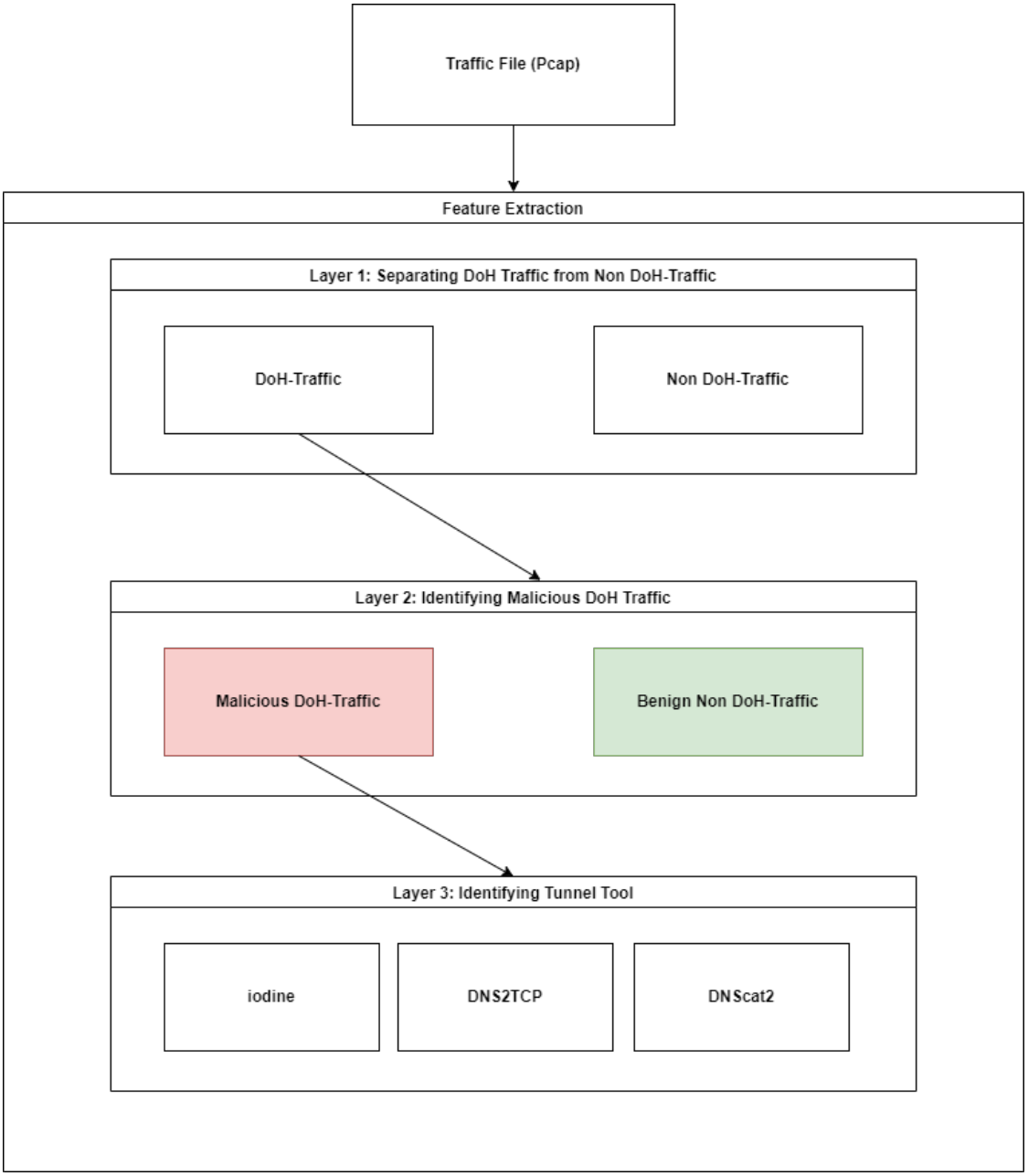


Figure 3.2: Illustration of the Three-Layered Approach

Chapter 4

Design

In the following Section, the design decisions for this thesis are presented. The literature research revealed different interesting and promising approaches, but the most convincing and matured approach in the survey findings was the one by [41] and [42]. The layer approach, where in the first Layer the DoH traffic is detected and in the second Layer the malicious traffic is detected, is a very convincing and inspiring work. Additionally, the feature analysis and the proposal of the best performing ML model of [42] is highly accurate. In terms of the feature selection, [42]'s work suited very well to the work of all the references found in the previous chapter. Therefore, this work is strongly based on their work.

For the detection of malicious DoH traffic, the two layered approach that [41] found will be approached, where the first Layer separates DoH traffic from classic DNS traffic and the second Layer detects malicious traffic and separates it from benign traffic. According to the findings of [42], the Machine Learning model that will do the detection will be the Light Gradient Boosting Machine, and for each Layer, the important features according to the results of the statistical tests will be involved. Finally, the data for the default data-sets will be taken from [36], whereas it will be built according to the work of [41] with two data-sets for each Layer and with 40'000 data points for each data-set.

The mentioned works will not be adopted exactly, since all those works were written in Python. SecGrid is mainly written in JavaScript, therefore the feature extraction will be implemented also in JavaScript to assure the compatibility. For the PCAP-file analysis, node-pcap implemented in JavaScript will be used instead of Scapy (Python), which possibly needs to be adjusted to be able to extract all the desired features.

The outline of this Chapter is as follows: first the data-set [39], which provides the training data for this thesis, is presented. The next two Sections describe how the traffic is filtered for HTTPS and how the TCP-flows are formed into clumps. The Section Feature Extraction presents all the features that are extracted from the TCP-clumps including their computations. In the next Section it is shown how the training data-sets are composed. The Section Light Gradient Boosting Machine presents the ML model that is used in this thesis and finally, the Section Architecture shows how all the presented parts are composed such that they result in a working prototype.

4.1 Data-Set

For this thesis, the data-set created by [36] and found at [39] is used. The data-set was created by using HTTPS traffic flows with two levels of distinct labels. The first level is assembled by normal HTTPS traffic and tunneled DoH traffic, the second level consists of benign DoH traffic and malicious DoH traffic. Figure 4.1 shows the illustration of how the data was collected. The traffic was collected by capturing HTTPS traffic from the web browsers *Google Chrome* and *Mozilla Firefox*. Using the browsers, he visited a set of the top 10'000 websites of *Alexa* to collect non-DoH-traffic data. Benign DoH-traffic data was collected by configuring both browsers to only use DoH connections instead of DNS traffic. To collect malicious DoH-traffic data, [36] deployed a network that simulates malicious DoH tunneling scenarios.

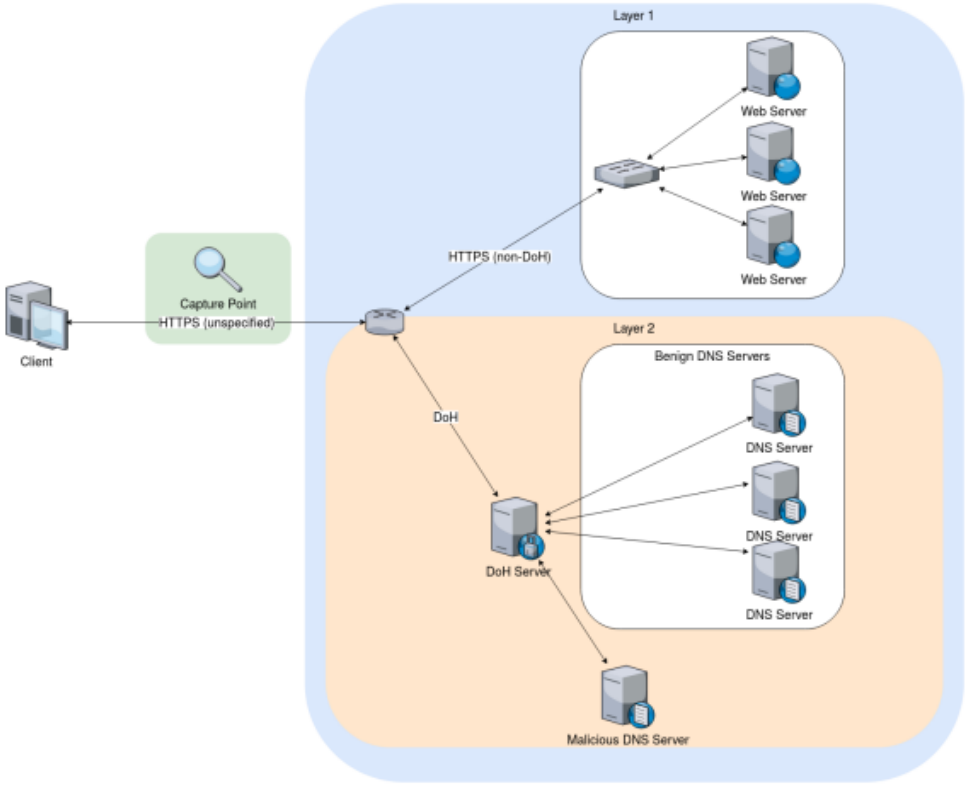


Figure 4.1: Illustration of the Collection of the Data by [36]

After analyzing some PCAP-files contained in this data-set in Wireshark, a surprising observation is made. Wireshark provides the function to filter one single TCP-flow. With this function, it becomes visible that many TCP-flows are not ended correctly with the double FIN-ACK packet sequence like it was shown in Chapter 2.3 and were just interrupted. Also, checking the successor PCAP-file does not show any end of TCP-flows of the predecessor file. But as [20] stated, it can be totally normal in the real world that TCP connections can be interrupted abruptly. This finding makes the approach of malicious DoH traffic detection of this thesis even more reliable and accurate and appropriate for a real world problem. As a consequence, each TCP-flow in the data-set that is not ending will be ended manually at the end of each PCAP file and a flag will be handed over to the feature collection. Since this

problem is not handled in any of the references, it can be a limitation of all these works that is tried to be solved in this thesis.

4.2 Clumping

Before starting a complete analysis, some data can already be filtered. As mentioned in Section 2.4, DoH works only on the HTTPS protocol, which only operates on port 443. Therefore, only flows whose either source port or destination port is port 433 will be considered further, every other flow which has nothing to do with the HTTPS port will be dropped.

[36] had the idea of a clumping process in which he merged packets of a same flow. To ensure that only packets of one flow are in this clump, he uses a timeout which limits the time interval of the packets, i.e. each packet whose timestamp exceeds the timeout is automatically put into the next clump. In this thesis, another clumping approach is followed (Figure 4.2). A DoH connection is always one single TCP flow, therefore one clump will be built with one entire TCP flow. The advantage of such a clumping process is that not every single packet data sequence has to be forwarded, but only a summary of the whole TCP flow, which saves a lot of memory capacity. Another advantage of this clumping process is that it allows to extract statistical features, e.g. the median of the packet times in the clump etc. This process will be declared in the next Section.

In the Section 4.1, the problem of the non-ending flows is discussed. It is also pointed out that those flows are considered, too. The pragmatic solution for this is that if the PCAP-file is over and the flow is still open, the flow is closed manually, and the flag *Status* will be passed as *Open*. For the flows which are closed correctly, the flag *Status* will be passed as *Closed*.

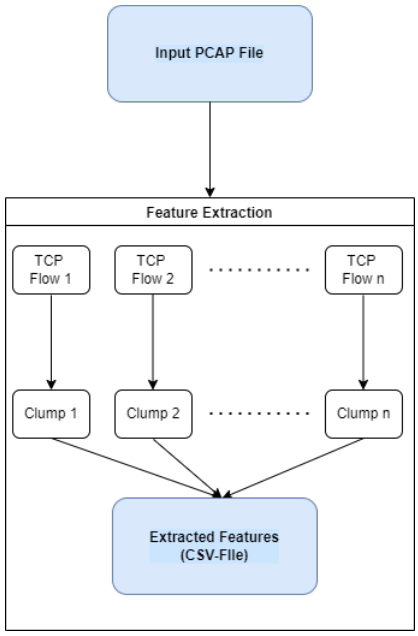


Figure 4.2: Illustration of the Clumping Process

4.3 Feature Extraction

The feature extraction is done using the data-set [39], whereas the extracted features rely on strongly on the features [36] computed and as already mentioned in Section 4.2, except that one clump consists of one TCP flow, from which all the features are extracted and computed. Statistical features are all computed with the same eight metrics, presented in Section 4.3.1. Header features of every flow are extracted according to the features presented in Section 4.3.2. The core features, which need to be computed, are presented in Sections 4.3.3, 4.3.4, and 4.3.5. Finally, the new-found features in this thesis are presented in Section 4.3.6. Totally, 41 features are presented in this Section.

4.3.1 Statistical Metrics

The statistical metrics used for the features in this work repeat for every property of a TCP-flow (*packet length*, *packet time*, and *packet request/ response time*), therefore they are presented in this separate Section. All the metrics are summarized in Table 4.1. The following metrics all consider the data-set $x_1, x_2, \cdots, x_n$, where x_1 is the first entry of the data-set and x_n is the last entry of the data-set. n is considered the highest index in the data-set. If the equation requires modifications on the data-set, they are mentioned separately in the respective Section.

Mean

The mean [45] $\bar{x}$ (Equation 4.1) or also the *sample mean* indicates the center of a data-set, or in other words the arithmetic average of a sample.

$$\bar{x} = \frac{\sum_{i=1}^{N}}{x_i} = \frac{x_1 + x_2 + \cdots + x_n}{n} \tag{4.1}$$

Median

The median [45] m (Equations 4.2 and 4.3) indicates the middle of data-set, but unlike the mean it is not affected by extreme values. It is defined as follows:

Consider the ordered data-set $x_1 + x_2 + \cdots + x_n$ starting from the smallest x_1 value and ending at the biggest value x_n.

If n is odd, then the median is the middle value of the ordered data-set.

$$m = x_{\frac{n}{2}+0.5} \tag{4.2}$$

If n is even, then the median is the average of the two middle values of the ordered data-set:

$$m = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} \tag{4.3}$$

Mode

The *mode* [45] is indicating the value that occurs the most in the data-set and is therefore another indicator for the central tendencies of the data-set. If there is no unique value that occurs the most, then the sequence of all values that occur the most indicate the *modal values*.

Variance

The variance s^2 [45] (Equation 4.4) is the "average" of the summed squared differences between each value of the data-set and the mean of the data-set. It is not quiet the average, since the sum is not divided by n but rather by $n-1$. The variance considers the spread tendencies of the data-set.

$$s^2 = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{n-1} \tag{4.4}$$

Standard Deviation

The standard deviation s [45] (Equation 4.5) is an indicator for the spread of the data-set. It is the positive square root of the variance.

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{n-1}} \tag{4.5}$$

Coefficient of Variation

The coefficient CV [46] (equation 4.6) of variation is used to compare the two different metrics, standard deviation and mean. It is computed by dividing the standard deviation by the mean and multiplying it by 100.

$$CV = s/\bar{x} * 100 \tag{4.6}$$

Skew from Mode

The Skew from Mode or also Pearson's first coefficient of skewness S_{kp1} [47] (equation 4.7) is used to compare the symmetry of two or more distributions of data-sets. It is computed by subtracting the *mode* from the mean m and dividing the difference by the standard deviation s.

$$S_{kp1} = \frac{\bar{x} - mode}{s} \tag{4.7}$$

Skew from Median

The Skew from Median or also Pearson's second coefficient of skewness S_{kp2} [47] (Equation 4.8) is used to compare the symmetry of two or more distributions of data-sets. It is computed by three times the difference between the mean and the median divided by the standard deviation.

$$S_{kp2} = \frac{3 * (\bar{x} - m)}{s} \tag{4.8}$$

No.	Metric	Format of Feature
1	Mean	*float*
2	Median	*float*
3	Mode	*int*
4	Variance	*float*
5	Standard Deviation	*float*
6	Coefficient of Variation	*float*
7	Skew from Mode	*float*
8	Skew from Median	*float*

Table 4.1: Statistical Metrics

4.3.2 Header Features

The identifying header features are the features *Source IP*, which is the IP of the client, *Destination IP*, which is the IP of the server, *Source Port*, which is the port of the client, and *Destination Port*, which is the port of the server. All the four features have the format *string*. Since these features are only passed to the set of extracted features for informational purposes and are not further used, the format *string* is appropriate. The feature *Duration* indicates the total period the connection lasted and has the format *float*. The two features *Flow Bytes Sent* and *Flow Bytes Received* sum up the total number of Bytes sent from the client to the server, or received from the client and sent by the server, respectively. Both the features have format *int*. The *Flow Sent Rate* (Equation 4.9) is computed by dividing the total number of sent Bytes by the client by the duration it lasted until the next packet was sent. Accordingly, the *Flow Received Rate* (Equation 4.10) is computed by dividing the total number of received Bytes by the client by the duration it lasted until the next packet was received. Both the features have the format *float*. Table 4.2 sums up all the features presented in this Section.

$$Flow\ Sent\ Rate = \frac{Total\ Number\ of\ sent\ Bytes}{Time\ last\ sent\ Packet - Time\ first\ sent\ Packet} \tag{4.9}$$

$$Flow\ Received\ Rate = \frac{Total\ Number\ of\ received\ Bytes}{Time\ last\ received\ Packet - Time\ first\ received\ Packet} \tag{4.10}$$

No.	Feature	Format
1	Source IP	*string*
2	Destination IP	*string*
3	Source Port	*string*
4	Destination Port	*string*
5	Duration	*float*
6	Flow Bytes Sent	*int*
7	Flow Bytes Received	*int*
8	Flow Sent Rate	*float*
9	Flow Received Rate	*float*

Table 4.2: Header Features

4.3.3 Packet Length Features

Packet length features are computed from the set of all extracted sizes (in Bytes) of every packet in one clump and by using the metrics presented in Section 4.3.1. Table 4.3 summarizes all eight packet length features.

