OSINT-based Email Analyzer for Phishing Detection

Master of Science in Technology Thesis
Cyber Security (EIT)
University of Turku
Department of Computing
2023
Francesco Pavanello

Supervisors:
Seppo Virtanen (University of Turku)
Jouni Isoaho (University of Turku)
Massimo Giaimo (Würth Phoenix)
Simone Cagol (Würth Phoenix)

UNIVERSITY OF TURKU
Department of Computing

Francesco Pavanello: OSINT-based Email Analyzer for Phishing Detection

Master of Science in Technology Thesis, 63 p.
Cyber Security (EIT)
February 2023

It is more and more common to receive emails asking for credentials. They usually say that there is some kind of issue that must be solved by accessing the involved service using the link inside the message text. These emails are often malicious, thought to steal users' or employees' credentials and gain access to personal or corporate areas.

This scenario is commonly known as phishing, and nowadays it is the most common cause of corporate data breaches. The attacker tries to exploit human vulnerabilities like fear, concern or carelessness to obtain what would be difficult to achieve otherwise.

Even if it is easy from an expert point of view to recognize such attempts, it is not so simple to automatize their detection, due to the fact that there are various techniques to elude systematic checks. Nevertheless, Würth Phoenix wants to improve their cyber defense against any possible threat, and hence they assigned me the task of working on phishing emails detection.

This thesis presents a novel program that can analyze all emails delivered to a specifically set up email server without any filtering on incoming traffic, which is then called a "spam-trap-box." Additionally, it is configured with accounts registered for domains owned by failed companies that used to operate in the same industry of Würth Phoenix customers. This way it is more probable to analyze traffic similar to the one in a real case scenario.

The innovative part of the analysis implemented is the use of Open Source Intelligence (OSINT) to compare the most relevant parts of an email with evidence of other phishing attempts indexed on the web, which are generally known as Indicators of Compromise (IoCs).

After the inspection, if an email is categorized as malicious, new IoCs are created to feed the Würth Phoenix Security Operation Center (SOC), which is the service responsible for the protection against cyber threats offered to their customers. The new indicators include more information than the ones used during the analysis, and the findings are inherent to clients' businesses, thus the SOC has more details to use while analyzing their email traffic.

Keywords: phishing, email security, OSINT

Contents

List of Figures

List of Snippets

List of Acronyms

ABAC Attribute-based Access Control

API Application Program Interface

ASCII American Standard Code for Information Interchange

DKIM DomainKeys Identified Mail

DMARC Domain-based Message Authentication, Reporting and Conformance

DNS Domain Name System

EBP Elastic Blockchain Proxy

ECS Elastic Common Schema

HTTP HyperText Transfer Protocol

IANA Internet Assigned Numbers Authority

ICMP Internet Control Message Protocol

IMAP Internet Mail Access Protocol

IoA Indicator of Attack

IoC Indicator of Compromise

IP Internet Protocol

KPI Key Performance Indicator

LDAP Lightweight Directory Access Protocol

MIME Multipurpose Internet Mail Extensions

OSINT Open Source INTelligence

PKI Public Key Infrastructure

POP3 Post Office Protocol - Version 3

RBAC Role-based Access Control

RFC Request for Comments

SIEM Security Information and Event Management

SMTP Simple Mail Transfer Protocol

SOC Security Operation Center

SPF Sender Policy Framework

TCP Transmission Control Protocol

TLS Transport Layer Security

TTPs Tactics, Techniques and Procedures

1 Introduction

Data breaches caused by phishing are the most expensive in 2022, as reported by the IBM annual report "Cost of a Data Breach 2022", available on their website [1]. In fact, the average cost of a data breach in this case is USD 4.91 million, as shown in Figure 1.1, extracted from the report, which means USD 0.26 million more than the previous year. It is also reported that phishing is the second most common cause of a breach, being the initial attack vector in the 16% of the cases happened in the 2022.

Considering these data, it appears obvious that companies need to improve their countermeasures against phishing, considering both the technological and human aspects. Besides all the training that a company provides to their employees to improve their awareness, it is known that people are and will always be the weakest ring of the chain. This is why it is necessary to work also on the detection technologies, trying to reduce the number of malicious emails that reach the employees' mailbox.

At Würth Phoenix they are developing a Security Operation Center (SOC), to offer their clients a complete and constant monitoring of their infrastructure and the related events.

In this thesis a new approach to recognize phishing emails is proposed. It is innovative due to the fact that the evaluation of a message is based mainly on data that is possible to gather using Open Source Intelligence (OSINT). These data

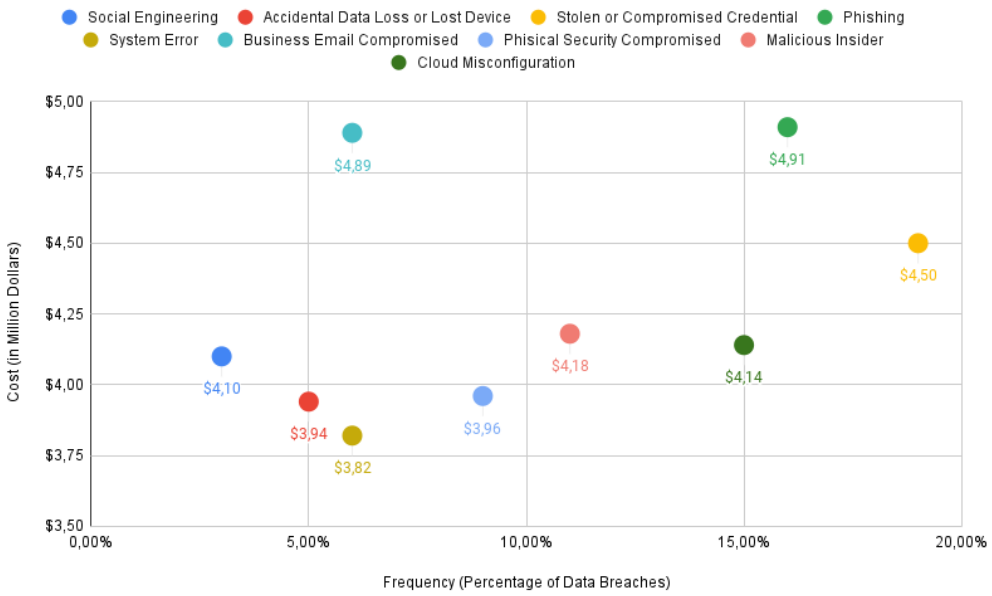

Figure 1.1: Average cost and frequency of data breaches by initial attack vector, data taken from the IBM annual report "Cost of a Data Breach 2022" [1]

are collected and made available for our analysis by two tools integrated into the Würth Phoenix SOC: SATAYO, entirely developed internally to the company, and OpenCTI. Both will be described in Chapter 4.

The idea is to configure a mail server without any traffic filter, hence called spam trap box, since it allows any message to be delivered. It is to notice that the server is configured using domains that were previously owned by companies operating in the same industry of Würth Phoenix clients, so that incoming phishing emails would probably target them too.

Then a Python program downloads all the emails stored on the server and analyzes them to determine if they are malicious. An email verification is done comparing the email headers with information publicly available on the Internet regarding known attacks and attackers, called Indicators of Compromise (IoCs), and considering some other factors and aspects of the messages. Finally, if that was the case, new IoCs are created and ingested by OpenCTI, so as the SOC can count on more

information while monitoring the customers' email traffic.

Besides the practical project, this thesis aims to answer the following questions:

- Which features of an email can reveal more than the others if it is a phishing attempt?
- Which open source information are more useful to detect ongoing phishing campaigns?
- How can the above data be combined to get the most accurate assessment?

The rest of the thesis is organized as follows. Firstly, in Chapters 2 and 3 the theory necessary to understand the work done is explained, starting from the technical aspects, such as email messages formats, email protocols and email validation against spoofing, moving to the description of phishing, Open Source Intelligence, Indicators of Compromise and Spam Filter. Of course, for each topic it is also highlighted why they are relevant for the development of the thesis project. Chapter 4 presents Würth Phoenix SOC infrastructure, to let the reader know the system in which the email analyzer is integrated. Finally, the idea behind the work and its design are exposed in Chapters 5, while the implementation details are described in Chapter 6, considering the results achieved by the first version developed. Final thoughts and remarks are provided in Chapter 7, along with the future steps that still need to be done.

2 Technical Background

Before discussing the core of the topic, it is necessary to introduce some technical concepts needed to fully understand what we are talking about. This chapter starts by explaining the protocols on which the exchange of electronic mails relay, outlining also the messages format. To be more precise, the dissertation will go through Simple Mail Transfer Protocol (SMTP), Post Office Protocol - Version 3 (POP3) and Internet Mail Access Protocol (IMAP). It is subsequently highlighted how security was not a priority when they were conceived. In the end, it is explained which mechanisms have been thought to patch the vulnerabilities over the years, in particular Sender Policy Framework (SPF), DomainKeys Identified Mail (DKIM) and Domain-based Message Authentication, Reporting and Conformance (DMARC).

2.1 Email Message Formats

Based on what is defined in RFC 5322 [2], an email message is composed of two parts: a header, containing all the information necessary to deliver it as well as the data about the sender, and a body, i.e. the message itself. Those are separated by an empty line.

In this original definition, the whole message must include only US-ASCII characters, accordingly to the SMTP protocol, since, when SMTP was designed in the 1980s, transmission channel capacity did not permit to transfer large data. To overcome this limitation, several other documents (RFC 2045 [3], RFC 2046 [4], RFC

2047 [5], RFC 4288 [6], RFC 4289 [7], RFC 2049 [8]) defined the MIME standard, which, as stated in RFC 6532 [9], "added support for the use of 8-bit character sets in body parts, and also defined an encoded-word construct so other character sets could be used in certain header field values". The latter RFC defines another improvement, allowing the "use of Unicode in mail addresses and most header field content".

Besides this character sets digression, the standard describes the email header as a dictionary. This means that each header line constitutes a field, which consists of a keyword, or field name, followed by its assigned value, called field body; the two are separated by a colon. There is not a specific order in which the different fields have to appear, and only the origination date field and the originator address fields should always be set in order to guarantee interoperability between different clients. The former has "Date" as field name, and a date-time as field body; it specifies the moment in which the email is pushed into the mail delivery system by its creator. The latter is composed of three fields: the from, the sender and the reply-to one. The first, representing the author or authors of the message, is formed of the field name "From" and a comma-separated list of one or more mailbox specifications. In case the list contains only one element, the sender field, whose field name is "Sender", is not necessary. Otherwise, it must be set with a single email address, the one responsible for the transmission of the message. Finally, the reply-to may be included to specify to whom replies must be sent. Its field name is "Reply-To" and its field body is a comma-separated list of one or more mailbox specifications.

All the other fields are syntactically optional, accordingly to the RFC 5322 [2], because, as Pete Resnick, the main RFC author, clarified me via emails, they have not encountered clients that have trouble with messages that do not contain these fields while they were writing the document. Anyway, it is necessary to specify at least one email address of a receiver while writing an email using a dedicated

application, otherwise it is not able to send the message. This can be done by setting at least one of the following three header fields: "To", "Carbon Copy" (or "CC") and "Blind Carbon Copy" (or "BCC"). The field body of all the three possibilities is a comma-separated list of one or more email addresses. The difference between them is the fact that the "To" field contains the primary recipients of the message, the "CC" contains the address of those recipients that the sender wants to inform about the message, even if it is not directly thought for them. Finally, the "BCC" contains the addresses that are not to be revealed to other recipients of the message.

Furthermore, every message should have a globally unique message identifier, set in the "Message-ID" field, and a reply should refer to the message, or messages, it is replying to by filling the "In-Reply-To:" and "References:" fields. The former contains the message ID of all the parent messages, while the latter is used to refer to a thread of the conversation. Another common field is the "Subject" one, which contains human readable text that introduces the content of the message. If it is a reply, the subject should start with "RE" followed by the content of the subject field of the original message.

It is important to keep in mind that the specification described in an RFC are not mandatory, they simply state that certain fields are required to interoperate well, hence anyone can do as they prefer. Finally, the headers here reported are the ones useful for the purpose of the thesis, but there are many others described in the standard.

2.2 Email Protocols

Entities involved in an email exchange are shown in figure 2.1. It is possible to notice that mail servers do not reside on user personal machines, otherwise they should always be on. The mail servers are usually shared between a lot of users, each having their personal mailbox hosted on it. A user interacts with the mail

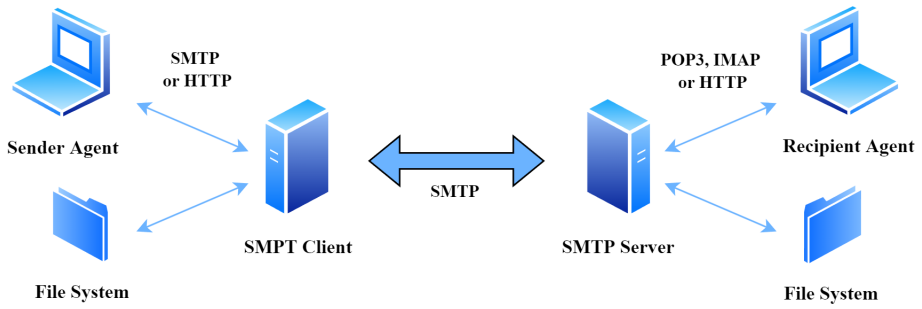

Figure 2.1: Email Exchange Scenario

server through a user agent installed on their PC. Furthermore, each mail server can act as both a client and a server, depending on whether it is on the sender or recipient side respectively.

