h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

University of Applied Sciences Darmstadt

Department of Computer Science

Conception and Implementation of an Open-Source Threat Intelligence System

Dissertation

for the Acquisition of the Academic Degree

Bachelor of Science (B.Sc.)

submitted by

Marius Benthin

Matriculation Number: 750131

Supervisor	:	Prof. Dr.-Ing. Oliver Weissmann
Co-Supervisor	:	Björn Bär

Marius Benthin: *Conception and Implementation of an Open-Source Threat Intelligence System*, © March 6, 2020

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, March 6, 2020

Marius Benthin

ABSTRACT

Over the past years, threat actors have continually improved their development life-cycle to swiftly build undetected versions of malware in time. However, malware will be discovered by anti-virus scanners sooner or later. Usually, the goal for malware authors is to have the sample stay undetected for as long as possible.

At the same time, social media networks and public analysis systems pave the way for threat analysts to resist the growing number of sophisticated malware. They enable timely exchange of information through the research community and sharing of samples. Malware analysts often collect this open-source intelligence manually to identify new attack strategies and take appropriate countermeasures.

In order to support their task, this thesis proposes the conception and implementation of an Open-Source Threat Intelligence system. It automatically gathers Indicators of Compromise with special regard to the platform Twitter.

First, the system monitors Tweets based on both the most frequently used keywords and respectively recognized users in the field of cyber security. Subsequently, it filters incoming Tweets to extract relevant messages only. The applied filter uses the best of several tested classifier models which have been trained with more than 14,000 labeled Tweets beforehand.

In addition, links to malware analysis platforms and blog articles are scanned for associated file hashes. Once a Tweet exceeds a certain distribution threshold, the system generates alert messages to notify malware analysts within the company.

Eventually, the results clearly show that emerging threat intelligence can be extracted automatically and analyzed accurately with a low false positive rate. Therefore, malware analysts recognize spreading threats faster.

ZUSAMMENFASSUNG

In den letzten Jahren haben Cyber-Kriminelle ihren Entwicklungszyklus stetig verbessert, um zeitnah unerkannte Versionen von Schadprogrammen herauszubringen. Früher oder später werden Schadprogramme von Anti-Virus Scannern entdeckt, wobei Malware Autoren üblicherweise ihre Dateien solange wie möglich unerkannt halten möchten.

Zeitgleich ermöglichen auch soziale Netzwerke und öffentliche Analyse-Plattformen der steigenden Anzahl an optimierten Schadprogrammen standzuhalten. Mit Hilfe der Systeme können Sicherheitsforscher in kürzester Zeit Informationen und Dateien mit der Community teilen. Malware Analysten sammeln diese öffentlich verfügbaren Daten meist manuell, mit dem Hintergrund neue Angriffsstrategien aufzudecken und entsprechende Gegenmaßnahmen einzuleiten.

Um diesen Prozess zu vereinfachen, wird in der folgenden Arbeit ein Konzept zur Implementierung eines Open-Source Threat Intelligence Systems vorgeschlagen. Hierbei werden automatisch Bedrohungsindikatoren, insbesondere im Hinblick auf die Plattform Twitter, gesammelt und ausgewertet.

Zu Beginn des Prozesses beobachtet das System neu erstellte Tweets von bekannten Nutzern, sowie Tweets mit häufig verwendeten Schlüsselwörtern im Bereich der IT-Sicherheit. Anschließend werden alle eingehenden Tweets entsprechend gefiltert, sodass nur noch die relevanten Nachrichten übrig bleiben. Der angewandte Filter verwendet einen Klassifizierungsalgorithmus, welcher nach Vergleich von verschiedenen lernenden Modellen ausgewählt wurde. Als Trainingsgrundlage wurden mehr als vierzehntausend annotierte Tweets verwendet. Des Weiteren folgt das System auch Verlinkungen zu Analyse-Plattformen und Blog Artikeln, damit weitere dazugehörige Dateiprüfsummen gefunden werden können. Sobald ein Tweet eine bestimmte Verbreitungsgrenze überschreitet, werden die Malware Analysten im Unternehmen darüber unverzüglich benachrichtigt.

Die Ergebnisse dieser Arbeit zeigen deutlich, dass neue Informationen über Bedrohungen automatisch gesammelt und analysiert werden können. Malware Analysten können somit schneller auf sich verbreitende Bedrohungen reagieren.

TABLE OF CONTENTS

I THESIS

1 INTRODUCTION 2
- 1.1 Acknowledgments . . . 3
- 1.2 Terminology . . . 4
- 1.3 Motivation . . . 7
- 1.4 Problem Formulation . . . 8

2 BASICS AND RELATED WORK 9
- 2.1 Open-Source Threat Intelligence . . . 9
- 2.2 OSINT Sources . . . 10
  - 2.2.1 Malware Sharing Platforms . . . 10
  - 2.2.2 Twitter . . . 11
  - 2.2.3 Pastebin . . . 11
  - 2.2.4 IT Security Blogs . . . 12
- 2.3 Related Work . . . 13

3 REQUIREMENTS ANALYSIS 15
- 3.1 Functional Requirements . . . 15
  - 3.1.1 Data Volume . . . 15
  - 3.1.2 Data Velocity . . . 15
  - 3.1.3 Feature Extraction . . . 15
  - 3.1.4 Security Measures . . . 16
  - 3.1.5 Data Classification . . . 16
  - 3.1.6 Extensibility . . . 16
  - 3.1.7 Partition Tolerance . . . 16
  - 3.1.8 Data Availability . . . 16
- 3.2 Non-functional Requirements . . . 17
  - 3.2.1 Performance . . . 17
  - 3.2.2 Reliability . . . 17
  - 3.2.3 Data Veracity . . . 17
  - 3.2.4 Maintainability . . . 17
  - 3.2.5 Usability . . . 17
  - 3.2.6 Supported Output Format . . . 18

4 CONCEPTION AND IMPLEMENTATION 19
- 4.1 Python . . . 19
- 4.2 Tweet Training Set . . . 20
- 4.3 Tweet Gathering Pipeline . . . 21
  - 4.3.1 User and Keyword Filters . . . 21
  - 4.3.2 Stream Modules . . . 22
  - 4.3.3 Message Broker . . . 23
- 4.4 Tweet Processing Pipeline . . . 25
  - 4.4.1 Workers . . . 25
  - 4.4.2 Capturing . . . 25

4.4.3 Curation . . . 25
4.4.4 IoC Extraction . . . 26
4.4.5 Pre-processing . . . 28
4.5 Tweet Classification . . . 30
4.5.1 Information Retrieval . . . 30
4.5.2 Classification Models . . . 32
4.5.3 Evaluation . . . 35
4.6 Persisting and Alerting . . . 37
4.6.1 Persisting . . . 37
4.6.2 Alerting . . . 38
5 CONCLUSION 39

II APPENDIX

A PYTHON CODE FOR THE CLASSIFIER EVALUATION 42

BIBLIOGRAPHY 47

LIST OF FIGURES

Figure 4.1 Tweet gathering pipeline 21
Figure 4.2 Tweet processing pipeline (see Chapter 4.4) 24
Figure 4.3 Tweet with obfuscation of URLs by @Cryptolaemus1 . 27
Figure 4.4 Pre-processing Tweets for feature extraction 29
Figure 4.5 Word2Vec example for word embedding 32
Figure 4.6 Different classification models 33
Figure 4.7 Triangle of Tweet relevance 38
Figure 4.8 Mattermost Tweet alert issued by @MBThreatIntel . . . 38

LIST OF TABLES

Table 2.1 List of malware sharing platforms 10
Table 4.1 Frequently used terms in Tweets about cyber security . 22
Table 4.2 Results for classification of relevant Tweets 35
Table 4.3 Results for classification of irrelevant Tweets 35

LISTINGS

Listing 4.1 Regex for MD5, SHA-1 and SHA-256 27
Listing 4.2 Regex for URI 27
Listing 4.3 Regex for IPv4 27
Listing 4.4 Regex for CVE 27
Listing 4.5 Regex for filename 27
Listing 4.6 Regex for filepath 27
Listing 4.7 Regex for email 27
Listing 4.8 Pre-Processing a Tweet published by @struppigel . . . 28
Listing 4.9 NER annotated Tweet published by @cyb3rops 31

LIST OF ABBREVIATIONS

IoC Indicators of Compromise

PDB Program Database

CVE Common Vulnerabilities and Exposures

TF-IDF Term Frequency–Inverse Document Frequency

NER Named Entity Recognition

GloVe Global Vectors for Word Representation

SVM Support Vector Machine

MLP Multilayer Perceptron

SGD Stochastic Gradient Descent

CNN Convolutional Neural Network

API Application Programming Interface

OSINT Open-Source Threat Intelligence

STIX Structured Threat Information Expression

Part I

THESIS

1 INTRODUCTION

According to the latest report of the Federal Office for Information Security in Germany, about 114 million new malware variants have been discovered, starting from June 2018 till the end of May 2019 [10]. While the amount of malicious programs has grown steadily over recent years, the effect of damage started to become more and more perceptible.

In May 2017, public attention was drawn to the ransomware WannaCry, which infected approximately more than 200,000 machines all over the world including critical infrastructure such as hospitals [17]. Only a month later, in June 2017, a second ransomware called NotPetya erupted, which particularly targeted companies in the Ukraine but also spread into other countries [26]. Hours before public institutions issued a warning people reported incidences in social media and shared knowledge about the threats.

Social networks have proven to be a treasury for event detection since they allow users from all over the world to produce, share and rate any kind of information not only in the instance mentioned above [9]. However, user-generated content is often unstructured, short and unreliable which impedes algorithms to extract knowledge straightforward [9].

In order to address this issue, researchers developed methods based on artificial intelligence to detect patterns or statistical distributions in huge amounts of natural language data. These approaches loom large when it comes to uncovering security incidences, for example recognizing malware outbreaks via social media. Currently, many organizations use cyber threat intelligence solutions to keep an overview about the threat landscape. While many commercial threat intelligence feeds are based on company-owned analysis data, the public sharing of threat information gains more and more importance. Threat analysts create communities to exchange knowledge regarding new discoveries and security breaches. The first reference for security researchers is often the fast moving platform Twitter because they quickly see recent activities about vulnerabilities and exploits.

Furthermore, Twitter users often link further material to enrich their contributions and enable other researchers to investigate a new threat. Therefore, sample shares such as VirusTotal or URLhaus are frequently used to distribute malware safely. Additionally, online analysis sandboxes like Hybrid Analysis execute the malware inside an isolated environment and provide the captured behavior to the community.

Overall, threat intelligence today is widely spread but dispersed and seemingly unclear. Consequently, a system is required to extensively collect, process and structure the information pieces on the web. It monitors the respective platforms and passes the aggregated results to the analysts to take countermeasures.

1.1 ACKNOWLEDGMENTS

Acknowledgments go to my supervisors Prof. Dr.-Ing. Oliver Weissmann and Björn Bär from the University of Applied Sciences Darmstadt for guiding me throughout the entire project period and offering me to write my bachelor thesis in cooperation with the G DATA Cyber Security AG.

To my advisor Van Vi and colleges from G DATA, for teaching me about malware analysis and inspiring me to research in the field of threat intelligence.

Last but not least, thank you to my whole family Christiane, Andreas, Tobias, Ingelore and my girlfriend Ann-Kathrin for tirelessly supporting and motivating me during my studies.