No.	Feature
1	Packet Length Mean
2	Packet Length Median
3	Packet Length Mode
4	Packet Length Variance
5	Packet Length Standard Deviation
6	Packet Length Coefficient of Variance
7	Packet Length Skew from Median
8	Packet Length Skew from Mode

Table 4.3: Packet Length Features

4.3.4 Packet Time Features

Packet time features are computed from the set of all extracted durations (in seconds) between the first packet was sent until the actual packet is sent in one clump and by using the metrics presented in Section 4.3.1. Table 4.4 summarizes all eight packet time features.

No.	Feature
1	Packet Time Mean
2	Packet Time Median
3	Packet Time Mode
4	Packet Time Variance
5	Packet Time Standard Deviation
6	Packet Time Coefficient of Variance
7	Packet Time Skew from Median
8	Packet Time Skew from Mode

Table 4.4: Packet Time Features

4.3.5 Packet Request/ Response Time Features

For packet request/ response features, first the difference of sending a packet from the client until the answer of the server was received (in seconds) is computed. From this difference, the metrics presented in Section 4.3.1 are computed. Table 4.5 summarizes all eight packet request/ response features.

No.	Feature
1	Packet Request/ Response Time Mean
2	Packet Request/ Response Time Median
3	Packet Request/ Response Time Mode
4	Packet Request/ Response Time Variance
5	Packet Request/ Response Time Standard Deviation
6	Packet Request/ Response Time Coefficient of Variance
7	Packet Request/ Response Time Skew from Median
8	Packet Request/ Response Time Skew from Mode

Table 4.5: Packet Request/ Response Time Features

4.3.6 Novel Features

In this thesis, new additional features to the features [36] developed are introduced. The *State* is a flag that indicates if a TCP session was closed or not. As seen in chapter 2.3, the initiator of the closure of a TCP session could either be the client or the server. If it is *closed* it means that the TCP session was closed and so the flag receives the value 1. If it is *open* it means that the TCP session was not closed correctly and so the flag receives the value 0. The values 0 and 1 in format *int* are chosen to be numerically, since the ML model is not able to process string-values. The feature *Number of Application Packets Sent* indicates the total number of packets sent by the client in the TCP flow, the feature *Number of Application Packers Received* correspondingly indicates the number of packets received by the client in the TCP flow. Both the features have the format *int*.

The feature *Number of ACKs Sent* sums up the total number of ACK packets sent by the client in the TCP flow, the feature *Number of ACKs Received* accordingly is the summed up total number of ACK packets received by the client in this flow. Both the features have the format *int*. The feature *Number of Retransmits Sent* sums up the total number of retransmission-packets sent by the client in the TCP flow, the feature *Number of Retransmits Received* sums up the total number of retransmission-packets received by the client in the TCP flow. Finally, the feature *Total Packet length* indicates the summed up length of all packets sent and received by the client in one flow. Table 4.6 sums up all the features introduced in this Section.

No.	Feature	Format
1	State	*int*
2	Number of Application Packets Sent	*int*
3	Number of Application Packets Received	*int*
4	Number of ACKs Sent	*int*
5	Number of ACKs Received	*int*
6	Number of Retransmits Sent	*int*
7	Number of Retransmits Received	*int*
8	Total Packet Length	*int*

Table 4.6: New Features compared to the Work of [36]

4.4 Training Data-Sets

Since the ML model is trained two times (Layer 1 and Layer 2), it is also necessary to have two different training data-sets. Layer 1 conducts the classification into non-DoH and DoH traffic, therefore the data contained in the training set for Layer 1 (TSL1) contains an equal amount of DoH traffic and non-DoH traffic, whereas the DoH traffic is split equal into benign and malicious traffic. [41] composed his training data-sets with 40'000 data points, thus the total amount of data contained in TSL1 is also 40'000 data points. The traffic data is marked with a flag called *doh* for indicating the ML model if the data is DoH traffic or non-DoH traffic. If the flag has the value 0, this means that the traffic is non-DoH traffic. If the flag has the value 1, this means that the traffic is DoH traffic. TSL1 has its features ordered according to [42] in Figure 3.1 on the left side, without the red marked features, but with additionally the new features introduced in this thesis appended at the end.

Layer 2 conducts the classification into benign traffic and malicious traffic, therefore the training set for Layer 2 (TSL2) contains equally split traffic into benign and malicious traffic. The malicious traffic is equally split into three different parts: malicious traffic from the tunnel tools *iodine*, *DNS2TCP*, and *DNScat2*. The total amount of data points in TSL2 is again 40'000 data points. The traffic is marked with a flag called *malicious*. If the flag has the value 0, this means that the traffic is benign. If the flag has the value 1, this means that the traffic is malicious. TSL2 has its features ordered according to [42] in Figure 3.1 on the right side, without the red marked features, but with additionally the new features introduced in this thesis appended at the end.

4.5 Light Gradient Boosting Machine

Light Gradient Boosting Machine (LGBM) [48] is a subtype of Gradient Boosting Decision Tree (GBDT). It is fast, efficient and has an excellent scalability when the number of features is high and the amount of data is large. To this end, it uses *Gradient-based One-Side-Sampling* (GOSS) and *Exclusive Feature Bundling* (EFB). GOSS introduces a native gradient weight handling, which means that instances which were not trained as successful as other instances contribute more to the information gain than the well-trained instances during the training process. EFB clusters mutually exclusive features into one single feature to reduce the amount of features. Mutually exclusive features are two or more features that never take non-zero values when at the same time another feature takes a non-zero value [48]. In this thesis, LGBM is used for both Layers of the malicious DoH traffic detection process due to its promising characteristics.

4.6 Architecture

After all design decisions are completed, it is now time to show how all the previously introduced parts work together in the SecGrid prototype for malicious DoH traffic detection. Figure 4.3 shows the architecture of SecGrid. The two parts which have to be adjusted are the Protocol Parser and the Feature Extractors, which are marked green in the Figure. The whole pipeline will be implemented there, all the other parts will not be affected. The Protocol Parser needs to be adjusted, since currently it only parses entire PCAP-files, but the approach of this thesis demands it to parse TCP-flows. Therefore, a whole new TCP-flow handling will be implemented.

The Feature Extractors will be adjusted according to Figure 4.4, which shows the whole pipeline of the new process. The PCAP-file to be analyzed is given as an input to SecGrid, which first will parse the file into the individual TCP-flows. The Feature Extraction extracts all the demanded features (see Section 4.3) and assigns them into the respective clump. The clumps in turn will then be written into a CSV-file. As soon as every flow/ clump is extracted from the input file, the finished CSV-file with all the feature information is then handed over to the Classification. The Classification is split into two parts (Layers): the first part Separates DoH traffic from non-DoH traffic and hands over only the DoH traffic clumps to the second part. In the second part, the DoH traffic is analyzed and malicious DoH traffic is identified. The output of SecGrid is then a file with classified DoH traffic, whereas the user will be noticed if there is malicious traffic recorded in the input PCAP-file.

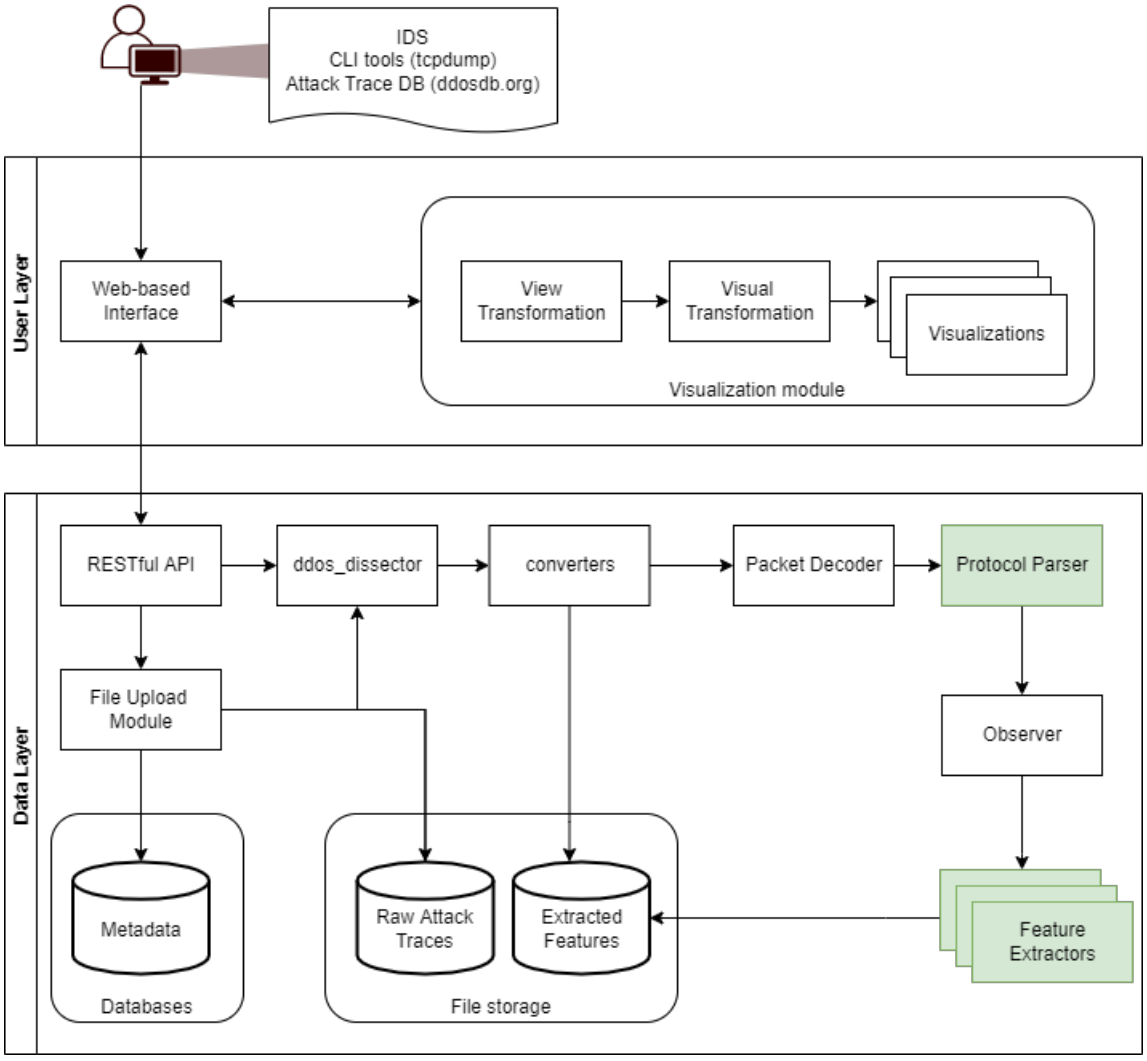


Figure 4.3: Components to be adapted

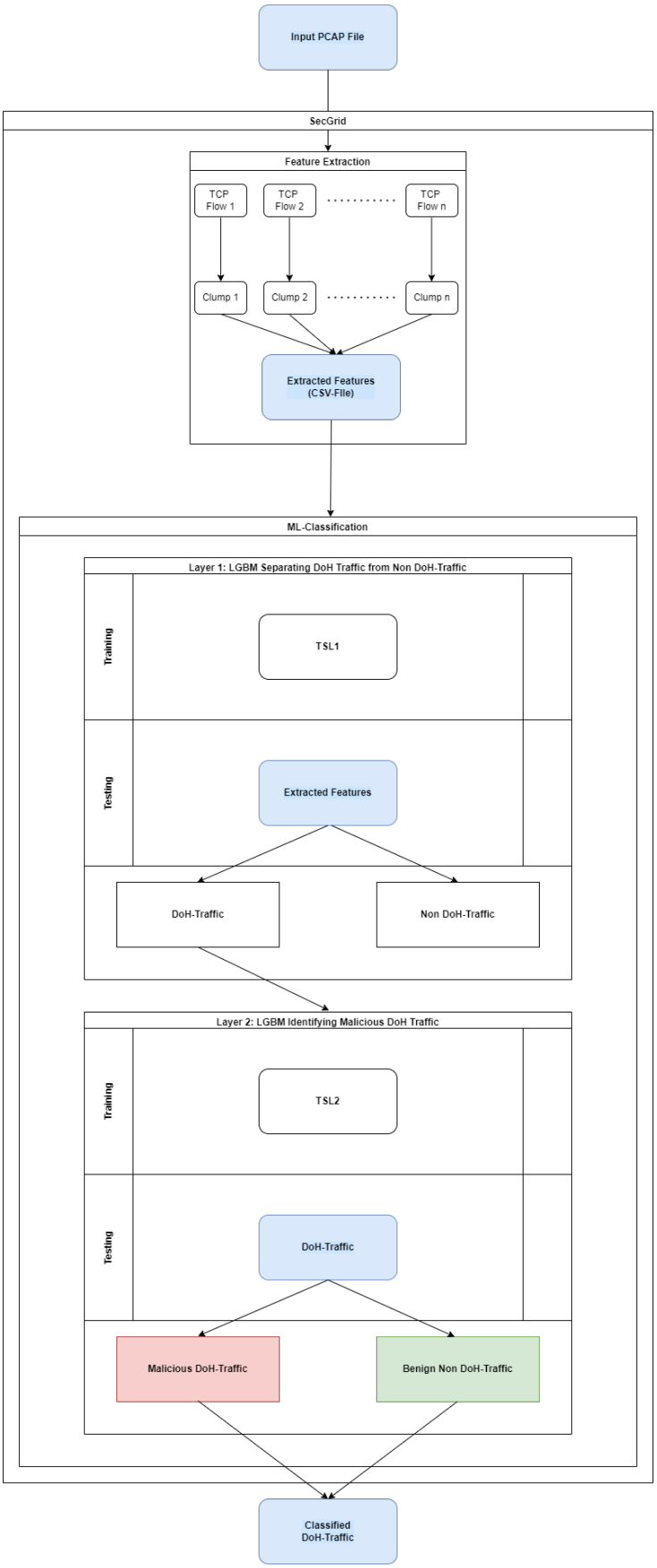


Figure 4.4: Illustration of the whole Classification Process of an Input PCAP File

Chapter 5

Implementation

This Chapter describes step by step how the prototype for malicious DoH traffic detection is implemented into SecGrid. First, Section 5.1 shows an overview where the whole pipeline of the newly implemented prototype is depicted. Section 5.2 describes the clumping process. In Section 5.3, the feature extraction is depicted, whereas the computation of the statistical metrics and every feature type (header features, packet information features, and the new features) is exemplified in the respective Subsection, inclusively how they are saved. Section 5.4 presents the training data-sets that are later used by the ML models. First the preprocessing procedure is described, followed by the compilation of the training data-sets. Finally, in section 5.5 the centerpiece of this thesis is presented: two LGBM algorithms which are trained to detect malicious DoH traffic within a two layered system. After this Section, the hyperparameter tuning of the two ML models is presented.

5.1 Pipeline

The implementation of every component results in a fully working prototype for the detection of malicious DoH traffic. Figure 5.1 shows the illustrated pipeline of this prototype. It takes a PCAP-file as input, which is handled like every other file in SecGrid according to Figure 4.6, i.e. it is forwarded by the RESTful-API, runs through the *ddos_dissector*, the *converters*, and the *Packet Decoder* until it arrives in the *Protocol Parser*. The *Protocol Parser* hands over every single flow to the *TCP Tracker*, which extracts the packet information and collects the flow information and gives it back to the *Protocol Parser*. The protocol parser hands over every single flow to the *Feature Extraction*, which computes all the feature values of the clumped flow, which is then saved into a CSV-file. This procedure is repeated until every flow contained in the input PCAP-file is analyzed, clumped and saved in the CSV-file. The CSV-file that contains all the clumps with the respective feature-values is then forwarded to the two-layered malicious DoH detection component. In Layer 1, the clumps are separated into DoH traffic and non-DoH traffic, whereas only the DoH traffic is forwarded to the second Layer. The second Layer checks if the DoH traffic is malicious or benign. If malicious traffic is found, it is saved into a CSV-file and forwarded to the *File Storage* of SecGrid.

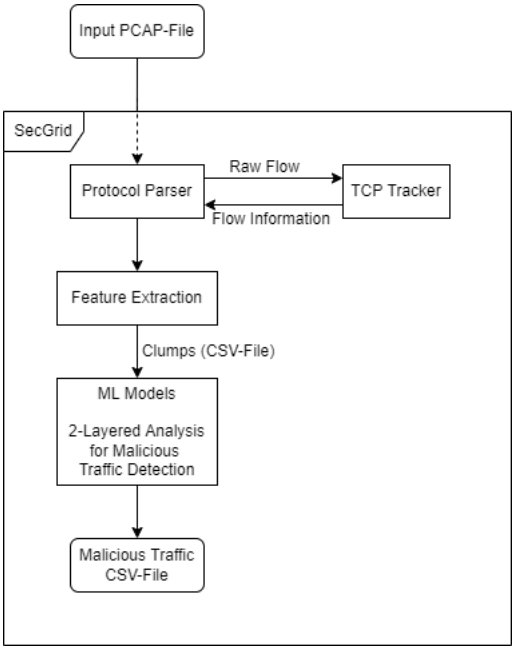


Figure 5.1: Pipeline of the Data-Flow

5.2 Clumping

For the clumping process, as introduced in Section 4.2, the TCP-flows of the input files are separated and each flow builds one clump. Therefore, the parser of SecGrid in the file *Pcap-Parser.js* and the *tcp_tracker* of *node-pcap* need to be adjusted, since the *PcapParser.js* does not handle TCP flows so far and the *tcp_tracker* does not handle the packet size information in a useful way for this work, and it does also not handle non-closing TCP flows. *Node-pcap* is forked locally into the project as a folder such that it can be modified, before it was included as a dependency.