2.2.1 Simple Mail Transfer Protocol (SMTP)

As it is possible to deduce from figure 2.1, electronic mail exchange is principally based on Simple Mail Transfer Protocol, defined in RFC 5321 [10]. This protocol oversees the transfer of messages from the sender's mail server (in this contest called SMTP client) to the recipient's one (referred to as SMTP server). In addition, it is also responsible for the communication between the sender agent, the machine on which an user compones the message, and SMTP client.

Since the TCP connection is opened by the machine that wants to send the message, and hence push it into the communication channel, SMTP is defined as a "push protocol". On the other hand, the protocol used to navigate on the Internet, called HyperText Transfer Protocol (HTTP), is a "pull protocol", due to the fact that the TCP connection is initiated by the machine that wants to retrieve a file, mainly web pages [11].

Although in the previous paragraph we talk about TCP, it is not the only possible transport over which electronic mails can be exchange. The only requirement is that the communications go through a two-way reliable ordered data stream channel,

since the SMTP server responds to each SMTP client's command with a reply that "may indicate that the command was accepted, that additional commands are expected, or that a temporary or permanent error condition exists" [10].

If TCP is used, IANA specifies that SMTP servers should constantly listen on port 25 for incoming requests. The main commands are, in order as they are send during an email transaction: EHLO, MAIL FROM, RCPT TO, DATA, QUIT. The first one is used to identify the SMTP client to the SMTP server, the second and the third to set, respectively, the sender and the recipient, the second to last is used to communicate that all the following lines, until a line containing a single period, are part of the mail data, while the last one is used to close the communication channel.

It is not mandatory for the mail transfer to be a peer-to-peer transaction, as portrayed in the scheme 2.1. In fact, a mail server can act, not only as an originating or delivery system, but also as a relay and gateway. The first two are represented in figure 2.1, since they are respectively the SMTP client, which introduces mails into the system, and the SMTP server, which is queried by the recipient's user agent to retrieve and read messages. Instead, both relays and gateways are not in the figure, but they should be in the middle between the client and the server. Accordingly to the SMTP RFC [10], the difference between them is that a relay transmits emails without any changes, except adding trace information, to another SMTP server, while a gateway performs the necessary transformations on the message in order to transmit it from a transport environment to another one.

One interesting vulnerability related to the configuration of a mail server is often exploited by spammers; it is known as "Open Relay". This configuration basically allows anyone to send emails to anyone else, due to the lack of an authentication requirement, and hence it leads to email address spoofing and spam. Often this is the default mail server configuration, since it is the simplest one. Nowadays, many servers maintain a blacklist of known open-relays, in order to not accept messages

coming from these ones [12]. To secure a relay, it should accept only:

- Messages from local addresses to local mailboxes
- Messages from local IP addresses to external mailboxes
- Messages from external IP address to local mailboxes

In other words, a relay should block emails coming from external IP addresses and intended for external mailboxes; of course, all the messages must be from authenticated and authorized users [13]. At section 2.3 countermeasures to email addresses spoofing are explained.

Due to the fact that SMTP is a push protocol, the email exchange needs to involve another protocol to retrieve the messages from the SMTP server, such as POP3 or IMAP. They are defined as Mail Access Protocol, and are described in Sections 2.2.2 and 2.2.3 respectively.

As a result of an increasing use of web apps to access one's mailbox, it is nowadays common to use also HTTP to interact with the mail server, both to send and retrieve messages. Anyway, since this protocol is not specifically intended for email exchange, it is not described in the thesis.

2.2.2 Post Office Protocol version 3 (POP3)

Post Office Protocol version 3 is defined in RFC 1939 [14]. It is a simple and limited protocol: in fact, it does not have the ability to keep state information across different sessions, and it does not offer a way to organize email messages on the server. Anyway, it is possible to create a hierarchy of folders locally on a client machine, even if this will not be synchronized on the other user's devices. POP3 connections are opened on the server by the client on port 110. As for SMTP, the server always answers the client, letting it know the outcome of the issued command.

A POP3 session progresses throughout three states: authorization, transaction and update.

During the first one, the client must authenticate itself sending username and password; if the operation is successful, the server retrieves all the resources associated to the client, giving it exclusive access.

The POP3 session enters hence the transaction state, and the client can retrieve messages. The user agent can often be configured to operate in two modalities: "download and delete" or "download and keep"; in the former case, it marks a message as deleted after having downloaded it, while in the latter the message is left on the server as it is. The possible commands issued to a POP3 server during this state are: STAT, LIST, RETR, DELE, NOOP, RSET. The syntax of the commands is described in the RFC 1939 [14].

When the client wants to log out, it issues the QUIT command; this makes the session enter the update state, during which the server removes all the messages marked as deleted from the mailbox.

2.2.3 Internet Mail Access Protocol (IMAP)

The most recent version of Internet Mail Access Protocol is defined in RFC 9051 [15]. This protocol overcomes POP3 limitations, for example giving the user the possibility of organizing messages in remote folders, i.e. mailboxes. Consequently, IMAP implementation is considerably more complex, since IMAP servers need to keep state information across different sessions. It also allow users "to search remote folders for messages matching specific criteria" and "to obtain components of messages", for example only the message header or just one part of a multipart MIME message [11].

If the IMAP server communicates on TCP, it listens on port 143 for cleartext communications or on port 993 for TLS ones. A message can be accessed using two

different identifiers: both are 32 bits numbers, but one is unique inside a mailbox, while the other is relative to the number of messages in the mailbox.

The former, called Unique Identifier (UID), is incremented each time an email is added to the mailbox; this means that UIDs are not contiguous, due to the fact that an email can be expunged from the mailbox but the UID of the subsequent messages are not updated. In fact, a UID should never change once it is assigned to an email. If used with the unique identifier validity value assigned to the mailbox (UIDVALIDITY), which usually is a 32-bit representation of the current date/time when the value is assigned, it is possible to globally identify the message not only inside the mailbox, but also in any subsequent mailbox with the same name forever.

The latter, referred to as Message Sequence Number, is a value between one and the number of messages in the mailbox. Hence, if a message is permanently deleted, all the subsequent messages' sequence number is decreased by one. This is useful both to access messages by relative position in the mailbox, as if it is an array, and to do mathematical operations to calculate, for example, how many messages have been arrived between other two how many massages in the mailbox have a greater or lower UID.

Another important feature is the possibility of assigning one or more flags to a message. Each flag represents a specific status or characteristic of the message, and it can be permanent or valid only during the current session. These named tokens are divided in two categories: system flags and keywords.

System flags are pre-defined in RFC 9051 [15], and begin with "\" and currently are:

- \Seen, used to mark a message as read
- \Answered, for messages the user has already answered to
- \Flagged, for highlighting an email as important

- \Deleted, to tell the server to delete the message when executing the EXPUNGE command
- \Draft, for messages to be completed
- \Recent, which exists only for backwards compatibility with older versions of IMAP, in the last one it is considered deprecated, and was used to mark messages arrived after the previous session

Unlike system flags, keywords are defined by the server implementation, and it may be possible for user to register new ones. They start with a "$", and some of the most useful are described in the RFC 9051 [15]. For example:

- $Forwarded, set by the email client to mark a message that has been forwarded to another email address
- $Phishing, set by the delivery agent to warn a user that the content of the message can be malicious, specifically a phishing attempt
- $Junk and $NotJunk, set by the user to mark a message with undesired (or desired) contents

As previously stated, IMAP introduces the possibility of organizing emails in different mailboxes, giving a user the possibility of creating new ones. Usually, all new messages are placed into the INBOX folder, which in fact is "the primary mailbox for this user on this server" [15].

As POP3, an IMAP session progresses through different states: Server Greeting, Not Authenticated, Authenticated, Selected, Logout. The first one is entered after a connection is established between the client and the server. After that, if the connection is not pre-authenticated, the connection enters the Not Authenticated state, where basically the user supplies their credential to the server. Then, in the Authenticated state, the client selects the mailbox that they want to access; once

this is done, the session progresses into the Selected state, during which the user is able to perform actions over the messages inside the mailbox. Finally, in the Logout state, the connection is being terminated. During each state, only a specific subset of all the commands is accepted by the server. IMAP commands used during the implementation of the email analyzer, divided per states as specified in the RFC 9051 [15], are:

- Not Authenticated State:
  - STARTTLS, to start a TLS negotiation
  - LOGIN, to identify the client to the server and use a plaintext password to authenticate this user
- Authenticated State:
  - LIST, to enumerate the mailboxes associated to the authenticated user
  - CREATE, to create the a new mailbox "Analyzed" where to move the analyzed emails
  - SELECT, to select the "INBOX" mailbox and access new messages
- Selected State:
  - SEARCH, to enumerate all the emails in the selected mailbox
  - FETCH, to retrieve data associated with each enumerated message
  - COPY, to copy the specified message into the "Analyzed" mailbox
  - STORE, to associate the flag "\Deleted" to the message in the "INBOX"
  - EXPUNGE, to permanently delete all the messages marked with the flag "\Deleted"
- Any State:
  - LOGOUT, to deauthenticate the current user and close the connection

2.3 Email Validation Against Spoofing

Email spoofing is based on the fact that an attacker sends a message using an email address that they do not own as sender address. It is usually done to impersonate someone else, in order to gain the recipient's trust and confidence; this is often the starting point of a phishing attack.

As countermeasure to spoofing, three standards were designed: Sender Policy Framework, defined in RFC 7208 [16], DomainKeys Identified Mail, defined in RFC 6376 [17], and Domain-based Message Authentication, Reporting and Conformance, defined in RFC 7489 [18]; if used together, they provide both email authentication and validation. "SPF primarily publishes information about what host addresses are authorized to send mail for a domain. DKIM places cryptographic signatures on email messages, with the validation keys published in the DNS. DMARC publishes policy information related to the domain in the From: header field of email messages." [19]

2.3.1 Sender Policy Framework (SPF)

A Sender record consists of a special DNS TXT record that defines which mail server can send emails for the domain the record is referred to. It is easily recognizable since it always start with "v=spf1" followed by a list of rules that define the allowed mail servers and are evaluated in order from left to right. Evaluations, carried out comparing the rules with the "mail from" SMTP header field of the received email, can match, not match, or return an exception. In the first and last case, process ends, otherwise it continues checking the subsequent rule.

Mail servers are identified in different ways, called mechanisms: "all", "include", "a", "mx", "ip4", "ip6", "exists" and "ptr" (this last one should not be used). They can directly identify a server using its IP address or a CIDR block of addresses ("ip4" and "ip6") or require additional DNS lookups to resolve a specific DNS record ("a",

"mx", "exists", "ptr"), which can refer either to the domain itself or to another, specified after the mechanism. The mechanism "include" is used to evaluate the SPF record of the specified domain, while the "all" one always matches and hence is used as explicit default in the end of the record. There is a limit of 10 DNS lookups to evaluate an SPF record that, if exceeded, causes the process to return a "permerror", and the record is considered invalid.

Mechanisms are used in conjunction with qualifiers that determine the result of the rule evaluation. If not specified, the default one is "+", which causes the eventual match to return a "pass" result, meaning that the server is allowed. The "-" qualifier defines a "fail", thus the server is explicitly not authorized. The "softfail" qualifier is represented by the " " and means that a strong policy on that host is not in place even if it is not trusted and further analysis are suggested. Finally, "?" stands for "neutral", which means that there is no definite assertion (positive or negative) about the client; it is more or less the same as the "none" result that is returned in case that none of the policies matches.

A simple example of SPF record can be:

v=spf1 ip4:154.10.12.252/30 mx include:example.com -all

which means that only the machines that has 154.10.12.253 or 154.10.12.254 as IP address, are present in an MX record of the domain the SPF record is about, and matches a policy in the SPF record of the domain "example.com" are allowed to send email for this domain. Since the default policy is a "fail", all the other machines are explicitly unauthorized.

It is to notice that SPF standard alone is not enough to prevent spoofing: since it evaluates only the SMTP header information, it is still possible to use an arbitrary email address in the "FROM" message header.

