Master Thesis

Using Certificate Transparency to detect malware in network telemetry

Bc. Jan Karsch

Supervisor: Ing. Jan Brabec

January 2021

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Karsch Jan**	Personal ID number:	**438028**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Cyber Security**		

II. Master's thesis details

Master's thesis title in English:

Using Certificate Transparency to detect malware in network telemetry

Master's thesis title in Czech:

Použití Certificate Transparency pro detekci malwaru ze síťového provozu

Guidelines:

This thesis is concerned with the use of Certificate Transparency (CT) service
to detect communication related to malware in enriched Netflow telemetry.
The concrete goals are:
1. Study the CT service and the details of X.509 certificates. Review relevant literature concerning the use of Certificate Transparency and other TLS-related information in malware detection.
2. Design features that can be extracted from data in CT and add automated pipeline for enriching of Netflow data with these features to existing security product backend.
3. Use the features in addition to other contextual Netflow data to classify network telemetry for malware. The choice of classifier should be sound from the perspectives of related art and various engineering tradeoffs that will be described in the thesis.
4. Experimentally evaluate the created classification pipeline.
a. Describe and use suitable evaluation scheme for this task.
b. If suitable, evaluate different classifier alternatives.
c. Using suitable methods evaluate the utility of individual features and perform ablation study to determine the added value of CT-based features.

Bibliography / sources:

[1] Laurie, Ben. Certificate transparency. Communications of the ACM, 2014, 57.10: 40-46.
[2] Fasllija, Edona, Hasan Ferit Enişer, and Bernd Prünster. "Phish-Hook: Detecting Phishing Certificates Using Certificate Transparency Logs." International Conference on Security and Privacy in Communication Systems. Springer, Cham, 2019.
[3] Criminisi, Antonio, Jamie Shotton, and Ender Konukoglu. "Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning." Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114 5.6 (2011): 12.
[4] Gustafsson, Josef, et al. "A first look at the CT landscape: Certificate transparency logs in practice." International Conference on Passive and Active Network Measurement. Springer, Cham, 2017.
[5] Anderson, Blake, and David McGrew. "Identifying encrypted malware traffic with contextual flow data." Proceedings of the 2016 ACM workshop on artificial intelligence and security. 2016.

Name and workplace of master’s thesis supervisor:

Ing. Jan Brabec, Department of Computer Science, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **29.07.2020** Deadline for master's thesis submission: ___________

Assignment valid until: **19.02.2022**

Ing. Jan Brabec Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt	Student’s signature

Abstract

The goal of this thesis is to explore Certificate Transparency service and determine whether it can be used as a valuable data source in the field of malware detection. Certificate Transparency serves as an additional layer for source integrity verification in Public Key Infrastructure. The service stores certificates into public logs, publishes an interface for certificate manipulation, and thus can be used as a data source. Append-only nature of logs creates a certificate database which can be subsequently processed and used for extraction of historical data as well as for enhancing existing database of features used for malware detection. Within the scope of this thesis is implemented an algorithm for processing the history of certificates for different hostnames and the extraction of newly designed features which are then analyzed. The final part of this thesis involves a multinomial malware classification with use of a random forest classifier and different sets of features for comparison. Evaluation of the model with new features resulted in improved malware classification and therefore the history of certificates has shown to be a valuable data source.

Keywords: tls protocol, certificate history, certificate transparency, malware, detection, machine learning, random forest

Abstrakt

Cílem této práce je prozkoumat službu Certificate Transparency a zjistit, zda by mohla být použita jako užitečný zdroj dat pro detekci malwaru. Certificate Transparency slouží jako dodatečná vrstva pro ověření integrity zdroje v rámci Public Key Infrastructure. Ukládá certifikáty do veřejných logů, poskytuje rozhraní pro manipulaci a lze ji tedy použít jako zdroj dat. Tím, že do logů lze certifikáty pouze přidávat, vzniká historické uložiště certifikátů, které mohou být následně zpracovány a použity pro extrakci historických dat a obohacení existující databáze příznaků pro detekci malware. Součástí této práce byla implementace algoritmu pro zpracování historie certifikátů pro odlišné hostnamy a extrakce nově navržených příznaků, které byly následně analyzovány. Na závěr je provedena multinomiální klasifikace malwaru s použitím modelu náhodného lesa a různých množin příznaků pro porovnaní. Model obohacený o nově navržené příznaky vykázal zlepšení v klasifikaci malwaru, a ukazuje se tedy, že historie certifikátů je užitečný zdroj dat.

Klíčová slova: tls protokol, historie certifikátů, certificate transparency, malware, detekce, strojové učení, náhodný les

Author statement for graduate thesis:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date ....

signature

Acknowledgements

First of all, I want to thank my supervisor Ing. Jan Brabec for his patience, guidance, overall support, and in general for giving me such a great opportunity to work on an interesting topic as well as be a part of his team.

Also, I would like to thank my dearest friends Jakub, Tomas, and Tomas for always trying to keep me sane, giving me great programming-related advice and general support throughout the studies.

Contents

Chapter 1

Introduction

Securing the Internet is indeed a challenging task and especially when it experiences enormous growth as in the past two decades. Critical internet mechanisms and protocols are regularly updated and evolving over time however by the same token also evolve internet threats which are more and more sophisticated and harder to detect. One of the solutions is the employment of Machine Learning for network log monitoring and threat detection in action.

Nevertheless, with evolving security mechanisms, general requirements of endpoint users have also developed, and thus it became almost a standard to apply encryption to ensure users' privacy whenever it is possible [1]. We are here referring primarily to **HTTP protocol** responsible for the majority of the Internet network traffic with **SSL/TLS protocol** as its main security enhancement ensuring confidentiality and integrity in terms of cybersecurity goals. With that in mind, all malevolent internet activities are way harder to detect as it is not possible to work with the message content itself and therefore adversaries also benefit from encrypted traffic because their actions can remain unseen. It is then appropriate to adopt and utilize different useful (meta)data sources such as **TLS certificates**.

An essential part of the TLS protocol is the so-called **Public Key Infrastructure** which brings a concept of trust and methods for authenticity verification of communicating sides to the whole system by implementing TLS certificates. Hand in hand with certificates also come so-called certificate-based threats like certificate misuse or compromised authorities. This system is built on the basis of general trust in third parties and whenever such a trusted party is compromised, the whole system is compromised as well. Exactly as have happened in 2011 when one of the trusted parties has been compromised and fraudulently issued a certificate for Google hostnames. As a result, Google has decided [2] to implement **Certificate Transparency** serving as an enhancement and additional verification layer for existing Public Key Infrastructure which deals with and mitigates previously mentioned certificate-based threats. Certificate Transparency is the main focus of this thesis primarily from the perspective of data source out of which might possibly be extracted various useful information for malware detection on the top of already existing ones. Therefore, the goal of this thesis is to enhance existing data that are already used for malware detection.

This thesis is structured into several chapters. In the very first **Chapter 2** we will dive into TLS protocol with a description of its main used mechanisms starting with encryption, followed by so-called TLS handshake and description of Public Key Infrastructure.

Chapter 3 thoroughly elaborates Certificate Transparency service with all used mechanisms, infrastructural components (certificate log, monitor, auditor) and internal processes.

Chapter 4 defines our general task from the perspective of machine learning and malware detection. This chapter outlines the problem of classification and model complexity with different evaluation metrics. As our goal is to extract new information useful for detection, it contains a description of the current state and covers the very first draft of designed features that could be extracted from Certificate Transparency.

In **Chapter 5** we will go through an extraction pipeline that has been implemented for Spark distributed environment suited for handling of big-data. Part of this chapter is a description of encountered obstacles and the final list of extracted features from Certificate Transparency.

Analysis of extracted data is part of **Chapter 6**. Besides global statistics, this chapter depicts a difference between malware and non-malware network data from the perspective of extracted information by comparing their data distributions plots.

The very last **Chapter 7** contains a description of the selected machine learning classification model as well as a final analysis of performed malware detection experiments with corresponding results.

Chapter 2

TLS protocol

More and more content is being served in encrypted form and special mechanisms have to be present to ensure that communication between parties is secured. In terms of cybersecurity, we aim to satisfy three main goals known under shortcut **CIA** which stands for **Confidentiality**, **Integrity** and **Availability**.

Confidentiality is a state when information is being available only to those subjects it was intended to be. Therefore, it represents privacy. Integrity represents a state when the transmitted information must be resistant to any unwanted modification or at least interested parties must know about such tampering actions and from a practical perspective, there has to be some detection mechanism. When going a bit deeper, and especially in the case of TLS protocol, we have to distinguish between data integrity and source integrity. Data integrity covers the state of transmitted data however source integrity ensures that in our system has not happened any modification of origin subject, therefore we always want to know who is the communicating party on the other side and also that this party is the one we want to actually exchange information with. Availability represents the state when the information which authorized subjects want to access, is available when needed [3].

Out of the three described general cybersecurity goals, even though availability is an important criterion, the main interests of this thesis are confidentiality and integrity, because that is essentially what TLS protocol aims for. Confidentiality, secrecy, and privacy are practically ensured by **data encryption**. Source integrity is ensured by **certificates** and *Public Key Infrastructure (PKI)*, data integrity by hashing in general and with so-called message authentication mechanisms.

Transport Layer Security (TLS), it is also often referred to as *Secure Socket Layer (SSL)*, which is its predeceasing version, is a widely used protocol providing privacy and data security in internet communication. TLS is currently in version TLSv1.3 published in 2018, the first SSL version was defined back in 1999. From a networking perspective, there is well-known *OSI reference model* which defines distinguish layers and hierarchy between different protocols [4]. This text will not go deeper, however, practical and key thing to note is that TLS sits on top of the Transporation layer (*TCP*[5] and *UDP*[6] protocols) and encapsulates incoming traffic from the above Application layer. Therefore

TLS is bound to Application layer protocols such as *HTTP*, *IMAP*, and many others.

For purpose of this thesis, the primary focus is put on HTTP, a very common and well-known Application layer protocol running over TCP, in combination with TLS it is called *HTTPS*, more in Section 2.3 .

This thesis builds primarily on the Certificate Transparency chapter (Chapter 3), nevertheless, it was critical to understand inner TLS concepts to fulfill the objectives of this thesis. That means, the scope of this chapter essentially covers practical aspects of TLS and implemented mechanisms in such a way, so we have no problem with understanding the following chapters. TLS communication has two main components: *handshake protocol* and *record protocol*. TLS works with modern and complex cryptographic mechanisms to ensure confidentiality, thus both symmetric and asymmetric cryptosystems are used and as was previously mentioned, hashing functions with message authentication mechanisms are also included to ensure data integrity.

2.1 TLS Handshake and Record protocol

Critical part of the TLS protocol is authentication of both communicating sides. Private and secure communication is provided by encryption, however, the question is: How two parties, unknown to each other, can start encrypting traffic and exchange any kind of information in a secure way? The usually known model is that ciphers take some secret key as an input, which is known to both parties. But in this situation, an encryption key in unencrypted (plain) form cannot be exchanged as it could be sniffed[1] by an adversary. That would be the case of so-called *symmetric encryption* which uses only one key for encryption as well as for decryption of the data. The solution for this is to use *asymmetric encryption*, which uses different keys for encryption and decryption.

2.1.1 Asymmetric encryption

Asymmetric ciphers allow us to use two different keys, public p_e and private p_d for encryption and decryption. If we define *enc* and *dec* as operations for encryption and decryption in asymmetric cipher with input data a and previous keys p_e and p_d, the following can be written

$$dec(p_d, enc(p_e, a)) = dec(p_e, (enc(p_d, a)) = a$$

thus, both keys in the same operation behave as inverse functions to each other. Take into account that this is extremely simplified and this thesis will not go into a more detailed explanation as there are also various different asymmetric ciphers, each based on a different mathematical principle.

Anyway, a requirement for such cipher is to have two different paired keys, public and private, whilst the public one is visible to anyone but the private key is always kept as a secret. Hence, it became a standard to use mathematical methods based on so-called

[1] In network security, sniffing stands capturing network traffic

hard problems which ensure that with knowledge of only a public key, it would cost an extensive amount of computational power to actually find matching private key with *brute-force*[2] or guessing methods. Among such hard problems belong *Integer factorization* used in *RSA cryptosystem* [7], *Discrete logarithm in cyclic groups* used in *Diffie-Hellman key exchange* [8] and last but not least, cryptography build on *Elliptic curves* [9].

The downside of asymmetric encryption is that it consumes and requires a lot of computational power because hard problems usually require longer keys to be "hard enough". For instance, RSA uses a key of length 2048 bits but AES[3] works with a key length of 128 bits. Therefore, TLS protocol uses in handshake asymmetric encryption only to negotiate and exchange initial information and then continues with a symmetric bulk cipher, we omit here asymmetric encryption used in certificates.

2.1.2 Symmetric encryption

Symmetric ciphers use the same key for data encryption and decryption and from the top level, two major categories are recognized: *Block ciphers* and *Stream ciphers* [10].

Block ciphers

Block ciphers take the input plaintext data and convert it into ciphertext by taking and encrypting text blocks of a specified length. There are several modes of operation, also known as chaining modes, which define and affect how are block fragments chained during encryption. The most trivial one is *Electronic Code Book (ECB)* mode that takes one block after another and encrypts it independently on others. The simplicity of this mode creates a potential attack vector as each data block is independent of others. That also means blocks of the same exact values will be encrypted into the same exact blocks of ciphertext and therefore leave some information about the plaintext. This specific mode is mentioned primarily for its simplicity and to point out how critical parameter it actually is.

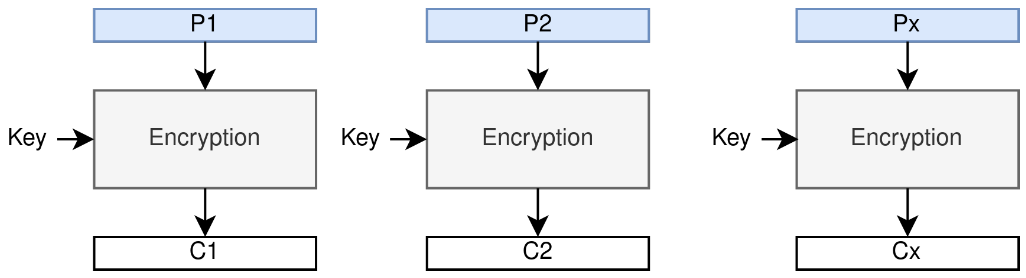


Figure 2.1: Encryption with Electronic Code Book (ECB) chaining mode, P1-Px represent blocks of plaintext and C1-Cx corresponding cipher text blocks

TLS v1.2 [11] used *Cipher Block Chaining (CBC)* mode, TLS v1.3 [4] however dropped its support, due to vulnerabilities coming from *padding oracle attacks* [12], and prefers so-

[2]Brute-force methods generate and try every single possible key from a keyspace.
[3]Advanced Encryption Standard (AES)

called *Galois/Counter (GCM)* mode. The advantage of GCM mode is that it also already provides data integrity verification. In the case of CBC chaining, it would be necessary to use additional *Message Authentication Code (MAC)* such as *CBC-MAC* or *HMAC*. Essentially, this method creates another block fragment for message verification and combines it with the ciphertext or the plaintext (based on the selected type) which serves as a data integrity proof of the whole transmitted data message [13].

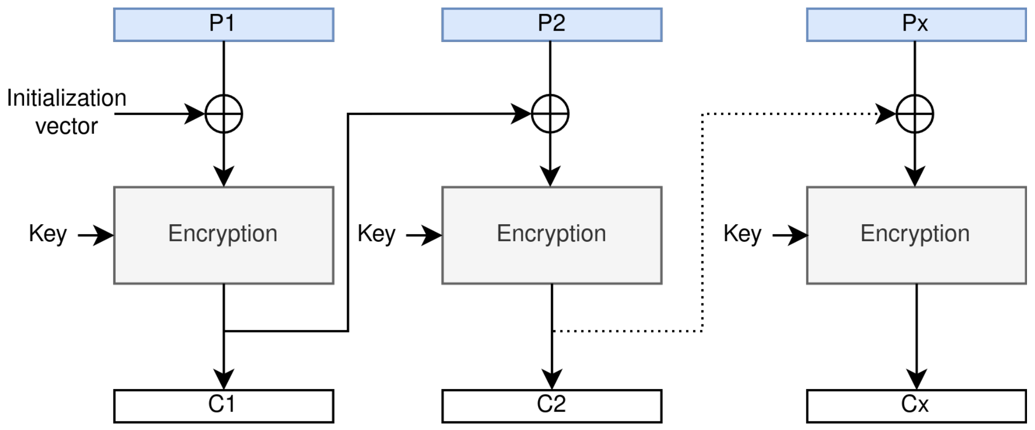


Figure 2.2: Encryption with Cipher Block Chaining (CBC) chaining mode, P1-Px represent blocks of plaintext and C1-Cx corresponding cipher text blocks

Throughout the years, as encryption standards and requirements developed, recommended encryption algorithms have also changed. For block ciphers, it was *Data Encryption Standard (DES)* cipher, developed in the 1970s with a key length of 56 bits, which is for modern computational standards too short and could be easily brute-forced (Space of 2^{56} possible keys). Based on that, in the 1990s *Triple DES (3DES)* was introduced and then broken with *Meet-in-the-middle attack*. NIST[4] has subsequently announced a competition in 1997 for a new encryption standard that should substitute DES for its brute-force vulnerabilities. The winning cipher was *Rjindael* and it became a new *Advanced encryption standard (AES)* until now [14, 15]. The size of the AES encryption key is 128, 192, or 256 bits with a block size of 128 bits.

Stream ciphers

One-time pad is considered to be a perfect cipher, it applies a completely random key, with the same length as a plain text, that can not be used more than once. To create a ciphertext, the One-time pad performs XOR operation with the key against the plain text.

Stream ciphers convert plain text into cipher text bit by bit. However, from nowadays' perspective, it is not possible to have a key with a length of the plain text, change it for every single transmitted message and use another communication channel to transmit the encryption key itself.

[4]National Institute of Standards and Technology of United States

Stream ciphers use a key of the fixed length and implement pseudorandom generators that generate so-called *key stream* with the same length as the input plain text, communicating parties then have to synchronize those generators, and based on that, there is no need to communicate generated keys. Conversion to ciphertext is exact same as in the case of One-time pad, XOR operation is performed between the generated keystream and plain text [10].

2.1.3 TLS Handshake

In the following text, we aim to cover both of the two main initial TLS parts: handshake and record protocol and describe, in a practical manner, every performed step. The most recent version of TLS is v1.3, older version v1.2 is still supported by TLS clients (Internet browsers in our case), versions v1.0 and v1.1 are both however deprecated and should not be used anymore. TLS handshake, among other things, differs across versions and the following text will outline only the most recent version v1.3 which became a new standard in 2018, and proposes a general improvement in performance and security in comparison with the previous version v1.2 [13, 16].

Handshake protocol

Handshake protocol serves as a mechanism for authentication of both communicating parties. It is responsible for negotiating necessary cryptographic parameters such as modes, keys, and authentication of both parties. [4]

The text further continues with a description of handshake v1.3 steps (*TCP handshake* (SYN, SYN-ACK, ACK) is omitted).

1. ClientHello

The client creates a request and initiates the connection, further provides the following information fields.

- Supported versions of TLS
- Client random data
- Supported Cipher Suites both for authentication and for key exchange
- Key Share is a list of public keys the client assumes the server will support. It allows all exchanged messages after ServerHello to be encrypted. This is actually one of the optimizations that took place in v1.3 because the server can start encrypting 1-RT[5] earlier.
- *Server Name Identification (SNI)* is the name of the destination server In a situation when a server hosts multiple virtual servers on a single IP address, without this information, it would not be able to tell which one to contact.
- Extensions

[5]One Round Trip - the amount of time between sending the data and getting an acknowledgment signal that it was received.

2. ServerHello

Response to the hello message from the client. The message consists of the following fields.

- Server Random Data
- Protocol version negotiated
- Selected cipher suite - Ciphers which will be used for symmetric encryption and hashing function for message authentication in Record Protocol
- Key Share - Public key sent by the server. The client can start encrypting after receiving this information. This is the security improvement in v1.3 over earlier TLS versions, where the whole handshake was exchanged in plaintext.
- Extensions

From now and on, communication is encrypted using the symmetric bulk cipher and MAC.

