UNIVERSITÀ
DI TRENTO

UNIVERSITY
OF TWENTE.

Department
of Information Engineering and Computer Science

Master's Degree in

Computer Science

FINAL DISSERTATION

IOCMonitor: Automatic extraction of cyber threat intelligence from open source data using NLP and Machine Learning

SupervisorUniTrento
Ivan Pashchenko

SupervisorUniPartner
Florian Hahn

Student
Miguel García-Mauriño, 219750

Co-supervisor (Fondazione Bruno Kessler)
Biniam Fisseha Demissie

ACADEMIC YEAR 2020/2021

Acknowledgments

First, I want to thank my research co-supervisor, Biniam Fisseha Demissie, for his dedication, patience and involvement. Without him, this work would not have been accomplished. I would also like to thank my supervisor, Ivan Pashchenko, for his help and orientation.

I want also to thank my friends and family, for their emotional support all along my studies, and especially to Mar a Molina for her unconditional support and love.

Abstract

In cyber security, having reliable, updated information, in the form of indicators of compomise, is critical for enhancing security and resilience. This information can be found on publicly available sources such as social media and blog publications. However, those publications are meant for other people to read, thus they are written in natural language and cannot be easily parsed by a piece of software. Indicators of compromise usually have a specific format that allows using regular expressions to find them, at the cost of a large amount of false positives. This work presents IoCMonitor*, a software system for IoC extraction that takes in account the text in which the IoC is presented and uses the context to validate it, in order to minimize the false positives. Experimental results show* IoCMonitor *performs well, with a high precision above 95%.*

Keywords

[Indicator of Compromise, Cyber Threat Intelligence, Natural language Processing, Machine Learning, cyberdefense]

Declaration of Originality

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given in the bibliography.

(Miguel García-Mauriño)

Contents

List of Figures

List of Tables

Chapter 1

Summary

Lately, there seems to be an increasing trend towards cyber attacks. Cybercriminals bypass protections and spread malware through systems, sometimes even compromising critical infrastructure such as the infamous Colonial Pipeline ransomware attack in the U.S.

The wide success of cyber attacks suggests current security measures could be further improved. One possible improvement on such security measures involves maintaining them as much updated as possible. To do so, it is important to gather timely information about ongoing or recent attacks, in order to be able to block or respond to the same attack in case it targets new organizations. This information is called Cyber Threat Intelligence (CTI) and might have different forms. Here, we focus on the Indicator of Compromise (IoC) form, which provides information about how to recognize an attack using information like e.g the hash value of its executable.

Indicators of compromise are usually found by security researchers and later published for others to read and use. Indeed, they are usually published in platforms like social media or blogs. This is useful for other researchers to read, but makes it hard to effectively include that information into the security measures of an organization. The reason for that is because it is written using natural language, which is informal and has a high level of abstraction. This makes it difficult to gather Indicators of Compromise from such platforms, but as stated above, this kind of CTI must be acquired as quickly as possible. Thus, some effort is worth to be made.

In this work, we aim to design and implement a software system (*IoCMonitor*) that effectively extracts IoCs from blog posts. To do so, we leverage the specific format of such IoCs to find candidates using regular expressions, and further process the post we found them in, using machine learning and natural language processing methods. The aim of this additional processing is to assess whether a candidate IoC is actually relevant for Cyber Threat Intelligence matters or not. Overall, we try to minimize the false positives, i.e the pieces of information that we report as Indicators of Compromise but are actually harmless.

In order to achieve that, we designed *IoCMonitor*, a software system that works as a pipeline: In the early stages, it gathers blog posts that might contain IoCs and extracts the candidates, and later it discards the ones that were not relevant.

The first stage is a simple web spider. It first reads a list of base blog URLs from a list in a file. After that, it accesses each blog and uses web scraping techniques to follow every link, looking for posts and articles. Every time it finds an article or post, if passes it to the next step, which involves preprocessing and downloading it. The preprocessing is aimed at providing more context for further steps and also discarding the most obvious false positives (in this case, URLs that can be followed by clicking on them).

Once the article is downloaded, three processes are started: extraction of candidate IoCs, topic modelling and classification. The extraction of candidate IoCs is carried out by a regular expression based tool, which leverages the fact that IoCs have a specific format (e.g MD5 hashes are represented as hexadecimal strings of 32 characters). Every piece of text matched by the tool is passed onto the next step. Meanwhile, topic modelling and classification processes work together to determine whether the full article is about a cyber attack or addresses a different topic. To do that, a pre-trained topic modelling tool is used in conjunction with a classifier trained by us. If the topic is deemed irrelevant, or no candidate IoCs are found, the post is discarded, otherwise it is passed to the validation process.

The validation process receives the full text and the candidates to indicator of compromise. First, it accepts as valid without further processing any IoC that is written in a way that prevents the reader from accidentally accessing it. For example, a URL might be written changing the `https` scheme to `hxxps`, so that nobody

accidentally accesses a malicious site. Furthermore, if the URL domain is found in a list of the most popular domains on the internet, it will be rejected, assuming that it won't be malicious. Conversely, if the domain has been recently registered, it will be regarded as possibly malicious.

If none of the previous cases apply, the sentence in which the candidate was found is taken in account. The sentence is syntactically analyzed and its main verb is compared with a list that contains the verbs that are often used for describing the behaviour of an attack. If the candidate IoC refers to a verb in that list, it is considered valid and stored in a database.

The database stores information about the extracted IoCs, including its content and type, but also has some validation information, details of which will be given below. It also stores information about where the IoCs were found. It stores the url of the post it was found in, as well as a set of keywords extracted from that post. Additionally, a timestamp is added to store the moment in time this IoC was detected by the system.

We ran an experiment for testing the precision of the system. To do so, we gathered every extracted IP, URL and hash value and checked it using virustotal. If virustotal classified it as dangerous, we considered that IoC as a true positive. Otherwise, we checked with kaspersky's platform. That is because we observed virustotal has some outdated information.

But we extracted other types of IoC in addition to IP, URLs and hash values. Concretely, we extracted email accounts, windows registry entries, filenames and filepaths. We could not validate those automatically and there were too many for manually validating them all, thus we took a random sample in such a way to have a confidence of at least 95% and validated the items on that sample.

The resulting precision was as high as 0.951, but it was not uniformly distributed. Email accounts had a higher precision and IP addresses, URLs and hash values were above 0.9 precision. Conversely, filenames and paths were below 0.6.

To sum up, we proposed *IoCMonitor* a new software system for extracting cyber threat intelligence in the form of indicators of compromise, in a way that minimizes false positives with high precision and could therefore be used in a non-supervised way to provide updated information to a CTI platform.

Chapter 2

Introduction

In this chapter, we present the reasons for this work, the challenges that it aims to overcome and its contributions.

2.1 Context and motivation

The recent increase on successful cyber attacks [27] suggests the fact that usual cyber defense measures are not effective enough for current cybercriminal activities. For example, in May 2021, a ramsonware attack severely threatened critical infrastructure in the U.S. [19], to the point that it forced an important oil pipeline to shut down temporally.

There seems to be a market trend towards ransomware, which has increased the frequency of these types of attacks [10]. The potential impact of an inadequate protection against cyber attacks range from data leaks and unauthorized usage of computing power to critical infrastructure disruption. This could render useless key facilities such as dams, power lines or nuclear power plants.

In this context, the fact that cyber criminals are able to continuously bypass defense measures like firewalls and intrusion detection systems and spread malware suggests that such measures are ineffective [25] or at least could be significantly improved.

One possible enhancement is to maintain these tools updated, not only in terms of software but particularly in terms of information. This means it would be useful to have information about how current and recent attacks behave, how to detect them and how they can be stopped. This kind of information is often referred to as Cyber Threat Intelligence (CTI). Gathering and sharing cyber threat intelligence is a powerful tool, as it provides updated information about currently active cyber attacks, allowing traditional tools such as firewalls to effectively block them. For example, let's imagine a piece of malware that uses the SMB protocol to move laterally inside an organization. A researcher could notice that and create a piece of CTI that states that upon detection of this malware in an organization, the port corresponding to the SMB protocol has to be closed. However, the longer time passes between the finding of that feature of the malware and the publication of the piece of CTI, the higher risk the organization is subject to. For that reason, being able to share and receive updated information is critical for this strategy to work.

This information can come in the form of indicators of compromise (IoC). In the previous example, the hash value of the malware executable or the SMB port would be indicators of compromise. Ideally, IoCs would be written in a formal language, so that it is easy to program the parser and extract the information automatically. Instead, in reality, researchers who find out those IoCs most often publish them in blogs and social media. Such platforms are not meant to be parsed by a piece of software, but rather to be read by a human. For this reason, they are written in natural language such as English, which is far easier and more pleasant for people to read, but is not formal, which makes it hard for software to effectively extract information from it.

Given the importance of IoCs for the present and future of cyber security, it makes sense to put effort in extracting such information from updated sources. However, as stated previously, these sources are often written in natural language, which cannot be trivially parsed by software.

2.2 Problem definition

Cyber security researchers often discuss details about ongoing or recent attacks on the Internet. These details are extremely valuable for cyber security, but there are two main challenges for extracting threat intelligence from this kind of sources.

First of all, security blogs and social media sites are meant for other researchers or users to read, thus it is not structured, making it hard for a piece of software such as a threat intelligence platform to extract the valuable knowledge from it. For example, a post reporting a malicious email "account criminal@example.com" could have sentences like "the victim received an email from criminal@example.com containing the payload", "criminal@example.com sent a suspicious email to the victim" or "the malware spreaded from a malicious DOC file that was attached to an email from criminal@example.com". While all the sentences mean that the email account is malicious, it is not trivial to extract this information automatically without human intervention. Thus, more complex processing needs to be carried out so that information can be automatically extracted in a useful way. This problem is not new. Indeed, Natural Language Processing is a field of artificial intelligence that develops methods for allowing software to extract knowledge from texts written in natural language.