So far, SecGrid used only the function of inspecting PCAP packets. *node-pcap* is able to inspect complete TCP flows. Therefore, the parser is modified such that it collects every single TCP flow and as soon as the flow has ended, the parser emits the flow to the feature extraction, thus the ending flows are analyzed. Each flow has a property called *state*, which is set to *CLOSED* after the flow was closed by the *tcp_tracker* of *node-pcap*. However, in a next step the non-ending flows have to be handled, since *node-pcap* is able to handle packets with non ending flows, but they are not terminated and emitted to the feature extraction. Therefore, the parser collects every flow in a JS object and every time a flow is returned closed, this flow is removed from the object. When the last packet is inspected by the *tcp_tracker*, the remaining flows in this object are all the flows which are non-ending.

The parser goes through this list and calls the function *forceClose* in the *tcp_tracker*. The *tcp_tracker* has to be adjusted such that it has this function *forceClose*, which closes each open flow as soon as the last packet has passed through the parser and the flow is still open. The *state* is set to *OPEN* and the flow is emitted to the parser. After the parser called the *forceClose* function and emitted the respective flow to the feature extraction, the flow is removed from the object. This process is repeated until the object is empty, meaning that all

the flows of the PCAP file are handled and emitted to the feature extraction.

The handling of non-ending flows is important since also non-ending flows can be affected by malicious influence. Since the *tcp_tracker* did not have a handling of such flows, it is crucial to implement it in this work because otherwise they would not have been further analyzed. Adding a new *state* (*OPEN*) aims to make the analysis even more precise compared to the related work, since there the aim of the flow-handling is mainly laid to the extraction of statistical properties.

5.3 Feature Extraction

In this Section the Feature Extraction is presented in detail. It is included in the file *MachineLearningFeatureExtractionDoH.js* as a subclass of the superclass *AbstractPCAPAnalyser.js*. The flows handed over by the parser are analyzed here, whereas the flow is an object called "session", which contains the information of a single flow extracted by *node-pcap*. The features of the "session" object used for the feature extraction of SecGrid are shown in Table 5.1.

It is critical to know about the *tcp_tracker* of *node-pcap* that it does not deliver all the information of a TCP packet. It delivers the information of the IPv4 layer and the TCP layer of every packet in the flow, but it ignores the frame information of the packet, i.e. in the case of the data-set [39] the *Linux cooked-mode capture (SLL)*, which is a pseudo-protocol for capturing traffic with several devices at the same time [49]. Every packet contains this information, which has the size of 16 Bytes in most cases, in some very rare cases it can also be more. But this fact does not affect the quality of the data used negatively, since the data is a cleaner representation of TCP data without the SLL information. In that sense, the exclusion of such data improves the accuracy of the data.

Before extracting the features, a simple and short process is used to verify that only HTTPS traffic is analyzed, namely by checking if the source port or the destination port of the flow is port 433. With this procedure, redundant traffic is filtered and not used for further analysis, which saves a lot of time for the actual analysis of HTTPS traffic, since this process can be time-consuming enough depending on the size of the input PCAP-file to be analyzed.

Feature	**Format**
connect_time	*float* (Time since 1970)
close_time	*float* (Time since 1970)
src	*string*, "IP:Port", e.g. "192.168.20.111:44310"
dst	*string*, "IP:Port", e.g. "1.1.1.1:433"
send_bytes_ip	*int*
send_bytes_payload	*int*
send_bytes_tcp	*int*
recv_bytes_ip	*int*
recv_bytes_payload	*int*
revc_bytes_tcp	*int*
send_acks	*Object*, Object{"ACK No.":"sending time"}

recv_acks	*Object*, Object{"ACK No.":"receiving time"}
send_packets	*Object*, Object{["Seq. No. + Data Length"]:"sending time"}
recv_packets	*Object*, Object{["Seq. No. + Data Length"]:"receiving time"}
send_retrans	*Object*, Object{["Seq. No. + Data Length"]:"sending time"}
recv_retrans	*Object*, Object{["Seq. No. + Packet Length"]:"sending time"}
state	*String*
total_packet_length	*array*, ["Packet Length"]

Table 5.1: Features of the Session Object forwarded by the *tcp_tacker* and used for the Feature Extraction

5.3.1 Header Features

The header features are extracted according to Section 4.3.2. To extract the *Source IP* and the *Source Port*, the feature *src* of the "session" object are used. For those two types, the functions *getIP* and *getPort* are implemented. *getIP* returns the characters before the colon in *session.src* or *session.dst* to get the *Source IP* or the *Destination IP*, respectively. The function *getPort* returns the numbers following the colon in *session.src* or *session.dst* to get the *Source Port* or the *Destination Port*, respectively. The extraction of the features *Destination IP* and *Destination Port* is identical to the extraction of *Source IP* and the *Source Port*, except that instead of *src*, *dst* is used. The feature *Duration* is extracted by computing the delta of *close_time* and *connect_time* of the object "session".

To extract the feature *Flow Bytes Sent*, the features *send_bytes_ip*, *send_bytes_payload*, and send_bytes_tcp are summed. *send_bytes_ip* is the sum of IP information (the size of the payload and the header) of the IPv4 layer of every packet in the flow. *send_bytes_payload* is the sum of the payload of the TCP layer of every packet in the flow and send_bytes_tcp is the header information (the size of the header) of the TCP layer in every packet in the flow. The feature *Flow Bytes Received* is computed similar to the feature *Flow Bytes Sent*, except that the features *recv_bytes_ip*, *recv_bytes_payload*, and recv_bytes_tcp are summed.

The extraction of the features *Flow Sent Rate* and *Flow Received Rate* is more complex. They require to first extract all the times of outgoing and incoming packets. Therefore, all the values in the objects *send_acks* and *send_packets* are extracted and pushed into an array whenever a packet was sent. From this array, the maximum value and the minimum value are taken and the delta of those two values is computed and returned as *Duration Sent*. For this process, the function *getDurationSending* was implemented. In a second step, the actual rate is computed by dividing the feature *Flow Bytes Sent* which was beforehand computed by the *Duration Sent*. This process was implemented in the function *computeRate*. The feature *Flow Sent Rate* is computed similarly, except that the function *getDurationReceiving* uses the objects *recv_acks* and *recv_packets* and the function *computeRate* divides *Flow Received Rate* by *Duration Received*.

5.3.2 Statistical Metrics

The statistical features are computed according to the Section 4.3.1. Every metric (see Table 5.2) has its own function, and the functions are designed to compute the metric of an array (here called "packets"). Thus, the function for every metric has only to be implemented once and still the metrics for the *Packet Length*, the *Packet Time*, and the *Packet Response and Request Time* can be computed by using the metric functions. To avoid errors caused by empty arrays, each function checks first that the input array is not empty, and then it computes the metric. If the input array is empty, it returns 0 instead. The metrics *Mean*, *Median*, *Mode*, *Variance*, and *Standard Deviation* are all computed by using the respective function of the *mathjs API* [50]. The functions *Coefficient of Variation*, *computeSkewFromMode*, and *computeSkewFromMedian* are computed according to the Section 4.3.1 and are also using functions of *mathjs* where needed.

Metric	**Function_Name(Input_Array)**	**Output Format**
Mean	*computeMean(packets)*	*float*
Median	*computeMedian(packets)*	*float*
Mode	*computeMode(packets)*	*int*
Variance	*computeVariance(packets)*	*float*
Standard Deviation	*computeStandardDeviation(packets)*	*float*
Coefficient of Variation	*computeCoefficientOfVariation(packets)*	*float*
Skew from Mode	*computeSkewFromMode(packets)*	*float*
Skew from Median	*computeSkewFromMode(packets)*	*float*

Table 5.2: Function Names of the Statistical Metrics

5.3.3 Packet Information Features

The packet information features, which include the information of the packet size, packet time and the response- and request time, are computed using the metric computation functions presented in the previous Section 5.3.2. The *Input_Array* for the packet length features is the feature *total_packet_length* of the object "session". This feature is the output of a retroactively implemented process into *node-pcap*. This process is implemented in this thesis, since *node-pcap* does not offer a feature which contains the packet length of every packet. The output is an array that contains the summed features *send_bytes_ip*, *send_bytes_payload*, and send_bytes_tcp of every packet that was inspected by *node-pcap*. The input array of the packet time features is extracted by the function *getTotalTimes*, which pushes every timestamp contained in the objects *send_packets*, *recv_packets*, *send_acks*, and *recv_acks* into one array, sorts the array, subtracts the smallest timestamp from every timestamp in this array and returns the array.

The input array of the packet response- and request time features is computed with multiple functions. The functions *getTimesSent(session)* and *getTimesReceived(session)* both receive the session object as the input. *getTimesSent(session)* extracts the values of the features *send_packets* and *send_acks* of the "session" object, pushes them into an array and returns a sorted array with all the timestamps when packets or TCP segments that have an ACK flag

(ACKs) were sent. *getTimesReceived(session)* is similar, but it returns a sorted array with the timestamps when packets or ACKs were received by extracting them from the features *recv_packets* and *recv_acks* of the object "session". Finally, the differences are computed by the function *getRequRespDifference* which takes the two sorted arrays of timestamps of sent and received packets as input and computes the time differences between outgoing and incoming packet timestamps at the client's side according to Algorithm 1.

Algorithm 1 Compute the Difference between outgoing and incoming Timestamps

Require: $timesSentSorted$, $timesReceivedSorted$
Ensure: $timesSentSorted.length \neq 0$, $timesReceivedSorted.length \neq 0$
  $n \leftarrow timesSentSorted.length$
  $m \leftarrow timesSentReceived.length$
  $requestResponseDifference \leftarrow []$
  for $i = 0..n$ **do**
    for $j = 1..m$ **do**
      if $timesReceivedSorted[j] > timesSentSorted[i] \wedge$
       $timesReceivedSorted[j] < timesSentSorted[i+1]$ **then**
        $difference = timesReceivedSorted[j] - timesSentSorted[i]$
        $requestResponseDifference.next \leftarrow difference$

5.3.4 Novel Features

The novel features proposed in this thesis are extracted according to Section 4.3.6. The state is computed with the function *getState(session.state)*, which takes the session state as input and returns 1 if it is *CLOSED*, if it is *OPEN* it returns 0. The values 1 and 0 are chosen since the LGBM algorithm is not able to handle the string values *OPEN* and *CLOSED*. The *Number of Application Packets Sent* is the length of the object *send_packets* and the *Number of Application Packets Received* is the length of the object *recv_packets*. The extraction of the *Number of ACKs Sent* and *Number of ACKs Received* is similar, since those two values are computed by measuring the length of the object *send_acks* and *recv_acks*. Again, the extraction of *nr_retrans_sent* and *nr_retrans_reveived* is similar to the two latter pairs, it is the length of the objects *send_retrans* and *recv_retrans*, respectively. The *Total Packet Length* is computed with the function *getTotalPacketLength(session)*, which takes "session" as the input and sums up the values *send_bytes_ip*, *send_bytes_payload*, *send_bytes_tcp*, *recv_bytes_ip*, *recv_bytes_payload*, and *recv_bytes_tcp*.

5.3.5 Saving Process

The extracted features have to be persisted for the model training phase. This assignment is performed by the function *createNewFlowData(session)*, which takes the "session" object as an input. Inside this function, all the needed values of the features are computed and stored into an object *newPacketMiningData*, which is basically a list of all the features. This list is then returned by the function. The output of the function is then pushed into a global array

called *result*. After the analysis input PCAP file is finished, the *result* array is written into a CSV file, which is then stored in the project for further usage.

5.4 Training Data Sets

With the feature extraction phase implemented as described in 5.3, training data needs to be preprocessed as shown in this section. The two layered approach for detecting malicious traffic requires two data-sets with different content, which are presented here. Thus, the data must first be extracted from the data-set [39] and bad data-points have to be removed. This step is described in the Section 5.4.1. After the preprocessing process, the training data-sets (TSL1 and TSL2) can finally be generated, these processes are described in the Sections 5.4.2 and 5.4.3.

5.4.1 Preprocessing

In a first step, the data has to be extracted from the data-set. The data-set is separated into two different directories, one for benign DoH and non-DoH traffic files and another one for malicious DoH traffic files. The directory with benign DoH and non-DoH traffic files is split into the two directories *Chrome* and *Firefox* which are the browser with whom the traffic data has been generated by [36]. Both of these directories are again split into four different directories called *AdGuard*, *Cloudflare*, *Google*, and *Quad9* which are the DoH servers which have been addressed to generate the traffic data. The IP addresses of the DoH resolvers which were used are listed in Table 5.3.

DoH Resolver IP-Address	
1.1.1.1	8.8.4.4
8.8.8.8	9.9.9.9
9.9.9.10	9.9.9.11
176.103.130.131	176.103.130.130
149.112.112.10	149.112.112.112
104.16.248.249	104.16.249.249

Table 5.3: IP-Addresses of the DoH Servers used for the Data-Set [39]

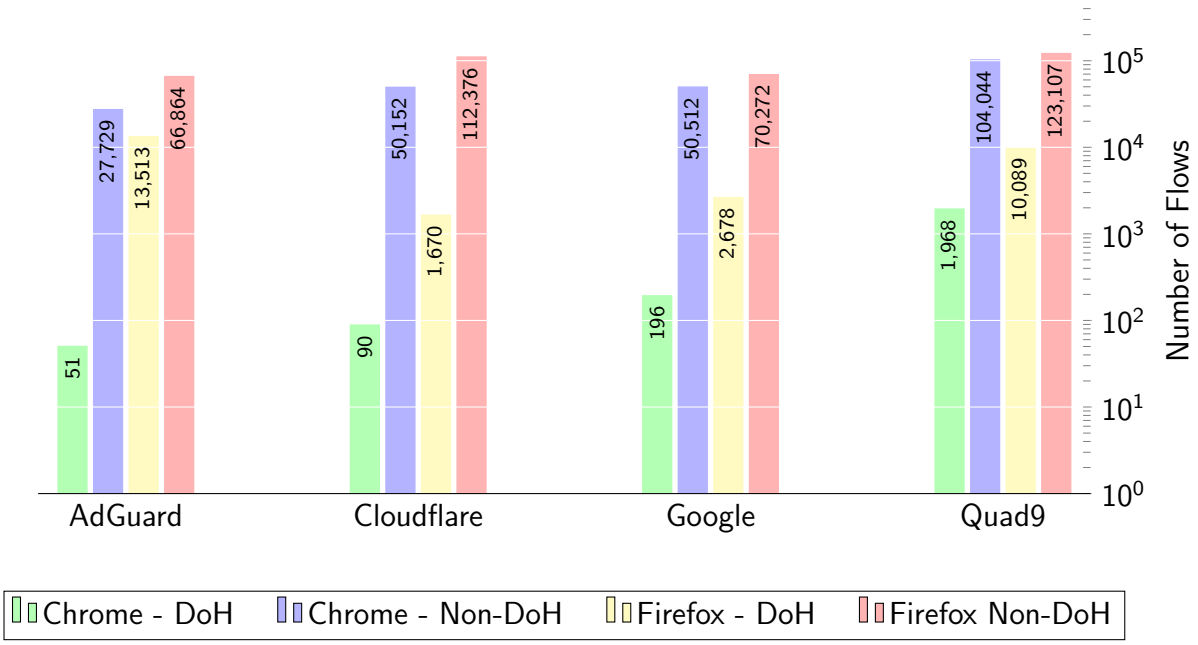
With this list, it is possible to distinguish between DoH data and non-DoH data in the data-set. In a further step, every folder is analyzed separately, and the data is saved in a separate CSV file. The features are ordered according to the order which was presented in Section 4.3. In addition to those features, two further features are added, one called *doh* which is set to 1 if the data is DoH traffic or 0 if it is non-DoH traffic, the other is called *malicious* and receives the value 1 if it is malicious traffic or 0 if it is benign traffic. Since every folder contains many files and not every file can be analyzed by handing them separately, the script *analyze.sh* is

written. This is a Unix-Shell script that basically steps through every file of the folder and executes the Feature Extraction process of SecGrid. Table 5.4 and Figure 5.2 show how many flows of each kind have been extracted from the respective directories after deleting the lines whose feature values are zero. These zero-lines are unusable and would only distort the ML model.

	AdGuard	**Cloudflare**	**Google**	**Quad9**
Chrome (DoH)	51	90	196	1968
Firefox (DoH)	13'513	1670	2678	10'089
Chrome (non-DoH)	27'729	50'152	50'512	104'044
Firefox (non-DoH)	66'864	112'376	70'272	123'107

Table 5.4: Number of total extracted benign DoH and Non-Doh Flows

Figure 5.2: Totally Extracted Flows (DoH and non-DoH)



The directory with malicious DoH traffic files is split into three different folders, *DNS2TCP*, *DNScat2*, and *iodine*, which are the tunnel tools which are used to generate the malicious traffic data. Since TSL1 will contain also some flows of malicious DoH data, the data in these directories must also be analyzed, the *doh* feature is set to 1. Table 5.5 indicates how many flows per tunnel tool were extracted, again after removing the lines with blank cells to avoid a distortion of the ML model.

For the second Layer, the benign and the malicious traffic data must be analyzed again. This time, the features will be ordered according to the order for the second layer in Section 5.4. The additional feature which is an indicator for benign or malicious traffic is this time called *malicious* and set to 1 if the traffic is malicious, if t is benign it is set to 0.

	DNS2TCP	DNScat2	iodine
No. of Flows	121'508	10'289	12'352

Table 5.5: Number of total extracted malicious DoH Flows

Since the analysis of the PCAP-files in the data-set in each case was exported to a CSV-file, the deletion of zero-values was feasible. Excel has the function to split text in columns, the splits are always made when a comma (",") was set. Then the columns *flow_sent_rate*, *packet_length_median*, *packet_time_median*, and *response_request_median* are filtered with the respective function in Excel. The column *flow_sent_rate* is chosen, since there it is an indicator for empty packet length arrays, the columns *flow_sent_rate*, *packet_length_median*, *packet_time_median*, and *response_request_median* are chosen since they are also indicators for empty arrays that were parsed. With this filter, it is possible to look for cells that contained zero values. The respective rows are deleted such that the finished table contains as few zero values as possible. The file is reset onto the format it had before by adding the columns together, separated with commas, such that it is possible to further process them.