3. Encrypted extensions sent by server

Additional TLS extensions sent by the server that are not related to parameter negotiation.

4. X.509 Certificates for server authentication

X.509 digital certificate sent by the server for authentication and proving source integrity, more in Subsection 2.2.1.

5. Certificate Verify

The server must prove that it owns the sent certificate from the previous step and has a matching private key to the public one included in X.509 certificate. The server will sign (encrypt) hash of handshake messages, the client can then verify (decrypt) this using server's public key from the certificate.

6. Server Handshake Finished

Verification of the whole handshake from the server-side. It uses a hash function on all handshake messages and verifies that there hasn't happened any tampering. Take into account that both sides sent Random Data in client and server hello. That is a critical component of initial unencrypted communication because it prevents possible replay attacks as those generated random data are unique to this session and keys.

7. Client Handshake Finished

Same as above, this is the verification that the TLS handshake was successful. The client calculates the hash of all transmitted handshake messages.

Handshake finished, both parties have authenticated each other and encrypted data exchange can begin in Record protocol

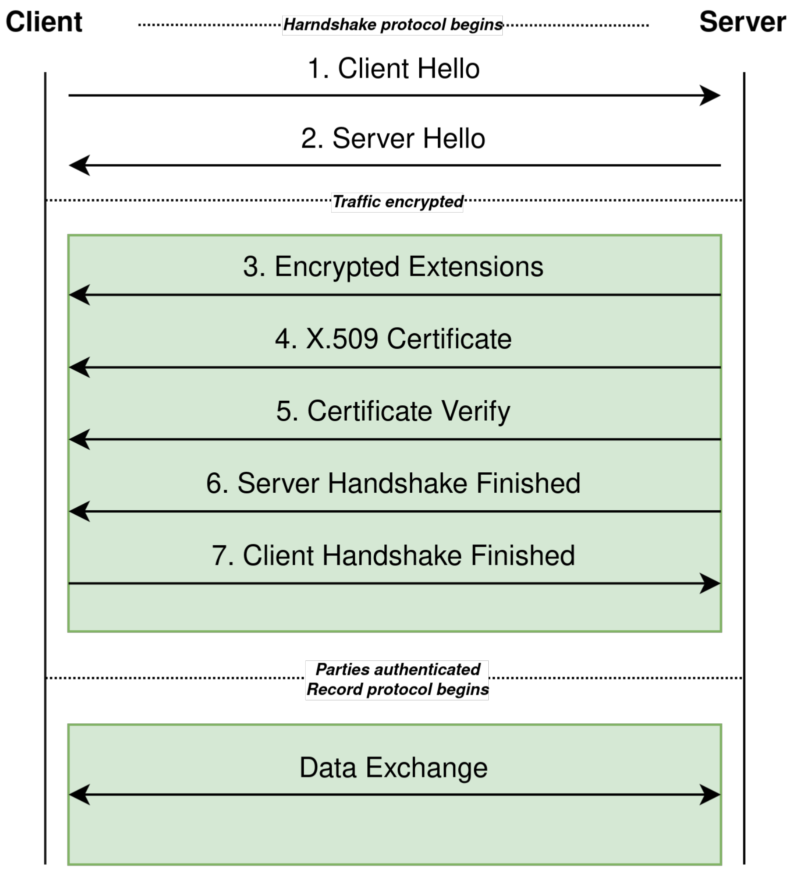


Figure 2.3: TLS handshake diagram

Record protocol

Record protocol is initiated right after the handshake protocol when parties have exchanged every necessary information. This protocol uses negotiated parameters and secures transmitted network traffic with a symmetric cipher. Record protocol also manages all messages that are about to be transmitted and divides them into blocks [4]. In the previous text, we have already mentioned that during transmission, data integrity has to be assured with *Message Authentication Code (MAC)* mechanisms such as *Hash-based message authentication code (HMAC)* or specific chaining mode such as GCM with *Galois Message Authentication Code (GMAC)* that already provides integrity check.

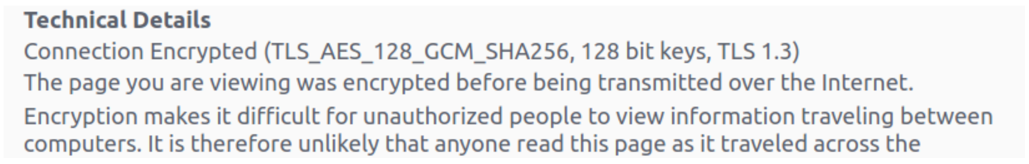

Technical Details
Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the

Figure 2.4: Negotiated cipher suite in Firefox internet browser on visiting google.com

On Figure 2.4 we can see negotiated cipher suite on visiting *google.com* hostname with the internet browser. Browsers bring many useful information regarding to general internet security and the negotiated suite is read as follows.

TLS - Protocol used

AES_128_GCM - Advanced Encryption Standard used for symmetric encryption with block size and key length of 128 bits in Galois/Counter mode.

SHA256 - Hashing function used during the handshake

TLS 1.3 - Version of TLS used

2.2 Public Key Infrastructure

So far, prime focus has been put on confidentiality (encryption) and data integrity (message authentication) of TLS protocol. In the very beginning of this section, there has however been defined also a need for source integrity, in TLS implemented by so-called certificates that are exchanged as a part of the previously described handshake.

One party needs proof that it is actually communicating with the party it wants and that is the purpose of *Public Key Infrastructure (PKI)* and *TLS/SSL certificates*, also referred to as *X.509 certificates* [17]. And yet again, asymmetric cryptography is a solution to this problem. When using asymmetric cryptosystems for encryption, the public key of the receiving party is used for message encryption, thus the sending party known only the receiving party can decrypt the message with its private key (the private key is kept as a secret).

What would however happen if both keys are switched with each other if the private key is used for "encryption" of the data? Confidentiality is immediately broken because public key is not kept as a secret, therefore anyone can decrypt the message. Nevertheless, as was described, one party needs proof that incoming data are from the right source, and encrypting messages with this party's private key is exactly what serves this purpose. The receiving party can then use the public key of the sending party, decrypt the data and by doing so, also verify that the sending party is the right one, because only the sending party has a matching private key, and thus, source integrity is also ensured. In TLS handshake, this functionality takes place in Certificate Verify step (Section 2.1.3).

Practically speaking, to be actually able to use asymmetric cryptography for source integrity verification, communicating parties need to store and in general, be able to obtain public keys of other parties. TLS clients apparently cannot handle all existing public keys and so was built a technology called *Public Key Infrastructure (PKI)*. This complex system comprises numerous policies, rules, and infrastructural components for management of certificates and cryptography based on public keys. For the purpose above, when clients need to have general access to public keys, a major requirement is transparency and general trust in this third-party system [17]. The following text of this section outlines the major processes and components on which PKI is built.

2.2.1 X.509 certificate

X.509 certificate is a digital document and a set of information that serves as an identification of a given subject during the authentication. It is created by a trusted party, a critical component in PKI called *Issuer* or *Certificate Authority (CA)* (Subsection 2.2.2) after *Certificate Signing Request* (Section 2.2.2). The certificate consists of the fields below [18].

Version - Certificate version (3 - X.509v3)
Serial Number - Unique identifier of certificate by an issuer
Issuer - Subject who created and manages certificate (Certificate Authority)
Public Key - Key for asymmetric encryption with corresponding information about used algorithm and key length.
Signature - Signature of the certificate with corresponding information about the used algorithm.
Not Before - Date of issuance
Not After - Date of expiration
Subject - Various information fields about subject who requested certificate.
Extensions - Optional and additional data

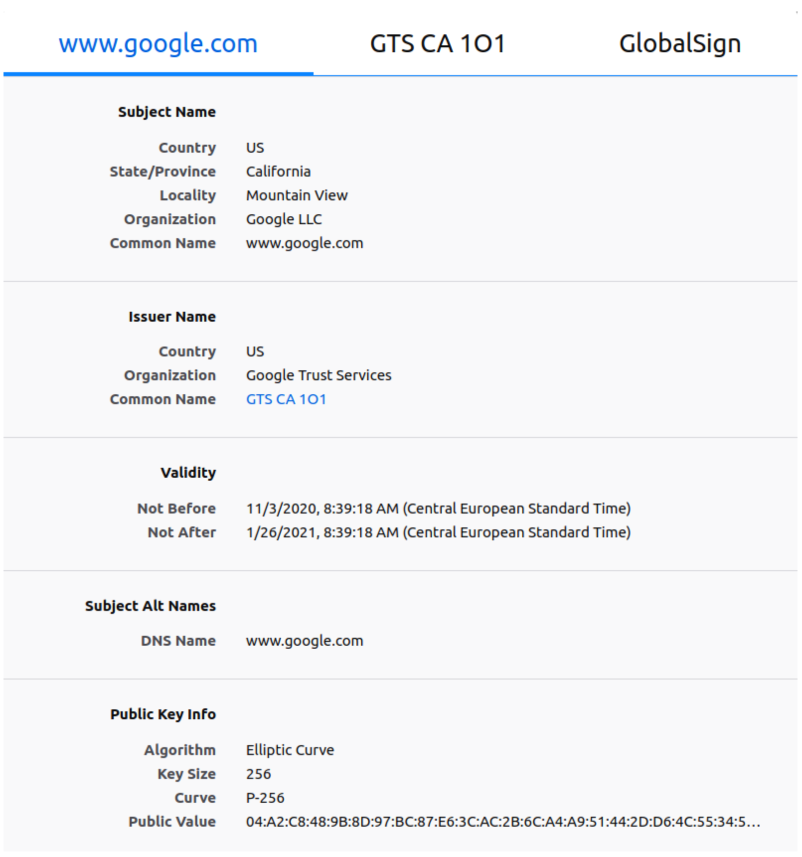
During the TLS Handshake protocol, there is a phase, where both sides exchange certificates to authenticate each other. In most of the cases, only the client-side does so (Section 2.1.3), there is however a possibility to extend the handshake with Client certificate request and subsequently, in response with Client Certificate, that is also known as *Two-Way TLS* [19].

As in the case of the negotiated cipher suite, internet browsers allow us to take a look into the downloaded certificate on a visited hostname. On the Figure 2.5 we can see part of the certificate for *google.com* hostname (As of December 2020). Except for the information fields that are described above, on the very top of the figure, we can see 3 different tabs, each for a different certificate regarding to *Chain of Trust*.

www.google.com - End certificate for a given hostname
GTS CA 1O1 - Intermediate authority that issued the end certificate for the hostname
GlobalSign - Root Certificate authority which certifies the intermediate one.

2.2.2 Certificate Authority

Certificate verification process needs a trusted parties that store, issue and manage certificates in general. These trusted parties are in PKI called *Issuers* or *Certificate Authorities (CA)*. Also, there exist so-called *Root Certificate Authorities*, *Intermediate Authorities* and *Subordinate Authorities*, together they create a *Chain of trust* where one authority is certified by another (Subordinate/Intermediate/Root). On top of this hierarchy are Root authorities which are trusted by clients, in the case of internet browsers usually after satisfying defined standards and requirements.



www.google.com | GTS CA 1O1 | GlobalSign

Subject Name	
Country	US
State/Province	California
Locality	Mountain View
Organization	Google LLC
Common Name	www.google.com
Issuer Name	
Country	US
Organization	Google Trust Services
Common Name	GTS CA 1O1
Validity	
Not Before	11/3/2020, 8:39:18 AM (Central European Standard Time)
Not After	1/26/2021, 8:39:18 AM (Central European Standard Time)
Subject Alt Names	
DNS Name	www.google.com
Public Key Info	
Algorithm	Elliptic Curve
Key Size	256
Curve	P-256
Public Value	04:A2:C8:48:9B:8D:97:BC:87:E6:3C:AC:2B:6C:A4:A9:51:44:2D:D6:4C:55:34:5…

Figure 2.5: Certificate view in Firefox internet browser on visiting google.com

During the TLS handshake (Figure 2.3), after receiving Certificate and Certificate verify messages, the TLS client (internet browser) does a sequence of operations to perform a source verification check (More in following Section 2.3). Part of this verification process is also Certificate Authority check from a received certificate. Each certificate is accompanied by information about the issuer, also with a digital signature created by this issuer. The client builds certification path consisting of the root authority certificate, certificates from all intermediate authorities and the downmost certificate of the server in the chain [20].

Certificate Signing Request

For successful registration of a public key, the subject has to create *Certificate Signing Request (CSR)* [21]. Matching public and private key pair has to be generated, then the subject can proceed and create the CSR. It has to attach the following fields to the request.

- Public Key
- Common Name
- Organization Name
- Organization Unit
- City
- State

- Country
- Email address

However, not all the fields above are mandatory as it varies from issuer to issuer.

Certificate revocation

Whenever a certificate has been compromised before the expiration, it should be revoked and put on a so-called *Certificate Revocation List (CRL)* which is held by the Certificate Authority. This is a list of certificates that should not be used anymore [17].

2.3 HTTPS and Internet Browsers

Hypertext Transfer Protocol (HTTP) [22] is an Application layer protocol responsible for common internet communication and serving of hypertext data. HTTP runs on the top of TCP protocol and with an extension of TLS, it becomes HTTPS [23], whilst browsers are HTTPS clients that request and fetch resources from servers. In the plain HTTP protocol, every transmitted data packet is readable, thus confidentiality and integrity are not ensured at all. As an extensive amount of internet content is being served, TLS-like protocol enhancement is necessary.

In our case, Internet browsers serve as TLS clients. They keep a list of trusted Root Certificate Authorities and during the TLS handshake perform multiple server authentication steps described in the text below. Also note, as TLS certificates are built on *the Chain of Trust* model, TLS clients have to build a verification path including certificates of each authority, starting with the root CA to the downmost leaf one [20].

Integrity - Verification of the Certificate Signature. Each certificate contains a signature from the issuer and root authorities use so-called *self-signed* certificates because they are globally trusted and issue certificates for themselves.

Validity - Current timestamp has to be between Not Before and Not After fields

Revocation Status - Certificate cannot appear on the Certificate Revocation List

Issuer - In the verification path, each certificate has an issuer field, which has to be the same as the subject field in the upper certificate. This check includes verification of public keys as the signature on a certificate is created by its issuer.

Other policies and constraints (Path length, Purpose of used keys, etc.)

Chapter 3

Certificate Transparency

Throughout the years, in TLS protocol have been exposed various vulnerabilities in cipher chaining modes but also in the whole TLS infrastructure on which are built certificate-based threats like certificate misuse or forged certificates. Designing a perfectly safe and secure protocol is indeed a challenging task. In particular, common protocols like TLS have high-security demands and therefore it is difficult to implement new functionalities and enhance existing architecture with additional components. This may however result in a situation when third parties join the game and start developing this new and additional component by themselves and also enforce it on all involved subjects.

In 2013, Google has decided to fix the TLS protocol and its infrastructural flaws by implementing so-called *Certificate Transparency (CT)* service [24, 25]. As a very well known and respected entity in the communication field, also with their own and widespread internet browser Chrome (with more than 60% users [26]), Google has enough resources for enforcing new networking rules and policies.

Certificate Transparency focuses on fixing flaws in the TLS certificate system and in HTTPS protocol where TLS takes place as a main underlying cryptography component. Just as TLS certificates and Public Key Infrastructure in general, CT aims to be transparent and open system for auditing and logging of certificates in almost real-time. It helps to identify certificates that have been issued by mistake or for example, it detects issuers that could have been compromised or gone rogue.

To detect forged SSL certificates, internet browsers already have implemented mechanisms that ask the origin server for its X.509 certificate and then verifying multiple certificate fields, as was described in Section 2.3. Nevertheless, this system still stands or falls on a single point of failure, which is trust in Certificate Authority, who we can think of as a maintainer (and also issuer) of a given certificate. When such authority is compromised, as has happened for example in the case of **DigiNotar** [27] back in 2011, it is impossible to take immediate security precautions and measures. Adversaries then have enough time to start issuing fraudulent certificates without anyone knowing that it is actually happening, including domain owners. The issuer, as a trusted party, then can forge completely new certificate for literally any domain. In DigiNotar case, it was a wildcard certificate for ***.google.com**. The certificate was used to perform MITM[1] attack against Google and it took more than a month until there were detected first certificate problems. Thus, Certificate Transparency aims for nearly real-time monitoring so the detection of malevolent behavior doesn't take numerous weeks but at max a few days [28].

3.1 Purpose of Certificate Transparency

The general purpose of Certificate Transparency is to fix TLS protocol flaws that could possibly jeopardize any of the involved parties. The first party that can come into mind are endpoint users, who access and consume content by using internet browsers and are utterly dependant on the whole designed system. Therefore, core functionality and support for Certificate Transparency has to be implemented in internet browsers by default. In terms of asymmetric cryptography in practice, the user's main requirement is to have a secure and fully working source integrity verification and encrypted communication. Given user thus always knows who is he communicating with and is assured that there is no possible way any unwanted party could read ongoing communication.

The second engaging party in the whole certificate verification process are origin servers (domain owners) that serve internet content to endpoint users and are also requested to send X.509 certificate to them so there can be secured communication between both.

Last but not least, we have certificate authorities, creators, and issuers of certificates, who should remain the most trusted party in the TLS system as they are prompted by browsers for verification of certificates which they originally issued. Even though endpoint users and domain owners are indeed relevant parties, Certificate Transparency service puts most of the focus on certificate-based threats coming primarily from crooked certificate authorities.

We have three major participating parties in our system. Attacks such as impersonation or man in the middle are undoubtedly real danger for Internet users. It also needs to be said however, certificate based-threats both defame and financially damage domain owners as well as certificate authorities.

Certificate Transparency service has three main goals regarding each engaging party [28].

- Certificate Authority can not issue a TLS certificate without the certificate being visible and transparent to the given domain owner.
- Provide thoroughly transparent and open auditing, monitoring system so anyone can check whether given certificate could have been compromised and issued.
- Curtail number of possible certificate-based threats that could harm endpoint users.

Based on that, Certificate Transparency aspire to create a transparent system and database of certificates that can be understood as another layer of necessary verification

[1]Man-in-the-middle attack

which is put aside of existing TLS procedures. Any involved party is able to publicly monitor and audit the TLS certificate system and hence the Certificate Transparency service remains tenable and further also verified.

Better overview and control over certificates provided by CT is followed by earlier detection of misused or maliciously acquired certificates and also by faster mitigation.

3.2 Infrastructure

Certificate Transparency does not intend to completely change or replace already existing protocols and applied procedures. In fact, and as was already written in the section above, CT should be understood as another additional verification layer that only enhances existing certificate verification procedures.

Following that, CT introduces three main infrastructural components certificate logs, auditor, and monitors, with each having a different function and responsibility [28, 25].

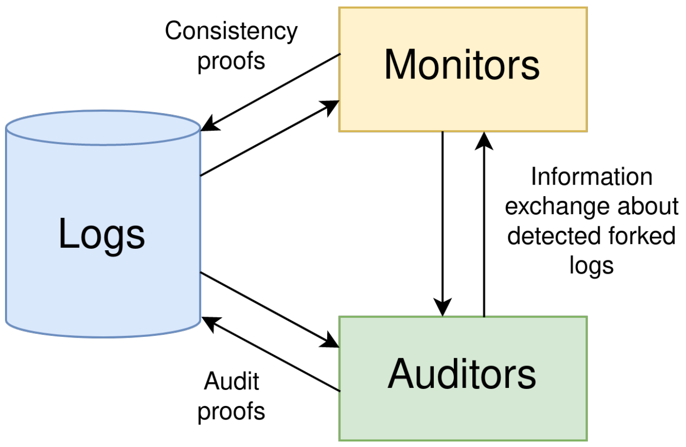


Figure 3.1: Certificate Transparency components

The diagram above shows that there is communication between each pair. Every operation visualized by arrows is performed asynchronously and periodically. Consistency proofs and Audit proofs are described in the following Subsection 3.3.3 and Subsection 3.3.4.

3.2.1 Logs

The most critical components of the CT system are certificate logs which operate as storage for TLS certificates. Storing certificates is not the only function they perform and to fulfill goals of Certificate Transparency service, logs also need to guarantee the following qualities [28].

- Logs are **append-only** and fully resistant to any tampering, remove attempts or any other behavior that could retroactively manipulate existing data.