1.2 TERMINOLOGY

In order to provide a fundamental understanding of the discussed topics, this chapter clarifies some of the frequently used terms in advance.

Indicators of Compromise

Indicators of Compromise (IoC) refer to a piece of evidence that is found on an infected machine [7]. They are mostly be used to uniquely identify a file, system path, network or email address. Regarding malware analysis, sample files are often referenced by a unique hash value such as SHA-256 or MD5.

Paste

The word Paste refers to a text-entity of the online service Pastebin[1], where any kind of text is stored and shared. Files have even been distributed via Pastebin as a Base64 string.

Common Vulnerabilities and Exposures

Common Vulnerabilities and Exposures (CVE) numbers specify public technical weaknesses or disclosures of popular software or systems [29]. The CVE number format consists of the discovery year and a consecutive number. CVE-2017-0144[2], for example, refers to a vulnerability in Microsoft's implementation of the SMBv1 protocol and has been exploited particularly by the malware WannaCry [17].

Term Frequency–Inverse Document Frequency

Term Frequency–Inverse Document Frequency (TF-IDF) represents a statistical approach to calculate the likelihood of occurrence for a single word inside a document stack [4]. The more often a word appears inside a document, the more important it is aligned to the word occurrence in the whole corpus. Consequently, TF-IDF metric is used to determine whether a given document is similar to other documents.

Named Entity Recognition

Unlike TF-IDF, the Named Entity Recognition (NER) tries to categorize each word into a predefined class that has been trained with well-known samples [4]. Subsequently, the distribution of classes inside a document provides indication about the similarity to other samples. This approach requires profound knowledge about the words used in the targeted domain.

1 https://pastebin.com
2 https://nvd.nist.gov/vuln/detail/CVE-2017-0144

Word Embedding

Word embedding is a technique where each word in a document is represented as a vector in a multidimensional space [4]. The objective is to cluster words that are used in the same context within a corpus to identify similar documents. Two frequently used algorithms are Word2vec and Global Vectors for Word Representation (GloVe).

Bag-of-words

The Bag-of-words model is another representation of a document where the words in each sentence are splitted into a list. In this context, the order and sentence structure is neglected, however, same words are added up to keep the occurrence frequency [4]. The model is often utilized in connection with feature extraction methods like TF-IDF.

Support Vector Machine

One-class Support Vector Machine (SVM) is a supervised machine learning classifier to predict whether a sample belongs to one of two classes [22]. First, the classifier needs to be trained with positive and negative samples to build a structure of comparative values. Afterwards, the classifier expects a numeric vector as input and assigns a binary value on behalf of the class.

Multilayer Perceptron

In machine learning, a Multilayer Perceptron (MLP) refers to a feed-forward neural network that expects several input values and generates one or more output layers [11]. Furthermore, it consists of a hidden layer with sufficient numbers of neurons which weigh out the given input values based on an activation function [11].

Activation Function

Furthermore, MLP uses activation functions to decide whether a neuron should be activated or not based on the input values [11]. While there are many different types of activation functions, the most widely used one is called "rectified linear unit". It maps all negative input values to zero which causes only about the half of neurons to be activated.

Stochastic Gradient Descent

The Stochastic Gradient Descent (SGD) is an iterative algorithm often used by MLP neurons to optimize a cost function [11]. It continuously adapts the function parameters until an approximated minimum has been found. This process is repeated several times with randomly selected samples out of the training set of every MLP.

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a regularized type of MLP and is commonly used for image recognition [11]. It has also been used in natural language processing, particularly when it is connected with word embedding [14].

F1-Score

The F1-Score is a metric to calculate the balance between precision and recall values of a binary classification model [4]. The higher the value, the less false positives and false negatives are encountered. It is calculated using the following formula [6]:

$$F1 - Score = 2 * \frac{precision * recall}{precision + recall}$$

V-Model

In connection with Big Data, the V-Model describes the most important characteristics for dealing with high data volume. Each criteria starts with the letter V and addresses one particular issue which occurs when processing data. The most significant characteristics are called volume, velocity, variety and veracity. In this thesis, an additional term called virality describes how often a user-generated item is shared [12]. The more often the item is shared, the higher the virality value.

1.3 MOTIVATION

During my work at G DATA in the virus analysis department, I was able to closely observe the daily issues of malware analysis.

It was a common and exhausting task for my fellow malware researchers to search for suspicious and unknown samples in the wild. Once a new malware variant was found and analyzed, an appropriate detection to recognize the malicious behavior was created.

A frequently used starting point of the search for novel samples was the social network where threat analysts share their recent discoveries. This everyday task is extremely time-consuming due to its high paced nature. It is also rather difficult to keep track of all new security events because of the diverse distribution methods.

Moreover, shared threat information does not always have the same quality. If the information has no IoC included for example, it is harder for malware analysts to write a proper detection.

Furthermore, examining all platforms manually and without any tools is impossible and often referred to as a waste of time.

Consequently, I considered it high time to develop a tailor-made system dedicated to gather IoC and download samples from publicly available sources and started reading about some early approaches already made. Fortunately, I soon came across the Twitter Streaming Application Programming Interface (API), offering easily accessible Tweets in near real-time. By taking advantage of this technology, it is possible to extract file hashes shared by well-known Twitter users automatically.

Discovering this opportunity, was the final motivation for me to start programming my own specific threat intelligence system that supports my malware analysis colleagues in their everyday tasks. I was curious regarding the challenges ahead and excited for possible extensions to further social network applications in the future.

1.4 PROBLEM FORMULATION

The game of cat-and-mouse between threat actors and the anti-virus industry has been there for quite a while but it speeds up rapidly. For example Emotet, originally a sophisticated banking trojan first discovered in 2014, continuously modifies its design to stay undetected on the victims machine [18]. Social networks and public analysis platforms are frequently reporting new malware variants and provide related IoC for further analysis.

However, the amount of security-related news published on the Internet is still too high to manually review each event on its own. In particular, anti-virus companies are interested in the rank and file of malware samples to create and optimize their detections. In order to protect customers, an automated process is required that collects and analyzes IoC to identify undetected malware samples.

The most valuable sources for hunting threat intelligence are malware sharing platforms and social networks. While processing structured data from sharing platforms is quite trivial, the extraction of unknown intelligence out of data written in natural language is more sophisticated. Nevertheless, recent research shows that it is feasible to extract information about vulnerabilities from user-generated content. Therefore, methods of artificial intelligence have been applied to filter and classify natural language text samples into categories. However, up to now no suitable solution for malware analysts to automatically collect, enrich and prioritize IoC derived from informal text like Tweets is available.

Consequently, a system has to be developed that monitors an extensible number of public threat intelligence platforms to collect novel IoC and undetected samples. With regards to Twitter, meaningful tracking words have to be identified which are mentioned in the majority of security-related Tweets. Additionally, Twitter communities in which threat analysts share their recent findings need to be discovered and have to be continuously observed.

At this point, it is critical to avoid disinformation by selecting only the most reliable users from these communities and develop appropriate safety precautions to prevent attacks on the system.

Furthermore, the system has to submit unknown samples to the automated malware analysis sandboxes and generate custom alerts about new threats. Mainly the latter requires a classification into relevant and irrelevant information with the goal to provide only unknown but suspicious samples to the malware analysts.

It has to be taken into account that within this research project only the free Twitter Standard API with several access restrictions can be used.

Finally, several issues coming with web applications have to be addressed such as downtime of external infrastructure.

The work begins reviewing what has already been researched in the field.

2 BASICS AND RELATED WORK

First, the common understanding of Open-Source Threat Intelligence (OSINT) is explored before related papers are briefly reviewed as basis for the thesis ahead.

2.1 OPEN-SOURCE THREAT INTELLIGENCE

In this day and age, the act of information sharing is one of the primary success factors to establish a sustainable IT business. A vital contribution to the knowledge exchange can be attributed to public available data sources. They often provide information to people anywhere in the world and allow them to take advantages of it. This kind of data is called OSINT and is described by the US Central Intelligence Agency as follows:

"OSINT is drawn from publicly available material, including:

- The Internet
- Traditional mass media (e.g. television, radio, newspapers, magazines)
- Specialized journals, conference proceedings, and think tank studies
- Photos
- Geospatial information (e.g. maps and commercial imagery products)" [5].

One special type of OSINT, that has evolved in recent years, is Open-Source Threat Intelligence. It designates to the process of gathering and analyzing publicly available information about cyber attacks or malicious programs.

The data often includes IoC that enable malware analysts to uniquely identify threats or security incidences. For example, if a new malware variant has been discovered, a related sample can be referenced using a hash function. Commonly, malware analysts use one of the three hash values including SHA-256 or the already deprecated algorithms MD5 and SHA-1 [28]. Further frequently mentioned IoC are CVE numbers, referring to a specific vulnerability that has been exploited.

As everyone can contribute to OSINT, it is highly recommended to verify its origin and veracity. It is already known that threat actors use deception techniques to specifically spread false flags and confuse threat analysts [13]. Therefore, OSINT should always be enriched with internal information and cross-validated over multiple platforms.

Since public malware samples can be downloaded and distributed by anyone on the Internet, it is essential that anti-virus solutions detect these files.

2.2 OSINT SOURCES

Open-Source Threat Intelligence is found on various websites and applications on the Internet. The following chapter examines sources frequently used by malware analysts to share information about new threats.

2.2.1 *Malware Sharing Platforms*

The most evident sources for public threat intelligence are malware sharing platforms where people upload samples to determine whether they are malicious. Many of these platforms work together with anti-virus vendors in order to scan the submitted files, using their protection technologies.

New unknown samples are often executed in isolated sandboxes to capture malicious behavior and take screenshots. The gained threat intelligence is commonly stored in a structured language that is machine-readable, extendable, consistent and exchangeable throughout organizations [7]. For instance, OpenIOC is based on the XML schema with more than 500 predefined IoC attributes to describe a threat or incident [19].

Another OSINT standard is called Structured Threat Information Expression (STIX), that has been published by the OASIS CTI Technical Committee and is based on the JSON format. Importing this kind of threat intelligence is quite easy as it is often well prepared and enriched with the necessary information.

A list of exclusive malware sharing platforms amongst others:

PLATFORM	VENDOR	LINK
VirusTotal	Google Inc.	virustotal.com
Hybrid Analysis	CrowdStrike	hybrid-analysis.com
URLhaus	abuse.ch	urlhaus.abuse.ch
Open Threat Exchange	AlienVault Inc.	otx.alienvault.com
MalShare	Community-driven	malshare.com
VirusShare	J-Michael Roberts	virusshare.com
VirusBay	Ido Naor & Dani Goland	virusbay.io
Maltiverse	Maltiverse	maltiverse.com
ANY.RUN	ANY.RUN LLC	app.any.run

Table 2.1: List of malware sharing platforms

2.2.2 *Twitter*

In the past, Twitter has turned out to be a great platform for applying arbitrary machine learning tasks regarding event detection and analysis [9]. The timely exchange of Tweets and easy access to the Twitter platform allow researchers to gather fresh and at least partly structured text messages from public user profiles.

Twitter provides different approaches to connect to its Twitter Developer API [30]. One of the API methods, called Twitter Stream, enables the automatic collection of Tweets in near real-time. Public Tweets are filtered by a custom list of either textual keywords, user accounts or geographic locations.