5.4.2 Training Data Set of Layer 1

The composition of the Training Data Set of Layer 1 (TSL1) must be chosen in a way such that the ML model of Layer 1 is able to distinguish between DoH traffic and non-DoH traffic. In Section 5.4 the total amount of 40'00 data-points is determined. Therefore, the ratio of DoH traffic and non-DoH traffic in the training data-set is set to have an equal distribution of both types. Important here is that all the features that were computed are pushed and saved into TSL1.

		Non-DoH (Chrome)	Non-DoH (Firefox)	DoH (Chrome)	DoH (Forefox)
Benign	**AdGuard**	2500	2500	25	3500
	Cloudflare	2500	2500	45	800
	Google	2500	2500	90	1300
	Quad9	2500	2500	1000	3240
Malicious	**DNS2TCP**	0		4000	
	DNScat2	0		3000	
	iodine	0		3000	
Total		20'000		20'000	

Table 5.6: Distribution of the Data-Points in TSL1

For the non-DoH data, the equal amount of 2500 data-points from every file listed in Table 5.4 is taken. This makes a total amount of 20'000 data-points of non-DoH traffic. To make sure that the data-points are as various as possible, the data-points are chosen randomly with

the function *sample()* that can be used for dataframes and which is taken from the Pandas library [51] in Python. The data-points are then written into a CSV-file called *TSL1.csv*.

The composition of the DoH data part of TSL1 is more complex, since especially the DoH traffic that was collected with Chrome is very sparse. Therefore, the distribution of the DoH traffic is chosen to be one half (10'000 data-points) benign and the other half malicious DoH traffic. From each of then benign DoH traffic data-sets which were gathered using Chrome, half of the available data was used for TSL1, which results in 1160 data-points. The remaining 8840 data-points are taken from the benign DoH traffic data-sets which were gathered using Firefox. The part with the malicious DoH traffic is easier to compose, since there are enough data-points available. From the data-set that was generated using the *DNS2TCP* tunnel tool, the amount of chosen data-points is 4000, since it is the largest data-set. The amount chosen from the *DNScat2* and *iodine* data-sets was 3000 each. For both parts of the DoH traffic data, the data-points are chosen randomly, again using the function *sample()* of Pandas. Table 5.6 shows the distribution of the data-points that were chosen for *TSL1.csv*.

5.4.3 Training Data Set of Layer 2

The composition of the Training Data Set of Layer 2 (TSL2) must be chosen in a way such that the ML model of Layer 2 is able to distinguish between benign and malicious DoH traffic. The total amount of data is set to 40'000 data-points according to Section 5.4. The ratio of the two types of traffic is set to have an equal distribution between the two types. Since TSL1 also contains benign and malicious traffic data, it is made sure that different data-points are used. Important here is also that all the features that were computed are given into TSL2. TSL2 is saved into a CSV file called *TSL2.csv*.

The composition of the benign traffic data is complex, as already stated in Section 5.4.2, since the data is sparse, especially the traffic that was gathered using the Google Chrome browser. Therefore, the other half of the available data is chosen from each of the files, which results in 1060 data-points. The remaining 18'840 data-points are taken from the benign DoH traffic data-sets, which were gathered using Firefox. The composition of the malicious traffic is again easier due to the huge amount of available data. The amount of data gathered using the tunnel tool *DNS2TCP* is chosen to be 8000 data-points, since it contains the most data, the amounts of data gathered using the tunnel tools *DNScat2* and *iodine* are chosen to be 6000 data-points each. Again, it was made sure that the data is chosen randomly by using the function *sample()* of Pandas. Table 5.7 shows the distribution of the data-points that are chosen for *TSL2.csv*.

	Benign (Chrome)	Benign (Firefox)	Malicious
AdGuard	25	10'000	0
Cloudflare	45	800	0
Google	90	1300	0
Quad9	900	6840	0
DNS2TCP	0		8000
DNScat2	0		6000
iodine	0		6000
Total	20'000		20'000

Table 5.7: Distribution of the Data-Points in TSL2

5.5 ML Model

As introduced in Section 4.5, Light Gradient Boosting Machine is chosen as the ML algorithm for both Layers. The whole pipeline is written in Python, since *scikit-learn* [52] is easy to apply, and it is an easily accessible API for ML algorithms. The LGBM-model is imported from *ligthgbm* [53]. To handle the data-sets, the Pandas library [51] is used, the training data-set for Layer 1 is saved into a file called *TSL1.csv*, the training data-set for Layer 2 is saved into a file called *TSL2.csv*.

5.5.1 Layer 1

The model of the first layer is trained with TSL1. To have the correct order of the features (according to [42]), an array with the ordered feature names called *featureSequenceLayer1* is defined, since *TSL1.csv* does not have the correct feature-order yet. The first feature is *doh*, which indicates whether the data-point is DoH traffic or not, as defined in Section 5.4.1. The file *TSL1.csv* is read by using the function *read_csv* of Pandas and its content is saved in the dataframe *trainingDataset1*. Subsequently, *featureSequenceLayer1* is used to compose the data-set for Layer 1, i.e. every column of the feature that is contained in the sequence is added to the training dataframe called *tsl1*.

Next, the matrix *X1* which contains the values of all the features used to train the model, and the y1 vector which contains the labels to train the model are extracted and saved into the respective variable. Since LGBM is not able to handle float values, the next step is to convert the floats into integers, which is done using the *LabelEncoder* of the *sklearn* API. To make sure that LGBM is able to process every value, the flow values of X1 and y1 are transformed with the *LabelEncoder*. Then the LGBM Classifier is initiated and trained with X1 and y1. The hyperparameters which are used are acquired in a Grid Search in Section 5.5.3.

After the model is trained, it is now able to predict the input data from the PCAP file to be tested. The file which contains the extracted features of the PCAP-file to be analyzed is loaded into a new dataframe called *testingDataset1*. Again, the features need to be ordered

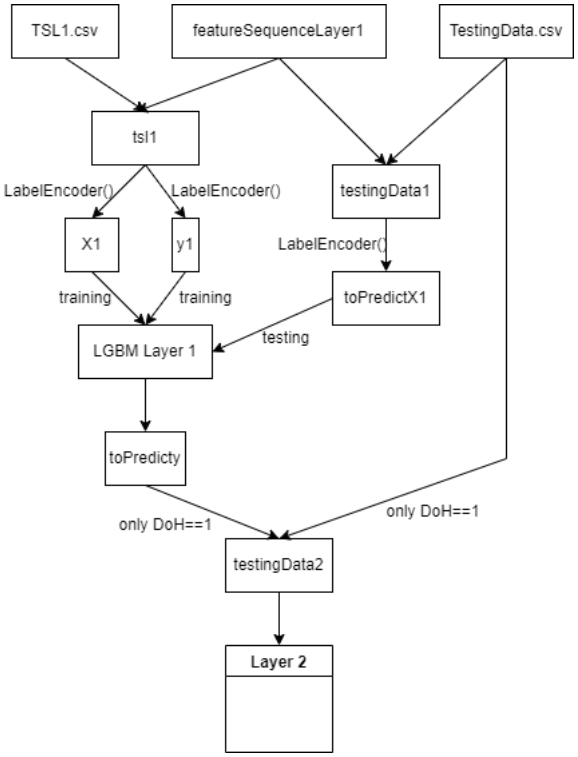


Figure 5.3: Illustration of the Data Flow of Layer 1

using *featureSequenceLayer1* to create the new dataframe *toPredict1*. After iterating over *testingDataset1* and extracting every feature-column needed, it is saved into the dataframe *toPredict1*. Now the matrix *toPredictX1*, containing all the feature values to be predicted, is extracted from the dataframe *toPredict1*. Again, the *LabelEncoder* ensures that the ML model is able to handle all the values contained in *toPredictX1*. Then the matrix *toPredictX1* is handed over to the LGBM model with the function *classifierLayer1.predict(toPredictX1)*, which predicts the input data and gives back the vector *toPredicty1*.

The resulting vector *toPredicty1* contains the predictions of every clump extracted from the input PCAP file. The value on each line is either 0 or 1, with 0 meaning that the clump is non-DoH traffic data and 1 meaning that the clump is DoH traffic data. The vector *toPredicty1* is then added to the dataframe *testingDataset1* by creating a new column called *DoH*. Subsequently, the data-points which have *DoH* equal to 1 are extracted from *testingDataset1* and saved into a new dataframe called *testingDataset2*, which is the clumps classified as DoH traffic data. Finally, *testingDataset2* is handed over to Layer 2, where it is further analyzed. Figure 5.3 illustrates the whole pipeline of Layer 1.

5.5.2 Layer 2

The model of the second Layer is trained with TSL2, with the features ordered according to [42] and the new features introduced in Section 4.3.6 appended directly afterwards. Before being able to start the training, the training data-set needs to be composed, since the input file *TSL2.csv* does not have the correct feature order yet. The feature sequence is defined

in the array *featureSequenceLayer2* with the first feature *malicious*, which indicates if the respective clump is malicious or benign traffic as defined in Section 5.4.1. The file is read by the function *read_csv()* of Pandas and saved into a pandas dataframe called *trainingDataset2*. Subsequently, an iteration through the array *featureSequenceLayer2* is conducted and every feature-name that is contained in this array is searched in the dataframe *trainingDataset2* and the respective column is saved into a new dataframe called *tsl2*.

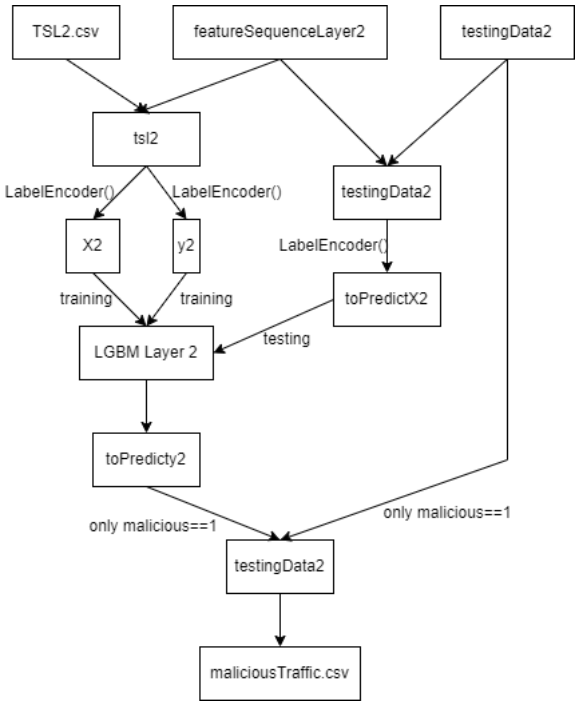


Figure 5.4: Illustration of the Data Flow of Layer 2

The next step is to extract the X2 matrix and the y2 vector. X2 is the matrix which contains the training values for the ML algorithm, and y2 is the vector which contains the labels which are needed to classify the data. After these two Objects are extracted, the values have to be transformed into integer values, since LGBM is not able to handle float values. This is done by using the *LabelEncoder*, according to the same procedure as in Layer 1. Finally, the data is prepared, and the ML model can be initiated and trained with X2 and y2. The hyperparameters which are used are acquired in a Grid Search in Section 5.5.3.

After the model is trained, it is ready to predict the input data, which was handed over to *testingDataset2* after the process of Layer 1. This data has already been separated from non-DoH traffic, thus the ML model of Layer 2 can now predict if the data is malicious or benign. Again, the data needs to be prepared by iterating over the array *featureSequenceLayer2*, looking up the feature in the dataframe *testingDataset2* and saving the respective column into the dataframe *toPredict*, but this time the feature *malicious* is left away. These values also need to be transformed into integer values by using the *LabelEncoder*, since the data-set contains many features which have the format float. Now the feature values are extracted from the dataframe *toPredict2* to construct the matrix *toPredictX2* which contains the feature values for the prediction. Then, the matrix *toPredictX2* is handed over to the LGBM model with the function *classifierLayer2.predict(toPredictX2)*, which predicts the input data and gives back the vector *toPredicty2* as the output.

This output vector *toPredicty2* contains the predictions of every data-point that was contained in the matrix *toPredictX2*. The value on each line is either 0 or 1, whereas 0 means that the TCP-flow is benign DoH traffic and 1 means that the TCP-flow is malicious traffic. *toPredicty2* is then added to the dataframe *testingDataset2* with the feature name *Malicious Traffic*. As a final step, the data-points that were predicted to be malicious are filtered from the *testingDataset2* checking if there is a data-point which has the value 1 in the feature-column *Malicious Traffic*. If there is a data-point that is malicious, it is extracted and stored in a CSV-file called *maliciousTraffic.csv*. Figure 5.4 illustrates the whole pipeline of Layer 2.

5.5.3 Hyperparameter Tuning

According to the documentation of LGBM [53], the algorithm has three hyperparameters that are the most important to obtain good results and to avoid over-fitting of the model. Those three hyperparameters are *num_leaves*, *min_child_samples*, and *max_depth*. Since the LGBM uses leaf-wise tree growth (see Figure 5.5) *max_depth* is the parameter to limit the tree depth and is set to -1 by default. *num_leaves* is the parameter that indicates how many leaves the model shall have and which controls how complex the model shall be. The number of leaves can be set to $num\_leaves = 2^{max\_depth}$ to have a symmetric tree model, but this should not be done because there is the risk of overfitting the model. Therefore, the parameter *num_leaves* is chosen deliberately smaller than 2^{max_depth}. By default, it is set to 31 by *scikit-learn*. Finally, the parameter *min_child_samples* indicates the number of data that is contained in a tree and depends on *num_leaves*. Small numbers can cause overfitting, large numbers can cause under-fitting. [53] states that setting this value to hundreds or thousands is enough, by default it is set to 20.

Furthermore, they mention the parameter *max_bin*, which indicates the maximum number of bins where the features will be bucketed, this means that LGBM packs continuous feature values into discrete bins. *max_bin* can be set between $0 < max\_bin < 255$. They recommend trying the *boosting_type*, which is the way the model is boosted, set to *dart*, by default it is set to *gbdt*. Ultimately, the *learning_rate* which controls the number of trees that are grown in the training phase of the model. It is set to 0.1 b default.

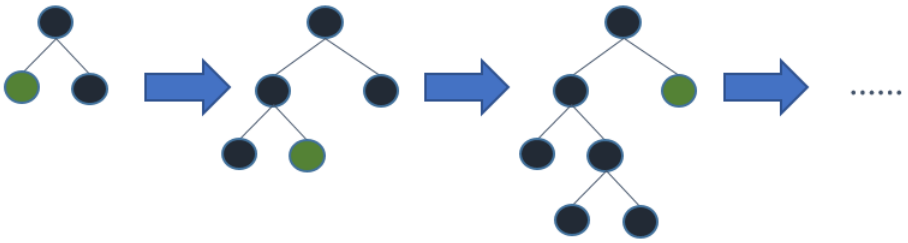


Figure 5.5: Leaf-Wise Tree Growth [53]

The setup for the Grid Search is as follows. The data is prepared like described in the Sections 5.5.1 and 5.5.2 and each time saved in the training matrix X containing the feature values and the training vector y containing the label values. Then, an array named *tunedParameters* containing the hyperparameters to be tuned is implemented. Finally, the LGBM Classifier Algorithm is initiated together with the Grid Search, where the ML model and the array

tunedParameters are handed in as input. After the Grid Search, the model is trained with each combination of hyperparameters to look for the best tuned parameters. The hyperparameters are tuned in small batches to avoid long training times. The Following sections present the values that are used for the hyperparameter optimization. After they are found, the hyperparameters are adopted to the respective ML model.

Layer 1

The following snippet shows all the values that are tested with the Grid Search. Since totally 20 Grid Search are conducted using at most three values per parameter and also using at most three parameters at once, it is disclaimed to show all the experiments separately. Instead, the list containing all the values that are used is shown.

```
tunedParameters = [
    {"max_depth": [6, 7, 8, 9],
     "num_leaves": [5, 8, 9, 10, 11, 12, 15, 20, 30, 40, 50, 100],
     "min_child_samples": [10, 30, 50, 70, 100, 130, 140, 145, 147,
     ↪  149, 150, 151, 152, 153, 155, 160, 170, 200, 250],
     "boosting_type": ["gbdt", "dart"],
     "max_bin": [200, 230, 240, 245, 250, 251, 252, 253, 254, 255],
     "learning rate": [0.05, 0.1, 0.15, 0.18, 0.19, 0.2, 0.21, 0.22,
     ↪  0.25]
     }
]
```

After all those values are tested, the maximum score of 96.81% is reached with the hyperparameters tuned seen in Table 5.8:

Hyperparameter	Value
max_depth	8
num_leaves	10
min_child_samples	151
boosting_type	"*gbdt*"
max_bin	255
learning_rate	0.2

Table 5.8: Tuned Hyperparameters of Layer 1

Layer 2

The following snippet shows all the values that are tested with the Grid Search: Since totally 18 Grid Search are conducted using at most three values per parameter and also using at most

three parameters at once, it is disclaimed to show all the experiments separately. Instead, the list containing all the values that are used is shown.

```
tunedParameters = [
    {"max_depth": [7, 8, 9, 10],
     "num_leaves": [10, 30, 40, 43, 44, 45, 46, 47, 48, 50, 55, 60, 70,
     ↪  80, 100],
     "min_child_samples": [100, 120, 130, 136, 137, 138, 140, 141, 142,
     ↪  143, 144, 145, 149, 150, 155, 160, 170, 200, 250],
     "boosting_type": ["gbdt", "dart"],
     "max_bin": [200, 230, 240, 245, 247, 250, 252, 254, 255],
     "learning rate": [0.05, 0.08, 0.09, 0.1, 0.11, 0.12, 0.15, 0.2, 0.25]
     }
]
```

After all those values are tested, the maximum score of 99.85% is reached with the hyperparameters tuned as listed in Table 5.9:

Hyperparameter	**Value**
max_depth	9
num_leaves	45
min_child_samples	142
boosting_type	"*gbdt*"
max_bin	255
learning_rate	0.1

Table 5.9: Tuned Hyperparameters of Layer 2

Chapter 6

Evaluation

This Chapter presents the evaluation of the newly implemented prototype for malicious DoH traffic detection. First, the accuracy of the feature extraction component is tested. To this end, the features that are extracted from a test file are compared with the features that *Wireshark* can extract. A second comparison is made between the features that are extracted from a test file by SecGrid and by [37]. Then the feature importance of Layer 1 and Layer 2 is tested by comparing the findings of this thesis to [42] and by discussing the novel features implemented in this thesis. Next, the Machine Learning models of Layer 1 and Layer 2 are evaluated with the conventional metrics that are used for the evaluation of Machine Learning models. Finally, a new data-set is used to test the accuracy of Layer 1 using other data than the data contained in the data-set [39]. Three experiments are conducted: in the first experiment the data from the new data-set is used to build a test data-set and then the model of Layer 1 which is tested with the original data predicts the new test data-set. In the second experiment, the new data-set is used to build a training and a testing data-set with which the model of Layer 1 is trained and tested. In the final experiment, the data from the new data-set is used to build a training data-set, with which the model of Layer 1 is trained, subsequently the model predicts data from the original data-set.