- Logs are **cryptographically assured**. Merkle Tree Hash mechanisms ensure that logs are cryptographically secure and forbid any kind of possible corruption, removal, or tampering actions.
- Logs are **publicly auditable**. Anyone can check its consistency and behavior.

Logs are not limited in number and are also entirely independent of others. That implies, there is no communication between them, and one specific certificate can appear in multiple logs. Even though there can be numerous certificate logs, modern internet browsers keep a list of logs which given browser supports. Usually, support of given certificate log is conditioned by various acceptance criteria [29]. Also, Google Chrome requires all certificates to be *CT-qualified* which means that a specific certificate is logged in multiple logs and not in the single one [30]. Usually, it is the certificate authority and domain owner who submit TLS certificates into logs. By submitting a valid certificate into a log, it immediately responds with a so-called **Signed Certificate Timestamp (SCT)**. SCT is log's promise that given certificate will be inserted into a Merkle Tree (Subsection 3.3.1) within a time window known as **Maximum Merge Delay (MMD)**. SCT is then bound to the inserted certificate and will always be a part of the server's response when a client initiates TLS connection. Therefore, the client can subsequently use obtained SCT to verify the certificate the TLS server responded with. There are three available methods (X.509 extension, TLS extension and OCSP stapling) for delivering SCT bound to the certificate. Each incorporated SCT consists of the following fields [31, 25].

- Log ID - Identifier of a certificate log that includes given certificate
- Timestamp
- Ct Extensions - Extensions will be part of Certificate Transparency in future versions.
- Signature - Hash of certificate and other fields combined

Concept of CT might bring to mind Blockchain Technology[32] as there are certain similarities in used mechanisms such as Merkle trees (Subsection 3.3.1) which are used in Blockchain to calculate the hash of one single transaction block. Though, in the current CT version, there are no other Blockchain concepts (mining) used in Certificate Transparency. However, there are articles for *Certificate Transparency using Blockchain (CTB)* [33] proposing a new system that adds a new verification layer that makes it impossible for a certificate authority to issue a certificate without obtaining approval from a domain owner.

X.509v3 extension

SCT is delivered as a part of the X.509 certificate. In this method, CA is the submitter of certificate into the log and then, single performed exchange operation between log, CA, and TLS server carry out that all of them have the certificate with SCT. Followingly, the TLS server will use this enhanced certificate as a response to the client's request [31].

TLS extension

During this method, the TLS server obtains a certificate from CA and is also the one who is responsible for submitting the certificate into the Certificate Transparency log. Log then responds with SCT which TLS server further use in TLS handshake as an extension as soon as a client initiates TLS connection [31].

OCSP stapling

Online Certificate Status Protocol (OCSP) is a protocol for verifying the revocation status of X.509 certificate. It was developed as a better solution over *Certificate Revocation List (CRL)* and it allows the client to contact certificate authority directly to check the revocation status of a given certificate. *OCSP stapling* goes a bit further and shifts the responsibility of this process to TLS server which is required to be additionally configured for supporting this method. The server can then send revocation information to the client as a part of the TLS handshake [31].

1. CA Submits certificate into a log and immediately obtains the SCT
2. TLS server receives this exact certificate from certificate authority
3. TLS server executes OCSP query for CA and receive OCSP response with SCT attached to it.
4. Client initiates connection and server will use SCT as a part of TLS OCSP extension during the handshake.

Google, with their Chrome browser, currently supports 12 different log operators whilst each is maintaining multiple regular logs [34]. Each log is then represented by public API so anyone can request via HTTPS GET and POST methods [25]. One of CT's main goals is to have minimal impact on existing TLS infrastructure, certificate logs are standalone independent units and thus can't be provided by any of the already existing stakeholders in existing infrastructure.

In 2016 [35, 34] however, Google has introduced a different type of log called Special Purpose Log. Logs with names ***Daedalus*** and ***Submariner*** are both dedicated to certificates that or not trustworthy in general. Specifically, Daedalus log is intended to be a storage for certificates that have already expired i.e. will not accept certificates with future *notAfter* field. Submariner log is dedicated to certificates coming from untrusted CAs. Those, that were trusted previously in the past or those that are not yet trusted but are new and to be included into Chrome CA trusted base.

3.2.2 Monitors

Although logs are cryptographically assured, CT still needs another system component that will watch for certificates in general. For instance, whenever there is a new logged certificate, the monitor will check that such certificate is actually visible in certificate log. Monitors can also watch for specific certificates, whether they behave correctly or don't have unwanted permissions. Monitors can also serve as data backups when logs go offline because they can keep whole copies of logs [28].

Practically speaking, that leads us to the question: Who might be the ideal candidate to take responsibility for the objective of certificate monitor? The answer is certificate authority. In the DigiNotar case, compromised authority issued a TLS certificate for **.google.com* hostname, and because Google is also the main issuer of certificates for this hostname, it was also in Google's best interest to find that someone is trying to compromise services they provide. This case is a bit specific since Google is the original issuer and also the domain owner. Nevertheless, in digital certificates system, certificate authority is the actual owner of the issued certificate and therefore should be concerned about protecting their provided services. In fact, it is not a rule that CA will take responsibility for certificate monitor purposes, however, it is a typical configuration of the CT system [28].

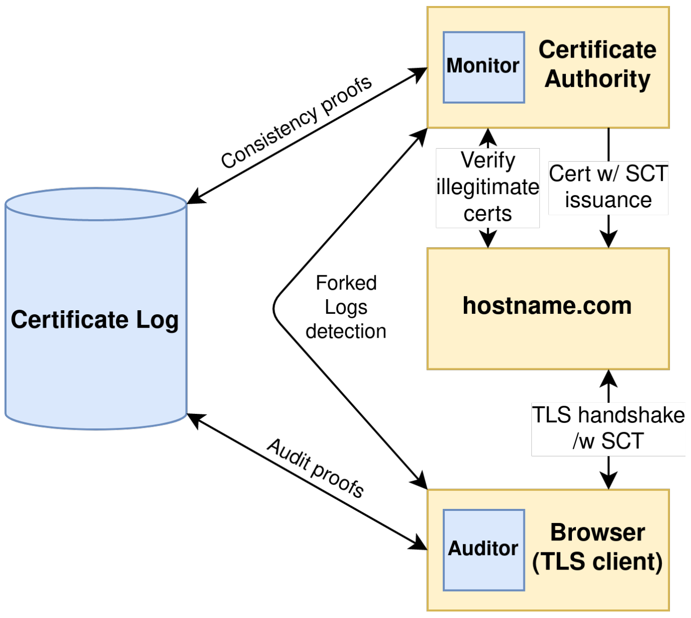


Figure 3.2: Typical Certificate Transparency infrastructure configuration

3.2.3 Auditors

As monitors focus more on integrity verification of submitted certificates, certificate auditors however verify overall integrity and standing of logs. Log's role is to also provide so-called *log proofs* and auditors aim to check whether these log proofs behave correctly. Essentially, log proof is a cryptographic hash that represents current state of the log and it is a literal proof that given log has not been corrupted in any way (more in the following Section 3.3).

3.3 Proofs

So far, only certificate logs have been introduced with their architectural and functional conception. This section however describes how log proofs and Merkle tree data structure work in practice so it is, in particular, an expansion to Subsection 3.2.1.

We will also put everything together and describe how two remaining CT components (Auditors and Monitors) practically interact with certificate log proofs and logs in general.

3.3.1 Merkle Trees

Requirements for certificate logs have already been defined in previous sections. Regarding to that, logs have to be **append-only** and **cryptographically secured**, that is in CT service implemented by so-called *Merkle tree*, also known as a *Hash tree*.

Merkle tree data structure is a binary tree where each leaf is enhanced with a cryptographic hash of a given leaf. Every non-leaf node is enhanced with cryptographic hash computed out of its children's node hashes which have been concatenated. This chain-mechanism results in one single cryptographic hash appended to the root of given Merkle tree, which is called **Merkle Tree Hash (MTH)**. Cryptographic hash function such as **SHA-256** is used for the hashing.

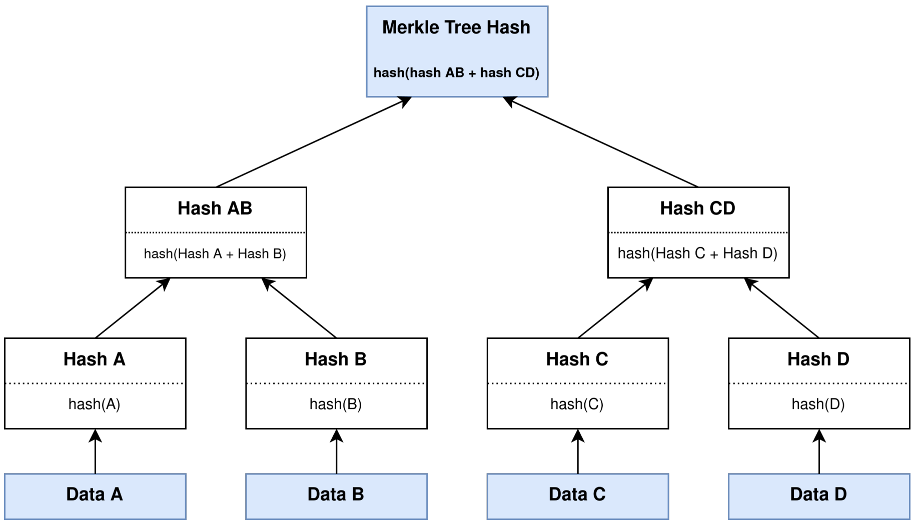


Figure 3.3: Merkle Tree example

The Merkle Tree Hash for nodes is then defined recursively as

$$MTH_{node} = SHA256(MTH(L_{CHILD}) + MTH(R_{CHILD}))$$

The calculation differs for nodes and leaves. For leaves, it is defined as

$$MTH_{leaf} = SHA256(leaf)$$

Let's continue with a more practical perspective about Merkle tree data structure and describe how are new data elements inserted.

Inserting (Appending) into a Merkle tree

Merkle tree is a simple binary tree with no requirement for any balance quality, however, inserting data into a Merkle tree differs a bit. Element insertion is explained in the following diagram figures.

Let's assume the same Merkle tree as depicted on Figure 3.3 and insert another sample element **Data E**.

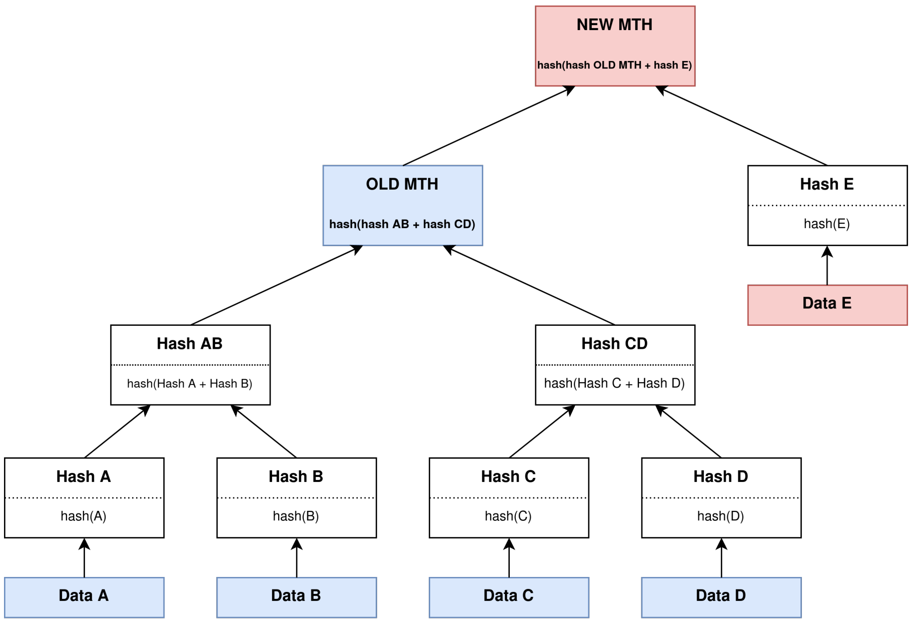


Figure 3.4: Insert operation of Data E

Merkle tree property is that each subtree hash is computed out of two elements (children). While inserting a new element into a tree, this element needs to "find" an already existing element to "pair with" and to compute a new hash which always also results in an absolutely new Merkle Tree Hash as depicted on Figure 3.4 [36].

Inserting another sample element Data F pairs with the only possible element, in this case, Hash E and creates a new subtree with **Hash EF** as a subroot and yet again inserting new element results into a new Merkle Tree Hash which is now computed out of **Hash ABCD** and **Hash EF** children. Important consequence of this insertion method is that each inserted data element will always remain a leaf of a given Merkle Tree and non-leaf (internal) nodes will always only represent calculated hashes.

3.3.2 Log Proofs

Relevant question to ask is what is the general advantage of Merkle trees regarding Certificate Transparency service. Accordingly, this section follows with a more practical point of view, answers the questions above, and describes how auditors and monitors interact with certificate logs.

We already know that the Merkle tree mechanism results in one single hash called Merkle Tree Hash representing the whole tree state. Accordingly, CT further proposes

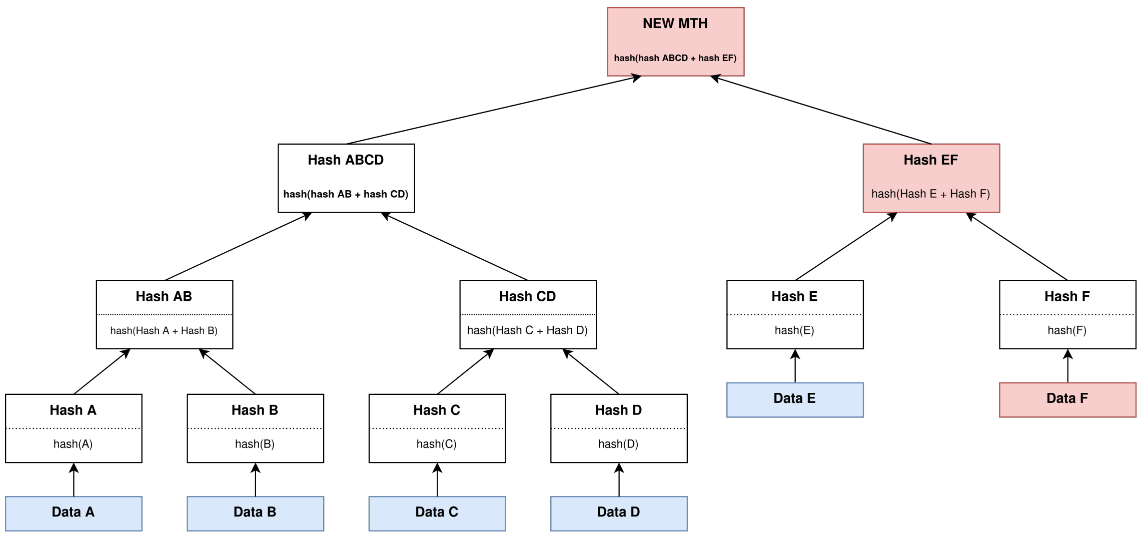


Figure 3.5: Insert operation of Data F

tree representation called **Signed Tree Head (STH)** which consists of the following fields [36, 25].

- **Version** - version of the protocol
- **Signature type** - Tree hash
- **Timestamp**
- **Tree size** - Number of entries/leaves in the Merkle tree (Not the number of nodes)
- **Merkle Tree Hash**

3.3.3 Consistency Proofs

Consistency is a general quality requirement for certificate logs. For this reason, interested parties in CT service need to be able to perform consistency checks and verify that any log hasn't been malformed or tampered in any way. The fact, that Certificate log is in consistent shape means, that there hasn't happened any kind of retroactive certificate modification or certificate removal [36, 25].

Regarding to insert operation in Merkle Trees, with every newly inserted data element, a new Merkle Tree Hash is computed and with new version of the tree, an earlier version of this tree has to be part of the new one (see Figure 3.5). Thus, consistency proofs are performed essentially whenever data insertion occurs and the new version of the log is published. Both monitors and auditors perform this kind of proof however monitors keep all log data and are able to calculate MTH by themselves and check STH using Certificate Transparency API.

We assume Merkle tree from Figure 3.4 where we have just inserted new element Data F and ended up with the tree on Figure 3.5. To prove that the earlier version of the tree is part of the new one, **Hash ABCD**, **Hash E** can be both taken and old MTH can be computed. To perform a consistency check, it is required to compute all intermediate nodes on the path to the root from our appended leaf. **Hash EF** is calculated out of **Hash E**, **Hash F**. Finally, new **Merkle Tree Hash** is calculated out of **Hash ABCD** and **Hash EF**

3.3.4 Audit Proofs

Auditor component, mainly represented by TLS clients, receives TLS certificate with Signed Certificate Timestamp and wants to verify whether given certificate is present in CT log and Audit proofs let TLS clients verify this presence [36].

Let's assume Merkle Tree depicted on Figure 3.6.

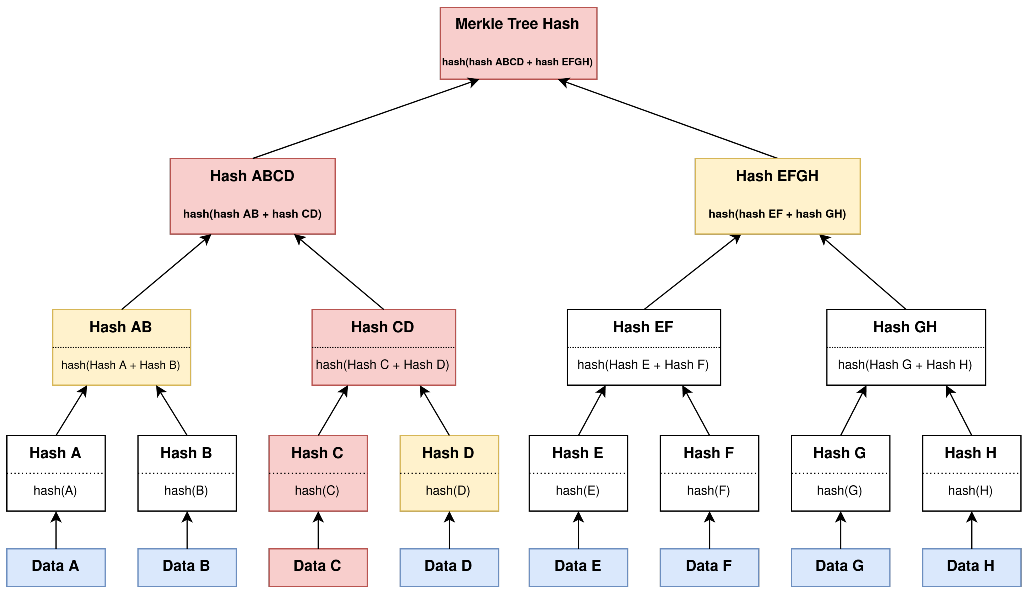


Figure 3.6: Audit Proof of Data C element

Audit proof for Data C element consists of hashes **Hash D, Hash AB, Hash EFGH** and computation of all intermediate hashes from Data C to the root.

- Hash C + Hash D result into Hash CD
- Hash AB + Hash CD result into Hash ABCD
- Hash ABCD + Hash EFGH result into MTH

By requesting an audit proof from the log, auditor receives and a list of nodes representing the audit path out of which it can compute Merkle Tree Hash and then verify it against the log itself. If the auditor computes MTH that does not match the MTH published by the log, that means, the certificate is not present in given certificate log.

By the end of this section, it is on point to tell the actual advantage of Merkle trees and the reason why Google has decided to implement this data structure. Yet again we are going to mention that CT has to be cryptographically secure and tamper-resistant as two general non-functional requirements. Hash functions are, apart from other properties, designed to be irreversible and therefore are a traditional mechanism in computer science how to create an integrity proof of some piece of data. Instead of Merkle trees however, the Hash chain, as more simple mechanism, could serve as well and satisfy previously defined requirements. Hash chain is a linear consecutive structure that repeatedly applies desired Hash function on an ordered list of inserted elements. Let's assume hash chain built out of element list **[A, B, C, D, E]**. Head of this hash chain is then computed as

$$head = hash(hash(E) + hash(hash(D) + hash(hash(C) + hash(hash(B) + hash(A)))))$$

by using a hash function strong enough like SHA-256, this mechanism definitely satisfies integrity goals. Consistency proof can still be performed as the older version presence in the hash chain can be verified after adding new elements. The main problem of hash chains arises when we try to perform audit proof and verify the presence of a single element in this linear structure. Assume we would like to find out whether element B is in the hash chain. Obviously, a hash computed out of preceding elements can be taken, which is, in this case, $hash(A)$, but to perform entire audit proof, it is required to recompute all successive hash values. Thus, the average complexity of this operation is $n/2$, where n represents the number of inserted elements.

