FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Detection of phishing domains in high-speed networks
Student:	Bc. Jakub Osmani
Supervisor:	Ing. Karel Hynek
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

Get acquainted with the network monitoring approaches and phishing phenomena. Design a phishing detection technique capable of processing ten thousand domains per second. Implement a prototype of a designed phishing detection approach in the form of a NEMEA [1] module. Test the implemented prototype and evaluate it using standard metrics (accuracy, speed, memory consumption) on the dataset provided by the thesis supervisor.

[1] T. Cejka, V. Bartos, M. Svepes, Z. Rosa and H. Kubatova, "NEMEA: A framework for network traffic analysis," 2016 12th International Conference on Network and Service Management (CNSM), 2016, pp. 195-201, doi: 10.1109/CNSM.2016.7818417.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 4 January 2023 in Prague.

Master's thesis

DETECTION OF PHISHING DOMAINS IN HIGH-SPEED NETWORKS

Bc. Jakub Osmani

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Karel Hynek
June 29, 2023

Contents

List of Figures

List of Tables

List of code listings

I would like to extend my thanks to my thesis supervisor, Ing. Karel Hynek, that provided me with much needed guidance, patience and encouragement. It is thanks to our regular meetings and productive discussions that the system was designed. I would also like to thank my parents, friends and my colleagues in Prague and Bratislava for their continued support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Section 2373(2) of Act No. 89/2012 Coll., the Civil Code, as amended, I hereby grant a non-exclusive authorization (licence) to utilize this thesis, including all computer programs that are part of it or attached to it and all documentation thereof (hereinafter collectively referred to as the "Work"), to any and all persons who wish to use the Work. Such persons are entitled to use the Work in any manner that does not diminish the value of the Work and for any purpose (including use for profit). This authorisation is unlimited in time, territory and quantity.

In Prague on June 29, 2023 .....

Abstract

This thesis is focused on the issue of phishing domain detection in high-speed networks, based on aggregated flow data of the CESNET network. The practical output of this thesis is a module for the NEMEA traffic analysis system, that filters large volumes of data and decides if a domain is malicious based on a set of indicators. As part of the thesis we also include an analysis of indicators unfit for this task as well as a practical dive into phishing in the Czech Republic.

Keywords phishing detection, traffic analysis, flow data, NEMEA, CESNET, TLS

Abstrakt

Diplomová práce se zabývá problematikou detekce phishingových domén ve vysokorychlostním provozu, a to na základě flow dat na síti CESNET. Výstupem je modul do systému NEMEA, který filtruje velký počet domén a rozhoduje zda je doména nebezpečná na základě sady indikátorů. Práce zároveň obsahuje popis indikátorů nevhodných pro detekci domén a praktickou analýzu phishingů v České Republice.

Klíčová slova detekce phishingu, analýza provozu, flow data, NEMEA, CESNET, TLS

Abbreviation List

AOL	America Online
API	Application Programming Interface
APWG	Anti Phishing Working Group
ARI	Automated Readability Index
CA	Certificate Authority
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart
CBC	Cipher Block Chaining
CDN	Content Delivery Network
CESNET	Czech Education and Scientific NETwork
CN	Common Name
CPU	Central Processing Unit
CSS	Cascading Style Sheet
CSV	Comma Seperated Values
CT	Certificate Transparency
DNS	Domain Name System
FQDN	Fully Qualified Domain Name
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure Hyper Text Transfer Protocol
IDN	International Domain Name
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IFC	TRAP Communication Interface
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
JSON	JavaScript Object Notation
MAC	Message Authentication Code
MPSV	Ministry of Labour and Social Affairs
NAT	Network Address Translation
NEMEA	Network Measurements Analysis
NUKIB	Czech National Cyber and Information Security Agency
OP	Observation Point
PSK	Pre-Shared Key
PaaS	Phishing-as-a-Service
RFC	Request For Comment
SLD	Second Level Domain
SMS	Short Message Service
SNI	Server Name Identifier
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLD	Top Level Domain
TLS	Transport Layer Security
TOR	The Onion Router
TRAP	Traffic Analysis Platform
UDP	User Datagram Protocol
URL	Universal Resource Locator
VPN	Virtual Private Network
eTLD	Effective Top Level Domain

Introduction

Our society is becoming more and more reliant on the usage of web services, and our identities being tied to the usage of the world wide web. This is clearly apparent in the statistics of internet access provided by Eurostat, where the amount of households in Europe with internet access was at 85% in 2016. The Czech Republic sat a little over 80% with their internet access in households, close to the then European average. [1]

To illustrate the issue more clearly, we may choose to look at the usage of social media. Using data provided by Statista, we may see that the penetration of active social media use in 2020 was at 53% in the Czech Republic. This is a staggering number, as active usage of social media had gone over the 50% mark. [2]

For most of the general populous, the general acceptance of internet usage and social media might not be alarming or even fruit for thought. From the perspective of cybersecurity practitioners however, this provides a wider attack surface for malicious actors. One such example could be social engineering and more specifically phishing, which is the main focus of this thesis. According to the European Union Agency for Cybersecurity, phishing had once again become the most common attack vector for initial access in 2022. [3] This means, that social engineering methods are still the most effective method for attackers to gain access to internal systems, credentials and personal information.

Phishing in the world of cybercrime is also proving to be a lucrative endeavour for the criminals and a rather costly problem for the rest. According to the IC3 (Internet Crime Complaint Center) Annual Report for the year 2020, the FBI received a total of $241,342$ reported phishing incidents, which amounted to a total of over 54$ million in losses. The total reported number of phishing incidents had increased year-to-year from 2019 to 2020 by over 100 thousand. [4] It is important to note, that these are the number of incidents *reported* to the IC3, and will be specific to the United States alone. For a more global picture, we may look to the report of the Anti Phishing Working Group (APWG) and their Phishing Trends Annual Report from the third quarter of 2022. In this Phishing Trends report, the group states that they recorded an all-time high of $1,270,883$ total phishing attacks, in Q3 alone, making it the worst quarter in APWG's history. [5]

To go a step further, not only is phishing still a prevalent threat, the technical skill and know-how required to successfully setup phishing is getting lower thanks to the emergence of *Phishing-as-a-Service (PaaS)*. In PaaS schemes, a malicious programmer creates a bundle usually tar getting a specific company or service, like the Czech Post. The customers of this programmer then buy said bundle, also known as a *Phishing Kit*, and use it with the purpose of automated phishing site deployment. All that is needed is access to funds and limited technical skill. [6]

In general we may say that the battle against social engineering attacks may be fought on two fronts. The first being prevention in the form of schooling and training, which is a valid way to combat phishing attempts and help in the recognition of phishing emails. It is however

important to note, that the general advice of looking for spelling errors and grammar mistakes has become outdated with the usage of higher quality phishing kits. [6] The second method involves technical measures, in the forms of monitoring, Intrusion Detection Systems (IDS) and multiple other methods. This thesis focuses on the technical aspect with a proof of concept detection system, that analyzes the metadata of captured HTTPS communication in the form of Flow Data, which is built upon the existing NEMEA framework and infrastructure utilized and built by CESNET.

First we will look at the State of the Art in Chapter 1, where we will provide a general overview of the information needed to fully understand the problems discussed in this thesis. Our overview includes a look at current phishing detection methods, where we also discuss their effectiveness and testing.

After the State of the Art chapter, we move on to the start of our practical contributions of our thesis. We start by discussing the data used for the preparation of our system, this is discussed in Chapter 2. Following the chapter on data, we provide some of our practical experience with the phishing landscape in the Czech Republic in Chapter 3. The dive into phishing is based on our numerous interactions with phishing, which includes active hunting, analysis and takedowns.

The last two chapters are focused on our design process, implementation and testing of the resulting phishing detection system. In Chapter 4 we go over our initial designs and ideas about the system, including our thoughts and reasoning behind the subsequent re-design. We also include relevant data and arguments on why certain tested indicators were discarded and deemed unusable for the detection of phishing domains. Lastly, in Chapter 5 we provide the results of our test runs on multiple sets of data, were we demonstrate the creation of a functioning proof of concept detection system.

Chapter 1
State of the Art

1.1 TLS

Transport Layer Security (TLS), is a security protocol originally proposed in January 1999, meant to be the successor to the Secure Sockets Layer (SSL) protocol. The original version of TLS, version 1.0, introduced major changes and security improvements over SSL version 2.0, this shall be discussed in Chapter 1.1.1. [7] Additionally, it is now possible to register valid TLS certificates via free services, which in the context of phishing and malicious misuse is a cause for concern. A short overview of free certificate providers will be discussed in Chapter 1.1.3.

A short overview of the TLS protocol is necessary, as the HTTPS communication analysed in this work is built upon the protocol, more specifically versions 1.2 and 1.3. The analysis of this traffic comes with its own specific challenges and requirements. For example the usage of Server Name Identification (SNI) data, which in itself was implemented as an extension of the TLS protocol, is vital for this thesis and the proposed detection methods. It is poignant to include that the SNI field is referred to as a list, and may therefore include more than one server name, as long as the names are of a different name_type. In the, possibly, near future the SNI field may no longer be visible thanks to extensions mentioned below. [8]

1.1.1 Quick Comparison of TLS with SSL

According to the proposed specification of SSL version 2.0 [9], the server would always send its certificate to the client, after receiving a *client-hello* message. This certificate would be included in the *server-hello* message. Here we may find the first major difference, and in the earliest version of TLS no less. Ever since TLS version 1.0, the sending of a server's certificate is flagged as *situational or optional*, along with the other messages of a *server-hello* message. [7] If we stay with the previously mentioned SNI field, added to the *client-hello* message, we may find another difference. This was added as an extension to the original TLS proposal, because of the real issue of Virtual Hosting. When Virtual Hosting was used, multiple domain names would share a server and thus domain name resolution would point to the same server. The usage of the optional SNI field allows a client to inform the server of which domain it is attempting to connect to. [8] The SNI field was then extended further with the proposal of Encrypted Client Hello messages in an IETF Draft, which proposes a scheme that results in SNI fields no longer being publicly readable and used in traffic analysis. [10]

From a security perspective, the SSL version 2.0 protocol is now considered insecure [11], and has been since March 2011. In this time, RFC-6176 [11] was published, which prohibited the use and support of SSL 2.0 due to security concerns that were not present in any versions of

TLS, after extensions. The issues included the usage of insecure hashes like MD5 and SHA-1 for Message Authentication Codes (MAC) and weaknesses that allowed a malicious party to perform Man-in-the-Middle (MitM) attacks against the handshake messages and to terminate sessions. RFC-6176 [11] also initiated changes in the TLS protocol, with the aim of phasing out the usage and support for the SSL v2.0 protocol.

More was to come in the form the Secure Socket Layer phase out, when RFC-7658 declared that the latest SSL version 3.0 protocol should also be considered insecure. [12] The call to arms stating the insecurity of SSL version 3 even stated, that any version of TLS is more secure than its predecessor. [12] To support this bold claim, several attacks and vulnerabilities are listed, including the POODLE attack [13] and the general flaw of SSL being in its inability to update itself. In-fact, the aforementioned POODLE attack [13] was deemed so critical, that the authors recommended the complete removal of SSL 3.0 as successful exploitation of the attack could lead to the extraction of HTTP headers such as cookies from encrypted communication. That was not the end of shakeups as TLS1.3 [14] brought in major changes from a privacy standpoint. The newer version enabled encrypted communication immediately after the completion of the KeyExchange section of the handshake. This means that the sent certificate is no longer visible to interceptors of traffic. Which is naturally cause for concern, when considering detection and identification of phishing domains.

1.1.2 TLS versions

While the previous section focused on a brief comparison between the now defunct SSL protocol and the modern TLS protocol, we feel that some elaboration on the TLS protocol itself is needed. Transport Layer Security brings with it its own unique set of challenges and specific behaviours that may not be familiar to all. While this thesis is not a deep dive into the ins and outs of all the version of TLS and encrypted communication in general, we believe that some surface level knowledge is required in order to understand the limitations in place, when performing network analysis on HTTPS traffic. All versions of TLS from 1.0 all the way up to 1.3 will be discussed in the chapters below, with the aim of providing a general description of the protocols and to attempt to summarize their differences.

1.1.2.1 TLS v1.0

Version 1.0 was the very first version of TLS, proposed by Christopher Allen and Tim Dirks in RFC-2246 [7] built with the aim of providing privacy and data protection during communication. This first iteration was built-upon the basis of SSL 3.0, and the RFC [7] itself states that:

> "The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate ..." [7]

While there may be differences between the TLS and SSL protocols, the option for downgrading from TLS 1.0 to SSL 3.0 had been provided. Before the exchange of data may occur, both sides participate in the Handshake Protocol, which allows the sides to agree on the used cipher-suites, exchange random values and check for the resumption of a previous session. If need be, certificates may also be exchanged between the client and the server via unencrypted traffic. This means, that monitoring services may collect data on exchanged certificates and initial handshake data, but none of the data sent later.

Later the Transport Layer Security protocol was extended by the addition of ciphersuites, SNI and more. Here-in lies the great advantage of TLS, as RFC-3546 [15] defined a method that allowed seamless usage and implementation of extensions. Using the proposed extended Client Hello method, clients could indicate to the server which extensions they intend to use or support.

The server would then respond with an extended server hello, essentially confirming the usage of said extensions.

1.1.2.2 TLS v1.1

Version 1.1 of the Transport Layer Security protocol was mainly a revision of the initial version, bringing with it some security fixes, clarifications and improvements in text. Introduced in RFC-4346 [16] the changes fixed a possible Padding Oracle attack against the CBC modes used in transport.

Premature closing of connections also no longer cause sessions to be non-resumable. The document also included informational notes on relevant attacks on TLS. No changes to the handshake protocol were initiated.

1.1.2.3 TLS v1.2

In August of 2008 a further revision of the TLS protocol was released. With the publication of RFC-5246 [17] TLS version 1.2 came to be, which continued the trend of improving the security of the protocol. Not only were there improvements from a security perspective, but additional functionality extensions were provided. When compared to version 1.1, version 1.2 also added support for a multitude of hashing algorithms as well as ciphersuites. Since the analysis of ciphersuites used is not the aim of this thesis, we will not go into further detail.

Lastly concerning certificates, if the client has no certificate to send then the client must send an empty list. The handshake agreement before encrypted communication was yet again unchanged. The document obviously includes many more changes, all of which were not mentioned in this chapter or thesis.

1.1.2.4 TLS v1.3

Ten years after the release of TLS version 1.2, the next version (1.3) was released in August of 2018 via RFC-8446. [14] As was tradition up to this point, the document served as a revision of RFC-5246 and added many improvements or changes. The proposed changes that are relevant to the matter of this thesis, were changes to the TLS Handshake protocol, in which there was given an option of changing the visibility of exchanged certificates as they may now be encrypted using Pre-Shared Keys (PSK). A necessary mechanism for the sharing of keys was also introduced, in the form of 0-RTT data. This mechanism allows some needed data to be added to the ClientHello message, thanks to which a PSK can be established during the Handshake phase of the protocol.

1.1.3 Free Certificate Providers

Our current encrypted communications ecosystem relies on a chain of trust, where a group of Certificate Authorities (CA) are trusted to identify that the certificate presented to us by the web server, is indeed valid and comes from a trusted certificate chain. [18] With HTTPS becoming the norm over plain and un-encrypted HTTP, an expected commercialization of the space was bound to occur. A big stepping stone in the wide adoption of HTTPS, was the question of obtaining a valid certificate for my website. For small and personal endeavours obtaining a valid certificate might be too expensive or too difficult to obtain. In addition, further technical barriers were present as users had to have a certain knowledge of key generation and server configuration. This was a problem, that Josh Aas and co. aimed to tackle, with Let's Encrypt. [18]

The purpose of Let's Encrypt [18] was (and still is) to further the efforts of mass HTTPS adoption, and provide an easy automated solution. Critically, the issuing of valid certificates is completely free and comes with ready-to-use tools to speed up the issuing and configuration process, such as *Certbot*. Lastly, the certificate is valid for 90 days.

However, now that valid and widely accepted certificates may be issued free of charge, we must consider the associated risk. Threat actors clearly have all to gain in this arrangement, as a trusted certificate is essentially an institutional stamp of approval to the regular user. To briefly dabble in the economics aspect; the issuing of free certificates lowers the startup operating costs of creating a phishing campaign, thus increasing the profit margin of the malicious actor. In an ideal situation, the actor has more than one service to choose from. Here we list and briefly describe other free providers of certificates.

ZeroSSL [19] is a similar provider to Let's Encrypt. Providing free SSL certificates, valid for 90 days, again via an automated procedure. The caveat being, that if the client wants a certificate that is valid for longer, there only other option is the paid 1-year certificate. Functionally, ZeroSSL is the same as Let's Encrypt.

Buypass [20] offers free certificate creation via an API and integration with the previously mentioned *Certbot* automation tool. The issued certificates are valid for 180 days, and are automatically renewed. The service also shares a feature with Let's Encrypt [18] and ZeroSSL [19], in allowing multiple distinct domains in a single issued certificate.

Ssl.com [21] offer a more restrictive service, in the form of a 90-day free trial of their basic plan. This plan includes only one Fully Qualified Domain Name (FQDN), meaning that wildcard domains are not supported, with one www subdomain.

This section does not serve the purpose of an exhaustive list of free TLS certificate providers, instead it is meant to illustrate that many services are available. In the context of phishing attacks and malicious campaigns, these services prove a useful ally to the actor.

1.1.4 Certificate Transparency Logs

Certificate Transparency (CT) Logs are a system meant to enable the public review and auditing of issued server TLS certificates by a Certificate Authority (CA). First proposed in RFC-6962 [22] and then later modified with a proposal of version 2.0 in RFC-9162 [23]. The basic idea behind the logs does not change between versions, it is still meant to provide transparency in the issuing of certificates, so that the public may review issued certificates and search through previously issued ones. Additionally, in RFC-6962 [22] the authors also stated their hopes that in the future certificates that are not included in transparency logs would not be accepted, effectively forcing Certificate Authorities to publish all issued certificates in the logs.

Importantly, CT Logs function as an append-only public log of issued certificates. [22] [23] This idea does not change across the versions. It is also allowed for any third party to submit data to the log, with certain requirements to prevent submission of garbage data. Here the two versions begin to differ. The original RFC [22] requires of each chain to be rooted in a valid CA Certificate, while the updated version [23] states that each chain is must end with a *trust anchor* that is trusted by the log. In addition, the log may also specify the maximum chain length it is willing to accept, this feature is mentioned in the second version of Certificate Logs.

At the centre of the log structure is the usage of a Merkle Tree, which is a binary tree structure with each non-leaf entry being a hash made-up of its children. The usage of a Merkle Tree is again true for both proposed versions of CT Logs. [22] [23] An in-depth discussion into the ins and outs of the implemented tree data structure is out of the scope of this thesis.

The feature that is of most import to us, is the defined web Application Programming Interface (API). Using this API we are able to receive information about the signed tree head, which includes information about the total size of the tree. We may also retrieve certificates at specific indexes in the tree using the `get-entries` call, where we specify a decimal start and end index. Additionally, using the API it is possible for clients to append new logs to the chain, and retrieve a list of all trust anchors or all entries by hash instead of indexes.

1.2 HTTP Protocol

The Hyper Text Transfer Protocol (HTTP), defined in RFC-2616 [24], is a cleartext application layer protocol, most commonly used in the communication between web server and client to serve the contents of web pages. In plaintext form, it is clear that this protocol is of great import, when discussing the detection of phishing websites. Ethical questions of cleartext traffic monitoring and subsequent analysis aside, the application is probably apparent to everyone. This analysis and monitoring is not feasible in the modern world, based on the information from previous sections. While we discussed the usage of encryption, to mask the exchanged data, there is still some information that may be gleaned from the sent requests of an HTTP exchange, if we consider live interaction with a site on our end. This option is discussed later, in Chapter 4.

Sticking only to the sections directly relevant, to the idea proposed in the paragraph above, the standard [24] defines HTTP headers that may be used by the server and the client. When the client is sending HTTP headers it is usually to send additional information, or identify itself in some capacity. These identification methods may be used by phishing actors, to effectively filter out received requests and data. An example of this may be found in the *Referer* HTTP header, which sends information about the site we are coming from. If the sent *Referer* value is not from a whitelisted set of domains, like say a URL shortener, the user is not served phishing content. Moreover, will most probably be stored on the back-end effectively allowing the threat actor to hide from researchers, or at the very least make things more difficult.

Another potentially relevant header is the *Origin* header, which was defined in the proposed standard for Web Origins [25]. The concept of Web Origins in short aids with security controls over a website, allowing the server-side to dictate which Origins, e.g. domains, are allowed to interact with the site. This header may (according to RFC language) be included when an automated request is sent, to identify the sender site. The usage may again resemble that of the *Referer* header, where phishing sites identify the *Origin* based on whitelists. It is however important to note, that phishing sites and services are growing ever more complex, and central APIs may be utilized for the collection of captured data. Here the need for *Origin* whitelisting is clear, as the actors would not want to allow any site to interact with the phishing APIs. Examples of this behaviour may be observed in the Practical Dive into Phishing in Chapter 3.

Modern HTTP communications make use of standardized headers, such as the *Origin* or *Referer* headers, but numerous applications make use of custom and sometimes undocumented headers. All of these may be used in efforts to avoid detection, or increase the discovery difficulty for security researchers.

1.3 Flow Based Monitoring

Flow based monitoring [26] is a method of network telemetry collection, that does not rely on the inspection of IP packets. Systems built upon Flow based Monitoring inspect (what could be called) the *metadata* of a connection, instead of dissecting each individual IP packet. While the decision to perform traffic data collection and inspection at a higher level comes at the cost of certain data-loss, this is compensated by the performance advantages over packet inspection. [27] The benefits mentioned before lead to the fact, that the monitoring of flow can be deployed on high-speed networks, such as those discussed in this thesis.

A more formal definition of flow data can be found in RFC-7011 [28], which reads as follows:

"A Flow is defined as a set of packets or frames passing an Observation Point in the network during a certain time interval. All packets belonging to a particular Flow have a set of common properties."

These common properties can be many things [28], such as the IP source and destination

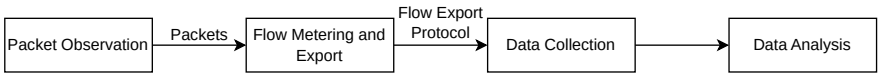


Figure 1.1 Diagram of a typical flow monitoring setup [31]

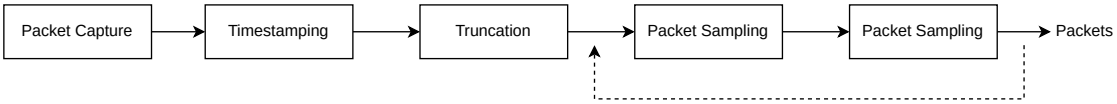


Figure 1.2 Diagram of the packet observation phase in Flow Monitoring [31]

ports, source and destination IP addresses or even application layer header fields. For the purposes of this thesis, these headers will include information from TLS handshakes, which come from an extension of the classic flow monitoring approach.

Collected data is then exported in a standard format, usually IPFIX [28] or NetFlow [29], for processing purposes in other systems. The IPFIX specification [28] describes the structure of each IPFIX message, which consists of a Message Header and Body. Contents of the Body are then made up of either Template Records, which define what data fields should be recorded and their interpretation, or Data Records that hold data as defined by Template Records. An extension of the Template Record field are Option Template Records, which add information about how to "scope the applicability of the Data Record" [28]. An IPFIX message need not necessarily be made up of all three mentioned record types, templates may be defined in bulk in a separate message after which the IPFIX body will only contain Data Records.