However, Twitter restricts the amount of entries inside a filter depending on the subscription level. The free standard API offers programmers to track 400 keywords, 5,000 user IDs and 25 locations at once. Later on, these limitations will impact parameters of the proposed Open-Source Threat Intelligence implementation.

Furthermore, Twitter defines several rules to track Tweets by keywords. First, the capitalization and order of each keyword inside the Tweet is ignored. A Tweet matches, if the keyword is contained as a word inside the text, hashtags, expanded or display URLs, as well as inside a mentioned Twitter user name.

Meanwhile, punctuation is not considered inside the Tweet itself. The punctuation used in the keyword list is treated as a significant character. For example, the keyword »malware« catches Tweets containing »Malware.«, »„MALWARE“«, »#malware« and also »MalwareMalware«.

In addition, it is possible to filter keyword junctions like »emotet trojan«, which will match any Tweet with both keywords.

Another special case comes with URLs where only the complete domain name and path are exactly the same without considering the URL scheme or "www." in front of the domain name.

Eventually, the choice of proper keywords is important to receive an extensive and highly tailored stream of Tweets.

2.2.3 *Pastebin*

During the observation of Tweets closely related to malware analysis reports, one platform has been frequently linked inside the Tweets apart from malware sharing platforms.

Pastebin, a website to upload raw text documents, is often used by malware analysts to share multiple IoC that do not fit into a single Tweet or even entire malicious files with the community. For a small fee Pastebin allows developers to scrape public accessible pastes via their scraping API.

Up until now, the methods are limited to retrieve either single pastes by their identifier or the whole stream of all pastes without any filtering except the amount. Since it is currently impossible to fetch pastes by user profiles

or keywords, it takes great effort to filter relevant pastes from those which are unrelated to malware.

2.2.4 *IT Security Blogs*

Social media networks like Twitter use short messages generated by any user, whereas technical and scientific articles are often long, detailed and created by security experts. They may also contain various IoC such as file hashes, Program Database (PDB) paths or obfuscated malicious links. Security analysts generally link their research reports in social networks to provide more detailed information that does not fit into a 280 character Tweet. For that reason, it is mandatory to fetch and analyze external security blogs such as Threatpost[1] or Securelist[2] as well.

1 https://threatpost.com
2 https://securelist.com

2.3 RELATED WORK

The established premises have been picked up by various researchers in the past. Especially regarding feature extraction and classification of Tweets about cyber security, advancing from longer scientific contributions.

One of the earliest approaches with regard to blog articles was conducted by Liao in 2016. He automatically extracts IoC and its coherence inside of technical articles, using a system called iACE [16]. Therefore, each sentence is translated into a dependency graph that describes the relation between IoC tokens and so called context terms. Out of 71,000 articles collected from 45 leading IT security blogs across 13 years, the system generates more than 900 thousand OpenIOC items. The precision of the generated IoC amounts up to 95 percent, while 10 percent of indicators inside the articles have not been detected.

One of the first to consider Tweets was Mittal, who developed a framework called CyberTwitter to generate timely threat alerts from Tweets for security analysts [21]. Based on a keyword list, the system analyzes new Tweets using a NER specialized on security vulnerabilities.

Further research was done by Le Sceller, who implemented a real-time and self-learned framework (SONAR) to detect security-related events on Twitter [27]. As an input, the system expects distinct taxonomy subjects with six categories such as vulnerability and malware. Each taxon is composed of a keyword list that is automatically extended over time, using the word embedding technique GloVe. The event clustering has been performed unsupervised with the locality sensitive hashing considering certain conditions. On the one hand, a Tweet has to exceed a certain threshold of the Shannon entropy. On the other hand, each cluster must include more than 10 Tweets of different authors. Over a time period of nine months in 2016 the system processed over 47,8 million Tweets and detect 759 events per month on average.

Subsequently, Behzadan gathered 21,000 cyber-related Tweets over a four-day period and verified two consecutive cascaded CNNs [3]. Therefore, the Tweets are first pre-labeled using the IBM Watson Natural Language Classifier[3] service, which suggested a cyber-related topic based on the Tweet content. Respectively, annotated Tweets serve as the initial training set to develop the cascaded CNNs that consist of a binary and multi-class model classifier. While the binary classifier decides whether a Tweet is relevant or irrelevant, the multi-class model assigns each relevant Tweet to one of the topics: vulnerability, DDos, ransomware, botnet, data leak, zero-day and general / marketing. The binary classifier achieves an F1-Score of 0.95 and the multi-class model an F1-Score of 0.82.

Dionísio, on the other hand, trained and applied a CNN to identify Tweets with security-related information about a certain IT infrastructure [8]. To achieve this, Tweets are first processed using a NER model that assigns a single label to each word. The labels are defined as organization, product or

3 https://www.ibm.com/cloud/watson-natural-language-classifier

asset, version number, threat or vulnerability and identifiers like CVE. Words that cannot be categorized are marked as irrelevant information. On three different data sets that consist of both positive and negative Tweets an average F1-Score of 0.92 has been achieved.

Almost simultaneously, Alves introduced a Twitter-based threat monitor (SYNAPSE) to filter, classify and cluster 195 thousand Tweets of 80 manually selected Twitter users [2]. To reduce the amount of Tweets the system selects only Tweets that match one term in a list of keywords regarding IT security infrastructures. Subsequently, the text is converted into a numeric vector by calculating the TF-IDF for each word inside the Tweet and the corpus. Afterwards, the extracted features were passed to both a SVM classifier and MLP to predict the relevance of a Tweet. The evaluation shows that the SVM classifier achieved the best accuracy with around 90 percent on three different data sets.

The same work also examines the ability to assign similar Tweets to clusters. Therefore, the researchers employ an online cluster, using within-cluster threat similarity algorithm and an offline cluster with the use of the k-means algorithm. Out of 1,754 relevant Tweets the clustering methods generate 65 clusters with five or more Tweets.

The latest paper was released by Le, who developed a novelty detection model to reveal recent discoveries about CVE on Twitter [15]. Since each CVE identifier has its own public vulnerability description, the model takes advantage of the current but also future threat summaries. First, the descriptions and Tweets are both pre-processed by removing stop words and irrelevant terms. Subsequently, the TF-IDF is calculated for each remaining word and the vectors are passed to four different classifiers, including the novelty detection model. To compare the accuracy, a subset of 3,000 Tweets from 50 Twitter accounts with more than 5,000 followers is taken and manually labeled. Accordingly, the training set consisting of 232 positive and 2,768 negative Tweets has been sent to four different classifiers, namely: SVM, MLP, CNN and Centroid (the novelty classifier). The Centroid computes the cosine similarity between the input Tweet and the Tweets labeled as positive. If the threshold is larger than the Centroid value, the Tweet is classified as abnormal and vice versa. With an F1-Score of 0.643 the Centroid achieved the best results followed by the one-class SVM with 0.629 accuracy. What stands out is that many Tweets often do not contain the CVE identifier.

These major works provide the foundation for the following system development, trying to improve shortcomings but also picking up the good approaches made before.

3 REQUIREMENTS ANALYSIS

The overall goal of the thesis system is to filter relevant Tweets and pass them on to malware analysts. Based on the described background with its respective conclusions and original ideas, the following pages give an in-depth look into the functional and non-functional requirements with particular focus on the V-model as means to the system's end. They are listed in no particular order.

3.1 FUNCTIONAL REQUIREMENTS

3.1.1 *Data Volume*

Both Twitter and malware sharing platforms produce a high volume of data over time. This data has to be stored efficiently in a database for example.

Furthermore, the principle of minimal data collection has to be applied without drawbacks to the data analysis and classification tasks to keep the quantity low.

3.1.2 *Data Velocity*

Closely connected to volume is the velocity which refers to the amount of incoming data in a certain period of time. The occurrence of Tweets about malware may arise, if a new exploit has been developed, on the one hand. On the other hand, the throughput it depends also on the time of day. Therefore, the system has to be able to process the amount of incoming messages at all times.

3.1.3 *Feature Extraction*

All the incoming filtered Tweets are first searched for IoC that leave a hint to malware information. This offers the opportunity to correlate Tweets with additional threat intelligence from other sources. The system has to detect the IoC, even if they are written in an obfuscated way.

Moreover, all the mentioned URLs have to be scraped once to identify file hashes that are referenced in technical blog articles. Feature extraction is thus a key functionality.

3.1.4 *Security Measures*

Handling user-generated content is always critical because it allows threat actors to spread fake information or even malicious code. For example, following aforementioned URLs to infected websites could lead to security issues.

Although, Twitter flags malicious and phishing links[1] it is essential to validate each input and store the results safely by escaping special character sequences and using prepared statements.

In addition, only encrypted network protocols must be used to communicate with external servers to prevent eavesdropping of API keys.

3.1.5 *Data Classification*

Once an IoC has been found either in a Tweet or behind a URL, a machine learning model has to predict whether the content is malware related or not. Based on a collection of already labeled Tweets, it has to determine the similarity to a new Tweet.

3.1.6 *Extensibility*

Extensibility has to be taken into account as well since OSINT is a fast moving field where platforms continuously arise and vanish. Functionalities of OSINT platforms are often added or deprecated to improve their services which requires developers to frequently check the API documentary. Consequently, the system has to be extensible in means of adding a wide variety of new or altered platforms.

3.1.7 *Partition Tolerance*

Moreover, OSINT services are not continuously available. They may not be reachable for several minutes, sometimes hours, or they block access to API functions because the request limit is exceeded. Therefore, the system has to comply with partition tolerance which means that it overcomes downtime of external and internal infrastructure but catches up the missing data holes afterwards.

3.1.8 *Data Availability*

Finally, the gathered threat intelligence must be available to malware analysts in order to get more details about a certain event. Therefore, the extracted IoC has to be enriched with internal threat intelligence. The more data points are connected with each other, the faster the malware analysis can be performed.

1 https://help.twitter.com/en/safety-and-security/phishing-spam-and-malware-links

3.2 NON-FUNCTIONAL REQUIREMENTS

3.2.1 *Performance*

Machine learning models vary in performance and therefore need to be well-balanced between accuracy and speed. Although the occurrence of false alerts has to be reduced to a minimum, it is not beneficial, if they are issued with a high delay. As a consequence, the classifier model has to warn malware analysts in less than a minute.

With regard to data velocity, attached workers have to be ready for immediate processing which requires the classification model to be build within less than a second.

3.2.2 *Reliability*

In addition, the OSINT system should be reliable with regard to the use of network application protocols. All targeted platforms have to be scraped regularly and unexpected server responses have to be logged for debugging. Issued alerts have to be transmitted to the recipients without loss of information. Accordingly, only application protocols based on connection-oriented communications should be considered.

3.2.3 *Data Veracity*

Similar to the security requirements, the veracity of information has to be validated before it is handed over to the analysts. In particular, user-generated content is prone to be false and unreliable because anyone can contribute to the community even without expert knowledge. Therefore, all gathered user data has to be examined and should not be used as a single decisive feature.

3.2.4 *Maintainability*

Apart from the initial programming and regular extensibility effort, manpower is required to maintain the system in a sustainable manner. Based on the latest experiences, one developer that monitors the current state of the external APIs and evaluates the produced logs once a week should be sufficient.