2.3.2 DomainKeys Identified Mail (DKIM)

Unlike SPF standard, DKIM aims to authenticate the message header, instead of the SMTP one. To achieve this, a special header, the DKIM-Signature header, is added to the message; it contains the signature of the message headers, the signature of the message body and all the information necessary to verify their validity. The header value is a tag-list, the most relevant of which are:

- "v=", the version of DKIM. To be compliant to the RFC 6376 [17], it must be 1
- "a=", the algorithm used to generate the signature
- "b=", the signature hash of the message headers listed in "h=" tag
- "bh=", the signature hash of the message body
- "d=", the Signing Domain Identifier (SDID), which is a domain name that refers to the identity claiming responsibility for the current message. Hence, it must correspond to the DNS name under which the DKIM key record is published
- "h=", the colon-separated list of the headers field names concatenated to validate the signature
- "s=', the selector used to identify the correct public key inside the domain key namespace

A simple example of DKIM-Signature header could be:

DKIM-Signature: v=1; a=rsa-sha256; d=example.com;
s=verona; h=from:to:subject:date;
bh=MTIzNDU2Nzg5MDEyMzQ1Njc4OTAxMjM0NTY3ODkwMTI=;
b=dzdVyOfAKCdLXdJOc9G2q8LoXSlEniSbav+y
uU4zGeeruD00lszZVoG4ZHRNiYzR

When receiving something like this, in order to verify the signature, the recipient retrieves the public key from the key namespace of example.com that is assigned to the selector "verona". This is made through a DNS query; since all the DKIM keys are stored in a subdomain named "_domainkey", the query in this case will be for 'verona._domainkey.example.com'. It returns a TXT record that in this case should look like:

v=DKIM1; p=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQ
KBgQDwIRP/UC3SBsEmGqZ9ZJW3/DkMoGeLnQg1fWn7/zYt
IxN2SnFCjxOCKG9v3b4jYfcTNh5ijSsq631uBItLa7od+v
/RtdC2UzJ1lWT947qR+Rcac2gbto/NMqJ0fzfVjH4OuKhi
tdY9tf6mcwGjaNBcWToIMmPSPDdQPNUYckcQ2QIDAQAB

where the "v=" tag represents the version of the DKIM key record, that, if present, must be the first tag of the list and must be set to "DKIM1", while the "p=" tag defines the public key requested. Other tags may be present.

After having retrieved the public key, it is possible to calculate the two signatures by hashing both the headers, concatenated as described by "h=" header, and the body of the message; this must be done using the algorithm indicated in the "a=" tag. To pass the validation, the hash of the body must match the value of the "bh=" tag, while the hash of the headers must match the value of the "b=" tag.

Also DKIM, used alone, has a weakness, since the domain specified by the "d=" tag is not required to be same as the domain used in the "From" header of the message. This means that attackers can use a domain they control to pass the DKIM validation in the DKIM-Signature header, while using a spoofed email address in the "From" message header.

2.3.3 Domain-based Message Authentication, Reporting and Conformance (DMARC)

DMARC was introduced to ensure that the SPF or DKIM weaknesses are not exploited. In fact, in order to pass the DMARC authentication check, a message not only needs to pass at least one between the SPF and DKIM checks, but it also requires that the related domain specified in the SMTP "From" header or the domain declared in the "d=" tag of the DKIM-Signature header matches with the domain indicated in the message "From" header. Since the DKIM-Signature header does not change if a message is forwarded, its alignment is more important than the SPF one.

This standard also offers the domain owner a way to define actions that message recipients must perform when they receive an email coming from that domain. Basically, it is possible to define two things: the policy to apply when receiving an email that seems to come from that domain but that does not pass the DMARC alignment, and the type of the report to send back to the domain owner.

There are three possible policies:

- "none", which means that the owner does not require the execution of any specific action
- "quarantine", which signifies that those emails that fail the DMARC check should be considered malicious
- "reject", which indicates that the emails that do not pass the alignment should not be forwarded further.

Instead, there are only two possible kinds of report:

- Aggregate reports, which usually are daily reports that sum up all the authentication results, regardless of the outcome, in order to provide the domain

owner with an overview of the real-world emails stream. This is useful to set up the most appropriate policy

- Failure reports, which are usually generated and sent to the domain owner as soon as an email fails the DMARC check. This type of report provide more information about the failure than the previous one

DMARC specifications are published in a DNS TXT record at the _dmarc subdomain. It contains a list of tags, of which "v=" and "p=" are mandatory. The former indicates the DMARC version, which must be the first of the list and its value must be "DMARC1" for the entire record to be considered valid. The latter specifies the policy that the recipient of an email that does not pass the DMARC check has to apply. The possible values are "none", "quarantine" and "reject", as previously described.

Other possible tags are the ones that tell recipients what kind of report to send and to whom to direct them:

- "rua", whose value is a comma separated list of "mailto:" followed by the address to send the aggregate report
- "ruf", which is a list as described for the previous tag, but it contains the addresses which a failure report should be forwarded to
- "fo", which provides requested options for generating a failure report

An example of DMARC TXT record is:

v=DMARC1; p=quarantine; rua=mailto:owner@example.com;
ruf=mailto:owner@example.com

3 Theoretical Background

In this chapter the background knowledge necessary to understand the need and the innovation behind the thesis will be explained. It starts with a brief description of Phishing Attacks, clarifying what they consist in and why on the one hand they are still feasible and common, while on the other so difficult to prevent. The chapter then continues with an overview of Open Source INTelligence, that clarifies what it is and why it became so important for the Cyber Security. Then it is illustrated what Indicators of Compromise are, comparing them with Indicators of Attack, and it is shown how to use them during an analysis. In fact, to create valuable IoCs is the aim of the email analyzer. Finally, while describing what spam filters are, the end of the chapter illustrates the difference between that technology and the one proposed in this thesis.

3.1 Phishing

The most common attack based on email spoofing is phishing. It consists in the attacker's attempt to trick their victim into providing credentials or other sensitive information, which can then be used to gain access to, for example, a bank account or private corporate area. In fact, the use of a spoofed email address is useful to gain the victim's trust. Phishing can also be the attack vector used to introduce a ransomware inside a company. In both cases, the message is thought to make people click on a malicious link, which usually conducts to a fake login page, or open

attachments that then run malicious code on the victim's machine, often installing malicious software. [20]

All the attacks that, like phishing, exploit human ingenuity to gain sensitive information are generally referred to as social engineering. In fact, their goal is to persuade people in performing some action that the attacker is not able to do on their own. These attacks usually rely on at least one of the following factors [21]:

- Reciprocation, or better the fact that people form implicit or explicit obligations towards each other
- Consistency, in the sense that people tend to be consistent with previous decisions
- Social proof, because people tend to act like those around them, in order to feel part of a group
- Likeability, which means that people trust who they like, find convincing, attractive or similar to themselves
- Authority, since people tend to obey those they fear, to avoid punishment
- Scarcity, usually of time, to persuade people to act quickly without letting them think too much about what they are doing

Moreover, there are different types of email phishing attacks [22] [23]:

- Mass or bulk phishing, which does not have a specific target, but rather the message is as general as possible and a large number of emails is sent, hoping that at least one recipient will be deceived
- Spear phishing, that on the other hand has a specific target, so the message must be built specifically for each attack, in order to seem legitimate. In fact, in these cases the attacker gathers as much information as possible before writing the message

- Whaling, which is a special sort of spear phishing targeting the most important profiles inside a company, in order to obtain administrative credentials

- Cloning, that consists in replicating or simulating a legitimate message, but substituting links or attachments with malicious ones

- Business Email Compromise (BEC), in which an attacker impersonates a prominent figure within a society, asking an employee or a partner outside the company to perform some actions

From the above, a phishing email may contain an unexpected sender, for example the team leader or another high profile of the victim's company, a promise of some gifts or benefits, a tone that conveys urgency or threat, misleading information about the sender or the reason behind the request in the message.

In their work about email phishing detection [24], Fette et al. listed the ten features they think are the most useful to implement a good phishing filter. Hence, based on these features, a phishing email could present IP-based URLs, links to a recently registered domain, links where the URLs displayed a domain that does not match the one really linked, links' text that contains words like "link", "click" or "here", MIME, HTML or JavaScript sections. Other features are the number of links, the number of domains, the number of dots in a single URL and the output of a spam filter.

Even if it is enough clear how phishing emails look like, it is not really easy to block any malicious attempts of delivering such emails. In fact, the sender can seem a legitimate one, or even be a trusted one, if the mailbox was previously compromised, the subject and the content may not look like malicious at all, and the IP address of the mail server, or the hostname itself, might not be in any blacklists. Consequently, if the filter fails to block it, the success of a phishing attempt depends exclusively on the recipients awareness.

Lain et al. explained in their work about phishing in organization [25] that some employees are particularly more vulnerable to this kind of attacks than others. They stated that both younger and older employees are more at risk than the others, as well as the ones with lower computer skills and the ones who use computers daily for repetitive tasks using always the same software. They also pointed out that, during the fifteen months of monitoring they carried out, more or less a third of the 14733 participants of their study clicked on a link or attachment contained in their simulated phishing emails, while a quarter performed dangerous actions; out of these, a quarter performed dangerous actions on more than one sample.

Additionally, they investigated the effectiveness of phishing warnings and training. According to the literature, they discovered that contextual warnings are really effective, although it is not true that the well detailed ones are significantly better than the more concise ones. Instead, contradicting prior research results, they found that the combination of simulated phishing exercises and voluntary embedded training is not helpful at all. However, Jampen et al. in their literature reviewed about phishing training [26] identified various factors that influence the effectiveness of such training and outlined how an effective training program should be designed and implemented.

Finally, Lain et al. also revealed that crowdsourced phishing detection can be efficient and sustainable in large organizations, a solution suggested previously in other two papers [27] [28]. The idea is detecting ongoing phishing campaigns thanks to employees' reports of suspicious emails. They believe that, if employees keep reporting suspicious emails over long period of time and their reports are sufficiently accurate and timely, organizations are able to detect and stop phishing campaigns with a short delay from their start. They also observed that the operational workload to process all the reported emails is acceptable even in large organizations.

To conclude, it is to mention that email is not the only mean used to carry out

phishing attacks. They can be conducted by calling the victim on the phone, and in this case it takes the name of vishing, from VoIP. Similarly, instead of using emails, it is possible to deliver the malicious message via sms, known as smishing, or posting a malicious link in a website, blog or social network [22] [29].

3.2 Open Source Intelligence (OSINT)

Open Source Intelligence is the branch of intelligence that collects and analyzes information only from publicly available sources. Nowadays, the main open source is the Internet, but OSINT gathers information also from offline data, like newspapers, government public documents, gray literature, radio or television transmissions, and generally everything that can be trustworthy and used to support a thesis or decision. Anyway, only online sources are consulted while performing the email analysis.

As described by Hwang et al. [30], the OSINT process can be divided into 5 steps:

- Identifying the sources, to determine what data are needed and where to find them
- Data collection, both using active and passive techniques
- Data processing, which means to filter and associate the findings
- Data Analysis, to derive a conclusion according to the investigation purpose
- Reporting, which means to summarize the evidence and write them in a report

The first big advantages of OSINT are the simplicity and the rapidity of gathering a large amount of information from different sources, while usually there is a lot of bureaucracy to fill before obtaining access to restricted documents and resources. Moreover, most open sources are freely accessible, or very cheap. This means that,

potentially, everyone is able to collect the information needed, and not only national intelligence agencies. Of course, the ability to determine which data and sources are more relevant and reliable is essential to take the best decision. Hence, with respect to other forms of intelligence, the challenge shifts from the data gathering to the data evaluation.

Other advantages are [30] [31] [32]:

- Clarity, since sources and information are validated during the processing step and in general most of the reference materials are gathered from other open sources
- Usability, due to the fact that there are not any communication restrictions on the data found
- Efficiency, not only in terms of time, but also of resources (man-hours)

On the other hand, it is possible to identify also some disadvantages of using OSINT [30] [32], first of all the fact that the amount of data collected may be too large and too heterogeneous for a practical analysis, making the evaluation more difficult. It is also important to keep in mind that not all the open sources are reliable and trustworthy. Furthermore, some organizations still underestimate the importance of OSINT data, due to the fact that they are publicly available, and sometimes researchers cannot freely access open source data because of the constraints of the internal computer network. Finally, malicious users can access all these public information to commit cybercrimes.

In fact, Hwang et al. [30] recognized the fact that three possible threats exist in the OSINT environment, that can lead to different cybercrimes:

- Data dissemination, in case confidentiality or availability of the open source data collected are not guaranteed. An attacker can use those information to

acquire knowledge about a specific victim and plan a targeted attack or delete some data to delay the research or to modify the outcome of the analysis

- Data privacy breach, if data are not anonymized before their publication. Typical uses of these data are spear phishing or identity theft

- Data forgery and alteration, which is possible when the integrity of the data gathered is not ensure. It can cause a diversion of the analysis results or the spread of fake news

Hence, it is fundamental to fulfill some security requirements while collecting and using open source data, such as data encoding and encryption, maintenance of backup and recovery, de-identification of personal information, environment monitoring against intrusion, use of signature or blockchain technology, user authentication, access control, forward secrecy [30].

Due to the number of sources to monitor, most of them accurately categorized by the project "OSINT Framework" [33], a lot of tools were developed through time to help people collect data. Generally, these tools are specialized in gathering a specific type of data, like email addresses, IP addresses, hostnames, files, from different sources.

They usually are open-source projects shared on GitHub, but we can mention also programming libraries and search engines that help scraping social networks and web pages. Anyway, there are also proprietary software that expose API for querying their database of collected data, sometimes for free under a certain threshold. Finally, it is possible to buy a license for big data analysis and visualization platforms, with the aim of simplifying the analysis and the correlation of the data gathered.