On the other hand, the previously mentioned NetFlow [29] protocol may also be split into a header and flow message sections. The flow message sections of a datagram may be sent in batches inside a single datagram, after the header. Let us focus on more relevant versions of NetFlow, in versions 5 and 9. Version 5 of NetFlow [29] includes general metadata in the Header section of the datagram, these include the version of NetFlow used, number of exported flows, counter of total flows seen and more. Continuing on to the body, it includes fields like source IP address, source TCP/UDP port, padding size and more. One of the main differences then lies in the fact, that NetFlow does not define template messages, which allow to set and control future interpretation of data. Version 9 [30] introduced templates in an effort to *future proof* the protocol, and avoid making too many changes to the base protocol in order to adapt to new challenges. Many of the obstacles of a new protocol may now bypassed by defining a new template, specific to the protocol.

Both the IPFIX and NetFlow export formats are commonly used in flow-based monitoring systems and solutions. The possible structure and typical setup of such a system is discussed by Rick Hofstede et al. [31], which they split into 4 parts, all of which are shown in Figure 1.1. These 4 parts need not always be the same, and may in-fact be merged together by certain solutions and implementations. Returning to the stages of flow-based monitoring, the first stage consists of the collection of data. This initial setup occurs on an *Observation Point* (OP) [31] and it is crucial to the whole system, as this where the analysed data comes from. In the *Observation Phase* the *OP* performs two operations under any circumstance, the capturing of packets and the addition of timestamps to the data. Timestamps are important for later analysis, when attempting to find correlations in data, or grouping flows together based on events, time windows etc. The *OP* may perform further operations, such as filtering and sampling of packets based on pre-defined rules with the aim of selecting only relevant or interesting packets from a stream, to reduce the amount of analysed data. Another optional step is *truncation*, which selects bytes from packets based on predefined lengths, this is known as a *snapshot length*. The result of this phase is in the form of packet data. A diagram of the whole first phase can be seen in Figure 1.2.

Following the *Observation Phase*, we may transition to the second phase that Rick Hofstede et

al. [31] called the *Flow Metering and Exporting* phase. The input into this phase are the results of the *OP* phase in the form of packets. These packets are then grouped and aggregated into *flows* which are then exported in specific formats as *flow records*. Packet aggregation that results in the creation of flows which is done by the *Metering Process* where packets are aggregated based on layout definitions of the flow, known as Information Elements. When the grouping is completed, flows are stored in a *flow cache* until their termination or expiration. Additionally, more sampling or filtering functions may be applied to the collected data. After the data has passed through all functions of the *Metering Process*, it is time for the *Exporting Process* to encapsulate all captured information in a standardized format. Commonly, the previously mentioned export formats of NetFlow (v5 or v9) or IPFIX are used. The *Flow Metering and Exporting* phase results in data exported in a format in accordance with a Flow Export Protocol specification.

It is important to note that the described Flow Monitoring system is a theoretical model based on common practices, and practical implementations may differ in certain areas. The description serves only as a high-level overview of how Flow Monitoring may function. To continue with the system described by R. Hofstede et al. [31], the next phase is labelled as the *Data Collection Phase*. The collection of data is done by *Flow Collectors* that can include more than one *Flow Collecting Process*, both these components put together work as consumers to the incoming data from flow exporters. Flow data that is then exported and collected by the collectors is pre-processed, if we assume that analysis is performed in a separate phase. Part of the mentioned pre-processing may be the anonymization of data. While flow data itself does not include sent data, and therefore already provides a certain level of anonymity, it is still possible to identify users and track their activity. To combat this further separation of the flow records from the individuals may be done, such as the anonymization of IP addresses. The usage of anonymizing techniques should however be considered on a case-by-case basis so that data vital to the goals of the analysis are not lost.

The final stage of *Data Analysis* may at times not be separated from the *Data Collection* phase, and instead they will work in tandem to provide on-the-fly analysis of data. We will consider the scenario where analysis is conducted in its own phase, and furthermore, we will consider two uses for the analysis. The first use-case is in the monitoring and reporting of network traffic, while the second is threat detection in the network, as stated by R. Hofstede et al. [31]. Let us first consider the usage of flow monitoring for the purposes of monitoring and reporting network traffic. Monitoring of traffic statistics should come as an obvious use-case, due to the fact that flow export devices are often placed at carefully considered spots in the network, and will therefore have access to large volumes of traffic data. By using the larger amounts of data to produce statistics, network engineers are able to detect which endpoints send the highest amounts of data and may either modify the network accordingly, create fallback plans or take other action. Other statistics may also include the time of peak network usage, outage tracking and more. Apart from the collection of statistics, the endpoints may also be used to monitor said numbers and report on anomalies or when a boundary is exceeded, like with bandwidth.

Another use for the *Data Analysis* stage is in its Threat Detection potential [31], where the analysis would be on a forensic level, ie. looking at historical stored data and determining which endpoints communicated or in live detection, where the endpoints serve as an IDS (Intrusion Detection System). Historical dissection of flows should be a self-explanatory and on occasion specific use-case, and so we will focus on the IDS nature of analysis. One potential solution involves the usage of reputation lists, where the destination IP addresses of connection are monitored and checked against blocklists to identify destinations with poor reputation. The identification of destination addresses rather than source addresses can be used to identify attackers inside the local network, as malware may often make connections to central malicious servers. Filtering of inbound IP addresses, ie. source addresses may be used to prevent known spam sources or other unwanted connections. Apart from the utilization of blocklists, flow based monitoring may also be used to identify incoming *bruteforce* attacks, attempting to guess remote connection passwords by repeated connections in quick succession, as well as the identification

of port scanning by monitoring for a wide array of incomplete TCP connections.

1.3.1 Nemea

In Chapter 1.3, we discussed the general potential setup of flow based monitoring systems, which demonstrated the complexity of even a simple setup. This chapter will focus on a specific flow monitoring system called NEMEA [27], upon which the thesis implementation is built. The Network Measurements Analysis (NEMEA) system is an online stream-wise traffic analysis tool, working on the Application Layer, capable of anomaly detection, statistics collection, data filtering and more. The capabilities of specific NEMEA instances is dependent on the deployed modules, as the system was designed to be extendable by developers with the intent of giving them the ability to create extensions specific to their needs. By enabling developers to create modules for the system, T. Čejka et al. [27] have given NEMEA the potential for many applications such as attack detection, identification of specific malware families in the network and potentially endless applications.

The inclusion of modules is a crucial feature in the system, and so we will focus on them a little more. In the paper modules are described as follows:

"Modules are independent processes interconnected by unidirectional interfaces for communication. The interfaces transfer data in the form of streams of messages — flow records, results of some analysis etc." [27]

A simple way to understand the thought behind modules is in a distributed system that communicates via network messages. Each module is a separate running program in the distributed system and each waits for input from another. In general, the structure of these distributed programs would be connected into a tree or a directed acyclic graph, where both of these topologies include a single entry node that distributes the initial data to following modules. Typically there will also be one final module, that is used to send alerts into either output files, databases, email or other form of messages. Another advantage of modular design is that it naturally lends itself to academic research purposes instead of being limited to purely practical deployments. The availability of the framework in research stems from the fact that libraries are available in three languages: C, C++ and Python [1]. Adding support for the Python language greatly increases NEMEA's validity as a prototyping framework in addition to production deployments, thanks to faster and more streamlined development.

Data transferred in the NEMEA infrastructure is stored in a specific binary format, called UniRec [27]. The used binary format of UniRec allows for the fast transfer and storage of data, this is done by allowing direct access to the information without parsing, by using a structure similar to C structs. The implemented format however, is not a direct copy of structs and contains many differences. Unlike structures in the C language, UniRec allows the usage of variable length fields along with template definition at run-time. Templates are a set of fields that will be contained in a record, they also define the specific format of a UniRec entry as they are generic structures of data before a format is used. To add to the previous point of generality, definitions are also needed, because that is how endpoints define the data format that they are capable of processing and what data format they export.

One of the crucial points of the NEMEA system is the Traffic Analysis Platform (TRAP) [27] as it provides the necessary communication interfaces and basic functions for every module. TRAP Communication Interfaces (IFC) are interfaces that allow modules to communicate with each other in a unidirectional format where each module may have more than one output or input interface. IFCs may also be connected to multiple output or input interfaces of other modules, allowing for robust infrastructure creation. Under the hood, the interfaces are a combination of UNIX sockets and TCP sockets, where UNIX socket communication is used for modules running on the same machine and TCP is used for network communication on foreign stations. Output of

[1] https://github.com/CESNET/NEMEA

IFCs may also be one of two special types, either as a *file* or *blackhole*, in the latter case all outputs are essentially discarded. Using the TRAP library also provides another important feature in its abstraction of network communication and handling of said communication into simple functions. Programmers that are not skilled in network programming may rely on the provided methods to receive and send data, to handle communication errors and generally offload the heavy-lifting of the networking process to the TRAP library. Additionally, TRAP has bindings in both the C and the Python language, thanks to which it is possible to write NEMEA modules in both languages.

1.4 Phishing

While the information provided in the Introduction serves the purpose of showing *why* phishing is an issue, we believe that some time should be dedicated to discussing the issue itself. Meaning, that a non-exhaustive breakdown of *phishing* is required, and it is provided in this section. All the information mentioned in the Introduction might not seem overly remarkable to the average internet consumer, but the en-mass adoption of the Internet has had a knock-on effect with the emergence of threats and threat actors in cyberspace. First, let us take a moment to look back at the origins of *phishing*, which can be found in the mid-1990s on the *America Online (AOL)* forum. This forum became the playground of Koceilah Rekouche [32] and the group around him, who would use social engineering methods to gain access credentials and personal information of other users. These techniques are still employed to this day, as Rekouche stated:

"Similar to today's phishing schemes, it involved tricking someone into trusting you with their personal information. In this case, a person who had just logged on to cyberspace for the first time would be fooled into giving up their password or credit card information" [32]

According to Rekouche [32], he and his team would attempt to convince users that they were some sort of official account and needed to confirm the credentials of the victims. This could be done by creating accounts with convincing names, like "BillingDept". Even after AOL attempted to combat the groups impersonation activities by blocking certain words in usernames, Rekouche et al. managed to bypass these restrictions by using similar looking letters and numbers. An example could be substituting the 'O' from AOL with a zero, resulting in A0L. The term *phishing* was later used in an automated tool created by Rekouche et al. [32]

In today's terms, the techniques and delivery method of Koceilah Rekouche might be considered *social media phishing* or *instant messaging phishing*. [33] Some other types of common phishing types include *email phishing*, *website spoofing*, *voice phishing* (vishing) and *sms phishing* (smishing). [33] [34] Putting together the information provided at the start of this section, with the various types of phishing discussed now, it should be clear that the importance of detection and prevention techniques is ever rising. In-fact, according to the Czech National Cyber and Information Security Agency's (NÚKIB) 2021 annual report [35], 90% of respondent organisations stated that they either fell victim to a phishing attack or an attack was at-least attempted.

The techniques used by Rekouche et al. have since been formalised and written down, with the specifics of each method summarized by Ezer O. Yeboah-Boateng and Priscilla M. Amanor [34] in their paper where they outline phishing threats to mobile devices. The work of Yeboah-Boateng et al. may be focused on threats to mobile devices, but the descriptions of the attack methods remain the same across devices and are therefore relevant to us. The general term of *phishing* may refer to the following, a malicious party creates a website with the sole purpose of impersonating a trustworthy website. Creation of the site may be done via automatic tools or manually, with the main goal being the creation of the most believable website possible. The delivery mechanism of these pages will typically be via email, which utilizes social engineering tactics of putting pressure on the user, pretending to be an authority within their organization or

faking an emergency. Parts of the mentioned *phishing* description are applicable to other more specific variants, which usually differ in the delivery mechanism.

Smishing [34], is the act of phishing via Short Messaging Services or text messages. Victims may often fall for two types of *smishing* attacks, the first being an impersonation attack. In the case of impersonation, the received text messages is assumed to be from a trusted source, like a banker, executive or direct superior. A possible pre-text for the received message from a unknown number may be a quick change of the impersonated person's phone number or the need to utilize their personal mobile phone. Impersonation attacks may request the victim to perform wire transfers, send documents or perform other potentially damaging acts. The second type of attack may be in the form of an emergency message, which informs the victim of a blocked bank account, frozen assets or other displeasing events. Typically the message provides a quick solution via a shared URL, which leads the victim directly to a phishing page. Another possibility is that a malicious attachment will be shared via SMS. The sharing of dangerous attachments is a staple of *email phishing*, so in this case it is purely the delivery mechanism that changes.

Vishing [34] acts as a short-hand for *Voice Phishing*, which involves directly contacting victims via phone calls. Attackers typically scour through leaks of client phone numbers, which they followup with calls motivated by financial gain. One of the main reasons why *Vishing* is a valid method of attack rests with the confidence and communication skills of the malicious actor, as highly charismatic and convincing callers can convince victims that they are who they claim to be. Additional technical methods may be implemented, such as caller ID spoofing, which greatly increase the believability of the attacker's call. When performing *vishing* the attackers may be able to convince victims to (unknowingly) be part of a money-laundering scheme, give out payment details or directly transfer finances.

Aleroud et al. [33] define two additional phishing methods; *Social Media Phishing* and *Instant Messaging Phishing*. The two proposed types are closely tied together, as modern social media often serves as an instant messaging platform. In phishing attacks utilizing social media, it is the delivery mechanism that changes the most. With the change of delivery mechanisms come certain roadblocks and challenges. Attackers may need to rely on account takeovers, in order to perform phishing of users efficiently or attempt to impersonate other users close to the victim, by acting much like attackers do with *smishing*.

1.5 Current Phishing Detection Systems

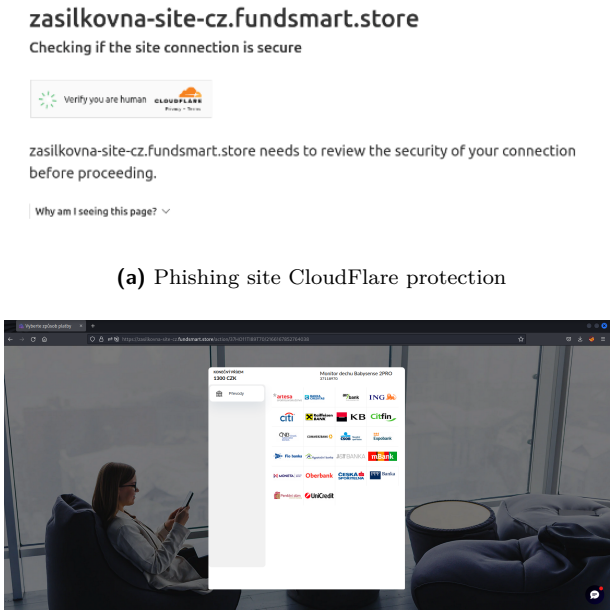
In this chapter, we discuss the state of the art methods for phishing detection, as used and proposed by other researchers and writers. The goal is not to only showcase methods close to those presented later in this thesis, but to also give a general overview of the phishing detection ecosystem. Since phishing may take on varied forms, as described in Chapter 1.4, so may the detection engines. Some implementations may focus on identifying phishing websites based on the served HTML content or visual similarity to the legitimate website, others inspect sent emails and determine if they contain certain indicators. Generally, antivirus solutions also provide some form of phishing website blocking or detection, these solutions however require presence of software on the client endpoint and may (in certain cases) impede the user's privacy.

A deep learning method of detection as presented by Yun Lin et al. [36] at the 30th USENIX Security Symposium, shows how hybrid deep learning may be used to identify phishing websites based on visual similarity identification. The proposed system may be adapted on-the-fly to detect phishing targeted at other brands, with Lin et al. [36] reporting high levels of true positive detections. One of the staples of the created detection system lies in its recognition of company logos, based on user-defined lists. The system uses the reference list to compare logos in the checked website, and sees if the images reach a certain threshold of similarity. Apart from image detection, the system is also able to identify input boxes on the site, working with the assumption, that a traditional phishing website asks for user credentials and uses the targeted company's logo to increase the site's trustworthiness. To go a step further, image detection is combined with a

Figure 1.3 Example of Phishpedia's resulting annotations as shown in the paper by Lin et al. [36]

certain comprehension of the domain name, which allows Phishpedia to put together multiple visual indicators and determine if a website is malicious, much like a well-trained human. A staple feature of Phishpedia lies in the addition of explaining how it identified a phishing site, adding annotations to the site's screenshot to outline the input boxes, similar logos and other indicators. An example of this is provided in Figure 1.3.

Lin et al. [36] note, that during the testing of Phishpedia's effectiveness, they utilised multiple VPN connections, to prevent phishing sites from profiling a single source IP address as problematic. Further more, they provide evidence, that phishing sites with logos of the targeted brands are more common than sites without related images and further more, that phishing websites will often include login forms. To further test their system, Lin et al. provide their methods of testing the model against adversarial attacks. It is important to note, that the paper written by Lin et al. focuses on phishing on a global scale, while this thesis focuses on localized phishing, targeting brands in the Czech Republic. The paper [36] states that Phishpedia may be adapted to detect phishing against local brands, by adding them to the list of targeted brands, which we argue may not always be the case. Our following observations are based on our practical experience with Czech phishing trends, from which we may conclude that it is common for phishing websites to include fake loading pages without logos or to require rudimentary captcha completion. Attackers may implement evasion methods, that do not target the deep learning model itself, instead they add random delays that require either active interaction from the crawling agent collecting screenshots for analysis, or prevent the screenshotting service from capturing a clear image entirely. It should be possible to adapt the service for localized needs, but the fact that the system relies on screenshots of phishing content cannot be removed. This should present attackers with a clear incentive to prevent the automated collection of screenshots. Examples of this behaviour may be observed upon browsing scanned websites on the public website screenshot service *urlscan.io*, where the sites will show fake 404 error pages in the urlscan results, but upon manual inspection of the live site they provide phishing content. Another hurdle for the vital screenshot collection process may be, that phishing sites often opt to use CloudFlare services, that block known problematic VPN connections, TOR exit nodes and more. An example of this



(a) Phishing site CloudFlare protection

(b) Phishing site actual content

Figure 1.4 Example of phishing sites using CloudFlare protection

behaviour is provided in Figure 1.4.

A much more simple method of phishing detection is to utilize lists of known malicious domains and URLs, known as blocklists. Adam Oest et al. [37] explore the effectiveness of using blocklists in their paper, in which they point out that while blocklists are widely adopted, they are also inherently a *reactive* measure. The reactive nature of creating lists of malicious domains and adding to them should not come as much of a surprise, as the phishing link must first be discovered, then reported and finally added to the list. Additionally, adding domains to lists should not be done without verification of their malicious nature, leading possibly to further delays. It is this window of time, between the creation of phishing and its addition to a blocklist, that the malicious actors make use of as it gives them time to exploit unknowing users and potentially make a return on their investment. To even further the effect of the mentioned slowed down reaction time, attackers may utilize evasion methods that make it harder to detect phishing sites using automated crawlers. These methods were briefly mentioned by us, when discussing possible bypasses of the screenshot system for the deep learning detection system, presented by Lin et al. [36].

Phishing creators may evade automated crawler checks using several methods, three of which are discussed by Oest et al. [37]. The three described methods are *cloaking*, *redirection* and *infrastructure compromise*. *Cloaking* is a method that we theorised about when discussing the possibilities for evasion of the system of Lin et al. [36]. It is a method that aims to hide the phishing content from automated blocklist infrastructure like crawlers and other automated services. Part of the described evasion methods [37] is the usage of blocklists, which are usually distributed with phishing kits and are used to identify known crawlers, automated checks or known security companies. When a source of a connection matches an entry in the attacker's blocklist, they are shown a fake page that may include fake error content or a benign page. Identifying a connection that is considered undesirable by the attacker may be based on several indicators including: Identification in HTTP headers, IP address-based geolocation and device characteristics.

Redirection methods aim to add layers of obscurity between the link a phished victim receives

and the final phishing page they end-up on [37]. With blocklists being often deployed as parts of email filters, preventing SPAM and discarding unwanted and potentially dangerous email messages, it is possible to bypass these blocklists using redirection links. Keeping a blocklist of redirection links is an almost futile effort, as phishers may use link-shortening services, infected websites or specifically crafted malicious websites. It is poignant to note, that solutions detecting phishing pages directly on the client's station would not be affected by the *redirection* method, as the final phishing page should still be detected. The final blocklist evasion method mentioned by Oest et al. [37] is described as *infrastructure compromise*, which we discuss in a bit more detail in Chapter 3. In essence, phishers compromise legitimate websites which they use to serve phishing content in specific directories, while the rest of the site remains as before. This method poses a significant issue for blocklist creators, as it may be in their interest not to blocklist the compromised site as a whole, but only the phishing URL. However, this care from the blocklist creators could lead to incremental bypasses of the list by continuing to change the URL path where the phishing is hosted without changing the domain itself.

In their paper on blocklists Oest et al. [37] also discuss how effective blocklisting methods are. For the purposes of evaluation they define the following two metrics; *Discovery* and *Detection*. *Discovery* refers to the blocklist's ability to identify new phishing URLs, this can be done by adding to the list based on live user reports, collection of phishing warnings from other ecosystems and more. A blocklist system with perfect *discovery* would know of all currently active phishing links. *Detection* is the system's ability to identify true-positive cases in a timely manner correctly. This metric can be further split into two sub-metrics of *speed* and *coverage*. It is desirable for blocklisting systems to minimize the time between discovering a new phishing link and adding it to their blocklist, this time between *discovery* and the subsequent blocklisting is the *speed* sub-metric. *Coverage* would then be the proportion of currently active phishing URLs, that are present in the list. Oest et al. [37] conducted experiments that tested the mentioned metrics of generally adopted and used blocklists on both mobile and desktop clients, including Google Safe Browsing [2] and Smart Screen [3]. In their experiments, Oest et al. [37] first established a baseline of metrics, with a set of phishing sites that did not use evasion techniques, these yielded results in the favour of blocklists. What the research did show however, was that the metrics of blocklists on mobile devices performed significantly worse than those of the desktop versions. Another point to make is that more advanced evasion techniques (CAPTCHA enforcement, mouse movement detection and others) significantly hindered the performance of the monitored services.

To emphasize, that a large portion of phishing detection research is dedicated to identifying phishing emails instead of phishing websites or domains, we provide another research paper that presents a method of identifying large-scale email phishing. Grant Ho et al. [38] came forth with a model for detecting *lateral phishing* at scale in enterprise settings. First, we should shed light on the term *lateral phishing*, as until now, we have only discussed phishing attacks that serve as the first stage of an attack. *Lateral phishing* serves as a tool for *lateral movement* on the network, which are techniques that attackers use to move along the network after initial compromise, usually attempting to compromise additional accounts, harvest credentials, data and more. [39] The kind of phishing that Ho et al. [38] discuss then often comes from compromised accounts inside a company network, which will often attempt to send malicious attachments or further specifically crafter phishing pages to other internal users. An example of such a possible phishing email is provided by Ho et al. [38] which we show in Listings 1.5. Internal phishing is much harder for employees to identify, because it comes from an inherently trustworthy source from within; this eliminates much of the typical advisories given by internal training programs. To combat this issue with identifying internally spread phishing emails, Ho et al. [38] used a large dataset of 113 million employee email messages that had been collected from 92 separate enterprise organizations, based on this large sample of data they were able to train a classifier

[2] https://safebrowsing.google.com/

[3] https://learn.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-smartscreen/microsoft-defender-smartscreen-overview

Code listing 1.1 Example of an anonymized lateral phishing email, as provided by Ho et al. [38]

```
From: "Alice" <alice@company.com>
To: "Bob" <bob@company.com>
Subject: Company X (New Contract)

New Contract

View Document [this text linked to a phishing website]

Regards,
Alice [signature]
```

that identifies phishing emails with 4 false-positives in a million messages.

When implementing a classifier of phishing emails, Ho et al. [38] decided to extract a set of most common indicators of a lateral phishing email of which they count three. The first is a high number of distinct recipients in a single email, to be precise, more than 90% of user-reported lateral phishing incidents in their data were sent to 25 or more distinct recipients. Building on this, the classifier extracts all destination headers of the email ie. the *TO*, *CC* and *BCC* headers. However, considering that some events may truly require 25 or more recipients, the system also calculates a similarity score between the current set of destination emails and all destination groups from the last month.

