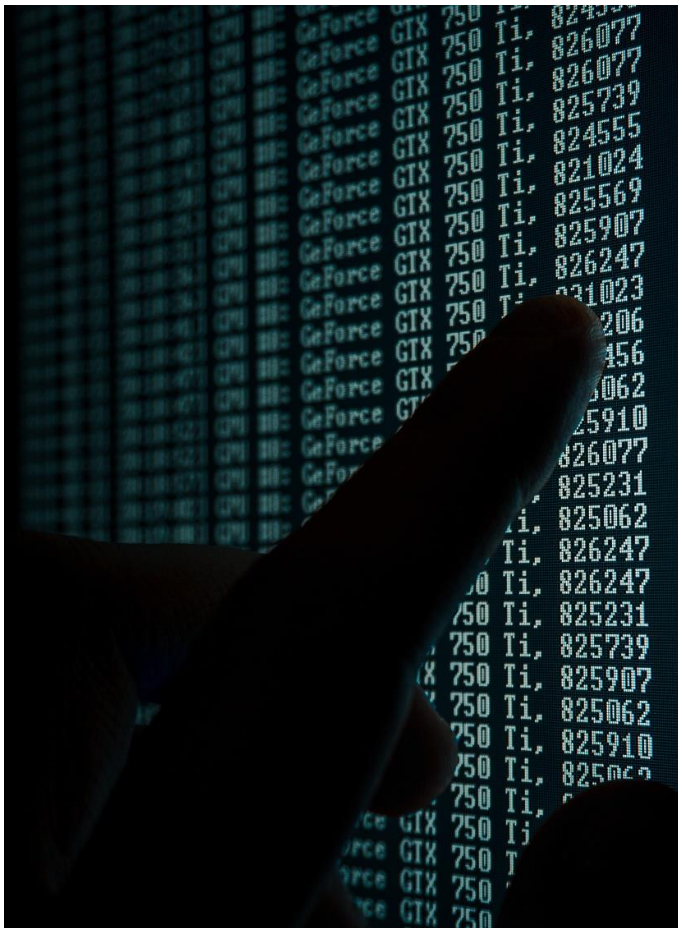
Early Vulnerability Detection Using Signals from Web

Master Thesis



Filip Adamik

Innovative Communication Technologies and Entrepreneurship
Aalborg University Copenhagen, Denmark
February – July 2020

Aalborg University Copenhagen
A.C. Meyers Vænge 15
2450 København SV

Semester Coordinator: Henning Olesen

Secretary: Maiken Keller

Semester: Spring 2020

Title: Early Vulnerability Detection Using Signals from Web

Project Period: February – July 2020

Semester Theme:
Master Thesis

Supervisor(s):
Emmanouil Vasilomanolakis

Project group no.: n/a

Members:

Filip Adamik

Pages: 68
Finished: July 2020

Abstract:

In this project we explore a hypothesis that previously unseen security vulnerabilities in software can be predicted from online sources before they are widely known. We analyse a vulnerability lifecycle and design a high-level architecture of a vulnerability detection system. We design and implement a neural learning model consisting of Convolutional and LSTM layers that can filter out useful information from the provided online sources. We collect data from three different online sources and obtain three different datasets compiled by other researchers. We use one of these datasets to construct training data and train the neural model. We investigate why the model fails to perform and gradually adjust the model and the data creation pipeline until promising preliminary results are reached. Further research is necessary to confirm these results and prove or disprove the underlying hypothesis.

Contents

List of acronyms and abbreviations

API	Application Programming Interface
ARIMA	Autoregressive Integrated Moving Average
ARIMAX	Autoregressive Integrated Moving Average with Exogenous Variables
ARMA	Autoregressive Moving Average
BERT	Bidirectional Encoder Representations from Transformers
CCC	Cambridge Cybercrime Center
CNN	Convolutional Neural Network
CPE	Common Platform Enumeration
CSV	Comma-separated Value
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DDoS	Distributed Denial-of-Service
DoS	Denial-of-Service
EMA	Exponential Moving Average
FN	False Negative
FP	False Positive
GPU	Graphics Processing Units
GRU	Gated Recurrent Unit
IRC	Internet Relay Chat
LDA	Latent Dirichlet Allocation
LSTM	Long Short-term Memory
MA	Moving Average
ML	Machine Learning
NER	Named Entity Recognition
NIST	National Institute of Standards and Technology
NLP	Natural Language Processing
NN	Neural Network
NVD	National Vulnerability Database
OSINT	Open-source Intelligence
POS	Part of Speech
RAID	Redundant Array of Independent Disks
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristics
SVM	Support Vector Machine
TF/IDF	Term Frequency/Inverse Document Frequency
TN	True Negative
Tor	The Onion Router
TP	True Positive
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

1 Introduction

The motivation for this project comes from the need to better anticipate fresh vulnerabilities. A reliable source of information about most vulnerabilities is the National Vulnerability Database (NVD), but it only lists vulnerabilities that have been discovered and disclosed. This is typically not the case for zero-days traded on the black market, which may go unnoticed for months or even years [1].

However, even before the vulnerability is publicly disclosed and patched, some information about it might appear online. In underground forums, malicious parties trade, exchange information about and collaborate to find and exploit new vulnerabilities. Several high-profile examples exist, where information about planned attacks and new exploits appeared on the dark- and deep-web forums [2–4].

It is in the interest of system administrators, security consultants and other IT professionals to learn about any new vulnerabilities or exploits as soon as possible. However, today it is effectively impossible to manually monitor the wealth of the forums, blogs and markets, both underground and overt, where an indication of a new vulnerability or an exploit might appear. A content discovery and aggregation service could channel these information into a single place, similar to how an RSS aggregator gathers updates from news websites into a single feed, but this does solve the information overload problem. To be useful to its operators, the system would have to filter out the noise and distil the data into usable information. Besides removing all irrelevant and duplicate content, the timeliness of the information is crucial. A short comment from a hacker forum about a previously unseen vulnerability is much more valuable than a well-cited factual article about a vulnerability that has been patched last week.

1.1 Problem Formulation

In this project we explore a hypothesis, that new, previously unseen security vulnerabilities can be detected by monitoring a variety of online sources, even before the knowledge of these vulnerabilities is widely circulated in the security community. Several works with a similar focus suggest that such a hypothesis might be true. These examples include early warning systems, which predict security incidents based on localised machine data or a single online source, as well as works that predict targeted hacker attacks based on online sources. The research hypothesis is therefore as follows:

Monitoring diverse online sources can detect previously unseen security vulnerabilities, before they become widely known.

In order to test this hypothesis we design and implement a machine learning model that can detect new vulnerabilities from input data. We train this model on historical vulnerabilities and past data from online sources. The focus is put specifically on textual data from blogs, discussions and forums. Due to the textual nature of the data and the complexity of detecting vulnerabilities in a narrow time-window, before they are widely known, we employ a neural network architecture and textual embeddings in the machine learning model. The problem formulation of this project can be formalised as:

How to design and implement a neural network model that detects previously unseen security vulnerabilities?

with the following sub-questions:

- *When is a vulnerability considered unseen and what is the time-frame for usefulness of this information from a defence perspective?*
- *What is the most suitable model architecture for this task?*
- *What data to use to train such a model? How to obtain and prepare this data?*

We aim to find answers to these questions during the course of this project.

1.2 Report Structure

This report documents the research steps towards answering the problem formulation. It consists of six chapters and four appendices. After this Introduction chapter we establish preliminaries, explain core concepts of this project and introduce several important data sources in Chapter 2. In Chapter 3 we review existing works and publications with focus similar to ours that predict security incidents or analyse temporal data. Chapter 4 follows with analysis of the problem and exploration of viable solution approaches. Description of the implementation of the technical solution is presented in Chapter 5. Overview of the experiments and the obtained results are presented in Chapter 6 and the project is concluded by discussing potential practical challenges and suggesting the focus of future works in Chapter 7.

Code repository, which was used to obtain and prepare the data, as well as implement and train the machine learning model, is attached to this report and is also available at `https://github.com/zafodB/VulnerabilityDetection`.

2 Background

In this chapter we provide a brief overview of the main areas of this project. In Section 3.1 we discuss the problem of software vulnerabilities and efforts to categorise and classify them. Definition of Open-source Intelligence follows with examples of significant intelligence sources. The fundamentals of Machine Learning and Natural Language Processing are described in their respective sections, followed by presentation of two general time series forecasting methods.

2.1 Security Vulnerabilities

Typically, any computer system has a number of use cases and intended ways to operate. To ensure that it is used within these limits, security policies and mechanisms are often in place. Sometimes, however, these measures are not sufficient to enforce the intended use and a *vulnerability* is present in a system [5]. Because vulnerabilities, if exploited, can result in substantial consequences, such as data and intellectual property loss or disruption to the service, significant effort is invested into design and implementation of robust security policies and mechanisms, as well as into detecting and mitigating existing vulnerabilities. In this section we focus on the latter.

2.1.1 Categorisation

While physical threats, such as susceptibility to flooding, humidity or unauthorised access could also be seen as certain vulnerable factors, we only consider vulnerabilities in software in this work. Broadly, these software vulnerabilities can be split into three categories: (i) vulnerabilities in design and specification, (ii) vulnerabilities in implementation, and (iii) vulnerabilities in operation and management [5]. Vulnerabilities that stem from wrong system operation or mismanagement are known to have caused many major incidents, they must often by addressed by providing good documentation and educating the users about the correct system use, as opposed to updating, fixing or patching the underlying software. This work focuses mainly on vulnerabilities in categories (i) and (ii) which will be discussed further in this report, unless mentioned otherwise.

Efforts to classify and categorise software vulnerabilities are well represented in the popular Common Vulnerability Scoring System (CVSS) framework [6]. This framework is used in vulnerability databases to assess and specify the severity of a given software vulnerability and it introduces fifteen granular categories for vulnerability classification. These are grouped in three groups: (i) base, (ii) temporal, and (iii) environmental metrics.

Base metrics form the core of the classification framework and include metrics such as attack vector (whether the attacker needs to have physical access to the device or not), user interaction requirement (whether a user action is necessary during the attack) or impact metrics (the impact of the vulnerability on confidentiality, integrity and availability of the system). Temporal metrics, such as maturity of the vulnerability and remedy availability, and environmental metrics, such as custom importance factors complement the framework.

The framework and the underlying metrics are useful to both understand the nature and scope of a given vulnerability, but also to determine its current status and likely further developments. CVSS is used in the National Vulnerability Database (described in Section 2.2.1).

2.1.2 Life Stages

The vulnerabilities are not static, but rather they evolve during time as the underlying software, hardware and processes change. While a vulnerability can be understood as a theoretical or potential flaw in code or design, to be of any value to the attacker, this flaw must be used in practice to circumvent the system security measures. When this happens, the vulnerability is *exploited*, often using a purpose-built code called an *exploit*. Not every software vulnerability ends up getting exploited and not all exploit codes are alike in terms of quality and availability.

CVSS captures this evolution in the Exploit Code Maturity metric (see Table 1a.). The exploit code naturally progresses from one stage to the other over time; the further stage it reaches, the the higher risk the vulnerability becomes. Presumably, the authors or vendors behind the vulnerable software correct and adjust their software to remove the vulnerability. Once this happens, the vulnerability can no longer be exploited, the security policies can be enforced effectively again and the weakness is removed. However, in practice this process is often not as straightforward. Depending on the complexity of the vulnerability and the underlying software, the vendor might not immediately release a fixed, patched version of the software. Before that happens, users might react by applying an alternative solution as a workaround or a temporary 'hotfix' is released by the vendor, which only remedies the vulnerability partially or temporarily. These stages are captured by a the Remediation Level Temporal metric in CVSS (see Table 1b.).

(U) Unproven → (P) Proof of Concept → (F) Functional → (H) High
(X) Not defined

(a) Exploit Code Maturity metric. Describes the availability and maturity of malicious code that can exploit the vulnerability. When the vulnerability is assesed, it is assigned one of the five letters (U, P, F, H or X) of the metric scale.

(U) Unavailable	(W) Workaround	(T)Temporary Fix	(O) Official Fix
(X) Not defined			

(b) Remediation Level metric. Describes if and how has the vulnerability been corrected. A single letter from the metric scale (U, W, T, O or X) is assigned when the vulnerability is assessed.

Table 1: Temporal CVSS metrics [6]

Looking at the problem from an attacker's perspective, we can describe several 'life stages' of vulnerabilities [1]. The vulnerability is said to be *born* when it is first introduced into the software by a mistake in design or implementation. It is *living* while it is present in the software. Ideally (for the attacker), a substantial number of users unknowingly use this vulnerable software, which allows for exploitation. Once the vulnerability is discovered, either directly by the vendor or by someone who discloses the discovery to the vendor and/or publicly, the vulnerability *dies*. That is because following the disclosure, the vendor typically issues an update or a fix, removing and 'killing' the vulnerability, which cannot longer be exploited.

Some vulnerabilities however, are never fixed if the product is no longer maintained by the vendor. These vulnerabilities are called *immortal* and remain in the software, allowing it to be exploited. Immortal vulnerabilities can be publicly known, or kept confidential by a malicious party [1].

The final category of vulnerabilities, recognised by [1], are so-called *zombie* or quasi-alive vulnerabilities. These are vulnerabilities that were once alive, but are not currently exploitable due to refactoring of the code base. The zombie vulnerabilities have not been consciously recognised as vulnerabilities by the vendor and may still remain (albeit currently unexploitable) in the most recent version. They may however be reintroduced as living vulnerabilities in next releases and pose a standing threat in the affected systems.

2.1.3 Discovery

What happens with a vulnerability after it has been discovered largely depends on the discovering party. If discovered by the vendor, vulnerability is patched and optionally disclosed afterwards. If a white-hat researcher makes the discovery, the vulnerability is disclosed publicly, typically following a 30-45 day 'protected period', during which the vendor is notified and allowed to patch the vulnerability [7]. If the vulnerability is discovered by a malicious party, it is not disclosed publicly, but is instead kept secret to allow for exploit development and following exploitation by the malicious party or a small circle of other trusted (malicious) parties. Alternatively it can be sold for profit to another interested party, which is also potentially malicious.

Some vulnerabilities can remain undisclosed for as long as 10 years, although most get publicly disclosed earlier than that [1]. The Software Vulnerability Lifecycle Model (Figure 1) shows these stages and the transitions between them as a state machine.

2.1.4 Vulnerability Market

Previously undisclosed vulnerabilities are traded both on the regular (overt) and black markets. Overt vulnerability market usually takes the form of *bug bounty* schemes, where the vendor pays the disclosing party for the vulnerability, with the intention to patch it before it is widely exploited. The monetary reward for disclosing a vulnerability (the bug bounty) is intended to motivate independent vulnerability research and disclosure. Depending on the system, vulnerability and exploit capabilities, bounties for exploits can reach as high as 2 million USD [9].

On the black market, hackers sell vulnerabilities and exploits to other malicious parties, use it to attack the affected systems and gain access to user data or cause a disruption in

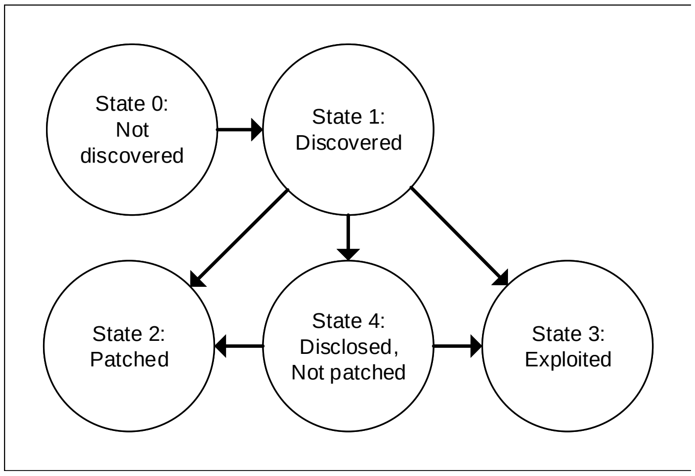


Figure 1: Simplified Software Vulnerability Lifecycle Model, showing different discovery stages of a vulnerability. Not all vulnerabilities reach all stages of the model. Taken from [8].

the service. The day of the public disclosure of an exploit can be referred to as Day 0. Since exploits are most worth before this day, they are often referred to as *zero day attacks*.