Yamin et al. [34] have been dealing with finding and categorizing according to the "Lockheed Martin Cyber Kill Chain" the main solutions currently available. The

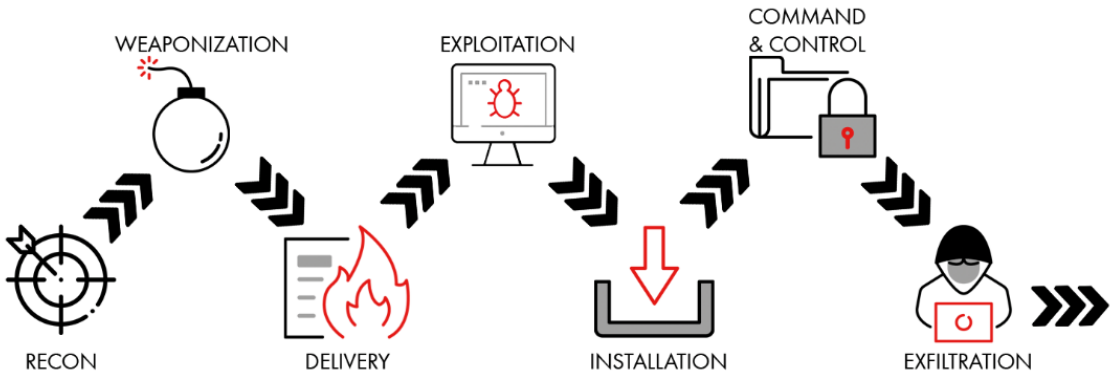

Figure 3.1: Lockheed Martin Cyber Kill Chain

Cyber Kill Chain developed by Lockheed Martin is a commonly used representation of the various stages of a cyber attack, as represented in Figure 3.1.

Besides the tool used, what can make the difference during OSINT researches is the ability of searching also inside the deep and the dark web, that actually represent the major part of the Internet. Indeed, this obscure side of the web is generally preferred by cybercriminals to share illegitimate content, because of the anonymity provided to the user by its design. Hence, it is the perfect place where to conduct OSINT investigations and to find evidence of a cybercrime, even if the de-anonymization of the threat actor is not trivial [32]. An open-source project on GitHub with the aim of collecting sources from this obscure part of the net is "deepdarkCTI" [35], which enumerates for example forums, telegram channels, and onion ransomware groups' websites.

3.3 Indicators of Compromise

Indicators of Compromise (IoCs) are basically the proof that a malicious attempt of compromising the machine was carried out. They are virtual artifacts that can reveal who tried to attack the machine or infrastructure, what they did, which

vulnerabilities they exploited, the technologies used, and the data exfiltrated; hence they are useful to determine the severity of the incident. In fact, no matter how attackers try to hide themselves, it is always possible to find some clues about them and what they did.

IoCs differ from Indicators of Attack (IoAs) since the latter are real time indications that an attack is underway, while the former are a posteriori traces of the intrusion. As explained by CrowdStrike [36], IoAs focus on detecting malicious patterns on the network or machine, hence they do not really care about which tools or exploits are used; their aim is to detect that something malicious is going on to stop it. Indeed, since IoAs look for tactics, techniques and procedures of targeted attackers, they allow the recognition also for zero-day attacks, which is not possible using only IoCs. However, these can be used to compare past episodes with what is going on now, helping to contain attacks earlier in the attack life-cycle, thus limiting their impact.

To better clarify, possible IoCs examples are:

- Malware signature, usually the md5 or the sha256 hash of the malicious software
- Files signature, the hash of complex documents that hide malware inside them
- Anomalous traffic, recorded on unusual times, directed to unexpected IP addresses, or unusual for the amount of bandwidth used
- Anomalous IP addresses or hostnames, that tried to connect to the monitored machines
- Anomalous geographical location of the machine that is trying to authenticate to a service

While IoAs can be:

- Anomalous code execution, which is performing suspicious actions
- Anomalous persistence, of allocated memory or established connections
- Unusual contact with a command and control site
- Lateral movement attempts

The scope of the email analyzer is to create valuable IoCs in order to feed the Würth Phoenix SOC with information that can help to detect future malicious emails. In this sense, we will extract the sender email address, the sender mail server hostname and IP address, the signature of any attachments, and the text in the subject header value field of those emails that, according to the heuristic OSINT analysis performed on their characteristics and content, are evaluated as malicious. The desired outcome is that this information can be correlated to the IoAs collected by the monitoring engine to more confidently and accurately determine whether incoming emails should be blocked or delivered.

3.4 Spam filter

Spam filters are programs that analyze emails to understand if they are spam or not, in case of binary classification, but some are also able to distinguish spam into different categories based on their topic [37] [38]. The analysis usually happens at SMTP level, by inspecting the content, email headers, the sender email address, attachments to decide whether to block the email or to deliver it. This means that emails are checked before they arrive to the recipient's inbox. Anyway, filters can be set also for outgoing emails to block malicious emails sent by a compromised mailbox. Those are commonly configured to block emails that contain viruses, ads, or do not pass and authentication check based on the DMARC record (described at Cap. 2.3.3).

The simplest ones scan an email searching for text that matches regular expressions, or looking for keywords that may suggest it is some unwanted content or a malicious attempt to make the reader perform some actions. Furthermore, the sender IP and email addresses are both compared with the ones previously blacklisted, to save the time of scanning the whole mail content. There is also the possibility, using data-driven programming languages, to define conditions that, if they occur, trigger some specific actions in response. It is finally possible to involve artificial intelligence in order to detect spam emails without depending on previous occurrences and analysis, and at the same time enabling more advanced features, for example image content recognition.

However, spammers are constantly improving their techniques to evade filters [39]. For example, they may manipulate the message using poisoning text or obfuscated words. The former refers to adding random words or specific ones targeted to the anti-spam filter and the victim, avoiding well-known spam words. The latter exploits special characters or some encoding techniques to trick the spam filter. Other possible methods to disturb textual filters are introducing random text in the email background, for example hidden HTML tags, embedding the spam content into images or attachments, inserting URLs that redirect to malicious sites or download harmful files, or including parts in different languages.

Spam filters differ from what is exposed here since this latter does not aim to avoid malicious or unwanted emails to reach the recipient mailboxes. Indeed, it works on emails that arrive to some mailboxes set up as honeypots or spam trap boxes, where everything is allowed to be delivered. The proposed solution examines all these emails, extracting IoCs that will be used by the email filter of the Security Operation Center to determine whether emails directed to corporate email addresses should be dropped.

4 Würth Phoenix SOC Infrastructure

This chapter aims to describe the technologies that constitute the system in which the email analyzer will be integrated into. The log management and Security Information and Event Management (SIEM) module of Neteye is the first to be described. Neteye is an ecosystem developed by Würth Phoenix able to monitor and control the infrastructure in which it is installed. The module with which the email analyzer interacts is basically composed by the ELK stack, so called since its components are Elasticsearch, Logstash, Kibana, to which Beats was later added. It is often also referred to as Elastic Stack. As stated on Elastic web site [40], Elasticsearch is a search and analytics engine, Logstash is a server-side data processing pipeline that ingests and elaborate data, making them available for Elasticsearch, Kibana is the application used to create charts and graphs using the data in Elasticsearch and finally, Beats is a family of lightweight, single-purpose data shippers.

Then there is the description of the Filigran platform called "OpenCTI", which is an open source platform useful to collect, visualize and correlate in only one place all the IoCs and Cyber Threat Intelligence knowledge collected by an organization. This is the main source from which the SIEM acquires information and data about threats, in order to be always updated; hence it will be in charge of collecting also the IoCs created by the project described in this thesis.

Finally SATAYO is presented, since it is the platform used by the SOC to perform the Exposure Assessment of clients' perimeter. It is connected with the email analyzer since this latter will use SATAYO results to perform the analysis of emails. In fact, each evidence showed in SATAYO is collected through an OSINT research, and the correlation with OSINT findings is one of the main innovative features of the proposed solution.

4.1 Elastic Stack

The Elastic Stack is the core of the log management and SIEM module of Neteye, which is a solution developed by Würth Phoenix and adopted by its SOC to monitor the environment of its clients. Therefore, it represents the final consumer of the IoCs that the email analyzer creates, with the purpose of improving the detection of phishing emails.

As mentioned before, it is composed by Elasticsearch, Logstash, Kibana and Beats, which are described in this section. Basically, it is used to collect, organize, analyze, enrich and visualize logs coming from the Neteye satellites installed inside the client network. It really helps the analysts to have always a clear overview of what is happening inside the client's environment, making them able to immediately notice if something is wrong thanks to a set of detection rules. This ability is constantly tested by Würth Phoenix SOC penetration testers and ethical hackers, that carry out all kind of attacks exploiting vulnerabilities thanks to the most recent Tactics, Techniques and Procedures (TTPs) to verify if the SIEM detects their malicious activities. This is why they define their SOC as Attacker Centric. In Figure 4.1 it is possible to observe how the log management module of Neteye is implemented using Elastic Stack components.

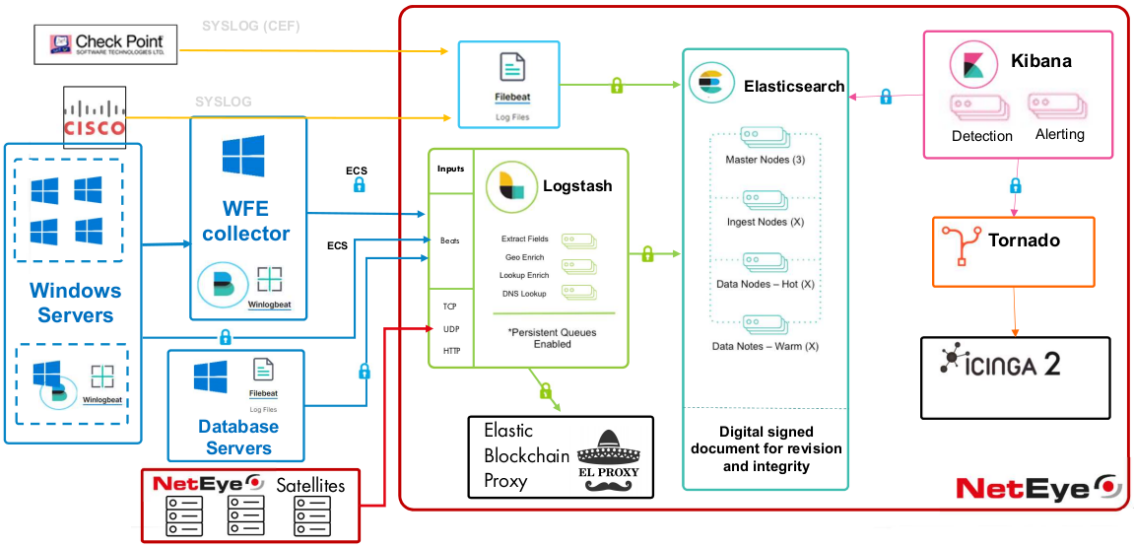

Figure 4.1: Neteye Log Management Architecture

4.1.1 Elasticsearch

Basically, Elasticsearch is the central component of the Elastic Stack and it consists of a distributed database (written in Java) where the data are stored and indexed. It receives raw data from various sources; this data can be, for example, logs or system metrics, and they are ingested by Logstash before being indexed by Elasticsearch. During the ingestion, as it is explained more in depth in Section 4.1.2, raw data are parsed, normalized and enriched. After this process, each log becomes a structured JSON document; hence, a document is a collection of fields that have their datatype. An Elasticsearch index is a collection of documents that are related to each other; indices are then divided into shards, which are Apache Lucene instances, distributed across different nodes and replicated to improve performance and resiliency. Cooperating nodes compose a cluster. Shards are also divided in segments, that are read-only piece of data available for search when Elasticsearch stops writing into them.

More specifically, an index is composed by settings and mappings. The former

establish where to store data, how many shards will an index have, how many replica, or the index-life-cycle policy. Settings can be static or dynamics; the difference is that dynamics can be updated. The latter define instead how documents and fields are stored and indexed. Dynamic mappings allows Elasticsearch to add new fields automatically while indexing a document based on matching conditions described in the so called dynamic templates. This is particularly useful to define types for unknown fields. Explicit mapping allows a user to precisely choose the set the type for known fields. Mapping also enforced the Elastic Common Schema, described in Section 4.1.2.

Finally, Elasticsearch uses a data structure called "inverted index", which enable user to perform very fast full-text searches. In fact, it lists every unique word that appears in any document and identifies all of the documents each word occurs in [41].

4.1.2 Logstash

Logstash is used to aggregate and process data before letting Elasticsearch ingest them. It ingests data from different sources, like logs, metrics, web applications, data stores, and various AWS services. Then, it parses each event, identify named fields to build structure, and transform them to converge on a common format known as Elastic Common Schema (ECS). In fact, it is able to elaborate data regardless of format or complexity [42]. For example, it can:

- Derive structure from unstructured data with grok
- Decipher geo coordinates from IP addresses
- Anonymize PII data, exclude sensitive fields completely

In the end, when data are ready, Logstash has the possibility of sending data to one or multiple output. In Neteye, it sends output to Elasticsearch and to El Proxy. The

latter is described later in this section. Of course, it is able to orchestrate parallel pipelines to improve performance and carry out different processes at the same time.