The other two indicators are the *lure* and the *exploit*, which come from a framework for spear phishing detection, as proposed by Ho et al. in 2017 [40]. The *lure* in this case is a keyword that the system looks for, if the analyzed email contains one of the keywords classified as malicious [4] a boolean value is set to *True*. Following the *lure-exploit* framework [40], the sent email should then contain some kind of malicious content ie. the *exploit*. In the case of lateral phishing emails as analysed by Ho et al. [38] they limit themselves to malicious URLs.

To decide if a URL is malicious, the system essentially states if it trusts the received link. Some URLs are implicitly trusted by the system, such as URLs placed on the company's trusted list or hyperlinks that are an exact match of their destination. After deciding if the link is inherently trusted, the system looks up the score of the domain in Cisco's top 1 million domain list, and assigns a score of 10 million if none is found. Lastly, a URL's rareness in the company is measured, if it is commonly seen on the network then it is given a high reputation score. Putting all the mentioned features together a classifier had been trained which, when paired with an extractor of said indicators, is able to decide whether an email is an *internal lateral phishing* attempt or not. The resulting classifier created by Ho et al. [38] has promising results, detecting both known and unknown phishing attempts in the data, along with keeping a very low rate of false positives.

To continue with the theme of adversarial tactics, we look at how used detection methods fair against adversarial phishing as summarized by Thomas Kobber Panum et al. [41] In their paper Panum et al. describe three methods of detection as observed in multiple research papers, when considering phishing emails. The first method is one that has been described previously in this thesis, which is based on visual similarity. In their description, Panum et al. reasonably assume that making the phishing message visually close to the impersonated site is desirable and will increase the likelihood of a successful phishing attack. Often, when a detection system is based on visuals, the system will attempt to imitate the behaviour of a human when looking at a phishing attempt or at the very least, the systems will attempt to guess how a human would interpret the received message. If the system is able to differentiate visually close and original messages, then we may use this to identify if a received message is similar to a set of benign messages and comes from a different origin. When all requirements of visual similarity and

[4] A set of malicious keywords was obtained from Baracuda networks for whom three of the authors work.

differing sources come together, the system may either use this as a single indicator of phishing, or some decide purely on these factors if a message is malicious. Panum et al. [41] also present multiple detection methods that rely on visual similarity, notably those that rely on some kind of distance calculation, because they later point out that these metrics may not be accurate when put up against adversarial phishers. The problem with pixel distance metrics rests in the fact, that human perception of marginal colour changes is not as precise, so almost imperceptible changes to the naked eye may greatly increase the values of the calculated distance metrics.

Following the visual similarity metric, Panum et al. [41] present what they call *Reverse Search Credibility*, which works with the assumption, that a credible website will appear in the relevant search results when querying search engines. The simple ways phishers could combat this simple metric, would be to utilize the *robots exclusion protocol* [42] specifying which parts of the website they allow crawlers to index in search engine results, and add relevant keywords to those allowed sections. This both cloaks the phishing content and gives the phishing site reverse search credibility. Another metric defined in the research by Panum et al. [41] is called *Channel Meta Information* which essentially amounts to metadata relevant to the given medium of transfer. For example with websites, the relevant meta information would be the used URL. To analyze URLs, lexical analysis may be utilized to test if a URL is attempting to impersonate a known benign URL. All-in-all Panum et al. [41] did not argue much in the favour of adversaries being able to combat this method. Here we provide our arguments, based on previously cited work in this section. One of the great sources for confusion of the meta information metric, when speaking about URLs, may be the usage of URL shortening services or the compromise of legitimate websites. If the phishers manage to compromise a benign site, then the lexical analysis should provide a close to 100% match and probably conclude, that the tested site is indeed benign. On the other hand, when we consider URL shortening services, the lexical analyzer will either not be able to analyze the shortener URL or will provide high amounts of false positives for benign, but shortened URLs. Even if the checker resolves the shortened URL in an attempt to find what it points to, the phishers may utilize automated crawler checks to prevent access to said phishing URLs. The last metric proposed by Panam et al. [41] relates to statistical modeling and by extension Machine Learning.

A method for the detection of phishing domains was proposed by Hossein Shirazi et al. [43] in their work for unbiased detection of phishing based on domain-based features. They utilized machine learning techniques in combination with a set of features that relate to the domain name of the phishing site. The decision to focus on the phishing site's domain name came from the presumption that the domain name is the main indicator of a phishing attack. With this in mind, they trained a machine learning classifier that they claim does not focus on search engine results, DNS records, third party services or URL-specific features. One of the first issues of the paper stem from the fact, that the classifier was trained on a limited set of trustworthy domains. The classifier was fed the top 1000 domains of the Alexa.com top ranked list [5], with the intention of training it to determine legitimate sites. This approach has one significant issue when considering network detection of phishing, in that domains on the network are primarily not from the Alexa.com list. In-fact a large portion of general network traffic comes from Content Delivery Networks (CDNs), analytics collectors and APIs which may often have traits that differ from the domains featured in the Alexa.com list. Our observations of these issues may be found in Chapter 2 and Chapter 4. The machine learning based classifier put forth by Shirazi et al. [43] was tested against a sample of top 1000 domains from the afor mentioned list of top websites and against 2013 known phishing sites collected from PhishTank.com and OpenPhish.com. Against the selected datasets, Shirazi et al. [43] present an accuracy of 97% - 99.7%, the paper did not however describe performance of the classifier on a live network over several hours or days.

[5] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

Chapter 2
Used Data

As previously mentioned, we had encountered papers that utilize lists of top domains with the aim of extracting common features of benign websites from them. We put forth the notion, that this is not an optimal solution for systems to be used in online phishing detection from live network traffic. In order to put some evidence behind our claims we provide a comparison of the most commonly occurring domains from a data set of captured network traffic on the CESNET network and some of the top domains from the top 1 million domains from Alexa.com [1]. The comparison of the top 10 most commonly occurring domains can be seen side-by-side in Listings 2. From the comparison we may observe that the Alexa.com list does not include a fundamental factor of phishing in the form of subdomains. The Second Level Domains (SLDs) may be close to the Alexa.com list, but there is another side to consider, when deciding which dataset to use. When picking a sample list of benign URLs or domains it is important to consider the use case and specificity of the detection system, as a heavily localised system will not benefit from using a list of global domains. In our case we are focusing on the Czech phishing ecosystem, so it makes more sense to extract features of benign domains from local traffic and data, instead of looking at a global dataset like the Alexa.com list.

Code listing 2.1 Top 10 domains on CESNET network over 24 hours

```
graph.facebook.com
outlook.office365.com
web.facebook.com
v10.events.data.microsoft.com
login.microsoftonline.com
gateway.icloud.com
edge-mqtt.facebook.com
self.events.data.microsoft.com
settings-win.data.microsoft.com
www.bing.com
```

Code listing 2.2 Top 10 domains on the Alexa.com list

```
google.com
youtube.com
baidu.com
bilibili.com
facebook.com
qq.com
twitter.com
zhihu.com
wikipedia.org
amazon.com
```

To perform a more proper comparison of the data, we extracted only the second-level domains from data collected on the CESNET network. In total, we utilized two separate sets of data from the CESNET network, one has been mentioned before and it was collected over 24 hours, while the other was collected over a longer period of over 48 hours. Our initial observations of features and testing of initial proof-of-concept implementations were based on the first set of data. After completing our first runs, we then tested the effectiveness of the implementations on the second (larger) set of data. The logic behind our reasoning being, that extracting features from a smaller

[1] http://s3.amazonaws.com/alexa-static/top-1m.csv.zip

time-frame will be less resource intensive and faster to tweak, debug or otherwise analyse. If the extracted features and information is then tested on data from a larger time frame, we may observe if the possible indicators of benign domains are valid only in the monitored 24 hours, or if they will produce similar results over longer spans of time.

After explaining our thought process behind the expansion of dataset size, we may now proceed to the comparison of SLDs in both sets of data. This comparison is available in Listings 2, where we may see that the most frequent second level domains remain pretty much the same. The major differences in the top 10 SLDs across the two CESNET datasets is in their position and in one or two completely different domains. On the other hand, if we now compare the two aggregated results to the Alexa.com list in Listings 2 we can see the global influence on the data. To specifically point out the differences, one such major difference is in the complete lack of `.cz` Top Level Domains (TLDs), which is understandable on a global scale. This illustrates our previous point about having to consider where we receive our definitions of benign domains from, and with what purpose.

Code listing 2.3 Top 10 SLDs on CESNET network over 24 hours

```
microsoft.com
facebook.com
apple.com
seznam.cz
live.com
google.com
googleapis.com
icloud.com
msn.com
office365.com
```

Code listing 2.4 Top 10 SLDs on CESNET network over 48 hours

```
microsoft.com
facebook.com
seznam.cz
live.com
googleapis.com
google.com
msn.com
sdn.cz
microsoftonline.com
apple.com
```

If we now consider the issue of extracting features of phishing domains, we will find that this problem is a lot more complex. First is the obstacle of obtaining confirmed true-positive phishing domain samples. As we mentioned with data selection for benign domains, one would think that it is important to utilize sets of data that are as localized as possible. While this notion is true, the cost-to-reward ratio may not be as good as with the recognition of benign domains. Some features will remain the same for phishing domains thanks to the nature of the endeavor for phishing creators. First, they must consider the setup costs, operational costs and potential returns, much like a legitimate business. In the special case of phishing campaigns, the startup costs entail the registration of domains and subsequent hosting of content. Lowering the initial expenses can be done by selecting domain names with less desirable TLDs, longer names and many more tricks. In a roundabout way we have described why using a less-localized set of phishing samples might not be harmful for the correct extraction of common features. After all, the motivation (for broad non-spear-phishing campaigns) remains the same and so it should be reasonable to assume that the methods will remain consistent as well.

Another factor to consider when attempting to think about phishing on a local scale, is that the actual phishing creators need not come from the targeted area. In-fact, it is common for one phishing creator to target multiple countries in the same campaign. This could be to improve the previously mentioned cost-to-reward ratio. Phishers also commonly do not create their own phishing from scratch, instead they use prepared scripts and source code from so called Phishing Kits. This issue and a more practical dive into phishing in the Czech Republic is discussed in Chapter 3.

Phishing data was collected from a public GitHub repository which hosted user reported phishing sites and domains. The repository is known as Phishing Database [2], from which we collected over 300 thousand phishing domains for analysis. When we compare the total amount of

[2] https://github.com/mitchellkrogza/Phishing.Database

stored phishing domains to the total amount of domains captured over 24 hours on the CESNET network, we will see that the ratio is much in favour of the regular traffic. This is expected, and we believe should be reflected in the analyzed data. If there are significantly lower amounts of phishing domains on the network, than benign domains, having equally large datasets for both is not only challenging but also impractical.

Chapter 3

Practical Dive into Phishing

The goal of this chapter is to discuss the trends, tendencies and common practices in the phishing space. Information included in this chapter is based on our practical experience and numerous interactions with phishing campaigns, including authorized creation and most importantly active hunting and takedowns. Another crucial point to keep in mind is that our practical experience is based on trends of phishing in the Czech Republic, and limited foreign experience with foreign campaigns. While campaigns do tend to originate on foreign hosting and servers, the commonly used phishing kits, naming conventions and targeted services may differ from those that occur in the Czech Republic. In this chapter we will discuss real-world phishing site examples, delivery mechanisms and methods, targeted brands, used services and provide examples with the ease of phishing creation. Please note that unless a reference is provided, the observations should be considered anecdotal, based purely on our experience.

3.1 Brand Targeting in the Czech Republic

Targeted brands of large-scale phishing campaigns in the Czech Republic seem to consist mostly of parcel delivery services, and online second-hand retail services. The targeting of reselling online shops is most probably down to the fact, that the nature of the exchanges mandates customer interaction. This expected and somewhat needed interaction is advantageous to the malicious actors, for two reasons. Firstly, the actor need not scrape the web for emails, go through leaks of databases or generally conduct potentially labour-intensive work of collecting emails or phone numbers. Instead, victims are presented to him as sellers, looking to sell off used items and therefore open to communication. Secondly, the most basic incentive is also present, and that is financial gain as the seller is presumably aiming to make money. If the malicious actor is then able to convince the users to communicate via services that are out of the scope of the application, they are also avoiding potential link crawling checks, when sending the phishing link. The premise of the phishing link is usually as follows; The malicious actor is in a foreign country and is willing to pay for the delivery of said goods, all that the seller must do is provide their payment details on the provided link with the premise of the buyer being able to give money directly to their card.

Parcel delivery services are victims to more straight forward phishing attacks, as it is somewhat reasonable to expect that a sizeable amount of the targeted selection of users will be expecting a parcel. The sent phishing link then contains information about needing to pay some kind of service fee, customs fees and more. Attacks like these tend to target payment details of victims, that may then be used for money laundering, by buying larger amounts of small vouchers or selling the information on an online carding forum. Untrue to popular belief, carding

forums are not only present on the *dark web* accessible through The Onion Router (TOR) network. Carding forums are common place on the so called *surface web* and they may or may not actually provide the desired service of delivering a usable payment card details upon purchase. In this case a *phishing of phishers* scenario occurs.

Universally acknowledged targets are not uncommon in the Czech phishing landscape, with the banking sector finding itself often in the cross-hairs of the phishing creators. Internet Banking phishing is still a prevalent part of phishing campaigns, though from our experience, they do tend to arrive in waves. The amount of identified campaigns tends to change from month to month, which can indicate separation of campaigns into phases. A preparation phase and the execution and delivery phase. Banks may also be targeted directly, or indirectly as part of other phishing campaigns against parcel delivery services or government organizations.

A trend that appeared in the Czech phishing landscape is the targeting of the Ministry of Labour and Social Affairs, with its Czech abbreviation *mpsv* being used in numerous domains. Using the certificate search service from Censys [1], we identified a possible first occurrence of a *mpsv* phishing domain in August of 2022. This was due to initiatives on the side of the ministry, aimed at extending a hand of financial aid to lower income families and groups during the tail-end of the COVID-19 pandemic and with the *Deštník Proti Drahotě*[2] initiative, which may be loosely translated as the *Umbrella Against Poverty*. Often registered in bulk, and distributed via SMS messages the sites appear mostly the same, and aim to extract banking login credentials of individual victims.

3.1.1 Parcel Delivery Services

Commonly distributed via SMS messages, making the attack a smishing attack [34], betting on the fact that it is common for victims to be awaiting a package delivery. Brands targeted by these attacks include the Czech Post, DPD and DHL. Outside of the Czech landscape it is common for other national carrier services to be targeted.

To support our claims that the industry of Parcel Delivery is a high-volume industry, we provide statistics relevant to the Czech Republic's postal and package delivery services in the form of plots. The Figure 3.1 shows the growth of total domestic parcel deliveries in the Czech Republic. The plot presents a growing trend, from the start of measurement in 2012 all the way to the final date, at time of writing, in 2022. Therefore, it should be reasonable to assume, that deliveries coincide with sent messages leaving the clients desensitized to the amount of delivery messages they receive. To further support this argument, we provide another plot, Figure 3.2, showing the total amount of imported parcels into the Czech Republic from the start of measurement in 2012 to the year 2022. We may observe a sharp decline after the year 2015, that soared back up to the original high between 2021 and 2022. Considering that the totals are in the thousands, we may observe the totals in packages in millions. An industry like this is ripe to be targeted by the creators of phishing campaigns.

Often the campaigns are paired in tandem with the phishing described in Chapter 3.1.2, with the main difference being in the delivery method. From our experience, two types of domains are the most prevalent in Parcel Delivery attacks. The first option is attacking trustworthy, registered and already up and running sites, that include security vulnerabilities. Often the sites run on a Wordpress server, and presumably are either using an outdated version, default credentials or an insecure plugin. The attackers identify these sites, perform a takeover of the site and upload a phishing kit into an obscure directory. An example image of such a phishing site may be found in Figure 3.3. At time of writing, the URL for this active DHL phishing attack was:

"https://clocare.in/wp-admin/js/a/tracking/fV5EjH/details.php"

[1] https://search.censys.io/certificates
[2] https://www.destnikprotidrahote.cz

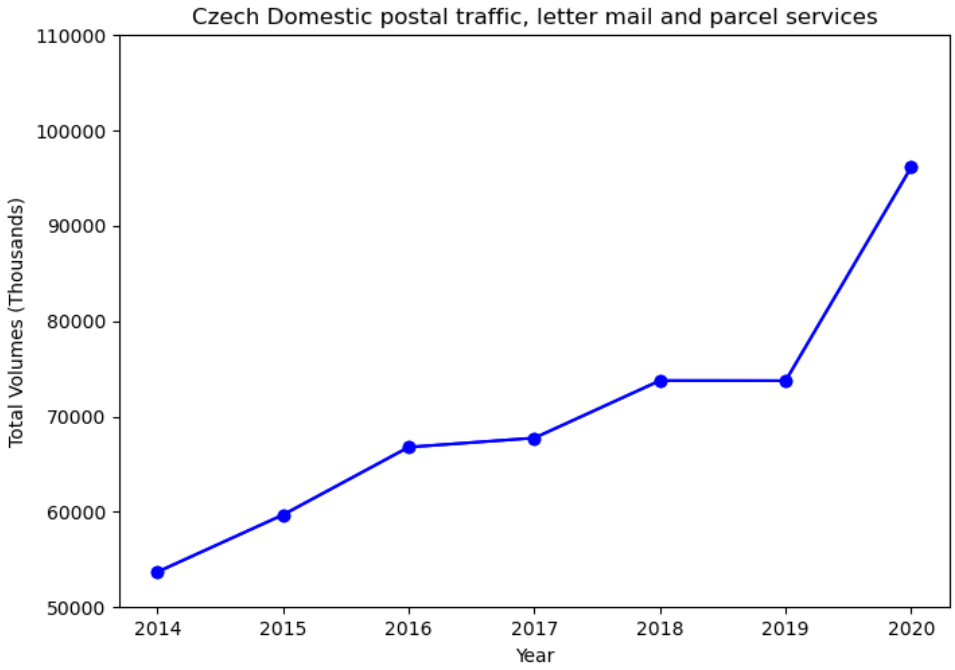


Figure 3.1 Statistics of total domestic parcel deliveries in thousands in the Czech Republic [44]

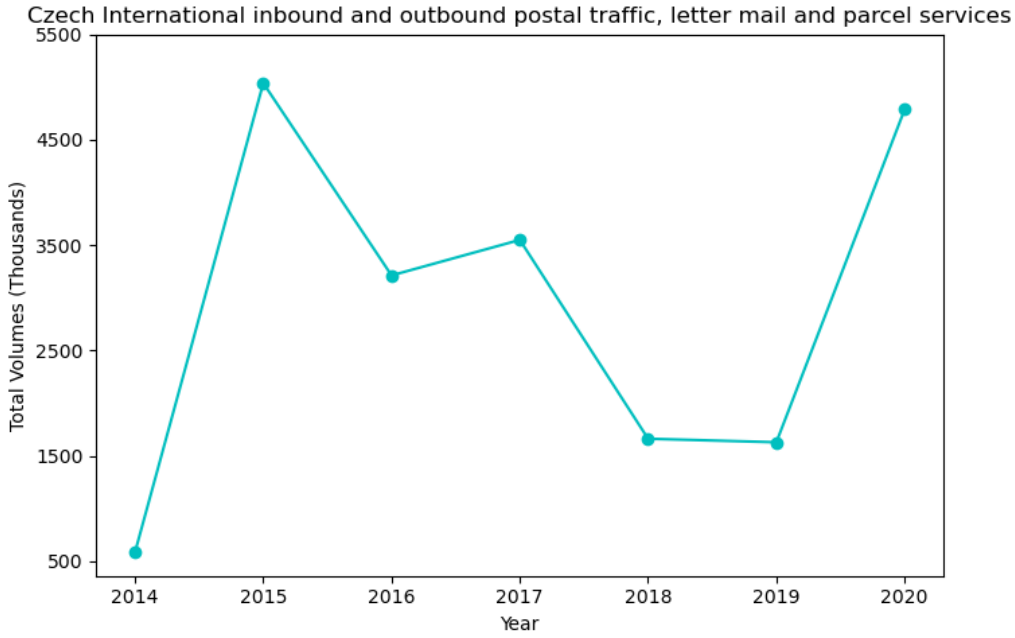


Figure 3.2 Statistics of total incoming postal traffic in thousands in the Czech Republic [45]

Upon dissecting the URL, we may notice that the phishing content is uploaded to a specific and often hidden directory. The phishing kit was uploaded to the *wp-admin* folder and further hidden in the *js/a* folder. We may check that the website was probably not created with the intent of hosting phishing content, by visiting the root URL, as can be seen in Figure 3.4. [3]

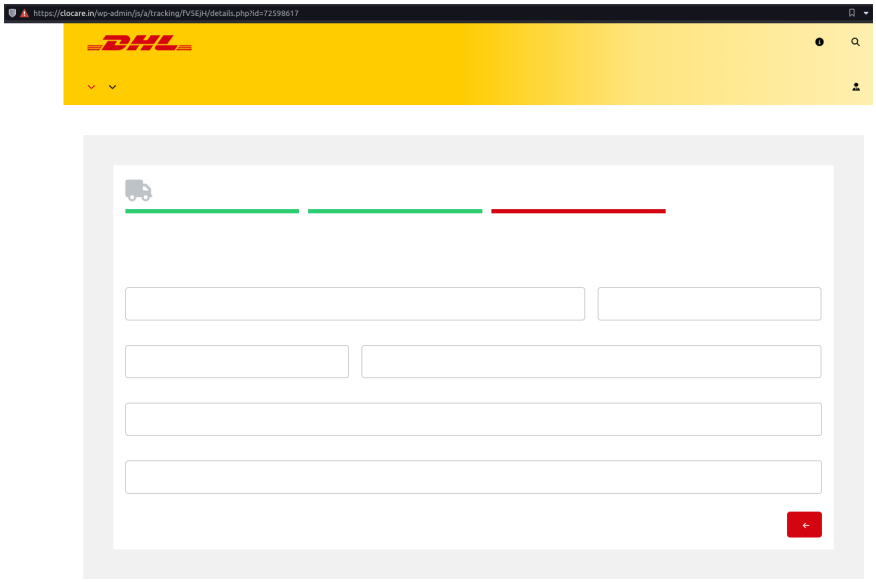


Figure 3.3 An example DHL phishing site hosted on an attacked Wordpress site

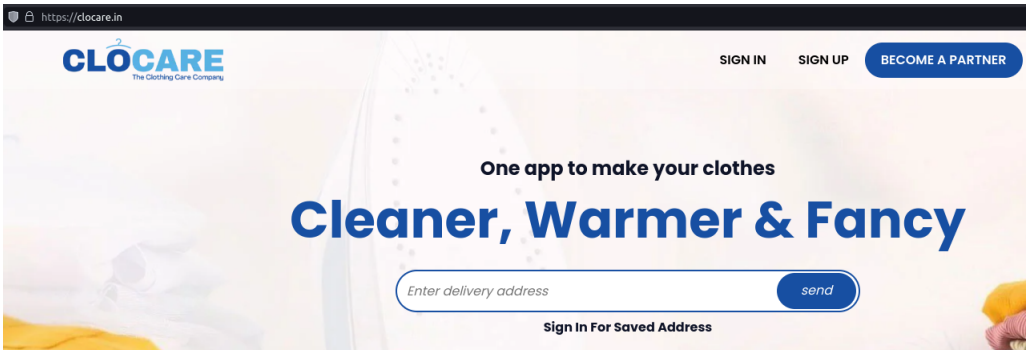


Figure 3.4 The original Wordpress site hosting non-phishing content.

This concludes our discussion and presentation of the proposed first method of phishing content hosting. To reiterate, the malicious actors identify vulnerable websites and upon successful breaching of the site, they upload an acquired phishing kit. These attacks are conducted en masse, as can be observed by the sites having the same directory structure. As this method was labelled to be the first, we now present the second which deals with TLS certificate registration and usage of subdomains.