6.1 Feature Extraction Accuracy

In this Section the accuracy of the Feature Extraction module implemented to SecGrid is evaluated. To this end, on the one hand, the extracted features of a test PCAP-file by SecGrid are compared to the features that *Wireshark* can extract. *Wireshark* is used since it is an established software for the analysis of internet data protocols which enjoys the confidence of scientists. On the other hand, they are compared with the features that [36] extracted from the same file. As a reference, the file "*doh_brw_mal_iodine.pcap*" which holds the requirements provided by [36] is used because it is a part of the data-set [39] which he created. It was created using the tunnel tool *iodine*. Furthermore, it contains ending and non-ending TCP flows, which is a part of the approach of this thesis. The first comparison is conducted between the reference PCAP-file and the result CSV-file "*doh_brw_mal_iodine.pcap-ML-features-DoH.xlsx*" of SecGrid in Section 6.1.1 to show the accurate depiction of the TCP

flows in this file. The second comparison in Section 6.1.2 is conducted between the result CSV-file "*doh_brw_mal_iodine.pcap-ML-features-DoH.xlsx*" of SecGrid and the result CSV-file "*doh_brw_mal_iodine_dohlyzer.xlsx*" of [37] to show and discuss the accordance and the differences between the two different approaches of feature extraction.

6.1.1 Comparison to Wireshark

In this Section, the extracted features of a reference PCAP-file analyzed with the SecGrid (Figure 6.1) prototype are compared to the features which *Wireshark* is able to extract. In *Wireshark*, the function which shows the statistic of every connection is used and shown in Figure 6.2. Only the most important features (*Number of Flows*, *Source & Destination*, *State*, *Total Number of Packets Total Packet Length*, and *Duration*) are examined, since there are totally 41 features and all of them are computed using those most important features.

Starting with the *Number of Flows*, SecGrid and *Wireshark* both extract 14 flows, and *Source & Destination* is the same on both sides. According to both, SecGrid and *Wireshark*, the file has 6 ending flows and 8 open flows. In Figure 6.2, this becomes visible in the Gantt Chart between the two columns *Rel Start* and *Duration*, where the first six flows are ending, and the last eight flows are cut off at the end of the column *Duration*. The row with the feature *Duration* shows that every number extracted by SecGrid is identical to the numbers extracted by *Wireshark*. In Table 6.1 the comparison of these three Features is listed.

Feature	*SecGrid*	*Wireshark*
Flows	14	14
Source & Destination	47688 → 433 47690 → 433 ⋮ 47714 → 433	47688 → 433 47690 → 433 ⋮ 47714 → 433
State	6 ending 8 open	6 ending 8 open
Duration (Seconds, rounded)	462.4 462.9 ⋮ 14.5	462.4 462.9 ⋮ 14.5

Table 6.1: Comparison of the extracted Features *Flows*, *Source & Destination*, *State*, and *Duration*

The *Total Number of Packets* is more complex. In the column of SecGrid in Table 6.2, there are two numbers listed, the left number is the extracted number of TCP segments that carry the ACK flag and Packets which were sent and received, and the right number is the total number of flow packets extracted from the packet length metrics. The column has more entries, since

the features of the flow metrics and the packet length metrics are extracted differently (see Section 5.3). The *Total Number of Packets* extracted from the packet length metrics (right) match exactly with the number of packets which *Wireshark* extracted. However, the number of packets which is extracted from the flow metrics is in every flow smaller than the number of packets extracted by *Wireshark*. This is caused by the fact that *node pcap* handles the packets according to their *Sequence Number*, if e.g. two ACKs have the same *Sequence Number*, this means that their packet length is summed. But this special handling has no impact on the next feature *Total Packet Length*, there the two numbers are identical.

Feature	SecGrid	*Wireshark*
Total Number of Packets	249; 306	306
	314; 370	370
	378; 432	432
	362; 422	422
	317; 373	373
	304; 362	362
	322; 379	379
	315; 353	353
	286; 321	321
	297; 320	320
	312; 327	327
	332; 342	342
	309; 310	310
	72; 73	73

Table 6.2: Comparison of the extracted Feature *Total Number of Packets*

The *Total Packet Length* of *Wireshark* is different to the one that was computed with SecGrid. This is caused by the rounding of the numbers in *Wireshark*, therefore also the numbers of *Sent Bytes* and *Received Bytes* is shown. Already in the first row, it becomes visible that in *Wireshark* a rounding error happens, since $13'000 + 26'000 \neq 40'000$. As mentioned in Section 5.3, *node_pcap* only delivers the size of the IPv4 packet and the TCP packet, the additional frame information (16 Bytes or more per packet) is neglected, and additionally this link is not observed in this ML approach. E.g. for the first flow, this means if the flow has 306 packets and 16 Bytes for each packet are neglected, then there are totally at least 4896 Bytes neglected. According to *Wireshark*, the packet size is between 39'000 and 40'000 Bytes and when the neglected 4886 Bytes are subtracted, then there are between 34'104 and 35'104 Bytes remaining, and the extracted 34'665 Bytes of SecGrid fit into this interval. Table 6.3 shows the comparison of the *Total Packet Length* extracted by SecGrid and *Wireshark*.

The comparison between these two files shows the fundamental correctness of depiction of TCP flows by the Feature Extraction module implemented in SecGrid. It is able to extract every single TCP flow in a PCAP-file (feature *Flows*) and to identify the source and the destination port of every single TCP flow (feature *Source & Destination*). An important point for the approach of this thesis is the recognition of ending and non-ending flows, which is done

correctly regarding the feature *State*. Furthermore, it correctly extracts every single packet of the flow correctly, which becomes visible by the comparison of the features *Total Number of Packets*, *Total Packet Length*, and *Duration*.

Feature	SecGrid	*Wireshark*
Total Packet Length (Bytes)	34665	40'000 (13'000 + 26000)
	37809	44'000 (15'000 + 28'000)
	39997	47'000 (19'000 + 28'000)
	40089	47'000 (19'000 + 28'000)
	38086	44'000 (16'000 + 28'000)
	36888	43'000 (15'000 + 27'000)
	36714	43'000 (16'000 + 26'000)
	37253	43'000 (15'000 + 27'000)
	34575	40'000 (14'000 + 25'000)
	35249	40'000 (14'000 + 26'000)
	36171	41'000 (14'000 + 27'000)
	37245	43'000 (16'000 + 27'000)
	35463	40'000 (14'000 + 26'000)
	10665	11'000 (4'000 + 8'000)

Table 6.3: Comparison of the extracted Feature *Total Packet Length*

flow_number	source_port	destination_port	state	duration	total_nr_of_packets	total_nr_of_packets	total_packet_length
1	47688	443	1	462.3848941	249	306	34665
2	47690	443	1	462.8844242	314	370	37809
3	47692	443	1	463.800112	378	432	39997
4	47694	443	1	462.845567	362	422	40089
5	47696	443	1	463.9019952	317	373	38086
6	47698	443	1	461.9127841	304	362	36888
7	47700	443	0	426.459563	332	379	36714
8	47702	443	0	365.95122	315	353	37253
9	47704	443	0	318.5305929	286	321	34575
10	47706	443	0	243.421689	297	320	35249
11	47708	443	0	182.6181252	312	327	36171
12	47710	443	0	122.320071	332	342	37245
13	47712	443	0	62.78638697	309	310	35463
14	47714	443	0	14.504071	72	73	10665

Figure 6.1: Summarized Evaluation of SecGrid

6.1.2 Comparison to Related Work

In this Section the result CSV-file "*doh_brw_mal_iodine.pcap-ML-features-DoH.xlsx*" of SecGrid (see Figure 6.1) and the result CSV-file "*doh_brw_mal_iodine_dohlyzer.xlsx*" computed with [37] by [36] (Figure 6.3) are compared. Considering the file computed with [37], the number of rows directly attracts the attention. There are totally 44 rows, i.e. 30 rows more than SecGrid. The rows contain the computed feature values of the different flows, whereas each flow is separated into small clumps limited by the timeout constant of probably 120 seconds. The timeout can be observed in the column *duration*. Additionally, the clumps are separated into the direction of the flow, *i.e.,* into requests from the client and responses from the server.

Ethernet | IPv4 · 1 | IPv6 | TCP · 14 | UDP

Address A	Port A	Address B	Port B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration	Bits/s A → B	Bits/s B → A
192.168.20.144	47688	1.1.1.1	443	306	40 k	125	13 k	181	26 k	0.000000	462.4002	237	458
192.168.20.144	47690	1.1.1.1	443	370	44 k	149	15 k	221	28 k	60.375722	462.8997	271	496
192.168.20.144	47692	1.1.1.1	443	432	47 k	215	19 k	217	28 k	121.231899	463.8155	332	488
192.168.20.144	47694	1.1.1.1	443	422	47 k	205	19 k	217	28 k	182.071160	462.8609	332	489
192.168.20.144	47696	1.1.1.1	443	373	44 k	153	16 k	220	28 k	242.848056	463.9308	277	494
192.168.20.144	47698	1.1.1.1	443	362	43 k	151	15 k	211	27 k	303.759651	461.9280	274	476
192.168.20.144	47700	1.1.1.1	443	379	43 k	180	16 k	199	26 k	364.669344	426.4756	316	495
192.168.20.144	47702	1.1.1.1	443	353	43 k	144	15 k	209	27 k	425.434456	365.9665	337	611
192.168.20.144	47704	1.1.1.1	443	321	40 k	129	14 k	192	25 k	486.167134	318.5458	357	651
192.168.20.144	47706	1.1.1.1	443	320	40 k	125	14 k	195	26 k	546.939999	243.4370	464	876
192.168.20.144	47708	1.1.1.1	443	327	41 k	129	14 k	198	27 k	607.999418	182.6334	638	1193
192.168.20.144	47710	1.1.1.1	443	342	43 k	148	16 k	194	27 k	668.809591	122.3354	1052	1767
192.168.20.144	47712	1.1.1.1	443	310	40 k	123	14 k	187	26 k	729.675559	62.8053	1830	3366
192.168.20.144	47714	1.1.1.1	443	73	11 k	31	4003	42	7932	790.510624	14.5193	2205	4370

Figure 6.2: Summarized Evaluation of *Wireshark*

Since the clumps vary widely from the result file of SecGrid also the computed values of the clumps are very different, and therefore they cannot be compared. But an interesting point is the number of rows in the file computed with [37], which is more than four times bigger than the number of rows in the file computed by SecGrid. This shows that the clumping process in SecGrid is more effective and memory saving than the one in [37]. Also that the Feature Extraction module of SecGrid does not distinguish the flow direction can be seen as a clear strength, since [42] removed the information about the IP addresses and the port information. It is possible that without this information, the ML model has a deranging side noise, since it does not know from which side the data flow is coming. The Feature Extraction model of SecGrid bypasses this problem by summarizing the whole flow into one row and extracting the measures from it. Thus, the ML model cannot be disturbed by the flow direction anymore.

SourceIP	DestinationIP	SourcePort	DestinationPort	TimeStamp	Duration	FlowBytesSent	FlowSentRate	FlowBytesReceived	FlowReceivedRate	PacketLengthVariance
192.168.20.144	1.1.1.1	47688	443	22.03.2020 17:24	122.713913	12293	100.1760901	24783	201.9575401	43297.54232
192.168.20.144	1.1.1.1	47690	443	22.03.2020 17:25	123.26621	14294	115.960408	27028	219.2652796	32724.03453
192.168.20.144	1.1.1.1	47692	443	22.03.2020 17:26	109.272989	17764	162.5653344	26636	243.7564877	15931.68451
1.1.1.1	192.168.20.144	443	47688	22.03.2020 17:27	120.816741	496	4.105391322	448	3.708095387	9
192.168.20.144	1.1.1.1	47694	443	22.03.2020 17:28	108.081764	17677	163.5521049	26587	245.9896935	28895.181
1.1.1.1	192.168.20.144	443	47690	22.03.2020 17:28	120.817	496	4.105382521	448	3.708087438	9
192.168.20.144	1.1.1.1	47696	443	22.03.2020 17:29	109.25696	14588	133.5200979	26957	246.730277	32553.52983
1.1.1.1	192.168.20.144	443	47688	22.03.2020 17:29	120.817004	496	4.105382385	448	3.708087315	9
192.168.20.144	1.1.1.1	47692	443	22.03.2020 17:29	105.742974	448	4.236688104	434	4.1042916	8.96
192.168.20.144	1.1.1.1	47698	443	22.03.2020 17:30	107.225311	14370	134.0168647	25817	240.7733749	18515.43063
1.1.1.1	192.168.20.144	443	47690	22.03.2020 17:30	120.817042	496	4.105381094	448	3.708086149	9
192.168.20.144	1.1.1.1	47694	443	22.03.2020 17:30	105.743031	448	4.23668582	496	4.690616444	9
1.1.1.1	192.168.20.144	443	47688	22.03.2020 17:31	98.007297	733	7.479034954	528	5.387353964	100.1475
192.168.20.144	1.1.1.1	47700	443	22.03.2020 17:31	109.291589	15711	143.7530568	25149	230.109199	31082.56132
192.168.20.144	1.1.1.1	47692	443	22.03.2020 17:31	120.83198	504	4.171081199	496	4.104873561	8.968858131
192.168.20.144	1.1.1.1	47696	443	22.03.2020 17:31	105.742976	448	4.236688024	496	4.690618883	9
1.1.1.1	192.168.20.144	443	47690	22.03.2020 17:32	97.954464	733	7.483068868	528	5.390259703	100.1475
192.168.20.144	1.1.1.1	47702	443	22.03.2020 17:32	109.198449	14492	132.7125077	27025	247.4852001	32456.16095
192.168.20.144	1.1.1.1	47694	443	22.03.2020 17:32	120.831974	504	4.171081406	496	4.104873765	8.968858131
192.168.20.144	1.1.1.1	47698	443	22.03.2020 17:32	105.74305	448	4.236685059	496	4.690615601	9
192.168.20.144	1.1.1.1	47704	443	22.03.2020 17:33	122.178773	13515	110.6165962	25056	205.0765398	34992.37398
1.1.1.1	192.168.20.144	443	47692	22.03.2020 17:33	97.774515	733	7.496841074	584	5.972926585	97.63265306
192.168.20.144	1.1.1.1	47696	443	22.03.2020 17:33	105.743016	448	4.236686421	496	4.690617109	9
192.168.20.144	1.1.1.1	47700	443	22.03.2020 17:33	105.742979	448	4.236687903	496	4.69061875	9
192.168.20.144	1.1.1.1	47698	443	22.03.2020 17:34	120.828028	504	4.171217625	496	4.105007822	8.968858131
1.1.1.1	192.168.20.144	443	47694	22.03.2020 17:34	98.01116	733	7.478740176	584	5.958505133	97.63265306
192.168.20.144	1.1.1.1	47706	443	22.03.2020 17:34	107.500912	13629	126.7803198	26128	243.0491008	34194.3546
192.168.20.144	1.1.1.1	47702	443	22.03.2020 17:34	105.74301	448	4.236686661	496	4.690617375	9

Figure 6.3: Cutout from the summarized Evaluation of the DoHlyzer [37] computed by [36]

6.2 Feature Importance

In this Section, the relevance of the features that are adopted from [36] and the novel features that are newly introduced in this thesis is discussed. The feature importance of tree based ML algorithms can be retrieved pretty simple in *scikit-learn* [54]. As an output, a bar plot is received where all the features are listed hierarchically according to their importance. The importance of a feature in a tree based ML algorithm means how many times the feature is needed for each tree that was built.

6.2.1 Layer 1

First the feature importance of Layer 1, which can be seen in Figure 6.4 is discussed. [42] found a feature sequence with descending order using statistical tests. This sequence is also used in this thesis. What attracts the attention at first is that compared to the feature importance in Figure 3.1, the feature importance found by SecGrid (Figure 6.4) is not the same. There are definitely correlations, like the feature *duration* or the feature *response_request_mode* which are in the top segment of both figures, but there is also the feature *flow_bytes_sent* which is situated in the middle of the hierarchy in Figure 3.1 but is in the second place in Figure 6.4. Also in the bottom section of both figures there are accordances and discordances. Regarding for example the feature *packet_length_variation*, which is situated in the bottom segment of both figures, accordances can definitely be found, but regarding the feature *packet_time_standard_deviation* disaccordance can be observed.