3.2.5 *Usability*

Usability is another non-functional requirement that should be considered to simplify the work for malware analysts. On the one hand, warning messages should be delivered through an intuitive channel that does not depend on a list of recipients. Furthermore, alerts have to be clearly structured and easy to understand to improve the workflows. In order to keep the amount

of alerts manageable, only Tweets that are shared more than three times should trigger a warning. More than three Re-Tweets discards too much valuable information.

3.2.6 *Supported Output Format*

Furthermore, output formats are structures that allow transport and migration of threat intelligence data. Hereby, the system relies on already defined languages such as STIX. Although there is currently no need to export threat intelligence, the databases should be designed with regard to STIX.

4 CONCEPTION AND IMPLEMENTATION

The following chapter describes the concept decisions and implementation details of the Open-Source Threat Intelligence system based on the aforementioned criteria.

First, the chosen programming language and compilation of the training set is explained to constitute the foundation of the following process.

Second, the Tweet gathering and processing pipeline are specified before the classification model and alerting mechanism are defined.

4.1 PYTHON

The system is written in the programming language Python 3.5.3. Python has been introduced in 1991 by Guido van Rossum and is used in many data analysis tools such as TensorFlow[1] [25]. Due to its clear syntax and good code readability, it is easy to use for beginners.

The comprehensive standard library of Python supports a lot of functions for straightforward use. Mainly HTTP communication, which is almost exclusively used by OSINT platforms to share information, is easy to implement in Python.

Furthermore, Twitter recommends to access their API via the Python library Tweepy[2], which provides a stable framework for the stream functionalities [30].

Pika[3], another Python library, has been selected to implement the Advanced Message Queuing Protocol in order to communicate with the middleware.

The machine learning tasks are conducted using the scikit-learn[4] framework and the NLTK[5] library for natural language processing.

Although many instructions of the system can be executed simultaneously, the global interpreter lock in the CPython implementation prevents the use of multi-threading [24]. The additional overhead that comes with multi-processing is accepted due to the overriding benefits.

Other programming languages in my skill set such as Java and C++ were not an option mainly because they do not have as many and usable modules for data analysis compared to Python.

1 https://github.com/tensorflow/tensorflow
2 https://developer.twitter.com/en/use-cases/academic-researchers/helpful-tools
3 https://github.com/pika/pika
4 https://scikit-learn.org
5 https://www.nltk.org

4.2 TWEET TRAINING SET

Twitter serves as the primary source for feeding the OSINT system because it is well-known to produce valuable human intelligence about malware [8] [2] [27] [3] [21].

The process of classifying Tweets into relevant and irrelevant is a supervised machine learning task. Therefore, a function has to be created based on knowledge that already exists and is proven to be valid. Subsequently, the function is only as good as the provided training data.

A common approach is that experts of the targeted domain are asked to annotate a sample set manually. The resulting data set is often more precise than automatic tagging because human analysts are able to detect false positives more accurately. Nevertheless, supervised machine learning requires a huge amount of data to achieve a high validity. Manually conducted, however, this task is rather time-consuming and expensive.

Therefore, contrary to related works that label relevant Tweets manually, this project aims to develop an improved automated approach with the use of internal G DATA intelligence. The first step is to gather a large amount of Tweets from the Twitter Stream API, deriving from commonly used keywords about cyber security. Additionally, user profiles of well-known malware analysts are also scraped to add older Tweets to the data set.

In the process, all Tweets without any file hash are discarded by default because at this point it is not possible to automatically recognize whether the Tweet is about malware or not. The underlying assumption is that all Tweets with a file hash related to a malicious sample deal with malware. Respectively, each collected hash is looked up in the internal G DATA intelligence to check whether it belongs to a known malware sample or not. If the file hash is known to G DATA and classified as malicious, it is assigned to the training set. Otherwise, if the file hash is known to G DATA but not classified as malicious, it is labeled as irrelevant. Unknown file hashes are discarded without further analysis.

Subsequently, the whole training set covers 29,369 Tweets created between September 2017 and October 2019. It consists of 14,670 relevant Tweets that include at least one reference to a real malware sample and 14,699 irrelevant Tweets with no obvious relation to malware.

Overall, the automatic process of labeling the Tweets is significantly faster than the manual approach. For each file hash it takes less than a second to return a verdict while a malware analyst takes more than three seconds. Within the framework of an anti-virus company the automatic approach proofs to be cost-effective without losing accuracy of the training set.

4.3 TWEET GATHERING PIPELINE

Based on the knowledge deriving from the training data, the system has been developed from the ground up. Therefore, the Tweets have to be filtered from the stream and temporarily stored in a queue. This Tweet gathering pipeline is explained in the following section.

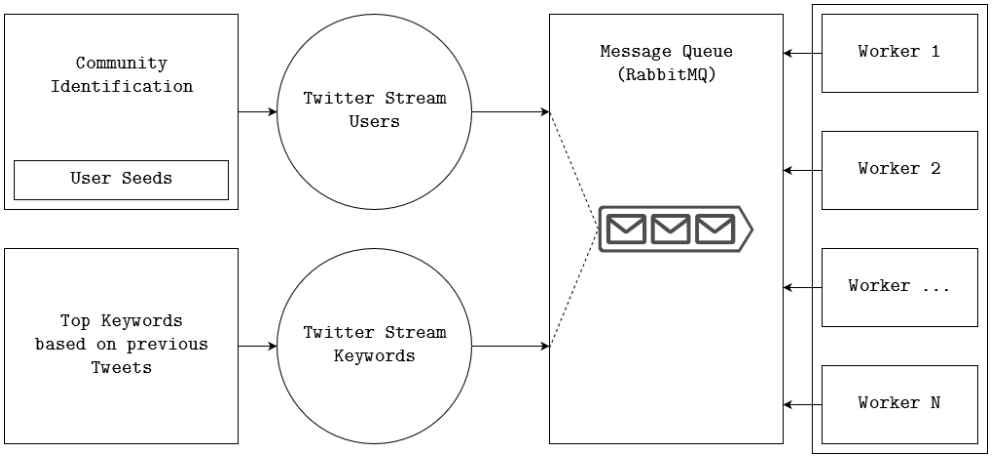

Figure 4.1: Tweet gathering pipeline

4.3.1 *User and Keyword Filters*

As already mentioned before, there are two different approaches to filter Tweets through the Twitter Stream API.

The first one expects a list of unique user identifiers that pertain to the Twitter profile of the malware researcher or security vendor. However, the API limits the amount of user IDs to 5,000 which means that only users who frequently post relevant content are tracked. Therefore, the malware analysts of G DATA were asked to provide a list of Twitter users that meet these requirements.

A seed list of approximately 50 users emerged that are used to identify further enriching profiles. All accounts that have been followed by users of this seed list are also analyzed to check whether any IoC is mentioned in their latest Tweets. If so, they are added to the tracking list in order to receive some more Tweets about cyber security.

As a result, more than 1,000 further users have been identified and added to the seed list. Sometimes users restrict the access or delete their profiles, so they have to be removed. Thus, all user profiles are checked daily, if they either changed their settings or deactivated their account.

The second approach includes the exploration of frequently used keywords that are mentioned in Tweets about cyber security. Even though, the list of those expressive keywords collected by the system is longer than 4,000 keywords, the limitations of the Twitter Standard API restrict the amount to

only 400. At the moment, this issue requires a careful choice of the right words as long as the free API access is used.

Therefore, all Tweets out of the training set are divided into tokens. Frequently used stop words such as "the" or "and" are removed. Subsequently, the remaining words are ranked based on their occurrences. The highest ranked 400 keywords are selected for tracking. This process is repeated daily to stay up to date with terms of novel malware variants.

A selection of frequently used keywords is provided below.

emotet	virustotal	ioc	epoch	urlhaus
payload	sample	threat	analysed	sha
malware	trendmicro	md5	njrat	detected
ransom	ransomware	binary	botnet	apt
trojan	hash	active	url	captured
attacker	port	zip	tcp	file
subject	exe	submitted	revenge	analysis
banking	wildfire	rat	lokibot	targets
nanocore	malspam	detection	agenttesla	malicious
campaign	received	gandcrab	infected	security

Table 4.1: Frequently used terms in Tweets about cyber security

4.3.2 *Stream Modules*

Based on the updated list of user IDs and keywords, multiple sensors are required to receive new incoming Tweets and store them temporally for further processing.

Therefore, two services have been developed that listen to the Twitter Streaming API, using the Python module Tweepy[6]. One service is responsible to follow the user profiles and the other to track keywords.

At certain points, the velocity of Tweets is expected to increase rapidly in case any malware outbreak or vulnerability is discovered. Accordingly, the producer-consumer pattern is applied where the sensors represent the producers. As soon as a Tweet arrives, the sensor immediately pushes it as a JSON string into a message queue to be ready for the next one.

Since both services require a high availability and a fast processing rate, they do not solve any additional tasks to catch up with the continuous stream. In case the service is not able to process the throughput in a reasonable time, the stream automatically terminates. A huge amount of Tweets depending on the allocated memory can be held temporarily for further processing via the message queue.

6 https://github.com/tweepy/tweepy

Due to further Twitter limitations, only two stream connections can be used at the same time. The most useful for this project are one for users and one for keywords. It is still enough to cover a wide range of incoming Tweets.

4.3.3 *Message Broker*

The message broker is used as a middleware to store incoming messages and distribute them to an arbitrary number of workers. Therefore RabbitMQ[7], a lightweight and easy to integrate message broker provided under the Mozilla Public License, has been selected. A dedicated RabbitMQ process in the network serves as a bridge between the stream modules and the workers that process the Tweets.