As mentioned before, the most important task of Logstash is to normalize data, abstracting from the sources, to simplify searches. This is achieved defining a common set of document fields with their datatype for data ingested into Elasticsearch, which is called Elastic Common Schema. For example, fields like client_ip, src_ip, source_ip, nginx.src_ip are normalized to source.ip with type ip. ECS unifies all modes of analysis available in the Elastic Stack, including search, drill-down and pivoting, data visualization, machine learning-based anomaly detection, and alerting. Hence, users can search with the power of both unstructured and structured query parameters and automatically correlate data from different data sources [43].

Logstash is also essential for the purpose of the thesis project, since it is in charge of enriching the events that arrive to the log management modules with additional information from external sources. For example, it is able to:

- Identify web services or vendors based on known IP addresses
- Add product information to retail orders based on product IDs
- Supply contact information based on an email address
- Add postal codes based on user coordinates

Anyway, in our case it will correlate the data collected on phishing attempts and obtained from OpenCTI with the new evidence that it receives in input, defining if these ones are also malicious or not.

Elastic Blockchain Proxy (EBP), a.k.a. El Proxy, permits to sign the logs in real-time in a blockchain stored in Elasticsearch in a dedicated index. Each log contains the hash of the previous log (calculated on a set of defined fields), thus to alter or delete a single log it is necessary to alter the entire blockchain from this point.

4.1.3 Kibana

Kibana is the frontend application that acts as a presentation layer. It is mainly used to visualize data, choosing among a lot of different libraries and representations, like bar charts, pie charts, tables, histograms, and maps [44]. It is also possible to create dashboards that aggregate what is stored by Elasticsearch in different visualizations, enabling users to drill down on data. Hence, it is the interface that analysts use to monitor their clients' infrastructure and to implement KPIs based on the data collected.

Furthermore, it offers the opportunity of detecting anomalies in the collected data thanks to the use of unsupervised machine learning features. This is particularly useful combined with the implementation of a mechanism to be notified when particular conditions match a detection rule.

Kibana is also the place where it is possible to manage all the Elastic Agent installed on every single host that the client wants to keep monitored. In fact, using the Fleet app as a control plane, the user can centrally manage agents as a fleet, having the possibility of obtaining "real-time view into agent status, remotely upgrade agents, execute queries on each host, and contain security threats" [45]. Basically, The administrator defines in Kibana policies that may apply to multiple servers; these policies are stored in Elasticsearch and when the Agent enrolls itself to the Fleet-Server, it downloads its configuration. Then it periodically checks for configuration changes.

Finally, Kibana is used to manage the security of the data collected and stored on Elasticsearch. It lets the user check each data flow to ensure that, where and when possible, the anonymization or pseudonymization of personal data are maximized, while the distribution of such data is minimized. It also helps to ensure that only authorized people have access to the data, by enabling both authentication, for example through LDAP or PKI, and access control, supporting methods like RBAC

or ABAC.

4.1.4 Beats

Beats are single-purpose lightweight data shippers written in Go. They are usually installed as a single unified Agents per host, having a single thing to install and to configure, instead of deploying multiple Beats, which guarantees a better scalability and maintenance over time [46]. In fact, usually multiple Beats are needed on a single host. Elastic Agents can also be configured centrally in Kibana through Fleet, as described in Section 4.1.3.

Beats accept inputs from different sources and write the output to either Logstash or Elasticsearch [47]. Each single Beats agent provides pre-built dashboards and parsing, based on ECS, for the most common sources through different modules. Indeed, they use and share the libbeat library, which provides output, configuration management, service and processing utilities.

There are several types of Beats, each one dedicated to ship and process a specific kind of data. Würth Phoenix SOC uses mostly:

- Filebeat, which collects logs from servers and devices. The most common sources are Apache, Squid and Paloalto, from which Filebeat gathers, for example, server logs and filestreams, TCP/UDP/Unix Sockets logs, or network devices Syslogs

- Winlogbeat, used to parse the Windows event logs, i.e. the logs that it is possible to see in the EventViewer of Windows, which include the ones about security, Powershell and Sysmon. This latter is a Windows daemon, that provides detailed information about process creations, network connections, and changes to file creation time

- Auditbeat, utilized to audit activities of users and processes on the system it is

installed on, like the changes of critical files or the installation of new packeges

- Metricbeat, that polls periodically applications to gather metrics and statistics, such as the usage of CPU and memory

- Heartbeat, applied to check if machines are up-and-running through ICMP (ping), TCP, HTTP; using the last two it is also possible to check TLS Certificate Validity

- Packetbeat, which sniffs the traffic on the Servers, capturing only the network metadata (e.g., source and destination IP address, network protocol), and correlating the requests with the responses

4.2 OpenCTI

OpenCTI is an open source platform that allows organizations to manage their cyber threat intelligence knowledge. It is useful to structure, store, organize and visualize technical and non-technical information about cyber threats that is possible to gather on the Internet. The former includes, for example, TTPs and observables, while the latter can be suggested attribution and victimology [48]. Figure 4.2 represents the main view of the OpenCTI instance managed by Würth Phoenix Cyber Security Team.

The OpenCTI platform provides a powerful knowledge management database with an enforced schema especially tailored for cyber threat intelligence and cyber operations. It offers multiple tools and viewing capabilities, making analysts able to explore the whole dataset by pivoting on the platform between entities and relations. It is also possible to set several levels of context for a given entity [49].

The most powerful feature of OpenCTI is that it can be connected to other platforms by configuring an appropriate connector, which allows both the import

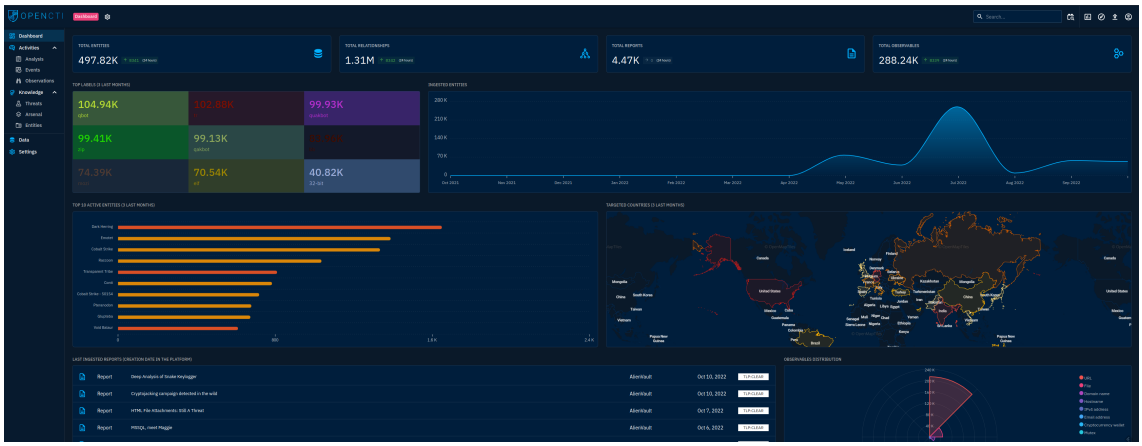

Figure 4.2: OpenCTI Main View Screenshot

and the export of data. For the purpose of the thesis, it represents the bridge between the email analyzer and Elasticsearch, setting up a connector to retrieve the IoCs regarding phishing emails and another one to push those information directly into Elasticsearch, into a dedicated index. The additional value of going through OpenCTI, instead of sending the data directly to Elastic Stack, is to further enrich end correlate the IoCs with the information retrieved from the other sources the platform is connected with.

Furthermore, the platform is queried before the IoCs are created. In fact, the blacklists indexed into OpenCTI about IPs, domains and URLs are retrieved by our email analyzer and used to determine if email senders have already been reported as malicious in the past. These lists come from sources like AbuseIPDB, Phishing Army, AlienVault and are constantly updated.

4.3 SATAYO

SATAYO stands for "Search All Things About Your Organization" and it is a platform developed by Würth Phoenix to offer the service of Exposure Assessment to

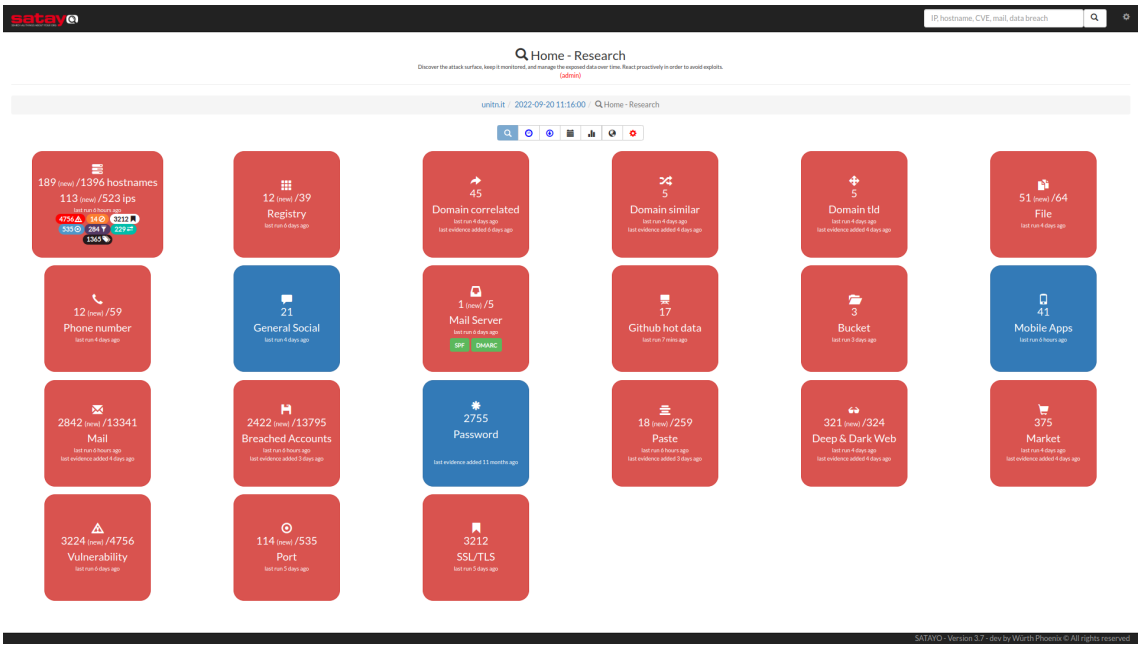

Figure 4.3: SATAYO Research Main View Screenshot

their clients. I personally contributed to the birth of the tool some years ago, during my bachelor thesis project, and I am still developing it. The service offered consists in scanning the whole Internet using an OSINT approach to detect all the possible information and access points an attacker can exploit to perform the first step inside an organization's virtual perimeter. Figure 4.3 provide an example of the items collected by SATAYO. Indeed, it aims to reproduce in the most accurate way possible the first stage of an attack, the reconnaissance, as represented in the Cyber Kill Chain, see Figure 3.1.

An Exposure Assessment starts from the company domain or some keywords and then, using different tools and techniques (like Google Dorks), it consists in gathering every evidence that can be correlated with the starting input; it then proceeds searching for additional information of those findings, going more and more in depth to create the most complete overview possible. An example of part of this process is provided by the Figure 4.4.

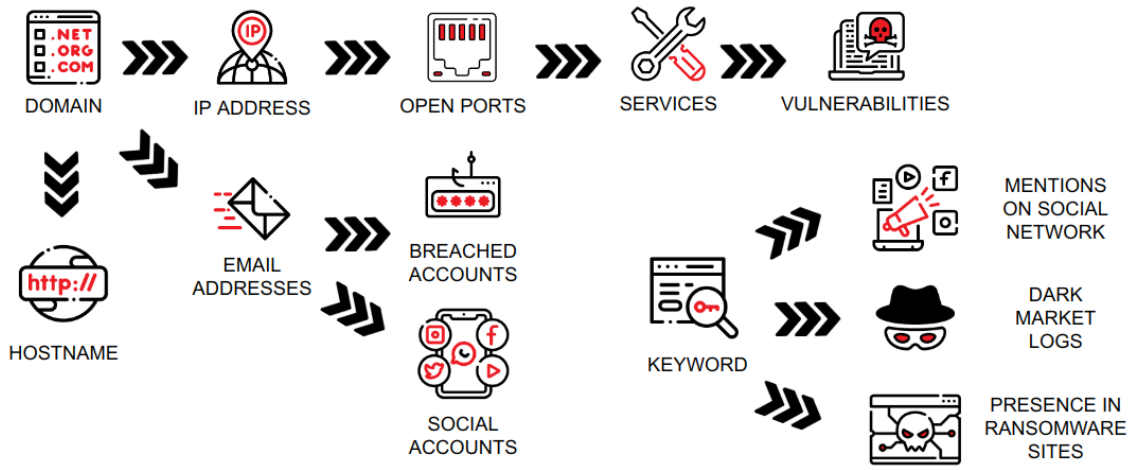

Figure 4.4: Exposure Assessment Example

Particularly relevant for the scope of the thesis project are the found domains that are somehow related to the company one. They may have been discovered because they have the same base name but a different top level domain, or because they resemble the original domain but contain a typosquatting, that means they might have a different letter, miss one, been created using punycode encoding or two letters might be reversed. In the first case, they are the result of a top level domain brute force, while in the second one of DNS fuzzing. Anyway, they are interesting since they represent a possible phishing domain if they are not legitimate. The idea is to compare the domains of the emails received with these domains to mark the emails as most likely malicious in case of matching.