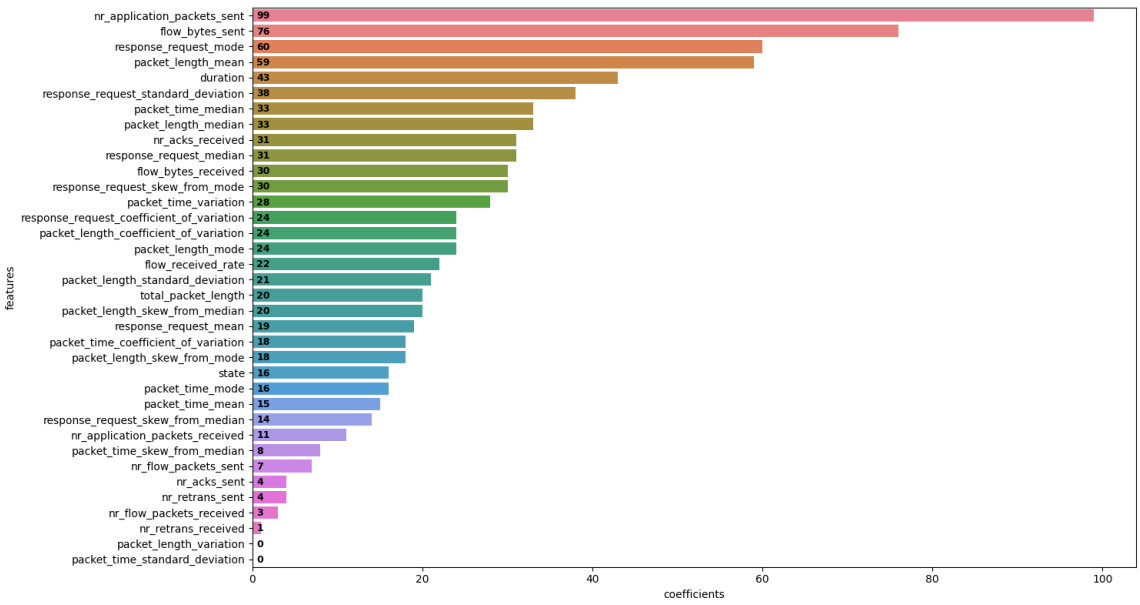


Figure 6.4: Feature Importance of Layer 1 found in this Thesis

According to [35], the duration of a flow is one of the most important indicators to differentiate between DoH and non-DoH traffic. The evaluation in Figure 6.4 confirms this statement, since the feature *duration* is found in the top features of the most important ones of the model. Finding the novel features amongst the top of the most important features is also an indicator

for the fundamental correctness of the feature extraction component. Another very important indicator is the amount of data that is sent and received in a flow. This statement is validated as well, since the two features *flow_bytes_sent* and *flow_bytes_received* and also other features that indicated the amount of data sent in a flow can be found amongst the most important features of the model. Finally, the feature *packet_length_mean* is found as the absolute most important feature. This finding is not surprising at all, since the size of the packet is also an important indicator for DoH and non-DoH traffic.

Some of the novel features that are introduced in this thesis have an impact on the model as well. At the top of the hierarchy is the feature *nr_application_packets_sent*. This finding correlates with the statement of [35], who said that the amount of data sent in the flow is one of the most important differences between a normal HTTPS and a DoH connection. Thus, it is not surprising to find this feature as one of the most important ones. Another feature that can be found amongst the top of the most important features is the *nr_acks_received*. This is not further surprising since if the number of packets sent is important, the number of ACKs also must have an impact on the model since received ACKs are the answer of the resolver that the packet was received. The feature *total_packet_length* can be found in the middle of the hierarchy. It again correlates with the statement of [35] that the amount of data sent in a flow is an important indicator for DoH or non-DoH traffic. The other features seem not to have a big impact to the model of Layer 1.

6.2.2 Layer 2

In a next assessment, the feature importance of Layer 2, which is visualized in Figure 6.5 is discussed. Compared to the feature sequence that [42] found, the order is slightly different. On the one hand, features like the *duration* and the *packet_time_skew_from_median* are found on top of the hierarchy in both sequences. On the other hand, features like *packet_length_coefficient_of_variation* or *flow_received_rate* are found on top of the sequence that [42] found, but in the feature importance figure of Layer 2 they are found in the middle or in the lower part of the sequence. In reverse, features like *response_request_median* or *flow_bytes_received* that are amongst the top of the most important features of the model of Layer 2 are situated in the middle or even in the bottom part of the sequence that [42] found.

Figure 6.5 shows additionally that some novel features have an impact on the model. The first feature found in the list is *nr_application_packets_sent*. Another feature that is found in the top of the hierarchy is the *state*. It seems like either of the two traffic types is interrupted more than the other one. The feature *total_packet_length* can be found in the middle of the hierarchy. The other features seem not to have a big impact to the model of Layer 2.

6.2.3 Discussion

The correctness of the sequences [42] seem to be only partially true, compared to the findings in this thesis. While there are in both Layers some correlations, there are also differences. The cognition is that the feature importance is slightly different for ML models which are trained

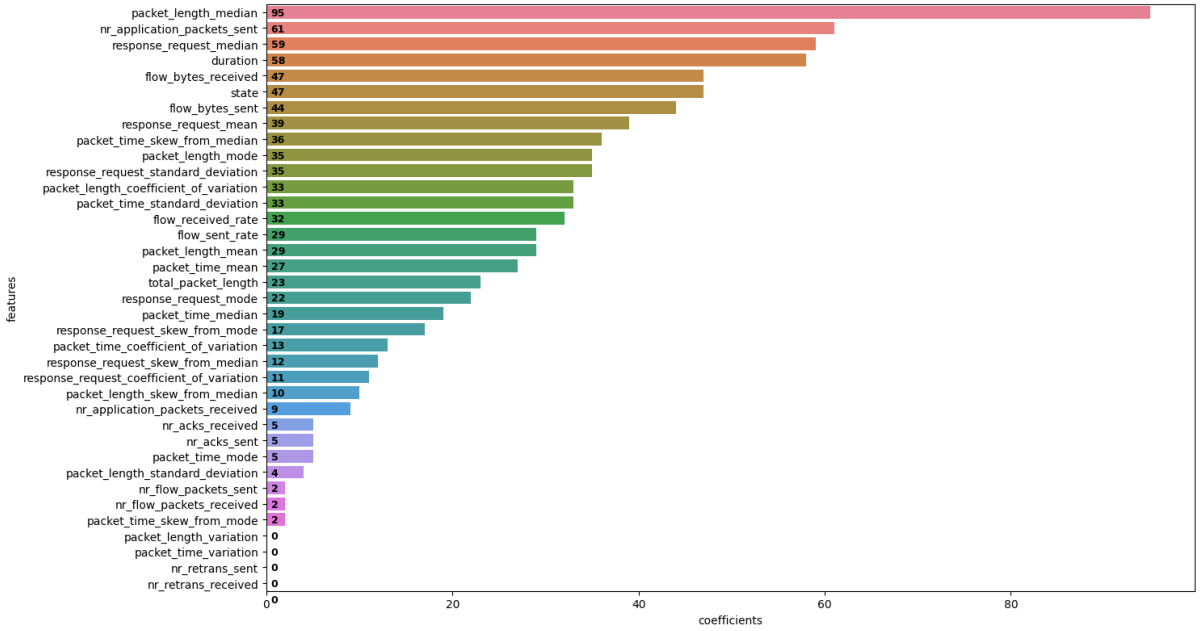


Figure 6.5: Feature Importance of Layer 2 found in this Thesis

with different training data-sets. Thus, it is important to keep all the features to train and test the ML model to not lose accuracy in the predictions of the ML model.

An interesting finding of this thesis is that some of the novel features seem to have an impact on the ML model and the consequential predictions. Especially, the *nr_application_packets_sent* and the *state* have to be highlighted. The *nr_application_packets_sent* can be found surprisingly on top of both of the evaluation figures of the two Layers. There seems to be a huge difference in the number of application packets sent between HTTPS connections and DoH connections, as well as between benign and malicious DoH traffic. For malicious traffic, this makes sense since the aim of malware is to exfiltrate data from the system of its victim in small slices to stay under the radar of warning systems. The other feature that needs to be highlighted is the *state*. Especially in the hierarchy of Layer 2 it is found on the top. There seems to be a context between either benign or more likely malicious DoH traffic and non-ending TCP flows.

6.3 ML Model

In this Section, the ML models are evaluated in therms of performance and accuracy. Before this can be done, all the metrics and figures that are used for this evaluation are presented. In a further step, the models of Layer 1 and Layer 2 are tested using all the presented metrics. Finally, to have a comparison with different data, the model of Layer 1 is tested using another data-set [55]. Unfortunately, no other data-set that satisfy the requirements for Layer 2 is available at the time of writing, therefore a cross-validation for Layer 2 is not possible.

6.3.1 Metrics

The evaluation of the ML models requires tautological metrics and figures, i.e. the performance, the accuracy, the recall, the precision, the F1-score, the specificity, the precision recall, and Receiver Operating Characteristics Curve (ROC). Therefore, they are explained in this separate section and subsequently used for all further ML evaluations.

Performance

On the basis of the findings of [42], the LGBM algorithm was implemented in this work as well due to its alleged outstanding performance. In the evaluation of this thesis, the performance is the time which is needed to train the model and the time needed to classify one clump. During the implementation phase, the observation was made that the measured duration of the whole process is never exactly the same, they always deviate slightly from each other. Therefore, the whole pipeline is run ten times and then the mean time is computed to ensure that an average performance time can be evaluated. Furthermore, the Random Forest (RF) model which was already implemented into SecGrid was used to compare if the performance of the LGBM algorithm is indeed much better than the performance of RF, like [42] stated.

Confusion Matrix

The confusion Matrix is a plot type that illustrates for the two classes 0 and 1 how many predictions for per class were successful and how many predictions were unsuccessful. It serves mainly to show the classification of the two classes, since it is possible that they are not equally successful or unsuccessful classified. Figure 6.6 shows the theoretic composition of a confusion matrix. If a data-point was labeled as positive and predicted as positive, it is called a *true positive* (TP). When a data-point is labeled as a positive but was predicted as a negative, it is called a *false positive* (FP). Equally, when a data-point was labeled negative and was predicted negative, it is called a *true negative* (TP), and when a data-point was labeled as negative but predicted as positive, it is called a *false negative* (FN) [56].

	Predicted as positive	Predicted as negative
Labeled as positive	80	20
Labeled as negative	5	195

Figure 6.6: Theoretical Composition of a Confusion Matrix [56]

Accuracy

The accuracy of an ML Classifier indicates how often the model made the correct predictions compared to the total number of data to be predicted. Equation 6.1 shows how the accuracy is computed [56].

$$accuracy = \frac{\#\ of\ correct\ Predictions}{\#\ of\ total\ Data\ Points} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Recall

The recall of an ML Classifier indicates the percentage of the relevant elements found by the classifier out of the total amount of elements that are truly relevant. Therefore, the amount of TP is divided by the sum of the TP and the FN, which can be seen in Equation 6.2 [56].

$$recall = \frac{TP}{TP + FN} \tag{6.2}$$

Precision

The precision of an ML Classifier indicates the percentage of the truly relevant elements out of the total amount of element predicted to be relevant. Therefore, the amount to TP is divided by the sum of TP and FP, which can be seen in Equation 6.3 [56].

$$precision = \frac{TP}{TP + FP} \tag{6.3}$$

F1-Score

The F1-score or also the *harmonic mean* indicates the balance between the recall and the precision. It is twice the product of the recall and the precision divided by the sum of the precision and the recall, which can be seen in Equation 6.4 [56].

$$F1 - Score = 2 * \frac{TP}{2TP + FN + FP} \tag{6.4}$$

Receiver Operating Characteristics Curve

The Receiver Operating Characteristics Curve (ROC curve) shows the rate of the amount of TP compared to the amount of TN classified by the model. In other words, it shows how many correct TPs and TNs are classified while the classification of FPs and FNs happens [56].

6.3.2 Layer 1 using Data-Set CIRA-CIC-DoHBrw-2020

For the evaluation of Layer 1 a special testing data-set was created. It contains 4000 equally distributed clumps, i.e. 2000 non-DoH traffic data and 2000 DoH traffic data. Since there is not enough benign DoH data available in the data-set [39], benign DoH data from TSL2 is taken to ensure that not the same data from TSL1 is tested and to complement 1000 clumps. The exact distribution can be seen in Table 6.4.

		Non-DoH (Chrome)	Non-DoH (Firefox)	DoH (Chrome)	DoH (Firefox)
Benign	**AdGuard**	250	250	25	125
	Cloudflare	250	250	45	125
	Google	250	250	90	125
		250	250	340	125
Malicious	**DNS2TCP**	0		4000	
	DNScat2	0		3000	
	iodine	0		3000	
Total		2000		2000	

Table 6.4: Distribution of the Data-Points in TSL1

Performance

The mean duration of the training of Layer 1 is 3.57 seconds, the prediction duration of Layer 1 is 0.04 Seconds, using 35 features. [42] stated that with the default hyperparameters, their model needed 87 seconds for the whole process of Layer 1, using 21 features.

Accuracy & Confusion Matrix

The accuracy of Layer 1 is 98.55%. Compared to [42], who received an accuracy of 99.78% it is slightly inferior, but it is still a good result. The confusion matrix of Layer 1 can be seen in Figure 6.7 on the left. It shows that out of 2000 clumps only seven non DoH traffic clumps are not classified correctly, whereas 51 DoH traffic clumps are not classified correctly.

Recall, Precision, F1-Score & ROC Curve

The recall for the non DoH traffic part is 100%, the recall for the DoH traffic part is 97.0%, which results in an average recall of 98.5%. The precision for the non DoH traffic part is 98.0%, the recall for the DoH traffic part is 100%, which results in an average recall of 99.0%. The F1-score for the non DoH traffic part is 99.0%, the recall for the DoH traffic part is 99.0%,

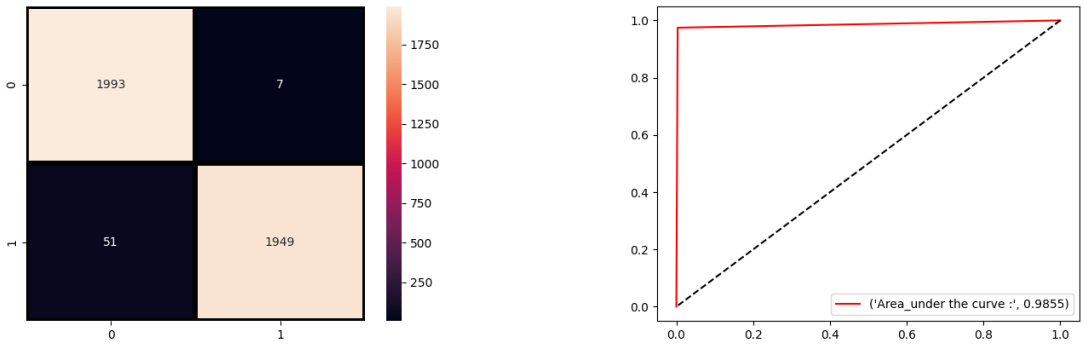


Figure 6.7: Confusion Matrix (left) and ROC Curve of (right) of Layer 1

which results in an average recall of 99.0%. Table 6.5 shows an overview of the computed values. Since the amount of non-DoH clumps and the amount of DoH clumps is equal, also the macro average and the weighted average are equal, therefore only the superficial term "average" is used here. The ROC Curve of Layer 1 can be seen in Figure 6.7 on the right. Again, it shows the accuracy of the model and makes certain that nearly every data-point is classified correctly, with an insignificant part of FPs and FNs.

	Recall	**Precision**	**F1-Score**
non-Doh	1.00	0.98	0.99
DoH	0.97	1.00	0.99
Average	0.985	0.99	0.99

Table 6.5: Recall, Prescison and F1-Score of Layer 1

Discussion

As already mentioned before, the accuracy of [42] is not quite reached, but the results are still excellent. Also the precision, the recall and the F1-score are exceptional. Concerning the performance, the model which was implemented into SecGrid is massively faster than the model of [42], although they used only 21 features unlike the model in SecGrid which uses 35 features. The prediction time, which lasted 0.04 seconds, is sensational and suits excellently for a system like SecGrid that promises fast detection times. It is not quite clear why the model of this thesis is faster to such an extent. A reason for this could be the different clump handling, but this is rather unlikely since the data structure is similar. Another reason for this difference could be the setup [42] used, but this also does not justify such a huge difference. The only way to find out the reason for this performance difference is to rebuild the work of [42] for a deeper insight.

6.3.3 Layer 2 using Data-Set CIRA-CIC-DoHBrw-2020

Similar to the evaluation of Layer 1 a special test data-set is compiled. It contains 4000 data-points which are equally distributed to 2000 benign DoH clumps and 2000 malicious DoH clumps. Again, since not enough benign DoH clumps are available, benign DoH clumps from TSL1 are taken to ensure that the test data-set contains 2000 benign DoH clumps and that no data from TSL2 is taken. The exact distribution of this test data-set is presented in Table 6.6.

	Benign (Chrome)	Benign (Firefox)	Malicious
AdGuard	25	250	0
Cloudflare	45	250	0
Google	90	250	0
Quad9	840	250	0
DNS2TCP	0		800
DNScat2	0		600
iodine	0		600
Total	2000		2000

Table 6.6: Distribution of the Data-Points in TSL2

Performance

The mean duration of the training Layer 2 is 4.18 seconds, the mean predicting duration is 0.084 seconds using 38 features. [42]'s approach had a total duration of 40 seconds for the training and the prediction phase of Layer 2 using 27 features.