The second challenge is that there is an overwhelming amount of online resources that provide such information, and it is infeasible for a human to extract the useful parts from all of it in real time. For testing our work we selected nine blogs (see Evaluation in section 5), which in total published 25 blog posts in a week (2021 July 5-11). Even though the aim of the evaluation process was not to provide complete, up to date CTI, this gives an intuition of the volume of work that would be devoted solely to reading blog posts searching for valuable pieces of cyber threat intelligence. This is the main reason why a system for automatic extraction is needed: security experts' time is too valuable for asking them to spend most of their time to reading publications

Those challenges tend to balance each other. The obvious solution to the second one involves fast, automatic processing of new information from online resources, which would allow security experts to employ their time in other matters. However, the first challenge acts as a counterweigth: the less involved humans become, the more precise the results from software system need to be. However, there are computational methods that look promising for allowing humans to become less involved in cyber threat intelligence gathering and updating.

In addition to the challenges, current tools such as `iocextract` [12] rely on regular expressions, which cannot take the context into account, thus being prone to false positives. Let's consider, for example, a blog post about a new vulnerability discovered on Xiaomi smartphones, affecting particularly the version `11.0.6.1` of the MIUI firmware. Any tool based on regular expressions will match the version number as an IPv4 address, as it has exactly the same format. Such false positives should be minimized because IoC extraction tools are meant to be used in a non supervised fashion, meaning there should not be human involvement in the process.

From the software system perspective, there are at least two problems that need to be addressed in order to effectively extract correct CTI. The first one is being able to recognize if a given article discusses a valuable topic for cyber threat intelligence or not. This is because often even technical blogs can publish off-topic articles such as tutorials. The second one involves assessing the validity of the extracted CTI. In particular, indicators of compromise have a specific format, but this format is shared with other items that are not IoCs. Thus, found IoCs must be processed in order to find out whether they are correct or not.

2.3 Contribution

The aim of this work is to design and implement *IoCMonitor*, a method and software system for extracting indicators of compromise from public sources such as social media and blogs, in a way that false positives are minimized. Therefore, the final target of this work is to provide updated, timely information to cyber threat intelligence platforms in order to improve cyber security of organizations and individuals.

The first problem we want to solve is assessing the relevance of a document such as a blog post or article. Irrelevant posts or articles can appear in the used source blogs, and even more likely when using linkedin groups. To solve that, we employ a topic modelling tool that classifies the article in one topic, in conjunction with a one class SVM novelty detector. To the extent of our knowledge, this method has not been used before.

The second problem is assessing the validity of each possible indicator of compromise. Pieces of text can resemble indicators of compromise but have a different meaning. This is why further validation is needed. This problem is tackled by using syntactic dependence analysis: sentences containing the possible IoCs are analyzed

and the occurence of keyword verbs is checked to accept or reject the IoC. A similar method has been explored by Liao et al. [15] but they employed different, more complex graph mining techniques.

In conclusion, the problems stemming from automatic cybert threat intelligence gathering, such as the assessment of document relevance and of indicator of compromise validity, have been addressed previously. However we propose a new system that integrates the solution to those problems to provide updated information that can be used for improving the security of organizations and individuals. Thus, *IoCMonitor* could significantly lower the overall number of successful cyber attacks.

2.4 Document Organization

The rest of the paper is organized as follows.

Chapter 3 explains shortly the most important concepts needed to understand this work.

Chapter 4 describes how IoCMonitor works.

Chapter 5 describes the experimental setup for testing IoCMonitor and shows and discusses the results.

Chapter 6 Shows and explains previous work on the automatic extraction of categorization of cyber threat intelligence.

Chapter 7 concludes the work and suggests future developments.

Chapter 3

Background

This chapter explains the core concepts needed for understanding this work. It is not meant to explain each topic exhaustively but to provide some context to minimize the technical background needed for understanding it.

3.1 Cyber Threat Intelligence

Cyber Threat Intelligence (often written as CTI) is a broad concept that refers to every piece of knowledge referring to cyber attacks [1]. This includes knowledge about the attack itself (e.g its behavior) and how to detect it (e.g where it comes from). This kind of knowledge is critical for organizations, as it can be used as the first line to prevent cyber attacks.

3.2 Indicator of Compromise

Indicators of compromise (IoCs) are pieces of CTI that refer to the detection of the attack. In particular, they provide evidence that a given machine or system has been attacked[16]. They can be network locations (such as IP addresses or URLs), filenames, hashes of malicious executables, Windows register entries, etc.

Below, we provide definitions of some categories of IoCs.

3.2.1 IP address

The IP address is a value that identifies a network resource (e.g a server) in order to make it accessible from the network. There are two versions of the protocol and thus of the addresses. The first one, IP version 4 (IPv4), is a four-byte value, usually represented as four one-byte decimal numbers (i.e values from 0 to 255) separated by dots (e.g 192.168.1.1). The second version is version 6 (IPv6). The IPv6 address is sixteen bytes long and is usually represented as eight two-byte hexadecimal values (i.e values from 0 to FFFF), separated by colons (e.g 2001:db8:85a3:8d3:1319:8a2e:370:7348).

It is important as an IoC because it can describe the location from which an attack is launched, or where something is downloaded or uploaded.

3.2.2 URLs

URLs are textual representations of the location of a resource (usually a file) on the web and are defined on RFC 3986 [5]. They usually have the following format:

```
https://example.com/index.php?f=1
```

We can differentiate four parts: `https` as the application level protocol or "scheme", `example.com` as the host, `/index.php` as the path to the file and `f=1` as the query parameter.

Some possible attacks on URLs rely on errors made either by the network or the user. The host name is resolved to an IP address using the DNS protocol. This protocol uses an untrusted transport protocol (UDP), that could potentially lead to a bit changing in the query or the response. Thus, a query for `example.com`

could be changed to `excmple.com`. That is because the "a" character is encoded as bin 01100001 and "c" as 01100011, thus it suffices to flip the second least significant bit of the "a" character for it to become a "c". An attacker could register the domain `excmple.com` and make it look like the original one in order to e.g steal data from the victim.

A similar attack could be executed but instead of relying on DNS errors, they could exploit the possibility of an user mistyping the target domain. For example, someone could write `wxample.com`, as the "w" is near to the "e" in the QWERTY keyboard. The rest of the attack would work as mentioned above.

In addition to the aforementioned specific attacks, URLs are used in the HTTP and HTTPS protocol for accessing files to download. That is the reason because they can point to malicious files. Therefore, an updated list of malicious URLs to be blocked can be very useful for an organization.

3.2.3 Hash values

Hash functions are cryptographic functions that are meant to be one-way. This means that given an input x it is easy to calculate its output $H(x)$, but the output or hash value $H(x)$ it is almost impossible to guess the input x. The most interesting features of these functions are:

- Arbitrary size inputs (thus accepting full files).
- Fast execution time.
- Two very similar inputs will yield very different outputs.
- Collusion free: It's almost impossible to find two inputs that will yield the same output

Hash values are a few bytes long, which make them ideal to describe large files. That's why they are used as IoCs for describing malicious files or malware.

The most common hash functions for describing files are MD5 and SHA256. MD5 provides a 128 bit output. As hash values are usually represented as hexadecimal strings, this means an MD5 hash is a hexadecimal string of 32 characters (as each character encodes 4 bits). Conversely, SHA256 yields a 256 bit output, which is represented as a 64-character hexadecimal string. More versions of SHA exist, including one that yields outputs of 512 bits length. Likewise, more cryptographic hash algorithms exist.

3.2.4 Email address

Email addresses are user identifiers used by email protocols SMTP, POP3 and IMAP for exchanging messages. They are constructed using the same format as the following example:

`name@example.com`

Here, `name` is the account name, while `example.com` is the host where that account belongs in. From the cyber threat intellingence perspective, they can be the source of attacks such as phishing campaigns. Phishing campaigns are attacks where someone, the attacker, sends an email to a lot of possible victims, in a way that it seems to be a legitimate email from somewhere else, such as a social network or a bank. This email usually has a link to a website that looks very much like the one of the mentioned social network or bank, but is controlled by the attacker. When the victim inputs their credentials to it, the attacker receives them and can take control of the account of the victim. Another, more sophisticated and risky attack is spear phishing. This kind of attack is directed towards a specific person or organization, and usually involves gathering data from them in order to make the phishing email more credible.

3.2.5 Windows Registry Entry

Windows Registry is a database where configuration settings for the Windows operating system and its programs are stored. It is hierarchical, meaning that its entries are accessed in a very similar way to a filesystem, for example: "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows". Since it contains configuration settings, a malware could modify it for hiding or being persistent after a reboot. Therefore, modified registry entries can be valuable pieces of cyber threat intelligence.

3.2.6 YARA rule

YARA is a popular tool for malware research that allows to write rules in an standadized way for identifying and classifying malware samples. Security researchers might publish them for others to identify a piece of malware they have discovered.

Their format is as shown in this example from the documentation[1]