Another useful information that it is possible to get from SATAYO is the list of registered email addresses. This is used in order to automatically detect if the sender is trying to impersonate a company's employee, comparing the sender email header fields and the part before the "@" of the sender email address, called username, with the one of the emails in the list. In case the domain in the received email address differs from the legitimate one, and the usernames match or the employee's name is

used to identify the sender, the email should be marked as malicious.

Finally, SATAYO enumerates all the subnet directly managed by an organization, and identifies its email servers, collecting additional information about the SPF and DMARC records. The email analyzer works with this data to understand if the email addresses of the input emails are spoofed or if these emails come from legitimate hosts.

5 Specification and Design

In the first pages of this chapter the idea behind the thesis project is explained, focusing on the needs of Würth Phoenix. It also summarizes how this solution will integrate with existing technologies described in Chapter 4. It is then explained how the email analyzer was conceived and designed, illustrating the reasoning behind the decisions made. It is also helpful to understand how all the knowledge presented in Chapters 2 and 3 is applied.

5.1 Motivation

At Würth Phoenix, they are developing a new service, the Security Operation Center, to offer a 24/7 monitoring service to their client. This thesis is about a part of it, the improvement of the detection of malicious emails, focusing especially on phishing attempts. Essentially, it is an email analyzer able to create valuable IoCs that the Würth Phoenix SOC would consume while checking the email traffic of their customers.

In fact, their infrastructure lacked something capable of generating internal evidences of phishing attempts, and the spam filter utilized was fed only with general IoCs publicly available on the Internet. The main advantage of implementing this solution is having constantly updated evidence that is also customized based on internal needs, since it is configured to analyze email traffic that simulates the one of their SOC customer.

5.2 Integration

As described in Chapter 4, the SOC environment already includes the Log Management and SIEM module of Neteye based on the Elastic Stack (Sec. 4.1), the Cyber Threat Intelligence platform OpenCTI (Sec. 4.2) and the OSINT platform SATAYO (Sec. 4.3).

In practice, the new development consists in a "Spam Trap Box" that takes advantage of the open source information collected by SATAYO and OpenCTI to evaluate the emails received and that creates IoCs to report the phishing attempts detected. This evidence is then passed to OpenCTI, which also enriches them with additional information and makes them available to the Elastic Stack. The latter uses these insights to spot attacks targeting Würth Phoenix SOC customers.

The technology developed is more or less a honeypot: basically, it is a mail server where domains previously owned by failed companies are registered. These companies used to operate in the same industry of Würth Phoenix customers, and this fact allows to operate with traffic similar to the one observed by the SIEM. In addition, any e-mail can be delivered, as no filter is placed, to have the widest possible range of samples for examination.

5.3 Conception

In this section the design of each part is described, reporting the thinking behind the decisions made. The structure of this section is similarly used in Chapter 6 to create a correlation between the design and the implementation of every components.

5.3.1 Spam Trap Box

The first challenge faced was understanding how to start receiving the required amount of emails. There were two possibilities, the first one would have been to

register a lot of email addresses with simple usernames, to make them be easily found by spammers. In addition, to give them even more visibility, it would have been useful to subscribe to different forums and websites using these emails. Nevertheless, considering the time required for this option to be ready and its uncertain outcome, I opted for the second one.

It was suggested to me by Corsin Camichel, a cyber security researcher who also had worked on his own spam trap box. It consists in searching for domains that once belonged to real companies that now are either bankrupt or have changed name. Indeed, by using domains that were previously in use, it is possible to get spam and malware practically within days of setting up the traps. This solution saved the time that would otherwise have been spent on disseminating and giving visibility to some new e-mail addresses specifically created to set up the spam trap box.

Of course, the more email addresses they have connected, both leaked or spread online, the more worthwhile it is acquiring them. Furthermore, for the purpose of the project, it is better to search for companies that operate in the same fields of Würth Phoenix clients. In fact, it is common that a phishing campaign targets or is reused for more than one company in the same sector of interest.

To detect the domain to buy, since Würth Phoenix is an Italian company, it was necessary to scrape the FALLCO portal, where all the documents and information about Italian judicial liquidations and corporate bankruptcies are stored [50]. This resulted in a list of interesting companies from which to search for possible domains to acquire. The most recently failed companies still owned their domain, so the research focused mainly on the legal procedures occurred between 2017-2020.

A subsequent skimming of the list was performed by considering how many emails registered under those domains have leaked online. To verify it, the Intelligence X search engine called Phonebook.cz [51] was employed. It allows a user to search

for subdomains, email addresses, or URLs starting from the given input domain. Only domains for which more than ten e-mail addresses were found were considered valuable.

The final step was checking on the Italian register website [52] if these domains were for sale. Some were even sold for free, hence one of these was purchased to develop the first test.

5.3.2 Analyzer

The following step was thinking about how to automate the analysis of the email arrived to the spam trap box. The simplest and fastest approach to allow a program to iterate over mailboxes is to store the credentials in a database, avoiding writing them into the code. In addition, a database can be useful during the debugging and testing phases of the program to store the results and manually verify that the program is working properly before sharing the outcomes with OpenCTI.

Next, Python was picked as programming language to develop the code whose aim is to retrieve and analyze the emails and, in case, also create the necessary IoCs. In fact, it provides some useful libraries that suit perfectly for the purpose of the thesis project.

The process starts with the download of each email: using the "imaplib" library and the credentials stored into the database, it is possible to interact with each user's mailbox and look for the new emails. It is important to keep everything in order, otherwise there is the possibility of working also with the already seen messages. For this purpose, the read emails must be moved from the "INBOX" to another folder. It is not a good move to delete them after the analysis, otherwise it would be impossible to reuse them in case new controls are added to the analysis or if something went wrong during the inspection, especially in the first phase of testing and debugging.

The core of the project was indeed to determine what information might reveal whether an email is malicious or not. For sure, the data contained into the header of the messages are the most useful, since they disclose the most about the sender and the origin of the email. For example, some clues are the presence in the "FROM" header field of a string which is intended to identify the sender and which is not an email, or if there is more than one email address but the "SENDER" header is not set; both cases usually indicate a sender obfuscation attempt.

Moreover, the SPF, DKIM and DMARC records related to the sender domain are checked, if they are set, to determine if they are valid or not. A negative result may suggest that the sender email address is spoofed. However, also the absence of this records related to the sender domain may indicate that the sender is not to be trusted. In fact, a company usually cares about these mail server configuration, to prevent the spoofing of their corporate email addresses.

In addition, the content of the "SUBJECT" header field is often another important hint to take into consideration. In fact, in case of phishing emails it commonly contains words that aim to make the user open the message and click on the link inside it or download the attachment, if there is one. The latter are also inspected to detect any eventual malware.

Finally, based on Section 3.1, the content of the message should also be taken into consideration, to verify if there are any IP-based URLs, links to recently registered domains, a high number of links or any other feature that may suggest that it is a phishing email.

Most of the data can be extracted using "meioc" (Mail Extractor IoC) [53], which is an open source project published on GitHub by Andrea Draghetti, an Italian cyber security researcher, known especially for his work to contrast phishing. I personally contributed to the project, fixing some bugs and implementing some new parts, such as the extraction of the DKIM and DMARC check results or the identification of

text in the "FROM" email header that is not an email address, added to perform sender obfuscation and impersonation.

Some checks involve the use of OSINT external sources, for example the one performed on the source IP addresses, or the one on the domain of the sender's email address, that are compared with publicly available blacklists. This means that the outcome strongly depends on the quantity and quality of the information gathered and supplied by SATAYO and OpenCTI: the better these two work, the more accurate will the results be. Anyway, this is the element of innovation introduced by this thesis: to analyze the parts of the email both by themselves and in comparison with OSINT evidences to get the best verdict possible.

In the end, if the email is considered a malicious one, an IoC is created. It contains the sender email address, the origin IP address, the subject of the email and eventually the link written inside the message text or the signature of the attachment. These elements will be added to the ones of the same category already collected by OpenCTI, and be ingested and indexed by the Elastic Stack in order to improve the filter on clients' email traffic.

6 Implementation and Verification

This chapter aims to explain how the thesis project was developed, going into implementation details. The email analyzer is currently a prototype that will be further developed at the company to add more features and to improve the functionality based on the findings in this thesis.

Anyway, the whole process is taken into consideration, starting from the identification of interesting domains to buy, moving on the code responsible for downloading the emails received, extracting the data considered more valuable for our purpose, and performing the analysis of such data. Indeed, all the libraries, tools and APIs integrated are presented, highlighting the changes made to some of these open source programs to accommodate our needs. In the end, a test case is taken into analysis, to point out how the first version of the email analyzer performs and to understand what it is possible to improve.

6.1 Spam Trap Box Configuration

The first step of the thesis project was to set up a spam trap box, which means to create a mail server that does not have any filters on the incoming traffic, letting every message to reach the "INBOX" folder.

Following the process described in Section 5.3.1, a single domain was chosen. It was previously owned by a company that operated in the sector of building and installations, and Würth Phoenix has some costumers working in the same industry.

A Postfix-like mail server was configured registering all the 21 email addresses found on Phonebook.cz [51], most of which have an employee's "surname.name" as username, while some others are more administrative ones and have for example "reception" or "info" as username.

As it was expected, in a few days a sufficient amount of emails to start thinking about how to conduct the analysis was collected.

6.2 Analyzer Development

The core of the thesis was the development of the email analyzer. It can be divided in two parts, the creation of the database and the coding of the program. The latter was written in Python and has itself been divided here into several parts, so as to facilitate both explanation and understanding.

6.2.1 Database

The database was designed to be as simple as possible; it consists of only 5 tables, in which all the data about the users created and their emails are stored. A UML of the database is reported in Figure 6.1. Except for the "account" table, all the other ones were created just as proof of the fact that the program is working correctly; this means that they are read-only tables consulted when needed. In fact, IoCs will be created and sent directly to OpenCTI.

In the "account" table are stored the credential to access to each different user space created on the mail server. This was necessary to implement the automated access to mailboxes on the server and the subsequent download of all the messages in the "INBOX".

The "email" table is the place where the most important data extracted from the analyzed messages are stored. In addition to the eml hash and the eml itself,

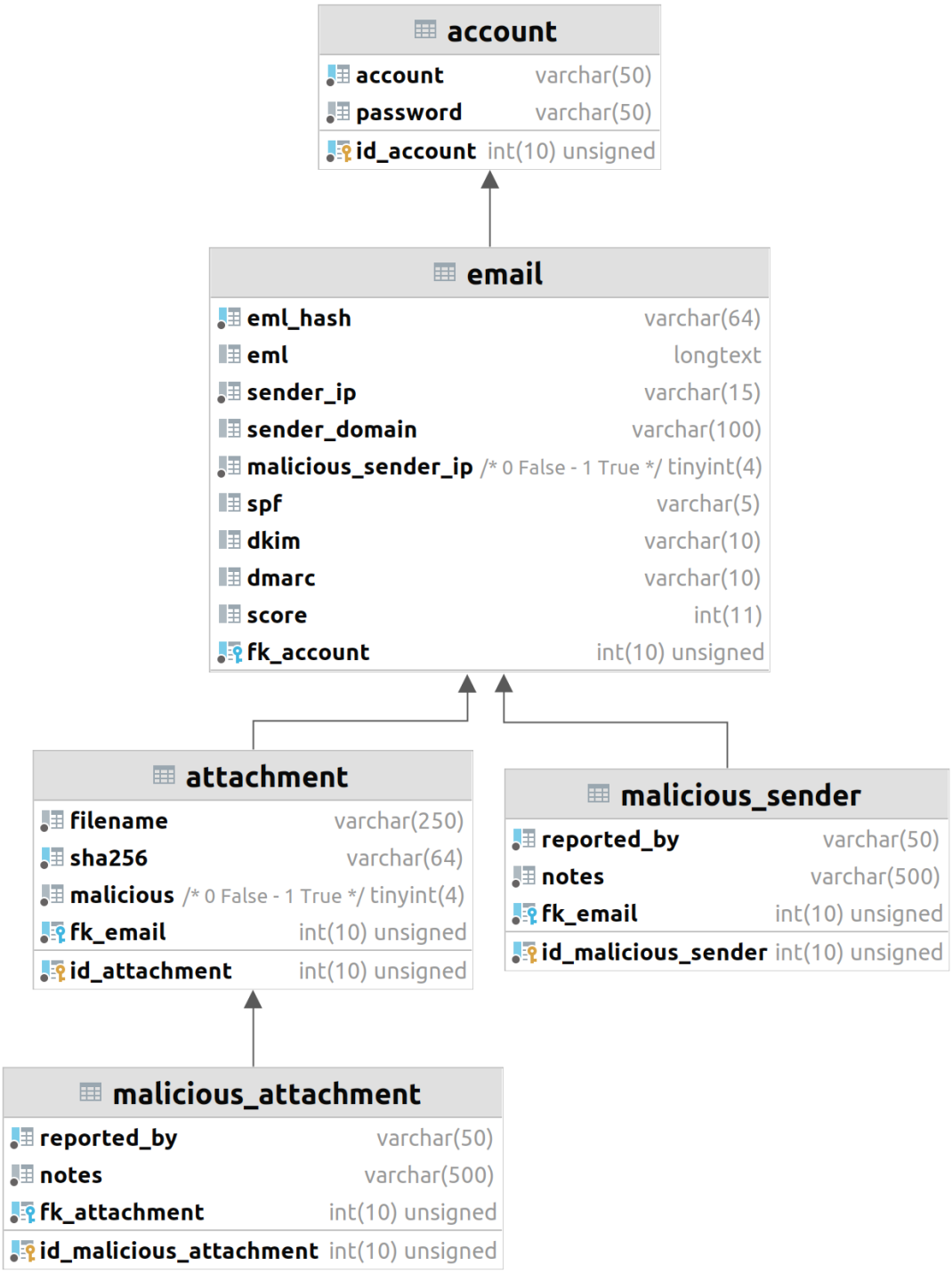

Figure 6.1: Database UML Representation

the IP address and the domain of the sender are saved. Furthermore, there are the results of the SPF, DKIM and DMARC records verification, as well as two fields that summarized the outcomes of the analysis, which are a flag that tells if the sender if malicious or not and a score that indicates how the email is evaluated, from 0 to 100. The higher, the more malicious the message is to be considered. Finally, every email is connected to the user from which it was downloaded.