That is the reason why it is on point to use Merkle trees, because tree data structure has a complexity of insert operation equal to a logarithm. Merkle trees are implemented by binary tree, in this case it is logarithm with the base of 2 and thus, binary tree with depth d, in level k, where $0 <= k <= d$, has 2^k elements and therefore it has 2^d leafs. Assume n representing a number of elements (in our case also a number of leaves), depth of a tree equals to $log_2 n$. And finally, the average complexity of proving the presence of some element in the Merkle tree is also logarithmic and not linear as it is in the case of a hash chain.

On Figure 3.6, we have 8 inserted elements (8 leaves) with depth 4 (level index 3) and thus it requires to compute $log_2 8 + 1$ hashes to get Merkle Tree Hash [25]. Because Merkle tree is not always full, the general formula for number of hashes then equals to

$$ceil(log_2 n) + 1$$

To give even more clear evidence that Merkle trees are ideal for our purpose, assume a log with 10 million certificates, a number of required hashes to compute then equals 24 and for certificate log with 100 million certificates, it is 27. As of November 2020, Google's log named Argon had a tree size of 960658713[2] certificates and hence the number of hashes required to verify a presence of an element equals 30 [25].

3.4 Usage in HTTPS and Browsers

Following text aims to describe inner certificate verification procedure in Auditor and thus in TLS clients which are in our case internet browsers. Nowadays, modern internet browsers support certificate transparency service by default and since 2018, Google requires in their Chrome browser all TLS certificates to be logged [37]. Regarding to that,

[2]Obtained via API request on Argon log, see Section 3.4 for API description.

except the normal certificate verification process, browsers are required to validate the SCT and thus, browsers are also required to implement three mechanisms from Subsection 3.2.1 for the SCT delivery. Certificate verification process then has the following order of performed operations [38].

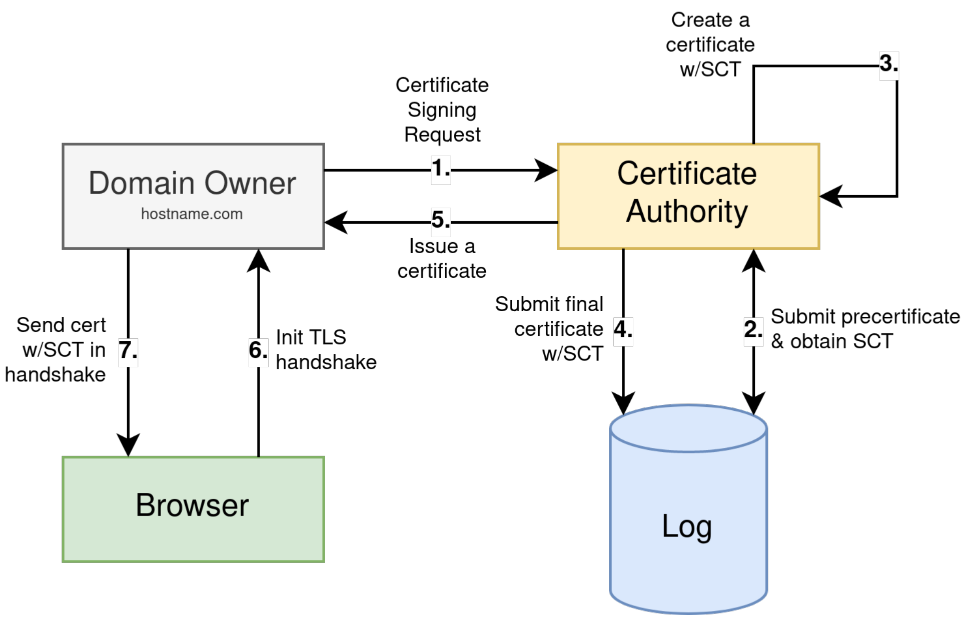
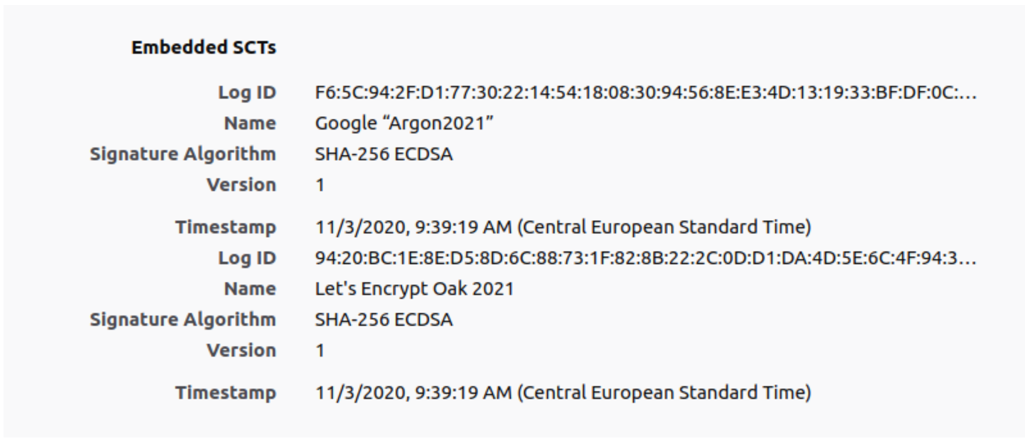

Create a certificate w/SCT
3.
Certificate Signing Request
Domain Owner
hostname.com
1.
Certificate Authority
5.
Issue a certificate
Submit final certificate w/SCT
4.
2.
Submit precertificate & obtain SCT
Send cert w/SCT in handshake
7.
6.
Init TLS handshake
Browser
Log

Figure 3.7: Certificate Transparency SCT delivery process to the browser using X.509 extension.

1. Domain owner creates Certificate Signing Request
2. CA creates a precertificate[3] and submission into a log. Certificate log responds with the SCT of the submitted precertificate.
3. Certificate Authority creates a certificate with obtained SCT.
4. Certificate Authority submits a final certificate to Certificate Log.
5. Certificate Authority issues created certificate with SCT to the domain owner
6. Browser initiates TLS handshake
7. Browser obtains the certificate with SCT. Verifies the certificate (Section 2.1.3) and then the inclusion of the certificate in a certificate log.

   (a) Check if Log ID is in the browser's allowed list of logs
   (b) Check whether SCT is not in the future
   (c) Audit proof - Ask CT log for audit path and inclusion of given certificate

[3]Precertificate is a special and invalid certificate only for the purpose of obtaining the SCT from CT log so the CA can subsequently create the final certificate with the correct signature. Thus, precertificates are used only when the SCT is exchanged as X.509 extension.



Embedded SCTs	
Log ID	F6:5C:94:2F:D1:77:30:22:14:54:18:08:30:94:56:8E:E3:4D:13:19:33:BF:DF:0C:...
Name	Google "Argon2021"
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	11/3/2020, 9:39:19 AM (Central European Standard Time)
Log ID	94:20:BC:1E:8E:D5:8D:6C:88:73:1F:82:8B:22:2C:0D:D1:DA:4D:5E:6C:4F:94:3...
Name	Let's Encrypt Oak 2021
Signature Algorithm	SHA-256 ECDSA
Version	1
Timestamp	11/3/2020, 9:39:19 AM (Central European Standard Time)

Figure 3.8: SCTs embedded in certificate on visiting google.com

Log's public API

Certificate log is the only CT component that should be publicly available and also should provide some information. All requests are performed via GET or POST HTTPS requests on ***https://<log server>/ct/v1/<function>***. The following list of functions describes the public API of any certificate log regarding to all previously described Certificate Transparency mechanisms [25].

- ***get-sth*** **- Get the most recent Signed Tree Hash**
- ***get-entries*** **- Get entries from the log**
- ***get-sth-consistency*** **- Perform Merkle Consistency Proof**
- ***get-proof-by-hash*** **- Perform Merkle Audit proof**

3.4.1 Third-party services

There are various third-party services that provide any kind of useful information about Certificate Transparency.

- **BadSSL**[4] - Service providing multiple ways how to test internet browsers for TLS certificate based errors including *no-sct* website which does not respond with TLS certificate enhanced with the SCT.
- **Merkle Town**[5] - Public website with statistical information about Certificate Logs.
- **crt.sh**[6] - Web interface for Certificate Transparency logs.
- **Google Transparency Report**[7] - Google's turn on search service and web interface for Certificate Transparency logs.

[4]https://badssl.com/
[5]https://ct.cloudflare.com/
[6]https://crt.sh/
[7]https://transparencyreport.google.com/https/certificates

Chapter 4

Task definition

Previous chapters dealt with two essential theoretical parts of this thesis: TLS protocol and Certificate Transparency. This chapter is however a starting point for practical aspects and goals this thesis wants to accomplish. There are few last topics, that are necessary to understand and need to be theoretically outlined: *Machine learning* and *Pattern recognition*, *classification* and *detection* based on input data called *features*. This chapter describes previously mentioned topics and serves as a definition of the solved problem.

Our general goal is to explore a new data source and experiment whether it could be actually useful for *malware*[1] recognition and classification. It usually refers to threats such as viruses, worms, ransomware, and many others. This type of harmful software is often spread across the internet network and therefore we are looking for ways how to detect such behavior by capturing network traffic and extracting useful information. For detection purposes are used various algorithms (models) that belong to the machine learning study field, which became extremely popular in the last decade.

4.1 Classification

From the top level of machine learning models, there are two main categories: *supervised* and *unsupervised* learning. Supervised learning is a family of algorithms that aims to create a mapping between input data x and output Y[2]. The goal is to create and approximate a model so that it is able to make accurate predictions for a new input data x' without knowing its Y'. Unsupervised learning, on the other hand, doesn't have available output Y for the model creation, thus the goal of these algorithms is understanding inner relations of the input x and looking for general behavior and patterns. Among unsupervised learning tasks belongs for example *clustering*, that aims to find whether the input data create more or less separated clusters and could be possibly categorized in any way [39].

In this chapter's introduction, we outlined the major goal of this thesis: **detection of malware**. We are working with already labeled network data logs, that are marked as *positive* (malware is present) or *negative/unlabeled* (presence is unknown), whilst the

[1] Shortened form of malicious software

[2] Y is the "teacher" or the "supervisor", hence "supervised" learning

positive class is further separated into specific malware types. The problem belongs to the family of supervised learning algorithms, our task is specifically called multiclass or multinomial classification. Classification is understood as a process of categorization based on input data, assigned category is often called *class* or *label*. Example of such classification task is recognition of handwritten digits by a computer and labeling with one of 10 corresponding categories (0-9 digits). Different simplified tasks could be the recognition of fruit based on input data such as height, width, weight, or color. Data used as an input to the classifier are called *features*. Besides classification, within supervised learning models, we also have so-called *regression* methods [39]. Regression models do not assign a label to the input data but try to predict a real number. Example of such problem could be a prediction of temperature based on other weather conditions. Being able to create useful model that is very flexible and universal in use, we *train model* with our *training data* sample and then perform evaluation and *model testing* with *test data* sample. An assumption is made that each object from the *dataset* is independent of others, even though that might not actually be true in practice [39].

We have a set of input labeled data $(x_1, y_1), (x_2, y_2), (x_n, y_n)$, where x_i are feature vectors and y_i are corresponding labels[3]. Classifier training is often based on so-called *objective function*, training phase then aims to optimize classifier in such way that the objective function is minimal or maximal (it depends on trained model). Already trained classifier is then a mapping function $f : X \rightarrow Y$, which for a new input x predicts class $\hat{y}$ [39, 40].

4.2 Model evaluation

Splitting our input dataset into two distinct parts (training and testing data) is primarily for the purpose of model evaluation. Essentially, it is not good practice to evaluate our model on training data (even though it might also be used as a relevant metric in some cases), because we automatically expect that our trained model will have a better performance on the data this model was trained with. Therefore, it is on point to simulate a case when the trained classifier takes completely new input data that it has never seen before and measure how such experiment performs.

Evaluation also deals with model complexity, regarding to that, we have to cover *overfitting*, *underfitting* terms and *bias-variance tradeoff*. Overfitting situation occurs when a model tries to learn every small data noise and thus the model's complexity grows. Underfitting, on the other hand, occurs when our model wasn't able to learn underlying patterns and its complexity is too low. These two contradicting reasons can cause high model error, this behavior is called bias-variance tradeoff. Model with high variance does not generalize and is too complex, thus overfits. Model with high bias does not benefit from training data as it possibly could, thus underfits [39].

[3]In case of previous fruit recognition, vector x_i with dimension 4 would consists of height, width, weight, color category and corresponding labels y_i could be for instance apple, orange, banana, pear

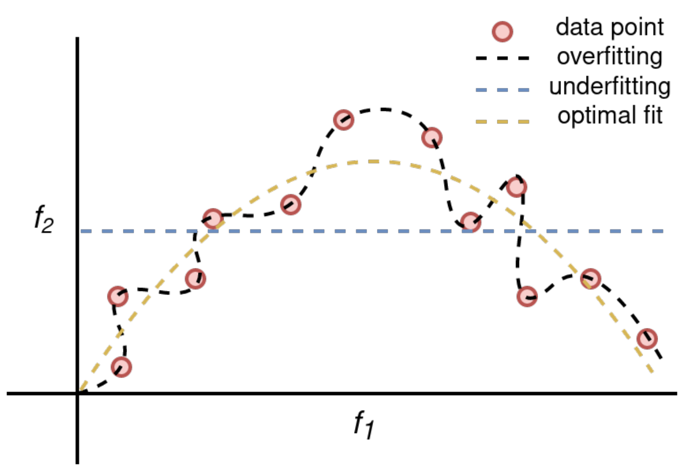


Figure 4.1: Overfitted, underfitted and optimal model example

The following text will cover specific model evaluation metrics. First measure that comes mind is model *error* or complementary model *accuracy*. Error is calculated as a fraction of number of samples that were incorrectly classified to all samples. Accuracy then as a fraction of number of correctly classified samples to all samples [39, 41].

$$Error = \frac{Number\ of\ wrong\ predictions}{Number\ of\ all\ predictions}$$

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Number\ of\ all\ predictions} = 1 - Error$$

Both error and accuracy however suffer from *class imbalance*. Let's assume a case with 100 testing samples out of which 90 are negative and 10 are positive. If the classifier trains itself in a way that it classifies everything as a negative, 10% error rate and 90% accuracy would be present, which might imply, that our model is performing quite well in fact it learned literally nothing from our data [41]. It is necessary to adopt more robust evaluation metric, section below aims to describe so-called *confusion matrix* and subsequent measures as a better option to error and accuracy.

4.2.1 Confusion Matrix

In binary classification with 2 distinct classes between which classifier decides, confusion matrix can be created consisting of the following categories [39, 42].

True Positive (TP): Samples classified as positive that are actually positive - correct classification.

False Positive (FP): Samples classified as positive that are actually negative - "false alarm" and also known as a Type I error.

True Negative (TN): Samples classified as negative that are actually negative - correct classification.

False Negative (FN): Samples classified as negative that are actually positive - Type II error

These categories are the building blocks of more interesting and relevant model evaluation measures covered in following Subsection 4.2.2 and Subsection 4.2.3. For purpose of this thesis and multinomial classification, confusion matrix becomes more complicated. In our case, *one-vs-all* confusion matrix, which assumes one class at a time, is used.

		Actual class	
		Malware	Not Malware
Predicted class	Malware	20	5
	Not Malware	3	12

Table 4.1: Confusion matrix example: 20 TPs, 12 TNs, 5 FPs, 3 FNs.

4.2.2 Precision

Precision, also known as a *positive predictive value*, is a measure of how big proportion of truly positive predictions are actually positive. In other words, it is a probability that positively labeled samples are truly positive [43].

$$Precission = \frac{TP}{TP + FP}$$

4.2.3 Recall

Recall, also known as a *sensitivity*, is a measure of how big proportion of all positive samples was labeled correctly [43].

$$Recall = \frac{TP}{TP + FN}$$

4.3 Features

Theoretical aspects of machine learning, classification, and model evaluation have been covered. The last remaining topic to touch is a description of specific classifier we have chosen for detection experiments, that will be however a part of Chapter 7. Classifier training requires input data called features. In the following text, we will touch our problem from the perspective of input data and most importantly from the perspective of Certificate Transparency as our data source for feature extraction. Our goal is to use TLS certificates as a data source for feature extraction and possibly also for malware detection on top of currently existing ones.

Certificate Transparency aims to be as transparent as possible and essentially thanks to that the whole system can work from a practical perspective. Certificate monitor component downloads whole certificate log and then perform monitoring over downloaded data, performs consistency proofs, and monitors whether it contains all logged certificates. For a purpose such as this one, certificate log implements an *API*[4] that allows anyone to interact with it. **Append-only** trait of a certificate log ensures that the whole log behaves like an enormous **historical storage** out of which literally anyone can download submitted TLS certificates with use of published API.

4.3.1 Current Approach

Traffic encryption indeed offers another level of privacy for common network users. On the other hand, it also gives another level of freedom to threat actors as their behavior can remain undetected. *NetFlow protocol* [44, 45] is a commonly used technology offering high-level network logs. Current features are based on the data extracted from the NetFlow protocol as well as on additional *Encrypted Traffic Analytics (ETA)* [46]. Each feature vector is identified by a subject that is called *hostname*, which may be in a form of an *IP address* (e.g. 8.8.8.8) or a *Domain name* (e.g. www.google.com). Hostname is extracted out of an initial packet from client hello message in TLS handshake (Subsection 2.1.3), where Server Name Identification (SNI) is sent.

As our starting point, there are already existing features extracted from TLS certificates, but note that these features are based on information from **one single certificate** for each corresponding hostname.

Chain Length: Integer - Chain of Trust length
Chain Validation Code: Enum - Validation code during chain of trust verification
Hostname Matches Certificate: Boolean - Requested hostname is (not) covered by the downloaded certificate
Not Before: Boolean - Certificate (in)valid
Not After: Boolean - Certificate (in)valid
Issuer Popularity: Integer - Statistics built on issuers from downloaded certificates

4.3.2 Designed Features

Creating features from one single certificate for each hostname is indeed a reasonable approach. The main motivation and questions for this thesis are: *What if we would have available a full history of certificates for each hostname? Would such approach give us more relevant information and would it be helpful regarding to malware detection?*. In the following sections and chapters, we will finally describe steps taken to answer the questions above.

Our initial hypothesis is that such data are helpful, offer a lot more contextual information in general, and extraction possibilities for each hostname than one single certificate approach. Further, then it has be to experimentally verified whether this hypothesis is correct or not. One of the first steps was the creation of a list of feature ideas that could be extracted out of the list of certificates. We therefore assume, this historical information for each hostname is available.

[4]Application Programming Interface

The general feature designing approach could be defined as: *"We want to come up with multiple information that represents how strong is given hostname regarding to certificates."*. By strength is meant for example its current validity or how many certificates are present for each hostname historically. For a better explanation, let's assume a case with two hostnames, first one with 10 total certificates historically, the second hostname with only 1 certificate, our approach then assumes, the first hostname is *"stronger"* regarding to certificates and more trustworthy because the underlying intuition assumes there is a lower chance such hostname would be a source of malware. This was our top-level underlying idea and our very first feature draft included the following.