2.2 Open-source Intelligence Sources

Intelligence in the cybersecurity world represents the data and information gathered to guide decisions about operation and security of software systems. Open-source Intelligence (OS-INT) comprises intelligence data that are collected from publicly available sources, such as websites, blogs, newspapers and the like. This is in contrast with to closed intelligence sources, such as intrusion detection systems, logs or network traffic metrics [10]. In this work we focus on publicly available sources that may provide useful data about cybersecurity, vulnerabilities and exploits. We use these sources to learn about the current state of the cybersecurity space and to detect useful intelligence traits among the irrelevant, outdated or misleading data.

2.2.1 National Vulnerability Database

The National Vulnerability Database (NVD)[1] is a curated list of known vulnerabilities and their details. It is free, publicly accessible and is maintained by National Institute of Standards and Technology (NIST) of the US Government, in cooperation with Common Vulnerabilities and Exposures list[2]. CVE gathers information about vulnerabilities from vendors, researchers and white-hat hackers and maintains a list of unique vulnerability identifier numbers called *CVE IDs*. CVE ID can be assigned to a vulnerability in two ways:

[1]`https://nvd.nist.gov/`, accessed 09 April 2020
[2]`https://cve.mitre.org/cve/`, accessed 09 April 2020

1. By a CVE Numbering Authority. The CVE Numbering Authorities are usually security researchers or software vendors who oversee vulnerability listings in their own products. The numbering authorities are assigned blocks of CVE IDs in advance, maintain an overview of the vulnerabilities in their own products and assign CVE IDs from their respective blocks. The numbering authorities then update the CVE central list.
2. By the maintainers of the CVE list who assign a CVE ID directly.

The CVE ID always has the form of `CVE-YYYY-NNNNN`, where `YYYY` is a year. This year indicates when the vulnerability was first assigned or when the vulnerability was first made public without having a CVE ID assigned. This is not necessarily consistent with the year the entry has been published in the CVE list, as the vulnerability could have been assigned internally by the vendor and not made public until later, or the vulnerability was published online before a CVE ID has been assigned[3].

NVD analyses and verifies the vulnerabilities in this list and calculates the Common Vulnerability Scoring System (CVSS) score. NVD is well known within the cybersecurity community and CVE IDs, together with other NVD data are often used when referring to vulnerabilities. NVD has also been used as a data source to predict time to the next vulnerability discovery in a given system [11]. However, due to its nature, vulnerabilities often only appear in the database after they have been confirmed, patched by the vendor and verified by the NVD maintainers. Since the vendor confirmation and patch release may last several weeks, the NVD may not be suitable as a source of the newest OSINT.

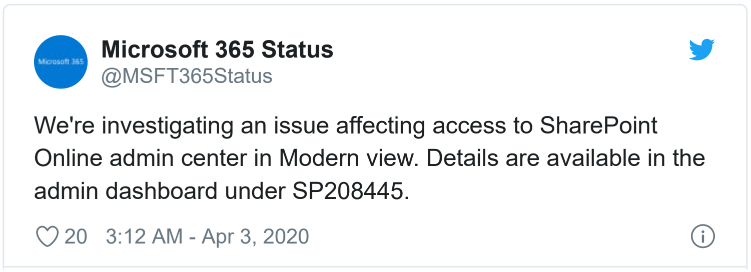
2.2.2 Twitter

Established in 2006, Twitter has since grown into a well known social media network with over 300 million users worldwide, which include politicians, celebrities and organisations. In many cases Twitter has proven to be an important source of breaking news, for example during elections [12], natural disasters [13] or terrorist attacks [14]. Twitter users interact with each other by publishing short messages called *Tweets*, that can contain media such as links, pictures or videos[4]. Twitter has a single type of account for all users, as opposed to *personal* and *page* accounts (such as Facebook) or ranked, privileged accounts (such as forums or Reddit). By default, all Tweets published by a given account are public, although this can be changed by the user to limit the audience to selected users. Twitter also offers access to its service and data via an API. This allows for development of automated accounts or bots. These are allowed on Twitter, provided they follow rules for automated accounts, such as not posting spam content and not engaging users without user's prior consent[5]. Some

[3]`https://cve.mitre.org/about/faqs.html#year_portion_of_cve_id`, accessed 29 June 2020, *archived*

[4]Tweets can be up to 280 characters long. This limit was increased in 2017 from previous 140 character limit for all languages except Chinese, Japanese and Korean.
`https://blog.twitter.com/official/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself.html`, accessed 14 April 2020, *archived*

[5]`https://help.twitter.com/en/rules-and-policies/twitter-automation`, accessed 14 April 2020, *archived*

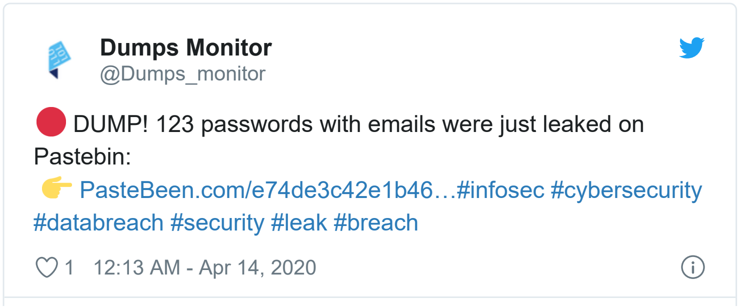


Microsoft 365 Status
@MSFT365Status

We're investigating an issue affecting access to SharePoint Online admin center in Modern view. Details are available in the admin dashboard under SP208445.

♡ 20 3:12 AM - Apr 3, 2020

(a)

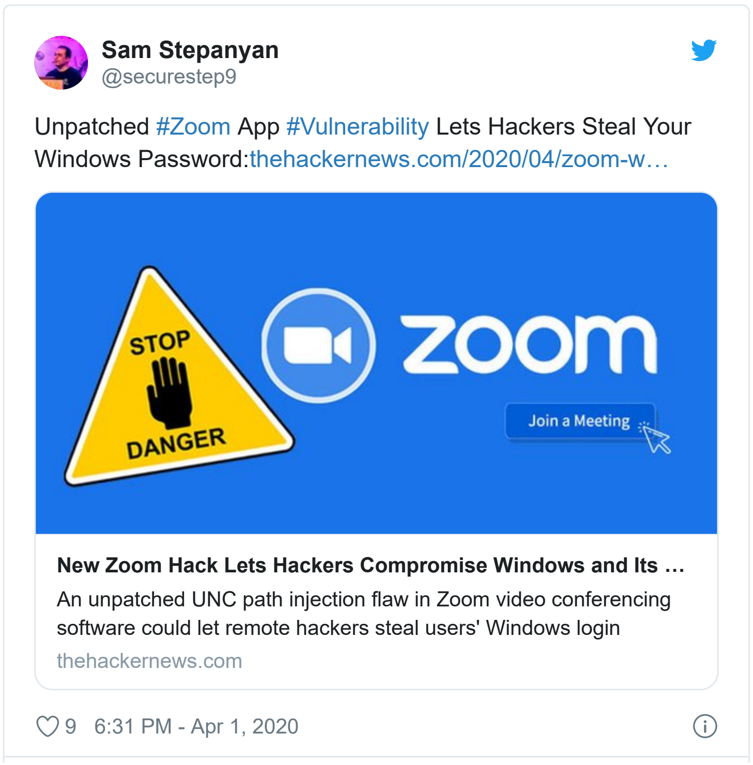


Dumps Monitor
@Dumps_monitor

🔴 DUMP! 123 passwords with emails were just leaked on Pastebin:
👉 PasteBeen.com/e74de3c42e1b46…#infosec #cybersecurity #databreach #security #leak #breach

♡ 1 12:13 AM - Apr 14, 2020

(b)



Sam Stepanyan
@securestep9

Unpatched #Zoom App #Vulnerability Lets Hackers Steal Your Windows Password:thehackernews.com/2020/04/zoom-w…

New Zoom Hack Lets Hackers Compromise Windows and Its …
An unpatched UNC path injection flaw in Zoom video conferencing software could let remote hackers steal users' Windows login
thehackernews.com

♡ 9 6:31 PM - Apr 1, 2020

(c)

Figure 2: Examples of Tweets: (a) Official vendor status account, (b) Automated (bot) account, (c) Personal account raising awareness of a fresh vulnerability.

of the popular automated accounts on Twitter include poem bot (@HaiQuBot) or personal care message bot (@tinycarebot).

In the cybersecurity community, individual users, researchers and organisations often share news about latest vulnerabilities, outages or disruptions (see example in Figure 2a and 2c). Popular automated account include data leak notification bots that tweet a warning when data such as emails and passwords are detected in some of the monitored sources (see example in Figure 2b).

In addition to basic site actions, such as post a Tweet or reply to a private message, the API offers access to full history of Tweets since 2006. Registered developers/applications can perform some API calls for free, while others (such as unlimited archive access) require purchase of premium access. Twitter has also been studied as a OSINT source for vulnerability prediction [10]. This is described in more detail in Section 3.2.

2.2.3 Reddit

Popular website on the public Internet, Reddit[6] is also a potential source of OSINT. Reddit offers a social news aggregation service, combined with a forum. The typical Reddit experience consists of forum categories called *subreddits*[7] Users post messages consisting of text or other media, such as videos, pictures or links. Other users can comment and vote on these messages and on other users' comments [15]. Most recent and most popular messages are usually shown on top of the subreddit or on the main Reddit page, which is advertised by Reddit as "the front page of the Internet". While the site allows pornographic material, provided it is clearly marked as such, illegal messages and comments are removed either by the subreddit moderators or by Reddit staff. Subreddits may also be removed entirely, if the messages in the subreddit frequently break Reddit rules [16].

Subreddits exist for a variety of topics, from entertainment (e.g. `r/memes`) to profession networks (e.g. `r/ITProfessionals`). There are several subreddits dedicated to cybersecurity and privacy, such as `r/cybersecurity`, `r/netsec` or `r/privacy`, which may provide a useful source of OSINT.

2.2.4 Paste websites

Due to their anonymity and ease of use, paste websites are another source of potential OSINT. *Paste* or *dump* websites are services that allow users to share textual data (*pastes*) by copying it onto the website, which then makes it publicly available under an individual link. The pastes do not need to be approved to be published and the users often do not need to register to create a paste, which makes paste websites an easy way to publicly share arbitrary textual data. Popular paste websites include `pastebin.com`, `gist.github.com` or `anonpaste.org`. These are closely monitored by data leak detection services[8], since user details, such as emails and passwords are often made available via public pastes [17]. Code snippets, including malware and exploits are also regularly shared via paste websites [18].

2.2.5 Deep- and dark-web

While the terms *deep-web* and *dark-web* are often used interchangeably, there is a substantial difference between the two. The deep-web is a portion of publicly accessible Internet, that is not indexed by search engines. The search engines typically crawl websites and based on the collected data construct a map of the crawled Internet, known as the *surface web*. Legitimate websites, that are not accessible to search engines make up a large portion of the deep-web. The reasons why search engines do not index these sites include dynamically built content, content requiring login or private content [19–21].

On the other hand, dark-web is used to refer to content that is not accessible with conventional browsers and requires specialised technology to interact with [21]. Several

[6]`https://www.reddit.com/`, accessed 15 April 2020

[7]Each subreddit is accessible via a URL pattern `www.reddit.com/r/{name of subreddit}`, so the they are often referred to together with the `r/` prefix, such as `r/security`.

[8]`https://haveibeenpwned.com/Pastes`, accessed 15 April 2020, *archived*

such technologies exist today, each allowing access to a distinct set of web pages. A typical example of such a technology is The Onion Router (Tor) network, which is an anonymisation network build on top of the Internet, offering privacy and encryption to its users [22].

The size of dark-web is difficult to estimate, with reports ranging from 6 000 [23] to 150 000 unique locations[9] [24]. However, 87% of the accessible web pages do not link to any other Tor locations, resulting in a very isolated page landscape. Furthermore, web pages on the Tor network appear and disappear frequently, which complicates regular visitor experience, but also automated data collection, since many of the `.onion` links may be temporarily or permanently inaccessible [25].

The anonymity and virtual untraceability of Tor allows for formation of both legal and illegal communities. Numerous forums on the dark-web are dedicated to facilitating discussion on topics such as drugs, hacking or child pornography [26]. Other sites take advantage of the anonymity of cryptocurrencies as a means of payment and serve as marketplaces, where users buy and sell goods and services [3].

2.3 Natural Language Processing

Since the early experiments with machine translation in the 1950s, the goal of Natural Language Processing (NLP) has been been to allow computers to process and analyse spoken and written human language. The field now consists of several distinct areas, each focusing on a different aspect of the language, such as lexical analysis, syntax analysis or pragmatic meaning representation [27]. The sequence of stages on the path from raw input text to intended digitised meaning forms the stratified model, illustrated in Figure 3.

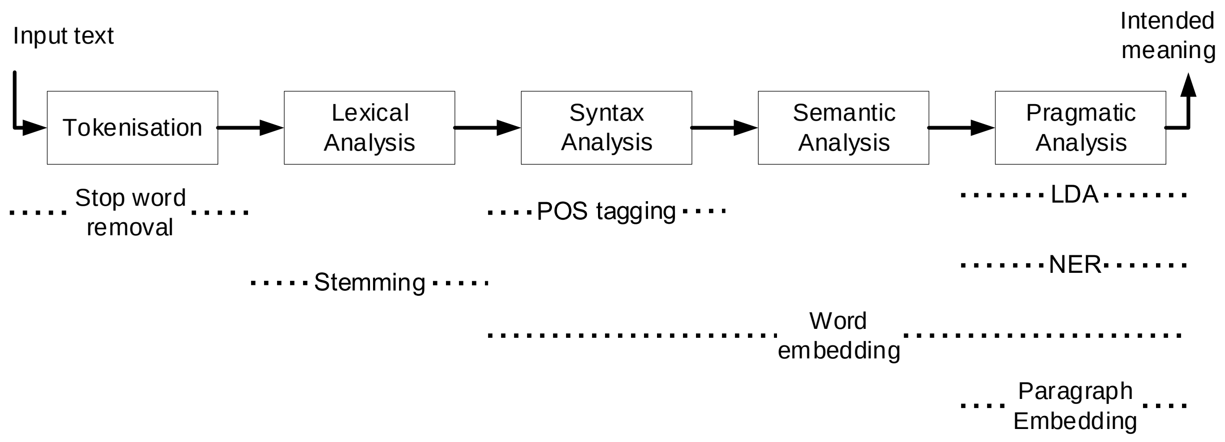


Figure 3: Boxes represent the Stratified Model, as presented in [27]. The bottom part of the figure represents selected methods of NLP and their approximate position in the Stratified Model.

[9]Official figure of unique registered Tor addresses, as measured by the maintainers of the Tor Project. `https://metrics.torproject.org/hidserv-dir-onions-seen.html`, accessed 27 April 2020.

There are multiple approaches and methodologies for each stage of the model. In this section we introduce few in further detail, which are the most relevant for our task or have been applied in similar works.

Stop word removal In a number of information retrieval, text classification and other tasks, very frequent words, such as *a*, *the* or *my* do not carry significant information value [27, pp. 455-484]. It has been demonstrated that excluding such words during data pre-processing improves the system performance [28]. Stop words are typically removed based on a static list of known stop words, which can be domain-specific.

Stemming As a part of lexical analysis, the goal of stemming is to reduce similar words to a single common root, or *stem*. This is achieved by identifying the stem of the word and discarding any affixes at the end, thus removing case, inflexion and sometimes other information [27, 28]. *Computer*, *computing* and *compute* are all mapped into a common stem *comput*. Although the effectiveness of stemming varies in different languages, it is often used in search tasks or when generating a list of keywords.

Part of Speech Tagging In a number of scenarios, such as topic mining or information retrieval it is useful to see a sentence not only as a list of words, but also to understand grammatical and syntactical relations between the words [27]. Part of Speech tagging assists in sentence segmentation by assigning a syntax label (tag) to every word in the sentence, such as *singular noun* or *verb*. The set of the tags used is dependant on the language of the text and the tagging system must understand and apply grammatical rules of that language.

Named Entity Recognition Both spoken language and written text are often ambiguous. *"I am going to New York."* could mean that the person is going to the New York City or to the state of New York. One of the ways to avoid this ambiguity is Named Entity Recognition (NER). This technique annotates or replaces words, often nouns, with a clear specific token, so that only the terms referring to the same real-world entity are assigned the same code. Similarly, if a real world entity is often referred to by several different terms (such as *Coca-cola*, *cola* and *coke*), all those terms are assigned the same token [29]. Recent approaches to NER are able to determine the most probable entity from the surrounding text. Recurrent Neural Networks, such as LSTM offer good performance on this task [30].