7 https://www.rabbitmq.com

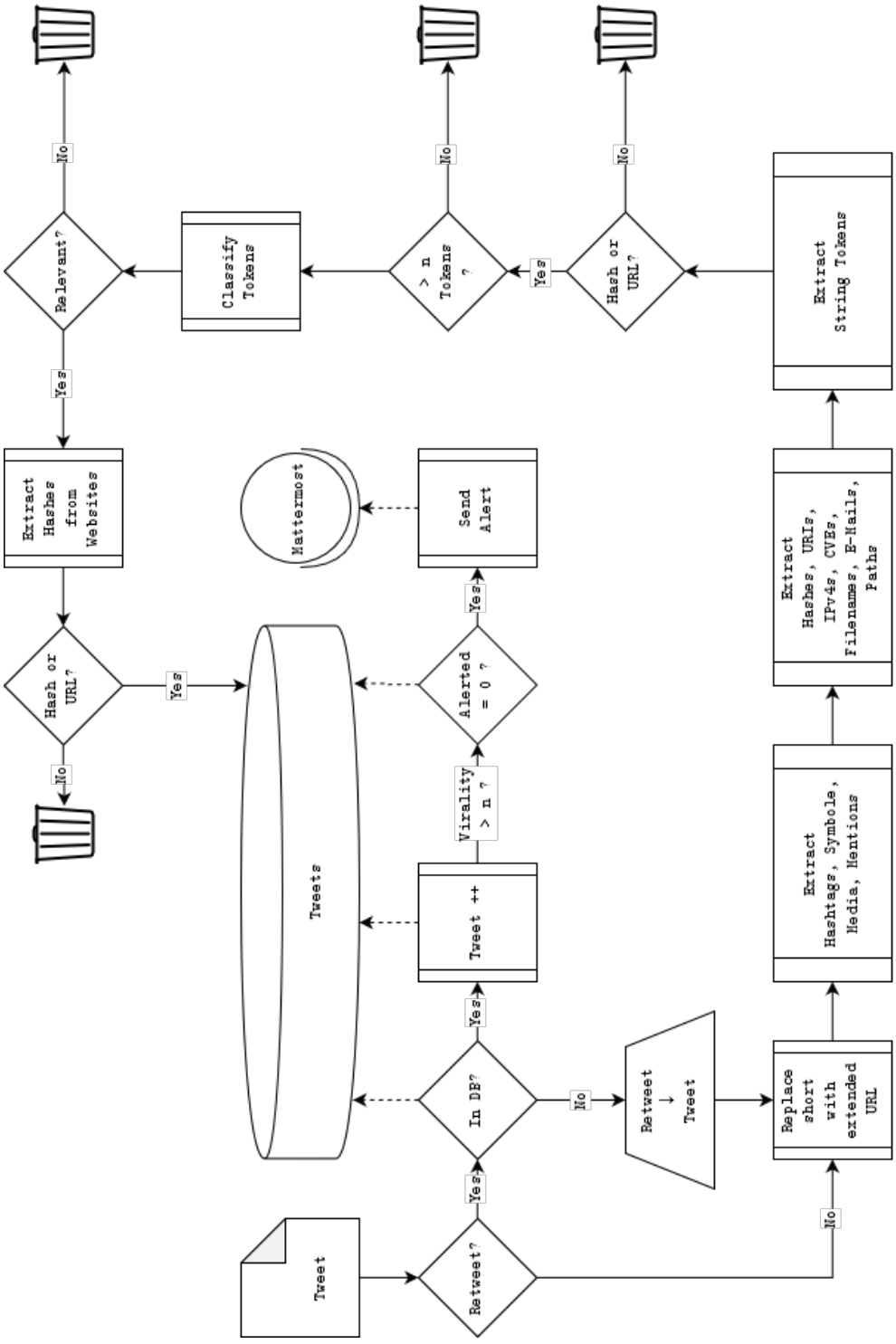

Figure 4.2: Tweet processing pipeline (see Chapter 4.4)

4.4 TWEET PROCESSING PIPELINE

After the Tweets are stored inside the message queue, a module is needed that works off the arising entries. The program sequences of this module is described as follows.

4.4.1 *Workers*

Workers are the core system components because they perform the extensive analysis and classification tasks of Tweets. They are defined as consumers because they continuously listen to the message broker for incoming Tweets and process them successively considering the principle of first in, first out. The number of active workers aligns with the data rate of Tweets which means that workers wake up to support dismantling the message queue. If the queue length decreases, the workers fall asleep again.

4.4.2 *Capturing*

Initially each worker checks whether the present Tweet is a new one or just shared by another user to identify the original Tweet. This step is required because Tweets are frequently shared by other users which causes the volume of data to increase fast while the value of the Tweet itself does not improve.

To tackle this issue, the Re-Tweet ID of the actually Tweet is looked up in the database to figure out whether it already exists. If so, a counter column inside the database table is incremented to define a virality value. Consequently, without loosing the information that a Tweet was shared again, the quantity can be reduced to a minimum.

A Tweet that has been shared more than three times triggers an alert message that transmits the Tweet over an incoming Mattermost[8] web-hook to the malware analysts. Tweets that do not exist in the database are inspected further to determine, if they are relevant.

4.4.3 *Curation*

Usually, Twitter reduces the length of large URLs by replacing it with a shortened URL that redirects to the actual one. However, many URLs pointing to analysis platforms also include a filehash as an unique resource identifier that is decisive during the filtering process.

For example, the short Twitter URL https://t.co/aNwXahHLuo redirects to Hybrid Analysis, while only the expanded URL https://hybrid-analysis.com/sample/750d172e2a... includes the file hash.

Therefore, all short URLs are first replaced with the extended URL that are also supplied within a Tweet message of the Twitter Stream API.

8 https://mattermost.com

Afterwards, all symbols, media URLs and user mentions are stripped. Otherwise, they distort the token and IoC extraction. However, hashtags are not removed because they are often part of complete sentences or act as highlighters.

4.4.4 *IoC Extraction*

Cleaned Tweets are the foundation of the IoC extraction. First the primary IoC, namely file hashes, and URIs are extracted to check whether the Tweet itself refers to malware. Additionally, information like CVE numbers, IPv4 and email addresses as well as file paths and names are searched.

Since file hashes follow a fixed character length and are expressed in hexadecimal notation, it is easy to recognize them inside natural text messages. However, file hashes are also used in other circumstances than cyber threats which requires semantic interpretation. For instance, SHA-256 is used by a Bitcoin miner to calculate the hash of a new block header that is less than or equal to the previous hashed transaction [1]. As Bitcoin hashes always start with exactly 16 zeros, they can be simply ignored, if the loss of a tiny minority of malware hashes is accepted by the stakeholders.

Furthermore,three frequently linked types of URIs have been treated separately by the system. The first ones are URIs pointing to malware sharing platforms like Hybrid Analysis. If such a link is recognized by the system, the IoC inside the URL path is extracted and processed by an API wrapper. Thus, the information about the sample is collected from the sources in a structured and more reliable way. As the sharing platform provides users to access samples, they are downloaded and imported into internal automated analysis systems.

Another type is blog articles where hashes are used to reference files or strings that belong to the website assets but not to the content. Before contacting a certain URL, the system first checks whether the page has been classified by G DATA as malicious or not. Only if the page is not malicious and has a high reputation according to G DATA, the content is fetched. Subsequently, the system removes scripts and style-sheet references from the raw page body before searching for IoC. If a file hash has been found, it is added to the list of hashes related to the Tweet.

The third type of URIs is directly pointing to malicious payloads or command and control servers via TCP and has to be treated with caution. Consequently, these URIs are not contacted during the IoC extraction, however, they are stored in the database to correlate them with other Tweets later on.

Commonly, many malware analysts obfuscate malicious URIs to prevent people from clicking on the links by accident. On the other hand, they want to impede the automated recognition of infected URLs via the Twitter API which causes the user to be blocked. Therefore it is difficult, for example, to identify and extract URIs that are not conform to RFC-3986 without loosing parts of the IoC.

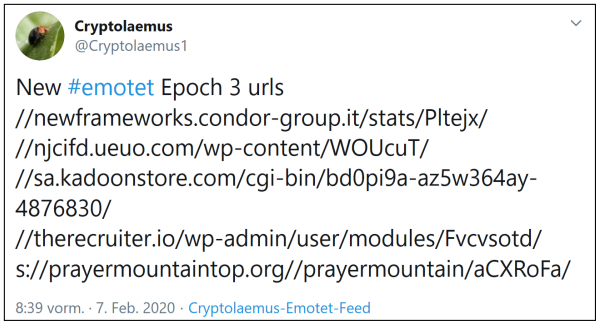

Figure 4.3: Tweet with obfuscation of URLs by @Cryptolaemus1

Nevertheless, many obfuscated IoC follow a common pattern that can be reversed without fuss. To do so, a generic regular expression is required to catch most of the IoC without trapping into many false positives. Each expression needs to be applied on the Tweets taken from a training set. Afterwards, they have to be validated manually to identify invalid IoC and adapt the search parameters.

As a result, the following regular expressions were developed to catch obfuscated IoC generically.

Listing 4.1: Regex for MD5, SHA-1 and SHA-256

```
/[a-f\d]{32,64}/i
```

Listing 4.2: Regex for URI

```
/[a-z]*:?//\S*/i
```

Listing 4.3: Regex for IPv4

```
/\d+\.\d+\.\d+\.\d+/
```

Listing 4.4: Regex for CVE

```
/cve.\d+.\d+/i
```

Listing 4.5: Regex for filename

```
/[^:/\*\?<>\|\s\.\\\]+\.(?:exe|dll|docx?|xlsx?|vbs|pdf|jar|bin|elf)/i
```

Listing 4.6: Regex for filepath

```
/[a-z]:\\\S+/i
```

Listing 4.7: Regex for email

```
/[a-z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])
    ?(?:\.[a-z0-9](?:[a-z0-9-]{0,61}[a-z0-9])?)*/i
```

In order to be able to correlate the matched sub-strings with each other, a function has been created that normalizes each type. Simultaneously, the IoC are removed from the Tweets to clean the content for the machine learning classification. Tweets without any file hash are promptly discarded because they are assumed to be irrelevant for malware analysts.

4.4.5 *Pre-processing*

In order to apply the classification tasks, the textual data has to be prepared. In the literature are many approaches on how to pre-process short messages.

The following steps are applied on an example Tweet published by G DATA malware analyst Karsten Hahn to illustrate the process.

Keeping in mind, that the URIs have already been extracted from the Tweet as described in the previous subsection.

Listing 4.8: Pre-Processing a Tweet published by @struppigel

```
What malware family is this? It's a RAT, written in VB6.
Has some spanish banking related strings in it. @verovaleros
```

At the beginning, the input text is converted into lowercase. Since the case sensitivity does not affect the semantics of a word, the algorithm should not differentiate between these characters.

```
what malware family is this? it's a rat, written in vb6.
has some spanish banking related strings in it. @verovaleros
```

Subsequently, all mentioned users starting with @ are removed from the Tweet as it draws no benefit to the meaning but distorts the sentence structure.

```
what malware family is this? it's a rat, written in vb6.
has some spanish banking related strings in it.
```

Furthermore, non-alphanumeric characters are replaced with a white space to receive words in natural language only to parse them properly.

```
what malware family is this  it s a rat  written in vb6
has some spanish banking related strings in it
```

Afterwards, the sentences are splitted by white-space sequences into a single vector consisting of unique words.

```
["what","malware","family","is","this","it","s","a","rat","written","in"
    ,"vb6","has","some","spanish","banking","related","strings"]
```

Finally, numbers and one-character words are removed from the vector, as well as stop words derived from the natural language toolkit [23].

```
["malware","family","rat","written","vb6","spanish","banking","related",
    "strings"]
```

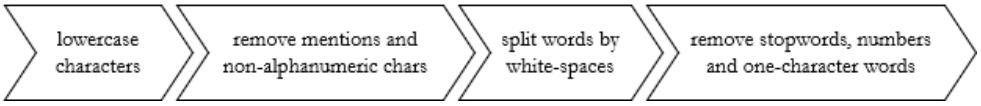

Figure 4.4: Pre-processing Tweets for feature extraction

Based on the originated vector, it is possible to determine whether Tweets are similar using a classification model.

4.5 TWEET CLASSIFICATION

By means of the pre-processed Tweet text, a metric has to be discovered that allows the system to compare it with Tweets of the training set.

4.5.1 *Information Retrieval*

Three commonly used approaches in natural language processing determine the similarity of documents (Tweets) within a corpus.