When attackers register a domain, they may often not include the keyword of the attacked service in the Second Level Domain. From now on, domains on a lower level than second will be referred to as *subdomains*. Instead, the one registered domain may be used to host multiple

[3] It is important to note, that we are in no way attempting to single out the website *clocare.in* specifically, their website merely serves as an example of a scenario discussed in this thesis. The administrator of the website was contacted, to notify them of the found phishing content.

separate phishing pages, the data of which would be collected in a single database on the web server. By registering a TLS certificate with a wildcard subdomain, the attackers may start creating the separate phishing pages on their subdomains. This is used to hide the subdomains from certificate search engines and data collectors, as the wild card can represent anything. An example of this may be presented in another real-world sample, found by us. One such phishing site may be found at: `https://dpd-hr.delivery-orden.xyz/track/5151413414`

Which was inactive at time of writing, but a screenshot was recovered from the urlscan [4] service. This screenshot is included in Figure 3.5. While the image itself is not remarkable with displayed content, when compared to other phishing pages the discussed matter is perfectly illustrated in the presented URL. The Second Level Domain of *delivery-orden* is in itself not conspicuous, but the used subdomain of *dpd-hr* demonstrates the versatility of subdomain hosting. Not only can multiple services be targeted from one domain, this may also include language mutations to target different countries with the same service. The provided example shows a phishing attack against the Croatian branch of DPD, but the subdomain could easily be changed to *dpd-cz* and the content translated to target the Czech branch.

Subdomains may also be used for the purposes of misdirection and confusion of crawlers along with other automated services that would rely on Certificate Transparency logs alone. Let us consider crawlers as used to identify subdomains, or a variant that performs a dictionary-based bruteforce attack to find registered subdomains of the main domain. Upon finding a valid subdomain of a site, the crawler may continue searching for more subdomains or check for common directories on the found site. To combat this, phishing creators may create cycling subdomains or register certificates with large amounts of subdomains, that lead to nothing.

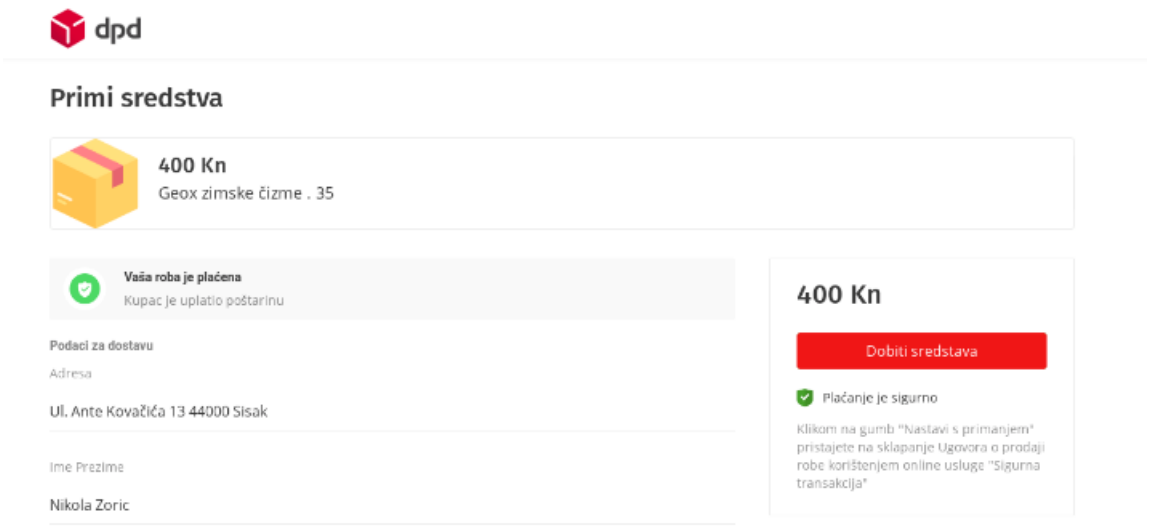


Figure 3.5 Subdomain hosted DPD phishing site [46]

Methods mentioned above are not an exhaustive list of phishing methods. Attackers may still register domains with the keywords directly in the registered domain, or innovate on new methods of distribution.

[4] https://urlscan.io/

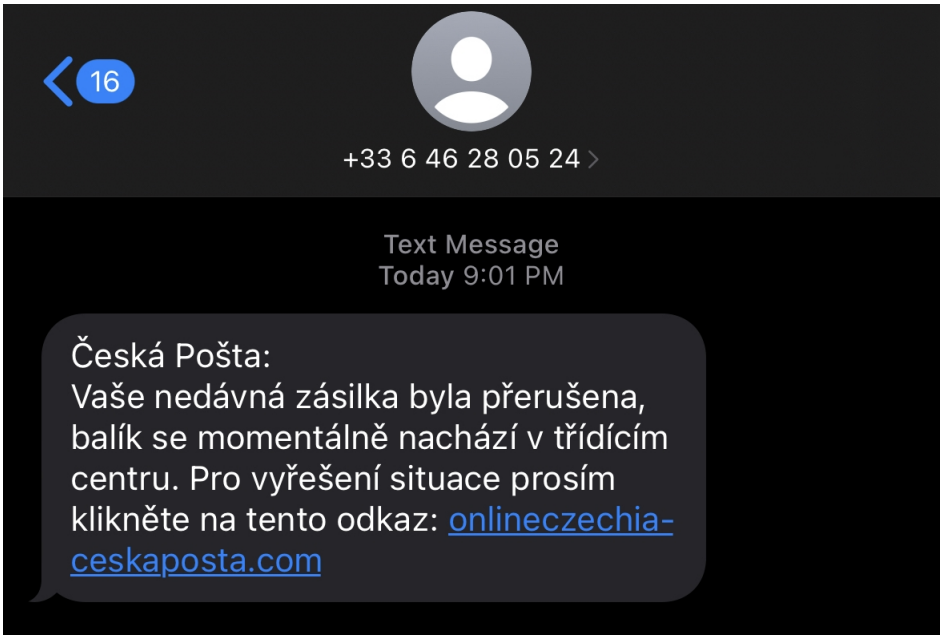


Figure 3.6 Example smishing message targeting the Czech Post

3.1.2 Reseller and Retail Applications

Reseller applications and second-hand marketplaces have become the perfect mechanism for phishing creators to find their victims, as they present themselves to them. The malicious party scours the application looking for postings that fit their criteria. Often they will attempt to find a way of contacting the victim out-of-bounds of the application via other messaging services, like WhatsApp, Signal or Telegram. Upon contacting the victim a prepared story is given, which leads to the malicious actor sending a phishing link. The link is sent under the pretense that the user will receive funds on their card or bank account after entering the needed information.

Used phishing pages and kits are slightly different from those used in the widespread Parcel Delivery phishing. While the same technique of phishing pages on subdomains is used, the pages are a lot more dynamic as the images and information included is tailored to the specific purchase. This makes the currently active URLs harder to identify for Threat Intelligence analysts and increases the likelihood of victims finding the sent phishing believable.

3.1.3 Domestic and Government services

The methods mentioned in Chapter 3.1.1 are applicable even when concerning domestic and government services. In the Czech Republic this has lately concerned the Czech Post (Česká Pošta) and the Ministry of Labour and Social Affairs (MPSV). Often, these services and institutions fall victim to smishing attacks, that are being distributed on a large scale among the population. Examples of the received messages may be observed in Figure 3.6 and Figure 3.7.

From the received messages we may observe common pressure tactics used in social engineering. The SMS conveys a sense of urgency, betting on the victim not taking their time to logically analyze the situation and notice the strange facts in the message. With the MPSV message we may notice that the incentive is financial gain, hoping that the victim will either be in a bad financial situation or greedy.

Phishing campaigns targeting the Czech Post are varied and common. Often they will include relevant keywords directly in the Second Level Domain name, and be registered in bigger waves.

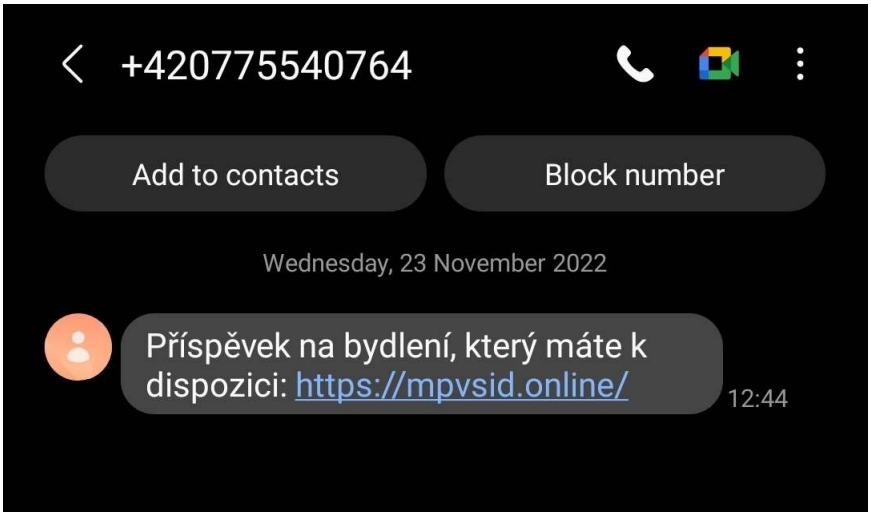


Figure 3.7 Example smishing message targeting MPSV

The purpose of Czech Post phishing is usually the collection of personal information, that ranges from the victims' address, phone number and name to their payment details, often requested under the guise of an additional fee. It is our assumption, that the collected phone numbers are then given out or sold for the further distribution of phishing via SMS.

In contrast, the MPSV phishing pages include multiple banking logins that are present with the excuse of a BankID login to claim the promised financial aid. Often the sites collect the initial login information, and then either request the second factor of the users' two factor authentication option or request the approval of a push notification. Thankfully, these domains are short-lived, resulting in periods of high turnover of MPSV phishing websites. Examples of domains that targeted the Ministry before, may be found on the website of the national registrar CZ.NIC that lists suspended domains [5].

3.2 Phishing Kits

The importance of Phishing Kits cannot be overstated, and for good reason. As national security institutions and service providers increase their attempts to spread awareness of phishing, the need to put up working phishing domains in a fast and efficient manner has increased. As mentioned in Chapter 3.1.1, using phishing kits also allows the attackers to spread phishing to vulnerable sites upon exploiting them. Kits not only save time, they also significantly lower the technical skill and knowledge required to setup phishing campaigns. This may be observed in the quality of setup of phishing sites, where the creators often demonstrate lack of Operational Security (OPSEC) or even the lack of technical prowess, as configuration files or even source code is available on the website due to incorrectly configured authorization. By lowering the barrier of entry, younger and often less experienced actors may join the space in hopes of achieving some sort of financial gain.

Advances in automation, and the ease of access to ready-to-use phishing sites are not all that the prevalence of kits has changed. With ever growing demand for the kits themselves and the results of phishing a secondary industry has sprung up, based around the distribution of phishing kits and reselling of stolen data. It is poignant to note, that the delivery of source code and maliciously obtained information does not happen exclusively on the *darkweb* or *deepweb*. In fact, there are plenty of *surface web* marketplace available. The administrative teams of these

[5] https://www.nic.cz/page/4310/aktualne-administrativne-vyrazene-domeny/ - keyword MPSV

Code listing 3.1 Czech Post phishing frontend configuration file

```
var url={
    "serviceURL":"https://cztcc.top",
    "redSwitch":0,
    "Visits":30,
    "country":"CN,CZ",
    "notCardNumber":"******,******,******",
    "config":2,
    "CPCurl":"https://www.ceskaposta.cz/",
    "TGAPI":"",
    "TGchat_id":""
}
```

sites have started refraining from unrestricted access to the uploaded free content, as they are an attractive and useful resource for threat intelligence researchers. The sites may now instead require users to be active posters before gaining access to the shared information, data and know how.

However, websites are not the only popular sources of the distributed content. The popular end-to-end encrypted messaging platform Telegram, previously hosted in Russia and at time of writing residing in Dubai, has also been used to distribute phishing source code and coordinate campaigns. Apart from the creation and organization of such campaigns, Telegram is also used for logging of collected user information. This can be done thanks to the provided web API and inherent ability to create automated bots. Such bots are given API keys, so that they may access chats that they have been invited to.

The described techniques and spreading of information via telegram was observed and discussed in an article published on Cyberark's blog, written by Zahi Ohana [47]. In the article Zahi writes, that they received a phishing link via sms. Upon investigating the site referenced in the sms, they found that the creators of the malicious site did not configure their access control correctly, and so Zahi et al. where able to access the phishing kit used to create the site. The kit included a configuration file, with the access ID of a telegram bot used for logging information. From here the blog [47] continues with exploration of the used services, joining the Telegram group and demonstrating the ease of phishing creation.

We managed to identify a similar situation, to illustrate that the usage of Telegram or a central logging web server has made its way to the Czech phishing landscape. A phishing site targeting the Czech Post had been identified at `https://ceskaposta-cz.xyz/payment.html` [6]. By analyzing the JavaScript code of the site, we found a configuration file located at `https://ceskaposta-cz.xyz/ResourceConfig/urlConfig.json` [7].

The contents of this file are listed in Listings 3.2, with comments removed due to the inclusion of unprintable characters. In the data of the configuration file, we may then find a central logging server in `https://cztcc.top` and two configuration variables; `TGAPI` and `TGchat_id` which would correspond to a Telegram bot API key and a chat id. The used settings may serve as a form of redundancy, if the logging server is removed from DNS records, then the creator simply adds Telegram information and the campaign will not be stopped. This file is a small example showing the advantages of phishing kits, and when compared to the information included in Zahi's Cyberark blog post [47] we may observe that the difference between Israeli and Czech phishing might no be so vast.

[6] UrlScan of phishing link - https://urlscan.io/result/894c50a9-ead1-449c-bb5d-c0373f7c5a8c/
[7] UrlScan of configuration file link - https://urlscan.io/result/8ee7a532-aace-4bdb-a144-9b414dbd804d/

Chapter 4
The Detection System

In this chapter, we will describe a phishing domain detection system, which is the aim and output of this thesis. Our main goal for the detection system was for it to be able to operate online over high-volume internet traffic, exported in the form of flow data. This presents many challenges that we describe in this chapter. One of the fundamental challenges lies in the general adoption of TLS, resulting in us being able to work with little more than a site's domain name. Furthermore, we cannot rely on the static analysis of accessed sites, thanks to the encryption provided by HTTPS connections. This added hurdle removes the often used signature-based detection methods, that could recognize previously identified phishing kits. We also decided that we would not go down the path of machine learning algorithms. Our reasoning behind this decision lies in our belief, that these systems often devolve into a sort of black-box system, which lacks transparency. With this our model may also be changed and adapted reasonably quickly, thanks to the fact that retraining will not be needed. Another challenge that we faced came from one of the requirements of the system, for it to be able to function over high-volumes of internet traffic. This required us to come up with solutions that would significantly lower the volume of data that would be worked on, as later operations may be more resource heavy. All of these decisions and testing will be covered in Chapter 4.1.

Based on our continual tests, measuring and proposal of improvements or changes we also encountered numerous issues with our design. This lead to the need of performing system overhauls, debugging and bringing forth more changes. All problems that we encountered during the testing process and the proposed solutions, or reasons for a lack of them, are described in Chapter 4.2.

The final look and functioning of our designed detection system is described in Chapter 4.3. When nearing the supposed end of our design process, we encountered issues during testing. These identified problems led us to believe that a larger redesign of the system was needed. Chapter 4.3 then contains our thoughts on how the redesigned system will address the mentioned issues and evaluates some parts of the system's performance.

4.1 Designing the System

Early on when re-creating the first proposals for our system we realised that a major hurdle to overcome would be the initial filtering stage. This initial filter, that we call the `pre-filter`, is a crucial part of our detection pipeline as the large volumes of incoming data need to be cut by significant amounts, before being sent for further processing. However, we also had to ensure that the `pre-filter` would not throw away potentially interesting domains while being able to consistently discard benign ones.

To provide some context for the actual need of creating an efficient initial filtering module we should discuss the realities of where the production system would be deployed. As mentioned in Chapter 1.3.1, we will be utilizing the NEMEA flow monitoring libraries. Thankfully, we did not need to build up the flow monitoring infrastructure from scratch, as CESNET already operates their own monitoring infrastructure, which includes collectors, exporters and all systems that allow us to receive flow data as we desire.

CESNET monitoring infrastructure operates on data from backbone internet connections, that run in the Czech Republic. This results in very high volumes of data, as the monitoring systems are able to collect data from connections with a bandwidth of 100GB. Considering that the current implementation of our system, and possible future extensions, will require more resource-heavy and time-consuming operations in later stages we had to ensure severe filtering of data. Our aim was to achieve over 60% of initial data filtering, while being able to retain domains of potential interest. The need for this became quite apparent when we reviewed our first proposal of the beginning of the system. Our rought first draft of the system, along with a proposed extension of this proof of concept can be seen in Figure 4.1.

The first step of our early draft would have included a database of current TLS certificates of legitimate websites, specifically those that we believed could be targeted by phishing attacks. When receiving flow data, the first module would query the database of inherently trusted certificates to see if the SNI extracted from the flow data is a trusted domain. If the domain is present in the database, it is discarded, as we can be sure that it will not be a phishing site. On the other hand, if it is not present in the database we might want to consider it. Of course, we realized that this setup has major flaws. Certificates of legitimate sites change often, they might not include all subdomains, Certificate Authorities may change and most importantly, for larger brands like Google, Meta or Microsoft the discovery and monitoring of all their site's certificates would be a time-consuming endeavour.

To add more criticism to our first thoughts, the design also didn't make adding new brands a simple process. If a user decides to add a new brand, which they worry may be targeted by phishing, they would first have to collect information on all sites that are associated with this brand. This collection could be done manually or automated using specific tools, but nevertheless the work would make it difficult to quickly adapt the system. In addition, we also realised that the database might be required to respond to hundreds of thousands of queries at a time, considering the nature of the network we are monitoring. This could lead to potential slowdowns of the system itself, as it's first stage would be reliant on the responsiveness of the database.

The later steps as shown in the diagram would concern domains that are not *official domains*, ie. the domain does not have a certificate record in our database. We did consider that this will affect the detection of sites using plaintext HTTP connections. However, we concluded that a separate and simpler detection system could be used for these types of connections, as this thesis is focused on detection in encrypted traffic.

4.1.1 Proof of Concept Detection Methods

Before explaining our reasoning behind the proof of concept detection steps, as can be seen in Figure 4.1, we should look at the theory behind our system. When preparing the first scheme, we were working with the idea of a detection system based on two kinds of indicators, the two types being *strong* and *weak* indicators. If we identify one *weak* indicator in a domain name, it does not immediately mean that we classify it as phishing. However, multiple *weak* indicators put together would increase the confidence of a domain being a phishing domain. This allows benign domains to have a couple of *weak* indicators, and still be considered harmless. On the other hand, *strong* indicators on their own are enough to increase our suspicion of a domain name. Here lies the issue, of selecting indicators that satisfy these criteria. In a best-case scenario, we would want *strong* indicators to be a feature that is significantly more common in phishing domains than in benign traffic. *Weak* indicators would then still have to be more present in phishing

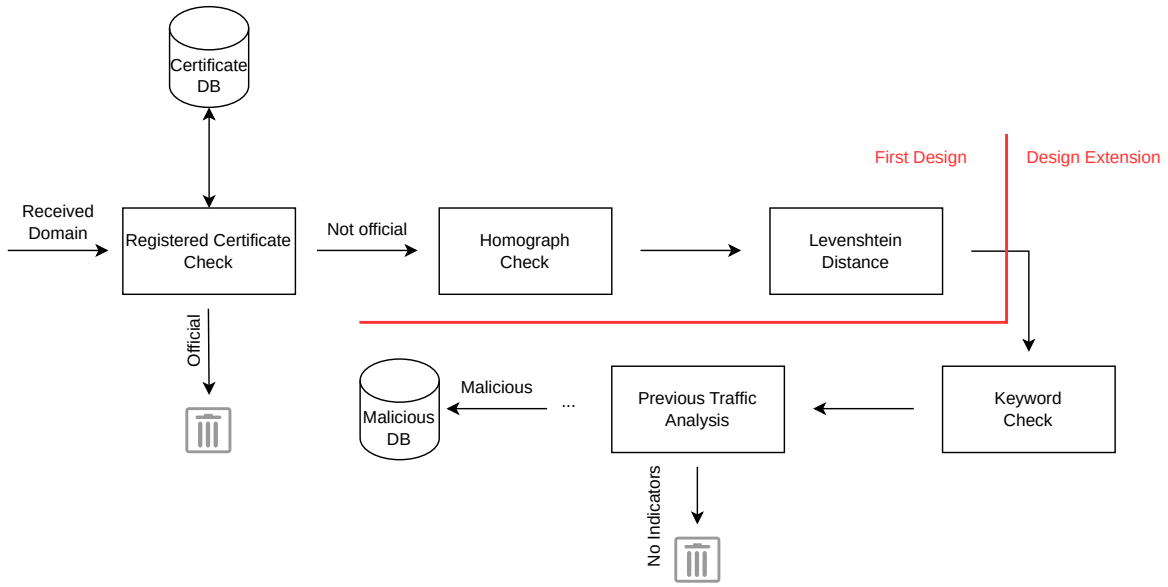


Figure 4.1 Proof of Concept System along with a proposed extension of it

than regular domains, but not so much that it is a unique feature. This delicate balance in the selection features makes it a very difficult problem. Now that we have described the more higher level concept of our intended two-type indicator system, we may return to the first two steps shown in Figure 4.1, specifically the steps before our proposed extension.

4.1.1.1 Proposed Weak Indicators

4.1.1.1.1 Homograph Check

The first of two proposed weak indicators was checking for a *homograph* attack in the domain name. A *homograph check*, would be a type of weak indicator. The technique of *homograph or homoglyph phishing* [48] relies on tricking the victim by registering domain names bearing a visual resemblance of the targeted site. A simple example of this is using the substitution of letters for numbers, sometimes referred to as *leetspeak*. In this alphabet the letter `o` corresponds to the number `0`, so a homograph of `facebook.com` could be `faceb00k.com`.

Another form of homoglyph attack utilizes International Domain Names (IDN). When using IDNs attackers the fact that letters from different alphabets may look the same, or close enough to the human eye. Like this phishers are able to register domains that have hyperlinks looking very similar to the targeted site, or ones that look exactly like the targeted domain. This is dangerous, because the often recommended security practice of mousing over hyperlinks, to view the actual underlying URL will not work. Most browsers however display Unicode characters using *punycode*, which displays non-ASCII characters in the limited alphabet of ASCII. An example of a punycode domain can be seen in [48], where the unicode domain *žugec.sk* is substituted by *xn–ugec-kbb.sk*. We do however acknowledge, that IDNs may be used for legitimate purposes. This is why we proposed their use as a weak indicator. We later scratched this idea, due to the low amounts of IDNs in the network.

4.1.1.1.2 Levenshtein Distance

The second proposed weak indicator was the usage of the Levenshtein Distance, an algorithm

for calculating the edit distance of two strings. This algorithm could be used to see if a keyword is present in the domain name or if a very close word is present. We considered the usage of this edit distance as a metric, because not all attackers will be obfuscating their domain names. In-fact, we argue that it might be more common to place a keyword, or a slight modification of it, directly in the domain name instead of using an IDN as mentioned in Chapter 4.1.1.1.1.

Using an edit distance metric like the Levenshtein Distance algorithm, we may also calculate the distance between two domains. Essentially stating how close one entire domain name is to another. This could again be used to indicate a phishing domain, as using a very close SNI on an unofficial domain should be suspicious at the very least. We tested this proposed usage, and found that there was no clear cut-off in distances from phishing and benign domains. After some consideration, we also proposed the usage of Levenshtein Distance on keywords in the domain name, instead of the whole name.