The table "attachment" contains, instead, the information about any attachments to each email. These are the filename, the hash obtained using the sha256 algorithm and a flag that reported if the file was already reported as a malware. Of course each entry of this table is related to one email in the "email" table.

The remaining two table "malicious sender" and "malicious attachment" archive additional information about, respectively, a malicious sender or attachment, indicating the source where the report was found and any further comments extracted from the source.

6.2.2 Emails Download

The first part of the Python program is responsible for downloading the messages stored on the email server. It initially gets the list of accounts and credentials from the database, and then it starts a cycle over this list. Thanks to the "imaplib" library, it is possible to interact with each user's mailboxes, after having established a TLS connection.

```
1 imap_host = 'mail.sec4u.co'
2 list_account = select_all(select_list_account)
3 for account in list_account:
4     id_account = account[0]
5     imap_user = account[1]
6     print('\n========= ' + imap_user + ' ==============\n')
7     imap_pass = account[2]
```

```
8
9       # connect to host using SSL
10      imap = imaplib.IMAP4_SSL(imap_host)
```

Snippet 6.1: Server Connection

After the connection is up, the program logs into the user space, using the "LOGIN" IMAP command. Then, it searches among all the mailboxes, through the "LIST" command, for the folder "Analyzed". In case it is not present, it is created using the "CREATE" command. This folder is important because it is the place where messages are moved to after been analyzed, so that in the "INBOX" there will always be only new messages that have still to be evaluated.

```
1 # login to server
2       resp_code, response = imap.login(imap_user, imap_pass)
3
4       # search and eventually create the Analyzed mailbox
5       resp_code, directories = imap.list(pattern="Analyzed")
6       if "Analyzed" not in directories:
7           imap.create(mailbox="Analyzed")
```

Snippet 6.2: Login & "Analyzed" Mailbox Creation

Subsequently, all the directories are listed, and the list is printed showing how many emails there are in each folder. The number of messages is obtained by running the "SELECT" command and reading the "EXISTS" response value for each mailbox.

```
1 resp_code, directories = imap.list()
2
3       print("\n========= List of Directories ================\n")
4       for directory in directories:
5           directory_name = directory.decode().split('"')[-2]
6           directory_name = '"' + directory_name + '"'
```

```
        resp_code, mail_count = imap.select(mailbox=directory_name, readonly=True)
        print(directory_name, ' - ', mail_count[0].decode())
```

Snippet 6.3: List all the Mailboxes

Finally, the "INBOX" folder is selected again, and using the "SEARCH" command it is possible to obtain the message identification number associated to every email here stored. The program iterates over this number, and retrieves each message using the "FETCH" IMAP command. Data extraction then begins.

```
    #Select Directory
    resp_code, mail_count = imap.select(mailbox="INBOX", readonly=False)

    #Retrieve Mail IDs for given Directory
    resp_code, mails = imap.search(None, "ALL")
    print("\n================================\n")
    print(f"Mail IDs : {mails[0].decode().split()}\n")

    #Analyze Messages for given Directory
    for mail_id in mails[0].decode().split():
        print(f"===== Start of Mail [{mail_id}] ====\n")
        ## Fetch mail data.
        resp_code, mail_data = imap.fetch(mail_id, '(RFC822)')
```

Snippet 6.4: Email Download

6.2.3 Data Extraction

As said in Section 5.3.2, the interesting data are extracted from a message using "meioc" [53]. It requires in input an email provided as eml file, and it saves the output on a json file.

```
        # Construct eml from mail data
```

```
        message_eml = email.message_from_bytes(mail_data[0][1])
        eml_hash =  hashlib.sha256(str(message_eml).encode('utf-8')
    ).hexdigest()
        # Saving the eml as a file to pass it as input to maioc
        with open("mail.eml", 'w') as mail_file:
            gen = generator.Generator(mail_file)
            gen.flatten(message_eml)
        # Run meioc
        os.system(f'python3 {meioc_path} --spf --dkim -x mail.eml -
    o meioc_results.json')
        json_file = open('meioc_results.json')
        meioc_results = json.load(json_file, strict=False)
```

Snippet 6.5: Data Extraction

The extracted data used during the analysis are:

- the eml of the message, extracted using the "email" Python library
- the hash of the eml, calculated through the "hashlib" Python library
- the sender IP address, taken from the list of relay IP address return by meioc
- the SPF check result, whose value is "True" if the check outcome is "pass", "False" otherwise
- the DKIM check result, whose value is determined as the SPF one
- the DMARC check result, as above
- the attachments data, which includes the filename and the sha256 extracted by meioc and the payload extracted instead thanks to the "email" Python library
- the possible displayed name written in the "FROM" header field

- all the email addresses written in the "FROM" header field
- the value of the "SENDER" header field, whether present
- the value of the "RETURN PATH" header field, whether present
- the value of the "REPLY-TO" header field, whether present
- the email receipt date
- the value in the "SUBJECT" header field
- the list of URLs in the message

6.2.4 Data Analysis

This is the most important section of the code, even if it is not completely implemented yet. Calibrating the influence given to each aspect analyzed on the final outcome requires a lot of time and attention, since the aim is to limit false positives as much as possible. Hence I sketched out a temporary score system that needs to be reviewed with my team supervisors. For these reasons, in the current section there is no code snippet, but rather everything is explained from a theoretical point of view, as it is thought to be once it will be completed and fully integrated into the SOC infrastructure.

It is possible to identify two approaches implemented to analyze the data extracted from an email. One consists in comparing header fields between each other, or looking at the value of one field by itself, in order to detect inconsistencies, suspicious information or structures that are not compliant with the RFC specifications. The other, instead, involves the use of OSINT sources to compare what is taken from the email with already existing indicators of compromise, to create new ones of these latter providing additional information or evidence.

The first check is performed on the eventual name written in the "FROM" email header field. Its presence is already somewhat suspicious, due to the fact that the RFC 5322 [2] says that the value of this header is a list of one or more email addresses, as explained in Chapter 2.1. Anyway, if present, this name is displayed in the email client, often replacing the following email. Hence, this verification consists in checking if the name is contained in the username of any of the following emails in this header field. If this is the case, it is considered acceptable. Otherwise, the email score representing how malicious the email is considered is increased.

Then, the "SENDER" header field value is also taken into consideration. This header is often not included in an email, since usually the address specified in the "FROM" header is also the real sender of the email, so adding also the "SENDER" header would create useless redundancy. Hence, if present, the check consists in understanding if its value is compliant with the RFC 5322 [2] specification. In case of negative response, the malicious score is increased, but with a very low incidence. In fact, this check alone is pretty much useless to identify phishing emails, but it can assume a value if considered along with other factors.

The moment in which the email was received is also a factor that has to be evaluated. Indeed, an attacker wants to contact the victims when they are more vulnerable, which means when they are isolated and not, for example, in the office, since we are considering phishing campaigns that target a company. Thus, emails can be considered suspicious if received during the weekend or out of the office hours. When this is the case, the score is increased.

The last check that does not involve any OSINT sources is the one performed against spoofing. Inspecting the output of meioc, the program verifies the validity of the DMARC, DKIM and SPF records, giving more importance to the DMARC evaluation, since this is the strongest check and includes also the DKIM and SPF verification, as described in Chapter 2.3.3. Anyway, each failure increases the score,

even if in different ways.

The first check performed using the information gathered by OpenCTI is the one over the sender IP address. In fact, it is verified that the IP has not already been reported in any blacklist that is publicly available and indexed in OpenCTI. The main sources are AlienVault and AbuseIPDB. Furthermore, the API of VirusTotal is called to make sure that the IP has not been notified there. Any match influences the score.

Subsequently, all the emails in the "FROM", "SENDER", "REPLY TO" and "RETURN PATH" email headers are analyzed. Firstly, each email username is compared to the list of employees found by SATAYO, since it may suggest an impersonation attempt, in case the domain does not match the company one. Moreover, in the same case just described, despite the result of the username verification, the domain is searched among the ones indexed by OpenCTI and SATAYO. The former collects domains that have previously been blacklisted by any other publicly available source; the primary source for this list is Phishing Army. The latter instead stored all the domains that are similar to the monitored one, which means that they are different for example because of a typosquatting. As before, any match raises the score.

The value of the "SUBJECT" header is analyzed using a list of words that are often included in this header when the email is a phishing attempt. Each word is searched inside the header value, which is commonly a short sentence. In a phishing scenario, the sentence invites the reader to open the email and download the attachment or click on the link written in the message, usually so as not to miss an opportunity or to check their bank account. Again, in case of any match, the score is increased.

Any attachment is then controlled using VirusTotal. Since the content might be legitimate and contain sensitive information, even if the spam trap box is built

using domains that do not belong to real businesses anymore, it is a good practice not to upload the whole file on the platform, but only its hash. Indeed, any user that bought a VirusTotal subscription can access any document uploaded. Using the hash, it is possible to check if it matches the one of any malware known by the platform. Same as in the previous cases, a positive result affects the score.

Last but not least, every URL extracted from the message is parsed, in order to verify if they include an IP address, a suspicious number of points, domains recently created or if they are contained in any blacklist. In addition to these features, also the number of links contribute to the incidence of this check on the final score.

When all checks are completed, the score is evaluated and if it is higher than an expected threshold calculated on the basis of the fields that it was possible to verify, the email is considered malicious, and an IoC created. This is added to a JSON file, where each element is a different IoC that reports the IP address, the domain of the sender email address, the value of the "SUBJECT" header field and the hash of any attachment. This JSON is then retrieved by OpenCTI using a connector, with the purpose of ingesting the new evidence. Subsequently, this information will be used by the SOC, which will have to block any further emails intended for Würth Phoenix customers having that IP or domain in the header fields reporting the sender data.

Finally, the analyzed email is moved to the "Analyzed" mailbox using the combination of the IMAP commands "COPY", "STORE" and "EXPUNGE". Anyway, the latter is run only after that the cycle on every email inside the "INBOX" ends, to delete all the email there at once. Lastly, the IMAP command "LOGOUT" is executed, to proceed with the next account.

```
1        imap.copy(mail_id, "Analyzed")
2        resp_code, response = imap.store(mail_id, '+FLAGS', '\\
   Deleted')
3
4        print(f"\n====== End of Mail [{mail_id}] ======\n")
```

```
5
6     response = imap.expunge()
7
8     imap.logout()
```

Snippet 6.6: Email Relocation and Logout

6.3 Detection Simulation

After five weeks of execution, the first version of the email analyzer generated 367 IoCs about phishing emails, after having analyzed more than a thousand of emails, 1146 to be precise. Checking manually all the ones for which an IoC was created, it was possible to assure that only 314 were actually to be reported. Furthermore, among all the others, 21 malicious messages were not detected.

At Würth Phoenix, part of the SOC is specialized in performing simulations of attacks to test both the infrastructure and the personnel of their costumers. Hence, they are used to perform phishing campaigns against a wide range of different targets. All their samples were sent to one of the accounts registered inside the spam trap box, in order to let the program analyze them.

Observing both how these new samples are rated and the evaluation errors previously identified, it is possible to state that the program is giving too much incidence to the structural analysis of the email headers, where it is performed the check to test if they are RFC complaint. On the other hand, more importance should be given to the analysis of both the "SUBJECT" header value and the URLs extracted, since 9 out of the 11 not identified cases contain in this header field at least one word present in the list of the typical phishing subject redacted on OpenCTI or a URL itself is contained in a blacklist.