4.5.1.1 *TF-IDF*

The first one is called TF-IDF where the words of a sample are converted into a numeric vector regarding their token occurrence probability inside the training set. It consists of the product of both term frequency (tf) and inverse document frequency (idf) where the result describes the relevance of a term (t_i) inside a document corpus (D) [4].

$$tf - idf(t_i, D_j) = tf(t_i, D_j) * idf(t_i)$$

The term frequency calculates the occurrence prevalence of t_i in a specific document (D_j) divided by the occurrence prevalence of the most frequently used term (t^{max}) in D_j.

$$tf(t_i, D_j) = \frac{\#(t_i, D_j)}{\#(t^{max}, D_j)}$$

While the term frequency depends on one document, the inverse document frequency considers t_i in context of the entire document corpus. Thus, it punishes frequently used words in all documents and promotes those words who rarely appear. Therefore, it is computed by calculating the logarithm of the number of documents in D divided by the sum of documents that include t_i.

$$idf(t_i) = \log \frac{|D|}{\sum_{t_i \in D}}$$

As soon as a new document is discovered, the TF-IDF is applied to each word inside the document. The resulting numeric vector is used to compare it with the Tweets in the training set later on.

4.5.1.2 *Named Entity Recognition*

Another approach in natural language processing is named NER and refers to an annotation technique which categorized words into defined classes [4]. Therefore, a topic-related dictionary is required that references a list

of words to their entities. During the process, each word in a document is assigned to the appropriate class in order to recognize the topic. Words that are not included in the dictionary are considered as insignificant.

Based on the amount and type of entities, it is decided whether the document is relevant or not. For example, entities in cyber security could be a vulnerability (VULN), companies (ORG), file types (FILE) or digital signatures (SIGN).

Listing 4.9: NER annotated Tweet published by @cyb3rops

```
Signed CVE-2020-0601 binary tagged on Virustotal
(SIGN)    (VULN)     (FILE)   (U)  (U)  (ORG)
```

As this method requires a comprehensive and continuously updated word dictionary about the targeted domain, it takes an extraordinarily high effort in both implementation and maintenance.

4.5.1.3 *Word Embedding*

Another approach is called word embedding which converts each token into a vector that is placed inside a multidimensional space [4]. Words that are frequently used within the same context are also grouped closely together in the vector space. Therefore, a great amount of sentences is necessary to increase the accuracy and prevent distortions caused by unusual phrases.

Based on the training samples, it is possible to build a word embedding model with the Word2vec algorithm proposed by Google researchers [20]. The implementation has been conducted using the Python module word2vec[9].

First, all pre-processed sentences are split by words to retrieve a Bag-of-words. Subsequently, all tokens that have more than 250 occurrences are mapped into the multidimensional vector space.

In order to visualize the distribution of words, the model has been narrowed down to a two-dimensional space. As can be seen in the image below, some words are closely together while others are standalone. On the one hand, words like "virustotal", "sample" and "analysed" are frequently used in the same sentences. On the other hand, commonly used Spanish words like "la", "el", "en" or "de" are also close together within a cluster. They are not removed during the data curation because the stopword list currently contains English words only.

Since many Tweets in the training set are published by automated systems (bots), they always follow the same structure. This in turn has a direct impact on the vector space and biases relevance of Tweets created by human experts.

9 https://github.com/danielfrg/word2vec

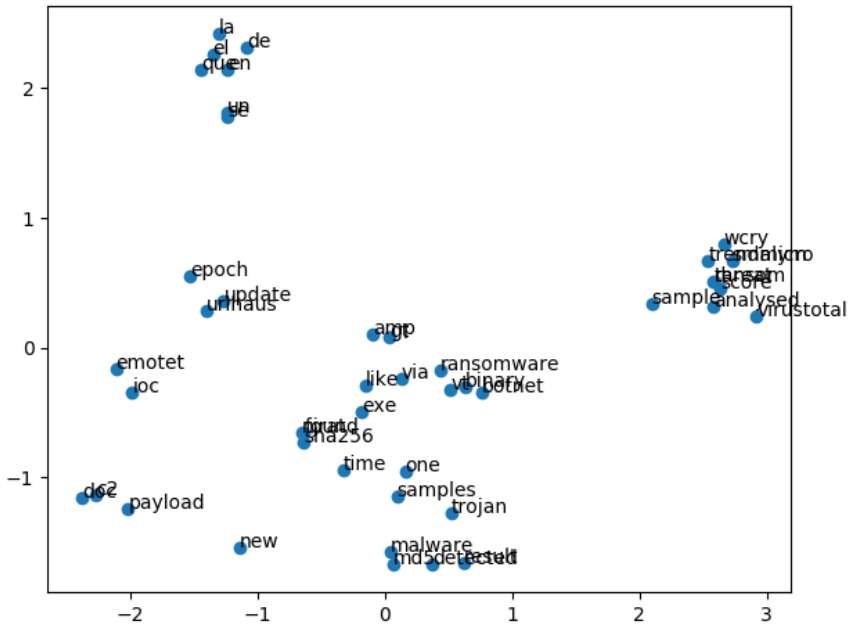

Figure 4.5: Word2Vec example for word embedding

4.5.2 *Classification Models*

In order to find a suitable classification model, the TF-IDF numeric vectors have been selected as training material. The reasons for choosing this information retrieval method are described in more detail in Chapter 4.5.3.

This work covers five different machine learning approaches that have achieved good results in the past. They have been evaluated using cross-fold validation based on the labeled Tweet set. 80 percent of the Tweets are randomly selected as training data to build the model and 20 percent are dedicated as test data to verify the results. The selected parameters and their achieved accuracy are explained in the following sections.

4.5.2.1 *Nearest Neighbors*

A nearest neighbor classifier searches for data points in the training set that are close to an unknown sample. One common used algorithm is called k-nearest neighbors which calculates the Euclidean distance to the closest labeled data points.

Therefore, a constant number of neighbors has to be selected that defines the amount of nearest samples. Based on a majority vote of the neighbor classes, the algorithm assigns the new data point to the mostly found class. The higher the number of neighbors the better the accuracy of the model considering a well-composed training set. At the same time, the computation effort grows and speed reduces with increasing neighbors.

Since performance is one of the major non-functional requirements, the k value was set to three in order to stay fast. The classifier has been build and

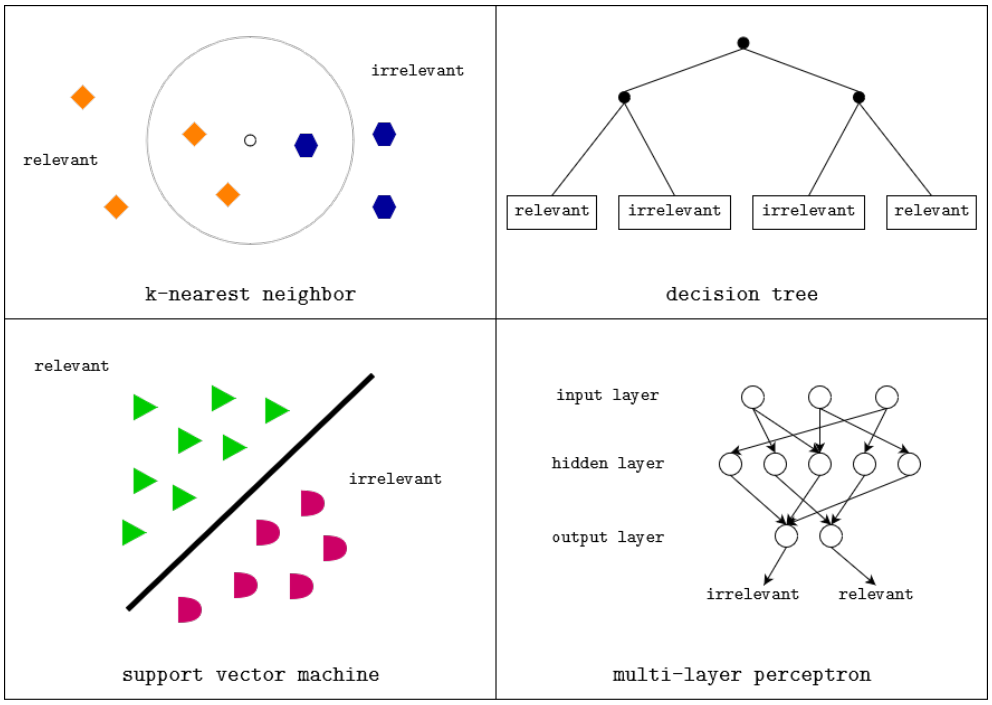

Figure 4.6: Different classification models

applied on the training set, using cross-fold validation within less than 2.6 seconds in total. It achieved an F1-Score of 0.97 for both classifying relevant and irrelevant Tweets.

4.5.2.2 *Decision Tree*

Decision trees consist of various decision nodes that are build up on attributes derived from the training set. At each node the tree decides whether the value of a sample belongs to one of two areas until it cascades down to a terminal node.

The tree depth has direct impact on the precision of the model but also decreases the speed similar to the nearest neighbor classifier.

On the one hand, decision trees are biased, if one class outweighs the others in a training set. Therefore, a well-balanced training set with almost equal amount of relevant and irrelevant Tweets is required. On the other hand, they also tend to overfitting which means that the tree is prone to classify new samples wrong because the branches are too fine granulated.

In order to tackle this issue, different selections of training samples are randomly passed through multiple decision trees. This model is called random forest in which the majority vote of all trees forms the label of a sample.

Keeping the performance in mind, the depth of a decision tree has been set to five layers. In less than half a second the classifier achieved a precision of 1.0 for relevant Tweets while the recall only amounts to 0.75. It clearly shows that the tree is suitable for classifying relevant Tweets with low false positive rate. However, irrelevant Tweets which are unlike each other cannot be predicted with a high accuracy.

Consequently, this overfitting property has been attempted to be resolved with a random forest consisting of 100 decision trees with a depth of five. In order to retrieve a deterministic cross-fold validation result, the random state has been set to zero. The evaluation shows the random forest significantly improving the F1-Score of both relevant and irrelevant Tweets with additional 0.82 seconds to train the model.

4.5.2.3 *Support Vector Machines*

Linear SVM are binary classifiers that define a linear discriminant function between two classes in a high-dimensional space [22]. Therefore, a hyperplane is searched that separates the data points (support vectors) of both classes with the maximum margin [22]. Subsequently, the training data has to be linearly separable which is not always the case.

Since the data set dynamically grows over time, non-linear separable states are likely to occur as well. If this is the case, the SVM classifier has to map the training data into a higher dimensional space to find a linear discriminant [22].

Furthermore, SVM can be optimized using a regularization value between 0.0 and 1.0 in which a high value increases the recall by minimizing the margin between the support vectors. The opposite applies to a low regularization value that tries to find a larger-margin hyperplane by accepting more false positives. As explained in the requirement analysis, the occurrence of false classification should be avoided to maximize the alerts of relevant Tweets and minimize noisy irrelevant Tweets. Accordingly, the regularization value was set to 1.0.

With an F1-Score of 0.99 the SVM achieves the best classification results in both classes relevant and irrelevant. The model has been trained with a peak time of 0.057 seconds.

4.5.2.4 *Multi-layer Perceptron*

MLP are neuronal networks that are capable to determine the relations between a given input and the expected output [11]. Therefore, they use a certain number of neurons in the hidden layer which adjust weights of a function until the accuracy is acceptable [11]. This process requires several iterations that slow down the training speed significantly compared to other models.

In this work, an MLP has been applied using the SGD algorithm that is known to perform well with large training sets. The rectified linear unit function has been chosen to activate the neurons with a regularization alpha value of 0.0001. Each sample is used five times until a sufficient optimization tolerance is achieved.