Latent Dirichlet Allocation Many use-cases require that a general theme or topic is extracted from the text. Latent Dirichlet Allocation (LDA) is statistical model that is used to categorise a set of documents into a given number of categories, based on frequency of different words in each document [31].

Word Embedding To analyse, compare and capture the meaning of individual words, it is useful to transform them from a text form into a continuous space that allows for such operations. Word embeddings are used to map words to a vector space with a relatively low number of dimensions especially in comparison to other encoding methods, such as one-hot encoding. Vector representations of two words with similar meaning are normally positioned close in the vector space [32]. This allows for a quick and inexpensive calculation of similarity between words, which is useful in a number of tasks, such as natural language understanding, machine translation or information retrieval.

While probabilistic models, knowledge bases and other methods can be used to produce word embeddings, neural networks offer the best performance and are widely used in practice, for example the word2vec/Skip-Gram embedding model [33]. A more recent model BERT introduces the use of a novel transformer neural network architecture, which achieves even further performance improvement by considering the surrounding context before and after each word [34].

Because word embeddings are not strongly task-specific, simpler models such as word2vec can be used to provide ready-made embeddings for the entire corpus, which simplifies implementation. On the other hand, since embeddings produced by BERT are context-dependent, they cannot be pre-calculated. However, pre-trained BERT models exist for corpora in several languages, as well as models, which are fine-tuned for application in specific fields, for example PatentBERT (patents), SciBERT (science papers) or DocBERT (patient records) [35–37]. This enables the user to calculate embeddings in real time, without the need to train a complex embedding model.

The idea of word embedding can also be extended and applied to sentences, paragraphs or entire documents [38, 39]. Such embeddings can similarly be used to determine semantic distance between two units of text.

2.4 Machine Learning

Machine learning is a field of computer science that essentially uses a large number of data to help a machine improve performance on a given task. This differs from other programming approaches that undergo the usual design path of detailed problem specification, followed by an implementation into a programming language [40, 41]. The field of machine learning has grown significantly in the past decade, with availability of cheap computing power and large amounts of data being arguably two of the contributing factors. A plenitude of different algorithms and approaches to machine learning exist and are widely used both in research and in the industry. In this section we introduce basics of supervised learning, including the commonly used SVM model. We then provide an overview of selected neural network architectures, as these offer state-of-the-art performance in a variety of tasks, including attack prediction [42]. Evaluation metrics and core unsupervised approaches follow towards the end of the section.

2.4.1 Supervised Learning

In supervised learning, the algorithm is presented with a set of data points together with an expected result for each data point. An algorithm that is to distinguish pictures of apples from pictures of bananas must be trained by looking over a large number of pictures, while the right answer is supplied with each picture (*this is a picture of a banana*, *this is a picture of an apple* and so on) – the *ground truth*. During this training, the algorithm gradually learns to detect identifying signs of apples and bananas and categorise previously unseen pictures as either 'apple' or 'banana' [40].

Marking input data as belonging to one of several distinct categories is known as labelling and is the core part of the *classification* problem. In classification, the output is discrete – such as categories, colours or a true/false label. When the algorithm is to output a continuous output, such as temperature, price or area, the problem is known as *regression*.

Linear Regression The most basic form of predicting a continuous output is the linear regression, which can be formally expressed as follows:

$$X * \theta = Y \tag{1}$$

where X is the input matrix, with rows representing the observations and columns representing the features of the input data, θ is the parameter vector and Y is a column vector with the ground truth values. The goal of linear regression is to optimise the parameter θ, such that the equation is satisfied for all rows of the input data. In practice however, the noise in the input data (and possibly also in the ground truth values) causes that the equation will never be fully satisfied for all rows of X. Instead, such θ must be found so that the difference between the predicted values Y_p and ideal values Y is as small as possible [40].

Support Vector Machine Linear regression outputs a continuous output. When the the problem requires classification, other algorithms must be used. One of the widely used classification algorithms today is the Support Vector Machine (SVM) model. This model classifies the input data into classes based on a decision boundary, also called *hyperplane*. Considering a simple dataset with only two features, we could visualise the data and the hyperplane on a plot (see Figure 4). Depending on the data, it is possible that several hyperplanes could equally well classify the training data. However, to ensure the best performance of the model on yet unseen data, such a hyperplane should be picked, that the distance between the datapoints and the hyperplane is maximised [43]. SVM achieves this by solving a quadratic programming problem [44].

Originally, the SVM could only classify linear problems. However, applying a method known as the *kernel trick* the model performs well also on non-linear problems. Kernel trick involves remodelling the input into a higher-dimension space, where it is easier to find a separating hyperplane [44].

2.4.2 Neural Networks

Somewhat analogous to the human brain, artificial neural networks have been envisioned as an approach to achieve parallel processing [41]. Network consisting of many neurons that individually process data and exchange information, have proven to be capable of approximating complex non-linear functions. The basic architecture of an artificial neural network

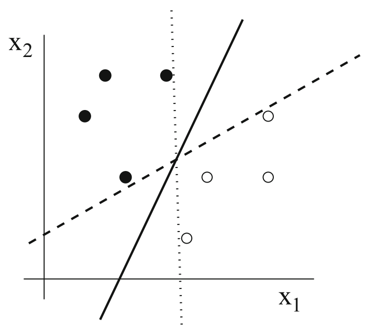


Figure 4: Exemplary dataset with two classes (black and white points), two features (x_1 and x_2 axes) and three candidate hyperplanes (full, dashed, and dotted line). The full black line is the best hyperplane as it maximises the margin between the hyperplane and the data. Taken from [43].

consists of an input, one or more hidden layers and an output layer. Each layer is made up of several *neurons* or *perceptrons*, which gather data on their input, perform an operation and then output the processed data further. The design parameters of the network determine the number of layers and neurons in each layer, the operation performed in each neuron (also called the *activation function*), allowed connections between neurons and whether there are any feedback loops between the layers. Figure 5 demonstrates a simple Neural Network with a single hidden layer. During the training, the network self-adjusts how much information is passed in and out of each neuron.

Most basic neural networks that do not have any loops (such as the one in Figure 5) are called feedforward networks, as the data only flows in one way. In some use-cases when processing continuous input, such as machine translation or event prediction, it might be useful for the network to retain some information between data inputs. For example in machine translation, the translation of a word may change depending on the words that come before it in a sentence. To achieve this, loops may be introduced into the network architecture that allow for retention of some data between inputs [40, 41].

LSTM Common RNN architectures today include Long Short-term Memory and Gated Recurrent Unit (GRU). LSTM proposes a memory cell with three gates [46]:

- *input gate* that controls how much new information enters the cell
- *forget gate* or *update gate* that controls the amount of information retained by the cell from the last run
- *output gate* that controls how much information is output from the cell

Such composition of gates allows the cell to retain information in the hidden state even throughout many input cycles. For certain NLP tasks, a modified LSTM cell called GRU has provided a performance improvement. GRU simplifies the cell by replacing the *input* and *forget* gate by a single *update* gate [40].

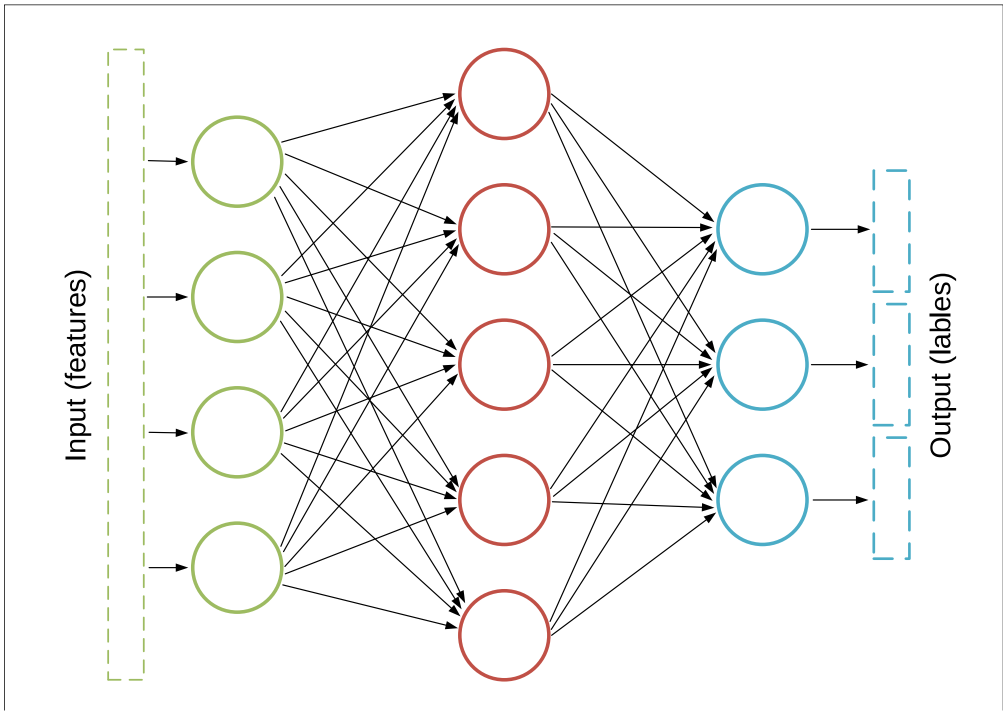


Figure 5: Simple feed-forward neural network architecture. Circles represent individual neurons. Green colour signifies the input layer, red colour the hidden layer and blue colour the output layer. Taken from [45].

	Predicted Positive	Predicted Negative	Total
Class Positive	*True Positive (TP)*	*False Negative (FN)*	p
Class Negative	*False Positive (FP)*	*True Negative (TN)*	n
Total	p'	n'	N

Table 2: Confusion matrix for positive and negative class.

2.4.3 Evaluation in Supervised Learning

To understand, adjust and evaluate the performance of neural networks, but also other types of algorithms, several metrics are commonly used. The evaluation approach depends on the availability of labelled data, as some metrics cannot be calculated without the ground truth labels. In this section we only consider evaluation metrics typically used with labelled data in a supervised setting.

In a classification problem with two classes, the outcome can be one of the four cases – TP, TN, FP and FN, as illustrated in Table 2. A simple measure of the classifier performance could be to count how many of the examples were classified correctly – to calculate *accuracy*:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N} \tag{2}$$

However, this metric does not perform well in cases where one class is more represented than the other class. Consider a login monitoring system that classifies login attempts as *normal* or *suspicious*. Most login attempts are from regular users and should be classified as *normal*, with only as small fraction of logins coming from a malicious party, which should be classified as *suspicious*. If the classifier predicted *normal* in 100% of the cases, the accuracy would remain high, despite the non-performing classifier [41].

To address this issue, recall and precision have been introduced:

$$\text{Precision} = \frac{\text{TP}}{p'} \qquad\qquad \text{Recall} = \frac{\text{TP}}{p} \tag{3}$$

Precision reflects the fraction of truly positive classes among all that were classified as positive. Recall reflects the fraction of correctly classified positive classes from all that should have been classified as positive. Different applications place different importance to precision and recall values. For example in the login system scenario it is not a a big problem if a legit user's login is marked as *suspicious*, but it is important that all malicious attempts are classified as such. In this system, the recall value should be as high as possible. In another scenario, an automated air-defence system can deploy missiles after a target has been classified as a *threat*. It is important that such system recognises all incoming attacks, but it is even more important that no civilian aircrafts are mis-classified as *threats*. The precision value of such a system should therefore be as high as possible. The balance between precision and recall can often be adjusted by changing the classification threshold [40].

To compare different algorithms using a single measure, or when the relative importance of precision and recall is not known, the F-score can be used to combine the two:

$$F_1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Subscript 1 in F_1 indicates that equal weight is given to both precision and recall. Variants of the metric exist, which give higher weight to one or to the other [43].

For balanced datasets[10], another metric is commonly used to compare different models – the area under the ROC curve. Receiver Operating Characteristics curve is a visualisation of TP and FP rate as the classification threshold varies. This can be useful to understand the performance of classifier with different settings. The area under the ROC curve is a useful single metric to directly compare different classifiers. The more the area approaches 1, the better the performance of the classifier [43]. Figure 6 illustrates an ROC plot.

2.4.4 Unsupervised Approaches

Unsupervised approaches do not utilise labelled data during training. This is often necessary, as labelled data may be difficult or impossible to obtain in sufficient quantity. Clustering, which is the most prominent example of unsupervised learning, groups data into clusters

[10]Balanced datasets are such datasets where the number of samples in each class is approximately the same. This is not necessarily true with data with a skewed distribution. For example, a cancer detection test could be truly negative in 90% of all tested cases. This would produce imbalanced dataset.

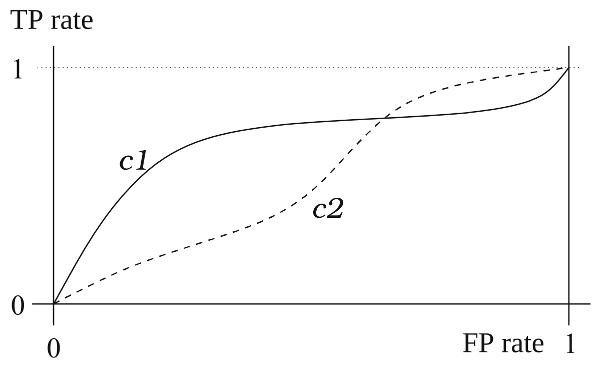


Figure 6: Two ROC curves for different classifiers. Ideally, the curve reaches TP rate of 1 and FP rate of 0, although in practice this typically does not happen. Taken from [43], edited.

based on similarity of the data features [41]. For example, a clustering algorithm may be trained to group customers into segments based on their purchase history, credit rating, age and education level.

An example of a simple clustering algorithm is K-means, which groups data into clusters based on proximity to a *centroid*. Centroids are calculated points that represent the average of a cluster. K-means takes the user-supplied parameter k, representing the number of desired clusters, and randomly splits the data into k clusters. It then iterates over individual data points, calculating the distance to all centroids and re-assigning the cluster membership, if the data point is not already belonging to the cluster represented by the nearest centroid. After each re-assignment, the centroids are recalculated and the algorithm finishes once all the data points are assigned the correct cluster [43].

2.5 Forecasting Methods

Drawing information from and predicting future values of a sequence of temporal observations is the domain of time series analysis. Time series operate with the assumption that the data have an underlying multivariate distribution, which was observed through time together with some amount of noise [47, 48]. To understand existing trends in the data and to predict future observations, models are constructed that attempt to fit the data as closely as possible. Two major classes of time series models are (i) moving average, and (ii) exponential smoothing model.

Moving Average Models The simplest moving average process, the first-order moving average process represents the relationship between the observed data and unknown amount of noise [47, 49]. In this process, the observation Y_t at time t depends on the current noise

e_t and the noise observed in the previous observation e_{t-1}, multiplied by a weight θ:

$$Y_t = e_t + \theta e_{t-1} \tag{5}$$

A number of models is based on the moving average. Autoregressive Moving Average (ARMA) introduces an autoregressive part to the moving average model, often denoted as ARMA(p, q). The parameters p and q determine how many past data points influence the current data point [47, 49]. In a simple ARMA(1,1) model, the observation at time Y_t depends on the current noise, noise in the one last observation, and value of the one last observation, multiplied by a parameter ϕ.

$$Y_t = e_t + \theta e_{t-1} + \phi Y_{t-1} \tag{6}$$

To extend the ARMA model on time series with trends, where the mean is not stationary, the Autoregressive Integrated Moving Average (ARIMA) includes the *integrated* part, which factors in the difference between current and past observations. Autoregressive Integrated Moving Average with Exogenous Variables (ARIMAX) further allows inclusion of external features in the model [4].

Exponential Smoothing Similarly as the autoregressive part, exponential smoothing takes into account the values of past observations [48]. The smoothed value S_t is calculated as:

$$S_t = \alpha y_{t-1} + (1 - \alpha)S_{t-1} \tag{7}$$

where α is a *smoothing factor*, such that $0 < \alpha < 1$. While exponential smoothing does not perform well for predicting long-term values, it is useful to filter out high frequency noise.

Due to the potentially disastrous consequences of unaddressed vulnerabilities, it is easy to see the necessity of finding and patching newly discovered vulnerabilities as early as possible. The preceding paragraphs presented a framework for classifying vulnerabilities and the vulnerability life-cycle, followed by a number of sources and techniques that can be used in early vulnerability detection. Works that combine and use these techniques follow in the next chapter.