Number of hostnames: Numerical - Each certificate is valid for a set of hostnames. This number represents sum of all set sizes across whole certificate history for given hostname.
Number of certificates: Numerical - Total number of certificates historically.
Hostname coverage by Wildcard certificate: Boolean/Numerical - Hostname is (not) covered by Wildcard certificate.
Certificate Authority code: Categorical - Categorical variable identifying Certificate Authority.
Issuer Suspicious: Boolean - Indicator whether certificate issuer is on the list of so-called suspicious issuers. By suspicious are meant issuers that are more likely to issue a certificate for any certificate-based threats. For example, *Let's Encrypt* CA issues certificates for free and in past years became favored over other authorities.
Certificate Authority popularity: Numerical - Number representing global statistics among Issuers and their popularity. For example a global count of certificates for each issuer.
Validity: Boolean - Information whether there is at least one valid certificate that covers given hostname.
Period of Issuance: Numerical - How often is given hostname certified.
Change of Certificate Authority: Numerical - How many authorities have issued a certificate for this hostname. Number of overlapping different authorities.

After this feature proposal, we continued with the implementation of extraction pipeline. The list of features above did not really consider any aspects regarding to implementation. During the implementation phase, numerous problems have been encountered that are discussed later in Chapter 5 and based on that, the final list of extracted features differs and its description is also part of the next chapter.

Chapter 5

Pipeline

For processing data from certificate transparency, it was necessary to implement an extraction pipeline in a distributed environment to handle a large amount of data (billions of certificates - 11TB) and this chapter primarily covers steps taken during this implementation phase. The following text starts with a description of our data source and proceeds with the pipeline itself.

For practical reasons, we wanted to process all available certificates that are stored in Certificate Transparency service and extract designed features for each hostname. This approach had to be however changed due to encountered problems and it was necessary to continue with more of a *"proof of concept"* approach. This decision with its consequences is described later as a part of this chapter.

5.1 Collecting certificates

Certificate Logs publish their API (Section 3.4) so other interested parties and components of the whole process can work properly. As was already described in Section 4.3, one of the available log's functions is *get-entries* which allows anyone to download certificates from a given log (primarily for purpose of Certificate Monitor). As the API and the whole service aim to be transparent and public, with the use of any of modern programming languages that already have available libraries for HTTPS requests, implementing such data collection algorithm is not a problem today.

For purpose of this thesis, already existing and fully working implementation has been used, which has also already been deployed in a production environment. As of December 2020, the algorithm collects certificates from 45 public Certificate Logs that are compliant with Chrome's Browser CT policy [34, 47]. It also incorporates third party solution for processing and handling of X.509 certificates named *ZCertificate* [48, 49] which defines the resulting schema of the output data [50].

Get-entries function's input is starting and ending index of the log's entry, based on that, our collection algorithm is executed daily and requests only such certificates that have been included to the Certificate Log since the previous day. Collected certificates are then stored and organized into distinct directories day by day into multiple *Apache*

Parquet [51] data format files (*ct/date=2020-12-14/...parquet*). Also note, that one parquet file contains more certificates (It varies a lot, from thousands to millions) and not a single one. Already collected and daily updated certificates were the starting point for implementation of the extraction pipeline. By February 2020, there were available approximately 11 Terabytes of collected certificates.

5.2 Extraction pipeline

Our deployment environment consists of Scala language and *Apache Spark* distributed ecosystems capable of processing large volume data on multiple instances and thus in a parallel manner. Apache Parquet format is this ecosystem's native data format and Spark framework offers various built-in functions for general data manipulation. The following text starts with an explanation of basic Spark operations and continues with a description of the implemented extraction pipeline separated into multiple subsequent parts.

5.2.1 Spark fundamentals

Spark framework essentially provides abstraction on loaded data with so-called *DataFrame* and *Resilient Distributed Dataset (RDD)* collections. As a distributed environment, Spark creates partitions of data and spreads them across nodes in a way so all performed operations can be executed as much as possible in parallel on different instances. Both DataFrames and RDDs offer an API with basic data manipulation operations and essentially we distinguish two types covered in the following text [52].

Transformation

Transformation operation fundamentally only converts the data from one form to another, but only between Spark collections. Thus, a transformation performed on RDD will always result in new, yet different, RDD. Also, note that transformations are not executed instantly, Spark framework builds so-called *Directed Acyclic Graph (DAG)* with operation references, and all performed transformation won't be executed until there is any *action* operation [52].

Map - Each RDD row of data type A is transformed into a single row of data type B.

FlatMap - Each RDD of data type A is transformed to a zero or multiple rows of data type B.

Filter - Keep only such rows that meet defined condition.

ReduceByKey - Reduce operation performed on elements with the same key. The transformation from data type A to data type A.

Action

Action operation executes created DAG with all performed transformations. Practically speaking, it converts the data from Spark framework collections to language-specific data type (e.g. from RDD[Int] to Array[Int]) [52].

Reduce - Aggregation of RDD rows with predefined function.

GroupBy - Aggregate RDD rows with the same key to iterable collection.

Count - Count number of rows.

Collect - Convert whole RDD into an array.

5.2.2 Data Load - Phase 1

In this phase, all collected certificates are loaded. Then each certificate is represented as an object for which had to be defined separate schema. It is not possible to infer schema from each row because it differs due to different X.509 extensions. Nevertheless, that wasn't an issue because we are interested only in a limited number of certificate fields, and for optimization purposes, it is always a better practice to work with as little data as possible. Based on that, the corresponding Scala class with the following fields has been defined.

Hostnames - Set of hostnames for which given certificate is valid. This information is extracted from *names*, *commonNames* and *dnsNames* certificate fields.

Issuer - Parsed information about the Issuer (Certificate Authority) that issued given certificate. Due to inconsistencies and issuer generalization, the implemented algorithm parses this information and identify issuers only by *Organization (O=)* field.

Time Information - Numerous certificate fields regarding to validity: *notBefore*, *notAfter*, *length*. Out of these fields, algorithm extracts whether given certificate is currently valid and *epoch time* representation of notBefore and notAfter timestamps.

Organization - Subject that created Certificate Signing Request.

Precertificate - Boolean indicator whether given certificate is a precertificate.

Serial Number - Unique identifier of a Certificate

Right after the certificate field filter, two major filter operations are performed. First filter for all precertificates, because it is relevant to work only with certificates dedicated for source verification (Precertificates contain *poison* in X.509 extension which invalidates them). Second filter for all redundant certificates. From our perspective, even though it is a whole CT's feature, having one certificate logged in multiple Certificate Logs causes a lot of data redundancy and it is on point to get rid of it as it would negatively influence computed features. For that purpose, distinct filter operation has been implemented to keep only one copy of each certificate.

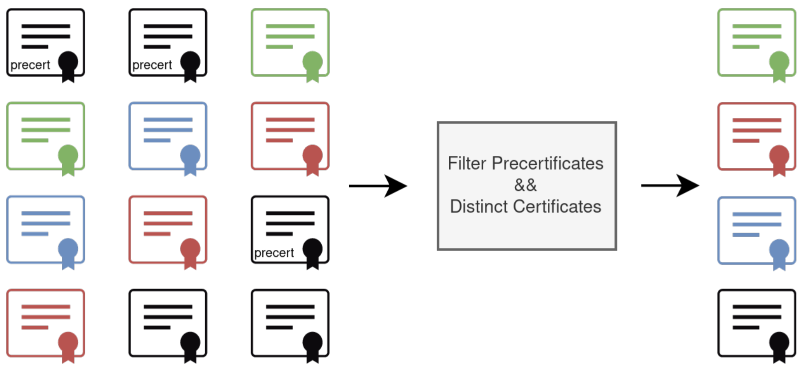


Figure 5.1: Filter and Distinct operations on loaded certificates

5.2.3 HostName Mapping - Phase 2

At this point, we have all relevant certificates, with no precertificates and redundant data. To make each hostname mapped on its certificate, the algorithm unwraps all its hostnames with flatMap operation and maps them to the original certificate, thus the result of this phase is a tuple *(hostname, certificate)*.

```
certificates.flatMap(certificate => {
    certificate.hostnames.map(hostName => {
        (hostName, certificate)
    })
})
```

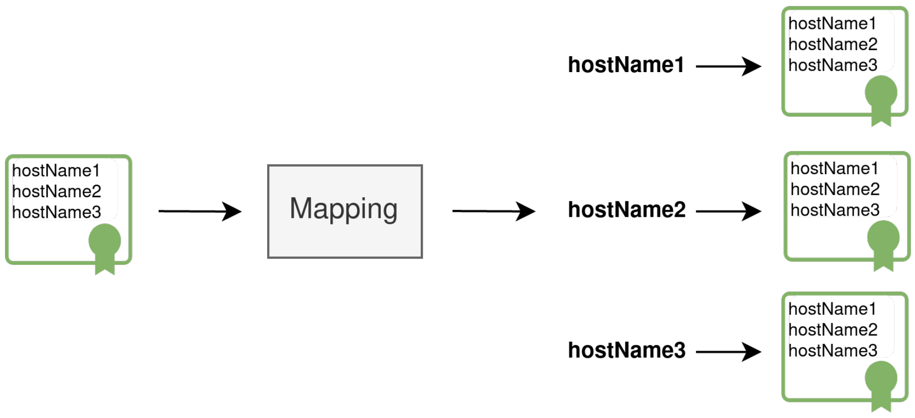


Figure 5.2: Mapping operation on one single certificate

5.2.4 Feature extraction - Phase 3

Mapping created in the previous phase is everything necessary for actual feature extraction performed on each hostname. There is a reduceByKey operation executed on the previous tuple, where the key here is the hostname. This operation aggregates all tuples with the same key and executes predefined functions. For calculation of a total number of certificates for every single present hostname, implementation would have the following form.

```
mappedData.map(mapping => {
    val hostName = mapping._1
    (hostName, 1)
}).reduceByKey((totalCertsX, totalCertsY) => {
    totalCertsX + totalCertsY
})
```

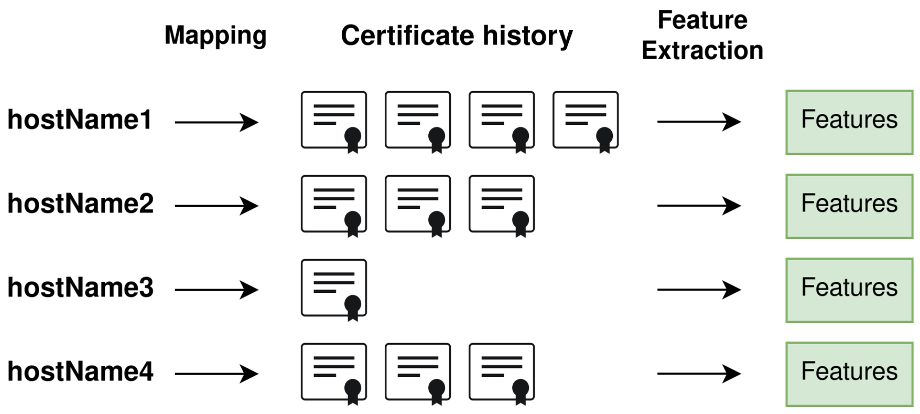


Figure 5.3: Feature extraction on history of certificates

With this approach various feature functions have been defined, an in-depth description of each is in the following Section 5.4.

5.2.5 Optimization Obstacles

The implemented pipeline has been tested with more than 60 unit tests and integration tests (verification of deployment environment). Initial thesis' goal to process whole Certificate Transparency and calculating features for each hostname showed out not to be that trivial. During the deployment were encountered obstacles regarding to amount of the data that could be processed at once with the designed pipeline.

Numerous experiments have been performed, with different time periods and different subsets of feature functions. Even though the possible amount of data to process enlarged with smaller feature subsets, processing of the whole collected Certificate Transparency wasn't successful. To fulfill the goal of this thesis, a general approach for the computation of features has changed, more in Section 5.3.

5.3 Proof of Concept approach

Proof of concept approach aims to reduce the amount of processed data at once and still successfully explore collected certificates with extracted features on the relevant size of the data sample. In the current feature cache, for each day are available different hostnames with corresponding calculated features which will be subsequently enhanced with new ones from certificates.

In this approach, specific time interval of one month is selected (August 2020, October 2020) and feature extraction is performed only on hostnames that appeared during this period. Implemented extraction pipeline uses a set of these hostnames to filter and keep only those certificates where given hostnames appears and after that, we process their whole certificate history. Based on that, the first initial step of the extraction pipeline had to be extended for another filter operation. The exported telemetry set was additionally enhanced for all wildcard hostnames. For example for *sub.domain.hostname.com* separate algorithm created 2 additional possible hostnames: **.domain.hostname.com*, **.hostname.com*.

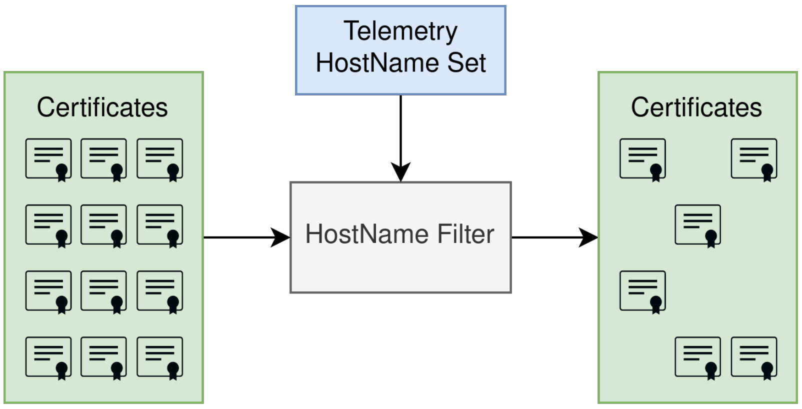


Figure 5.4: HostName filter on loaded certificates with hostname set extracted from the telemetry

This approach is fully working and allows to extract all necessary information to fulfill goals of this thesis. However, the downside is that it only allows to detect malware for hostnames that appeared in the past and from production perspective, to perform real time malware detection on new observed hostnames in telemetry, it would require to change the approach and re-implement the extraction algorithm. Results of this thesis will decide whether to invest additional resources and enhance the pipeline which would then be able to run in production environment and extract the features automatically. One of the viable approaches would be to scale the input certificates and compute them in batch-like manner. Anyway, this also depends on other factors that would be necessary to decide, such us how often would this pipeline be executed and whether to recompute the whole Certificate Transparency on every execution or just update already computed features. Updating approach can however affect final feature set because then it would not be possible to extract totalUnique features in general.

5.4 Extracted features

This chapter contains description of the final list of designed and extracted features from the Certificate Transparency logs.

Total HostNames: Int

Each certificate contains a list of hostnames for which it is valid. This feature is a sum of all these list lengths across history of certificates for one hostname. Take into account that the resulting number contains redundant hostname.

Total Unique HostNames: Int

Total size of HostName set consisting of all hostnames that appeared in the history of certificates. Resulting number contains only unique hostname.

Total Unique Wildcard HostNames: Int

Total size of HostName set consisting of all wildcard hostnames that appeared in history of certificates. Resulting number contains only unique wildcard hostnames.

Total Certificates: Int

Number of all unique certificates that appeared in history for given hostname.

Total Valid Certificates: Int

Number of all unique certificates that appeared in history for given hostname and are currently valid.

Total Issuers: Int

Number of all unique Issuers that appeared in history for given hostname.

First Certificate Date: Long

Timestamp (in epoch time) of the first certificate that appeared in history for given hostname.

Total Suspicious Valid Issuers: Int

Number of all currently valid issuers that have been marked as suspicious which refers to authorities that issue certificates for free (For example Let's Encrypt).

Total Suspicious Certificates: Int

Number of all certificates that have been issued by a suspicious issuer in the history of certificates for given hostname.

Maximum Overlapping Issuers: Int

Maximum number of overlapping issuers (multiple various issuers with a valid certificate at the same time) in history of certificates for given hostname.

Wildcard Coverage Count: Int

Number of all wildCard hostnames that cover given hostname in the history of certificates.

Absolute Coverage Count: Int

Number of all certificates in which appear given absolute hostname. By absolute is meant the exact same hostname for which is computed the feature vector.

Is Wildcard Root: Boolean

Indicator whether given hostname is also often used as a wildCard root. Assume wildCard hostname **.hostname.com*, *hostname.com* is then considered as a wildCard root.

Total Unique Organizations: Int

Total number of unique organizations that appeared in the history of certificates for given hostname.

Chapter 6

Analysis

At this point, features have been successfully extracted from Certificate Transparency for desired hostnames. It is however on point to verify and evaluate how designed features perform with *data mining* experiments. In this chapter, we will take a closer look at each feature, propose numerous plots and compare their distributions.

Features have been calculated for two distinct time periods, August 2020 and October 2020, verification experiments have been then performed on both. Results for August are covered in this chapter but October resulting plots can be found in Appendix A. For statistics that appear in this section and in the following Chapter 7, take into account that in this section we only assume unique hostnames that appeared in network traffic, thus the numbers of positive samples differ. Besides that, the entire following classification is multiclass, however in this section, we only assume two classes: **positive** and **negative**.

6.1 August 2020

This section covers feature analysis performed on features extracted from all unique hostnames that appeared during the August 2020 time period.

6.1.1 Global Statistics

Table 6.1 shows a total number of hostnames extracted from telemetry for a total of 1 135 766 have been successfully found and matched some certificate. A total of 189 556 have remained unmatched, thus no certificate has been found for them.

Exported	Matched	Unmatched	Positive	Negative
1325322	1135766	189556	2130	1132041

Table 6.1: Total hostnames count for October 2020

Table 6.2 shows top 10 issuers with the most of the certificate and top 10 subject organizations that created certificate signing request. First of all, Let's Encrypt [53] as well as Comodo [54] certificate authorities in the lead of issuer statistics are not surprising.

Both authorities are one of those we mark as suspicious as they both issue certificates for free. On the right side, in organization statistics, we can see that majority of certificates have this field blank (empty). In certificate signing request, the organization field is not mandatory thus it can remain unfilled and this was also expected observation.

Top 10 Issuers		
Issuer	**Certs**	**%**
let's encrypt	5289134	39.7
comodo ca limited	4538413	34.1
cpanel, inc.	851812	6.4
digicert inc	737914	5.5
globalsign nv-sa	396351	2.9
godaddy.com, inc.	248519	1.9
cloudflare	184085	1.4
sectigo limited	162237	1.2
amazon	143394	1.1
geotrust inc.	109704	0.8
Total 594	**13314036**	**100**

Top 10 Organizations		
Organization	**Certs**	**%**
blank	11495272	86.3
cloudflare	320903	2.4
netflix, inc.	294591	2.2
incapsula inc	99714	0.7
google inc	66704	0.5
google llc	35230	0.3
firebase, inc.	23626	0.2
fastly, inc.	19889	0.1
facebook, inc.	18289	0.1
microsoft	17820	0.1
162799	**13314036**	**100**

Table 6.2: Global issuer and organization statistics for August 2020

6.2 Feature analysis

Each feature is in this section analyzed by plotting the distribution (histogram) of each class (positive and negative) and then visually comparing observed patterns between classes. As we are performing only visual comparison and the evaluated dataset is highly imbalanced, the vertical axis is not useful, and therefore it was hidden on all following plots. Also, take into account that this whole analysis step is performed to understand the data and possibly observe completely new patterns. Although there are also cases where distributions of both classes don't show any significant difference. That does not however automatically mean given feature is useless and should not be used as there might be for example some unobserved correlation behavior with other features. Based on that, for detection experiments described in Chapter 7 haven't been excluded any designed features.

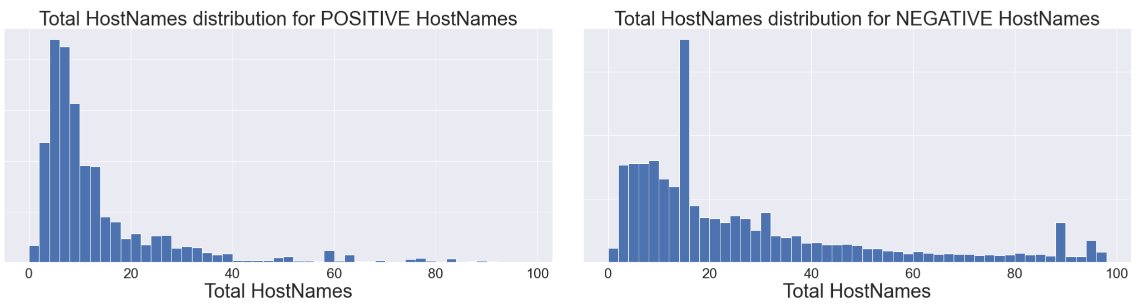


Figure 6.1: Total HostNames feature distributions

For this one and following plots, we have to take into account the class imbalance. Therefore, we are looking for significant patterns. In the case of total hostnames feature as depicted on Figure 6.1, we have histogram distributions with a bin size of 2 and positive samples seem to generally have historically fewer hostnames (with redundant ones). Nevertheless, that could be caused by different patterns, for instance by total number of certificates or by general hostname age.