Shortly beyond the linear SVM the F1-Score amounts to 0.99 for both relevant and irrelevant Tweets. Nevertheless, the model training time requires more than 302 seconds which is significantly higher than the described other models.

CLASSIFIER MODEL	PRECISION	RECALL	F1-SCORE
k-nearest Neighbors	0.97	0.97	0.97
Decision Tree	1.00	0.75	0.85
Random Forest	1.00	0.87	0.93
Linear SVM	1.00	0.97	0.99
Multi-layer Perceptron	0.99	0.98	0.99

Table 4.2: Results for classification of relevant Tweets

CLASSIFIER MODEL	PRECISION	RECALL	F1-SCORE
k-nearest Neighbors	0.97	0.97	0.97
Decision Tree	0.79	1.00	0.88
Random Forest	0.88	1.00	0.94
Linear SVM	0.97	1.00	0.99
Multi-layer Perceptron	0.98	0.99	0.99

Table 4.3: Results for classification of irrelevant Tweets

4.5.3 *Evaluation*

After comparing the methods of information retrieval and classification models, a finial decision of the most suitable algorithms has to be made. Keeping in mind that a low false positive rate and a high throughput of Tweets is necessary, both criteria have to be included into the decision.

All three information retrieval approaches have already been applied successfully in different research projects regarding Tweet classification in the past.

As NER requires huge effort to define entities and related keywords, it does not fit into the project's time frame. Furthermore, this thesis focuses on binary classification only which is not the primary goal of the NER model. However, NER offers potential for future work because it allows to assign documents to different categories such as software vulnerabilities or malware samples.

Since word embedding considers the distribution and order of words inside a document, it particularly focuses on the sentence structure. As already mentioned, many bots frequently post threat intelligence on Twitter with exactly the same syntax. In order to prevent an overweight of syntactical features derived from these Tweets, this information retrieval method is not used within the system.

TF-IDF estimates the relevance of a word by relating it to the whole document corpus but disregarding the syntactical structure. Contrary to word embedding, this method decreases the bias of similar Tweets by bots inside the training set because it punishes frequently used words. Furthermore, the training samples do not have to be manually labeled by human analysts as it is required by NER.

Overall, TF-IDF turns out to be the most suitable algorithm compared to the other two. Consequently, the numeric feature vectors of the TF-IDF are fed into the classification model to be selected.

The five classification models are verified regarding their precision, recall and performance. The evaluation of the decision tree model shows that it is prone to overfitting because precision and recall are unbalanced. Although the decision tree is the second fastest built classifier of the tested models, it is not suitable due to the classification imbalance.

By using the random forest the F1-Score can be slightly improved with additional performance overhead. Nevertheless, the drawback of overfitting is not sufficiently narrowed and the training speed is too slow.

The k-nearest neighbor classifier constantly achieves good results in both precision and recall. Similar to the random forest the speed is not fast enough to immediately attach a new worker to the stream processing.

Multi-layer perceptron, on the other hand, reaches the best F1-Score of 0.99 along the linear SVM which qualifies both for the Tweet classification, considering accuracy only. However, the multi-layer perceptron requires 300 seconds more to build the model than defined in the non-functional requirements. The extreme performance loss makes the multi-layer perceptron unsuitable for the binary classification task.

Finally, the linear SVM with the best values in both F1-Score and speed is selected as classification model within the OSINT system. Building the classifier in about 0.057 seconds more than fulfills the non-functional requirements to build the model in less than a second. Furthermore, the precision for relevant Tweets during cross-fold validation is 1.0 which shows that false classification of Tweets about malware is very unlikely.

Detailed extracts of the program code are written in the Appendix A to illustrate the working result.

4.6 PERSISTING AND ALERTING

After the Tweets are processed and classified by the workers, the relevant Tweets have to be persistently stored and passed to the analysts after exceeding the virality threshold.

4.6.1 *Persisting*

Tweets are always composed of the same entities, however, they can consist of multiple values for each entity. For example, a Tweet may contain multiple hashtags, user mentions or extracted IoC.

On the basis of available hardware resources, the decision fell on a relation model where multi-value columns are represented as a JSON array. Overall, the database holds the following attributes about an analyzed Tweet.

- `Tweet ID [BIGINT]`: Tweet identifier number assigned by Twitter
- `User ID [BIGINT]`: User identifier number assigned by Twitter
- `Created at [DATETIME]`: timestamp of Tweet creation
- `Retweets [INT]`: number of Re-Tweets
- `Alerted [TINYINT]`: flag that indicates if alert is triggered
- `Tokens [JSON]`: string tokens derived after pre-processing
- `Text [TEXT]`: originally Tweet text
- `Language [VARCHAR]`: language of Tweet text
- `URIs [JSON]`: extracted URIs
- `CVEs [JSON]`: extracted CVE numbers
- `Email [JSON]`: extracted email addresses
- `Filename [JSON]`: extracted filenames
- `File hashes [JSON]`: extracted file hashes
- `File paths [JSON]`: extracted file paths
- `IPv4 [JSON]`: extracted IPv4 addresses

In the future, the design and performance of the database can be improved by creating additional referenced tables. Nevertheless, the basic approach is currently sufficient to query and insert rows in reasonable time below one second.

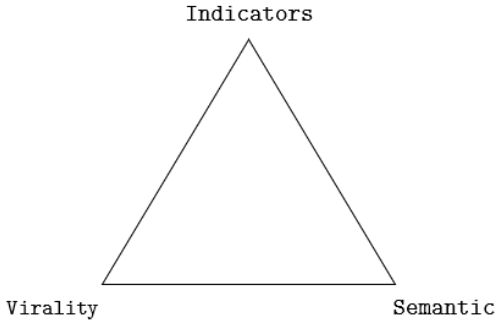

Figure 4.7: Triangle of Tweet relevance

4.6.2 *Alerting*

The relevance of a Tweet is assessed by the three criteria pictured in 4.7. As already described in Chapter 4.4.4, a relevant Tweet includes at least one indicator represented as file hash. Moreover, the semantics are evaluated based on the TF-IDF, using the linear SVM classifier to decide whether the Tweet is about cyber security. Eventually, all Tweets have a unique virality value that defines how often a Tweet has been shared before. If the virality exceeds a certain threshold and the other two are fulfilled, the system triggers an alert message to inform malware analysts.

Depending on how many alerts per minute are required by the stakeholders, this value can be adjusted over the time. During the evaluation phase, a Tweet requires three additional Re-Tweets to issue a warning. The alert is dispatched via a Mattermost API web-hook to a dedicated channel that is open to malware analysts (see 4.8). In the rare event an irrelevant Tweet still passing the relevance filter, the analysts can reject it and the Tweet is added to the irrelevant training set.

From January 2020 to February 2020 the number of incoming relevant Tweets ranged between 60 to 316 per day from which one to 29 Tweets triggered an alert.

At this point, the functionality scope of the system ends since the human experts take over the analysis of the provided threat intelligence.

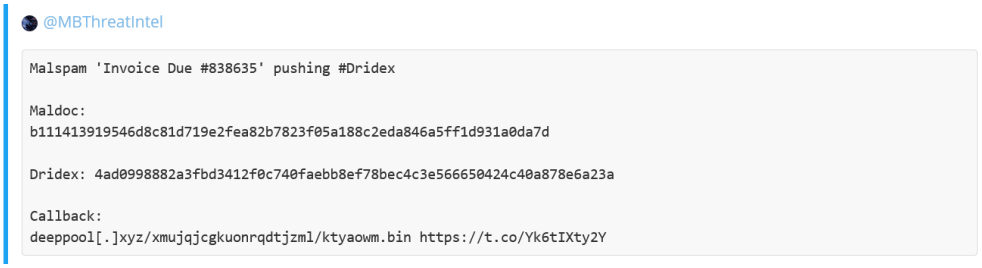

Figure 4.8: Mattermost Tweet alert issued by @MBThreatIntel

5 CONCLUSION

Open-Source Threat Intelligence is an important component of the defense against the growing number of malware. The system developed in this thesis automatically collects this information from Tweets and their linked content such as malware sharing platforms or IT security blogs. It is able to distinguish between cyber security related and non-related topics with a high probability. The combination of IoC extraction, semantic classification and virality fulfills the system functional and non-functional requirements sufficiently.

On the one hand, the volume of threat intelligence is kept low by storing only relevant Tweets and discarding duplicates such as Re-Tweets. Yet, the system counts how often a Re-Tweet appears in the stream and can therefore determine a virality value.

On the other hand, the velocity is handled using a message queue which is processed by a flexible number of workers. The higher the amount of incoming Tweets, the more workers are pulled up and visa verse.

Feature extraction is performed with use of regular expressions to catch IoC from Tweets and their linked content, even if they are obfuscated. To avoid contact with malicious URLs the system checks whether it is clean according to G DATA beforehand. Only if a file is discovered in the Tweet or at an external web page, the Tweet is further processed by the system.

Subsequently, three different information retrieval methods have been discussed in order to find a suitable metric for natural language processing. As a result, TF-IDF has been chosen since it allows to compare the similarity of short messages to each other.

Furthermore, five classification models have been trained with more than 14,000 relevant Tweets about malware and evaluated regarding accuracy and performance. Both the MLP and linear SVM classifiers achieved the same F1-Score of 0.99, however, the linear SVM has the fastest performance overall. Therefore, the decision for information retrieval and classification fell on the TF-IDF together with the linear SVM model.

In the final stage, a Tweet that exceeds a virality threshold of more than three Re-Tweets, triggers the system to send a notification to the malware analysts for further inspection.

In the future many more sources of threat intelligence can be added to the system, such as monitoring malware related forums in the dark web.

In addition, the incoming amount of Tweets about malware can be increased by upgrading the Twitter Stream API to track more than 400 keywords and pull up additional stream listeners. The selection of relevant keywords, on the other hand, can be optimized by using word embedding techniques based on Tweets in the training set.

Furthermore, the training set contains the same number of relevant and irrelevant Tweets, yet, the amount of irrelevant Tweets in the stream is much higher. Therefore, it can be extended with Tweets that look similar to the relevant Tweets but are not actually about cyber security.

Eventually, the thesis clearly shows that Open-Source Threat Intelligence can be gathered automatically and malware analysts receive information about malicious samples more conveniently and faster. By providing the data fast and efficiently the system is an economical and technical advantage for the company.

Part II

APPENDIX

A

PYTHON CODE FOR THE CLASSIFIER EVALUATION

The following program code written in Python 3.5.3 has been used to extract IoC and evaluate the classifier models. The complete code cannot be disclosed due to corporate secrecy.

Pre-compile regular expressions for IoC extraction and fetch stop word list.

```
import re
import sys
import json
import time
import nltk  # version: 3.4.5

# regex for user mentions and non alphanumeric characters
regex_user = re.compile('@[a-z\d_]+')
regex_anum = re.compile('[^a-z0-9]')

# regex for IoC
regex_hash = re.compile('[a-f\d]{32,64}', flags=re.I)
regex_uris = re.compile('[a-z]*:?//\S*', flags=re.I)
regex_ipv4 = re.compile('\d+\.\d+\.\d+\.\d+')
regex_cves = re.compile('cve.\d+.\d+', flags=re.I)
regex_file = re.compile('[^:/\*\?<>\|\s\.\\\]+\.''(?:exe|dll|docx?
 ↪|xlsx?|vbs|pdf|jar|bin|elf)', flags=re.I)
regex_mail = re.compile("[a-z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-z0-9](?:
 ↪[a-z0-9-]{0,61}[a-z0-9])?""(?:\.[a-z0-9](?:
 ↪[a-z0-9-]PYZob0,61}[a-z0-9])?)*",flags=re.I)

# download stop word list
nltk.download('stopwords', print_error_to=sys.stdout)
stopwords = nltk.corpus.stopwords.words('english')

.....................................................................

[nltk_data] Downloading package stopwords to nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Extract IoC from Tweets and tokenize the remaining text. Store the results inside JSON file.