3 State of the Art

Hackers, white-hat bounty hunters and software vendors focus primarily on finding existing vulnerabilities in a particular software, based on penetration testing, known weaknesses or flaws in design and implementation, with the intention to exploit, disclose or patch them. However, a number of researchers addressed the aspect of systematically predicting and detecting new vulnerabilities, by means other than directly attempting to compromise the given software.

OSINT is often a major data source for these systems, therefore we first describe how OSINT can be collected from various sources and processed. Existing datasets and software tools are also mentioned here. Vulnerability prediction using OSINT and other data sources is described after that.

3.1 Open-source Intelligence Collection Systems

Data relevant for Open-source Intelligence can be collected from a variety of surface-, deep- or dark-web sources. As shown in Section 2.2, many popular surface-web sites cater to the need of automated data access by providing date feed APIs. When a website does not offer such APIs, or even actively prevents automated data collection, as many dark-net sites do, the data collection poses a much larger challenge.

As the underground sources often engage in diverse malicious activities, such as drug trading or terrorism, researchers in fields other than computer security also tend to monitor and analyse them. This results in existence of field-agnostic tools, that are able to collect data regardless of the later application. For example, [50] focuses on discovering new and categorising existing `.onion` sites on the Tor network in an automated fashion.

ACHE and AHMIA Focusing generally on dark-web forums, [51] offers a targeted dark-web crawler architecture. *Targeted* or *focused* crawlers are such crawlers that prioritise and automatically decide which pages to visit and save, based on the given target area or topic. Typical approach is to specify the target area using a list of keywords. If a keyword is found in the page URL or body, the page is saved, otherwise it is discarded. Popular contemporary implementation of a focused dark-web crawler is ACHE[11], which offers dark-web crawling by connecting to Tor via a proxy. An alternative to the targeted approach is the untargeted crawler, which collects all available pages that it encounters. Example implementation of such a system on the Tor network is the automated crawler AHMIA[12]. Both targeted and untargeted approaches are capable of accessing and capturing dark-web content, but cannot easily circumvent anti-crawler measures, such as CAPTCHA.

CrimeBB Focusing more narrowly on forums related to cybercrime, [52] presents a crawler tool used to monitor several selected forums. Authors deploy this tool to continuously crawl

[11]`https://github.com/VIDA-NYU/ache`, accessed 27 April 2020, *archived*

[12]`https://github.com/ahmia/ahmia-crawler`, accessed 27 April 2020, *archived*

the forums for new content and save this information to a database called *CrimeBB*, making it available to other researchers. The tool combines an automated crawler with manual input of a human operator to circumvent CAPTCHA and create user accounts on forums that require it. At the time of publication in 2018, the CrimeBB dataset consisted of 48 million posts by over 1 million users and has grown since.

AZSecure Similarly utilising selected cybercrime forums as the primary data source, the AZSecure Hacker Asset Portal presented in [53] describes an architecture to monitor and analyse OSINT in real time. The system consists of a data collection part, which collects code snippets or attachments posted by users of one English language and one Russian language forum; an analysis part, which uses Latent Dirichlet Allocation to identify the functional area of the exploit as either *web*, *network* or *system*; and a presentation part that reports on and displays these findings. Data used in this work, as well as Twitter collections, Dark-web market data, IRC logs and other have been made freely available online[13] as a series of *Intelligence and Security Informatics Data Sets* [54].

BlackWidow A recent 2019 study presents a technical framework for extracting OSINT form a number of dark-web forums [23]. This framework consists of five stages: (i) planning, (ii) collection, (iii) processing, (iv) analysis, and (v) dissemination. In the first stage, forums are handpicked and connection to these forums is established manually, mainly registering user accounts and solving CAPTCHA. In the second stage, the forums are crawled automatically, using several containerised crawlers in parallel to speed up the collection, while still mimicking user browsing habits to avoid anti-crawling measures deployed by the forums. The third stage is responsible for parsing the collected files, but also automatically translating foreign language content to English, which was not observed in other works. The last, Analysis stage of BlackWidow is revisited in Section 3.2.

17 Forums Another work that collects deep- and dark-web cybersecurity intelligence is [3]. The proposed system collects data from underground forums and markets, using a custom crawler that addresses link collection and accessibility challenges, although no implementation details are given on these. The total amount of collected data is more than 11 000 products from 17 markets and more than 160 000 posts from 21 forums, although less than 20% of these are considered as cybersecurity relevant by the authors. We revisit the data analysis part of this work in the following section.

DNM Archives dataset Not all efforts to crawl and capture dark-web content come from the research community. One of the few large public datasets focused on dark-web data was compiled independently, funded by a crowdfunding campaign [55]. The *Darknet Market Archives* dataset provides raw `html`, `css` and image files, collected between 2013 and 2015 from 89 markets and 37 forums. Crawlers were deployed manually by the authors in regular intervals and required creating user profiles on portion of the sites and manual CAPTCHA

[13]`https://www.azsecure-data.org/home.html`, accessed 27 May 2020, *archived*

solving. The data amounts to over 43 million files or 1.5TB, however as acknowledged by the authors, may not fully represent accurate state of the sites at any given time, due to frequent updates, instability of Tor connections and attrition rate of dark-web pages. Nevertheless, the dataset poses as an important resource in dark-web research due to its breadth and depth.

Widening the focus to collect OSINT from a variety of sources, [2] describes an early warning system that integrates with Twitter and deep- and dark-web pages. Twitter data is collected via the Twitter API from a manually compiled list of trustworthy security researchers. Deep- and dark-web data is made up of a manually compiled list of 200 sites about hacking, financial fraud, phishing, ransomware and similar, which are periodically crawled and their contents saved to a database, using customised crawlers and parsers. Focused crawling is employed due to the size of the page pool, although no details are provided on circumventing anti-crawl measures. It is important to note, that although similar to ours, the cited work does not specifically focus on predicting new vulnerabilities, but rather on predicting cyber-attacks in general, including DoS attacks and exploitation of known vulnerabilities. The data analysis part of this paper is also revisited in Section 3.2.

CLUO Further away from the dark-web, [56] focuses on designing a platform capable of collecting OSINT from a number of diverse sources. This platform, named *CLUO* allows for data inflows from web crawlers, forums, blogs, social networks and databases. In the exemplary case study, the authors collect 1 billion documents from forums, blogs and news sites using the platform. However, the platform serves more as a central data repository to enable subsequent processing and analysis, as all of the mentioned connectors must be manually configured by the user and do not, on their own, solve the mentioned accessibility problems.

An even wider array of sources is used to gather intelligence data in [4]. The data is used to predict cyber-attacks that are likely to happen in the near future, based on latent signals identified in the data. Apart from deep- and dark-web, twitter and blogs, this work also monitors existing vulnerabilities, published NVD entries and signals from a network of honeypots deployed by the researchers. For deep- and dark-web, the techniques described in [3] are used. Like [2], this work is similar to ours, with a wider focus on predicting cyber-attack in general, rather than on predicting new vulnerabilities.

While each of the works listed in this section has a slightly different focus, it is apparent that intelligence collection from the web attracts considerable attention from researchers and communities. Table 3 provides an overview of the listed papers with a breakdown of their main focus areas.

3.2 Data Analysis for Attack Prediction

Supposing we have collected data from a number of various sources, the next step would be to extract the useful information from this data and filter out the noise. This information can then be analysed and used to predict imminent cyber-attacks.

Reference	[50]	[51]	ACHE	AHMIA	[52]	[53]	[54]	[23]	[3]	[55]	[2]	[56]	[4]
Type	R	R	T	T	R, D	R	D	R	R	D	R	R	R
Crawling	●	●	●	●	●			●	●	●	●	●	●
Deep-&Dark-web	●		●	●	●	●	●	●		●	●		●
Forums					●	●	●		●	●	●		●
Markets								●	●	●	●		●
Diverse Sources							●	●			●	●	●
Cyber-crime					●	●	●	●	●		●		●

Table 3: Overview of OSINT Collection Systems in Section 3.1. 'Type' indicates the type of resource: ***R*** *– Research,* ***D*** *– Dataset,* ***T*** *– Tool. Bullet (●) indicates that the resource focuses on the given area.*

There is significant research interest in detecting ongoing and predicting imminent and future attacks, breaches and vulnerabilities. In this section we group the existing works by their main data analysis method. We first illustrate the breadth of the field by listing works that use traditional forecasting approaches or simpler machine learning models, other than neural networks. Later we introduce works that are more similar to the focus of our work, which use neural networks for data analysis and prediction.

3.2.1 Simple Forecasting Approaches

A simple, yet effective dictionary-based method for generating threat warning is presented in [2]. The textual data collected from various sources gradually passes through a set of dictionary filters that filter out stop words, common English and Italian words and known technical phrases. Counts of remaining words are accumulated and highest-counting words are presented as *possible threads*, together with additional context information.

Common statistical method used to find trends in time-bound data is Moving Average (MA) and its derivatives, such as Exponential Moving Average (EMA) or Autoregressive Integrated Moving Average (ARIMA). These methods have been applied to predict future threats from web signals [4] and local machine logs [57]. While these methods are effective in predicting numerical values, they are not optimised for working with text and may produce large number of false positives without careful pre-processing of the input data.

Focusing closely on textual features of collected data, the BlackWidow system [23] uses Latent Dirichlet Allocation (LDA) to detect the general topic of the given data-point (for example *botnet*, *database exploit* or *DDOS*). The counts of different categories are then aggregated and projected in a time-series to display current trends. Additionally, BlackWidow infers user relationships based on the proximity of messages in forum threads and identifies individual users' identities across forums. This helps to illustrate the overlap between otherwise isolated communities on the dark-web.

3.2.2 Non-neural Machine Learning Approaches

Arguably the simplest machine learning algorithm – linear regression has been used in [11] to predict time in days until the next vulnerability, exclusively using data from NVD. In addition to linear regression, other approaches, such as Mean Square Error or Gaussian Process were also tested, although as the authors note no accurate model could be build from NVD due to incompleteness of the data.

Random Forrest Classifier Random Forest classifier, which offers a clear insight into the most significant variables in the classification task [40] has been used to predict probability of a security breach in [58]. This work focuses on predicting security breaches only within a target organisation with data collected from security posture assessment systems and malicious activity detection systems deployed within the organisation's network, in addition to public OSINT sources. The reliable data sources and subsequent good data quality allowed for a carefully crafted feature set.

Bayesian Classifier Slightly different in focus, [59] presents an intrusion detection system based on traffic data collected in a local network. This work analyses time-bound data by slicing them into segments of variable duration, where each segment represents a 'connection session' with a period without any traffic before and after the session. These segments are classified as *malicious* or *benign* using a Naive Bayes classifier on a low number of features, such as destination port or packet Time-to-live.

SVM Popular across a variety of supervised use cases, the Support Vector Machine algorithm is often used in cyber threat analysis. Both [53] and [60] employ SVM to classify collected code snippets into programming languages, before using LDA to detect the functional area of the code (such as *web*, *system* or *network*). In [60] SVM offers the best performance on the given task, compared to other approaches, such as k-Nearest-Neighbour or Naive Bayes classifier.

To predict whether a given vulnerability will be exploited, [61] proposes a DarkEmbed model. This model uses Skip-Gram embeddings of deep- and dark-web forum posts, as well as additional vulnerability features, such as CVSS score as captured from NVD. Anti-virus database and Metasploit are used as the ground truth for training the network, providing an overview of publicly exploited vulnerabilities.

In a similar system, [62] relies on manually selected features from NVD and keywords identified in Twitter posts. An SVM classifier labels vulnerabilities as likely and unlikely to be exploited.

SVM and other supervised approaches require labelled data to train, which are not always readily available. To address this issue, semi-supervised methods can offer good performance with a lower number of labelled samples. In [3], the authors demonstrate the advantages of this approach by using co-training with SVM on a dataset with labels available for only 25% of the items. In this work, the task consists of classifying individual items from dark-web marketplaces and posts on dark-web forums as relevant or non-relevant for purposes of further cybersecurity analysis. Combination of co-training and SVM outperforms vanilla SVM, Random Forrest, Naive Bayes and Logistical Regression classifiers.

Unsupervised Approach While [63] and [64] focus on analysis of network data, rather than textual data, they provide an example how unsupervised machine learning methods can be used to draw information from time-bound data. In [64], each data point includes network metrics such as number of packets and entropy of source IP address during a span of a fixed duration – 1 second. These data points are then grouped using hierarchical clustering into 6 distinct groups, corresponding to phases of a DDoS attack. [63] builds on top of this system by including two other forecasting techniques: exponential smoothing method of time-series analysis and Markov Chain probabilistic modelling method. By combining these, the authors were able to reduce the false-alarm rate of the DDoS prediction system.

3.2.3 Neural Network Approaches

Predicting malware attacks in an organisational setting, [4] utilises Recurrent Neural Networks for time-series analysis. This work considers a wide span of time-bound textual data as the input for the neural network, including tweets, deep- and dark-web posts, NVD or honeypots. The data is first pre-processed in order to adjust potential time difference (lag) caused by different sources reacting to the same event at a different time (for example a honeypot signal may indicate a malware attack before someone mentions it on Twitter). This correlated data is then supplied to a GRU. Authors claim that this approach outperforms more traditional time-series analysis method ARIMAX, although closer details of the neural network architecture are not provided.

LSTM on Twitter Data Another type of RNN architecture, LSTM cells are used for DDoS attack prediction from tweets in [42]. The goal of this work is to predict the attack target before the attack happens, motivated by common expressions of negative emotions, anger or hate towards organisations on social media. Sudden spike in such expressions might indicate an upcoming attack. The architecture of the neural network presented in this work consists of word embedding layer, two CNNs and an LSTM layers as follows:

1. The input tweets are processed into word embeddings.
2. The most important features of each tweet are extracted using the first CNN.
3. The most important tweets of each day are selected using the second CNN.
4. Most important *daily digests* are supplied to a stream model, consisting on one or more LSTM layers, depending on the configuration.
5. Output of the stream model is passed to a softmax classifier that outputs the most probable attack target.

The stream model configuration determines whether the model accumulates daily (single LSTM layer), weekly (two LSTM layers) or monthly (three LSTM layers) data. The neural network is trained end-to-end, with the exception of word embeddings, where pre-trained embeddings based on Skip-Gram algorithm are used. The neural network architecture used in this work is illustrated in Figure 7.

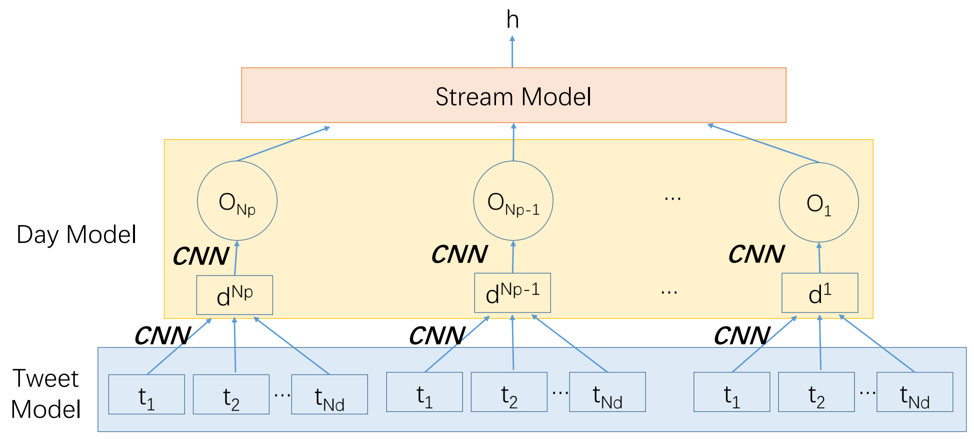


Figure 7: Architecture of a DDoS prediction system. T_{Nd} represents the embedded tweets from the given day. Figure taken from [42]

.

LSTM on Structured Data LSTM has been used to predict cybersecurity-related events also in [65]. The authors collected security system alerts generated during a hacking competition, from which they extract metrics such as port numbers, alert severity or alert signature. Training sequences are generated by sliding a window of size 100 with stride of 25 over the time-sorted data. A model described in this work predicts the next security alert from the previous alerts. A single LSTM layer is connected to a dense layer with the number of nodes corresponding to the number of features of an alert.

To verify that a network can learn anything from the data, cross-validation is commonly used. During cross validation, data is divided into batches and in each training iteration, random batch is selected to evaluate the performance. In [65], authors instead use *predictive classification*, where the data is kept in chronological order and the earlier data is always used for training, while the later data is always used for evaluation. During the training iterations, the split between train and test set gradually changes, so that in the end, 80% is used for training and 20% for testing.