7 Conclusion

Summing up the whole thesis, the task here addressed is creating an email analyzer able to provide constantly new evidence of phishing attempts. This is to help keep the Würth Phoenix SOC up-to-date with the latest information on cyber attacks. The innovation introduced is the combination of email structural validation with OSINT evidence comparison, making it possible to go beyond a mere text analysis. The necessary open source information is previously collected from different sources by other platforms that are part of the SOC, such as SATAYO and OpenCTI.

I contributed both theoretically and practically. First, I thought about what information would be most useful to take into consideration and to analyze, and I found out the best way to create the spam trap box. Then I wrote the code for the entire analyzer. Of course, my supervisor helped me in case I had a doubt or I needed to choose between more options or implementations. When I started, they suggested me some ideas, for example how to identify domains previously owned by failed companies, or tools from which I could have taken advantage, like meioc, but the whole project was completely assigned to me and I was given full decision-making autonomy, especially since I had already demonstrated my expertise by developing SATAYO with them.

Based on what is explained in Chapters 2 and 3 and on how the final score changes while adjusting the weight of each check, it emerges that the main features of an email that reveal if it is a phishing attempt are:

- the value of the headers that contain information about the sender
- the value of the "SUBJECT" header
- the validity of the DMARC record
- the domain in the URLs of the message

Moreover, the open source information that turned out to be the most useful in identifying ongoing phishing campaigns are the blacklisted domains, but also the fuzzed domains that are correctly registered and really similar to a company's one, the blacklisted IP addresses.

These data can be combined in order to get the most accurate assessment possible. It is worthwhile to search for the domains extracted from the sender headers or the URLs in the message within the list of blacklisted and fuzzed domains. Similarly, the sender IP address should be searched within the list of blacklisted IP addresses. These checks should affect the final evaluation the most.

Immediately after, the greatest importance should be given to the value in the "SUBJECT" header, which can also be compared with a list of known phishing words or phrases, and to the validity of the DMARC record. All the other features taken into consideration should be considered as additional information able to increase or decrease the score, but alone they are pretty much useless to identify phishing attempts.

In the end, it is possible to consider the overall thesis as a success. The suggestion of looking for domains previously owned by another company let me save a lot of time, and adds the value of considering phishing campaigns whose targets are business sectors of interest. Furthermore, the outcomes are quite good, even if the number of false positive was too high. In fact, 367 emails were identified as phishing attempts, after having analyzed 1146 samples, but only 314 were actually to be reported. Additionally, among all the others, 21 malicious messages were not

detected.

I think that the problem lies more in the scoring system than in the verification performed, hence the work is proceeding in the right direction. Perhaps it might be useful to reconsider how to score, instead of simply adjusting the incidence of each check, and add more sophisticated text inspection.

Hence, after the necessary improvements, starting from what is suggested in the end of Chapter 6.3, and the appropriate verification, the next step is to create the connector on OpenCTI to make it ingest the IoCs created by the analyzer. Consequently, the SOC will benefit from this new source of malicious evidences, created on the bases of its particular needs.

To conclude, I think that this work can be imported easily into any environment able to provide the OSINT evidences needed for the analysis. In fact, it does not depend on the technologies adopted in Würth Phoenix, but it was adapted to them. The only thing to keep in mind is that the more pertinent the data furnished as input to the program are, the more valuable the outcomes will be, and the same holds for the domain chosen to configure the spam trap box.

References

[1] IBM, *Enterprise security solutions*. [Online]. Available: `https://www.ibm.com/security` (visited on 10/15/2022).

[2] P. Resnick, *Internet Message Format*, RFC 5322, Oct. 2008. [Online]. Available: `https://www.rfc-editor.org/info/rfc5322`.

[3] N. Freed and N. S. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC 2045, Nov. 1996. [Online]. Available: `https://www.rfc-editor.org/info/rfc2045`.

[4] N. Freed and N. S. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*, RFC 2046, Nov. 1996. [Online]. Available: `https://www.rfc-editor.org/info/rfc2046`.

[5] K. Moore, *MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text*, RFC 2047, Nov. 1996. [Online]. Available: `https://www.rfc-editor.org/info/rfc2047`.

[6] J. C. Klensin and N. Freed, *Media Type Specifications and Registration Procedures*, RFC 4288, Dec. 2005. [Online]. Available: `https://www.rfc-editor.org/info/rfc4288`.

[7] J. C. Klensin and N. Freed, *Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures*, RFC 4289, Dec. 2005. [Online]. Available: `https://www.rfc-editor.org/info/rfc4289`.

[8] N. Freed and N. S. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples*, RFC 2049, Nov. 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc2049.

[9] A. Yang, S. Steele, and N. Freed, *Internationalized Email Headers*, RFC 6532, Feb. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6532.

[10] J. C. Klensin, *Simple Mail Transfer Protocol*, RFC 5321, Oct. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5321.

[11] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-down Approach*. Boston, MA, USA: Pearson, 2016.

[12] SecPoint, *What is an Open Mail Relay?* [Online]. Available: https://www.secpoint.com/what-is-an-open-relay.html (visited on 08/25/2022).

[13] JANET, *Repairing open mail relays*. [Online]. Available: https://web.archive.org/web/20080224195334/http://www.ja.net/services/mail/janet-spam-relay-tester-and-notification-system/repairing-open-mail-relays.html (visited on 08/25/2022).

[14] M. T. Rose and J. G. Myers, *Post Office Protocol - Version 3*, RFC 1939, May 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1939.

[15] A. Melnikov and B. Leiba, *Internet Message Access Protocol (IMAP) - Version 4rev2*, RFC 9051, Aug. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9051.

[16] S. Kitterman, *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*, RFC 7208, Apr. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7208.

[17] M. Kucherawy, D. Crocker, and T. Hansen, *DomainKeys Identified Mail (DKIM) Signatures*, RFC 6376, Sep. 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6376.

[18] M. Kucherawy and E. Zwicky, *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*, RFC 7489, Mar. 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7489.

[19] J. R. Levine, *Email Authentication for Internationalized Mail*, RFC 8616, Jun. 2019. [Online]. Available: https://www.rfc-editor.org/info/rfc8616.

[20] A. Almomani, B. B. Gupta, S. Atawneh, A. Meulenberg, and E. Almomani, "A Survey of Phishing Email Filtering Techniques", *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2070–2090, 2013.

[21] A. Ferreira, L. Coventry, and G. Lenzini, "Principles of Persuasion in Social Engineering and Their Use in Phishing", in *Human Aspects of Information Security, Privacy, and Trust*, T. Tryfonas and I. Askoxylakis, Eds., Cham, CH: Springer International Publishing, 2015, pp. 36–47.

[22] P. Kalaharsha and B. M. Mehtre, "Detecting Phishing Sites - An Overview", *CoRR*, vol. abs/2103.12739, pp. 1–13, 2021. arXiv: 2103.12739. [Online]. Available: https://arxiv.org/abs/2103.12739.

[23] A. Draghetti, *Phishing: Tecniche e strategie di un fenomeno in evoluzione*, 2019. [Online]. Available: https://www.andreadraghetti.it/wp-content/uploads/2020/08/2020.04.23-Tesi-Andrea-Draghetti-Web.pdf.

[24] I. Fette, N. Sadeh, and A. Tomasic, "Learning to Detect Phishing Emails", in *Proceedings of the 16th International Conference on World Wide Web*, New York, NY, USA: Association for Computing Machinery, 2007, pp. 649–656. [Online]. Available: https://doi.org/10.1145/1242572.1242660.

[25] D. Lain, K. Kostiainen, and S. Čapkun, "Phishing in Organizations: Findings from a Large-Scale and Long-Term Study", in *2022 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA, 2022, pp. 842–859.

[26] D. Jampen, G. Gür, T. Sutter, and B. Tellenbach, "Don't click: towards an effective anti-phishing training. A comparative literature review", *Human-centric Computing and Information Sciences*, vol. 10, pp. 1–41, Dec. 2020.

[27] P. Burda, L. Allodi, and N. Zannone, "Don't Forget the Human: a Crowd-sourced Approach to Automate Response and Containment Against Spear Phishing Attacks", in *2020 IEEE European Symposium on Security and Privacy Workshops(EuroS&PW)*, Los Alamitos, CA, USA, Sep. 2020, pp. 471–476.

[28] T. Moore and R. Clayton, "Evaluating the Wisdom of Crowds in Assessing Phishing Websites", in *Financial Cryptography and Data Security*, G. Tsudik, Ed., Berlin, DE: Springer Berlin Heidelberg, 2008, pp. 16–30.

[29] A. Aleroud and L. Zhou, "Phishing environments, techniques, and countermeasures: A survey", *Computers & Security*, vol. 68, pp. 160–196, 2017. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0167404817300810`.

[30] Y. Huo, Y.-W. Hwang, I.-Y. Lee, H. Kim, H. Lee, and D. Kim, "Current Status and Security Trend of OSINT", *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–14, 2022. [Online]. Available: `https://doi.org/10.1155/2022/1290129`.

[31] D. Lande and E. Shnurko-Tabakova, "OSINT as a part of cyber defense system", *Theoretical and Applied Cybersecurity*, vol. 1, no. 1, pp. 103–108, 2019.

[32] J. Pastor-Galindo, P. Nespoli, F. Gómez Mármol, and G. Martínez Pérez, "The Not Yet Exploited Goldmine of OSINT: Opportunities, Open Challenges and Future Trends", *IEEE Access*, vol. 8, pp. 10 282–10 304, 2020.

[33] Justin Nordine, *OSINT framework*. [Online]. Available: `https://github.com/lockfale/osint-framework` (visited on 09/17/2022).

[34] M. M. Yamin, M. Ullah, H. Ullah, B. Katt, M. Hijji, and K. Muhammad, "Mapping Tools for Open Source Intelligence with Cyber Kill Chain for Adversarial Aware Security", *Mathematics*, vol. 10, no. 12, pp. 1–25, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/12/2054.

[35] fastfire, *deepdarkCTI*. [Online]. Available: https://github.com/fastfire/deepdarkCTI (visited on 09/17/2022).

[36] CrowdStrike, *IOA VS IOC*, May 2021. [Online]. Available: https://www.crowdstrike.com/cybersecurity-101/indicators-of-compromise/ioa-vs-ioc/ (visited on 09/18/2022).

[37] F. Jáñez-Martino, E. Fidalgo, S. González-Martínez, and J. Velasco-Mata, "Classification of Spam Emails through Hierarchical Clustering and Supervised Learning", *CoRR*, vol. abs/2005.08773, pp. 1–4, 2020. [Online]. Available: https://arxiv.org/abs/2005.08773.

[38] U. Murugavel and R. Santhi, "Detection of spam and threads identification in e-mail spam corpus using content based text analytics method", *Materials Today: Proceedings*, vol. 33, pp. 3319–3323, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785320333903.

[39] F. Jáñez-Martino, R. Alaiz-Rodríguez, V. González-Castror, E. Fidalgo, and E. Alegre, "A review of spam email detection: Analysis of spammer strategies and the dataset shift problem", *Artificial Intelligence Review*, vol. 56, pp. 1145–1173, 2023.

[40] Elastic, *What is the ELK Stack?* [Online]. Available: https://www.elastic.co/what-is/elk-stack (visited on 09/25/2022).

[41] Elastic, *What is Elasticsearch?* [Online]. Available: https://www.elastic.co/what-is/elasticsearch (visited on 09/27/2022).

[42] Elastic, *Logstash*. [Online]. Available: https://www.elastic.co/logstash (visited on 09/30/2022).

[43] M. Settle and M. Martin, *Introducing the Elastic Common Schema*, Feb. 2019. [Online]. Available: https://www.elastic.co/blog/introducing-the-elastic-common-schema (visited on 09/30/2022).

[44] Elastic, *What is Kibana?* [Online]. Available: https://www.elastic.co/what-is/kibana (visited on 10/01/2022).

[45] J. Skowronski, *Elastic Agent and Fleet make it easier to integrate your systems with Elastic*, Aug. 2021. [Online]. Available: https://www.elastic.co/blog/elastic-agent-and-fleet-make-it-easier-to-integrate-your-systems-with-elastic (visited on 10/01/2022).

[46] Elastic, *Elastic Agent*. [Online]. Available: https://www.elastic.co/elastic-agent/ (visited on 10/02/2022).

[47] Elastic, *beats*. [Online]. Available: https://www.elastic.co/beats/ (visited on 10/02/2022).

[48] Filigran, *OpenCTI*. [Online]. Available: https://github.com/OpenCTI-Platform/opencti (visited on 10/08/2022).

[49] Filigran, *OpenCTI*. [Online]. Available: https://www.filigran.io/en/products/opencti (visited on 10/08/2022).

[50] FALLCO, *Portale dei Creditori*. [Online]. Available: https://www.portalecreditori.it (visited on 10/11/2022).

[51] Intelligence X, *Phonebook.cz*. [Online]. Available: https://phonebook.cz (visited on 10/11/2022).

[52] *Register.it*. [Online]. Available: https://www.register.it (visited on 10/11/2022).

[53] A. Draghetti, *meioc*. [Online]. Available: `https://github.com/drego85/meioc` (visited on 10/13/2022).