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        threat_level = 3
        in_the_wild = true
    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"
    condition:
        $a or $b or $c
}
```

In this case, files containing any of the given strings will be reported as`silent_banker`. Rules can be more complex and include tools such as regular expressions.

3.3 Natural Language Processing

Natural language processing (NLP) tries to automatically extract information from a natural (i.e human) language. This is not trivial because human communication is ambiguous and very context-dependant. In this work, we use tools for topic modelling and syntactic dependence extraction. Topic modelling aims to find out the different topics a set of texts has, generally by finding its keywords and their co-ocurrence. For example, a text related to cyber security might have the words “malware”, “vulnerability” or “exploit” ocurring much more frequently than a text about a different topic.

3.3.1 Dependency extraction

The other piece of natural language processing we are using here, syntactic dependence extraction, focuses on finding the gramatical relationship between the words in a sentence. As an example, take the sentence “the quick brown fox jumps over the lazy dog”, “quick” and “brown” refer to “fox”, which is the subject to “jumps”, and so on.

The dependency extraction outputs a directed graph whose vertices are the words in the sentence. The edges connecting pairs of words are labeled, describing the nature of the relationship. For example, there would be an edge between “fox” and “jumps” with a label meaning “subject”.

3.4 Machine Learning

Machine learning (ML) is a field of computer science that focuses on creating algorithms that allow to automatically create models for solving problems such as classification. The particularity of these models is that they are created only by example, i.e by processing data in a phase called “training”. During training, the classification algorithm processes a dataset and tunes its internal parameters so that when it processes a new datapoint it is able to predict its category. The training dataset can be labeled, which means that every datapoint has a category attached. Examples of such classification algorithms are support vector machines or neural networks. Classification usually involves representing the data as points in the space, and finding a function such that given a datapoint as an input, it outputs its category. To do so, a possible approach is to define a border that separates one category from another, and check in which side the new datapoint is.

[1] `https://yara.readthedocs.io/`

3.4.1 Novelty detection

In the field of machine learning, novelty detection is one of the possible problems to solve. It has very much the same characteristics of the classification problem, but with one difference. While classifiers are trained with a dataset containing datapoints from every category, novelty detectors are only trained with one category. The target is to classify every new datapoint as "known" or "novel". In this case, known datapoints would belong to the same category as the ones used for training, while novel datapoints are of a different category.

3.4.2 Support Vector Machine

A support vector machine (SVM) is a machine learning algorithm that represents the data as points in an euclidian space. For training, it needs a labeled dataset of examples. Here, "labeled" means that during the training phase, the algorithm has a way to know which category a particular datapoint belongs to. Using those points, the algorithm calculates a hyperplane that separates points of different categories from each other, in a way that there is maximum distance between the hyperplane and the closest points to it [7]. Figure 3.1 gives an intuition of how it works.

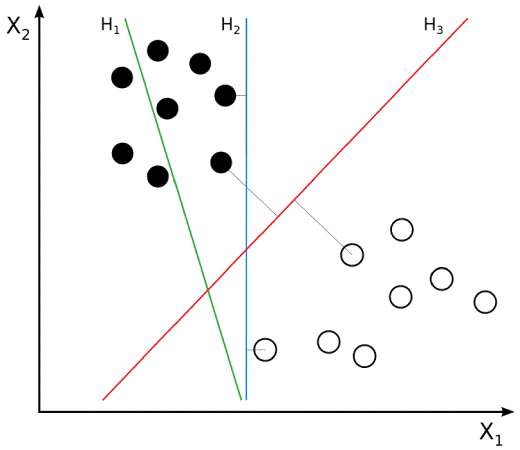

Figure 3.1: H_1 does not separate the classes. H_2 does, but only with a small margin. H_3 separates them with the maximal margin.
Image and caption by User:ZackWeinberg, based on PNG version by User:Cyc - This file was derived from: Svm separating hyperplanes.png, CC BY-SA 3.0, `https://commons.wikimedia.org/w/index.php?curid=22877598`

This in principle would only work for linearly separable items, but it is common to use kernel functions: auxiliary functions that help to re-locate datapoints in a way that, after applying the function, datapoints become linearly separable.

Chapter 4

Methodology

This chapter describes how IoCMonitor works. The first section provides a general, simplified idea, while the ones following it are dedicated to each of the major steps taken to solve the problem.

4.1 Overview

The purpose of this section is to provide a simplified, high-level idea of how *IoCMonitor* works. In its pipeline, first of all articles and posts are downloaded and preprocessed in order to provide context and filter out some possible false positives. After that, candidate IoCs are extracted and the whole text is validated as relevant for cyber security. Finally, the context in which each candidate appears is validated and, in case they are not discarded, they are stored in a database.

IoCMonitor architecture is shown in figure 4.1.

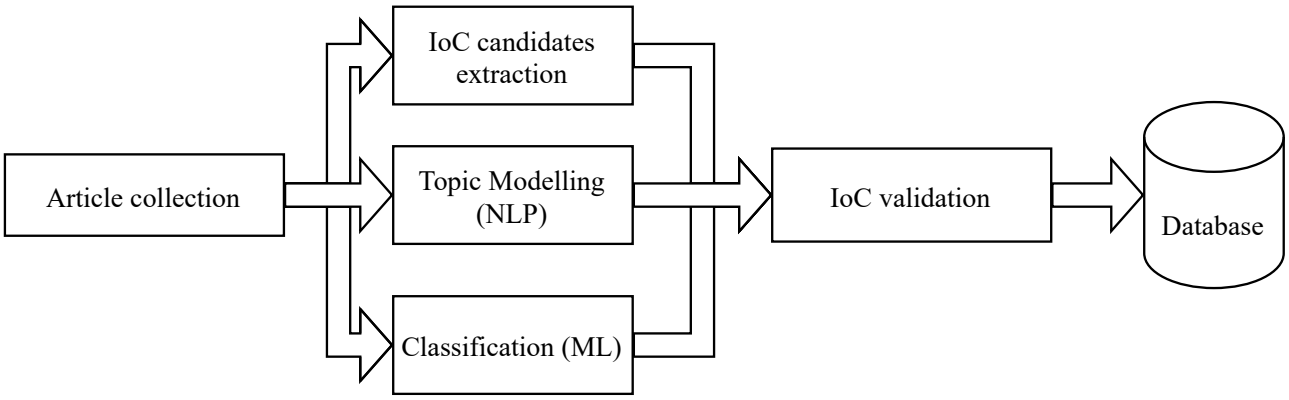

Figure 4.1: High level representation of the system

First of all, the articles are collected from a set of predefined sources in a normalized way, using a web crawler and web scraping techniques . In this stage, lists are processed so that later steps can take in account the context these list appeared, rather than only the contents of the list item. Additionally, obvious false positives such as clickable urls are removed.

After being downloaded, articles are filtered taking into account both their length and whether they have possible indicators of compromise (IoC) using regular expressions. During this step, we also extract the candidate IoCs that will be validated later. The reason we can use regular expressions for this matter is that for the IoC categories we are looking for, there is a well defined format that they will match. However, not everything that matches the regular expression will be an indicator of compromise. For example, URLs are very common to be found in websites, and finding them does not mean that they are necessarily malicious. That is why we say the pieces of text matched by the regular expressions are "candidate IoCs" and we need to further validate them to assess whether they are actually IoCs.

In order to validate the candidate IoCs, three processes are followed. Two of them take in account the whole article. They are aimed at assessing if an article is relevant for cyber security matters or not. The first process involves using a natural language processing tool to perform topic modelling on the artcile, while the second one

uses a machine learning approach. We mix the output of both processes together using a heuristic, and reject the article if it is not related to cyber-security.

If the article is not rejected, it is passed to the last validation phase. Here, each candidate is taken in account along with the sentence it is found in. The sentence is syntactically analyzed and depending on which verbs the candidate refers to, the candidate is accepted or rejected. Moreover, if it is an URL, additional checks are performed. For instance, if the url's host is among the most popular domains, it is rejected from being a valid IoC. Conversely, if it was registered recently, it is accepted regardless on the syntactic analysis.

If the candidate is accepted, it is stored in a database. The database additionally stores information on the post the IoC was found on, as well as the moment in time.

To sum up, there are six different processes: article collection, candidates extraction, article topic modelling, article classification, validation of the extracted candidates and storage. The article collection stage downloads and preprocesses the articles and passes their text to the next phases: candidates extraction, topic modelling and article classification. Candidate IoCs are extracted using regular expressions, leveraging the specific format of IoCs. The topic modelling and classification of the article are performed almost together and allow us to discard the whole article if it is about a different topic. If their article is not discarded, candidates are validated taking in account their context and later stored in a database.

4.2 Article collection

This stage is meant for accessing articles or blog posts on the web and convert it into text. This is needed so that later steps do not need to deal with the mix between HTML tags and natural language. In addition, here is the only point where the post can be pre-processed taking in account all the features it could have, such as links and lists. This is the reason why lists are marked so that later steps can take in account the previous sentences in order not to lose context. Moreover, this is the only moment where there is access to the link tags (`<a>`), and thus we can filter clickable urls away.

The articles are collected from different sources. We have worked with the "Advanced Persistent Threats (APT) & Cyber Security" group on Linkedin and with a set of security blogs.Figure 4.2 shows the process.

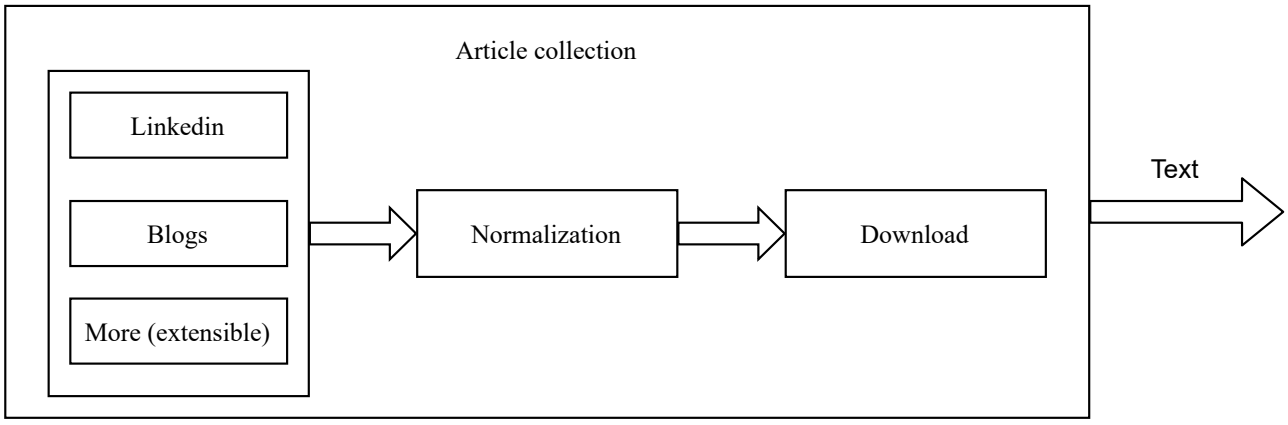

Figure 4.2: Detail of the article collection stage

We use web scraping techniques for collecting the URLs, for which we developed two different tools. The first one monitors the view of the Linkedin group feed and extracts the URLs that point outside Linkedin, discarding social media sites such as Twitter and YouTube.

The second tool is a custom web crawler. It follows all the URLs under a given path in a breadth-first fashion. Let us take, for example, Webroot's blog front page[1], and one of its articles[2]. The article URL starts with the same path as the front page. This fact allows the web crawler to discard other sections of the website that might be linked from the blog, as well as links to completely different websites. The crawler reads a list of blog base URLs from a file, which makes it easy to extend with more source blogs. For each one of the blogs. It accesses their front page and stores in a queue every URL in it that starts with the same path as the blog front page. This