Predicting cyber-attacks and other security-related events is an active research area. A variety of data sources from local and private to publicly available can be used to gather insightful information for automated prediction systems. Despite its apparent abundance, systematic gathering of data from public data sources is often challenging and time-consuming. Several works, which focus on this problem were presented in the beginning of this chapter. In the second part, works which focus on the analysis of the data are introduced. While traditional algorithms play significant role in time series analysis, we can see a strong focus on supervised machine learning approaches, namely SVM. Neural networks and LSTMs are also being introduced to the problem as seen in some of the most recent works. Further analysis of the vulnerability prediction problem follows in the next chapter.

4 System Design

The previous chapter introduced a number of existing systems and approaches to cyber-event detection. However, all of these differ in their focus from our work. In this chapter we explore the problem of early vulnerability detection in closer detail. We begin by highlighting the key features of the envisioned zero-day warning generator system. We then identify the key states of the Vulnerability Lifecycle Model where our system should focus. Based on these findings, we set out a number of goals for our envisioned system. In the last section of this chapter we introduce vulnerability detection as a machine learning problem and formalise requirements on training data and label generation.

4.1 Envisioned Vulnerability Detection System

In the Introduction chapter we describe a hypothetical alert system that gathers and filters data from online sources. The alerts generated by this system should assist system administrators in learning about new vulnerabilities, so that they can react appropriately and minimise the risk that a previously unseen vulnerability is used to exploit their IT infrastructure. Additionally, the alert system could also be used by software vendors to find any new vulnerabilities relating to the software they maintain. The general architecture of the alert system could consist of three subsystems, described as follows:

1. **Data discovery and collection subsystem**, which is responsible for autonomous gathering of data from a number of diverse relevant sources. These are primarily forums, blogs, social networks and other online locations where information about previously unseen vulnerabilities might appear. The *discovery* of new data sources should ideally be automated, due to sources and communities emerging and disappearing frequently in the OSINT space. The data *collection* should also be automated and continuously ongoing to ingest new data.
2. **Analysis subsystem**, which handles the data filtering and information extraction. This part must identify information about previously unseen vulnerabilities in the incoming data. To achieve this, it should filter out all content that is not related to vulnerabilities and exploits. It should detect topics in the data and identify if a single topic appears across different sources to remove possible duplicated or very similar information. Finally, it should be able to identify long-term topics in the data and should be able to detect which information provides "breaking news", and which only relates to vulnerabilities in a more high-level, general way.
3. **Presentation and alert generation subsystem** that handles the data presentation to the end user. This could include dashboards, alert services or APIs. The data presented to the user should be concise, timely and as reliable as possible given the inherently untrustworthy nature of the underlying sources.

This architecture is illustrated in Figure 8. All three subsystems are complex and include their own design and implementation challenges. In this project we focus primarily on the Analysis subsystem, while keeping in mind the high-level architecture.

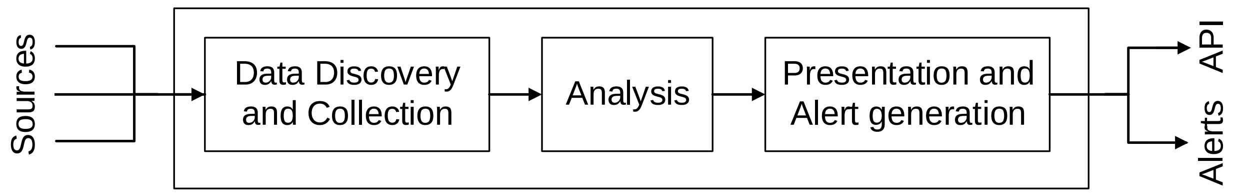

Sources
Data Discovery and Collection
Analysis
Presentation and Alert generation
API
Alerts

Figure 8: High-level architecture of a hypothetical vulnerability alert system.

4.2 Target Vulnerability Lifecycle States

The goal of the Analysis subsystem is to identify new vulnerabilities. However, defining a vulnerability as 'new' is ambiguous. Considering the Simplified Vulnerability Lifecycle Model (see Figure 9), we can identify specific life states of vulnerabilities, which the model should focus on.

The vulnerability begins its lifecycle as *Not Discovered* (State 0 in the model). Once the vulnerability has entered the *Discovered* state (State 1), some information about it may potentially appear on the web. Ideally, the system should detect these to identify vulnerabilities, which are in the Discovered state. From the defensive perspective, the system should detect vulnerabilities before they are turned into exploits and enter the *Exploited* state (State 3). However, even as an exploit is developed for a give vulnerability, the system should be able to identify the indicative signals, as this allows the system operators to anticipate a potential attack.

When the vulnerability enters the *Disclosed* state (State 4), the software vendor is made aware of the vulnerability and may begin to develop a patch. However, before the patch is available and applied, the vulnerability can still be exploited. The system should be able to identify vulnerabilities in this state, especially if the vulnerability progressed quickly through the preceding states.

Once the vulnerability enters the *Patched* (State 2) a patch is made available by the vendor. We assume that this patch is applied on the effected software immediately and therefore the vulnerability ceases to pose a thread. The system should not report on vulnerabilities that have entered this state. Figure 9 highlights the lifecycle states where our system should focus.

4.3 Goals

Based on the interaction with the other parts of the Vulnerability Detection System and the vulnerability lifecycle states, we can formalise the main goals of our Analysis subsystem.

- The Analysis subsystem should identify emerging vulnerabilities in the input data. These vulnerabilities must be in State 1, State 3 or State 4 of the vulnerability lifecycle model.
- The data pre-processing and analysis should be automated. User input must not be necessary to process, clean, rank or highlight data.

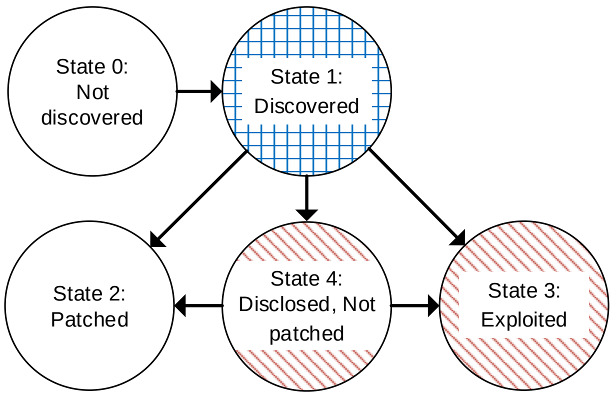


Figure 9: Simplified Vulnerability Lifecycle Model [8] with highlighted states where the vulnerability detection system should focus on. ▦ *indicates primary focus,* ▧ *indicates secondary focus.*

- The Analysis subsystem should prioritise identifying vulnerabilities, which it has not encountered before.

As stipulated by the high-level system architecture, the Analysis subsystem is not concerned with data collection. The collected data are fed as an input and are processed as necessary, inside the Analysis subsystem.

On the other side, the output of the Analysis subsystem is the list of newly discovered vulnerabilities. This list should be as detailed as possible, so that the system users can use this list to take preventive measures based on the extracted condensed information. The items on the list should not be repetitive and should not contain vulnerabilities that have already been reported, even if these are still in States 1, 3 or 4 of the lifecycle model. However, the presentation of this list to the end user, as well as the potential alert generation is not handled in the Analysis subsystem, but rather in the Presentation and Alert Generation subsystem further downstream.

4.4 Solution Design

Due to the complexity of the tasks of the Analysis subsystem, it is not trivial to implement a solution by using existing, off-the-shelf systems and frameworks. Some of the tasks of the subsystem, such as information filtering, topic detection or duplicate removal are already reliably solved by existing products, for example social feed algorithms or email spam filters. However, the requirement of a very narrow, focused attention window differentiates this problem from these examples, and renders the existing solutions unfit.

Several of the works cited in Section 3 utilise machine learning to address needs similar to ours. This comes naturally, as machine learning algorithms offer state-of-the-art performance on numerous complex tasks and have been used extensively in the fields of NLP and forecasting.

4.4.1 Vulnerability Detection as a Classification Problem

The machine learning model should ideally be capable of picking up signals about vulnerabilities in any system, including systems that it has never encountered before. Data from the NVD indicate that just a few software products contain the majority of all reported software vulnerabilities. Just 50 products (less than 0.1%) with most reported vulnerabilities comprise around 11% of all reported vulnerabilities (see Appendix A). While maintaining the usefulness, we can reduce the complexity of our machine learning model by limiting the focus of the analysis subsystem only on the top N software products with most vulnerabilities.

This allows us to frame the task as a classification problem, where the machine learning model attempts to predict, which of the known N software products (classes) are most likely to contain a new vulnerability. While N should be as large as possible, ideally $N =$ number of all software packages, in practice the system can be useful even with smaller N.

Supervised learning is generally used for classification tasks when labelled data is available. Since in our case, both historical data from diverse online sources, as well as historical data about vulnerabilities are available, we can train our machine learning model in a supervised setting.

4.4.2 Machine Learning Model Architecture

Several diverse approaches to supervised classification can be used. In many applications, neural networks offer superior performance due to their ability to capture complex abstract patterns in the data. Due to the varying levels of quality, reliability and structure of online sources, it is desired for our model to be able to understand tacit, underlying trends and topics. For this reason we employ a neural network model, as it is more likely to learn these than other, simpler supervised approaches.

Designing a neural network requires setting a number of hyperparameters. These determine the number and type of layers in the network and the number of neurons in each layer. They also specify the activation function in each layer or the loss algorithm used to train the network. There are few guidelines on designing an optimal neural network model architecture and often the choice of hyperparameters is left to be determined experimentally by the researcher [40, 66].

One approach to designing a neural network is to adapt an existing architecture, which is known to perform well on a similar task and adjust it as necessary. The system described in [42], which shares key features with ours, mainly the focus on time-based textual data and event prediction, uses a neural network to predict DDoS attacks (see Figure 7 on page 25). We base the design of our network on this work and adapt it so it better fits the problem at hand.

In particular, the cited paper specifies processing input data on Tweet Sub Model, Day-level Sub Model and Stream Sub Model. Tweet Sub Model is designed to capture "*text-level features for individual Tweets*" [42], the Day-level Sub Model captures the features of the Tweets in a given day and the Stream Sub Model captures features that extend throughout several days.

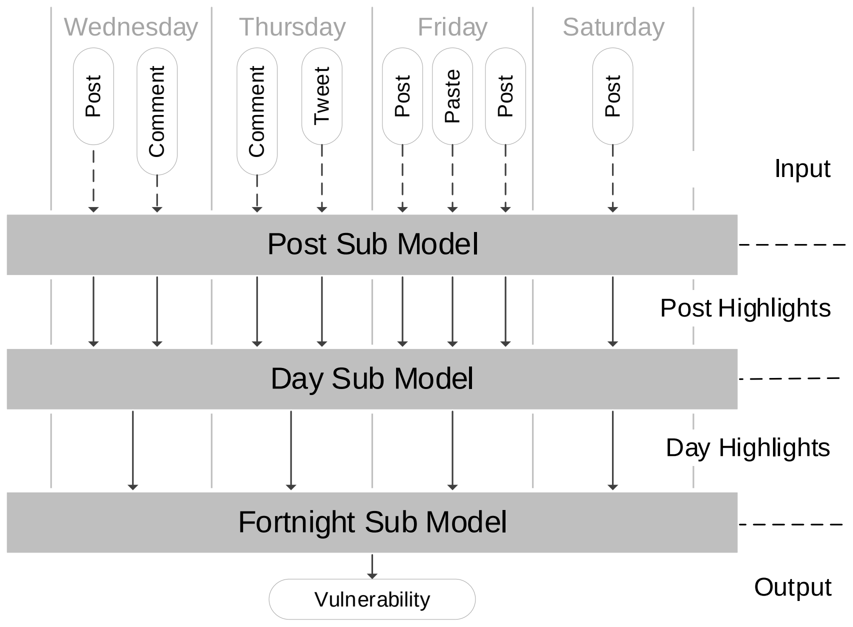


Figure 10: Data flow between three sub models. On top, the individual posts are ingested. Each sub model performs processing on the data and passes the output to the next sub model. The architecture of the model is inspired by [42].

In our neural network, the first sub model is the *Post Sub Model*, which is somewhat analogous to the Tweet Sub Model, but adjusted to operate on a wider variety of sources. This first sub model captures the most informative features of each text post (forum post, Tweet, comment, or other text-based input data) and passes them onto the *Day Sub Model*. The Day Sub Model extracts the most important features from all posts of each day. The output of the Day Sub Model is passed further onto the *Fortnight Sub Model*, which is somewhat analogous to the Stream Sub Model, but consists of fewer layers. The Fortnight Sub Model gathers the most important features from a specified number of days. This could be 14 days, if the sub model is to capture posts spanning two weeks, but could also be adjusted to more or fewer days if necessary.

Post Sub Model Each of the three sub models consists of one or more trainable layers with neurons and zero or more non-trainable layers. In the Post Sub Model, there is a single convolutional layer that convolves over the post text and filters out the most important features. Its outputs are passed to a max-pooling layer that reduces the dimensionality of the data by dropping the features identified as least important by the convolutional layer.

Day Sub Model As each post is processed separately by the Post Sub Model, all posts from the same day must be joined together before they are further processed. This is achieved by a concatenation layer that operates on the outputs of the Post Sub Model. The concatenated posts are processed by a trainable convolutional layer that filters out the most important features of each day and a max-pooling layer to reduce the dimensionality.

Fortnight Sub Model Similarly as in the Day Sub Model, in the Fortnight Sub Model the inputs are first concatenated together. This data now represents the highlights of each day, joined together. Until now, each post was processed independently of the other posts and each day was processed independently from the other days. However, to capture possible trends and topics throughout the time, in the Fortnight Sub Model an LSTM layer is introduced. This layer sequentially processes the concatenated output of the Day Sub Model, maintaining internal state throughout the processing. Finally, the output of the LSTM layer is connected to a dense layer, which performs the final classification.

Hyperparameters In this architecture, several hyperparameters must be set that further describe the model. These are:

- Stride, number size of filters in the convolutional layers in the Post Sub Model and Day Sub Model
- Size and stride of max-pooling layers in the Post Sub Model and Day Sub Model
- Number of neurons in the LSTM layer in the Fortnight Sub Model
- Activation functions in the Convolutional layers and the final dense layer
- Loss function
- Optimisation function and learning rate

Most of these hyperparameters must be set experimentally, depending on the performance of the model. We describe them in further detail in Section 5.3. The full machine learning model is comprised of the three sub models. During training, evaluation and prediction, the sub models always act together.

4.4.3 Data Model

Before the model can perform any classification, it must first be trained. As this is a supervised setting, we must supply labelled data to the model. The selection and structure of the data determines the performance and utility of the model. For example, if the labels do not accurately represent the real-world patterns behind the data, even a highly complex model would have trouble learning anything useful.

Obtaining Labelled Data In supervised learning, obtaining good and reliable labels to use for training is often a challenge. In an hypothetical system that classifies social network posts as hate speech, it may be easy to obtain large quantities of social network posts. However, labelling each post as hate speech or not to produce training data for this is a tedious task that requires large human effort in reading, comprehending and labelling each individual post.

Sometimes, instead of humans labelling each individual data point, the labels might be inferred from other information. In the hypothetical hate speech detection system, we could achieve this by producing a list of offensive keywords. If a word from this list is present in a post, the post is labelled as 'hate speech'. This removes the requirement for manual human work, but introduces other problems, potentially reducing the quality of the labels. Consider a post which quotes and condemns other person's hate speech. This post would likely get mislabelled by a keyword-based system.

Label Formalisation In our scenario, the ground truth data is the presence of a new vulnerability at a given point in time. However, this definition is somewhat abstract and must be formalised before it can be used as a label in a dataset. Normally when past vulnerabilities are references, a CVE-ID is used. However, this ID does not get assigned until the vulnerability is properly discovered and investigated, which is too late for our system. Sometimes fresh vulnerabilities may be referred to by name, such as *Heartbleed* or *Spectre*. However, this is not a reliable label, as not all vulnerabilities are known by a unique name.

For our use case, a vulnerability may be understood as *(i)* a novel way to breach *(ii)* a specific system. If a system administrator was provided with both *(i)* and *(ii)*, they would effectively know about the vulnerability, without knowing the vulnerability name or its CVE-ID. As hinted in Section 4.4.1, our proposed solution focuses primarily on predicting the affected system or a software package.