In order for us to find domains that include close keywords a threshold of minimum required edit distance would have to be established. This could yet again serve as a form of *weak* indicator, due to the nature of edit distances. For example, the valid banking domain of `csob.cz` is equally distant from `cs0b.cz` as it is from `csoa.cz`, making it difficult to precisely pick out which of these two domains might be malicious. This reasoning however could lead to high noise in the system, and potentially even increase the likelyhood of a False-Positive detection.

4.1.2 Proof of Concept Extension

Quickly after creating the initial design, we came forth with an extension of it, proposing further indicators and methods of phishing detection, as can be seen in Figure 4.1. In the second section of the diagram, we provide two more possible indicators of a malicious phishing domain in the form of checking for keywords in second and lower-level domains. These keyword checks would again be a form of weak indicator. Additionally, we proposed the idea of performing analysis of previous traffic, related to the domain. This would require significant correlation work, but if possible it would serve as a *strong* indicator.

4.1.2.1 Keyword Check

The Keyword Check is a weak indicator that is different from the Levenshtein Distance indicator, in that it checks for the direct presence of a keyword in the domain. This method is again relying on the fact, that phishing creators will include specific keywords to increase the trustworthiness of their domain name, and thus social engineer the victim into lowering their guard. Moreover, phishers now often include keywords not directly in the SLD but in lower subdomains., which we can easily look for using regular expressions.

Looking for specific keywords in domains lower than the second level serves a purpose. This may be used in addition to the Levenshtein distance calculation, in case of noise in the domain causing the distance level to be too high. Additionally, it may be used to attempt an combat the evasion methods used by certain malicious actors. Specifically, those that use a benign looking SLD, but host content on a lower-level domain which includes a keyword in its name. With this indicator, the selection of keywords is of vital importance. Proper selection of desired keywords to match can influence how the system behaves, more general keywords lead to more matches and more data for later observation.

4.1.2.2 Previous Traffic Analysis

Moving on to the first *strong* indicator proposed in our initial design, is the analysis of previous traffic. The logic behind this step may be explained using Figure 4.2 for reference. Often, malicious sites load images, Cascading Style Sheets (CSS) or fonts directly from the site they are trying to impersonate. This allows them to keep up with changes to the targeted site, but may

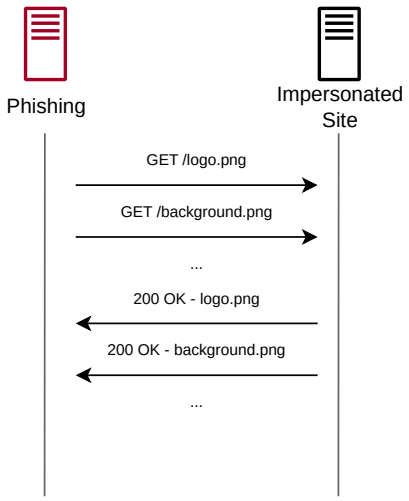


Figure 4.2 Visualization of a phishing site loading content dynamically from the targeted site

also be used to tell apart legitimate sites from the malicious ones. An important note is, that this content is loaded and requested on the client-side, not the on the server-side. This is what would allow us to group together traffic and identify possibly malicious behaviour. In the case of a dynamic site like this, the first request would be towards the phishing domain followed by a series of loading requests to the targeted site. Again, it is important to consider, that we do not see the transferred content or the full URLs of the sites. Thus we may only rely on effective grouping of traffic, telling us that the requests truly did happen in the order we assume they did.

These indicators of grouped traffic, need not be only dynamically loaded content. Some phishing sites also forward data entered by victims directly to the target sites. Allowing them to test if the entered credentials are correct, and in the case of internet banking, also perform immediate transactions or forward two-factor authentication data. This process is visualized again in Figure 4.3. If the data is forwarded from the client-side directly, then we would again be able to predict a timeline. The site is loaded, which equates to traffic pointing to the suspected domain. After that, traffic from the malicious domain will be interlaced with communication going towards the targeted site. Possibly in predictable intervals.

If such correlation was possible, it could be one of the strongest devised indicators. And it might indeed be possible for traffic collectors on local network deployments, where they have access to information of individual end-user devices and what domain each is communicating with. In our situation however, we do not have access to such precise information. What we are often left with instead, is usually hidden behind Network Address Translation (NAT), meaning that we do not have precise information of which end-user device communicated with what domain at what time. The common adoption of NAT networking makes our proposed correlation at-best difficult, and at-worst impossible. Considering that the targeted site may be a commonly visited site, like internet banking, it is entirely feasible that one user would be visiting a phishing site and another user, on the same NAT network, would be viewing the targeted site. Such occurrences would greatly increase the rate of False Positive detections. We acknowledge that this detection method may be of great use for detectors on local networks, as previously stated, but for our purposes it is not practical, and so the proposed method of traffic correlation was scratched entirely.

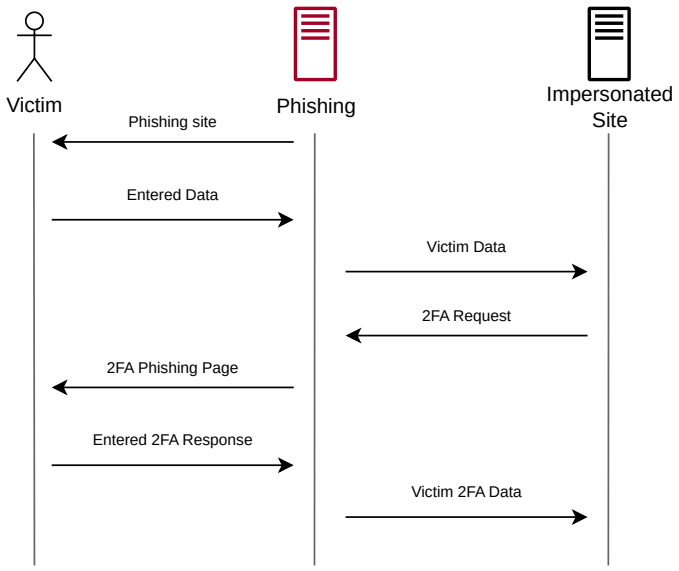


Figure 4.3 Visualization of a phishing site forwarding victim data to the targeted site

4.1.3 Evaluating the Proof of Concept

Even before performing real world testing of the proposed system, we decided to evaluate the difficulty of each task and how applicable the methods were for our needs. Again, we had to keep in mind that the system must be able to process high volumes of data and therefore work fast. We also introduced a two indicator system of *weak* and *strong* indicators. This meant that we had to come up with indicators fitting into these boxes.

4.1.3.1 Evaluating the Registered Certificate Pre-Filter

To ensure, that later stages could perform more resource intensive computations, the initial stages must significantly cut-down the received data. We refer to this first stage as the `pre-filter`.

Our first proposal for the `pre-filter` was described in Chapter 4.1 as the *Registered Certificate Check*. Thinking that there is no reason to analyze domains we know are trustworthy, we put-forth the idea of checking the received SNI against a database of *official* certificates. This database would include certificate information for each official domain of the targeted brand. We then realized that populating such a database would require continuous and significant effort. Not only in keeping the database up-to-date, but to also monitor all services of a brand and the domains that relate to them.

This method is sure to only filter out benign domains, but it also fails at identifying previously unseen safe domains. Also, the module would have to handle hundreds of thousands of subsequent queries to the database, which could lead to system slowdowns. Considering that the Registered Certificate Checking module would be able to filter only a limited set of domains, and potentially bottle-neck the system, we were forced to remove it from our designs.

4.1.3.2 Evaluating the Weak Indicators

In Chapter 4.1.1.1 and Chapter 4.1.2.1 we proposed three *weak* indicators for our proof of concept system. To reiterate, weak indicators are attributes of a domain name that do not indicate malicious intent on their own. Only when multiple such indicators are put together it is reasonable to assume that the domain is dangerous.

First we proposed the **Homoglyph Check**, specifically we wanted to check if a International Domain Name (IDN) was used. IDNs are represented in punycode, which we could identify, interpret and then attempt to see if the domain is targeting a monitored brand. In the end we considered this attribute a non-factor, due to the low occurrence of IDNs on the CESNET network. However, thanks to the modular nature of the utilized PyTrap framework a future extension of the system may implement a module for IDN detection.

Our second weak indicator was the **Levenshtein Distance** algorithm. This step would entail calculating the edit distance of a received domain from the domain of a monitored brand. For this instance we performed rudimentary testing and found that calculating the Levenshtein distance of two domains does not yield favourable results. In order for us to be able to differentiate between a benign domain and a potentially dangerous domain, we would have to be able to set a consistent threshold. Domain names however, include too many variable sections for this to happen.

The third and final weak indicator of **Keyword Checks** monitors if a keyword is directly present in any part of the domain name. This technique may seem close to the previously mentioned edit distance check, but lacks the disadvantage of producing noisy data. It does however rely on keywords picked by the user, if the picked keywords are too general the resulting data will appear noisy. We consider this indicator as useful, and so it was used in the final system.

4.1.4 Pre-Filtering Module

After deciding that cutting down the amount of received traffic was our first priority, we decided that the best course of action was to identify where phishing domains differ from benign ones. If we would be able to pick out features, that are fast and simple to programmatically find, we could utilize them in the pre-filtering stage. The requirements were focused on speed, discarding of common domains and allowing interesting domains to pass through. For us to be able to extract these features, we analysed the datasets described in Chapter 2 and performed a comparative analysis of the results.

4.1.4.1 Picking Features of Domain Names

In order for us to be able to perform a comparative analysis of benign domains and confirmed phishing domains, we had to first identify which domain name features to compare. For us to be able to pick apart the features of domain names, we looked at the general structure. In general a domain is made up of three parts, the Top Level Domain (TLD), Second Level Domain (SLD) and lower level domains (subdomains). Often, we may not give much thought to the kind of information that is included in all the mentioned sections. Features or common patterns that we identified as important are included in our mind-map, which can be seen in Figure 4.4. The domain name as a whole may also be used to extract meta-information. When we say meta-information, we mean information that is not included directly in the domain name itself but a direct consequence of the structure, chosen words, TLD and so on. An example could be a statistical feature of the domain.

Our first attempts at picking out features of domain names began with the assumption, that we will perform a fast online analysis of the whole domain name. When considering this, we looked at the overall goals that a phishing creator will be trying to achieve with the whole domain name. Two goals will often be at odds with each other. On one hand, the creator will want to obfuscate their intent, making it hard for detection engines to recognize their domain as phishing and the same for the end-user. On the other hand, the malicious party will also want their domain to be believably close to the site of the targeted service. Looking at these two contradicting goals, we came up with two possible indicators that a site may not be benign and should be let through the pre-filter, these are described in Chapter 4.1.4.2 and Chapter 4.1.4.3.

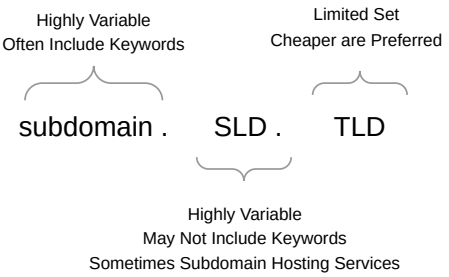


Figure 4.4 Visual Analysis of a general domain name

Later we also aimed at picking apart what information could be gleaned from each section of a domain name, specifically what is important to phishing creators. Starting from the top, we may first look at the TLD. This section of the domain name is more straightforward, thanks to the fact, that TLDs may only be selected from a limited set of approved names. One of the common and desirable Top Level Domain names is *.com*. Often using a more common and desired TLD makes acquisition of the domain name more expensive, which may not be desirable for the phishers. This was briefly discussed in Chapter 3.1, where we looked at possible incentives for the creators of phishing. One of the major incentives will often be financial gain, which would be minimized when attempting to acquire expensive domains. However, selecting less popular TLDs may also serve a strictly practical purpose. Let us demonstrate this with a simple example, let us say that a phisher is targeting `google.com`, then they could search for the availability of domains like `google.tech`, `google.xyz` and so on. If one of these very close domains are free, then the phishing creator may have acquired a very believable domain name.

The second section of domain names is the Second Level Domain, which is highly variable when compared to the previously discussed Top Level Domain. This is where creativity of the phishing creators comes into play. As discussed in Chapter 4.1.1.1.1 phishers may register domains with very close names to the targeted brand. For example, when attempting to copy the official page of `mbank.cz` the attackers may use a domain like `nnbank.cz` or `mbamk.cz`. We have however noticed a trend of phishing domains not including the name of the targeted brand directly in the SLD, instead they host multiple pages targeting a variety of brands on different subdomains. This has in-fact become a service offered by providers like Duck DNS [1], where the provider allows a user to acquire a subdomain on their domain with the DNS record pointing to the user's server.

Moving over to all domain names lower than the SLDs, which we will call subdomains we may notice the same problem of high variability in domain names. However, if the creators of the phishing campaign are not masking their intent, the subdomains will often directly include keywords related to the targeted brand. When phishers opt to use this method, they will often registered a TLS certificate with a wildcard like so: `*.domain.com`, which allows them to use HTTPS connections for any subdomain. This also makes it more difficult for Threat Intelligence teams to identify their malicious subdomains from information stored in the site's HTTPS certificate.

4.1.4.2 Shannon's Entropy of Domain Names

If we first consider the goal of evasion, our thoughts went to the calculation of Shannon's entropy. In layman's terms, Shannon's entropy states how difficult it would be for us to predict the following character in a stream. [49] The higher the entropy, the higher the difficulty. Considering this, we may reasonably assume that malicious domains would have a higher than average level

[1] https://www.duckdns.org/

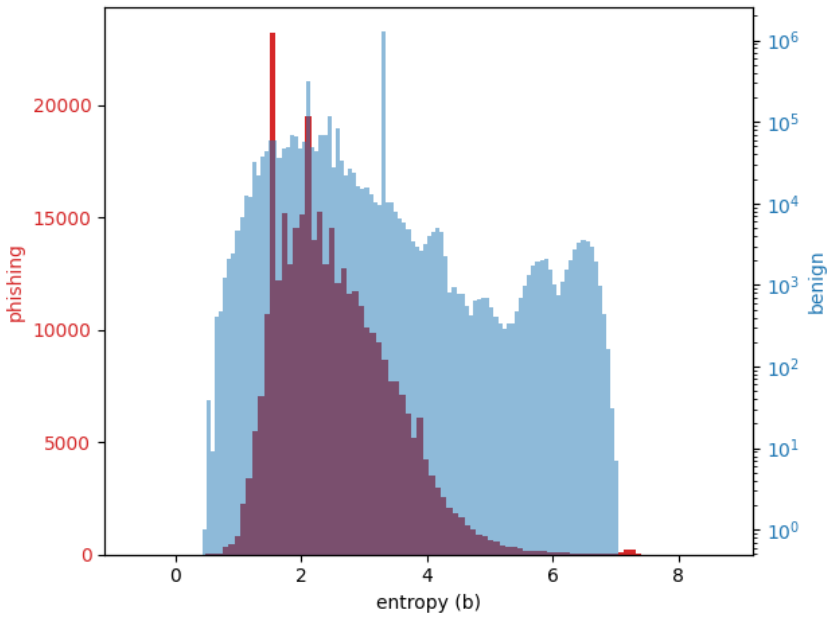


Figure 4.5 Comparison of entropy histograms for the benign and phishing datasets

of entropy. This should be the case in circumstances, where the creators decided not to include keywords directly in the domain name, and are instead attempting to prevent any linguistic analysis of their domain, making it a kind of adversarial attack.

We performed a calculation of entropy for each domain in our benign set and in the phishing set, which we then compared. Our calculation was done according to the following formula [50]:

$$H(X) = -\sum p(x) \log_2 p(x)$$

Where $p(x)$ is the probability of occurrence of a given letter in the domain. This we calculated as the total amount of occurrences of a single letter, divided by the length of the permitted domain name alphabet. After performing all calculations we plotted out the results for each set into a histogram, which we then compared as can be seen in Figure 4.5. Based on the results we noted that entropy in the benign dataset was generally high, with significant overlap in entropy values with the phishing data. Therefore we deemed the metric as not useful for our purposes. After another overview of the benign data, we came forth with a theory, as to why the general entropy was so high.

Upon reviewing the benign data, we noticed that some domain names were captured only in part. This meant that the SLD and TLD might have been cut off, or in some cases the TLD only. At first reading this might be a cause for concern, and the presented histogram will certainly be, in a sense, skewed. We however take this as hard proof that relying on entropy is not the right path, and for a simple reason.

Looking at some of the domain names with entropy on the higher-end we may find examples like the following: `smqcazyccvfgwyvzoy3q-phtu4e-019b28156-clientnsv4-s.akamaihd.net`. The presented domain is clearly intact and received without error. The high entropy of roughly $6.897b$ is caused by the seemingly highly random subdomain, and this is a pattern that we observe even with the incomplete domain names. An example of an seemingly incomplete SNI may be: `p4-hdrcfnvjejvyw-7wekomgzbcjt5322-106089-i2.valid.gexperiments5`. While the entropy of this domain name is a high outlier at $7.052b$, completion of the domain name would likely only increase the resulting entropy value or affect it minimally.

Moreover, looking at the histogram for benign data in Figure 4.5, we may see that the overlap between benign entropy values and entropy in phishing would not be affected enough to warrant entropy as an indicator. Setting a reliable threshold of entropy, to classify data as phishing, would require much larger differences in calculated results. This is the result of severely underestimating the presence and nature of Content Delivery Network (CDN) domains along with highly randomized subdomains of specific services, like analytical collectors.

4.1.4.3 Coleman-Liau Readability Index

The Coleman-Liau [51] readability index is used to determine how difficult it is to understand the analyzed piece of text. Originally, the algorithm was intended to tell the level of education required for a person to understand some given text. This is why the index formula relies on the presence of sentences and letters, which we attempted to adapt for our purposes.

We chose the Coleman-Liau index from the plethora of readability indexes for our testing, because of the simple calculation formula, lending itself for our requirement of high-speed processing. The reason why we tested the validity of using a readability index for filtering, was the second goal of phishing creators that we mentioned in Chapter 4.1.4.1. When preparing for a phishing campaign, often the phishers will pick domain names that should be simple to understand, include common words, phrases, related keywords and potentially may include multiple subdomains. All of this combined may then result in phishing domains like `zasilkovna-site-cz.fundsmart.store` [2].

The formula for the calculation of the Coleman-Liau readability index is as follows [51]:

$$0.0588L - 0.296S - 15.8 \tag{4.1}$$

Where L is the average number of letters per 100 words and S being the average number of sentences per 100 words. After deciding to use this readability index for the very simple formula, we also had to decide what constituted a sentence and a letter in a domain name. Letters remain unchanged, they are simply limited to the allowed alphabet in domain names. When it came to sentences, we decided that the whole domain name would be one single sentence. To instead satisfy the requirement of having at-least 100 words, we treated each domain level as a word. Based on this we then calculated the average *letters* per word and average *sentences* per word, both of which we then multiplied by 100:

$$L = \frac{letters}{words} * 100$$

$$S = \frac{1}{words} * 100$$

This is not ideal for the computation of the Coleman-Liau index, but we decided to test if the adapted index calculation could be applied in our specific use-case. If the results demonstrably favoured one side of the classified domain names, we would have another useful, simple and fast indicator.

Again, we calculated the indexes for all domains in a benign set of data and compared that with the resulting calculations of the same index for a set of phishing domains. These results were then plotted out into histograms for comparison in Figure 4.6.

Looking at the resulting Coleman-Liau index calculations as seen in Figure 4.6, we may first see that most domain names have a low readability index. This goes for both phishing and benign samples, and for good reason. Benign domains will often want to be simple to read, to look attractive to users, especially with the boom of internet commerce. Conventional phishing domains will also want to be highly readable, to appear trustworthy to the victim.

[2] Found on urlscan.io

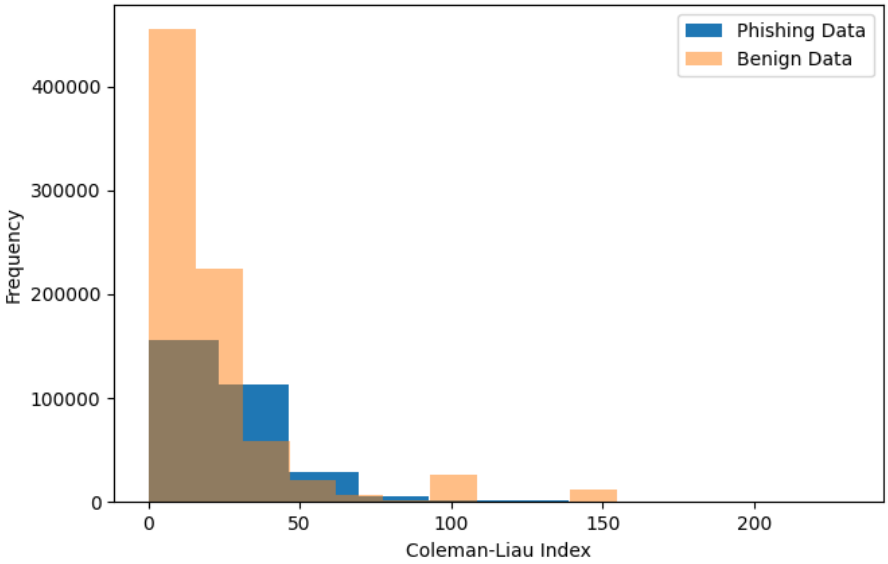


Figure 4.6 Frequencies of Coleman-Liau calculations for Benign and Phishing data

It is important to recall that our calculation of the index is adapted in an attempt to apply a calculation meant for longer text to domain names. This is clear in the results, as we had to clean the data before plotting it out. Results for the phishing dataset included a total of 21578 negative values, out of a total of 328085 phishing domains, roughly 6.5%. The same was observed for the benign data, where we identified a total of 170490 negative index values out of the total 1374238 domain names, around 12.4%.

Our second observation from Figure 4.6 is the significant overlap in values for benign and phishing domains. While there are some differences, the gaps are not wide enough to constitute a reliable indicator that would not result in a high False-Positive rate. Based on this we conclude that the adapted Coleman-Liau calculation is not suitable for our proposed system.

4.1.4.4 Automated Readability Index

A very close readability metric to the Coleman-Liau index is the **Automated Readability Index** [51]. Much like the Coleman index, the Automated Readability Index (ARI) relies on the usage of simple variables: words, sentences and letters. While the Index is again meant to be used on larger text samples, we attempted to use the same logic as with the Coleman-Liau index in Chapter 4.1.4.3. Characters are simple, each character except for dots is accounted for. Words are each domain level, and the sentence count remains at one. This is then put into the ARI equation (4.1.4.4).

$$4.71(\frac{characters}{words}) + 0.5(\frac{words}{sentences}) - 21.43 \tag{4.2}$$

Unlike when calculating the Coleman-Liau index, the recommended text length for ARI calculation is 10 pages, which is hard to translate into domain names. That is why we input the numbers directly in the equation, without any modification or multiplication. Thanks to the similar calculation to the Coleman index, we also performed a comparative analysis, which can be seen in Figure 4.7.