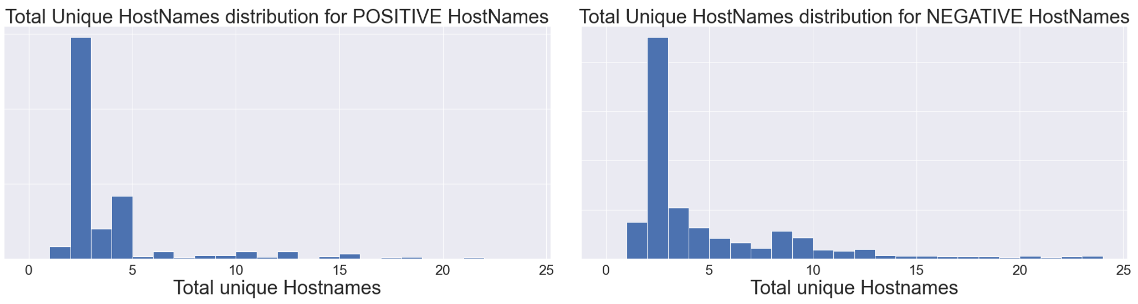


Figure 6.2: Total Unique HostNames feature distributions

In the case of total unique hostnames (Figure 6.2), distributions are with a bin size of 1 and do not show anything particularly interesting. The majority of hostnames are in the same bin with 2 unique hostnames on both sides.

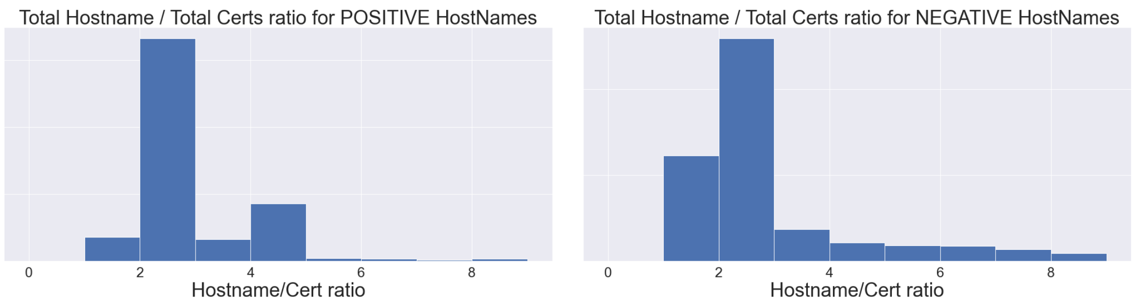


Figure 6.3: Total average HostNames per certificate distributions

Figure 6.3 shows an average number of hostnames in one certificate. Distributions are similar and we can see that the average certificate has 2-3 hostnames.

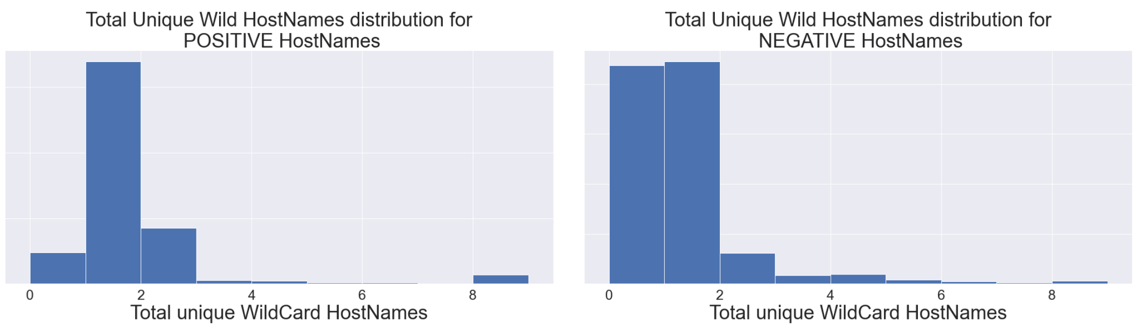


Figure 6.4: Total Unique WildCard HostNames distributions

Figure 6.4 shows that positive samples tend to have less number of unique wildcard hostnames than negative ones.

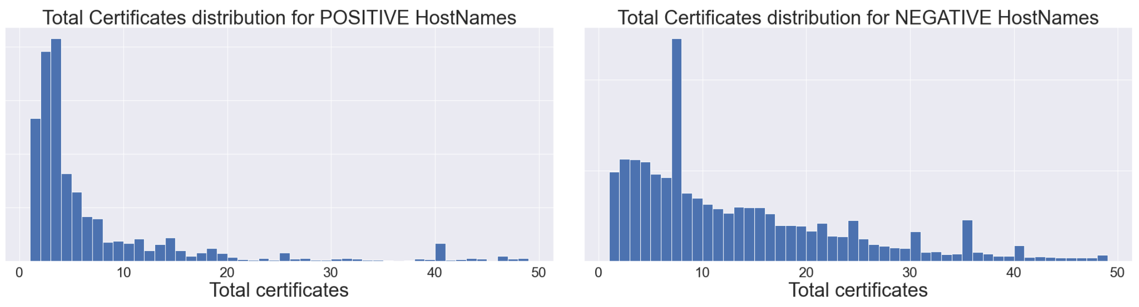


Figure 6.5: Total certificates distributions

Total certificates distribution plots, depicted on Figure 6.5, show that positive hostnames might generally have fewer certificates and also possibly be younger than negative ones.

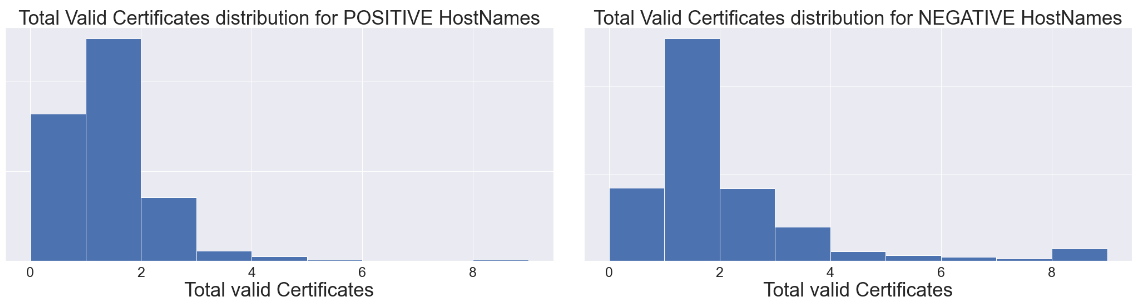


Figure 6.6: Total valid certificates distributions

Number of valid certificates (Figure 6.6) does not seem to be affected by class that much.

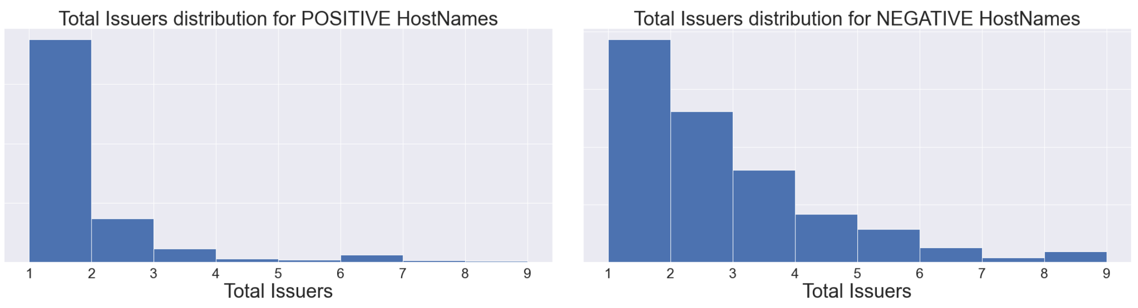


Figure 6.7: Total issuers distributions

As every certificate has an issuer, histogram (Figure 6.7) starts from 1 on the horizontal axis. And yet again, a number of issuers might be influenced by hostname age and thus might be assumed that positive samples generally have fewer issuers historically.

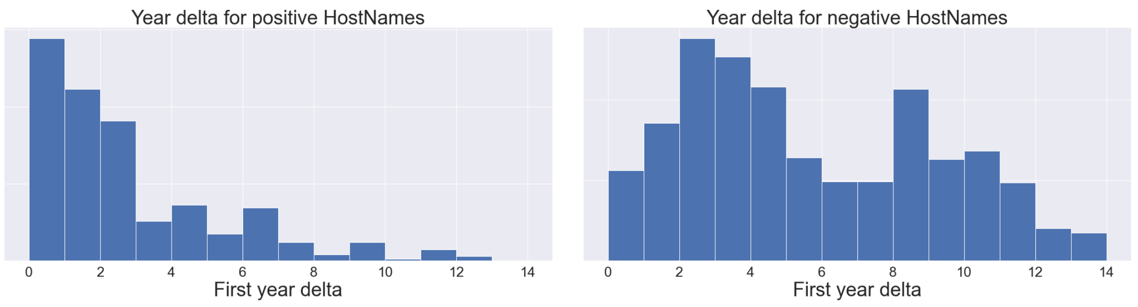


Figure 6.8: First certificate date in years

For better visualization, we have extracted only year information from the first certificate date feature and calculated the number of years to the current one (2020). As depicted on Figure 6.8, general behavior seems to be that positive hostnames are younger than negative ones, which probably has an influence on previously mentioned features like total hostnames.

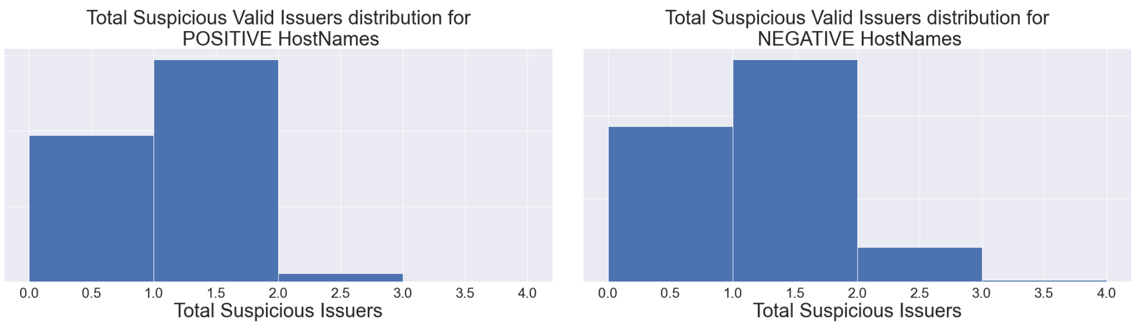


Figure 6.9: Total suspicious valid issuers distributions

No difference in distributions of total suspicious valid issuers on Figure 6.9.

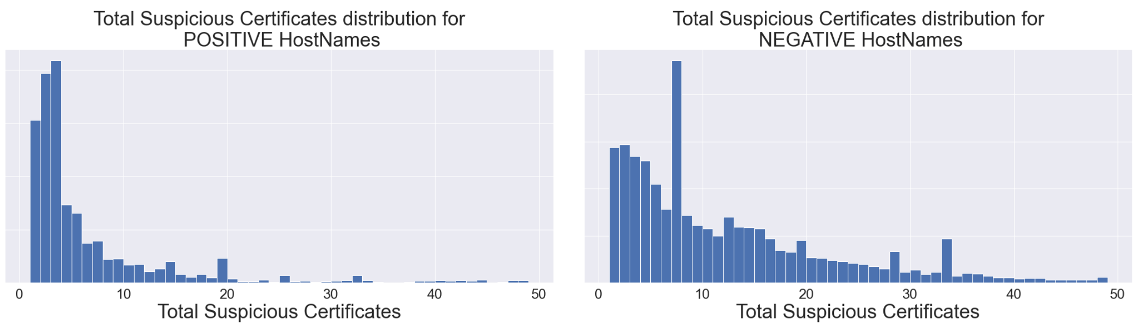


Figure 6.10: Total suspicious certificates distributions

As in the case of total certificates, positive hostnames, as depicted on Figure 6.10, have less suspicious certificates historically.

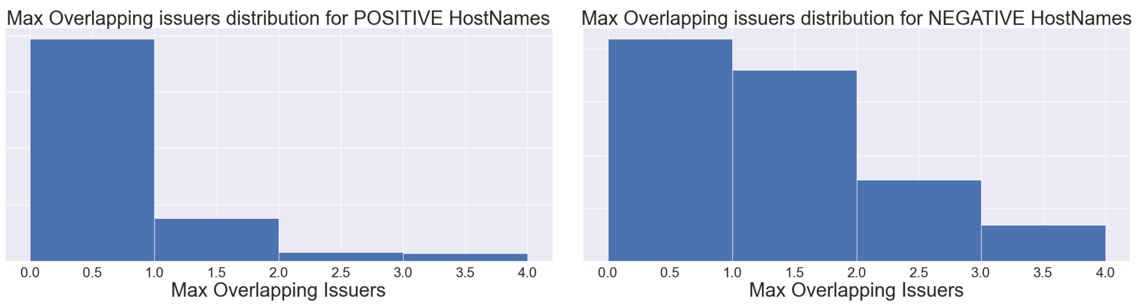


Figure 6.11: Maximum overlapping issuers distributions

Because positive hostnames have fewer issuers and thus also less overlapping issuers (Figure 6.11).

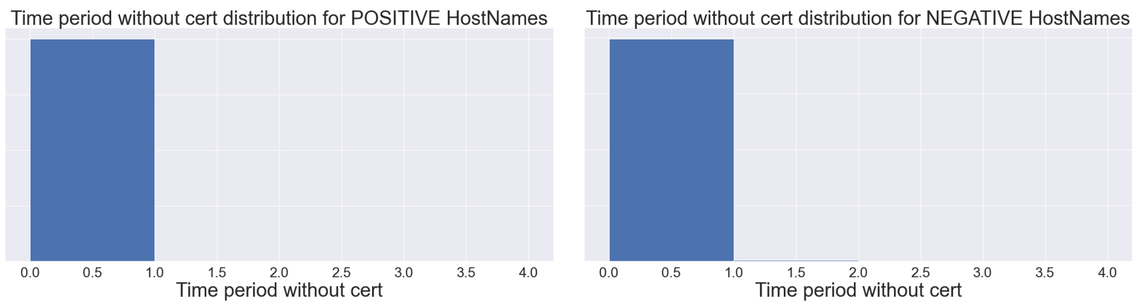


Figure 6.12: Time period without certificate feature distributions

Both distributions show that feature representing time period without a certificate (Figure 6.12) has literally no effect and we can think of removing it from extracted features.

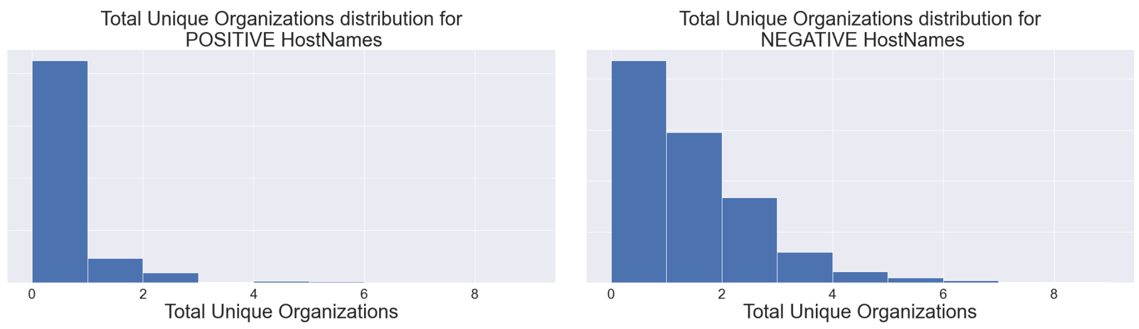


Figure 6.13: Total Unique Organizations distributions

Positive samples tend to have less unique organizations (Figure 6.13). However, global statistics at the beginning of this section showed that the majority of certificates has organization field unfilled.

Validity				**Is WildCard Root**			
Positive		**Negative**		**Positive**		**Negative**	
1435	695	949035	183006	0	2130	4	1132037
67 %	33 %	84 %	16 %	0 %	100 %	0 %	100 %
Valid	**Nonvalid**	**Valid**	**Nonvalid**	**True**	**False**	**True**	**False**

Table 6.3: Validity of HostName and Is WildCard Root

The left side of Table 6.3 shows the number of valid certificates between both classes. 67 % of positive hostnames and 84 % of negative hostnames have a valid certificate. On the right side is a statistic for is wildcard root feature showing that this feature is almost useless as it was true only for 4 samples out of both classes and we can think about removing it.

Absolute/WildCard/Mixed Coverage					
Positive			**Negative**		
927	1116	87	492778	456396	182867
44 %	52 %	4 %	44 %	40 %	16 %
Absolute	**WildCard**	**Mixed**	**Absolute**	**WildCard**	**Mixed**

Table 6.4: Only Absolute/WildCard/Mixed coverage statistics

Statistics in Table 6.4 show hostname coverage only and only by absolute or wildcard certificates and also a number of hostnames that are covered by both. For the positive sample, there are fewer absolute certificates than wildcards and as was expected, number of hostnames with mixed coverage is very small. On the other hand, in the negative sample, there are more hostnames with absolute coverage. The observer difference is however not significant.

To conclude this chapter, we have observed all extracted features with distribution comparison. There are some patterns for total-based features that differ between positive and negative samples, however, this behavior might be caused by the general age (Figure 6.8) of positive respectively negative hostnames. Is wildcard root and period without certificate features might be excluded from the feature list as both shows literally no patterns.

Chapter 7

Detection

This chapter covers detection experiments with a suitable selected classifier model. We describe *Random decision forest* that has shown to be very efficient in a field of general recognition and thus has been chosen as the main and only experimental model. Note that, this thesis does not aim to find the most suitable malware recognition model with the use of certificate-based features. Rather than that has been chosen a classification model that has previously shown to be an effective option in a field of malware detection with the use of highly class imbalanced data, where observing true positives is like looking for a needle in a haystack. Random Forest is also a type of *non-parametric* model which does not assume any shape that the learned mapping function should have (parametric models), instead, it tries to estimate a mapping that gets as close as possible to training points [39].

7.1 Random Decision Forest

Random decision forest, as a general name of this model, can be used for various problems starting with classification, regression, and other recognition tasks such as *manifold learning* [40]. Our task is malware recognition and multiclass classification therefore all detection experiments are performed with *Random Forest Classifier* model. By Random Forest, we are here referring to the most common and used implementation, the Breiman forest [55] which uses *CART algorithm* [56] for the model's training phase.

Decision Forests are one of so-called *ensemble methods* which train multiple models of input data and then aggregate their classification results. In the case of Random Forests, multiple *Decision Tree* models are trained on random input feature subsets with the use of *bagging* method, therefore it is crucial to describe underlying concepts of the Decision Tree model to understand how this ensemble method works in general. In performed experiments, trained Random Forest models aggregate results from Decision Trees by so-called *majority vote* method, which takes results from all trained trees and resolute classification decision is made by the most occurring class [39].

7.1.1 Decision Trees

Decision tree model resembles a mathematical tree model known from graph theory and the CART algorithm specifically implements a binary tree. Tree consists of internal nodes, edges, and root node whilst each internal node is used as an arbitrate decision point that splits input data (based on defined test function) into distinct parts starting with the tree *root*. Input data are recursively propagated through the trained tree until one of the leaf nodes occurs which is accompanied by the final classification result.