The system or systems affected by each vulnerability can be learned from the NVD data. The time factor, indicating when did the vulnerability reach the Discovered state or one of the following states can also be inferred from the vulnerability database. We assume that if a vulnerability was discovered publicly (that is, not by the software vendor or a white-hat hacker) and discussed on the web, this happened before the vulnerability was investigated and added to the NVD/CVE list. We further assume that the time frame for this delay is in the order of days or weeks. Once the vulnerability has been recognised and assigned a CVE ID, it is arguably not considered *new* anymore, even if it has not reached State 2 (Patched) of the Vulnerability Lifecycle Model yet.

Therefore, all posts published on the web up until the vulnerability is assigned a CVE ID (and thus listed in the vulnerability database) might contain information about a new, unseen vulnerability and are potentially relevant for our system. A complete row of the data can then be modelled as follows:

$$< p_1 \, , \, p_2 \, , \, \dots \, , \, p_k \, , \, B_V > \qquad\qquad B_V = [s_1 \, , \, s_2 \, , \, \dots \, , \, s_N] \tag{8}$$

where a single training row, denoted by angle brackets, consists of up to k posts and a ground truth label B_V. The label corresponds to vulnerability V and contains a list of up to N software packages s_n, that are affected by V. Each post p must be published before a CVE ID was assigned for V.

Data Representation There are multiple approaches to turn text into a numerical representation that can be interpreted by a machine learning model. The more traditional vector space model approach captures the meaning of text as vectors based on the word frequency and their co-occurence [67, 68]. Many NLP applications today utilise the more recent text embedding approach [33] to achieve this, as it offers superior performance to other methods [69]. However, text embedding can be performed on text units of different sizes. Besides the most popular word embeddings, methods for sentence, paragraph and even document embedding exist [38]. As the granularity moves towards larger units of text, the overall size of the encoded data shrinks on the expense of details captured by the embedding. Document embedded with paragraph embedding is shorter and carries fewer details than the same document embedded with word embedding.

In our scenario it can be expected that some of the posts might carry specific keywords indicating a presence of a new vulnerability, such as system names or technologies used. These concrete terms would be lost or significantly diluted if sentence or paragraph embeddings were used. For this reason we prefer word embedding in this project.

As pre-trained off-the-shelf word embedding models such as BERT or Word2vec generally offer good performance, it is not necessary to include an embedding layer in the machine learning model and train it together with the other layers. Instead, the word embeddings should be produced with an external embedding model during data pre-processing, independently of the vulnerability detection model, and serve as an input to for the vulnerability detection model.

In this chapter we introduce the envisioned end-to-end system, followed by defining what a 'fresh' vulnerability is in terms of Vulnerability Lifecycle Model states. We define a high-level machine learning model architecture consisting of three sub-models and specify, how the training data for the model must be represented. In the following chapter we describe the implementation of these concepts.

5 Implementation

Based on the goals specified in the previous chapter, as well as the high-level architecture of the machine learning model we proceed with implementation and training of the model. In the following sections we first describe criteria on data sources for our system and select sources from which we collect data according to these criteria. We then describe this collection process and architectures of the collection systems, as well as obtained data and sourced datasets.

In the second part of this chapter we introduce particulars of the neural model implementation, including an overview of layers and model hyperparameters.

5.1 Automated Data Collection

A natural first step in analysing signals from the web is to gather data that might contain these signals. As outlined in Section 2.2, there are numerous diverse communities, platforms and channels online, which may carry insightful information. We employ automated methods to discover and save data from these platforms as a first step towards their analysis.

As noted by [70], the data collection and preparation is often a bottleneck that consumes significant time and resources in machine learning projects. To maximise the potential information yield of the collected data, we assess the prospective data sources by these criteria:

- *Data Quality — Is there a high chance that the resource contains useful data for our task?* Discussions about vulnerabilities and zero-day attacks are often kept within small communities of trusted individuals. As vulnerabilities and associated exploits may be created with a malicious intent, platforms which either explicitly focus on such content or implicitly allow its sharing due to little or no content moderation are preferred by professional hackers. For example, sharing of exploit code is much easier in a dedicated closed hacking forum, than it is on Reddit.
  The data need not necessarily be in text format, however it should carry useful, timely information about vulnerabilities in their early life stages.
- *Data Quantity — Is there enough data to allow for meaningful model training?* Due to the small size of the window between when the data is first available until it becomes outdated, it is to be expected that such data is sparse and must be filtered out from a large amount of irrelevant noise. Because of the high expected noise ratio, enough data should be collected, so that it is useful even after the noise is filtered out.
- *Ease of Data Collection — Is it possible to collect sufficient amount of data without extensive manual effort?* Tue to the necessity of collecting a large amount of data, the data collection system should be highly automated and should minimise the necessity for operator input. The availability of APIs for data access reduces this problem, as do automated crawler systems to an extent. On the other hand, elaborate CAPTCHA systems, as seen on dark-web or complex configuration requirements of honeypot networks are examples of factors which prohibit effective autonomous data collection.

Data source	**Quality**	**Quantity**	**Ease**	**Used**
Blockchain		●	●	
Blogs	●			
Deep-/Dark-web	●	●	○	Yes
Honeypots	●			
Internet traffic		●		
Paste websites	○	●	●	Yes
Reddit	○	●	●	Yes
Twitter	○	○	●	Yes

Table 4: Evaluation of data sources by the data quality, data quantity and ease of data collection. Bullet (●) indicates that the source meets the criterion. Circle (○) indicates that the source partially meets the criterion.

A number of diverse data faucets, which are described in detail in Section 2.2 is considered against these criteria, namely: Twitter, Reddit, Paste websites, multiple deep- and dark-web forums and marketplaces. In addition, security researchers' blogs, blockchain, honeypots and internet traffic are also evaluated. Table 4 presents an overview of this evaluation, while the following paragraphs provide furhter justifications for each source.

Blockchain Popular public blockchains, such as Ethereum or Bitcoin might carry some information about new vulnerabilities, either as embedded transaction messages or in the patterns of transaction destinations and amounts. However, the information is very sparse and practically impossible to interpret due to the privacy-focused nature of these blockchains. The data, however is easily accessible and available in large quantities.

Security Researchers' Blogs Security researchers' blogs have been used as a data source for cyber-event prediction in [3, 4, 56]. These blogs, written by professional researchers, activists or white-hat hackers carry high-quality, timely information that would be useful for our system. However, the number of these blogs and of the articles published on them is not sufficient for construction of a dataset suitable for training a neural network. Furthermore, the list of the blogs must be compiled manually by the operator to maintain the high information quality with some level of operator involvement during the collection process as well.

Deep- and Dark-web Many underground hacker forums, communities and markets are hosted on deep- and dark-web. These carry information of both good quality and sufficient quantity, provided that initial entry barriers, such as forum registration or user verification are overcome. Addressing these barriers, as well as solving technical challenges associated with automated data collection from sites on the deep- and dark-web, in particular high attrition rate, slow connection speed and anti-crawling measures deployed by the sites reduce the ease of automated collection.

Honeypots Similarly as blogs, honeypots were previously used as a intelligence monitoring tool to predict cyber-events in [4]. It is assumed that a honeypot network deployed and controlled by researchers might get targeted by a fresh, previously undisclosed vulnerability, thus yielding high-quality intelligence. The disadvantages of this approach are the complexity of deploying and operating such a network, as well as the difficulty to estimate the amount of useful data collected by this method.

Internet Traffic Analysis of incoming and outgoing internet traffic has been used to predict cyber-incidents localised to a given network group or organisation [58, 59, 63]. Odd traffic patters, source IP addresses and ports may indicate a previously unseen breach. However, this method has been exclusively used in a controlled, demarcated environment where the incoming and outgoing traffic can be easily monitored. Employing this approach on public internet is unfeasible due to the vast volume of traffic, as well as number of required monitoring probes.

Paste websites Paste websites are often the source of database leaks and for this reason they are intensively monitored by the data privacy community. They are also used to share code snippets, including malicious code and exploits. These only make up a small portion of all data pasted to these websites and therefore must be filtered out from the noise. However, paste websites are mostly public and due to the ongoing data leak monitoring efforts, first- and third-party APIs are available. This allows for easy collection of large amount of data.

Reddit Reddit also concentrates IT professionals and enthusiasts, and has sections dedicated to cybersecurity and online privacy with member numbers in tens of thousands. However, Reddit is also moderated and has previously deleted entire forum sections that violated its policies. For this reason it is not likely utilised by black-hat hackers who require non-moderated platforms to share illegal content and exploits. Data can be collected from Reddit using both official and third-party APIs.

Twitter For many, Twitter is not only a social network, but also a source of breaking news [13, 14]. In the cybersecurity space this manifests sharing of in information about latest data leaks and vulnerabilities. This data can be accessed via Twitter's official API, which imposes a pricing scheme with only a small quantity of data available for free[14]. Anti-crawl measures, such as rate limiting are deployed by Twitter that pose a significant challenge when attempting to collect data in other way than via the API.

Base on the outlined criteria, we select deep-/dark-web, paste websites, Reddit and Twitter as sources for our data collection. In the following sections we describe the general setup of

[14] `https://developer.twitter.com/en/pricing/search-fullarchive`, accessed 09 June 2020

the collection system and particulars of data collection from each source.

5.1.1 Collection System Setup

The main purpose of the collection system is to save and reliably store the raw collected data for later analysis and pre-processing. The collection scripts are developed and tested locally and then executed on a remote server.

The remote server is deployed as a Docker container, running Python 3.6 in an interactive mode[15]. Containerised version of Python has been selected due to simplicity of its deployment and management (as opposed to a standalone virtual machine). The container is hosted in Microsoft Azure Container Instances[16] service and provided with network access. As Docker containers are inherently ephemeral and maintain no data when shut down[17], a persistent storage space is attached to the container. Decoupling of the storage from the container simplifies versioning and backup of the captured data. Microsoft Azure Files[18] is used to host the collected data and provide a storage space to the container.

5.1.2 Deep- and dark-web

To collect data from deep- and dark-web we first identify a number of relevant web locations by searching in known surface-web forums, link lists[19], and surface, as well as hidden, search engines. We focus primarily on hacking forums and exclude any websites which require payment or a "proof of interest" (usually an example piece of code written by the user to demonstrate their interest and skills) to access their contents. Similarly we exclude forums where an invite from an existing member is required to access the content. The output of this investigation phase is a list of prospect dark-net and deep-net forums to crawl (see Appendix B).

The ACHE crawler[20] is deployed as a Docker container in Microsoft Azure Container instances. Its configuration files are stored in a dedicated location in Azure File Storage service and it is operated in a server mode, which means that it can be operated via an exposed API interface. As the crawler cannot directly access the Tor network, it is paired up with a Tor proxy[21] service, also deployed as a Docker container. This proxy can interact with the Tor network and serves as a transparent proxy for the crawler. Figure 11 depicts the architecture of this setup.

For sites that require signing in or solving CAPTCHA to access their content, this is carried out manually by the operator before the crawl is started. The session cookie issued by the site is extracted and provided to the crawler, which uses it to simulate a real-user

[15]`https://hub.docker.com/_/python`, accessed 09 June 2020

[16]`https://azure.microsoft.com/en-us/services/container-instances/`, accessed 09 June 2020

[17]`https://docs.docker.com/storage/`, accessed 09 June 2020, *archived*

[18]`https://azure.microsoft.com/en-us/services/storage/files/`, accessed 09 June 2020

[19]Link lists are websites which aggregate links to other, un-indexed websites, often on the dark-web. Example of a link list: `https://www.deeponionweb.com/`, accessed 13 June 2020

[20]`https://github.com/VIDA-NYU/ache`, accessed 27 April 2020

[21]`https://hub.docker.com/r/dperson/torproxy/`, accessed 13 June 2020.

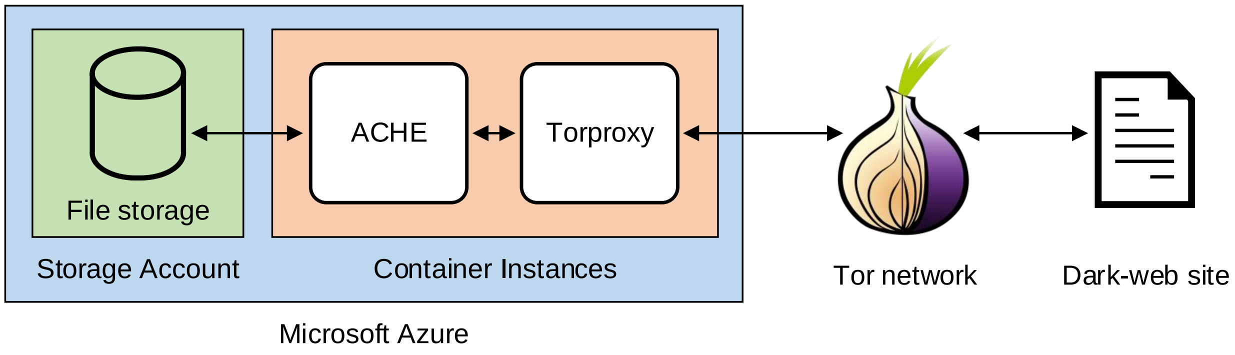


Figure 11: Overview of the ACHE crawler setup. Crawler and proxy to the Tor network are deployed as docker containers. Crawler has access to file storage system to save retrieved pages.

browsing. The crawler maintains a backlog of all links that it encounters, which are located within the target website and visits them sequentially, saving each page it visits to the storage. However, the speed of the crawl is limited to prevent site deploying anti-crawl and anti-DoS measures against the crawler's IP address.

Collected Data In the first run, the crawler was deployed on three forums from the list (see Appendix B). It successfully obtained around 6 thousand pages from *0day Forum*. In case of *Torigon*, the crawl was not successful due to poor accessibility of the site, with only around one out of every five requests receiving a response. It is suspected that this was due to site issues and not anti-crawl measures, since the accessibility problems persisted even when the access was attempted via an alternate Tor circuit or at a later time. In case of *cryptBB* the crawl was not successful due to inability to replicate a singed-in user session in the crawler. Even despite mimicking cookies and the user-agent string of a manually established session, the crawler was not able to obtain responses to its HTTP requests. We suspect that discrepancies between crawler's requests and legitimate users' requests were identified by the site's anti-crawling measures and purposefully ignored.

5.1.3 Pastebin

Pastebin[22] was launched in 2002 is one of the most well-known paste websites. It does not encrypt users' contents and publishes all pastes by default (although private-only pastes are available to premium users). For these reasons we choose to collect data from Pastebin as the representative of paste websites.

Pastebin offers an API access to premium members, but due to its popularity, a number of third-party services exist that aggregate and publish Pastebin content. Pastebeen[23] is an example of such service and provides paste monitoring and analysis via a web interface.

[22]`https://pastebin.com/`, accessed 14 June 2020
[23]`https://pastebeen.com/`, accessed 14 June 2020, *archived*

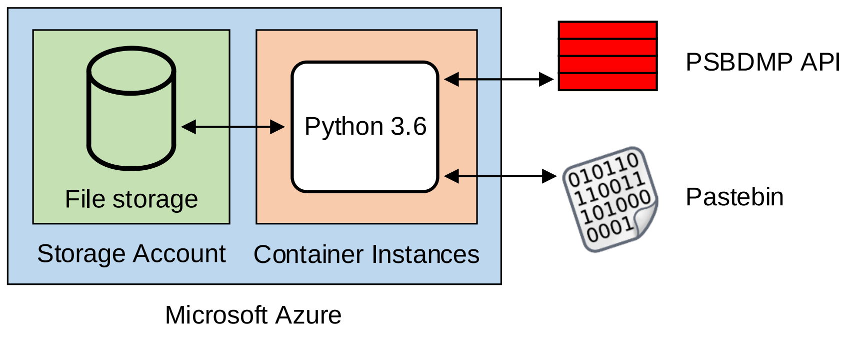


Figure 12: Setup for collecting data from Pastebin. Paste IDs are retrieved from a third-party API (PSBDMP) and pastes are retrieved directly from Pastebin.

Another third-party service PSBDMP[24] offers API keyword search in the public Pastebin pastes.

To collect data from Pastebin we first compile a set of keywords used to identify (potentially) relevant pastes (see Appendix C for full list). We use this list to query the PSBDMP search API and retrieve URIs of relevant pastes. The pastes are then fully downloaded directly from Pastebin and saved to storage. This setup is illustrated in Figure 12.

Collected Data Using this setup and a list of 20 keywords, we were able to retrieve 2579 pastes. This number could be easily extended by including further keywords in the list. However, PSBDMP is a third-party service, which does not appear to be maintained anymore[25], as of June 2020. Because of this, it cannot be guaranteed that all pastes that contain the requested keyword are returned, or even identified by this API. Possible remedy to this issue is to compare paste counts for a given keyword returned from PSBDMP with another source, such as PasteBeen, or use the official Pastebin API.