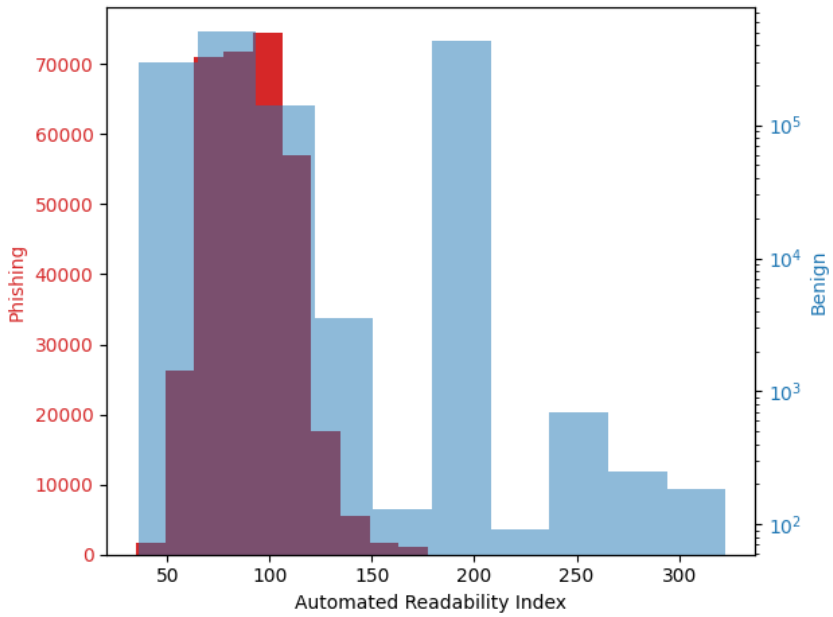


Figure 4.7 Frequencies of ARI values for Benign and Phishing data

Based on the comparison of ARI values for benign and phishing data in Figure 4.7 we could see results very close to those of the Coleman-Liau index. This should not be surprising, due to similar equations and variables of the metric. Much like the Coleman calculation, we also had to attempt and adapt the calculation to fit a domain name. Which might be reflected in the comparison results. From Figure 4.7 we may conclude, that there is significant overlap in the Advanced Readability Index values for both phishing and benign data. Resulting in no clear separation between the two. An argument could be made, that from the index values of 150 and higher, there is clear dominance for benign domains and thus may be grounds for identification of benign domains. We however argue, that this is down to the separation in quantities of data between the benign set collected from the CESNET network and the phishing data.

4.1.4.4.1 On other Readability Indexes

After we conducted our tests of the Coleman-Liau index and the Automated Readability Index, we were forced to completely abandon the idea of readability calculations as a viable metric. We do however feel, that since there are a plethora of readability algorithms available we should take a moment to explain why we did not select them.

There was a specific reason why we picked out the mentioned methods, instead of the others. When considering which algorithms are viable for our use-case, we had to consider our goals. One of the most important goals is speed, meaning that resource intensive operations should not be considered in the `pre-filter`. Considering that readability algorithms tell us how difficult it is to understand a given piece of text, it is reasonable that a part of them rely on linguistic analysis. This kind of analysis requires processing of an entire string, often also requiring pre-processing before extracting linguistic features. If we consider that the system will receive thousands of domain names per second, then the cost of operations starts to increase dramatically.

First let us explain why the Coleman-Liau index and the ARI were considered viable. Let us re-visit the very simple calculation of the index from Equation 4.1.4.3. One of the main advantages of this metric lies in the fact, that it relies on three simple variables. The three

variables of L, S and word count are very simple to extract from text and from domain names, with our proposed modification. Extracting information from domains without having to rely on more complex linguistic analysis algorithms brings about the advantage of speed, which is critical for our system, as mentioned before. These features are shared by the Automated Readability Index as well, which was why it was the second considered metric.

Other notable readability metrics include [51] the SMOG index, the Flesch-Kincaid index and the Gunning Fog index. **SMOG** [51] relies on the extraction of *polysyllables* from the text, which does require more searching and context-aware calculation than our simple modifications of Coleman and ARI. The **Flesch-Kincaid index** [51] relies on a similar feature in *syllables*, which present the same obstacle. Lastly the **Gunning Fog index** [51] presents an idea of *complex words* in its equation. A *complex word* is a word made-up of three or more syllables, with the exception being verb forms that include three syllables thanks to the addition of -ed or -es, proper names or combinations of short words. From these restrictions we can see that the calculation of the Gunning Fog index, might be the most complex of all presented.

We hope that the reasoning behind our limited consideration of algorithms is now clearer, and illustrates the difficulty of selecting appropriate methods fitting into our restrictive criteria. Selecting metrics that are common enough for phishing domains, and less common in benign domains is a difficult task in itself. When considering the added restrictions of our requirements for speed and high-filtering capabilities it should be clear, why we were forced to test many metrics. It would also explain why the readability indexes failed in our conditions, as systems with less strict requirements may find them useful. We also admit that perhaps a different modification of the index calculations may be viable. This would however require a second thesis or evaluation, that we were not able to conduct.

4.1.4.5 Domain Name Splitting

Based on the ideas written about in Chapter 4.1.4.1, we decided to extract features of domain names after *splitting* them by each *domain level*. Here we separate the SNI into three distinct sections, the Top Level Domain (TLD), the Second Level Domain (SLD) and all lower-level domains we call subdomains. Each level has certain characteristics, specific to phishing or benign domains that we thought may be useful in the `pre-filter` module.

Based on our visual observations of domain name features, as shown in Figure 4.4, we proposed the usage of three features for the `pre-filter`. The first feature was inspecting the used **Top Level Domain (TLD)**. This decision relied on two assumptions, first that phishing creators are focused on the financial side of phishing. Meaning that registering domains with cheaper and less common *TLDs* is more cost effective, as a *.com* domain may be significantly more costly than a *.xyz* or *.io* domain name. Our second assumption was that the malicious actors would search for *duplicates* of brand domain names, with less desirable *TLDs*. As an example, consider the official site of *brand.com*. An attacker might then check if *brand.xyz* is an available domain, and if that is the case, acquire it.

After checking the *TLD* we continued down the levels based on our visual analysis, and landed at the **Second Level Domain (SLD)**. Here we concluded that while the usage of keywords may occur, we saw a rising trend in utilizing services of Dynamics DNS Resolution. Using these services, phishers may utilize trusted Second Level Domains to host brand specific phishing on a subdomain. Thus if we were able to identify either a private SLD or Dynamic DNS provider, we would consider the domain interesting. Based on the fact, that it already passed one filter. We called this step the *Subdomain Hosting Check*

Instead of using a linear approach, we put the check of *Subdomain Hosting* and a **Keyword Matching** check in a logical OR. Thus allowing a more robust filtering system, that does not let through only one specific type of phishing, which would be the case for linear deployment. Thus, the virtual *last step* of the proposed `pre-filter` was a simple check for the presence of an interesting keyword.

All the methods put together form the `pre-filtering` module, that we used in our first testing deployments. After much deliberation and testing we were able to create a filtering module, that fulfilled our criteria of speedy performance, high cut-down of data and allowing potentially interesting data to pass through.

4.1.4.5.1 (e)TLD Check

Our initial attempts at creating a *TLD-based* filter focused on the difference in occurrence between *TLDs* in phishing data and in our collected benign data. Using this we would be able to construct a list, that a filtering module simply matches for using an efficient system like *regular expressions*.

When checking if a domain name contains a less common, or potentially suspicious TLD, we initially failed to factor-in the existence of *effective TLDs (**eTLDs**)*. This is where our simple and general split of domain names into three simple sections does not work as well. An *eTLD* is made up of the TLD and SLD, forming an *eTLD* like *.com.au*. Only below the *eTLD* we may find what would normally be considered the second-level or the actual domain name. Taking this into account, we re-calibrated and performed yet another comparative analysis of benign data and phishing samples. This time we counted occurrences of both *TLDs* and *eTLDs* based on a public dataset from Mozzila [3]. We kept a local copy of the data, cleaned it of comments and utilized only the *ICANN Domains* section of the set. From here our method was simple, we counted the occurrences of either *eTLDs* or *TLDs* in phishing and benign data.

After we calculated the difference in occurrence between the two sets we could see a clear split in the frequencies of top level domains. As we suspected, *eTLDs* and less common *TLDs* were heavily favoured in the phishing data. Here it is important to keep in mind, that the dataset of benign data is larger than the phishing set. If a *(e)TLD* is significantly favoured by the malicious data, it is a clear indicator of heavy usage on the phishing creator side. Also, when there was no difference in usage (ie. the difference is 0) we still considered the *(e)TLD* favoured in phishing, due to the larger sample size of benign data.

Based on the results we were able to create and test a module that filtered domains based on the used *(e)TLD*. This small filter was the first step to our first functioning `pre-filtering` module. We did however notice, that potentially interesting domains were being thrown away due to them not having a less common *(e)TLD*. To makeup for some potential loss, we utilized our practical experience with phishing domains as described in Chapter 3 and decided to add three top level domain names; `.cz`, `.com` and `.jp` all three of which we noticed were often used in phishing domains confirmed by us.

While this decision did increase the total throughput of the *(e)TLD-based* filter, we knew that additional steps would follow after. And so we could focus our efforts on lowering the total amount of data allowed to pass in the next stages.

4.1.4.5.2 Subdomain Check

When analyzing Second Level Domains for common features in phishing, we decided to rely on our practical experience with phishing domains. Often times, phishing content is hosted on a subdomain of either a trusted SLD or a completely benign one. Identifying a completely benign SLD as a phishing site, without checking the subdomains, is not possible. This is why we turned to the second option, where phishing creators use trusted sites to host their content.

Dynamic DNS (DDNS) resolution services, like *duckdns.org* [4] allow users to essentially point a subdomain of *duckdns* to an IP address of their choice. Utilizing these kinds of services, malicious parties only need access to a public IP address, where they host the phishing content itself. After securing a way to make their content publicly available, the attackers can identify an available

[3] https://publicsuffix.org/list/public_suffix_list.dat
[4] https://www.duckdns.org/about.jsp

subdomain on the service providers site. The subdomains will often be named according to the targeted brand, including keywords or the name of the brand directly. An example of this can be seen in the following domain: `postaksnakq.duckdns.org`, which was targeting the German bank *Postbank*.

Additionally, cloud providers also allow users to specify the name of their hosted site, which is also included in the subdomain. This is different from Dynamic DNS services, in that most of the time, the malicious content is then uploaded directly to the utilized cloud. For us to be able to identify these services, both Dynamic DNS and subdomain hosting, we needed another set of data.

Thankfully the list from Mozilla used in Chapter 4.1.4.5.1 to identify frequency of *eTLDs* and *TLDs* in both benign and phishing data also included a list of *Private Domains*. These domains include cloud hosting and DDNS providers. This time we did not perform a comparative analysis, instead if a domain included a name from the list, we let it pass through the filter.

Domains from the Mozilla private domain list are not especially common on the CESNET network, and so this on its own achieved significant cut-down of domains allowed to pass. This filtering module was put after the initial *(e)TLD* filter, and in parallel with the next function in Chapter 4.1.4.5.3.

4.1.4.5.3 Keyword Check

Phishing creators often have an incentive to include keywords in the phishing domain name, as part of the social engineering nature of phishing attacks. Making victims believe, that the domain is trustworthy or related to the targeted brand or service is vital. For the attackers to achieve this, they may include keywords in the domain name.

Let us remind ourselves, that our main goal with the `pre-filtering` module is to reduce the large volumes of received traffic, with the aim of allowing later processing. However, when filtering out received domains, we must make sure that most of the discarded domains are benign and potentially interesting domains are sent for later processing.

One of the simplest ways to decide if a domain is interesting for closer inspection is through checking for the presence of selected keywords. This allows for the monitoring of domains including specific brand keywords, or vocabulary targeting a specific sector. If a new trend is uncovered, the filter may also be adapted to look for the new words in domain names.

The search is done by transforming the received list of keywords into a set of regular expressions, which are then matched for over levels of the domain name. We limited the filter to the 4th level of domain name, purely because of performance. Allowing the filter to search through all the levels of a domain name could slow down the `pre-filtering` system and thus not fulfil one of our main requirements. Moreover, from our practical experience with phishing domains we concluded that phishing domains with more than 4 levels of domain name are not overly common.

Performance of this filtering function was acceptable for our needs, and so we added it to the resulting module. Filtering domains based on the presence of keywords is done after *(e)TLD* filtering and in-parallel with *subdomain hosting checks*, set in a logical OR.

4.1.4.5.4 Common Check

After putting together all the functions of *(e)TLD checks*, *Subdomain Checks* and *Keyword Checks* we found significant amounts of similar domains passing through the filters. Along with potentially interesting domains, we found repeating benign SNIs.

Thanks to noticing a pattern, we decided that an additional filtering function was required. The main purpose of this function would be to quickly filter out *common domains*, that we know pose no risk. For us to be able to identify these domains, we put together a list of the most

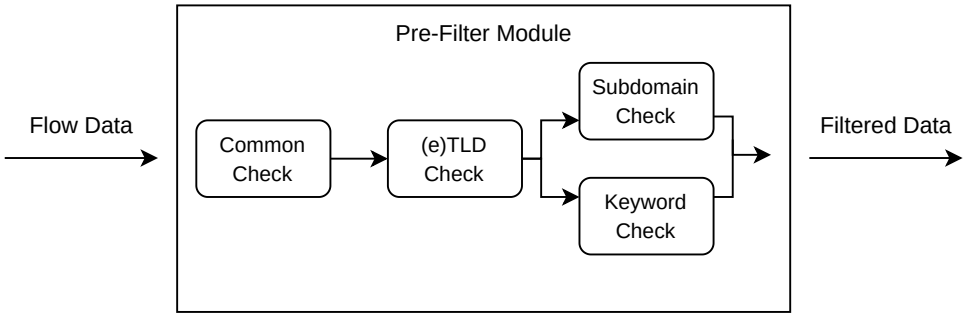


Figure 4.8 Visualized pre-filtering module

common domain names passing through the filtering functions. This resulted in a long list, from which we selected the top 1000 and used them as a reference list to check incoming domains.

It is important to note, that we did not count the occurrence of complete SNIs, instead we only looked at the SLD and TLD. This was done, because a pattern was seen in the output where domain names of CDN services were allowed to pass. Moreover, because of the *Keyword Check* function discussed in Chapter 4.1.4.5.3 we observed the official domains of brands whose keywords we selected in the output. This is why we also manually extended the list to include official domain names of selected brands.

The function first reads the first 1000 entries in the specified common filter wordlist, after this our manually added domain names are concatenated to the list. When all known SNIs are loaded into memory, they are converted into one long regular expression. This expression is then used to check if a match is found in any of the received domain names. All matched domains are discarded and the remaining domains are sent to the *(e)TLD Check*.

4.1.4.6 Resulting Pre-Filter Module

Our working `pre-filter` module consisted of the four functions described in Chapter 4.1.4.5 put together. This module is the first line between data received from the Flow Monitoring collectors on the CESNET network and the rest of the detection system. After filtering the received data, domain names that pass the filtering checks are exported for further processing by the following modules.

This module is visualized in Figure 4.8, where we can see how the previously mentioned functions are put together. The module, along with all other modules in this thesis, was written in the Python programming language, using the *pytrap* library [5]. Design-wise, the filtering functions are included in a single object, the *FilterObject*. All four functions are then called directly from this object, with filter lists passed to the individual filtering functions. The structure of the whole module can be seen in the following directory structure:

```
/
├── filterObject.py
├── prefilter.py
├── modules
│   ├── common_filter.py
│   ├── etld_filter.py
│   └── subdomain_filter.py
├── common_filter.txt
├── etld_filter.txt
├── subdomain_filter.txt
└── keyword_filter.txt
```

[5] https://nemea.liberouter.org/doc/pytrap/index.html

Code listing 4.1 pytrap driver of the pre-filter module

```
while True:
    try:
        data = trap.recvBulk(rec, time=10, count=100000)

        pd_data = pd.DataFrame(data)
    except pytrap.FormatChanged as e:
        fmttype, inputspec = trap.getDataFmt(0)
        rec = pytrap.UnirecTemplate(inputspec)
        data = e.data
    except:
        continue
    if not data:
        break

    caught,total_time = f.filter(pd_data)
    caught = caught.reset_index(drop=True)
    caught_js = caught.to_json(index=False, orient='table')
    caught_js = loads(caught_js)['data']

    for json_data in caught_js:
        trap.send(bytearray(dumps(json_data), "utf-8"))

trap.finalize()
```

Interfacing with the *pytrap* API is done in the *trapPrefilter.py* file, where the *FilterObject* is imported from the *filterObject.py* file. *FilterObject* then calls functions from all scripts included in the *modules/* directory to perform the filtering operations as illustrated in Figure 4.8. Data for the individual filtering functions is then included in the text files.

The code shown in Listings 4.1.4.6 shows the main code, that is responsible for handling all *pytrap* communication. With the current *pytrap* implementation, the code runs in a continuous loop waiting for the buffer of collected data to reach our set limit of 100000 collected domains. After the buffer reaches the desired capacity, or if no new data is received after X seconds, we then proceed to filtering. We expect the received flow data to be in a specific UNIREC template format, which we specify at the start of the script. To check if the received data is in our expected format, we use the *try-catch* block. When the format is not met, *pytrap* can still attempt to parse the data and extract the used format.

After parsing all information from the received flow monitoring systems, we store the information as a *Pandas Dataframe* [6] and pass the domain names to the previously mentioned *FilterObject*. From this object, we call all four filtering functions, and pass it the required lists in text files. The order in which functions are called can be seen in Figure 4.8. All domains that did not pass the filtering checks are dropped from the *Dataframe*, and converted to JSON format. This converted data is then sent for further processing, using the *pytrap* communication interface.

Our initial tests of the finalized module were conducted on the sets of data collected from the CESNET network, which we used as our reference benign set. Used data sets were discussed in Chapter 2. These tests showed us that the designed module is capable of filtering over 85% of received traffic, leaving only 15% or less for analysis. This is promising and desired, as the resulting data also includes potentially interesting domain names.

After completing initial tests on local data, we deployed the module to CESNET infrastructure to ensure live behaviour was close to our local testing. When deployed on the live monitoring

[6] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

infrastructure, we observed that the filtering capabilities remained about the same as when tested locally. Based on these tests, we concluded that the `pre-filter` fulfils our needs and can be used in deployment.

4.1.5 Tested Detection Methods

In Chapter 4.1.1 and Chapter 4.1.2 we proposed multiple methods, that we thought might be suitable for the detection of malicious domains in network traffic. After performing a comparative analysis of the described indicators, we came to the conclusion that they were not suitable for distinguishing phishing domains from benign ones. Our reasoning behind this was explained in the relevant chapters.

Our initially proposed detection methods were based on a simple premise of *weak* and *strong* indicators. The idea behind a *weak* indicator is that on its own, it cannot indicate a phishing site or malicious intent. When multiple are grouped together however, it may point to the potential malicious intent of a domain. *Strong* indicators should present a much higher level of confidence that a SNI was registered for phishing purposes. The model was also intended to aid us in the discovery of features that could be used to identify phishing domains. Based on our propositions and subsequent tests, we were forced to rule out most, if not all, previously mentioned indicators.

After discarding multiple of our first proposals, we came forth with two indicators that form the current detection stage. While the system is currently limited, we would do well to remind ourselves that further research is required to build a fully-fledged detection engine. Returning to the two indicators, the first is discussed in Chapter 4.1.5.1 and is based on our discussion in Chapter 1.1.3 where we discuss providers of free TLS certificates. The main idea lies in the collection and filtering of *Certificate Transparency* logs. The second discussed indicator uses *Passive DNS* to perform checks if the observed domain is new in the network or not. Our decision to use the results of *Passive DNS* collection were again based on our practical experience with phishing.

4.1.5.1 Certificate Transparency Log Collection

Certificate Transparency (CT) Logs are a system meant to allow review and monitoring of TLS certificates issued by Certificate Authorities. In *Threat Monitoring* circles, these logs are also used to monitor domains that include specific keywords and could be targeting a monitored company. The information included in CT Logs encompasses all the information that is in a TLS certificate, meaning that we know the issuing date, validity length, start of validity and SNI for which the certificate had been issued. Issuers also keep a historical collection of these logs, allowing for backward searches. A more complete account of what Certificate Transparency Logs include and how they work may be found in Chapter 1.1.4.

Looking back at the information discussed in Chapter 3 where we dove into local phishing trends based on our experience in the space, we may find a use for CT Logs as a method of detection. As mentioned previously, creators of phishing are often driven by the prospect of financial gain. When taking this into account it is reasonable to assume that they would want to lower their operating costs to a minimum, making free services like *Let's Encrypt* a perfect option. This idea may be utilized with respect to Certificate Transparency Logs, which we collect, process and store. If a domain then passes through the initial filter, it is checked against our gathered database and if it is included we forward it down the pipeline.

This setup of directly checking against our collected database is not the preferred method, because of two reasons. First, certificates with wildcards in the Common Name (CN) will be harder to detect in a direct query. If a certificate includes the domain name `*.google.com` then all subdomains of `google.com` should be matched, this would however not be possible with a direct query checking for the presence of `translate.google.com`. The second reason lies, yet again, in the question of speed. Having to either batch the received domain names into groups

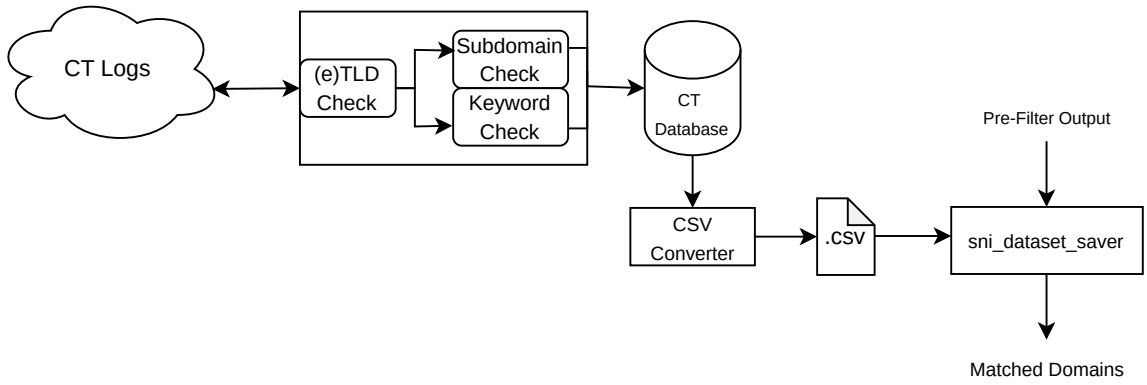


Figure 4.9 Design of the Certificate Transparency Log collector module

to extract from the database, or send a query for each SNI individually we would end up with a high-latency system, dependent on the processing speed of the used database.

Another issue lies in the number of certificates to store. It is not feasible to keep a complete copy of the *Certificate Transparency* tree locally, due to the high volumes of issued certificates. For example, Let's Encrypt issues anywhere between 2 to 3 *million* certificates per day, as of 2023 [7]. This is why we decided for a system that collects information on certificates, that only include desired keywords. An illustration of the *CT Log* filtering module, may be seen in Figure 4.9.

Each day the collector runs, keeping a database entry of the last *tree size* that had been checked, to use as the starting index of the next run. After iterating through all the data received, the system filters based on registered domain names, by using a smaller version of the *pre-filtering* module, as discussed in Chapter 4.1.4.6. The smaller filter only matches for keywords and subdomain hosting. When the filtration is finished, all passed domain names are committed to a database. This database of collected SNIs is then converted into a CSV file, that is used by the *sni_dataset_saver* NEMEA module [8], for the purposes of matching incoming domain names with those in the CSV. The matcher module uses a *trie* structure to perform the searches.

It is also important to note, that the *Keyword Check* section of the CT Log collector module uses a separate curated wordlist, from the one used in the `pre-filter` described in Chapter 4.1.4.6.

Our tested implementation of this module received and processed data from the *Let's Encrypt* Certificate Transparency log only, because of the reasons outlined before. Extending the module to work with multiple Certificate Authority (CA) logs is possible, and encouraged for future research. The database domain table had been created with an *issuer* column, as well as the table holding information on the last index visited. Thus extending the module should be a matter of creating configuration files, error handling and slight rewrites of the main code.

4.1.5.2 Passive DNS API

CESNET hosts a public DNS server, that users are allowed to use. Internally a Passive DNS API collector had been created, to monitor if a domain name had been seen before on the network. The response of the API contains some basic information about the domain, when it was first seen and when it was last seen.