Accuracy & Confusion Matrix

The accuracy of Layer 2 is 99.975%. Compared to [42] who received an accuracy of 99.996% it is again slightly inferior, but it is still a good result. Figure 6.8 on the left shows the confusion matrix of Layer 2. It depicts that only one benign DoH traffic clump is wrong classified, the malicious DoH traffic clumps were all correctly predicted.

Recall, Precision, F1-Score & ROC Curve

The averages of recall, the precision and the F1-score are all 100%. This means that for each of the three metrics, the benign DoH traffic as well as the malicious DoH traffic was computed to 100%. They are all listed in Table 6.7. The ROC Curve of Layer 2 can be seen in Figure 6.8 on the right. It reinforces again how precise the model is, since the angle in the top left corner is nearly 90°, which means that there is only one FN and no FPs.

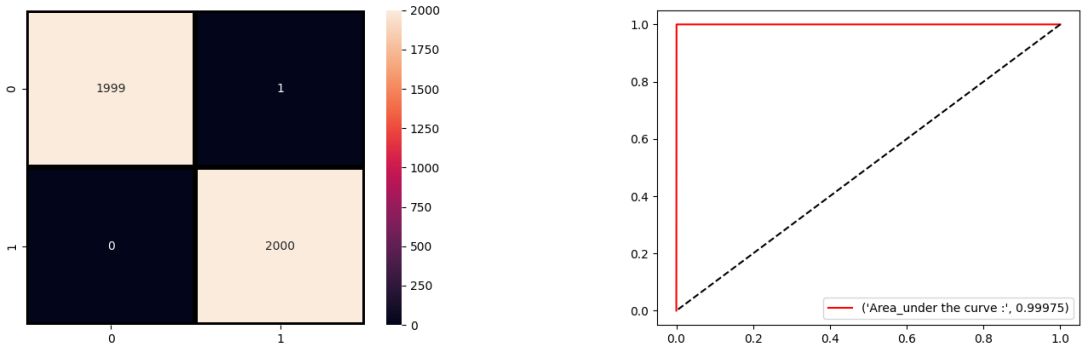


Figure 6.8: Confusion Matrix (left) and ROC Curve of (right) of Layer 2

	Recall	Precision	F1-Score
non-Doh	1.00	1.0	1.0
DoH	1.0	1.00	1.0
Average	1.0	1.0	1.0

Table 6.7: Recall, Precising and F1-Score of Layer 1

Discussion

As already mentioned before, the accuracy of [42] is not quite reached, but like already in the first Layer, the results are still excellent. The computed values for the precision, the recall and the F1-score is outstanding and cannot be better. It means that the model is able to predict nearly every malicious clump extracted from the data-set [39] correctly. Concerning the performance of the model implemented into SecGrid, it is massively faster than the model of [42]. With a prediction time of 0.084 seconds, the model is extremely fast, which is perfect for a system like SecGrid that relies on fast results. As such, the performance achieved here could motivate the usage of such a model for real-world use cases like intrusion detection and mitigation. To summarize, the whole pipeline delivers astonishing results while using the data-set [39].

6.3.4 Performance using Random Forest Algorithm

[42] found that the LGBM algorithm is much faster than the Random Forest (RF) algorithm. The intention of this section is to verify this statement. Therefore, instead of LGBM, the RF algorithm is implemented. To do so, there are only three lines to be changed in the code: the import of LGBM has to be changed to the import of RF and the two lines where LGBM is initiated have to be changed to the initiation of RF. Since [42] used the default modulations of RF with no further hyperparameter tuning, they are also used for the verification in this work.

As done in Sections 6.3.2 and 6.3.3, the whole pipeline is run ten times, while the durations of the training and the durations of the predictions phases of both Layers are measured. Then,

the mean duration of all the four different durations is computed. The training phase of Layer 1 lasted 17.74 seconds, the prediction phase of Layer 1 lasted 0.036 seconds. Thus, RF needs more than four times as much time than LGBM needs for the training of Layer 1, whereas the duration of prediction phase is slightly smaller. The training phase of Layer 2 lasted 14.94 seconds and the prediction phase of Layer 1 lasted 0.060 seconds. Thus, RF needs more than twice as much time as LGBM for the training phase, whereas the prediction time of RF is slightly smaller.

The finding of this Section is that the LGBM algorithm is a good choice concerning the performance. Although the RF algorithm is slightly faster in predicting, it is in return extremely slower in predicting. However, it cannot be negligible that no hyperparameter optimization is conducted for the RF algorithm. Comparing the results with [42], it is noticeable that their model of Layer 1 needed 4991 seconds and the model of Layer 2 needed 890 seconds for the whole process. Again, only rebuilding their approach will show how this huge duration differences arose.

6.4 Layer 1 using a Different Data-set

Since the results of Layer 1 and 2 are truly impressive, the aim of this Section is to validate these good results, when transferring the model to a different data-set. The original plan was to take a data-set that contains traffic for both Layers, but unfortunately [39] is the only available data-set that contains malicious DoH traffic. Therefore, only Layer 1 can be tested with other traffic data. To this end, a second data-set taken from the IEEE Dataport [55] is introduced. This data-set contains DoH and non-DoH traffic collected by using the two Browsers Chrome and Firefox. In addition, the two data-sets were generated by using different modulations and addressing multiple DoH resolvers. The list of the IP-addresses of the DoH resolvers can be found in Table 6.8. It is conspicuous that they used partially the same DoH resolvers, but also other DoH resolvers that [39] did not use. Furthermore, the amount of DoH resolvers used is bigger in [55] than in [39]. All these differences are good for the validation with other data than the original data-set because it is an ultimate test to see if the system can also handle data from non-lab-generated data [39]. Furthermore, none of the references used for this work did such an examination of their approaches using another data-set, therefore this examination is also a validation of the data-set[39].

DoH Resolver IP-Address		
88.198.91.187	88.198.91.187	104.22.72.65
104.16.249.249	104.16.248.249	104.22.73.65
104.16.249.249	104.16.248.249	176.9.1.117
146.112.41.2	146.112.41.3	176.9.93.198
185.43.135.1	185.235.81.1	5.1.66.255
104.236.178.232	195.30.94.28	159.69.198.101

Table 6.8: IP-Addresses of the DoH Resolvers used for the Data-Set [55]

6.4.1 Preprocessing

The data-set has to be preprocessed like the former data-set. Therefore, the two folders with one containing the traffic data that was collected using Chrome and the other one containing the data that was collected using Firefox is analyzed with the *Feature Extraction* component of SecGrid. To differentiate between non-DoH and DoH traffic, the IP addresses introduced in Table 6.8 are used. The analysis ended up in four different CSV-files, i.e. clumps of Chrome non-DoH traffic, Chrome DoH traffic, Firefox non-DoH traffic and Firefox DoH traffic.

6.4.2 Jeřábek et. al Data-set as Test-Dataset

For the execution of this examination a test data-set has to be created. This test data-set contains 4000 equally distributed clumps from all the four referred CSV-files in Section 6.4.1. This means that from each of the CSV-files, 1000 random data-points are collected and saved into a new file called *IEEETestDataset*. This test data-set is then saved as *IEEETestDataset.csv* and used as the prediction data-set of Layer 1. In the following sections, the results of this test are presented.

Accuracy & Confusion Matrix

The accuracy of Layer 1 using the data-set [55] as testing data is 62.55%. The confusion Matrix in Figure 6.9 on the left shows that the non-DoH clumps are predicted mostly correct, with a negligible amount of 70 clumps which were predicted incorrect. But looking at the DoH clumps, the confusion matrix unveils that only about 30% are predicted correctly. This is a poor result, since the primary task of this layer is to classify the clumps correctly into DoH and non-DoH clumps for the further processing in Layer 2.

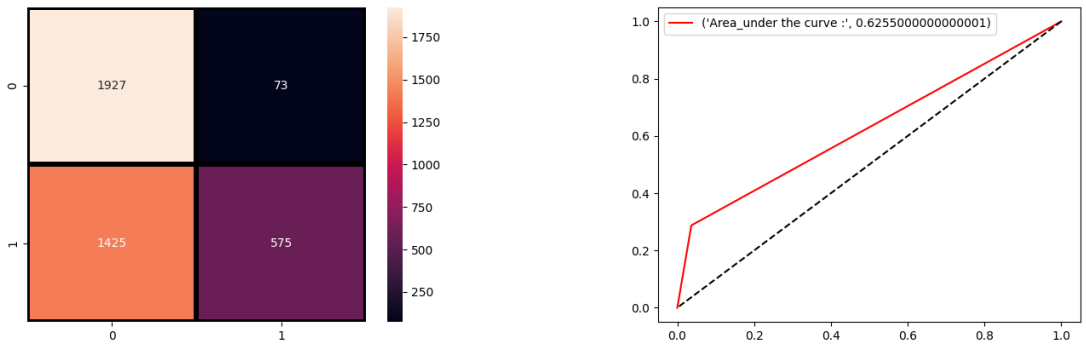


Figure 6.9: Confusion Matrix (left) and ROC Curve of (right) of Layer 1 using Data from [55] as Testing Data-Set

Recall, Precision, F1-Score & ROC Curve

The average recall of Layer 1 using data of the data-set [55] as prediction data is 63.0%, the average precision is 73.0% and the average F1-score is 58.0%. All the values can be seen in Table 6.9. Figure 6.9 on the right shows the ROC-curve, which reinforces the inaccuracy of Layer 1.

	Recall	Precision	F1-Score
non-Doh	0.96	0.57	0.72
DoH	0.29	0.89	0.43
Average	0.63	0.73	0.58

Table 6.9: Recall, Precision and F1-Score of Layer 1 using Data of [55] for the Test Data-Set

Discussion

This examination showed that the newly implemented model has difficulties to handle other data than data from the data-set [39]. On the one hand, the model is highly accurate in predicting non-DoH data, but on the other hand, only about 30% of the DoH data was predicted correctly. This contradicts the findings of Sections 6.3.2 and 6.3.3, where it is proven that the model is very accurate. It seems that Layer 1 is able to separate data from the original data-set, but as soon as other data is involved it is not accurate anymore. This may indicate that the dimensions used for the model may vary between settings in the user-agent or the resolver. Thus, more research in this direction would be needed, which would reproduce and test earlier results and suggest optimizations.

6.4.3 Relying Exclusively on Jeřábek et. al Data-set

The findings of Section 6.4.2 are humbling. Thus, in this Section the data-set [55] is tested if it is generally suited for the separation of non-DoH and DoH traffic. Therefore, a training data-set of the same size as the original training data-set is created, i.e. it contains 40'000 clumps. These 40'000 are divided in half, whereas the first half contains 10'000 clumps non-DoH traffic collected using Chrome and 10'000 non-DoH clumps collected using Firefox. The other half contains 10'000 clumps of DoH traffic collected with Chrome and 10'000 clumps of DoH traffic collected using Firefox. This data-set is the saved in the CSV-file called *TSL1IEEE.csv*. This training data-set is now used to train Layer 1 and after the training it is tested with the testing set *IEEETestDataset.csv*.

Accuracy & Confusion Matrix

The accuracy of Layer 1 using the data-set [55] as training and testing set is 98.85%. The confusion matrix in Figure 6.10 on the left reveals that the non-DoH traffic clumps as well as the DoH traffic clumps are predicted predominantly correct. Only 32 non-DoH traffic clumps

and 14 DoH traffic clumps are predicted incorrectly, thus those numbers are negligible. This is an excellent result which already indicates that the data-set is in principle eligible as training data-set for Layer 1.

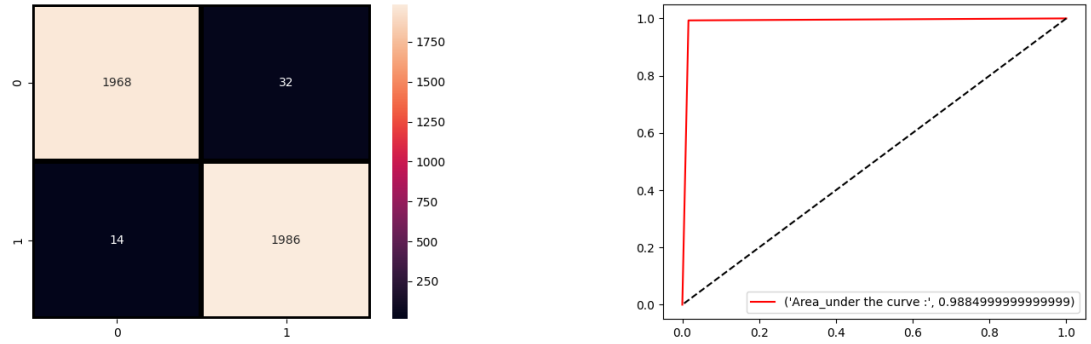


Figure 6.10: Confusion Matrix (left) and ROC Curve of (right) of Layer 1 using Data from [55] Training- and Testing Data-Set

Recall, Precision, F1-Score & ROC Curve

The average recall of Layer 1 using data from [55] as training and testing data-set is 99.0%, the average precision is 99% and the average F1-score is 99.0%. Table 6.10 shows that only the recall of the prediction of non-DoH traffic clumps and the precision of the prediction of DoH traffic clumps is 98.0%, all the residual values are 99.0%. The ROC curve in Figure 6.10 reinforces that the model is exact and that there are nearly no miss-predictions in the model.

	Recall	**Precision**	**F1-Score**
non-Doh	0.98	0.99	0.99
DoH	0.99	0.98	0.99
Average	0.99	0.99	0.99

Table 6.10: Recall, Precision and F1-Score of Layer 1 using Data of [55] for the Training and the Test Data-Set

Discussion

This experiment shows that if the DoH detection component of SecGrid is applied to data that comes from the same data-set is highly accurate with an accuracy of 98.85%. Only a negligible part of both, non-DoH traffic clumps and DoH traffic clumps, is predicted wrong. Additionally, the viability of the data-set [55] is proven when the training and the testing data-set come from it. This means that the DoH detection component of Layer 1 is basically exact, but it cannot be applied to detect DoH traffic from the dataset [55] while it is trained

with data from the data-set [39]. This could further highlight that in a realistic setting, one might have to train a model for each device or resolver. However, more research is needed in this direction.

6.4.4 Jeřábek et. al Data-set as Training Data-Set

The last experiment of this Chapter is to test if the data of [39] is better predictable using data from [55] as training data. To do so, the same training data-set is used as in Section 6.4.3. As the testing data-set, the same data-set is used as in Section 6.3.2.

Accuracy & Confusion Matrix

The accuracy of Layer 1 using data from the data-set [55] as training data-set and the test data from the data-set [39] is 72.35%. The confusion matrix in Figure 6.11 reveals that the non-DoH traffic clumps are predicted nearly perfectly. But as already experienced in Section 6.4.2, the prediction of the DoH traffic clumps is not good. With nearly 50% correct predictions, it was at least not as poor as the experiment result in Section 6.4.2, but this result is still not good.

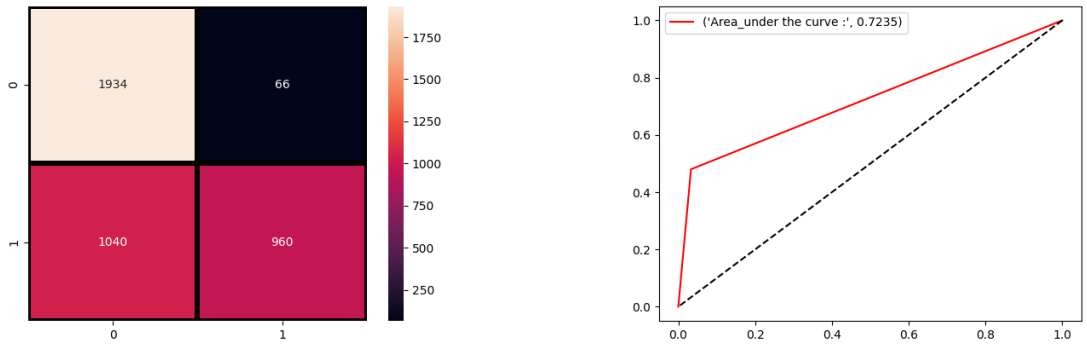


Figure 6.11: Confusion Matrix (left) and ROC Curve of (right) of Layer 1 using Data from [55] as Training Data-Set

Recall, Precision, F1-Score & ROC Curve

The average recall of Layer 1 using data from [55] as the training data-set and the test data from the data-set [39] is 72.0%, the average precision is 0.79, and the average F1-score is 71.0%. Table 6.11 shows that especially the recall of the prediction of the DoH traffic clumps is poor, since it is under 50%. Figure 6.11 clarifies that the performance of this model is also not sufficient.

Discussion

This experiment shows again that the two different data-sets are not compatible to each other. Although the accuracy with 72.35% was about 10% higher than the result of the experiment in Section 6.4.2, this is still not the desired performance. The awareness that can be gained from this experiment is that the different browser settings and the different DoH-servers that were used to generate the data-set [55] disturb in some manner the model when it is trained with data from the data-set [39].

	Recall	**Precision**	**F1-Score**
non-Doh	0.97	0.65	0.78
DoH	0.48	0.94	0.63
Average	0.72	0.79	0.71

Table 6.11: Recall, Precision and F1-Score of Layer 1 using Data of [55] for the Training Data-Set

Chapter 7

Summary, Conclusions, Limitations, and Future Work

In this thesis, a malicious DNS-over-HTTPS detection prototype was implemented into SecGrid. Its components are a feature extraction and a Machine Learning pipeline. The feature extraction component is able to extract informational as well as statistical features out of TCP flows that are contained in an input PCAP-file, whereas the ML pipeline is able to detect malicious DoH traffic within two steps using the values that were extracted in the feature extraction. The first step is to separate DoH traffic from normal HTTPS traffic, the second step is to detect malicious DoH traffic.