As was stated in the text above, in all non-leaf nodes happens data split with defined test function. This is the most important part of the tree-building algorithm as it directly influences final classification results. The splitting function decides what would be the best split regarding to *data purity measure* and when the split at each node is as pure as possible, the more confident we are in resolute classification. As a purity measure, we use so-called *information gain (IG)* as an objective function which is defined as an expected reduction of *entropy (H)* when the split is created. Information gain in one internal node S is defined as follows [39, 40].

$$IG(S) = H(S) - \frac{|S_l| \cdot H(S_l) + |S_r| \cdot H(S_r)}{|S|}$$

$$H(X) = -\sum_{c \in Y} p(c) \log p(c)$$

$p(c)$ is a frequency of class c in split X

For better understanding, when split X contains elements of only a single class, $p(c)$ equals 1, thus $\log p(c)$ equals zero and whole entropy of split X equals 0. On the other hand, when each class has the same frequency and therefore all classes are equally distributed, calculated entropy is maximal and the split is *"impure"*. During the training phase, tree is recursively built by maximization of information gain measure at each node. During the testing phase, the decision tree recursively applies all previously defined (learned) tests on unseen data points until it reaches a leaf node where the resolute classification decision is made, either by most occurring class in the split or by class distribution.

Generally speaking, the decision tree is a model that is easy to interpret and explain with an ability to handle categorical data (no need for one-hot encoding). On the other hand, the trained model can result in high variance (overfitting). Last but not least, trees are very sensitive to data changes, thus even small changes can have a big impact on trained model. The strength of the Random Forest model comes from the bagging procedure which tends to minimize the previously mentioned high variance of the tree model and extensively improve its performance [39].

All performed experiments have been done using Spark library for machine learning called MLlib [57]. However, this library has shown to be ineffective [58] and specifically for Random Forest classifier, we use improved *Optimized Random Forest on Apache Spark (ORaF)* implementation [59].

7.1.2 (Hyper)parameters

Parameters of the model are variables which given algorithm learns during the training phase, such as individual nodes and splits in decision tree model or weights in neural networks.

Hyperparameters are used as an additional training algorithm's input that influences final model behavior. It varies from a given method but usually has to do with underlying concepts of selected model. In the case of Random Forest model, most of the hyperparameters are derived from the Decision Tree model [40].

Number of estimators specifies an actual number of trained Decision Trees in Random Forest Model.

Purity measure decides how will be the data splits in trees created. Besides *Information Gain*, another commonly used measure is *Gini Impurity*.

Maximum depth of each tree.

Minimum Information Gain for a split to be considered at a tree node.

7.2 Experiments

The final section of this thesis summarizes all performed detection experiments and by the same token as in analysis Chapter 6, detection experiments have been performed on the same two distinct time periods: August 2020 and October 2020. First three weeks of each time interval have been used as a training sample and the remaining month interval as a testing sample.

For comparison of our enhanced model, we had to first define so-called *baseline model* consisting of **342** already existing features, which belongs to categories below.

Anomaly detectors - Detectors built on HTTP access logs telemetry data [60].

Global information - Popularity features from various sources such as Alexa [61].

TLS Handshake features - For instance, features based on used TLS cipher suites.

Baseline model is therefore our starting point for model comparison experiments. Other following experiments incorporate all features in baseline models as well as features that have been extracted from Certificate Transparency. Each model has been trained with the following hyperparameters.

Hyperparameters	
Number of estimators (trees)	70
Purity measure	Information Gain (Entropy)
Maxium depth	50
Minimum Information Gain	0

Table 7.1: Hyperparameters used for Random Forest classifier training

Experiment evaluation contains a binary confusion matrix included primarily for orientation between positives and negatives. Substantial included measure is observed number of classes which given model classified as positive with precision over 0.9. Last but not least, to see how specific features roughly perform, all models enhanced with new features have a feature importance table attached. This table contains each new feature with its position among other features the model was trained with and also count measure. Count represents how many times has given feature appeared as a decision in a tree node in all model's trees divided by total number of trees, thus it represents an average measure for one tree.

7.3 August 2020

This section contains 3 trained models: baseline, model with only a boolean CT indicator, and model with all extracted features.

7.3.1 Baseline

		Actual	
		Positive	**Negative**
Prediction	**Positive**	9839	22784
	Negative	25513	-

Table 7.2: Confusion Matrix for August 2020 baseline model

Table 7.2 describes the confusion matrix of the baseline model with 9839 correctly classified positive samples. Because of the large amount of true negatives, to speedup computation, they are filtered out in the evaluation pipeline and we do not have them available. TNs are also not necessary for calculation of precision and recall metrics.

Precision >=	**Total classes**	**TPs**	**FPs**	**Average Recall**
1	13	1205	0	0.40
0.95	15	4854	178	0.47
0.9	17	4926	184	0.50

Table 7.3: Total TP classes with precision over 0.9

Total of 17 different malware classes have been classified with precision over 0.9. 15 of them with precision over 0.95 and 13 with precision equals to 1. That means, when trained model classified one of those 13 classes, it was correct in 100 % of cases. The average recall of those 13 classes was 0.4 and thus we can say 40 % of positive samples from 13 different classes have been correctly classified on average.

7.3.2 CT indicator

This model was a baseline model enhanced only of a boolean indicator whether we have or do not have extracted features from Certificate Transparency for given hostname. This experiment is performed primarily to identify how strong signal this indicator is.

		Actual	
		Positive	**Negative**
Prediction	**Positive**	9013	22925
	Negative	26339	-

Table 7.4: Confusion Matrix for August 2020 CT indicator model

Precision >=	**Total classes**	**TPs**	**FPs**	**Average Recall**
1	14	340	0	0.46
0.95	16	3969	163	0.52
0.9	17	3986	164	0.50

Table 7.5: Total TP classes with precision over 0.9 of CT indicator model

Table 7.4 and Table 7.5 above state that indicator feature did not significantly improve the baseline model as the total classes with precision equal 1 increased only by 1, with average recall 0.46. On the other hand, note that a total number of true positives decreased and we can assume that this indicator feature is not very useful.

Feature Importance		
Position	**Feature**	**Count**
47	ctIndicator	34.83

Table 7.6: Feature importance statistics for CT indicator model

In this case, we also have available feature importance measure, that has been described above. In this case, the indicator feature has placed on the 47th position out of 343 features the model was trained with and in average, it was used almost 35 times in each decision tree model.

7.3.3 CT Features

Finally, this section contains results for a model trained with extracted features from August 2020.

		Actual	
		Positive	**Negative**
Prediction	**Positive**	10086	22451
	Negative	25266	-

Table 7.7: Confusion Matrix for August 2020 CT features model

Precision >=	Total classes	TPs	FPs	Average Recall
1	20	2032	0	0.68
0.95	22	6320	58	0.71
0.9	26	9856	276	0.72

Table 7.8: Total TP classes with precision over 0.9 of CT features model

On Table 7.7 confusion matrix we can see the number of true positives has slightly increased over the August 2020 baseline model. Nevertheless, the major classification improvement of this enhanced model can be seen on Table 7.8. Total of 26 classes has been correctly classified with precision over 0.9, 22 classes with precision over 0.95, and 20 classes with precision equal to 1. The average recall for those 20 classes is 0.68 thus we have correctly classified 68 % of positive samples out of them on average.

And last but not least in the Table 7.9 we can see all features extracted from Certificate Transparency sorted in descending order by the right-most column representing the average feature count in one tree. Out of 357 used features, we can see that 7 of them are in the top 20 starting with totalHostNames feature. On the other hand, and as was expected, the least used designed features are timePeriodWithoutCertificate and isWildCardRoot which already showed to be ineffective in Chapter 6.

Feature Importance		
Position	**Feature**	**Count**
13	totalHostNames	73.03
14	totalSusCertificates	71.29
15	totalCerts	69.94
16	totalUniqueHostNames	68.71
17	wildCardCoverageCount	67.39
18	absoluteCoverage	64.71
20	firstCertDate	63.74
23	totalUniqueWildCardHostNames	61.80
24	totalValid	60.84
39	totalIssuers	45.10
43	totalSusValidIssuers	41.13
45	totalUniqueOrganizations	39.71
47	maxOverlappingIssuers	38.44
87	timePeriodWithoutCertificate	10.56
130	isWildCardRoot	5.57

Table 7.9: Feature importance statistics for CT features model

7.4 October 2020

The same detection experiment as in the case of August 2020 (except the CT indicator one) has been performed in October 2020 time period.

7.4.1 Baseline

		Actual	
		Positive	**Negative**
Prediction	**Positive**	3058	201
	Negative	5071	-

Table 7.10: Confusion Matrix for October 2020 baseline model

Baseline model has correctly classified 3058 positive samples on October 2020 time period out of which were classified 7 total malware classes with precision equals to 1 and an average recall of 0.48.

Precision >=	Total classes	TPs	FPs	Average Recall
1	7	310	0	0.48
0.95	9	916	8	0.58
0.9	11	3005	185	0.59

Table 7.11: Total TP classes with precision over 0.9 of baseline model

7.4.2 CT Features

		Actual	
		Positive	**Negative**
Prediction	**Positive**	4281	32
	Negative	3848	-

Table 7.12: Confusion Matrix for October 2020 CT features model

Precision >=	Total classes	TPs	FPs	Average Recall
1	16	1479	0	0.67
0.95	20	4274	28	0.69
0.9	20	4274	28	0.69

Table 7.13: Total TP classes with precision over 0.9 of CT features model

Table 7.12 and Table 7.13 contains results of the October 2020 model enhanced with CT features. In comparison with previous baseline model and by the same token as in August 2020 models, we can see significant classification improvement. Total number of true positives has increased by more than a thousand but most importantly this model was able to classify 20 classes with precision over 0.95. 16 classes then with precision equal to 1 and average recall of 0.67.

Table 7.14 covers feature importance for October 2020 model. Resulting table shows that new features are still in the lead out of all 357 total used features starting with totalSusCertificates feature from the very first list position which was present almost 103 times in each trained Decision Tree. This quite resembles Table 7.9 from August as the importance and order have not changed that much as well as the last two features in the list.

Feature Importance		
Position	**Feature**	**Count**
1	totalSusCertificates	102.89
3	firstCertDate	99.69
5	totalHostNames	95.36
6	totalUniqueHostNames	94.43
7	wildCardCoverageCount	91.89
9	totalCerts	90.13
10	totalUniqueWildCardHostNames	87.17
12	absoluteCoverage	83.54
14	totalValid	81.31
28	totalIssuers	64.63
34	totalUniqueOrganizations	61.49
41	totalSusValidIssuers	55.91
44	maxOverlappingIssuers	53.11
91	timePeriodWithoutCertificate	10.21
181	isWildCardRoot	2.81

Table 7.14: Feature importance statistics for CT features model

Chapter 8

Conclusion

In this thesis, we focused on exploring Certificate Transparency as a possible data source for extraction of features and their application in the field of malware detection. It was necessary to study TLS protocol and especially focus on Public Key Infrastructure which uses the concept of certificates to provide source integrity verification. Essential part of this thesis is the Certificate Transparency service which fixes infrastructural flaws in TLS protocol and generally mitigates certificate-based threats. Except that, its public and transparent nature allow us to use it as a historical data source of TLS certificates for an enormous number of hostnames.

As a very first practical step was created draft of features that could be extracted out of certificate history, then we continued with implementation of extraction pipeline in Spark environment. Several optimization obstacles have been encountered during this implementation phase which resulted in a change of the approach to decrease the amount of processed data at once. Input hostnames were limited to those that appeared in telemetry during **August 2020** respectively **October 2020** time period.

Extracted features have been analyzed by plotting distributions between positive and negative malware class and visually comparing observed patterns. Given the analysis, the strongest observed signal was **firstCertificateDate** feature which has shown us that positive hostnames are generally younger than negative ones. Based on that, we might assume that other designed features (such as **totalHostnames**) are affected by this behavior and it is appropriate to perform additional analysis. Nevertheless, based on plotted data distributions, two features have proven to be completely useless, specifically **isWildcard-Root** and **timePeriodWithoutCertificate**.

Final malware detection experiments have been performed with Random Forest classifier, current state-of-the-art solution for structured data, that has shown to be effective in the field of malware multinomial classification. For both time periods have been defined so-called baseline models consisting of already existing features which we then used for comparison with models enhanced with newly designed ones.

For **August 2020** time period, the baseline model was able to correctly classify 17 different malware classes with precision over 90 %, however the model with CT features was able to correctly classify **26 classes with a precision of over 90 %**.

For **October 2020** time period, the baseline model was able to correctly classify 11 different malware classes with precision over 90 %, however the model with CT features was able to correctly classify **20 classes with a precision of over 90 %**.

The models' performances have significantly improved in both cases and thus we can assume that the history of certificates is indeed a useful data source. For both time periods, we have also got so-called **feature importance** statistics that tell us how many times is given feature used as an arbitrate decision point in the Decision Tree model. In both cases, around 50 % of new features were in the top 20 out of 357 total features.

Designed features have shown to be useful as they improved classification with random forest model. However, and as was stated before, it is appropriate to perform additional feature analysis to see whether the firstCertificateDate signal is the strongest among others, and by the same token, there are isWildCardRoot and timePeriodWithoutCertificate ineffective features that could be removed in next experiments.

Speaking of future work, current pipeline implementation consists of filter operations that reduce the amount of data processed at once. However, it is impractical to use additional hostname filter data that are used as an input to the job execution. To automate the whole computation pipeline, implementation would need some batch-like approach which would reduce the amount of data as well.

Bibliography

[1] Google. HTTPS encryption on the web, . URL https://transparencyreport.google.com/https/overview?hl=en. Accessed on 2020/12/18.

[2] Google. Community : Certificate Transparency, . URL https://certificate.transparency.dev/community/. Accessed on 2020/12/18.

[3] F5. What is the CIA Triad?, 2019. URL https://www.f5.com/labs/articles/education/what-is-the-cia-triad. Accessed on 2020/12/07.

[4] E. Rescorla. The transport layer security (tls) protocol version 1.3. RFC 8446, RFC Editor, August 2018. URL https://www.rfc-editor.org/rfc/rfc8446.txt. Accessed on 2020/12/18.

[5] Jon Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. URL http://www.rfc-editor.org/rfc/rfc793.txt. Accessed on 2020/12/18.

[6] J. Postel. User datagram protocol. STD 6, RFC Editor, August 1980. URL http://www.rfc-editor.org/rfc/rfc768.txt. Accessed on 2020/12/18.

[7] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL https://doi.org/10.1145/359340.359342.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. doi: 10.1109/TIT.1976.1055638.

[9] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177): 203–209, January 1987. ISSN 0025-5718.

[10] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001. ISBN 0-8493-8523-7. URL http://www.cacr.math.uwaterloo.ca/hac/. Accessed on 2020/12/07.

[11] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. URL http://www.rfc-editor.org/rfc/rfc5246.txt. Accessed on 2020/12/18.

[12] Nick Sullivan. Cloudflare: Padding oracles and the decline of CBC-mode cipher suites, 2016. URL `https://blog.cloudflare.com/padding-oracles-and-the-decline-of-cbc-mode-ciphersuites/`. Accessed on 2020/12/18.

[13] Nick Sullivan. Cloudflare: A Detailed Look at RFC 8446 (a.k.a. TLS 1.3), 2018. URL `https://blog.cloudflare.com/rfc-8446-aka-tls-1-3/`. Accessed on 2020/07/12.

[14] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[15] Specification for the advanced encryption standard (aes). Federal Information Processing Standards Publication 197, 2001. URL `http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf`.

[16] Cloudflare. What is a TLS handshake?, . URL `https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/`. Accessed on 2020/07/12.

[17] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. RFC 5280, RFC Editor, May 2008. URL `http://www.rfc-editor.org/rfc/rfc5280.txt`. Accessed on 2020/12/18.

[18] Cloudflare. What is an SSL certificate?, . URL `https://www.cloudflare.com/learning/ssl/what-is-an-ssl-certificate/`. Accessed on 2020/07/12.

[19] Dani Grant. CLoudflare: Introducing TLS with Client Authentication, 2017. URL `https://blog.cloudflare.com/introducing-tls-client-auth`. Accessed on 2020/07/12.

[20] ssl.com. Browsers and Certificate Validation, 2018. URL `https://www.ssl.com/article/browsers-and-certificate-validation/#certification-paths-and-path-processing`. Accessed on 2020/12/18.

[21] M. Nystrom and B. Kaliski. Pkcs #10: Certification request syntax specification version 1.7. RFC 2986, RFC Editor, November 2000. URL `https://www.rfc-editor.org/rfc/rfc2986.txt`. Accessed on 2020/12/18.

[22] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. URL `http://www.rfc-editor.org/rfc/rfc2616.txt`. Accessed on 2020/12/18.

[23] E. Rescorla. Http over tls. RFC 2818, RFC Editor, May 2000. URL `http://www.rfc-editor.org/rfc/rfc2818.txt`. Accessed on 2020/12/18.

[24] Google. Certificate Transparency, 2013. URL `https://www.certificate-transparency.org`. Accessed on 2020/11/23.

[25] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, RFC Editor, June 2013. URL https://www.rfc-editor.org/rfc/rfc6962.txt. Accessed on 2020/12/18.

[26] StatCounter Global Stats, 2020. URL https://gs.statcounter.com/. Accessed on 2020/11/23.

[27] Threatpost. Final Report on DigiNotar Hack Shows Total Compromise of CA Servers, 2012. URL https://threatpost.com/final-report-diginotar-hack-shows-total-compromise-ca-servers-103112/77170/. Accessed on 2020/11/23.

[28] Google. What is Certificate Transparency?, 2013. URL https://www.certificate-transparency.org/what-is-ct. Accessed on 2020/11/23.

[29] Certificate transparency log policy, 2017. URL https://github.com/chromium/ct-policy/blob/master/log_policy.md. Accessed on 2020/11/23.

[30] Google. Certificate Transparency for Site Operators, 2013. URL https://chromium.googlesource.com/chromium/src/+/master/net/docs/certificate-transparency.md#basic. Accessed on 2020/11/23.

[31] Google. How Certificate Transparency Works, 2013. URL https://www.certificate-transparency.org/how-ct-works. Accessed on 2020/12/13.

[32] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. URL http://www.bitcoin.org/bitcoin.pdf. Accessed on 2020/07/12.

[33] D. S. V. Madala, M. P. Jhanwar, and A. Chattopadhyay. Certificate Transparency Using Blockchain. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 71–80, 2018. doi: 10.1109/ICDMW.2018.00018.

[34] Google. Certificate Transparency - Known Logs, 2013. URL https://www.certificate-transparency.org/known-logs. Accessed on 2020/11/23.

[35] Certificate Transparency for Untrusted CAs, 2016. URL https://security.googleblog.com/2016/03/certificate-transparency-for-untrusted.html. Accessed on 2020/11/23.

[36] How Log Proofs Work, 2013. URL https://www.certificate-transparency.org/log-proofs-work. Accessed on 2020/12/13.

[37] Sectigo. What Google Chrome's New Certificate Transparency Requirement Means to Your Organization, 2018. URL https://sectigo.com/resource-library/what-google-chromes-new-certificate-transparency-requirement-means-to-your-organization. Accessed on 2020/11/23.

[38] Bingyu Li, Fengjun Li, Ziqiang Ma, and Qianhong Wu. Exploring the security of certificate transparency in the wild. In Jianying Zhou, Mauro Conti, Chuadhry Mujeeb Ahmed, Man Ho Au, Lejla Batina, Zhou Li, Jingqiang Lin, Eleonora Losiouk, Bo Luo, Suryadipta Majumdar, Weizhi Meng, Martín Ochoa, Stjepan Picek, Georgios Portokalidis, Cong Wang, and Kehuan Zhang, editors, *Applied Cryptography and Network Security Workshops*, pages 453–470, Cham, 2020. Springer International Publishing. ISBN 978-3-030-61638-0.

[39] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 7 edition, 2014. ISBN 978-1-4614-7137-0. URL `http://faculty.marshall.usc.edu/gareth-james/ISL/`. Accessed on 2020/12/13.

[40] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. *Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning*, volume 7. NOW Publishers, foundations and trends® in computer graphics and vision: vol. 7: no 2-3, pp 81-227 edition, January 2012. URL `https://www.microsoft.com/en-us/research/publication/decision-forests-a-unified-framework-for-classification-regression-density-estimation-manifold-learning-and-semi-supervised-learning/`.

[41] Google. Classification: Accuracy, . URL `https://developers.google.com/machine-learning/crash-course/classification/accuracy`. Accessed on 2020/12/13.

[42] Google. Classification: True vs. False and Positive vs. Negative, . URL `https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative`. Accessed on 2020/12/13.

[43] Google. Classification: Precision and Recall, . URL `https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall`. Accessed on 2020/12/13.