[1] `https://www.webroot.com/blog/`

[2] `https://www.webroot.com/blog/2021/06/01/oh-no-a-client-failed-a-pen-test-now-what/`

is done iteratively with every URL in the queue until it's empty. Additionally, every URL that is stored into the queue is collected as an article for further processing.

Once the URLs of the articles are collected, every article is accessed and normalized. To do so, Firefox browser "Reader mode" is leveraged. This mode is meant to keep only the title and contents of an article, and show it with a uniform style. An example can be found on figure 4.3. Its left side is the original page from a Webroot's article. Besides the article itself, it has navigation sections (on the top and left parts), an advertisement, a clickable tag cloud and the Twitter feed of the company. On the right side of the figure, conversely, we can only see the name of the website, the title of the article, its author and its contents. This mode allows us to discard webpages that are not articles, and eases the web scraping process. This is because we are using Selenium for web scraping. Selenium is a tool for automating a web browser via programming. That allows us to leverage the built-in facilities of browsers such as javascript execution, for activities like scraping a dinamic website. Reader mode makes scraping easier because in addition to presenting a uniform look, the HTML code is also uniform, which allows us to extract the contents of any article without knowing its HTML/CSS structure beforehand.

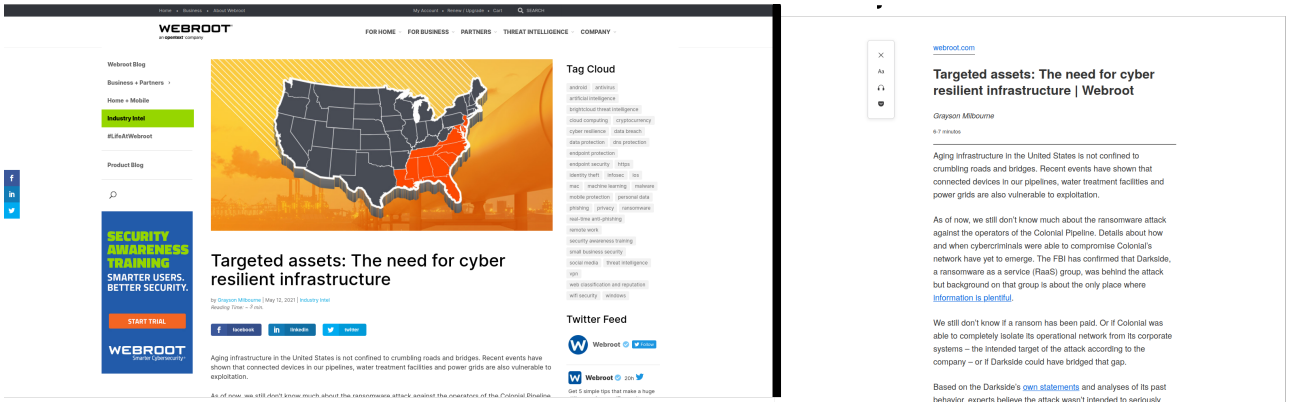

Figure 4.3: Example of browser's reader mode. The original page (left) is stripped of additional features in order to provide only the contents of the article in a clean way (right)

In addition, two further steps are taken in order to minimize the number of false positives:

- Every clickable link whose text is the same as the URL it points to (i.e every `<a>` tag whose `href` is the same as its text) is deleted. The underlying assumption is that writers won't make it easy to click on links to malicious URLs.

- List items are marked so that, if they are to be processed as a sentence (see 5.1.1), the sentece immediately before the list is processed too, in order to provide some context. For example, let us consider the following fragment:

  The hashes related to this piece of malware are:

  - 48307C22A930A2215F7601C78240A5EE
  - 5F0F1B0A033587DBCD955EDB1CDC24A4
  - C1159FE3193E8B5206006B4C9AFBFE62

  When the first item in the list is processed to assess whether it is a true positive, the sentence "The hashes related to this piece of malware are: 48307C22A930A2215F7601C78240A5EE" is evaluated instead of only the list item "48307C22A930A2215F7601C78240A5EE"

After the normalization takes place, the article content is extracted as plain text for further processing. It is then checked against a set of regular expressions provided by python module `iocextract` [12] in order to find IoC candidates. The regular expressions work in this case because the IoC categories we are looking for have specific formats. As an example, IPv4 addresses are represented as four numbers separated by points, URLs always have at least `<scheme>://<host>`, usually followed by a path, and hash values are represented as hexadecimal strings.

If no piece of text is matched in the article, or if it is shorter than a hundred words, the article is rejected. The length check prevents the system to process pieces of texts that are not really articles, as we have observed the average word count of technical articles and posts is higher than one hundred.

In summary, articles are first accessed on the internet and normalized using the browser in an automatized way. After that, they are processed to add context to list items and to remove pieces of text that would be matched as IoCs but aren't one, such as clickable links. Finally, its length is checked and a regular expression based tool is used in order to look for candidate IoCs. If the article or post is sufficiently long and has candidates, it is passed to the next step.

4.3 Topic Modelling

The goal of this step is to determine whether the article is related to cyber security. This is used to discard false positives that might appear by accident in articles that are not security reports. For example, URLs are by nature abundant on the internet, thus posing a risk to process an article that contains candidate IoCs as URLs but does not contain any security related information. In order to classify the article as a whole, we employ two methods. The first one involves using an already existing natural language processing tool called *Mallet*. After training this tool, it outputs the likelihood for a new text to belong to one of the topics it was trained with. The sencond method we use to classify the article is a SVM novelty detector. To use it, after training with relevant examples, texts are vectorized and classified as novel (irrelevant) or not (relevant). Finally, the results from both methods are mixed using a heuristic.

The process is shown in figure 4.4. To achieve a better result at topic modelling, the text is firstly processed in order to get a more generic form. Using Python's Natural Language Processing Toolkit (`nltk`) [6], every word is converted to its root and base form (e.g. "was" $\rightarrow$ "be", "criminals" $\rightarrow$ "crim") in order to improve the topic modelling tools output. This process is called stemming and lemmatization, respectively.

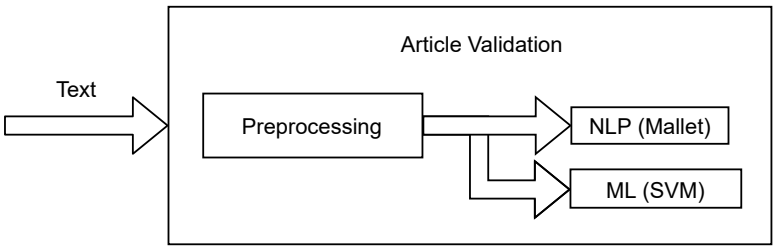

Figure 4.4: Detail of the topic modelling stage

After that, the text is fed to two different tools. The first one is a one-class SVM that performs novelty detection [23], while the other is a natural language processing tool that provides topic prediction. It is called *Mallet* [17].

The SVM performs novelty detection, predicting whether or not the given text belongs to the same category as the ones it was trained with. However, as explained in the Background section (see 3.4.2) it does not work with words, but rather with vectors only. This is why words need to be vectorized, or transformed into vectors. The algorithm used for vectorizing the documents is TF-IDF based. TF-IDF (Term Frequency - Inverse Document Frequency) is a way of scoring terms in a set of documents.It is calculated as the ratio between the number of occurrences of the term among all documents (Term Frequency) and the number of documents it appears (Inverse Document Frequency). The reason behind that is that very common words usually provide very little new information, but those that appear scarcely but in the right places carry much more information.

After the vectorization, the SVM can be used. For training, we used a set of relevant cyber security articles, thus making it the "known" category. Later, when a new article needs to be classified, it is vectorized and the SVM predicts the category of the vector. If it is classified as novel, it means it is not relevant. Conversely, if the SVM classifies it as known, the meaning is that it is probably relevant.

The natural language processing tool, *Mallet* yields the probability of a given text to belong to one of the topics it was trained to classify, as exemplified by table 4.1. The topics are defined by a set of keywords. In order to determine which specific settings to use, we gathered a test dataset, apart from the training one. The test dataset was composed by articles from two different sources. The first source was Hackernews' specialized

Article	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7
relevant_1	0.3%	44%	0.4%	8%	7%	7%	31%
irrelevant_8	5%	14%	31%	2%	26%	5%	12%
relevant_13	0.3%	57%	11%	11%	1%	8%	27%

Table 4.1: Example output from `Mallet`. The "Relevant" articles were taken from hackernews' blog website, and the "Irrelevant" one came from BBC's news website.

security blog[3], while the second source was BBC's generic news[4] feed. Articles from the first source were labeled as relevant or positive and those that came from the second one were labeled as irrelevant or negative.

A classification setup is the configuration regarding the number of topics and keywords used by *Mallet* and the heuristics used for classification. We considered a specific classification setup "A" better than "B" if "A" predicted correctly more relevant articles than "B" (i.e had less false negatives or higher recall). If they had the same precision, we considered "A" better than "B" if "A" predicted correctly more irrelevant articles than "B" (i.e had less false positives or higher precision). We are aware that a setup that classifies every article as "relevant" would be better than many others using this method, but we don't think this greatly affects the overall performance because the remaining steps don't take a lot of time and false positives will be discarded later.

We experimented with different numbers of topics and keywords per topic. For each of the possibilities, we tested different heuristics in order to select the best settings. Those heuristics took into account the two most frequent topics to be either at the top two or the bottom two for each category on the test dataset. For example, for the data shown in table 4.1, the top two topics for the relevant articles are topics 2 and 7, while the bottom two are topics 1 and 5. We experimented with three heuristics: "top", "bottom" and "both". In the example, the "top" heuristic would classify as "Relevant" any article whose most likely topics are 2 and 7 (in any order), the "bottom" heuristic would classify it as "Relevant" if its least likely topics are 1 and 5, and the "both" would combine them and classify as "Relevant" the articles whose most likely topics are 2 and 7 and its least likely ones are 1 and 5.