The modular architecture of SecGrid facilitated the implementation of the two components, therefore no drastic changes of the architecture of SecGrid had to be done. The feature extraction could be implemented as a subclass of the already available *Abstract PCAP Analyser* class. The *PcapParser* had to be adjusted such that it parses TCP-flows. *node-pcap*, which was previously integrated in the project as a node module, had to be forked and changes concerning the handling of non-ending and packet information extraction had to be done. The feature extraction uses the information that is parsed by the *PcapParser* to compute all the needed features.

Further, the two layered Machine Learning detection component was implemented, whereas both Layers use the same algorithm, namely the Light Gradient Boosting Machine Classifier. Both of these models have to be trained with already available data, therefore the data-set CIRA-CIC-DoHBrw-2020 [39] was used. This data-set was analyzed with the feature extraction component and two different training data-sets were composed, one for each layer and containing the data that is important for the respective Layer. The ML models of Layer 1 and 2 are trained with the respective data-set and are then ready to predict the input PCAP file. Joining the two components, feature extraction and ML model results in the complete prototype for the detection of malicious DoH traffic.

During the implementation, the *TCP_Tracker* of *node-pcap* had to be adjusted such that it can handle non-ending TCP flows. It was noticed that there are other parts of the *TCP_Tracker* that are not completely or not at all implemented, such as the handling of TCP packets different from *ACK* or *FIN* packets. Further, the handling of half-closed TCP flows is not handled,

i.e. flows that are ended using only one *FIN/ACK* sequence. As seen in the evaluation of the feature importance, the *state* is an important feature for both, the separation of non-DoH and DoH traffic and the detection of malicious DoH traffic, therefore, by following up the completion of the handling of non-ending TCP flows it is possible to get an even more important feature that helps for the two detection processes. Generally, the features seem to be computed correctly, whereas the novel features must be highlighted, since some of them have a huge impact on the prediction of the ML models. A further interesting finding in the evaluation is the rounding error that happens in *Wireshark*. The producers could in the future take in account to show the absolute values of the total extracted packet sizes instead of rounded values to avoid this error.

Another point of the feature extraction module is that when huge amounts of data are analyzed, it can claim some time. Although it is clear that files that contain some Gigabytes of traffic just need time to be fully analyzed, this process could still be accelerated. This could be accomplished by adapting the *TCP_Tracker* such that it extracts the data from the PCAP-file in such a way that there is no need for feature extraction to iterate over the same data again. An example therefore is that the *TCP_Tracker* could compute the time between the first and the current packet or the request- and response time at the point when it retrieves the packets. Currently, this is done in the feature extraction, which means that in the whole process it is required to iterate several times over the data and this implicates that feature extraction process is slowed substantially.

To conclude, the two layered detection of malicious DoH traffic is extremely accurate while the training and testing data originates from the same data-set. However, as soon as the testing data comes from another data-set than the training data, already the accuracy of the first Layer gets extremely poor. This is probably due to different browser settings and different DoH resolvers that were used to gather data of the two data-sets. Furthermore, both of the used data-sets are lab-generated data-sets, which do not precisely reflect real world data. Another limitation is that only one existing data-set was found that contains non-DoH traffic, benign and malicious DoH traffic, whereas benign DoH traffic flows are limited to about 30'000 which was just enough to cover the needed flows for the two data-sets. This quantitative lack of data limits the power of the prototype, since the data cannot reflect real-world data enough. Since only the first Layer of the prototype could have been tested with other data, it is uncertain that the second Layer is precise in detecting malicious DoH traffic or not. The ML models performed very good with the available data, but there is no evidence for the usefulness of the prototype using real-world data. Additionally, the malicious DoH traffic was gathered using DoH tunnel tools, but it is also possible to use the DoH protocol without tunnel tools. Thus, this is a further limitation of the prototype and the assumption can be made that the second Layer could also be imprecise when using other data than the data from the data-set [39].

The evaluation of the ML models showed that the prototype basically performs good while the data-sets were analyzed separately, the major problem of the project is the lack of data. In the future, the data-sets could be further enhanced. This means that the data should not only be lab-generated, but also generated in the real world. Another point is that the detection of malicious DoH traffic could become more precise if the data is gathered only from the communication with one resolver and one browser setting at once, since the evaluation using two different data-sets for training and testing revealed predicting constraints although

both the data-sets contain DoH data. Further, to make especially the malicious traffic more reliable, either the data should be generated by readjusting real cyber-attacks or even using data of officially verified cyber-attacks using DoH should be used to bring the data as near to real-world scenarios as possible.

Bibliography

[1] Ghacksnews, "Mozilla plans to roll out dns over https to us users in late september 2019 #firefox #doh #dns #httpshttps://t.co/yoo2xkf1th pic.twitter.com/deyouvedov," Sep 2019, accessed on January 17, 2022. [Online]. Available: https://twitter.com/ghacks/status/1170365128874897410

[2] S. García, K. Hynek, D. Vekshin, T. Čejka, and A. Wasicek, "Large scale measurement on the adoption of encrypted dns," *arXiv preprint arXiv:2107.04436*, 2021.

[3] K. Hynek, "The prevalence of dns over https," Sep 2021, accessed on January 17, 2022. [Online]. Available: https://blog.apnic.net/2021/09/13/the-prevalence-of-dns-over-https/

[4] "Adopting encrypted dns in enterprise environments," Jan 2021, accessed on January 17, 2022. [Online]. Available: https://media.defense.gov/2021/Jan/14/2002564889/-1/-1/0/CSI_ADOPTING_ENCRYPTED_DNS_U_OO_102904_21.PDF

[5] A. Turing and G. Ye, "An analysis of godlua backdoor," July 2019, accessed on January 17, 2022. [Online]. Available: https://blog.netlab.360.com/an-analysis-of-godlua-backdoor-en/

[6] C. Cimpanu, "Iranian hacker group becomes first known apt to weaponize dns-over-https (doh)," August 2020, accessed on January 17, 2022. [Online]. Available: https://www.zdnet.com/article/iranian-hacker-group-becomes-first-known-apt-to-weaponize-dns-over-https-doh/

[7] Arno0x, "Dnsexfiltrator," Dec 2017, accessed on January 17, 2022. [Online]. Available: https://github.com/Arno0x/DNSExfiltrator

[8] M. Franco, J. Von der Assen, L. Boillat, C. Killer, B. Rodrigues, E. J. Scheid, L. Granville, and B. Stiller, "Secgrid: a visual system for the analysis and ml-based classification of cyberattack traffic," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*. IEEE, 2021, pp. 140–147.

[9] P. V. Mockapetris, "Rfc1035: Domain names-implementation and specification," 1987.

[10] T. H. Kim and D. Reeves, "A survey of domain name system vulnerabilities and attacks," *Journal of Surveillance, Security and Safety*, vol. 1, no. 1, pp. 34–60, 2020.

[11] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc2616: Hypertext transfer protocol – http/1.1," 1999.

[12] D. Borman, B. Braden, V. Jacobson, and R. Scheffenegger, "Tcp extensions for high performance," *RFC7323*, 2014.

[13] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: John Wiley and Sons Inc, 2013.

[14] E. Rescorla, "Rfc2818: Http over tls," 2000.

[15] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (ssl) protocol version 3.0," RFC 6101, Tech. Rep., 2011.

[16] T. Dierks and C. Allen, "Rfc2246: The tls protocol version 1.0," 1999.

[17] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the "s" in https," ser. CoNEXT '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 133–140. [Online]. Available: https://doi.org/10.1145/2674005.2674991

[18] G. Coulouris, J. Dollimore, T. Kindberg, and G. Blair, *Distributed Systems*, 5th ed. Boston, MA, USA: Addison-Wesley, 2012.

[19] S. Calzavara, R. Focardi, M. Nemec, A. Rabitti, and M. Squarcina, "Postcards from the post-http world: Amplification of https vulnerabilities in the web ecosystem," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 281–298.

[20] G. Savary, "Tcp series #2: How to close tcp sessions and diagnose disconnections?" April 2017, accessed on February 23, 2022. [Online]. Available: https://accedian.com/blog/close-tcp-sessions-diagnose-disconnections/

[21] P. Hoffman and P. McManus, "Rfc 8484: Dns queries over https (doh)," 2018.

[22] P. Hoffman, "Representing dns messages in json," 2018.

[23] T. Böttger, F. Cuadrado, G. Antichi, E. L. Fernandes, G. Tyson, I. Castro, and S. Uhlig, "An empirical study of the cost of dns-over-https," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 15–21.

[24] N. Z, "Dns security and privacy — choosing the right provider," April 2018, accessed on February 23, 2022. [Online]. Available: https://medium.com/@nykolas.z/dns-security-and-privacy-choosing-the-right-provider-61fc6d54b986

[25] C. Lu, B. Liu, Z. Li, S. Hao, H. Duan, M. Zhang, C. Leng, Y. Liu, Z. Zhang, and J. Wu, "An end-to-end, large-scale measurement of dns-over-encryption: How far have we come?" in *Proceedings of the Internet Measurement Conference*, ser. IMC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 22–35. [Online]. Available: https://doi.org/10.1145/3355369.3355580

[26] K. Bumanglag and H. Kettani, "On the impact of dns over https paradigm on cyber systems," in *2020 3rd International Conference on Information and Computer Technologies (ICICT)*, 2020, pp. 494–499.

[27] "Lua," accessed on February 24, 2022. [Online]. Available: https://www.lua.org/

[28] F. Daragon, "Lua web application security vulnerabilities," May 2014, accessed on February 24, 2022. [Online]. Available: https://www.syhunt.com/en/index.php?n=Articles.LuaVulnerabilities

[29] J. Armer, "Psixbot infostealer uses dns over https," December 2019, accessed on February 26, 2022. [Online]. Available: https://www.infoblox.com/wp-content/uploads/threat-intelligence-report-psixbot-infostealer-uses-dns-over-https.pdf

[30] P. T. I. TEAM, "Psixbot now using google dns over https and possible new sexploitation module," Sepember 2019, accessed on February 26, 2022. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/psixbot-now-using-google-dns-over-https-and-possible-new-sexploitation-module

[31] T. Keary, "Pcap: Packet capture, what it is & what you need to know," May 2021, accessed on February 24, 2022. [Online]. Available: https://www.comparitech.com/net-admin/pcap-guide/#:~:text=pcap%20files%20to%20record%20packet,you%20can%20view%20through%20Wireshark.

[32] "Wireshark," accessed on February 26, 2022. [Online]. Available: https://www.wireshark.org/

[33] Q. Huang, D. Chang, and Z. Li, "A comprehensive study of {DNS-over-HTTPS} downgrade attack," in *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*, 2020.

[34] D. Hjelm, "A new needle and haystack: Detecting dns over https usage," *SANS Institute, Information Security Reading Room*, 2019.

[35] D. Vekshin, K. Hynek, and T. Cejka, "Doh insight: Detecting dns over https by machine learning," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–8.

[36] M. MontazeriShatoori, "An anomaly detection framework for dns-over-https (doh) tunnel using time-series analysis," Ph.D. dissertation, University of New Brunswick., 2020.

[37] ahlashkari, "Dohlyzer," Nov 2019, accessed on February 10, 2022. [Online]. Available: https://github.com/ahlashkari/DoHlyzer

[38] P. Biondi, "Scapy project," 2021, accessed on February 10, 2022. [Online]. Available: https://scapy.net/

[39] "Cira-cic-dohbrw-2020," accessed on February 10, 2022. [Online]. Available: https://www.unb.ca/cic/datasets/dohbrw-2020.html

[40] S. K. Singh and P. K. Roy, "Detecting malicious dns over https traffic using machine learning," in *2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)*. IEEE, 2020, pp. 1–6.

[41] Y. M. Banadaki, "Detecting malicious dns over https traffic in domain name system using machine learning classifiers," *Journal of Computer Sciences and Applications*, vol. 8, no. 2, pp. 46–55, 2020.

[42] M. Behnke, N. Briner, D. Cullen, K. Schwerdtfeger, J. Warren, R. Basnet, and T. Doleck, "Feature engineering and machine learning model comparison for malicious activity detection in the dns-over-https protocol," *IEEE Access*, vol. 9, pp. 129 902–129 916, 2021.

[43] R. Mitsuhashi, A. Satoh, Y. Jin, K. Iida, T. Shinagawa, and Y. Takai, "Identifying malicious dns tunnel tools from doh traffic using hierarchical machine learning classification," in *International Conference on Information Security*. Springer, 2021, pp. 238–256.

[44] M. Al-Fawa'reh, Z. Ashi, and M. T. Jafar, "Detecting malicious dns queries over encrypted tunnels using statistical analysis and bi-directional recurrent neural networks," *Karbala International Journal of Modern Science*, vol. 7, no. 4, p. 4, 2021.

[45] S. M. Ross, *Introdutory Statistics*, 4th ed. Cambridge, MA, USA: Academic Press, 2010.

[46] T. Jain and V. Ohri, *Introdutory Statistics*. Dehli, India: Global Publications Pvt. Ltd., 2021.

[47] A. K. Awasthi, *Statistics*, 1st ed. India: Darbose Inc., 2013.

[48] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[49] Wireshark, "Sll wireshark," 2020, accessed on March 18, 2022. [Online]. Available: https://wiki.wireshark.org/SLL#:~:text=Linux%20cooked%2Dmode%20capture%20(SLL,or%20can't%20be%20used.

[50] Mathjs, "Mathjs api," accessed on March 18, 2022. [Online]. Available: https://api.mathjs.org/

[51] Pandas, "Pandas library," 2022, accessed on March 29, 2022. [Online]. Available: https://pandas.pydata.org/

[52] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[53] LGBM, "Lgbm," 2022, accessed on March 29, 2022. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/

[54] scikit-learn developers, "Feature importance," 2021, accessed on April 8, 2022. [Online]. Available: https://inria.github.io/scikit-learn-mooc/python_scripts/dev_features_importance.html

[55] K. Jeřábek and S. Stuchlý, "Dns over https network traffic," 2021, accessed on March 29, 2022. [Online]. Available: https://dx.doi.org/10.21227/96ea-2055

[56] A. Zheng, *Evaluating Machine Learning Models*, first ed. ed. Sebastopol, CA, USA: O'Reilly Media, Inc, 2015.

[57] avcontentteam, "Tcp series #3: network packet loss, retransmissions, and duplicate acknowledgements," 2016, accessed on April 12, 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/

[58] C. Greer, "Tree based algorithms: A complete tutorial from scratch (in r python)," 2017, accessed on April 12, 2022. [Online]. Available: https://accedian.com/blog/network-packet-loss-retransmissions-and-duplicate-acknowledgements/

[59] J. Von der Assen and L. Bolliat, "Ddosgrid (v2)," 2022, accessed on April 13, 2022. [Online]. Available: https://github.com/ddosgrid/ddosgrid-v2

Abbreviations

ACK	Acknowledgement packet of a TCP connection
BRNN	Bi-Directional Recurrent Neural Network
DL	Deep Learning
DNS	Domain Name System
DoH	DNS over HTTPS
EFB	Exclusive Feature Bundling
FIN	Finish packet of a TCP connection
FN	False Negative
FP	False Positive
GBDT	Gradient Boosting Decision Tree
GOSS	Gradient-based One-Side-Sampling
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IPv4	Internet Protocol version 4
JS	JavaScript
LGBM	Light Gradient Boosting Machine
ML	Machine Learning
PCAP	Packet Capture
PKI	Public Key Infrastructure
RF	Random Forest
ROC	Receiver Operating Characteristics
RST	Reset packet of a TCP connection
SSL	Secure Sockets Layer
SYN	Synchronization packet of a TCP connection
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TN	True Negative
TP	True Positive
TSL1	Traing Set for Layer 1
TSL2	Traing Set for Layer 2
UDP	User Datagram Protocol

Glossary

Random Forest A tree based Machine Learning algorithm.

Retransmission Packet This packet type is to prevent packet loss. If a packet is not acknowledged in a certain time frame, the sender retransmits the packet because it assumes that the packet has been lost [57].

Tree Based Machine Learning Algorithms This is a type of supervised learning algorithm, mostly used for classification, where the population is split into two or more homogeneous sets [58].

List of Figures

List of Tables

List of Algorithms

Appendix A

Installation Guidelines

A.1 Feature Extraction

The Feature Extraction component is located in the Folder "Scource_Code/ddosgrid-v2". It can be installed and run using the installation and setup guidelines of the repository on Github [59].

The project is located on the branch "doh".

A.2 ML Pipeline

The Machine Learning Pipline component is located in the Folder "Source_Code/LGBM". Do the following steps:

- Make sure you use an IDE which is compatible for Python, e.g. *PyCharm*.
- Make sure that every import of the respective document is installed.

The complete pipeline is located in the folder "ml". The required files (training- and testing-data-sets) are already prepared, just run the file *lgbmClassfier.py*.

The evaluations are located in the folder "evaluation". Make sure that the imports are installed. The required files are already prepared, just run the respective file.

The GridSearch is located in the folder "gridSearch". Make sure that the imports are installed. The required files are already prepared, just run the respective file.

Appendix B

Contents of the Repository

The repository contains the following directories:

- **Data-Set**: Contains a file with the URLs where the complete data-sets (CIRA-CIC-DoHBrw-2020 and IEEE-Data-set) can be found.
- **Documentation**: Contains a directory with all the images used for the documentation, a directory with a reference that could not have been accessed with the UZH-VPN, a directory with the thest files used for the evaluation in Section 6.1, and the documentation as PDF-file.
- **Presentations**: Contains a directory with all the images used for the final presentation, the Intermediate Presentation and the Final Presentation as PDF-files.
- **Source_Code**: Contains the two directories ddosgrid-v2 and ml-pipline. ddosgrid-v2 contains the source code of SecGrid with the newly implemented Feature Extraction component. ml-pipeline contains the separate ML pipeline.
- **Train_Test_Files**: Contains all the training and tesing data-set used for the ml-pipeline and the evaluation of it.