[44] B. Claise. Cisco systems netflow services export version 9. RFC 3954, RFC Editor, October 2004. URL `http://www.rfc-editor.org/rfc/rfc3954.txt`. Accessed on 2020/12/18.

[45] Cisco. Introduction to Cisco IOS NetFlow - A Technical Overview, 2012. URL `https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow/prod_white_paper0900aecd80406232.html`. Accessed on 2020/12/18.

[46] D. McGrew and B. Anderson. Enhanced telemetry for encrypted threat analytics. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, pages 1–6, 2016. doi: 10.1109/ICNP.2016.7785325.

[47] Google. The List of Chrome CT logs, . URL https://www.gstatic.com/ct/log_list/log_list.json. Accessed on 2020/12/13.

[48] ZMap. Zcertificate - github. URL https://github.com/zmap/zcertificate. Accessed on 2020/12/13.

[49] ZMap. The ZMap Project, . URL https://zmap.io/. Accessed on 2020/12/13.

[50] ZMap. ZCrypto - Schemas, . URL https://github.com/zmap/zcrypto/blob/d3043756f9f43af2ff0ecd05ff83e3e2abc26437/zcrypto_schemas/zcrypto.py#L308. Accessed on 2020/12/13.

[51] Apache Software Foundation. Apache Parquet, 2018. URL https://parquet.apache.org/. Accessed on 2020/12/13.

[52] Apache Software Foundation. RDD Programming Guide. URL https://spark.apache.org/docs/latest/rdd-programming-guide.html. Accessed on 2020/12/13.

[53] Internet Security Research Group (ISRG). Let's Encrypt - Free SSL/TLS Certificates. URL https://letsencrypt.org/. Accessed on 2020/12/18.

[54] Sectigo Limited. ComodoCA Official Site | Comodo SSL Certificates Official Site. URL https://ssl.comodo.com/. Accessed on 2020/12/18.

[55] Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[56] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[57] Apache Software Foundation. MLlib - Apache Spark's scalable machine learning library, 2018. URL https://spark.apache.org/mllib/. Accessed on 2020/12/16.

[58] Starosta Radek. Distributed algorithms for decision forest training in the network traffic classification task. Bachelor's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2018.

[59] Cisco. ORaF (Optimized Random Forest on Apache Spark), 2018. URL https://github.com/cisco/oraf. Accessed on 2020/12/16.

[60] Martin Grill. *Combining network anomaly detectors*. PhD thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2016.

[61] Alexa Internet. Keyword Research, Competitive Analysis, & Website Ranking | Alexa. URL https://www.alexa.com/. Accessed on 2020/12/18.

Appendix A

October 2020 analysis

This additional section covers feature analysis on October 2020 time period in the same way as was performed in Chapter 6 on August 2020.

A.1 Global Statistics

Exported	Matched	Unmatched	Positive	Negative
3985523	3428876	556647	3235	3424005

Top 10 Issuers				Top 10 Organizations		
Issuer		**Certs**	**%**	**Organization**	**Certs**	**%**
cloudflare		38485425	55.9	cloudflare	38659133	56.2
let's encrypt		14492728	21.1	**blank**	27945685	40.6
comodo ca		9269812	13.5	netflix, inc.	294629	0.4
cpanel, inc.		2311821	3.4	incapsula inc	110820	0.2
digicert inc		1055602	1.5	google inc	85474	0.1
globalsign nv-sa		585554	0.9	google llc	45362	0.1
godaddy.com, inc.		558339	0.8	facebook, inc.	30517	0.1
sectigo limited		415604	0.6	firebase, inc.	23944	0.1
amazon		241196	0.4	fastly, inc.	22225	0.1
geotrust inc.		229828	0.3	microsoft	19078	0.1
Total	**693**	**68833323**	**100**	**314090**	**68833323**	**100**

A.2 Feature analysis

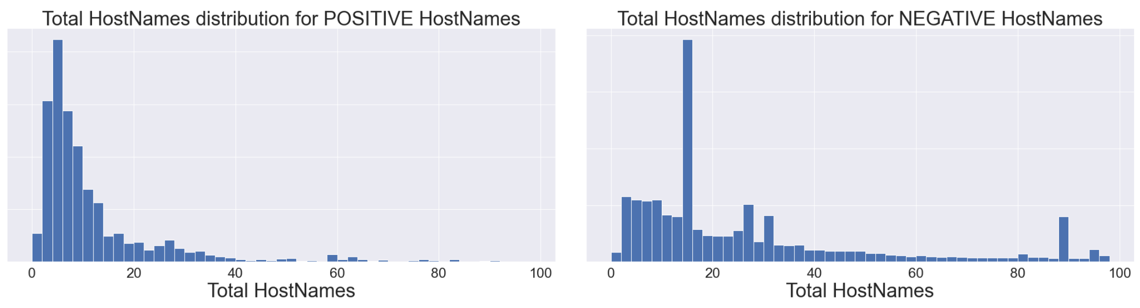


Figure A.1: Total HostNames feature distributions

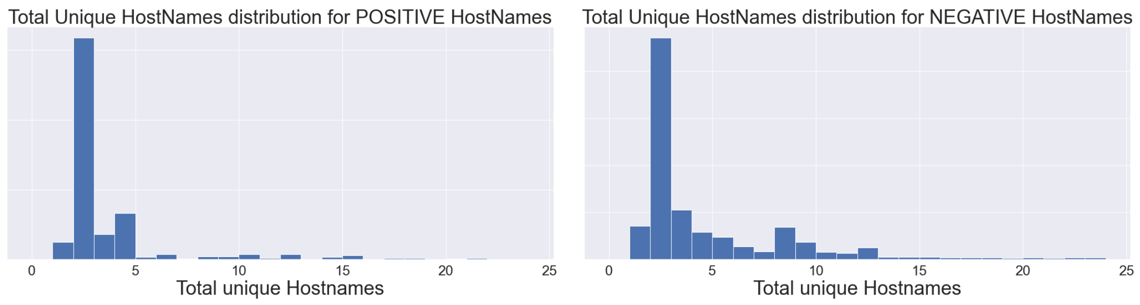


Figure A.2: Total Unique HostNames feature distributions

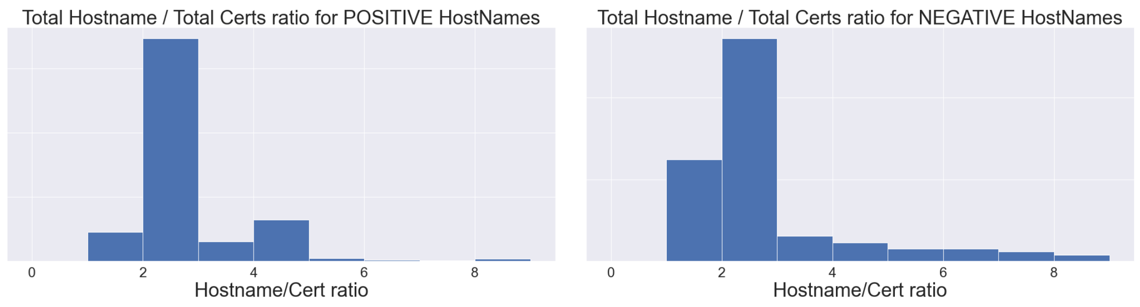


Figure A.3: Total average HostNames per certificate distributions

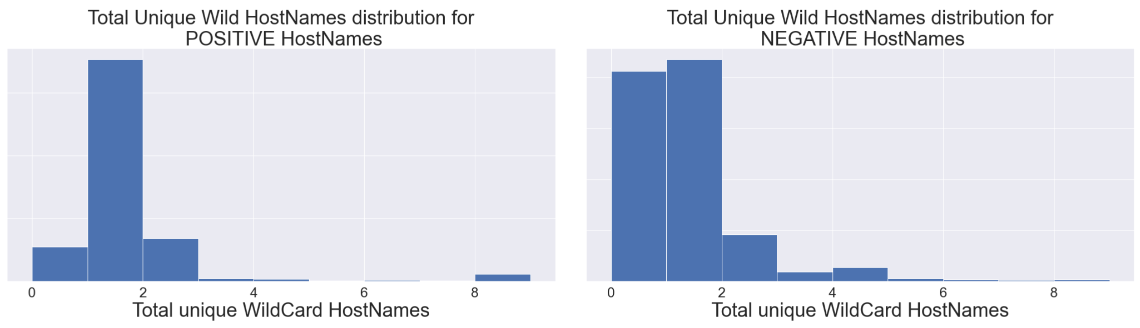


Figure A.4: Total Unique WildCard HostNames distributions

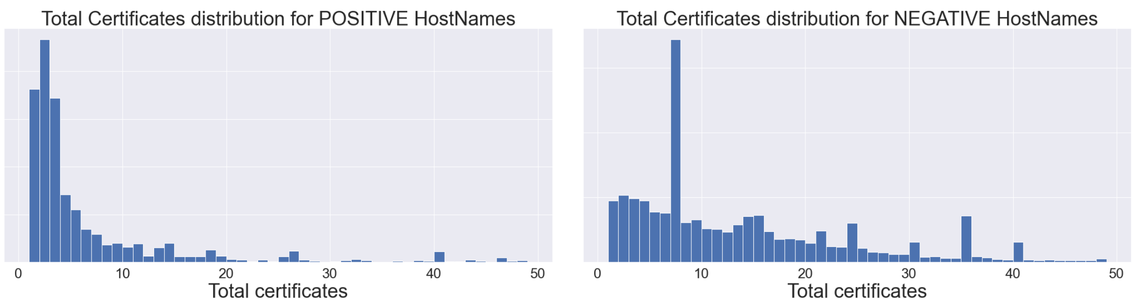


Figure A.5: Total certificates distributions

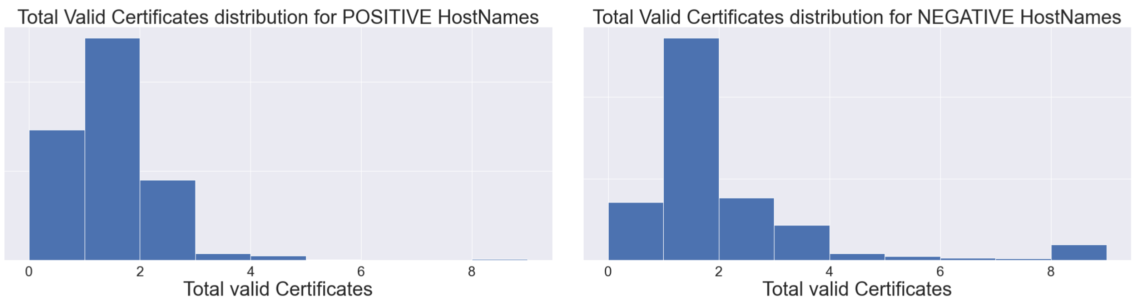


Figure A.6: Total valid certificates distributions

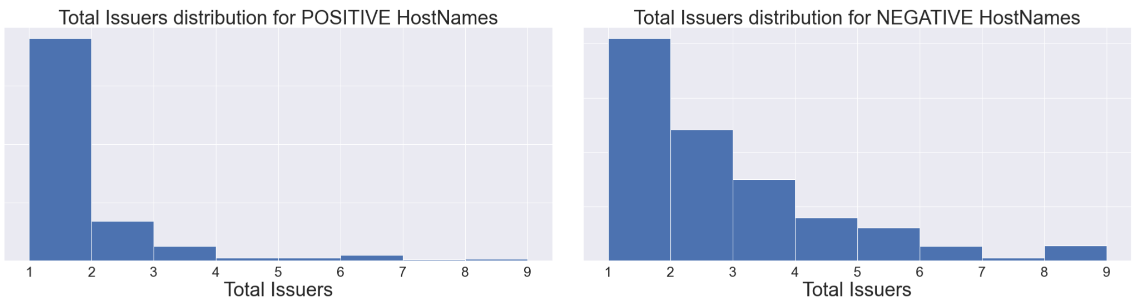


Figure A.7: Total issuers distributions

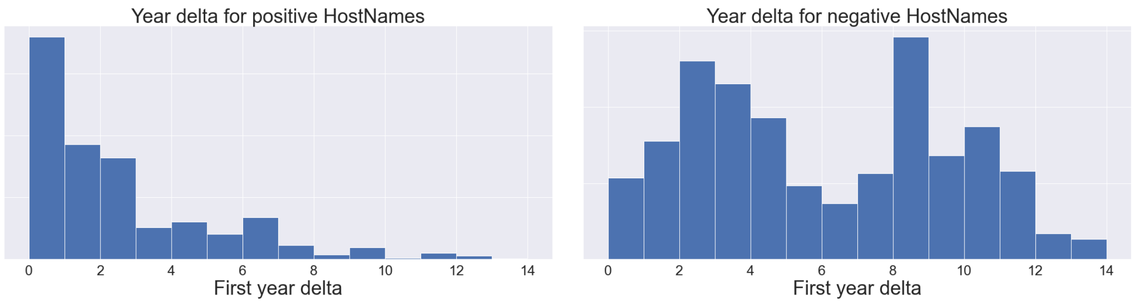


Figure A.8: First certificate date in years

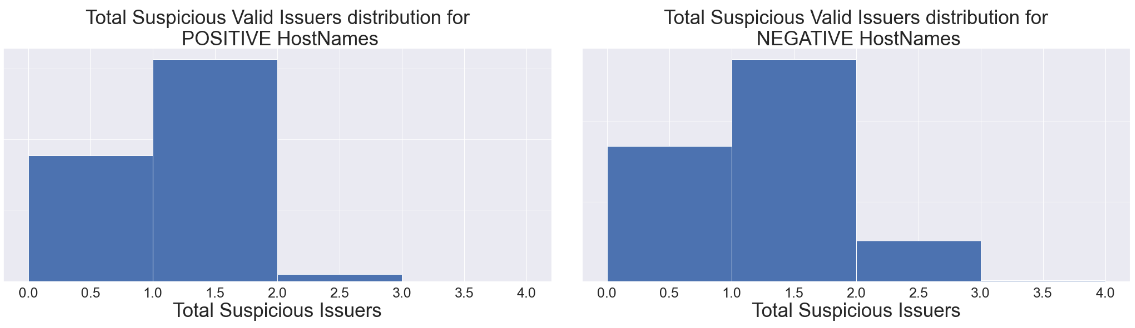


Figure A.9: Total suspicious valid issuers distributions

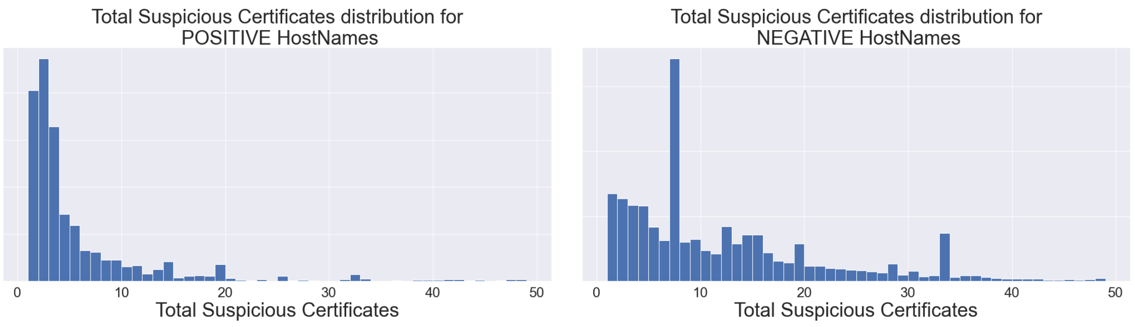


Figure A.10: Total suspicious certificates distributions

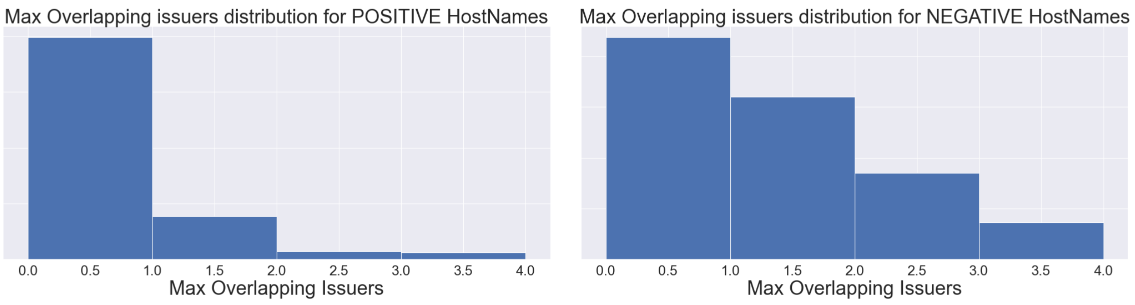


Figure A.11: Maximum overlapping issuers distributions

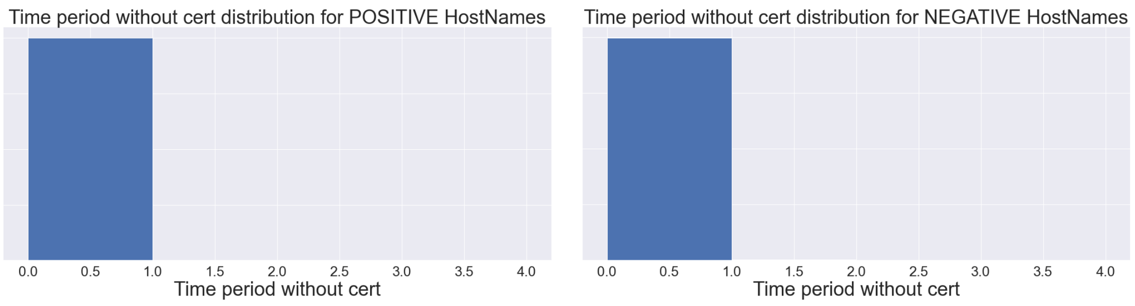


Figure A.12: Time period without certificate feature distributions

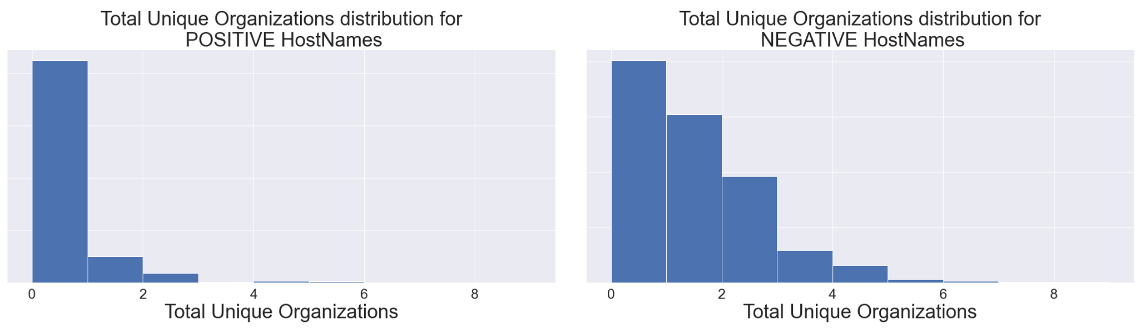


Figure A.13: Total Unique Organizations distributions

Validity			
Positive		**Negative**	
2291	944	2953415	470590
70.8 %	29.2 %	86.3 %	13.7 %
Valid	**Nonvalid**	**Valid**	**Nonvalid**

Is WildCard Root			
Positive		**Negative**	
0	3235	9	3423996
0 %	100 %	<0.1%	99.9%
True	**False**	**True**	**False**

Table A.1: Validity of HostName and Is WildCard Root

Absolute/WildCard/Mixed Coverage					
Positive			**Negative**		
1561	1552	122	1356222	1652245	415538
48.3 %	47.9 %	3.8 %	39.6 %	48.3 %	12.1 %
Absolute	**WildCard**	**Mixed**	**Absolute**	**WildCard**	**Mixed**

Table A.2: Only Absolute/WildCard/Mixed coverage statistics

Appendix B

Attached files

The attached files contain source code for data mining as well as code for hostname parsing, expanding and manipulation.

File	**Description**
ctMining.ipynb	Data mining experiments in Jupyter notebook
data.csv	Data sample for mining experiments
dataLoad.py	Data loading scripts for mining experiments
hostparser.py	Implemented scripts for hostname processing

Table B.1: List of attached files.