We also calculated the ratio between common and technical words. This was done by counting the number of errors reported by `enchant` [13] spellchecking library and dividing it by the total number of words, under the assumption that there would not be many typos in professional blogs or news sources, thus the spelling errors should be words that are too specific to the domain that they are not included in the library's general english dictionary. We calculated this dictionary word density for our test dataset and fitted the data from each category to normal distributions for finding at which ratio value they crossed in order to set a threshold. This can be seen in figure 4.5.

The threshold found with this method was at 0.0757. However, we finally discarded this method because it was outperformed by the final setup of *Mallet* and the SVM, and could not contribute more precision or recall to it.

The best setup that we found was seven topics with eleven keywords with the "top" heuristic. Additionally, if an article is classified as "Irrelevant", it is passed to the SVM in order to confirm it. If it is labeled as "Irrelevant" by the SVM, the article is discarded, otherwise it is classified as "Relevant" and passed onto the next step of the pipeline. This method had a recall of 0.97 and a precision of 0.87.

In conclusion, the topic modelling tries to avoid processing articles that have no relation with cyber security. To do so, two methods are used: a one-class SVM and a topic modelling tool. The SVM first vectorizes the text and tries to tell whether it is in the same category as the cyber security related texts it was trained with. Meanwhile, the topic modelling tool, *Mallet*, outputs the probability of the text to belong to the different topics it was trained with. Having both results (the category according to the SVM and the topic prediction by *Mallet*, they are mixed together using the following heuristic: an article is classified as relevant if it is classified as relevant by the SVM and *Mallet* outputs as most likely the same two topics as it does with the relevant articles in the test set.

[3] `https://thehackernews.com`
[4] `https://bbc.com`

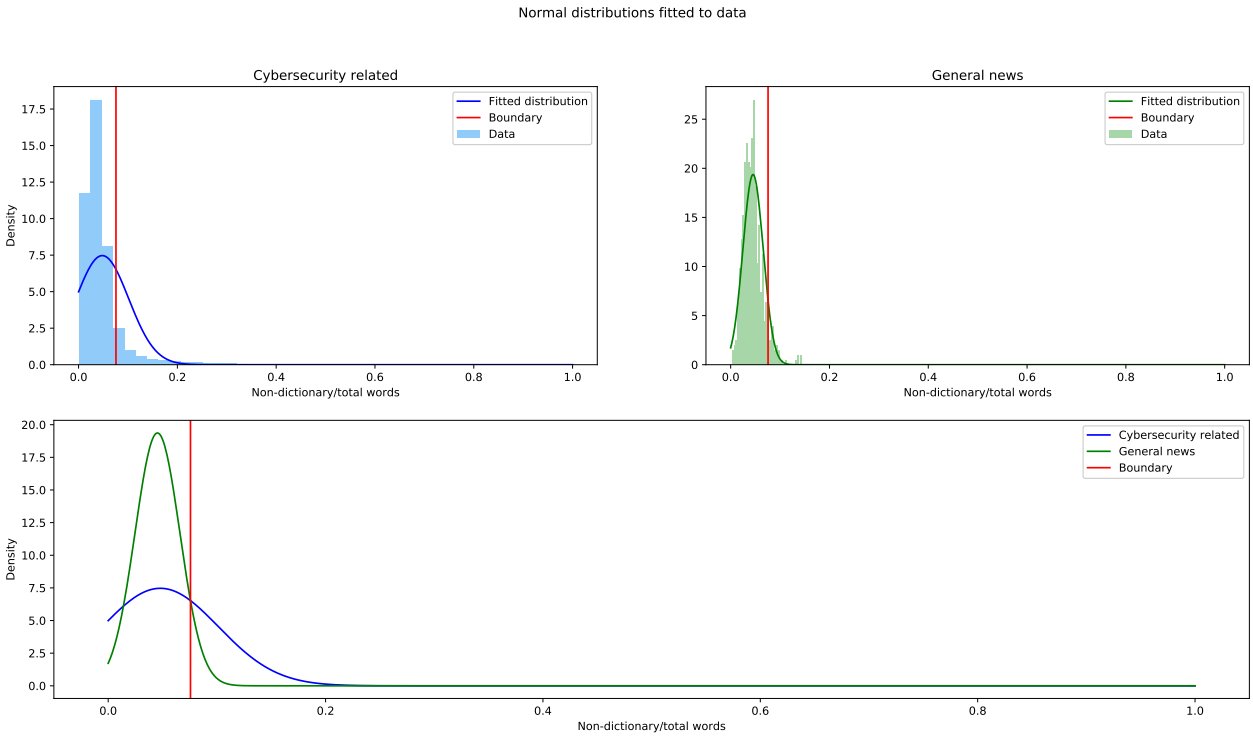

Figure 4.5: Specific to common words ratios calculated from the test dataset and threshold set

4.4 Indicator of Compromise validation

The Indicator of Compromise validation is the last validation stage of *IoCMonitor*. Candidate IoCs reaching this point are supposed to belong to a post or article related to cyber security, but that does not mean they are true indicators of compromise. In order to check them, we need to validate them individually because, as the topic is already related to cyber security, some of them will be valid but others could be invalid. Some IoCs like IP addresses, email accounts and URLs might be written in a way that puts special attention in making them understandable but hard to accidentally access them. If that is the case, they are directly accepted. In addition to that, URLs are treated in a special way. First, each URL host is looked up into two lists: the most popular domains and the recently registered ones. If the host belongs to the first one, it is classified as a false IoC and rejected. Conversely, if it is a recently registered domain, it is classified as a true IoC and stored. If the host is not in any list or the candidate IoC is not an URL, the full sentence containing it is taken in account. The sentence is syntactically analyzed, yielding a directed graph that is followed from the candidate to the main verb. If any of the encountered verbs is found in a custom list of relevant verbs, the IoC is accepted, otherwise it is rejected.

As outlined in figure 4.6, the IoC validation stage has two inputs. The first one comes from the topic mod-

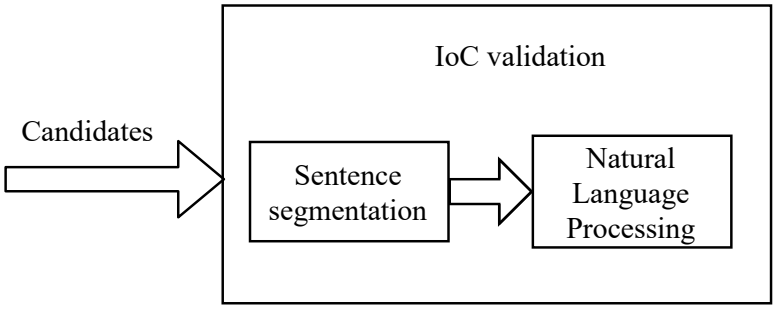

Figure 4.6: Detail of the IoC validation stage

elling step. If the topic of an article is understood as irrelevant, every candidate to IoC from it is rejected.

The second input is the list of candidate IoCs. Those candidates are extracted using a set of regular expressions, provided by the `iocextract` tool [12] for most IoC types, along with custom regular expressions for filenames, paths and Windows registry entries.

In total, we are able to match:

- IP addresses
- URLs
- Emails
- hashes
- YARA rules
- Windows registry entries
- Filenames
- File paths

After they are extracted, the article they come from is split into sentences using `spacy` [11], each sentence containing a candidate is then validated.

A set of methods are used to determine whether or not an IP address, an email account or URL is a valid IoC. First, we consider only the text referring to the candidate. Sometimes, URLs, emails and IP addresses are written in a way that they are not accessible by accident (e.g `1.1.1.1` is written as `1[.]1[.]1[.]1` or `https://google.com` as `hxxp://google[.]com`). Article writers do that to prevent their readers from accessing potentially harmful internet location, thus, we consider this as an indicator that the given candidate is a valid IoC. Therefore, we directly store it. Alternatively, if the URL refers to a domain found in Alexa top one million [30], a list of the most popular domains, or the IP is among the IANA reserved IP addresses, it will be rejected. Additionally, we consider URLs more suspicious, and therefore more likely to be valid IoCs, if the domains they point to were registered recently. This is because we assume a common practice by cyber criminals is registering a new domain for launching attacks such as phishing. In order to keep track of new domains we use `Hunting-Newly-Registered-Domains` [29]. We run this tool every day for downloading the new domains list provided by Whoisds[5] website. Thus, we have a list of domains and when they were registered. Moreover, we use the same tool to detect new domains that are similar to popular ones. This is because there is a possible attack where cybercriminals rely on possible typos made by users, mimicking the original website in order to steal data. For example, someone could be trying to access `facebook.com` but accidentally type `fscebook.com` in the address bar of the browser. A cybercriminal could have registered `fscbook.com` and created a login page that is indistinguishable from the legitimate one, and the victim would probably try to log in, giving their credentials to the criminal.

However, sometimes, candidate IoCs are not written in a special way, and there are other pieces of text that could be IoCs of a different type, such as hash values or email accounts. When we cannot directly accept or reject a candidate, we take in account the full sentence. To do that, we use `spacy` [11] to split the text into its sentences and to further process the relevant ones. Using syntactic analysis, we determine which verb is the candidate complement to, and compare it with a list of verbs that could describe an attack. We made the list by running the tool without this filter and manually writing down the verbs in true positive sentences, such as "download", "connect" or "send" . If the main verb is not in the list, the candidate is rejected. Figure 4.7 shows an example of the output dependency graph. The sentence is "HelperDll.dll downloads an encrypted payload from moviehunters[.]site". In this case, the candidate IoC is "moviehunters[.]site". As we want to find which verb it is complement to, we follow the graph arrows in reverse (i.e from head to tail) until we find a verb. By doing so, we get the sequence "moviehunters[.]site","from","payload","downloads". "Download" is included in the list of verbs that could describe an attack, therefore, the IoC is considered valid.

Summarizing, we consider two subsets of IoC categories: emails, IPs and URLs on the one hand and the rest on the other. Email accounts, IP addresses and URLs are treated in a special way because they might

[5] `https://whoisds.com/`

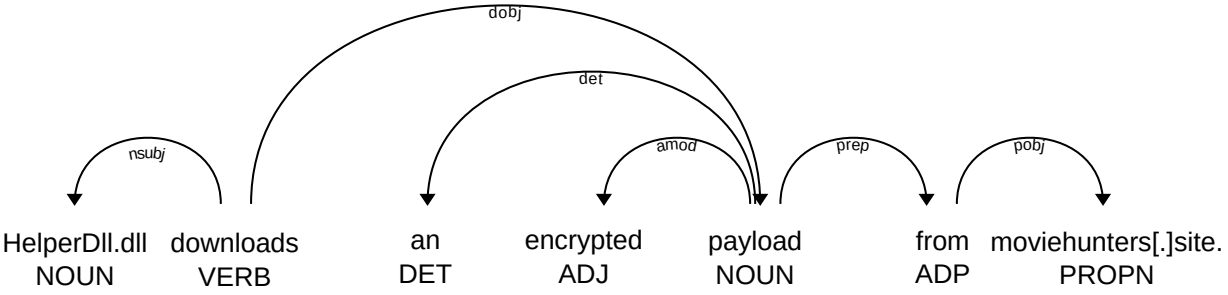

Figure 4.7: Example of syntactic analysis, as outputted by `spacy`

be written in a way that prevents a reader from accidentally accessing them, which confirms they are true IoCs. Furthermore, a URL pointing to a popular domain is probably not malicious and one pointing to a recent domain is more probably dangerous. Finally, all IoC categories follow a final check, where the sentence they belong to is analyzed and the path between the candidate IoC and the main verb is followed. If a verb belonging in a list of relevant verbs is found on that path, the candidate is classified as an actual IoC.

4.5 Data storage

After extracting all the indicators of compromise, we need to store them in a database. The reason for that is we want to be able to access them in order to create reports or export the data to a CTI platform. Furthermore, we store data for validation purposes.

The ER diagram of the database is shown in figure 4.8. It stores the indicator and its type, along with the

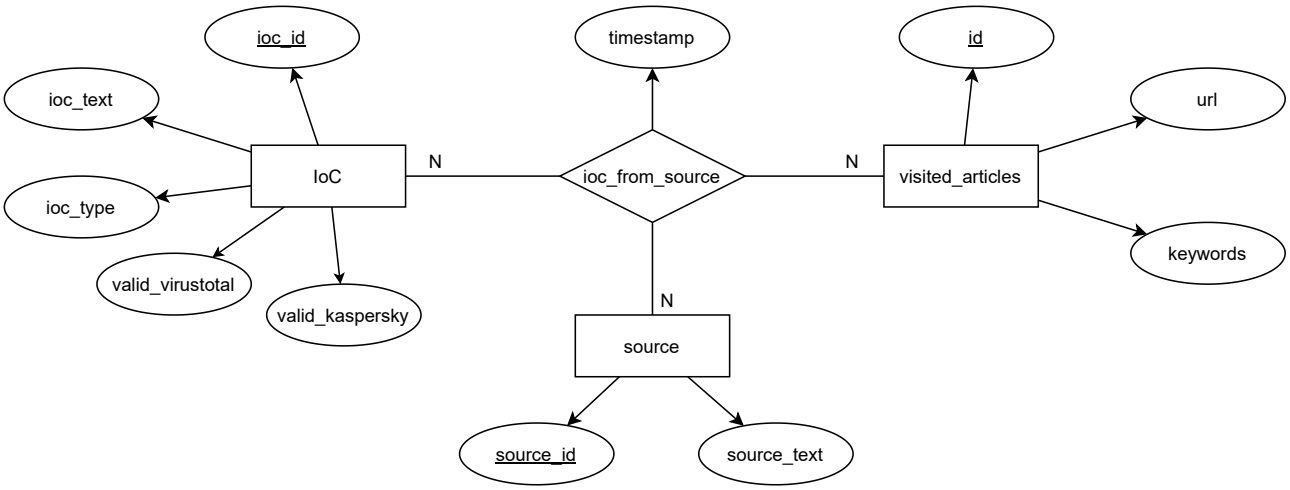

Figure 4.8: Entity-Relationship diagram of the database

validation data gathered from virustotal[6] and kaspersky[7]. The validation process consists on looking up every piece of IoC inserted in the database, i.e considered valid, on virustotal search engine. If it is found and labeled as risky by the engine, we consider it a true positive and store a "true" value in the corresponding field of the database. Otherwise, the same process is repeated with kaspersky's search engine. We use both engines because on the one hand, virustotal can provide potentially more accurate results, as it uses several search engines. On the other hand, it caches the results, storing some outdated data. That's why we double check with kaspersky.

Additionally, the database keeps track of every visited article, its url and keywords summarizing it(which are extracted using `textacy` [8]). Storing the URL of visited articles allows us to visit all articles only once,

[6] `https://www.virustotal.com/gui/home/search`
[7] `https://opentip.kaspersky.com/`

preventing us from processing the same article several times if it was linked from more than one page. Finally, it stores the source that URL came from. This was used mostly to differentiate which IoCs were extracted from Linkedin and which ones from each blog. We are also able to export the gathered data as json, in a way that can be easily processed or converted into other formats understandable by CTI platforms such as STIX [4]. An example can be found below:

```
[
{
 "type": "IP",
 "content": "135.181.170.169",
 "time-seen": "2021-05-21 17:13:38",
 "url": "https://www.anomali.com/blog/
 ↪ threat-actors-use-msbuild-to-deliver-rats-filelessly"
 },
{
  "type": "hash",
  "content": "87bfbbc345b4f3a59cf90f46b47fc063adcd415
  ↪ 614afe4af7afc950a0dfcacc2",
  "time-seen": "2021-05-25 19:11:58",
  "url": "https://blog.malwarebytes.com/
  ↪ threat-analysis/2020/11/malsmoke-operators-abandon-
  ↪ exploit-kits-in-favor-of-social-engineering-scheme/"
},
{
  "type": "URL",
  "content": "http://help365.us",
  "time-seen": "2021-05-25 16:44:30",
  "url": "https://blog.malwarebytes.com/cybercrime/2021/03/
  ↪ software-renewal-scammers-unmasked/"
}
]
```

Chapter 5

Results

In this chapter we show and discuss the experimental results we got from the evaluation of our system. First, we describe the settings used for running the experiment. After that, we explain how we validated the gathered data. Finally, we show the results and discuss them.

5.1 Experimental setup

We conducted an experiment on Ubuntu 18.04 with an Intel Core i58400 CPU and 16 GB of RAM. We used python interpreter 3.8.10 for executing the program and MySQL server 5.7.34 for the database. For the version listing of every library we used, please refer to Appendix B. The sources were LinkedIn group "Advanced Persistent Threats (APT) & Cyber Security"[1] and a set of blogs including Webroot[2], AT&T Cybersecurity blog[3], Dancho Danchev's blog[4], Hexacorn[5], Kaspersky's Securelist[6], Malwarebytes[7], Nakedsecurity[8], Taosecurity[9] and Trendmicro[10]. The system was left running for five days.

5.1.1 Data validation process

As mentioned in Section 4.5 For validating IP addresses, hash values and URLs, we used virustotal[11] and kaspersky[12] search engines. The process was as follows:

1. Look up the IoC in virustotal search engine.
   (a) If it was classified as dangerous, mark it as true positive and finish.
   (b) Otherwise, continue.
2. Look up the IoC in Kaspersky's search engine.
   (a) If it was classified as dangerous, mark it as true positive and finish.
   (b) Otherwise, mark it as a false positive and finish.

We used virustotal because it is supposed to check several different sources, including kaspersky. This in principle would suffice, but we observed that there were some true positives that were not detected by virustotal. This was because it did not check its sources every time, but rather keep a cache. The IoC we checked was marked

[1] https://www.linkedin.com/groups/3054767/
[2] https://www.webroot.com/blog
[3] https://cybersecurity.att.com/blogs/
[4] https://ddanchev.blogspot.com/
[5] https://www.hexacorn.com/blog/
[6] https://securelist.com
[7] https://blog.malwarebytes.com/
[8] https://nakedsecurity.sophos.com/
[9] https://taosecurity.blogspot.com/
[10] https://blog.trendmicro.com/
[11] https://www.virustotal.com/gui/home/search
[12] https://opentip.kaspersky.com/

as not dangerous years ago, and was never updated in the cache when other sources, such as kaspersky, started considering it dangerous. For this reason, we decided to double check every false positive reported by virustotal. We used kaspersky's search engine to do so, as we could access their API free of charge.

Emails, registry entries, filenames and filepaths were validated manually. However, there were too many detections, especially in the case of email accounts. Thus, we decided to take a random sample such that the confidence interval was at least 95% with a 5% error margin. The sample of size N was taken by performing a query to the database with a random ordering and using only the first N results.

5.2 Limitations

The nature of the evaluation process implies that we do not have any knowledge about how many IoCs actually where in the corpus of documents we analyzed. Furthermore, even though we extracted a fairly high number of individual IoCs, we used very few blog sources, which means we might have overfitted or underfitted those blogs and upon testing the *IoCMonitor* system with exactly the same settings but with a different list of sources, a different precision score could have been achieved. On the other hand, it is possible that our ground truth, virustotal and kaspersky database, is to some extent wrong. Some IoCs were found in articles published merely days before the validation process. Therefore, the database could be slightly outdated. After all, the ultimate goal of this work is to gather such indicators of compromise faster than the current process carried out by antivirus or other security platform vendors. All in all, we recommend caution when using *IoCMonitor*.

To sum up, the following limitations are found:

- We don't have knowledge of which actual IoCs were present in total in the posts we analyzed. Thus, we cannot calculate the recall of our method.
- The number of used sources is small, which means some artifacts could have appeared in the evaluation. For example, the used methods could be overfitting or underfitting those sources.
- The ground truth used in evaluation might be slightly wrong, biasing the results towards a lower precision.

5.3 Evaluation

IoCMonitor collected 39853 individual IoCs. The results and sample sizes are shown in table 5.1.

IoC type	Number of detections	Sample (confidence)	True positives (precision)	False positives
Email	22807	378 (95%)	370 (97.9%)	8
URL	4837	4837 (100%)	3648 (90.3%)	391
Hash value	5719	5719 (100%)	4759 (91.7%)	432
Filename	286	165 (95%)	93 (56.7%)	71
Filepath	312	231 (99%)	278 (54.8%)	229
IP address	6165	6165 (100%)	1758 (95.2%)	88
Registry entry	13	13(100%)	13 (100.0%)	0
Total	39853		10919 (95.1%)	1219

Table 5.1: Summary of the results

Figure 5.1 shows the distribution of the extracted IoC. Email accounts are the most common type of IoC found, followed by hash values and URLs. IP addresses represent also a significant share, but filepaths, filenames and registry entries are very uncommon.

The large amount of email accounts found might be because of the long, exhaustive lists of email-like addresses related to attacks found on Dancho Danchev's blog [13]. However, it might not be some sort of sample bias. Email accounts are very cheap and easy for cybercriminals (or anyone) to create. Moreover, phishing and other email-related attacks seem the easiest to spot and especially to track down to the email account. That

[13]For example:`https://ddanchev.blogspot.com/2021/05/profiling-currently-active-portfolio-of_31.html`

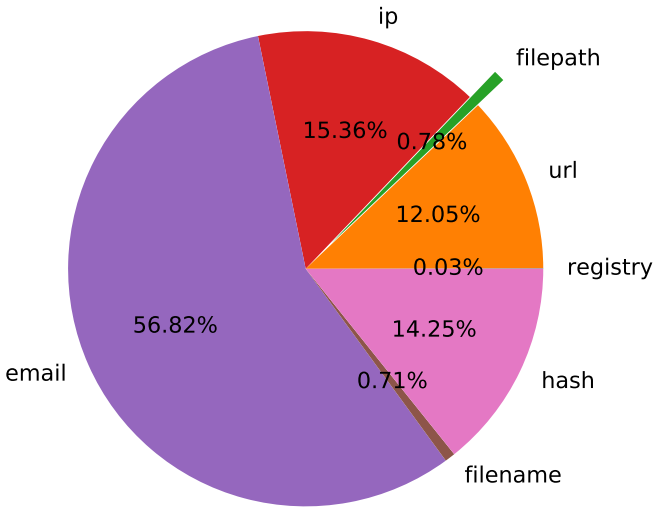

Figure 5.1: Distribution of extracted iocs by type

fact, on the one hand, would make email accounts less reusable by cybercriminals, leading to creation of more accounts. On the other hand, if email accounts are easier to detect and link to an attack, this type of IoCs would be published more frequently than others.

By contrast, filenames and paths, hash values or registry entries all define some aspect of a piece of malware, which requires a significant amount of work to be produced. This could explain why they are less common than email accounts. However, anti-malware rules based on filenames and paths can be trivially bypassed by malware designers. Thus, publishing this types of IoCs could be perceived as less useful and therefore done rarely. Nevertheless, hash values are, by definition, a good way of describing a file, given that they are short but almost unique for each file. For this reason, hash values would be widely published whenever an attack including malicious files is carried out.

Finally, IP addresses and domains usually cost money, but are also a valuable asset for cybercriminals. URLs can be used to make the victim download malicious files or fill a phishing form, for example. Some of those could not even be purchased by cybercriminals, but refer to a compromised site that should now be considered malicious or dangerous. For example, let's picture a highly sophisticated attack occurred on Facebook. The attackers might change the login form in a way such that the plaintext passwords are sent to their own server, in an attempt to avoid trying to crack the hashed passwords. In this case, the url referring to Facebook[14] would be considered malicious at least as long as the attack is active. The same holds for IP addresses, but users or potential victims will probably be more reticent to accessing an url whose domain is not a symbolic name but an IP address. Thus, numeric IP addresses would make more sense to appear in behavior descriptions of pieces of malware or specific attacks, where the software is the agent that tries and connects using the numeric form of the IP address. This could justify the fact that we detected more URLs than IP addresses.