5.1.4 Reddit

It is possible to access Reddit data both via an official Reddit API or one of the third-party tools. The official API is free and open for everyone, provided that client issuing requests is authorised via OAuth and does not exceed 60 requests per minute[26]. Third party services, such as Pushshift[27] provide access even without authentication.

Our collection system uses the *submissions* endpoint of the Pushshift API to retrieve Reddit posts as this removes the need to authenticate our client. Submission in this context represents a top-level post posted to a subreddit, and its associated media attachments, but not the associated replies. The API only returns maximum of 1 000 posts per request, but allows modifying the requests using a number of query parameters, such as keywords,

[24]`https://psbdmp.ws/`, accessed 14 June 2020, *archived*
[25]`https://twitter.com/psbdmp/status/1240730515738632194`, accessed 14 June 2020
[26]`https://github.com/reddit-archive/reddit/wiki/API`, accessed 14 June 2020, *archived*
[27]`https://pushshift.io/api-parameters/`, accessed 14 June 2020, *archived*

submission time or score. Returned from the API are JSON data with detailed information about each submission.

We collect data from subreddits *r/cybersecurity*, *r/security* and *r/netsec*. To avoid the limit of 1 000 returned submission per request, we issue multiple requests and include `after` and `before` parameters in each query to limit our search to a window of limited time. If this window is small, the number of requests in the window will not reach the 1 000 submissions limit. Window of size one month has proven to be sufficiently small.

Collected Data We collected around 130 000 posts from the three mentioned subreddits that range from May 2013 to April 2020. The collected data contains the submission timestamp, submission text, links to any attached media, username of the uploading user, full URL of the submission and other meta-data.

5.1.5 Twitter

Twitter provides an official API to access it's data. It is free to apply for access to this API, but the applications are reviewed and approved by Twitter, provided they follow its policies. The API section dedicated to searching and retrieving Tweets is divided into two endpoints: (i) a *full-archive endpoint*, which provides access to all Tweets since 2006, and (ii) a *30-day endpoint*, which provides access to Tweets published in the last 30 days since making the request[28].

Both of these endpoints are available in three tiers, which determine the maximum amount of data that can be retrieved from the endpoint and allowed query parameters. More fine-tuned queries, for example keywords, phrase matching, or hashtag search are only available in the higher tiers. The tier also determines the maximum allowed length of the search query, with lower tiers only able to issue shorter queries[29]. The lowest *standard* tier is available for free, while the higher *premium* and *enterprise* tiers must be purchased. To the best of our knowledge, there are no third-party services that provide reliable access to Twitter data. Twitter prohibits any non-API based automation in its policies and takes measures to prevent actions such as crawling and site scraping.

To gather data for this project, we were granted limited (*sandbox*) access to Twitter's Premium tier search API. Sandbox access means that the developer may use some of the premium functionalities, but the number of requests and retrieved Tweets is limited (see Table 5).

Collected Data Using Python interpreter deployed in a docker container, we download data from this API by a list of keywords in each query. Posts should contain a hashtag that matches at least one of the query keywords. In both the 30-day and full-archive endpoint,

[28]`https://developer.twitter.com/en/docs/tweets/search/overview/premium`, accessed 14 June 2020, *archived*

[29]The query length is measured in total number of characters, not the number of keywords. For example, search query `#cve OR #nist` (14 characters) is considered shorter than search query `#vulnerabilities` (17 characters).

Endpoint	# requests	# Tweets	Query Length	Query parameters
Full-archive	50	5 000	256 characters	Standard
30-day archive	250	25 000	128 characters	Standard

Table 5: Rate limits for Sandbox access to Premium tier Twitter API. Standard Query parameters mean, that the query cannot utilise Premium search operators, such as filtering by follower count or user location.

the number of available Tweets is higher than the allowed rate, however the total number of successfully saved Tweets is slightly lower than the maximum rate due to a number of requests being used up for developing and testing the code. The saved data are in JSON and contain the Tweet text, timestamp, list of hashtags, associated media, information about the user, such as Twitter handle, username or number of followers, and other data about the Tweets.

5.2 Datasets

As outlined previously, there are several projects that collect, store and analyse various OSINT sources. Several of these efforts have resulted in publication of extensive datasets, which are made available publicly to support further research. This reduces the effort to discover and collect data from primary sources, which is a tedious and time-consuming process, as outlined in the previous section.

During this project we obtain data from three different datasets. In this section we provide an overview of these datasets and the contained data.

5.2.1 CrimeBB

In a 2018 work, authors from Cambridge Cybercrime Center (CCC) of the University of Cambridge introduced a data collection framework and a CrimeBB dataset of forum posts collected with the framework [52]. At the time of publication of the cited paper, the dataset comprised 48 millions of posts made by 1 million users. These posts were collected from several public underground forums, primarily focusing on communities related to cybercrime. The CrimeBot framework, described in this work addresses a number of common data collection issues, such as CAPTCHA, rate limiting and other anti-crawl measures. Moreover, it delivers on non-technical challenges associated with automated data collection, such as completeness of the collected data, incremental collection of new items and efficiency of the collection process.

From each visited page, the crawler saves the sub-forum, thread, post and user details. It saves all content available on four[30] forums at the time of the crawl, with retrieved posts ranging from January 2007 to January 2018.

[30]The crawled forums were: *Hackforums*, *Offensive Community*, *Kernelmode* and *Multiplayer Game Hacking*.

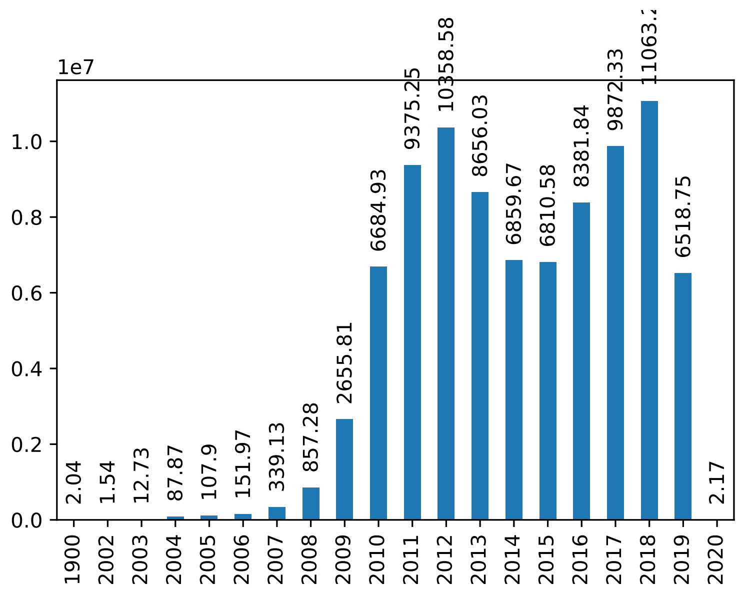


Figure 13: Distribution of posts over years in the CrimeBB dataset. The y-axis is in tens of millions, year 1900 denotes missing data.

Upon entering a Data Protection Agreement with CCC[31], the CrimeBB dataset was made available for use in this project. The collection of the data has been ongoing since the publication of the paper and the scope has been extended to include a number of other relevant deep- and dark-web forums. The size of the dataset thus now exceeds 88 million posts and spans from 2002 until 2020. Figure 13 shows the distribution of these posts over the years.

The provided data contains the text of the relevant sections (posts, user details etc.), stripped of the HTML control elements and other irrelevant strings. Other media often present on forums, such as pictures and videos, are identified in the text by annotations, but their raw form is not part of the dataset.

5.2.2 Dark Net Market Archives

A comprehensive dataset focusing on underground markets was published in 2015 [55]. This dataset includes a collection of smaller datasets between the years 2011 and 2012 and material crawled between 2013 and 2015. It contains data from 89 dark- and deep-web markets, 37 forums and few other sites and there are over 43 million files in the dataset, 1,5 TB in size. However, as most of the site content is saved raw, including unparsed HTML and CSS files, images, thumbnails, not all of this material is immediately useful without further processing. While a large portion of the items traded and discussed on the markets are illegal drugs, it is likely that sections that specialise on cybercrime and vulnerabilities exist on a number of markets and forums. This subset of the captured material is of our interest. The dataset is

[31]`https://www.cambridgecybercrime.uk/`, accessed 15 June 2020

downloaded and decompressed on a CLAAUDIA[32] virtual Linux server.

5.2.3 Dark Net Forums (AZ Secure)

The final dataset sourced for this project is the Dark Net Forums dataset[33], published on the AZSecure portal of the University of Arizona, which was introduced in Section 3.1. Comparatively smaller than the previous two, this dataset contains over 120 000 posts across three underground forums[34] focusing on cybercrime. The posts range from April 2015 until September 2018 and include details such as post title, text, timestamp and author username. A significant part of the posts that were scrapped from the Rutor forum (up to one third of the dataset) is in Russian language. The posts are available in textual form, without any accompanying media, HTML or CSS files.

5.3 Model Training Setup

The data sourced from the primary sources and from the datasets are used to train the machine learning model described in Section 4.4.2. To achieve this, the data must first be pre-processed and the model must be implemented in code and initialised. The prepared data can then be used to train the model in various configurations to experimentally find the most suitable model configuration.

5.3.1 General Setup

The data preparation and experiments are carried out on the CLAAUDIA platform. The data is loaded from the database and pre-processed on a virtual server deployed in the Compute Cloud. The data is transferred to AI Cloud for training and evaluation. The AI cloud is based on two NVIDIA DGX-2 servers, each containing 16 GPUs optimised for machine learning workloads and fast /acrshortraid persistent storage.

5.3.2 Input Data

Due to its volume, the CrimeBB dataset is selected as the main data source. The amount and the consistent structure of the data allows to select as few or as many items as necessary. As performing the pre-processing and training on the entire dataset would not be feasible, data from year 2016 is selected arbitrarily for development of the data pre-processing pipeline and the machine learning model. This speeds up the experiments, while allowing for the use of data from the other years, if an increased amount of training data is necessary for evaluating the model and validating its performance.

The ground truth data are obtained from NVD, which offers a free public data feed in a CSV format. The data obtained from other sources are not used during model development

[32]`https://www.claaudia.aau.dk/platforms-tools/compute/compute-cloud/`, accessed 15 June 2020
[33]`https://www.azsecure-data.org/dark-net-markets.html`, accessed 15 June 2020, *archived*
[34]The forums included are *Rutor*, *Wallstreet* and *Silkroad*.

and training, due to low volume of the data (crawled sites) or inconsistencies in the data structure and need for subsequent extensive parsing (Dark Net Forums dataset).

5.3.3 Data Pre-processing Pipeline

The CrimeBB dataset is sourced from the Cambridge Cybercrime Center and stored in a PostgreSQL database on a CLAAUDIA server. The data corresponding to the year 2016 is extracted from the database into a CSV file.

Label Construction NVD vulnerability records are read from storage and examined. Each record contains a list of software products affected by the given vulnerability, specified as a Common Platform Enumeration (CPE) string. CPE uniquely represents hardware or software product by its vendor, name, version, edition and other details. For example, CPE version 2.3, which is used in NVD, allows specification of the following parameters: *part, vendor, product, version, update, edition, language, software edition, target software, target hardware* and *other* [71]. As outlined in Section 4.4, we can focus the model by only predicting vulnerabilities in the top N most popular software packages. In this project we set $N = 50$ and identify the top N most 'popular' software packages based on the number of vulnerabilities, which affect a given software package. Under 'software package' we understand all configurations, that share common *vendor* and *product*. For example, following configurations captured as CPE strings would all be considered as belonging to a single software package, as they all share vendor and product name (highlighted in bold):

- `cpe:2.3:o:`**`microsoft:windows_2003_server`**`:64-bit:*:*:*:*:*:*:*`
- `cpe:2.3:o:`**`microsoft:windows_2003_server`**`:itanium:*:*:*:*:*:*:*`
- `cpe:2.3:o:`**`microsoft:windows_2003_server`**`:r2:*:*:*:*:*:*:*`
- `cpe:2.3:o:`**`microsoft:windows_2003_server`**`:sp1:*:*:*:*:*:*:*`
- `cpe:2.3:o:`**`microsoft:windows_2003_server`**`:sp1:*:itanium:*:*:`

In some experiments, the definition of 'software package' is adjusted to include *vendor*, *product* and *version*.

From the CrimeBB dataset and the NVD data we create training rows for the model. We assume that any post that was published on the web in a time window W before a given vulnerability was listed in the NVD database could potentially include relevant novel information about that vulnerability. The optimal size of the window W must be determined experimentally.

Each training row contains one or more posts and a ground truth label. The ground truth label is a sparse binary vector with N dimensions. It is constructed for each vulnerability by examining, which of the top N software packages are affected by that vulnerability. If a software package is affected, this is indicated by 1 in the binary vector. This is similar to one-hot encoding vectors used in a single class classification problems.

The posts in each training row are selected at random from all posts that were published within the time window W. This time window is set experimentally, but is generally several days or weeks. The number of posts in each day of the window is also set experimentally, but remains constant for all rows of a training set.

Text Embedding The text embeddings of the posts in the training row are calculated when the training rows are being constructed. A *bert-as-service*[35] python implementation of the popular BERT model is used to calculate word embeddings for the posts. The *bert-as-service* server is running in 4 threads on the CLAAUDIA Linux server and serves requests issued by the script responsible for generating training rows. A pre-trained BERT-base uncased model, released by Google[36] is used by the *bert-as-service* server. A default configuration, which produces sentence-level embeddings is overridden, so instead word-level embeddings with 768 dimensions[37] are obtained.

As word embeddings in BERT are context-dependent, entire sentences are processed simultaneously by *bert-as-service*. The maximum number of words in a sentence can be configured at the server start-up as `max_length`, but may not be exceeded or adjusted during operation. When operating in word embedding mode, *bert-as-service* always returns embeddings with dimensions (`max_length`, 768). If embeddings for a short sentence are to be calculated, the sentence is padded until `max_length` is reached; sentences longer than `max_length` are split into smaller parts. The current model implementation requires that all model rows have the same size in every dimension. To satisfy this requirement we must normalise the number of sentences in each post to `max_sentences`. Posts that are shorter than `max_sentences` are padded with zeros, while posts longer than `max_sentences` are trimmed.

For the posts in the CrimeBB dataset we have experimentally established values of these two parameters to `max_length` = 25 and `max_sentences` = 15 as a suitable compromise between extensive padding (if either parameter is set too high) and losing information due to trimming (if either parameter is set too low).

Storing Training Data The generated data are stored on disk to allow transfer between servers and testing different model configurations on the same dataset. While only the post embedding vectors and the binary label vectors are necessary for the model training, original post in text form, as well as raw textual form of the affected software packages are also saved together with the training data to allow eventual debugging and investigation.

Due to the size of the post embeddings, they are saved in a binary *.npy* format, native to the popular Python mathematics library Numpy which is used to represent the embeddings during runtime. Disk location of these files, as well as post texts, affected software packages and binary target vectors are then saved in separate CSV file. For model training, data is transported to the AI Cloud machine, where it can be accessed rapidly from the optimised RAID storage.

[35]`https://github.com/hanxiao/bert-as-service`, accessed 29 June 2020, *archived*

[36]`https://github.com/google-research/bert`, accessed 29 June 2020

[37]The number of embedding dimensions is a feature of the selected pre-trained model and cannot be configured.

5.3.4 Neural Network Implementation

The neural network model is implemented in Python, using Keras functional API[38]. This API runs atop the TensforFlow machine learning library[39]. Keras is used in this project due to its simple way of defining and connecting layers, sufficient configuration options for the task at hand and overall popularity in the machine learning community.

The three conceptual sub models specified in Section 4.4.2 are implemented in a single Keras model. Sequential processing of multiple parts of the input by the same set of weights is necessary in two parts of the model: (i) each post must be processed individually by the Post Sub Model, and (ii) each day must be processed individually by the Day Sub Model. To achieve this, *shared layers* are introduced to the model. Before passed to a shared layer, the data stream is divided into multiple sub-streams, representing the individual posts/days, which need to be processed separately. In a shared layer, these individual sub-streams are processed iteratively, using the same set of neurons and weights. After a shared layer, the sub-streams are joined by concatenation before further processing.