```
# extract IoC from Tweets
def get_iocs(string):
    iocs = {
        "hashes": re.findall(regex_hash, string),
        "ipv4s": re.findall(regex_ipv4, string),
        "mails": re.findall(regex_mail, string),
        "uris": re.findall(regex_uris, string),
        "cves": re.findall(regex_cves, string),
        "files": []}
    for key, values in iocs.items():
        for ioc in values:
            string = string.replace(ioc, "")
    iocs["files"] = re.findall(regex_file, string)
    return string, iocs

# remove user mentions and non alphanumeric characters
# split words, remove single or numeric chars and stop words
def get_tokens(string):
    text = string.lower()
    text = re.sub(regex_user, " ", text)
    text = re.sub(regex_anum, " ", text)
    words = []
    for word in re.split(r'\s+', text):
        if len(word) > 1 and not word.isnumeric() and
        word not in stopwords:
            words.append(word)
    return words

def extract_features(string):
    string, iocs = get_iocs(string)
    words = get_tokens(string)
    return words, iocs

# extract all IoC from samples in training set
corpus, labels = [], []
with open('tweets_large.json', 'r') as file:
    t_start = time.time()
    for tweet in json.load(file):
        tokens, indicators = extract_features(tweet["text"])
        if len(tokens) > 0:
            corpus.append(' '.join(tokens))
            labels.append(tweet["relevant"])
    t_end = time.time()
```

```
print("Processed %s Tweets in %s sec" % (len(labels),␣
 ↪t_end-t_start))
with open('corpus.json', 'w') as file:
    json.dump(corpus, file, indent=4)
```

```
.....................................................................

Processed 27712 Tweets in 3.586724281311035 sec
```

Train five different classifiers, serialize the model and evaluate accuracy. More information about the used TF-IDF vectorizer can be found at scikit-learn.org.

```
# scikit-learn version: 0.22.1
from sklearn.svm import LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# initialize TF-IDF vector and learn vocabulary from corpus
tf_idf = TfidfVectorizer()
X = tf_idf.fit_transform(corpus)

# split training set into 20 % test data and 80 % training data
# random state is set to static number to get reproducible results
X_train, X_test, y_train, y_test = train_test_split(X, labels,␣
 ↪test_size=0.2, random_state=42)

# list of tested classification models
names = ["Nearest Neighbors", "Decision Tree", "Random Forest",␣
 ↪"Linear SVM", "Multi-layer Perceptron"]
classifiers = [
    KNeighborsClassifier(3),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=100,␣
 ↪random_state=0),
    LinearSVC(C=1),
    MLPClassifier(alpha=0.0001, random_state=0)]
```

```
# cross-validation for each classifier
for name, clf in zip(names, classifiers):
    t_start = time.time()
    clf.fit(X_train, y_train)
    pred = clf.predict(X_test)
    t_end = time.time()
    print(name, "- Confusion Matrix:\n")
    print(classification_report(y_test, pred,␣
 ↪target_names=["irrelevant", "relevant"]))
    print(name,"- Accuracy: %s" % accuracy_score(y_test, pred))
    print("Training Speed:%s sec" % (t_end - t_start))
    print("\n\n")
```

```
.....................................................................

Nearest Neighbors - Confusion Matrix:

              precision    recall  f1-score   support

  irrelevant       0.97      0.97      0.97      2700
    relevant       0.97      0.97      0.97      2843

    accuracy                           0.97      5543
   macro avg       0.97      0.97      0.97      5543
weighted avg       0.97      0.97      0.97      5543

Nearest Neighbors - Accuracy: 0.9687894641890673
Training Speed: 2.5553009510040283 sec



Decision Tree - Confusion Matrix:

              precision    recall  f1-score   support

  irrelevant       0.79      1.00      0.88      2700
    relevant       1.00      0.75      0.85      2843

    accuracy                           0.87      5543
   macro avg       0.89      0.87      0.87      5543
weighted avg       0.90      0.87      0.87      5543

Decision Tree - Accuracy: 0.8699260328342053
Training Speed: 0.4612576961517334 sec
```

```
Random Forest - Confusion Matrix:

              precision    recall  f1-score   support

  irrelevant       0.88      1.00      0.94      2700
    relevant       1.00      0.87      0.93      2843

    accuracy                           0.93      5543
   macro avg       0.94      0.94      0.93      5543
weighted avg       0.94      0.93      0.93      5543

Random Forest - Accuracy: 0.9348728125563774
Training Speed: 1.27939772605896 sec



Linear SVM - Confusion Matrix:

              precision    recall  f1-score   support

  irrelevant       0.97      1.00      0.99      2700
    relevant       1.00      0.97      0.99      2843

    accuracy                           0.99      5543
   macro avg       0.99      0.99      0.99      5543
weighted avg       0.99      0.99      0.99      5543

Linear SVM - Accuracy: 0.9852065668410608
Training Speed: 0.056516408920288086 sec



Multi-layer Perceptron - Confusion Matrix:

              precision    recall  f1-score   support

  irrelevant       0.98      0.99      0.99      2700
    relevant       0.99      0.98      0.99      2843

    accuracy                           0.99      5543
   macro avg       0.99      0.99      0.99      5543
weighted avg       0.99      0.99      0.99      5543

Multi-layer Perceptron - Accuracy: 0.9866498286126646
Training Speed: 302.31182861328125 sec
```

BIBLIOGRAPHY

[1] Harsh Agrawal. *What is a Bitcoin Hash?* https://coinsutra.com/bitcoin-hash/. Accessed: 23 February 2020.

[2] Fernando Alves et al. "Processing Tweets for Cybersecurity Threat Awareness." In: *CoRR* (2019). arXiv: 1904.02072.

[3] Vahid Behzadan et al. "Corpus and Deep Learning Classifier for Collection of Cyber Threat Indicators in Twitter Stream." In: *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, Dec. 2018. DOI: 10.1109/bigdata.2018.8622506.

[4] Taweh Beysolow II. *Applied Natural Language Processing with Python*. Ed. by Celestin Suresh John et al. pp. 10-11, 50-51, 57-62, 128-133. Apress, 2018. ISBN: 978-1-4842-3732-8. DOI: 10.1007/978-1-4842-3733-5.

[5] Central Intelligence Agency. *INTellingence: Open Source Intelligence*. https://www.cia.gov/news-information/featured-story-archive/2010-featured-story-archive/open-source-intelligence.html. Accessed: 23 February 2020.

[6] Nancy Chinchor. "MUC-4 evaluation metrics." In: *Proceedings of the 4th conference on Message understanding - MUC4 '92*. Association for Computational Linguistics, 1992. DOI: 10.3115/1072064.1072067.

[7] Cyber Threat Intelligence Technical Committee. *Introduction to STIX*. https://oasis-open.github.io/cti-documentation/stix/intro. Accessed: 23 February 2020.

[8] Nuno Dionísio et al. "Cyberthreat Detection from Twitter using Deep Neural Networks." In: *CoRR* (2019). arXiv: 1904.01127.

[9] Amosse Edouard. "Event detection and analysis on short text messages." PhD thesis. Université Côte d'Azur, Oct. 2017. URL: https://tel.archives-ouvertes.fr/tel-01679673.

[10] Federal Office for Information Security. *Report on the State of IT Security in Germany 2019 published*. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Lageberichte/Lagebericht2019.pdf. Accessed: 23 February 2020.

[11] Palash Goyal et al. *Deep Learning for Natural Language Processing*. Ed. by Celestin Suresh John et al. pp. 46-53. Apress, 2018. ISBN: 978-1-4842-3684-0. DOI: 10.1007/978-1-4842-3685-7.

[12] Alvin Junus et al. "CommunityAware Prediction of Virality Timing Using Big Data of Social Cascades." In: *2015 IEEE First International Conference on Big Data Computing Service and Applications*. IEEE, Mar. 2015.

[13] Kaspersky Lab. *Threat Actors Master 'False Flags' Tactics to Deceive Victims and Security Teams*. https://www.kaspersky.com/about/press-releases/2016_false-flags. Accessed: 23 February 2020.

[14] Yoon Kim. "Convolutional Neural Networks for Sentence Classification." In: *CoRR* (2014). arXiv: 1408.5882v2.

[15] Ba Dung Le et al. "Gathering Cyber Threat Intelligence from Twitter Using Novelty Classification." In: *CoRR* (2019). arXiv: 1907.01755.

[16] Xiaojing Liao et al. "Acing the IOC Game." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. ACM Press, 2016. DOI: 10.1145/2976749.2978315.

[17] Peter Mackenzie. *WannaCry Aftershock*. https://www.sophos.com/en-us/medialibrary/PDFs/technical-papers/WannaCry-Aftershock.pdf. Accessed: 23 February 2020.

[18] Malwarebytes. *Let's talk Emotet malware*. https://www.malwarebytes.com/emotet/. Accessed: 23 February 2020.

[19] Mandiant. *The OpenIOC Framework*. https://web.archive.org/web/20170610203815/http://openioc.org/. Accessed: 23 February 2020.

[20] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space." In: *ICLR* (2013). arXiv: 1301.3781v3.

[21] Sudip 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*. IEEE, Aug. 2016. DOI: 10.1109/asonam.2016.7752338.

[22] M. N. Murty and Rashmi Raghava. *Support Vector Machines and Perceptrons*. pp. 43-51. Springer International Publishing, 2016. ISBN: 978-3-319-41062-3. DOI: 10.1007/978-3-319-41063-0.

[23] NLTK Project. *Natural Language Toolkit*. https://www.nltk.org/. Accessed: 23 February 2020.

[24] Python Software Foundation. *Global interpreter lock*. https://scikit-learn.org/stable/modules/neighbors.html#classification. Accessed: 23 February 2020.

[25] Guido van Rossum. *Guido van Rossum - Publications*. https://gvanrossum.github.io/Publications.html. Accessed: 23 February 2020.

[26] Anna Sapienza et al. "DISCOVER: Mining Online Chatter for Emerging Cyber Threats." In: *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*. ACM Press, 2018. DOI: 10.1145/3184558.3191528.

[27] Quentin Le Sceller et al. "SONAR: Automatic Detection of Cyber Security Events over the Twitter Stream." In: *Proceedings of the 12th International Conference on Availability, Reliability and Security - ARES '17*. ACM Press, 2017. DOI: 10.1145/3098954.3098992.

[28] Marc Stevens et al. "The First Collision for Full SHA-1." In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 570–596. ISBN: 978-3-319-63688-7. DOI: 10.1007/978-3-319-63688-7_19.

[29] The MITRE Corporation. *Common Vulnerabilities and Exposures*. https://cve.mitre.org/. Accessed: 23 February 2020.

[30] Twitter, Inc. *Twitter Developer Labs*. https://developer.twitter.com/en/labs. Accessed: 23 February 2020.