The precision value (0.951) is high, but it is not evenly distributed among every IoC type, as shown in figure 5.2. We note registry entries have a precision of 1. This is probably because the regular expressions used for extracting them are restrictive enough to ensure they only match the desired IoCs. In addition, there are few other reasons to speak about those types on specialized cybersecurity publications. Moreover, registry entries have a very specific format, and it is extremely unlikely that we have missed any of those during the extraction.

Filenames and paths are the opposite case, and are found with very low precision (less than 0.6). Even

[14] `https://facebook.com`

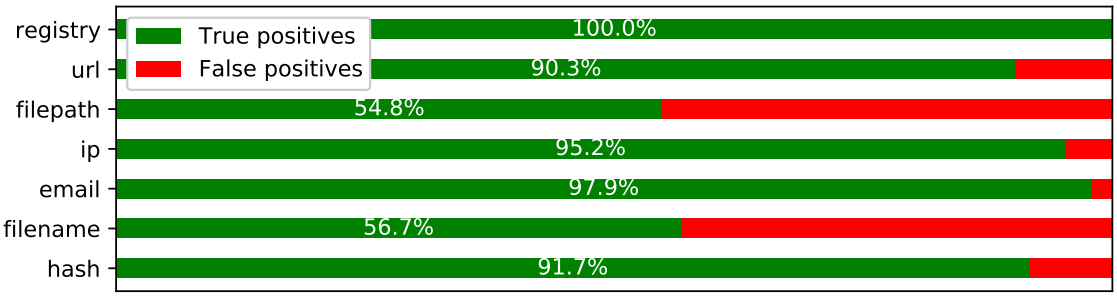

Figure 5.2: Precision of the IoC extraction by type

though they are not very common (see figure 5.1 and table 5.1), they usually appear as part of log transcriptions or commands in tutorial posts. Hence, it is very challenging to find their context and whether or not they should be considered malicious. In addition, we could not adapt the method we used for other IoCs in order to be valid for filenames and paths.

Hash values are detected with 0.917 precision. That is a good mark, in particular if we take in account that it is extremely unlikely for an arbitrary file to have a specific hash value which was calculated from another input. Therefore, the 8.3% false positives should not in principle be an important issue. However, there is still room for improvement, notably in the cases where the hash value is presented next to a filename without much context.

URL and IP addresses are detected with a fairly high precision too (above 0.9), with IP address detection being more accurate than URL (0.95 compared to 0.9). The reason for this difference is probably that URLs are more commonly used to refer other internet resources than IP addresses. Even in the context of an attack description, IP addresses will most likely refer to details of an attack, while URLs could also refer to additional resources such as information o other attacks. As mentioned in chapter 4.2, `<a>` tags whose `href` are the same as their text are deleted. This filter could be refined to improve the precision. For example, some `href` fields might include query parameters or anchors that are not in the text of the tag.

Email accounts are the most accurately detected of all IoCs (0.97). The reason is probably similar to IP addresses: in the context of cybersecurity blogs, it is not unusual to find lists of malicious email accounts, while legitimate accounts might appear more scarcely.

Chapter 6

State Of The Art

In this chapter, we discuss about closely related previous works. Automatic cyber threat intelligence extraction has received some attention. Most approaches [14, 9, 18, 3, 26, 22, 20] use Twitter as their source of information, but others [21, 28] use blogs, either in a hybrid blog-twitter approach or only blogs. Likewise, the machine learning approach is common for classifying whole documents [14, 9, 3, 26, 22], but NLP methods are also widely used for extracting specific information [18, 21, 28].

6.1 Extracting CTI using Machine Learning

Le et al. [14] use a Centroid novelty classifier on tweets, to differentiate between cyber-threat relevant or irrelevant. Dionisio et al. [9] also process tweets, classifying and labeling them regarding the target, attack or vulnerabilities involved. They gather tweets from a set of selected accounts and pre-filter them using a list of keywords defining the assets. Then a convolutional neural network classifies the tweets as relevant/irrelevant and a bidirectional long short term memory labels the relevant ones. Similarly, Alves et al. [3] extract intelligence as IoCs from Twitter. To do so, they gather tweets from a set of accounts, filter them using keywords referring to a list of assets and classify them as relevant or irrelevant using an SVM. After that, the tweets are clustered using Clustream [2] and IoCs are generated by tagging each cluster's centroid, using regular expressions. Trabelsi et al.[26] monitor twitter in order to find publication of zero-day vulnerabilities. They collect tweets by searching for relevant terms, cluster them comparing keyword frequencies and drop clusters that match keywords with known CVE descriptions. Additionally, they use a trust model for assessing the information quality of the tweets. Sceller et al. [22] try to detect and classify cyber security events that are discussed on twitter. They gather tweets by searching keywords and cluster them using an otimized version of 1-nearest neighbour. Then, they compare frequency of keywords to classify the clusters. Additionally, they detect new keywords to updated the search terms.Sabottke et al. [20] detect usage of exploits in the wild by searching for the term "CVE" on twitter and classify the results in threat categories using a SVM.

We notice there is a trend towards a process which first gets the documents (in this case mostly from twitter), classifies them as relevant or irrelevant, clusters them by topic extracts information from each cluster. However, some works do only a part of this process. In our case, we follow the novelty detection approach as [9, 3], using a one-class SVM, but we didn't cluster the documents later. That is because we think twitter publications will, by nature, feedback each other into talking about specific topics, but blog posts are both less abundant and more likely independent from each other. Additionally, we don't extract information using pure machine learning methods such as a bidirectional long short term memory [9] or an SVM [20], nor we use simpler methods like bare regular expressions [3] or word frequency alone [26, 22]. However, integrated both of the simpler methods in our work.

6.2 Extracting CTI using NLP

Mittal et al. [18] extract cyber intelligence from the Twitter stream and generates report according to an ontology. They use a list of keywords for Twitter search and use a named entity recognizer for adding semantics to each term in the tweet. Then the ontology is created using an extended form of the Unified Cybersecurity Ontology [24] and uses semantic web rules for reasoning about its content and generating alerts. Sapienza et

al. [21] try to find terms referring to cyber threats. To do so, they consider twitter but also blogs and dark web forums, filter out known words and check co-ocurrence of unknown words with threat-related terms. Also outside twitter, Zhao et al. [28] extract indicators of compromise from text in blogs, focusing on domain tagging. They use a convolutional neural network for domain identification and syntactic dependence for IoC extraction. Likewise, Liao et al. [15] use syntactic dependence and graph mining techniques to extract and validate IoCs, actually achieving a very high precision of 0.98.

In our work we also consider sources outside twitter. Additionally, instead of using a named entity recognizer [18], we use syntactic dependence to reason about the found IoCs [28, 15], but we do not use neural networks nor graph mining. Additionally, we had a slightly worse result than Liao et al. **??** (0.95 compared to 0.98).

6.3 Summary

In summary, most approaches follow to some extent the process shown in Figure 6.1.

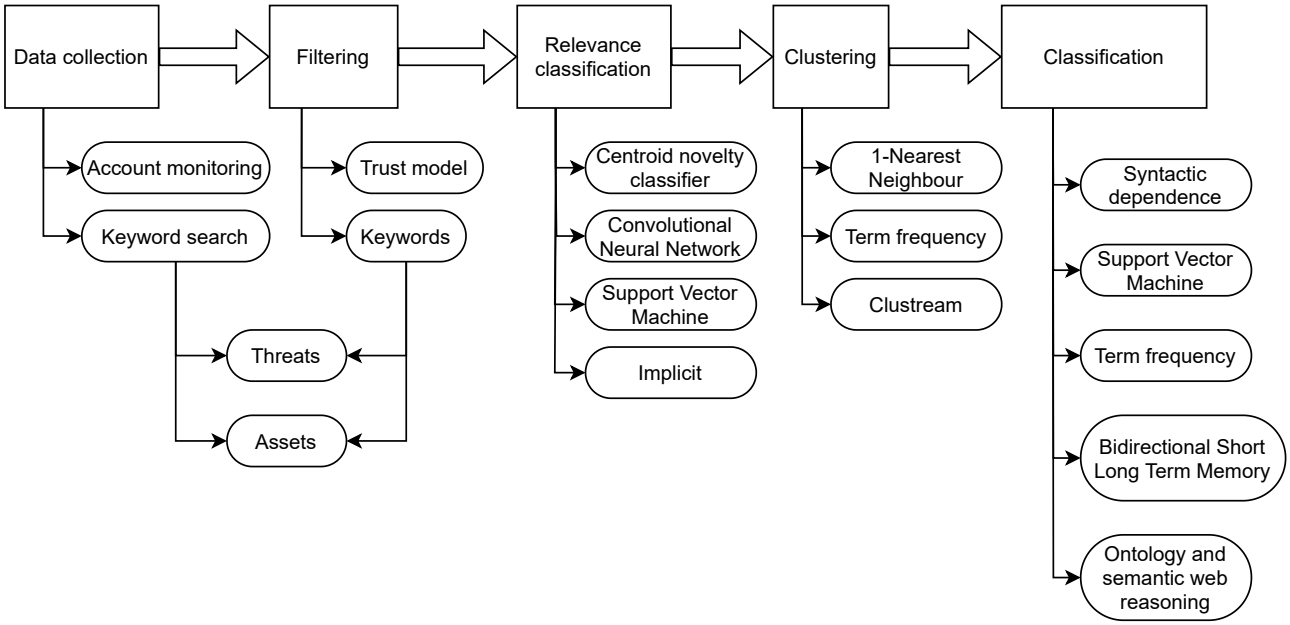

Figure 6.1: Summary of the CTI extraction process generally followed in the literature