We decided that this information could be used as an indicator of phishing, for domains

[7] https://letsencrypt.org/stats/
[8] https://github.com/CESNET/Nemea-Modules/tree/master/sni_dataset_saver

Code listing 4.2 Batching logic of Passive DNS Module

```
if len(buffer) >= 10 or (time.time() - start) >= 10:
    pd_buffer = pd.DataFrame(buffer)
    pd_buffer.insert(1,'Status',404)
    pd_buffer.drop_duplicates('TLS_SNI',inplace=True)
    pd_buffer.reset_index(inplace=True)

    asyncio.run(main_worker(pd_buffer))

    pd_buffer = pd_buffer[pd_buffer.Status != 200]

    pd_buffer = pd_buffer.loc[:,["TLS_SNI"]]
    caught_js = pd_buffer.to_json(index=False, orient='table')
    caught_js = json.loads(caught_js)['data']

    for data in caught_js:
        try:
            for data in caught_js:
                trap.send(bytearray(json.dumps(data), "utf-8"))
        except Exception as e:
            print(e)

    buffer = []
    start = time.time()
```

that have passed through our filters up-to this point. If a domain name passed through our `pre-filter`, then it is probably using an un-common *(e)TLD*, is not common on the network and is either hosted as a subdomain or includes specific keywords. The domain name must have also passed through our *CT Log* filtering module, which indicates that the TLS certificate had been issued by a monitored Certificate Authority. In the case of our testing, the monitored CA was *Let's Encrypt*.

After a domain name passes through all the described filters, we send a request to the *Passive DNS API* located at `passivedns.cesnet.cz`, using a valid API key. The HTTP status code of the server's response indicates, if the domain name is new to the network or not. An *HTTP 404 Not Found* response shows, that the SNI has not been observed on the network or requested through the DNS server. If this is the case, we may then put together all collected information up to now and presume that a new wave of phishing attacks is occurring.

In this case we may look back at how we described the workings of phishing campaigns in Chapter 3, where we state that the creators of these campaigns often make them in waves. Meaning that a mass registration of domain names occurs, which are then filled with content, back-end servers and data collectors are prepared (and possibly tested) and then these sites are disseminated en-mass to the public.

For us to be able to detect these domains, we created a NEMEA module that receives data from the previous stage in *JSON* form and sends asynchronous HTTP requests to the Passive DNS API. These requests are done in batches of 300 or if no new batch has been filled in the last 10 seconds, which can be seen in Listings 4.1.5.2. After fulfilling one of the two conditions, the collected domains are converted into a *Pandas DataFrame* and sent for asynchronous processing in the *main_worker* function.

After completion of the asynchronous run, we filter out all previously seen domains, which can be seen in the check *pd_buffer[pd_buffer.Status != 200]*. We also then ensure, that only the expected fields are kept in the *DataFrame* and any duplicate occurrences of SNIs are dropped by the *drop_duplicates* function. The collected information is then converted to *JSON* format and

sent to the output set when running the program.

One of the advantages of this module is, that it may be used with other Passive DNS collecting APIs, other than the one deployed by CESNET. If a user would want to deploy a local instance of a Passive DNS observer, they only need to ensure that the API returns a *404* HTTP status code on unseen domains and a *200* HTTP status code on seen domains. Utilizing the module after this is a simple case of slightly modifying the used module.

4.2 Encountered Issues

When performing brief test runs of the individual modules, we encountered some unexpected issues that were either not documented in the referenced material, or that we simply did not foresee. This chapter is dedicated to describing these issues and how we went about solving them.

4.2.1 Strict Pre-Filter

After implementing the `pre-filter` module as described in Chapter 4.1.4, we performed testing of the filtered data against our CT Log matcher described in Chapter 4.1.5.1. Our aim was to see if the design of the initial filtering module was not too strict or restrictive, and allowed later modules to receive sufficient amounts of interesting data.

The conducted tests resulted in no interesting domains being let through the Certificate Transparency matcher, meaning that the initial filtering phase does not let enough interesting domains through. A simple solution would be to increase the amount of keywords matched by the first filter, or to make them more general. This would however significantly slow down the initial module, due to it being implemented in the python programming language.

It was for the reasons of speed and ease of implementation that we decided to extend the keywords matched by the *CT Log* filter and use it instead as the front-line filter. With the usage of the *sni_dataset_saver* module, increasing the set of matched keywords with more general words would not slow down the module, thanks to it being implemented in the `C++` language.

Additionally, we could be sure that in the current implementation the filter would still be able to efficiently filter out most of the received data. This is down to the fact, that only certificates from the Let's Encrypt Certification Authority were processed. Like this the resulting system works like a funnel, with the first phase receiving large volumes of data with the volume being quickly reduced in each additional stage.

4.2.2 Poorly Documented Let's Encrypt CT Log

During implementation and testing of the Certificate Transparency Log collecting module, we focused on the transparency log of *Let's Encrypt*. We found that their *CT Log* was poorly documented and, specifically the API. In this chapter we address the two issues that we identified during testing as a result of poor documentation. In Chapter 4.2.2.1 we discuss the undocumented ratelimit present in the API. After, in Chapter 4.2.2.2 we discuss the discovery of certificate information issued by CA's other than Let's Encrypt, in the results of the Let's Encrypt *CT Log* API.

4.2.2.1 CT Log Ratelimiting

When we were creating the module responsible for the collection and processing of TLS certificate information included in the Certificate Transparency logs of Let's Encrypt, we were testing different batch sizes of information. As described in Chapter 1.1.4, the API outlined in RFC-6962 [22] allows the user to specify a starting index in the certificate tree and an end index.

Using the start and end index, we decided to iterate over the tree in larger batches, in order to process data in larger chunks and limit our amount of API requests. Originally, we used batches of over 100000 domains, considering the fact that Let's Encrypt issues about $2 - 3$ *million* certificates each day. In the documentation of the API, we looked for information on request ratelimits to avoid a potential blocklisting of the system's IP address and found none.

We then tested if our collector correctly receives the data and is able to process it. The processing of data mainly entails parsing out the SNI and passing it to the internal filtering functions of the module. Here we found no issue.

What we failed to identify was the fact, that if the amount of data requested is too large, the API does not return all requested data. In-fact, we discovered that a hard upper-limit of 246 domains is set, and if the amount of desired information exceeds this upper-limit, the API simply returns 246 entries. This information was not included in any of the checked documentation [9] [10] and was discovered after checking data that we collected over several weeks.

Moreover, the initial proposal for Certificate Transparency Logs and their APIs in RFC-6962 [22] states the following on ratelimits:

> Logs MAY restrict the number of entries that can be retrieved per "get-entries" request. If a client requests more than the permitted number of entries, the log SHALL return the maximum number of entries permissible. These entries SHALL be sequential beginning with the entry specified by "start". [22]

While the Let's Encrypt API does not violate the RFC, and is in-line with the specified behaviour the maximum limit of received certificates is nowhere to be found. After correcting our mistake and figuring out the upper-limit by trial and error, we could resume our collection of data.

4.2.2.2 Certificates of Other CA's

Our expectation when deciding to use the *Let's Encrypt* Certificate Transparency Log was that the output would include information on certificates from *Let's Encrypt* only. This was a presumption that we based on the lack of information about other Certificate Authorities in the documentation, and our own assumptions.

After starting a test deployment of the *CT Log* collector module only, we performed periodic checks of the accumulated data. By having a decent overview of the data that passes through the CSV matcher, we hoped to be able to spot potential bugs in the form of domains that should not have been let through. Another possibility could have been, that the specified lists of keywords, (e)TLDs and subdomain hosting services would be too restrictive and so low amounts of data would be collected. We also wanted to ensure that sufficient amounts of data were being accumulated, to properly saturate the database.

During these periodic reviews we started to notice unexpected domain names in the output. However, the domain names were not unexpected because of the lack of keywords or content matching our specified filter. Instead the SNIs came from certificates that had been registered under Certificate Authorities other than Let's Encrypt. This was when we realized that the Certificate Transparency Logs operated by Let's Encrypt are not strictly limited to Let's Encrypt certificates. This was a slight mishap on our part, and was not found in any way in the CT Log documentation.

Since our focus in this research phase of the module was to collect information from Let's Encrypt we decided to limit the processed certificates to those issued by Let's Encrypt only. This was done by performing a simple check, as can be seen in Listings 4.2.2.2. The shown example loads information about the certificate's subject, issuer, issuing date and date of expiry. It then performs a check to see if the subject has a Canonical Name (CN) and if the Certificate

[9] https://letsencrypt.org/docs/ct-logs/
[10] https://letsencrypt.org/2019/11/20/how-le-runs-ct-logs.html

Code listing 4.3 CT Log module check for Let's Encrypt issuer

```
if leaf_cert.LogEntryType == "X509LogEntryType":
    data_string = Certificate.parse(leaf_cert.Entry).CertData
    subj = crypto.load_certificate(
        crypto.FILETYPE_ASN1,data_string
        ).get_subject()
    issuer = crypto.load_certificate(
        crypto.FILETYPE_ASN1,data_string
        ).get_issuer()

    date = convert_time(str(
        crypto.load_certificate(
            crypto.FILETYPE_ASN1,data_string
            ).get_notBefore()
        ))
    dateAfter = convert_time(str(
    crypto.load_certificate(
        crypto.FILETYPE_ASN1,data_string
            ).get_notAfter()
        ))

    if subj.CN and issuer.organizationName == "Let's␣Encrypt":
        domains.append(
            {
                'TLS_SNI':subj.CN,
                'TIME_SEEN':int(datetime.datetime.now().timestamp()),
                'TLS_VALIDITY_NOTBEFORE':date
            }
        )
```

Authority is indeed Let's Encrypt. This example is for an *X509* certificate data structure, but the *CT Log* tree may also include extra data structures. Handling and parsing of the *other* certificate structure is handled in the *else* branch with very close parsing.

4.2.3 Official Domains in CT logs

As mentioned in Chapter 4.2.2.2 we maintained a steady overview of the information stored by our *CT Log* accumulator module. During these routine checks we noticed that some brands and institutions used Let's Encrypt certificates either for testing purposes or for official use. We found their domains, because our keyword matcher was looking for words targeting these brands or institutions.

This was a surprise to us, as we did not expect official institutions to utilize the services of Let's Encrypt, even for testing purposes. Looking through the database we found multiple instances like these, and decided that a blocklisting feature was required.

Blocklisting in the *CT Log* collector module is done via a configuration file, where users directly input regular expressions of domain names to discard from the received data. This information is stored in a database that is initialized or modified based on the configuration file.

Altering the design of the module was straight-forward. If we look back at Figure 4.9 we can see a clear cut-off in filtering before committing data to the database. We simply added a second database and a quick matching check against the filtered data to see if undesired domains passed through. Matched SNIs are discarded and not committed to the database.

The reason why we are not concerned with database speeds as much in the *CT Log* collection module lies in the very concept of it. Again, looking to Figure 4.9 we can see that the matching of flow data or *pre-filter output* is done by the *sni_dataset_saver* module. The dataset saver module does not interact with the databases in any way. Instead the collector is periodically run, to ensure up-to-date information is being stored, and so lower processing speeds than in the detection pipeline are acceptable.

4.3 Redesigned System

After reviewing all the encountered issues as described in Chapter 4.2, we decided to perform a redesign of the system. The initial and working implementation was a combination of the previously described `pre-filtering` module, the *CT Log* collection and matching module and the *Passive DNS* requester module. In this system, the detection is done by the two later modules, relying on matching of domain names in our collected Certificate Transparency database and subsequent checking if the matched domain name has been seen before by the CESNET DNS.

We have mentioned before, that one of the issues encountered in the system was that the initial filtering module was too strict. The filter is itself made-up of four separate functions, that pass the resulting data in a sequential chain between themselves. First a check occurs, to see if the SNI is present on our list of common benign domains, indicating that it will be of no interest. The second check looks at the *(e)TLD* of the site, checking it against commonly used phishing *(e)TLDs*. Now the domains that pass through these two functions are given to a function that performs two checks, which are paired in a logical *OR*. Checking for specified keywords or if a site is using a subdomain hosting service, based on a wordlist.

On the other hand, the module responsible for the collection of Certificate Transparency logs and subsequent matching against these collected logs is a lot simpler. While it does use a smaller version of the initial filter, the module does not perform any significant database work when interacting with live data. In-fact, the matching module used is a pre-built NEMEA module called the *sni_dataset_saver* which takes a CSV export of the database and then looks for matches in received data. The problem lies in the fact, that the received data is already heavily filtered and so the probability of matching interesting domains in the database is low.

A possible solution could be to widen the keyword list of the initial filter. This would however cause a significant increase in the size of the resulting streams, which would then forward data to the following module with data that is highly likely to not contain domains of interest.

Instead we opted for a solution, that switches the positions of the modules, placing the Certificate Transparency matcher as a front-line filter. Like this we have a higher chance of identifying interesting domains with TLS certificates registered under Let's Encrypt, as they won't be discarded by the first filter. The initial filtering module has instead been moved into second position, as it still looks for desirable properties in domain names and will be able to filter some benign domains that pass through the *CT Log* matcher. An illustration of the changed system can be seen Figure 4.10.

We did not only switch places of the two modules. For the keyword list of the *CT Log* module we expanded the list to more general words that would allow a higher throughput of domains. In order for us to perform efficient matching with the selected keywords, we had to re-saturate the database with a historical collection of data. We opted to collect certificate information on domains about a month old, banking on the fact that general phishing campaigns are not created too far in advance. This collection however, did take up precious time for testing deployments and debugging.

The system might not include many steps for detection, but we firmly believe that the issue of detecting phishing domains based on flow data only is a complex one, and further research is needed. Modules in the system may be expanded in the future and easily adapted to change their behaviour based on additional research and longer running tests.

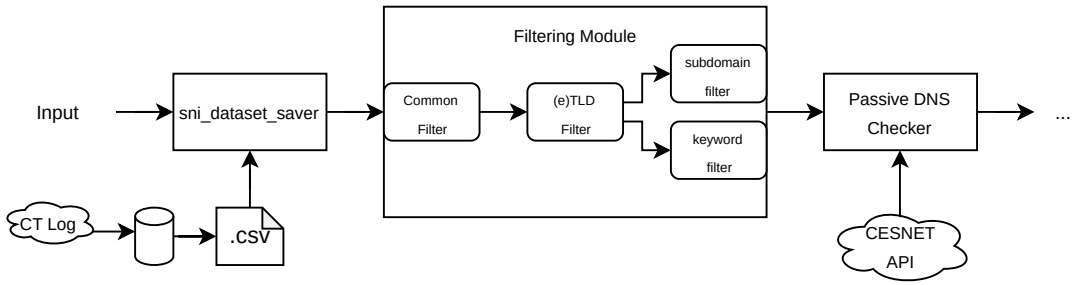


Figure 4.10 Diagram of the whole system post-redesign

Chapter 5
Testing Results

After our in-depth discussion of the design philosophy, testing and thought processes behind our implementations comes time to perform tests on real-world data to demonstrate how and if the system is capable of detecting phishing domains. It is our firm belief that performed tests should be done on a control set of data, where we can be sure of the format of phishing domains and that there is even a phishing domain present. However, we also believe that the control data should not be the entire basis of our conclusion, and so another test based on real data with high uncertainty should be done as well.

In-line with our reasoning we performed testing on two sets of data. Both of the tests were done using the redesigned system, as described in Chapter 4.3. We consider these tests to be the final test results of this thesis, and will base our conclusions on them. Our descriptions of these tests, the results and performance are described in separate chapters. The first tests of control data are written about in Chapter 5.1. On the other hand, descriptions and results of the testing of data directly from the CESNET network are included in Chapter 5.2.

5.1 Control Data

When we moved on to the testing and evaluation of our designed system, we decided that a set of control data would be required. It was necessary for the data to include one or more phishing domains, for which we could be certain of their malicious intent. Without the inclusion of phishing in the data, we could not perform a proper evaluation of the system. The need for absolute certainty of the presence of phishing domains in the captured traffic came from the fact, that monitoring traffic on the CESNET network we could not be sure that a phishing attack would occur during our testing. Like this we have at-least one set of data, for which we can decide if the system is capable of `True Positive` phishing detections and evaluate possible `False Positives`.

5.1.1 Collection of Control Data

As mentioned before, we needed to be sure that the inspected data would include confirmed phishing domains. This is a challenge in its own right, as we first need to find and confirm the phishing sites and then come up with a way to monitor and collect the data. We opted to test the system on aggregated data from a local network, where we performed emulated regular user behaviour by briefly browsing the web. We interlaced the browsing with visits to four phishing sites, which we confirmed made their way into the resulting captures.

Capturing of traffic was done using the WireShark traffic capture program, and exported

SNI	CT Log	4-Filter	Passive DNS	Detected
`dhl-colis-support.com`	✓	✓	✓	✓
`ceska-posta.mailservice-delivery.com`	✓	✓	✓	✓
`dhl-delivery-contact.com`	✓	✓	X	X
`dpdservice.uk`	X	X	X	X

Table 5.1 Detection Status of Control Phishing in Each Stage

into a *.pcap* file. Our capture was limited to TLS traffic only, as all other traffic is of no use for the system. The raw *.pcap* files had to be converted into files in the binary *unirec* format, as expected by processing modules. This was accomplished using the *ipfixprobe* [1] program, which allowed us to convert a packet capture file into the desired *unirec* format.

After conversion we checked the integrity of exported files, to ensure that a corrupted file was not generated and that the results did meet our format expectations. This was done using one of the pre-built NEMEA modules, called *logger* [2] which allowed us to view the contents of the unirec files in text format. Upon confirming that the format did in-fact meet our expectation, we could proceed with local testing.

5.1.2 Local Tests on Control Data

A local evaluation of the system was performed in order to have finer control and insight into the whole detection process. We also did not make use of the NEMEA Supervisor, which essentially deploys and manages each of the modules. Instead we tested each module individually, but in the sequence in which they would be deployed. Like this we took the role of the NEMEA Supervisor and had finer control over each of the modules, allowing us to better inspect the behaviour, output data and metrics.

Initially we performed testing of the Certificate Transparency matching module, the output of this module was then fed to the four-part Filtering Module and finally to the Passive DNS checker. We expect the results of this testing to be the four phishing domains, we simulated the user visiting.

Importantly, one of the phishing domains did not have a Let's Encrypt certificate and as expected it did make it did not pass through the *CT Log* filter. Three of the four phishing domains passed, while the remaining traffic was fully filtered out. With only three domains to process, we did not measure performance of each individual module, as it would not be relevant to real world applications.

After passing through the first filter, we used the resulting data as input for the 4-part Filtering module, which were all let through again. And finally these domains were passed to the Passive DNS checker, which filtered-out all passed in domains. The results of our local testing may be seen in Table 5.1.

The phishing domain of `dpdservice.uk` did not pass the first Certificate Transparency filter, because the used TLS certificate was not issued by Let's Encrypt. We picked this domain on purpose, to demonstrate this limitation of the current system. In addition, tow other domains passed all stages and one domain was discarded by the final Passive DNS check.

Discarding of one of the domains was due to us having to rely on user-identified phishing domains, to supply into our testing data. Thus we could not guarantee their age and if they had previously been seen by the Passive DNS system. We would also like to highlight, that during the testing run on control data, there were no `False Positive` identifications of phishing domains.

[1] https://github.com/CESNET/ipfixprobe
[2] https://github.com/CESNET/Nemea-Modules

Filter Stage	Input Size	Output Size	Filtered Percentage
`sni_dataset_saver`	-	887	-
`4-Part Filter`	887	125	85.91%
`Passive DNS`	125	31	75.2%

Table 5.2 Amount of CESNET Data Filtered in Each Stage

5.2 CESNET Data

After completing our tests on the control data, in which had a 100% certainty for the inclusion of phishing in captured traffic, we moved-on to a short capture on the CESNET network. This capture was of live data on the monitored network, and we were not sure if it included active phishing domains.

The capture included domains that had passed through the deployed *sni_dataset_saver* filter, which relies on the data from our Certificate Transparency database. Thus, the capture is not large enough to demonstrate performance of the system and overall metrics. It does however serve the purpose of demonstrating effectiveness of our system on real data.

We believe that this demonstration is often missing in other literature, which we reviewed in Chapter 1.5. It is our belief that this evaluation on data, that does not guarantee results and instead reflects the traffic that the system would encounter in a production deployment is most fruitful for future research.

5.2.1 Collection of CESNET Data

Data collected on the monitored CESNET network was aggregated by Flow Data collectors, that are part of the general Flow Monitoring infrastructure. This data was then forwarded to the deployed Certificate Transparency matching module. Server Name Identifiers that passed through this filter were stored in an output file, and downloaded locally.

The resulting sample included around 880 unique domain names. For the 4-step Filter to be able to work with this sample, we had to convert it into the binary unirec format. A simple *python* script utilizing the *pytrap* library managed to quickly modify the output.

5.2.2 Local Tests on CESNET Data

Even with the data exported from the CESNET network we decided to stick with locally managed testing, to keep the previously mentioned inspection capabilities. Observing, debugging and analyzing inspected domain names on live deployments would not be possible, due to the large volumes of data processed.

Unlike with the tests of Control Data in Chapter 5.1 we only had two stages to test locally. We also had no prior knowledge of included phishing domains, so the evaluation of `False Negative` detections was not possible. Additionally, for us to be able to check the output for `True Positive` detections, we had to rely on public resources like `urlscan.io` or manually checking the domains.

The total amount of unique domain names in the set added-up to 887, which we converted and passed along to the 4-part Filtering module. We expected this followup module to cut-down the data even more, especially when we consider that it filters out common domains on the network and focuses on subdomain hosting, keywords and (e)TLDs common in phishing. Our expectations were met, with the filter allowing only 125 domains through, thus filtering out over 80% of received data.

Checks against the Passive DNS API are time sensitive, and require fresh data and domains.

This is why we expected a high percentage of discarded domains. Our expectations were met again, with the module discarding over 70% of received data. The results and filtering-rate of each module may be found in Table 5.2. After checking the results of the final stage, we identified one suspicious domain: `dhltrackyourrordernow.com`. The domain was sadly inactive and we found no evidence of previous phishing activity on public search sites.

5.3 System Metrics

The evaluations of the system's effectiveness conduct in Chapter 5.2 and Chapter 5.1 was done on data that was not sufficiently large for an evaluation of the system's performance metrics. For this reason, we exported another capture from the CESNET Network, to perform another evaluation of each of the modules. The data has a total size of 1026049 flows, which were collected over 75.640 seconds.

Metrics that we evaluated were Execution Time in Chapter 5.3.1, Memory Consumption in Chapter 5.3.2 and CPU Consumption in Chapter 5.3.3. Measuring of this data was done on each module separately, to more clearly illustrate the strengths and the weaknesses of the system. Moreover, the speed of the system is mostly dependent on the first module and how much data it is capable of filtering.

We find it important to point out, that the *sni_dataset_saver* module which we use as the first filtering module is built using `C++`. Utilizing a language like `C++`, with processing speeds comparatively faster than the remaining `Python` modules results in the speed of the system being mostly dependent on the first module. Considering that the initial module also filters a large portion of data, the processing requirements for the remaining modules are less strict.

5.3.1 Execution Time

To begin with our measurement of total execution time, we started off with the Certificate Log matching module, the *sni_dataset_saver*. In the case of this module, it is meant to be run continuously, and so the start of a run has a longer startup. This is due to the program having to first load the *csv* file containing data to filter by, and then construct a *trie* structure out of this data.

It is for this reason that we made a small addition to the source code of the *datasaver* module, and only measured total execution time starting right before the main work loop and stopping after exiting it. The results of this measurement was a total elapsed time of $1037575\mu s$ or $1.037575s$ for a total of 1026049 records. This is a speed of about 1 million domains per second. For an even more staggering number the total number of output domains was 416, which is about 0.04% of the total received data.

From here we can clearly see, that the speed of the system is determined mostly by the first module. Thanks to the efficient implementation in a fast language like `C++` and good usage of data structures, the module is capable of achieving high-speed processing.