First of all, data is collected. That can be achieved by monitoring twitter accounts, performing a search on keywords, monitoring blogs or forums, etc. After that, some form of filtering might be carried out. This can be by using a threat model or by checking occurrence or absence of keywords. In both this step and the previous one, keywords usually refer to either threats or assets, or both. After the first filtering stage, the documents are classified in relevant/irrelevant. This step usually involves machine learning, using classifiers such as a centroid novelty classifier, a convolutional neural network or a support vector machine. However, some works simply assume their gathered documents are relevant because of their method for data collection and filtering. The next step is usually clustering. Here, machine learning and natural language processing can mix, with some works using classic clustering algorithms such as 1-nearest neighbour, modern ones such as clustream, or a more NLP oriented one, like the term frequency. Finally, the classification phase tells whether a piece of CTI is valid or not. In this case, as well as in the previous step, NLP and machine learning mix, thus, we can see works using syntactic dependence and term frequencies, but we can also find support vector machines and bidirectional long-short term memories.

In our case, we collected data from blog posts and articles, used a SVM novelty detector as well as a topic modelling tool for relevance classification, didn't do clustering and used syntactic dependence for classification

Chapter 7

Conclusions

In this work we presented *IocMonitor*, a piece of software for automatically extracting cyber threat intelligence from publicly available sources such as blog posts, in the form of indicators of compromise.

The purpose of this work was to provide a reliable method for discarding false positive IoC, so that *IoCMonitor* can be used continuously to provide data to cyber threat intelligence platforms with minimum human supervision or intervention.

We have shown a method with a high precision (0.951) using *Mallet* and a one-class SVM as a topic modelling tool, regular expressions to find possible IoCs and syntactic dependency analysis to assess the validity of those IoCs and discard the false positives.

The work was focused on effectively discarding false positives on IP addresses, URLs, email accounts and hash values, thus not being very effective in assessing the validity of filename and filepath IoCs.

Further work could improve the performance of the proposed system by refining some of the pre-filtering, in particular which URLs are deleted from the page before processing, as stated in Chapter 5.3. Also, more keyword verbs could be proposed for the validation phase. Moreover, automatic conversion to an standard CTI exchange format could be added.

Appendix A

Acronyms

CTI	*Cyber Threat Intelligence*
IoC	*Indicator of Compromise*
NLP	*Natural Language Processing*
ML	*Machine Learning*
SVM	*Support Vector Machine*

Appendix B

Software library versions

beautifulsoup4	4.6.0
bs4	0.0.1
iocextract	1.13.1
mysql-connector-python	8.0.24
nltk	3.5
numpy	1.19.5
pickleshare	0.7.4
rake-nltk	1.0.4
regex	2020.11.13
reportlab	3.4.0
requests	2.21.0
scikit-learn	0.23.2
scipy	1.5.4
selenium	3.141.0
spacy	3.0.6
textacy	0.10.1
tokenizers	0.9.4
urllib3	1.22

Bibliography

[1] Md Sahrom Abu et al. “Cyber threat intelligence–issue and challenges”. In: *Indonesian Journal of Electrical Engineering and Computer Science* 10.1 (2018), pp. 371–379.

[2] Charu C. Aggarwal et al. “A Framework for Clustering Evolving Data Streams”. In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. VLDB ’03. Berlin, Germany: VLDB Endowment, 2003, 81–92. ISBN: 0127224424.

[3] Fernando Alves et al. *Processing Tweets for Cybersecurity Threat Awareness*. 2019. arXiv: 1904.02072 [cs.CR].

[4] Sean Barnum. “Standardizing cyber threat intelligence information with the structured threat information expression (stix)”. In: *Mitre Corporation* 11 (2012), pp. 1–22.

[5] Tim Berners-Lee, Roy T. Fielding, and Larry M Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan. 2005. DOI: 10.17487/RFC3986. URL: https://rfc-editor.org/rfc/rfc3986.txt.

[6] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. ” O’Reilly Media, Inc.”, 2009.

[7] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 1992, pp. 144–152. ISBN: 089791497X. DOI: 10.1145/130385.130401. URL: https://doi.org/10.1145/130385.130401.

[8] Burton DeWilde. *textacy: NLP, before and after spaCy*. https://github.com/chartbeat-labs/textacy. 2019.

[9] Nuno Dionsio et al. *Cyberthreat Detection from Twitter using Deep Neural Networks*. 2019. arXiv: 1904.01127 [cs.LG].

[10] Cath Everett. “Ransomware: to pay or not to pay?” In: *Computer Fraud & Security* 2016.4 (2016), pp. 8–12. ISSN: 1361-3723. DOI: https://doi.org/10.1016/S1361-3723(16)30036-7. URL: https://www.sciencedirect.com/science/article/pii/S1361372316300367.

[11] Matthew Honnibal and Ines Montani. “spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing”. To appear. 2017.

[12] InQuest. *iocextract (v1.13.1)*. https://github.com/InQuest/python-iocextract. 2019.

[13] Dom Lachowicz. *Enchant spellchecking library*. https://github.com/gfek/Hunting-New-Registered-Domains. 2013.

[14] Ba Dung Le et al. *Gathering Cyber Threat Intelligence from Twitter Using Novelty Classification*. 2019. arXiv: 1907.01755 [cs.CR].

[15] Xiaojing Liao et al. “Acing the IOC Game: Toward Automatic Discovery and Analysis of Open-Source Cyber Threat Intelligence”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: Association for Computing Machinery, 2016, pp. 755–766. ISBN: 9781450341394. DOI: 10.1145/2976749.2978315. URL: https://doi.org/10.1145/2976749.2978315.

[16] Nate Lord. *What are Indicators of Compromise?* 2020. URL: https://digitalguardian.com/blog/what-are-indicators-compromise.

[17] Andrew Kachites McCallum. "MALLET: A Machine Learning for Language Toolkit". http://mallet.cs.umass.edu. 2002.

[18] S. Mittal et al. "CyberTwitter: Using Twitter to generate alerts for cybersecurity threats and vulnerabilities". In: *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2016, pp. 860–867. DOI: 10.1109/ASONAM.2016.7752338.

[19] Nitin Naik et al. "Cyberthreat Hunting - Part 1: Triaging Ransomware using Fuzzy Hashing, Import Hashing and YARA Rules". In: *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. 2019, pp. 1–6. DOI: 10.1109/FUZZ-IEEE.2019.8858803.

[20] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. "Vulnerability Disclosure in the Age of Social Media: Exploiting Twitter for Predicting Real-World Exploits". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 1041–1056. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/sabottke.

[21] Anna Sapienza et al. "DISCOVER: Mining Online Chatter for Emerging Cyber Threats". In: *Companion of the The Web Conference 2018*. International World Wide Web Conferences Steering Committee. International World Wide Web Conferences Steering Committee, 2018, 983–990.

[22] Quentin Sceller et al. "SONAR: Automatic Detection of Cyber Security Events over the Twitter Stream". In: Aug. 2017, pp. 1–11. DOI: 10.1145/3098954.3098992.

[23] Bernhard Schölkopf et al. "Support Vector Method for Novelty Detection". In: *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS'99. Denver, CO: MIT Press, 1999, pp. 582–588.

[24] Zareen Syed et al. "UCO: A unified cybersecurity ontology". In: *Workshops at the thirtieth AAAI conference on artificial intelligence*. 2016.

[25] Wiem Tounsi and Helmi Rais. "A survey on technical threat intelligence in the age of sophisticated cyber attacks". In: *Computers & Security* 72 (2018), pp. 212–233. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2017.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0167404817301839.

[26] S. Trabelsi et al. "Mining social networks for software vulnerabilities monitoring". In: *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*. 2015, pp. 1–7. DOI: 10.1109/NTMS.2015.7266506.

[27] Rishi Vaidya. "Cyber security breaches survey 2019". In: *Department for Digital, Culture, Media and Sport* 66 (2019).

[28] Jun Zhao et al. "TIMiner: Automatically extracting and analyzing categorized cyber threat intelligence from social data". In: *Computers & Security* 95 (2020), p. 101867. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2020.101867. URL: http://www.sciencedirect.com/science/article/pii/S0167404820301395.

[29] gfek. *Hunting newly registered domains*. https://github.com/gfek/Hunting-New-Registered-Domains. 2018.

[30] zeroth. *Top One Million Ranked Websites as ranked by Alexa*. https://github.com/zer0h/top-1000000-domains. 2016.