The next module is the 4-Stage Filter module, which was built using *python* and the *pytrap* library. Keeping in mind, that the volume of received data is bound to low, especially when compared to the volumes processed by the first module, the filter did well. After receiving 416 records, it filtered them down to 303 records in $0.86s$. This processing speed is fast enough to keep-up with the first module, and so we find it acceptable.

Lastly, we tested the Passive DNS requesting module. We expect this module to have higher latency due to it having to rely on response speeds of the CESNET API endpoint. The module also lacks a ratelimiting system, which should be added in followup testing and research. With the total input for the Passive DNS requester being 303 records (not unique domains), the module took a total of $8.810s$ to process. This is a speed of about 34 domains per second, with 0 domains passing this stage.

Module	Input Size	Output Size	Elapsed Time
`sni_dataset_saver`	1026049	416	1.037575s
`4-Part Filter`	416	303	0.86s
`Passive DNS`	303	0	8.810s
`CT Log Collector`	2569713	1819693	19884.4s

Table 5.3 Processing Speeds of Each System Stage

Additionally, we performed a measurement of the actual Certificate Transparency collecting module. We stress that this module is not part of the detection pipeline, and so fast execution time is not a priority. The only requirement is that the module is fast enough to complete execution, before the next automated run. We confirm that this is indeed the case. The collector processed a total of 2569713 certificates in 19884.4s or 5 hours. Since the collector is run once a day, this speed is sufficient for our needs.

The resulting speeds along with the total size of input and output data for each stage is included in Table 5.3. Based on the table, we may conclude that the system satisfies our set-out goal of being able to process at-least 10000 per second. Our conclusion that the system satisfies our speed requirements is based on the fact that the total time to collect about 1 million domains was 75s. If we then add-up all execution times included in Table 5.3 we arrive at a total processing time of roughly 10.7s, which is vastly lower than the time to collect 1 million records.

5.3.2 Memory Consumption

When considering the performance of complex systems, memory often becomes an issue. We expect memory usage to be high in most of the modules, due to having to keep domain names and other flow information loaded in memory. Our testing was again chronological, utilizing the same dataset as in Chapter 5.3.1.

Starting off with the *sni_dataset_saver* module, we expected high memory usage due to the creation of a complex *trie* structure in memory and the subsequent loading of flow data. For the measurements, we used the *valgrind* [3] memory checking tool to check the highest total amount of memory taken-up by the module. The results of this test were a total of 2.041GB allocated on the heap. Our prediction of large memory consumption was correct, this however should not be a significant issue on modern servers.

We would also expect, that the highest memory usage will be in the first module, as it processed the highest volumes of data. Modules following the *sni_dataset_saver* module have the benefit of having to process comparatively very low amounts of data. The 4-Part Filter was processing a total of 416 domains for which it allocated a maximum of 0.4262GB of memory. This is an acceptable number, considering the usage of the *python* programming language with which we have limited control over memory allocations.

As for the CT Log collection module, we expected it to have a large consumption of memory. This is down to two reasons; the first is simply the vast volumes of data processed in each run, with at-least a million certificates to process each day. Secondly, the collector was implemented in the python language, due to processing speeds not being our main concern with this module. In our test, the collector processed a total of 2569713 certificates while taking up only 0.59GB of memory. This number is much lower than the expected number, and a good sign for the module.

The last measured module is the Passive DNS API checking module, which had an even lower amount of input data to process. A total of 303 domains to evaluate. Based on this we expected the memory usage not to be too high, and this was confirmed. The module used a maximum

[3] https://valgrind.org/

Filter Stage	Input Size	Output Size	Memory Usage
`sni_dataset_saver`	1026049	416	$2.041GB$
`4-Part Filter`	416	303	$0.4262GB$
`Passive DNS`	303	0	$0.076GB$
`CT Log Collector`	2569713	1819693	$0.59GB$

Table 5.4 Total Memory Usage of Each System Stage

of $0.076GB$ of memory. Measuring of memory consumption with the python modules was done using the standard *resource* library, which implements the standard *getrusage* [4] function.

All results may be found organized in Table 5.4 which includes the filtering stage referenced, the total size of input data for the stage, the total size of output data for the stage and then the total memory usage in *GigaBytes (GB)*. Adding up the memory usage of each stage, excluding the CT Log collector, we get a resulting usage of $2.543GB$.

5.3.3 CPU Usage

For this section we opted for a crude measurement in the form of observing the output as provided by the *htop* program. We performed tests of each module individually and in the expected sequence, chaining the output data as inputs for the following module.

Much like with the metrics of memory usage in Chapter 5.3.2, we expect the first module to have the highest CPU utilization, simply due to the number of domains to process and the initial *trie* construction. After running the module and observing the output of *htop*, we noted the maximum CPU consumption of roughly 40%. This number aligns with our prediction, as the module was processing a total of 1026049 flows.

The 4-Part Filter was expected to have a much lower CPU consumption percentage, than the Certificate Transparency matching module. With the total amount of domains to process being at a low 416, our expectations were instead subverted with the highest recorded CPU consumption resting at 66.7%. Such high consumption was however short lived, as the module finished processing about as fast as it started. Taking into account the process startup and the use of *python* might explain the momentary spike.

Lastly the Passive DNS API checker, which processed 303 domains. With this module we expected higher CPU usage, due to the numerous HTTP connections that had to be established and managed. Our expectations were somewhat confirmed, but we were again surprised with the CPU percentage spiking to about 70.5%. We again attributed this to costly startup operations and heavy network communications.

We did not include the Certificate Transparency collecting module measurements in this chapter for the simple reason that all the operations are performed on offline data. Meaning that the module is not explicitly part of the detection pipeline. As mentioned in Chapter 4.3, the detection pipeline relies on the exported data of the collected database. It is therefore acceptable for the collector to have higher CPU consumption than the rest of the modules, in-fact it may even be deployed on a different server with the export being transferred to the *sni_dataset_saver* module.

All CPU consumption results can be found in Table 5.5, where we included the filtering stage, total size of input data for the stage, total size of output data for the stage and the highest CPU consumption percentage seen in *htop*.

[4] https://manpages.debian.org/bookworm/manpages-dev/getrusage.2.en.html

Filter Stage	Input Size	Output Size	Maximum CPU Usage
`sni_dataset_saver`	1026049	416	40%
`4-Part Filter`	416	303	66.7%
`Passive DNS`	303	0	70.5%

Table 5.5 CPU Consumption Percentage of Each System Stage

Chapter 6
Conclusion

In this thesis we put forth a working concept of a system, that is capable of detecting some phishing domains based purely on collected TLS metadata in captured Network Flows. The system was evaluated on a set of control data, where we had absolute certainty about the presence of phishing domains and fine control over the dataset as a whole. Using this control data in Chapter 5.1 we were able to demonstrate that the system is capable of detecting phishing domains in network traffic, but also where the limitations of the current system lie.

We also tested our design against a set of data collected on the CESNET network, which was meant to provide a snapshot of production-level traffic. This test was conducted in Chapter 5.2, where we had no certainty about the presence of phishing traffic. The results of this test were promising however, as the output featured a potentially malicious domain, that was inactive at time of testing.

Moreover, we argue that our work deviates from other related works by performing tests on production-grade traffic data. In the literature that we reviewed it was often the case, that the system designs were based on commonly visited domain names. This kind of data however does not reflect the vast majority of domains that are present in regular internet traffic. Furthermore, the systems are often evaluated on the same data that was used during their initial design phases. This can lead to the creation of systems that are capable of very specific detections only.

Our evaluation of the system's overall performance metrics was conducted in Chapter 5.3, where we utilized another set of flow records collected on the CESNET network. Metrics were measured for each module individually, with the specific monitored metrics being: execution time (Chapter 5.3.1), memory consumption (Chapter 5.3.2) and CPU usage (Chapter 5.3.3). A part of these measurements was to demonstrate that the system does indeed fulfil our set out goal, of processing 10 thousand domains per second.

The output of our work may be the early concept of a system for phishing detection in high-speed networks, which certainly needs further research and followup work. But it is not the only contribution of this work. We also performed tests of potential indicators of a phishing domain, which included Shannon's Entropy (Chapter 4.1.4.2), Coleman-Liau Readability Index (Chapter 4.1.4.3) and Automated Readability Index (Chapter 4.1.4.4). Results of these tests showed, that when taking into account the domain names that are part of general network traffic, it is difficult to pick out statistical features that clearly differentiate the phishing domains from benign ones. More specifically we demonstrated that Shannon's Entropy, Coleman-Liau Readability Index and the Automated Readability Index are not fit for use as phishing indicators in high-speed network traffic.

We also included our initial system designs, and thoughts. Our reasoning is an important part of the result as a whole, and we wanted to demonstrate the thought process behind it. More specifically, we also wished to demonstrate why certain detection methods or possible indicators

were initially considered and later discarded without the need for testing.

Apart from demonstrating clear thought and work put into thesis, we also included a section on the issues encountered with our design. In Chapter 4.2 we recounted some problematic behaviour of our modules, and errors that were caused by our design decisions. To help better understand our thought process, we provided possible solutions and the solution that we ended up with.

Another contribution of this work is in the brief analysis of phishing trends in the Czech Republic, based on our experience with active discovery and removal of phishing domains. As part of this section we describe the often targeted brands and common practice with phishing campaigns in these sectors.

Overall we believe that our thesis provides contributions into the field of phishing detection, without the usage of phishing kit signatures or machine learning. While the system may not be complete, we consider it a stepping stone for the network monitoring research teams that should be expanded upon. Additionally, we also believe that our brief analysis of phishing trends in the Czech Republic and testing of detection metrics that are unfit for use in high-speed networks further our contributions in the space of phishing detection, and should be able to serve as the basis of further research in the detection of phishing domain names.

Bibliography

1. EUROSTAT. *Internet access and use statistics-households and individuals*. 2016. Available also from: https://ec.europa.eu/eurostat/statistics-explained/index.php?oldid=379591.
2. TANKOVSKA, H. Active social media penetration in selected European countries in 2020. *Statista*. 2021. Available also from: https://www.statista.com/statistics/295660/active-social-media-penetration-in-european-countries/.
3. ENISA. Threat landscape 2022. *ENISA, Oct*. 2022. ISBN 978-92-9204-588-3. Available from DOI: 10.2824/764318.
4. INVESTIGATION (FBI), Federal Bureau of. 2020 IC3 Annual Report. 2020. Available also from: https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.
5. APWG. Phishing Activity Trends Report Q3/2022. 2022. Available also from: http://www.apwg.org/reports/apwg%5C_report%5C_Q3%5C_2022.pdf.
6. MICROSOFT. *Microsoft Digital Defense Report*. Microsoft Redmond, 2021. Available also from: https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RWMFIi.
7. ALLEN, Christopher; DIERKS, Tim. *The TLS Protocol Version 1.0* [RFC 2246]. RFC Editor, 1999. Request for Comments, no. 2246. Available from DOI: 10.17487/RFC2246.
8. 3RD, Donald E. Eastlake. *Transport Layer Security (TLS) Extensions: Extension Definitions* [RFC 6066]. RFC Editor, 2011. Request for Comments, no. 6066. Available from DOI: 10.17487/RFC6066.
9. ELGAMAL, Dr. Taher; HICKMAN, Kipp E.B. *The SSL Protocol*. Internet Engineering Task Force, 1995-04. Internet-Draft, draft-hickman-netscape-ssl-00. Internet Engineering Task Force. Available also from: https://datatracker.ietf.org/doc/draft-hickman-netscape-ssl/00/. Work in Progress.
10. RESCORLA, Eric; OKU, Kazuho; SULLIVAN, Nick; WOOD, Christopher A. *TLS Encrypted Client Hello*. Internet Engineering Task Force, 2022-10. Internet-Draft, draft-ietf-tls-esni-15. Internet Engineering Task Force. Available also from: https://datatracker.ietf.org/doc/draft-ietf-tls-esni/15/. Work in Progress.
11. POLK, Tim; TURNER, Sean. *Prohibiting Secure Sockets Layer (SSL) Version 2.0* [RFC 6176]. RFC Editor, 2011. Request for Comments, no. 6176. Available from DOI: 10.17487/RFC6176.
12. BARNES, Richard; THOMSON, Martin; PIRONTI, Alfredo; LANGLEY, Adam. *Deprecating Secure Sockets Layer Version 3.0* [RFC 7568]. RFC Editor, 2015. Request for Comments, no. 7568. Available from DOI: 10.17487/RFC7568.

13. MÖLLER, Bodo; DUONG, Thai; KOTOWICZ, Krzysztof. This POODLE bites: exploiting the SSL 3.0 fallback. *Security Advisory*. 2014, vol. 21, pp. 34–58. Available also from: https://www.openssl.org/~bodo/ssl-poodle.pdf.

14. RESCORLA, Eric. *The Transport Layer Security (TLS) Protocol Version 1.3* [RFC 8446]. RFC Editor, 2018. Request for Comments, no. 8446. Available from DOI: 10.17487/RFC8446.

15. BLAKE-WILSON, Simon; MIKKELSEN, Jan; NYSTROM, Magnus; HOPWOOD, David; WRIGHT, Tim. *Transport Layer Security (TLS) Extensions* [RFC 3546]. RFC Editor, 2003. Request for Comments, no. 3546. Available from DOI: 10.17487/RFC3546.

16. DIERKS, Tim; RESCORLA, Eric. *The Transport Layer Security (TLS) Protocol Version 1.1* [RFC 4346]. RFC Editor, 2006. Request for Comments, no. 4346. Available from DOI: 10.17487/RFC4346.

17. RESCORLA, Eric; DIERKS, Tim. *The Transport Layer Security (TLS) Protocol Version 1.2* [RFC 5246]. RFC Editor, 2008. Request for Comments, no. 5246. Available from DOI: 10.17487/RFC5246.

18. AAS, Josh; BARNES, Richard; CASE, Benton; DURUMERIC, Zakir; ECKERSLEY, Peter; FLORES-LÓPEZ, Alan; HALDERMAN, J. Alex; HOFFMAN-ANDREWS, Jacob; KASTEN, James; RESCORLA, Eric; SCHOEN, Seth; WARREN, Brad. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2473–2487. CCS '19. ISBN 9781450367479. Available from DOI: 10.1145/3319535.3363192.

19. ZEROSSL. *SSL Certificates - ZeroSSL* [online]. [visited on 2023-04-10]. Available from: https://zerossl.com/features/certificates/.

20. BUYPASS. *Buypass - Go SSL* [online]. [visited on 2023-04-10]. Available from: https://www.buypass.com/products/tls-ssl-certificates/go-ssl.

21. SSL.COM. *Free SSL - ssl.com* [online]. [visited on 2023-04-10]. Available from: https://www.ssl.com/certificates/free/.

22. LAURIE, Ben; LANGLEY, Adam; KASPER, Emilia. *Certificate Transparency* [RFC 6962]. RFC Editor, 2013. Request for Comments, no. 6962. Available from DOI: 10.17487/RFC6962.

23. LAURIE, Ben; LANGLEY, Adam; KASPER, Emilia; MESSERI, Eran; STRADLING, Rob. *Certificate Transparency Version 2.0* [RFC 9162]. RFC Editor, 2021. Request for Comments, no. 9162. Available from DOI: 10.17487/RFC9162.

24. NIELSEN, Henrik; MOGUL, Jeffrey; MASINTER, Larry M; FIELDING, Roy T.; GETTYS, Jim; LEACH, Paul J.; BERNERS-LEE, Tim. *Hypertext Transfer Protocol – HTTP/1.1* [RFC 2616]. RFC Editor, 1999. Request for Comments, no. 2616. Available from DOI: 10.17487/RFC2616.

25. BARTH, Adam. *The Web Origin Concept* [RFC 6454]. RFC Editor, 2011. Request for Comments, no. 6454. Available from DOI: 10.17487/RFC6454.

26. SPEROTTO, Anna; SCHAFFRATH, Gregor; SADRE, Ramin; MORARIU, Cristian; PRAS, Aiko; STILLER, Burkhard. An Overview of IP Flow-Based Intrusion Detection. *IEEE Communications Surveys & Tutorials*. 2010, vol. 12, no. 3, pp. 343–356. Available from DOI: 10.1109/SURV.2010.032210.00054.

27. CEJKA, Tomas; BARTOS, Vaclav; SVEPES, Marek; ROSA, Zdenek; KUBATOVA, Hana. NEMEA: A framework for network traffic analysis. In: *2016 12th International Conference on Network and Service Management (CNSM)*. 2016, pp. 195–201. Available from DOI: 10.1109/CNSM.2016.7818417.

28. AITKEN, Paul; CLAISE, Benoît; TRAMMELL, Brian. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information* [RFC 7011]. RFC Editor, 2013. Request for Comments, no. 7011. Available from DOI: 10.17487/RFC7011.

29. NETFLOW, Cisco. Netflow services solutions guide. *Cisco System, Inc., http://www. cisco. com/en/US/docs/ios/solutions_docs/netflow/nfwhite. html*. 2007.

30. CLAISE, Benoît. *Cisco Systems NetFlow Services Export Version 9* [RFC 3954]. RFC Editor, 2004. Request for Comments, no. 3954. Available from DOI: 10.17487/RFC3954.

31. HOFSTEDE, Rick; ČELEDA, Pavel; TRAMMELL, Brian; DRAGO, Idilio; SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. Flow Monitoring Explained: From Packet Capture to Data Analysis With NetFlow and IPFIX. *IEEE Communications Surveys & Tutorials*. 2014, vol. 16, no. 4, pp. 2037–2064. Available from DOI: 10.1109/COMST.2014.2321898.

32. REKOUCHE, Koceilah. Early phishing. *arXiv preprint arXiv:1106.4692*. 2011. Available from DOI: 10.48550/arXiv.1106.4692.

33. ALEROUD, Ahmed; ZHOU, Lina. Phishing environments, techniques, and countermeasures: A survey. *Computers & Security*. 2017, vol. 68, pp. 160–196. Available also from: https://www.sciencedirect.com/science/article/pii/S0167404817300810.

34. YEBOAH-BOATENG, Ezer Osei; AMANOR, Priscilla Mateko. Phishing, SMiShing & Vishing: an assessment of threats against mobile devices. *Journal of Emerging Trends in Computing and Information Sciences*. 2014, vol. 5, no. 4. Available also from: https://etarjome.com/storage/btn_uploaded/2020-09-12/1599891065_11216-etarjome%5C%20English.pdf.

35. NÚKIB. Zpráva o stavu kybernetické bezpečnosti České republiky za rok 2021. 2022, no. 8.

36. LIN, Yun; LIU, Ruofan; DIVAKARAN, Dinil Mon; NG, Jun Yang; CHAN, Qing Zhou; LU, Yiwen; SI, Yuxuan; ZHANG, Fan; DONG, Jin Song. Phishpedia: A Hybrid Deep Learning Based Approach to Visually Identify Phishing Webpages. In: *USENIX Security Symposium*. 2021, pp. 3793–3810. Available also from: https://www.usenix.org/system/files/sec21-lin.pdf.

37. OEST, Adam; SAFAEI, Yeganeh; ZHANG, Penghui; WARDMAN, Brad; TYERS, Kevin; SHOSHITAISHVILI, Yan; DOUPÉ, Adam; AHN, Gail-Joon. Phishtime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. 2020, pp. 379–396. Available also from: https://www.usenix.org/system/files/sec20-oest-phishtime.pdf.

38. HO, Grant; CIDON, Asaf; GAVISH, Lior; SCHWEIGHAUSER, Marco; PAXSON, Vern; SAVAGE, Stefan; VOELKER, Geoffrey M.; WAGNER, David. Detecting and Characterizing Lateral Phishing at Scale. In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, 2019, pp. 1273–1290. ISBN 978-1-939133-06-9. Available also from: https://www.usenix.org/conference/usenixsecurity19/presentation/ho.

39. MITRE ATT&CK®, 2019-07. Available also from: https://attack.mitre.org/tactics/TA0008/.

40. HO, Grant; SHARMA, Aashish; JAVED, Mobin; PAXSON, Vern; WAGNER, David. Detecting Credential Spearphishing in Enterprise Settings. In: *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 469–485. ISBN 978-1-931971-40-9. Available also from: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ho.

41. PANUM, Thomas Kobber; HAGEMAN, Kaspar; HANSEN, René Rydhof; PEDERSEN, Jens Myrup. Towards Adversarial Phishing Detection. In: *13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20)*. USENIX Association, 2020. Available also from: https://www.usenix.org/conference/cset20/presentation/panum.

42. KOSTER, Martijn; ILLYES, Gary; ZELLER, Henner; SASSMAN, Lizzi. *Robots Exclusion Protocol* [RFC 9309]. RFC Editor, 2022. Request for Comments, no. 9309. Available from DOI: 10.17487/RFC9309.

43. SHIRAZI, Hossein; BEZAWADA, Bruhadeshwar; RAY, Indrakshi. "Kn0w Thy Doma1n Name": Unbiased Phishing Detection Using Domain Name Based Features. In: *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. Indianapolis, Indiana, USA: Association for Computing Machinery, 2018, pp. 69–75. SACMAT '18. ISBN 9781450356664. Available from DOI: 10.1145/3205977.3205992.

44. EUROPEAN COMMISSION - DIRECTORATE-GENERAL FOR INTERNAL MARKET Industry, Entrepreneurship; SMES. Eurostat, 2023-05. Available also from: https://webgate.ec.europa.eu/grow/redisstat/databrowser/bookmark/b8edd95c-d80c-449c-b0a1-0f5e1a7fc9f6.

45. EUROPEAN COMMISSION - DIRECTORATE-GENERAL FOR INTERNAL MARKET Industry, Entrepreneurship; SMES. Eurostat, 2023-05. Available also from: https://webgate.ec.europa.eu/grow/redisstat/databrowser/bookmark/d297ee2e-464c-483f-ab12-9346cf6a7ff9.

46. URLSCAN.IO. urlscan.io, 2022-12. Available also from: https://urlscan.io/result/9195e461-079b-41ac-b3d1-34ce204e703b/.

47. ZAHI OHANA, Cyberark. *Phishing as a Service* [online]. 2023-02-16. [visited on 2023-05-04]. Available from: https://www.cyberark.com/resources/threat-research-blog/phishing-as-a-service.

48. ZUGEC, Martin. Bitdefender, 2022-06. Available also from: https://www.bitdefender.com/blog/businessinsights/homograph-phishing-attacks-when-user-awareness-is-not-enough/.

49. LESNE, ANNICK. Shannon entropy: a rigorous notion at the crossroads between probability, information theory, dynamical systems and statistical physics. *Mathematical Structures in Computer Science*. 2014, vol. 24, no. 3, e240311. Available from DOI: 10.1017/S0960129512000783.

50. BROMILEY, PA; THACKER, NA; BOUHOVA-THACKER, E. Shannon entropy, Renyi entropy, and information. *Statistics and Inf. Series (2004-004)*. 2004, vol. 9, pp. 2–8.

51. ZHOU, Shixiang; JEONG, Heejin; GREEN, Paul A. How Consistent Are the Best-Known Readability Equations in Estimating the Readability of Design Standards? *IEEE Transactions on Professional Communication*. 2017, vol. 60, no. 1, pp. 97–111. Available from DOI: 10.1109/TPC.2016.2635720.

Contents of attached medium

- `datasets` ..... used data that does not include personal information
  - `domains_nocnt.txt` ..... testing dataset of CESNET domains
  - `for_measurement_an.trapcap` ..... testing anonymized traffic capture of CESNET
- `src`
  - `filter` ..... source code of resulting NEMEA module
    - `4_stage_filter` ..... 2nd filter stage source
    - `ct_log_collector` ..... source code of CT Log collector
    - `passive_dns_requester` ..... 3rd filter stage source
    - `csv_builder.py` ..... source code of script for CSV export of CT database
  - `thesis` ..... LaTeX source of thesis
- `text` ..... thesis text
  - `thesis.pdf` ..... thesis in PDF format