[38]`https://keras.io/guides/functional_api/`, accessed 29 June 2020
[39]`https://www.tensorflow.org/`, accessed 29 June 2020

	Name	**Type**	**Shared**	**Hyperparameters**
Post Sub M.	`post_highlights_convolution`	Convolution layer	Yes	# filters Kernel size Stride size Activation function
	`shared_post_maxpool`	Max pooling layer	Yes	Pool size Stride size
Day Sub M.	`day_highlights_convolution`	Convolution layer	Yes	# filters Kernel size Stride size Activation function
	`shared_day_maxpool`	Max pooling layer	Yes	Pool size Stride size
Fortnight Sub M.	`LSTM`	LSTM layer	No	# units
	`Dense`	Output layer	No	# units Activation function

Table 6: Trainable layers of the model implemented in Keras and their hyperparameters.

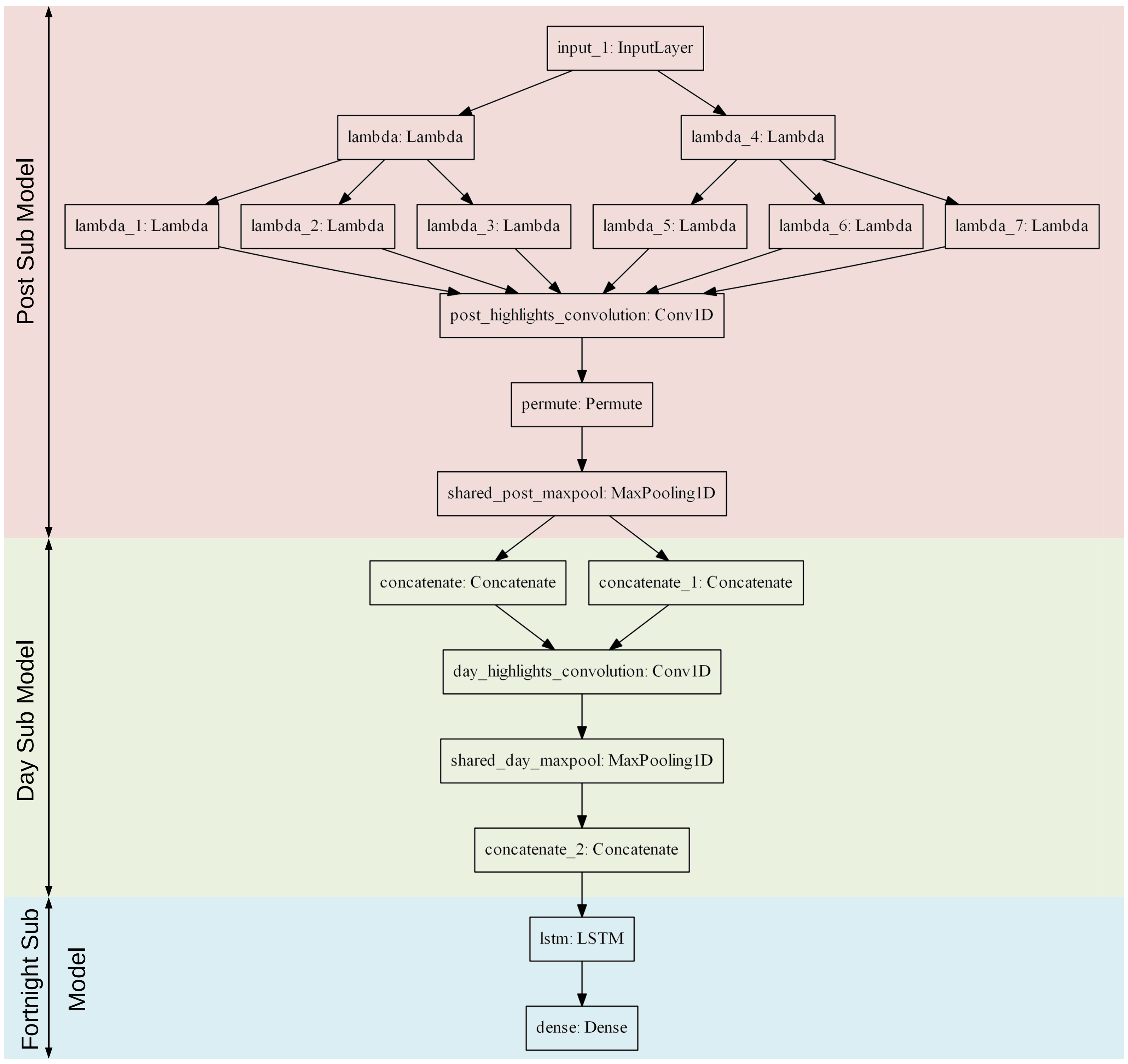


Figure 14: Full neural network implemented in Keras. Colours denote the conceptual part of the model. The branching in Labda layers represents posts/days, which are processed individually in the shared layers. This diagram illustrates input with two days and and three posts per day in order to fit on screen. In the real model implementation, the number of days and number of posts per day would be much higher.

Figure 14 depicts all layers of the model implemented in Keras. An overview of the trainable layers is provided in Table 6. The other layers present in the model, such as the *input*, *permute*, *concatenate* or *lambda* layers help shape and manipulate the flow of data through the model, but are not trainable and are omitted from the overview.

Both convolutional layers are Keras `Conv1D` layers that convolve over the input in a single dimension. The number of filters (# filters) hyperparameter determines the number of concurrent filters that are convolved over the data. The kernel size determines the size of each of these filters, while the stride size determines how much is the filter shifted in each convolution step.

The Max Pooling layers (`MaxPool1D` in Keras) reduce the space by taking the maximum value from a sliding window (pool). Pool size parameter determines the size of this window, while stride size determines how much is this window shifted in each step.

The LSTM layer is a Keras implementation of recurrent neural network architecture proposed in [72]. Number of units, which determines the dimensionality of the output space, is a mandatory hyperparameter that must be specified by the user. The internal activation function and recurrent activation function are kept on default values.

The final Dense layer determines the prediction output of the model. The number of units in this layer must be equal to the number of dimensions N of the binary target vector. Further hyperparameters that apply to the entire model are optimisation function, learning rate and loss function.

During training and evaluation, the data is provided to the model using the Tensorflow Dataset class, which loads data lazily from disk as needed and thus can support large datasets which do not fit into memory. The class also supports batching and pre-loading data ahead of time. Data stored on disk in CSV and NPY files are provided to the Dataset class via a generator function. During training, the model weights are periodically saved to disk to allow further analysis.

Selecting and sourcing data from primary sources and datasets is described in the beginning of this chapter. Together with the implementation details of the six trainable model layers and associated hyperparameters, the system is prepared for training and evaluation in the next chapter.

6 Experiments and Results

Using the data creation pipeline and the model implementation described in the previous chapter we conduct several experiments. By varying parameters of the data creation pipeline and the model hyperparameters we aim to confirm the initial hypothesis that signals from the web can predict occurrence of a new software vulnerabilities.

To measure the outcome of our experiments we use standardised metrics. Accuracy, precision and recall are among the most popular metrics for evaluating performance of a supervised model. In our case, the accuracy metric may be misleading, since the ground truth vector is sparse and a non-performing classifier, which always predicts 0 would achieve high levels of accuracy. We primarily focus on measuring recall throughout the experiments, as it is important that the model identifies as many potential new vulnerabilities as possible. Recall is preferred over precision, since missing a potential vulnerability has worse consequences than generating a false alarm by mis-classifying non-malignant signals as a vulnerability.

6.1 Experiment Methodology

The experiments are carried out in iterations with four steps: (i) Data Generation, (ii) Model Training, (iii) Model Evaluation, and (iv) Analysis of results. In each new iteration a new idea or approach is tried out that may improve the model performance and is typically based on the findings of the previous iteration.

6.1.1 Data Generation

In stage (i) the parameters of data creation are adjusted, such as the number of vulnerabilities in the dataset, number of rows per vulnerability and number of posts per training row. As the CrimeBB dataset is several orders of magnitude larger than the other available structured data, it is exclusively used to generate training data in all iterations. However, the type and amount of data from this dataset that is included in the training data is adjusted in each iteration.

As the Data Generation process relies heavily on disk I/O operations and calculation of word embeddings, and is carried out on a Compute Cloud server (not the more optimised AI Cloud), it takes time in the range of several hours hours for a dataset of several thousand rows to be generated. For this reason, this stage is intertwined with stage (ii) in some iterations. As the dataset is written to disk sequentially, it is possible to proceed to stage (ii) and start the initial model training on a fraction of the complete dataset, while more rows are still being generated.

6.1.2 Model Training

During stage (ii) the training data is transferred to AI Cloud and the model is trained on this data. Hyperparameters may be adjusted between iterations, or between subsequent runs within a single iteration. The model may be trained several times in a single iteration, to test out several configurations and dataset sizes. The model training is carried out in batches[40] and the trainable model parameters are adjusted after each batch. Usually, the model is trained for several epochs[41] and the model weights and configuration are saved after the last epoch.

6.1.3 Model Evaluation

Model Evaluation is carried out to understand the performance of the model and provide basis for the analysis of the results. As the performance of the model is reported continuously during training in Keras, this stage can sometimes be integrated in stage (ii). In machine learning practice it is generally not acceptable to use the training data to evaluate the model, however in case of a non-performing model, this is clearly demonstrated already during training and it is not necessary to conduct a formal evaluation using a test dataset. In those cases the model evaluation stage is skipped and the iteration proceeds directly to stage (iv).

6.1.4 Analysis of Results

Based on the performance of the model, the results are analysed and the iteration is reviewed for mistakes that could influence the results. The possible factors contributing to the model performance are considered and the next iteration is planned accordingly.

In total, four complete iterations and one iteration without phases (iii) and (iv) are completed throughout the course of the project. The following sections describe these iterations in closer detail.

6.2 Experiment iterations

The experiment process starts before the first iteration by implementing, debugging and optimising the model with sample data on a local machine. The optimisations include configuring the model to run with GPU support and adjusting the data loading process from in-memory pre-loading to lazy generator-based loading, where the data is loaded from the disk as needed. This prepares the data pipeline and the model for the first iteration.

6.2.1 Iteration 1: Baseline

The first iteration verifies that the model can be trained and evaluated using the input data. This iteration also serves as a baseline for further improvements of the model. The overview of the model parameters can be found in Table 7.

In this iteration the selection of certain parameters is based on good performance on similar tasks. The ReLU activation function is selected in the convolutional layers due to its

[40]One batch is several data rows grouped together.
[41]An epoch is a single pass through the entire dataset.

popularity and recommendations by several practitioners [73–75]. The Adam optimiser has also been recommended as an well-performing overall [76, 77]. The choice of the activation function in the last layer and the loss function was based on the need of multi-label classification, which is not possible with all activation and loss functions. The combination of Sigmoid activation function and Binary cross-entropy was recommended as suitable combination for this task [78–80].

The rest of the parameters were estimated based on the nature of the data and on the values of similar parameters in other projects, mainly in [42].

Results Already during training of the model with these parameters it can be seen that the recall measure does not raise above zero. It is likely that the high value of accuracy (>0.9) is due to the model classifying every example as negative. Model Evaluation is carried out that confirms this.

6.2.2 Iteration 2: Parameter tuning

We speculate that the poor performance of the model may be caused by mis-configuration of model's hyperparameters. In the Iteration 2 we therefore gradually adjust several of the hyperparameters and observe the performance during training. Changes include replacing ReLU with a similar variation Leaky ReLU in order to avoid the potential vanishing gradient problem, increasing the number of units in the LSTM layer and adjusting filter number and kernel sizes in the convolution layers. The size of the dataset is also increased to include more vulnerabilities. Full configuration can be found in Table 10 in Appendix D.1.

Results While in this configuration the Recall occasionally reaches values of around 0.08, this does still not indicate that the model successfully learned anything from the data.

6.2.3 Iteration 3: Increased Time Window

As several modifications of hyperparameters do not help to improve the model performance, we focus next on the data generation process. The posts that should indicate presence of a new vulnerability are picked at random from window of W days before the vulnerability was added to the NVD. We assume that the time window of $W = 7$ is too short for the vulnerability signals to manifest sufficiently in the posts. In the Iteration 3 we therefore increase the size of this window and the total number of posts per training row. The larger sample size should help the network discover medium- to long-term trends in the data. The full configuration can be found in Table 11 in Appendix D.2.

Results Unfortunately, increasing the time window W and the number of posts in each training row does not increase the recall value above zero. This result is replicated with several re-created datasets and with even further increased number of posts per day.

Parameter	**Value**
Data Generation	
# vulnerabilities	15
# rows per vulnerability	20
Window size W	7 days
# posts per day	3
Target vector length N	50
Maximum sentence length (`max_length`)	25
Maximum # sentences per post (`max_sentences`)	15
Data from year	2016
Model hyperparameters	
`post_highlights_convolution` # filters	40
`post_highlights_convolution` Kernel	4
`post_highlights_convolution` Stride size	1
`post_highlights_convolution` Activation function	ReLU
`shared_post_maxpool` Pool size	10
`shared_post_maxpool` Stride size	5
`day_highlights_convolution` # filters	40
`day_highlights_convolution` Kernel	1
`day_highlights_convolution` Stride size	1
`day_highlights_convolution` Activation function	ReLU
`shared_day_maxpool` Pool size	5
`shared_day_maxpool` Stride size	3
LSTM # units	50
Output layer activation function	Sigmoid
Optimiser	Adam
Learning rate	0.01
Loss Function	Binary Cross-entropy
Notes	
First iteration to validate the model and serve as a baseline for improvements.	

Table 7: Overview of data creation and model parameters in experiment Iteration 1.

6.2.4 Iteration 4: Pre-filtered Data

We further speculate that the poor performance of the model is caused by very noisy data. Manual screening of several dozen forum posts reveals that while the posts are generally focused on topics related to computers, very few posts focus specifically on vulnerabilities, security or hacking. We assume that by filtering out non-relevant posts we help the model pick up meaningful signals from the data.

Blacklist and whitelist keyword-based filters are considered as suitable approaches to filtering out noisy and non-relevant data. Blacklist-based filtering scans the post for a specified list of unwanted keywords and if a match is found, the post is discarded. Examples of unwanted keywords could be *carding*, *games*, *drugs* or *pharmacy*. Whitelist filtering scans the post for a list of desired keywords. If none of the desired keywords is found in the post, the post is discarded. The advantage of the blacklist approach, is the reduced risk of accidentally losing relevant data, which might occur if the list of desired keywords in the whitelist approach is not comprehensive enough. However, this also results in more noise in the dataset as it is challenging to encompass all potentially irrelevant terms.

In Iteration 4 we employ whitelist approach to reduce the non-relevant data as much as possible. We construct the list of desired words from the names of the top N affected software configurations, which form the target vectors in our dataset. We further extend the keyword list with the most informative words from the textual descriptions of vulnerabilities, which are a part of the NVD data. As these descriptions are written in standard English, they naturally contain many words with no information value (such as *computer*, *found*, *have*). These words are too generic and should not be included in the list of desired words. Term Frequency/Inverse Document Frequency (TF/IDF) is used to identify the most informative words from the vulnerability descriptions and remove terms which are too generic. The configuration names and the most informative terms identified by TF/IDF are then joined together into a list of desired words. Finally, all words shorter than 4 characters are removed from the list. This list forms the basis of the whitelist approach to produce a dataset, which contains fewer non-relevant, off-topic posts.

Results While a manual inspection confirmed both the validity of the list of desired keywords, and the increased relevance of the filtered dataset, this approach still failts to increase the recall value above zero.

6.2.5 Iteration 5: Adjusted Classification Threshold

Experiments in the previous iterations demonstrated that the model is not sensitive to changes in data creation pipeline or changes in hyper-parameters. Since in a typical model changes in these parameters should measurably manifest in model performance, there may be a previously undiscovered mis-configuration in the model that prevents the model from arriving at useful predictions.

The analysis of the model in the previous iterations focused on comparison of individual parameters to similar models or best practice in the field. In Iteration 5 the focus is shifted towards closely examining the performance of the model at hand with existing and artificially generated data. It is discovered that regardless of the input data, the neurons in the output layer achieve very low levels of activation, approximately between 0.0001 and 0.1. The default Keras implementations of Binary Accuracy and Recall metrics use threshold of 0.5. Due to the low activation values of the output layer neurons, these are never classified as *Positive* using the default threshold. Decreasing the classification threshold to 0.1 or 0.05 resolves this problem and immediately raises model performance in the recall (and precision) metric.

This experiment iteration has not been fully completed during the course of the project, but preliminary evaluation results on a small number of previously unseen data indicate the Area under the ROC Curve ≈ 0.77, which suggests that a well-performing threshold can be found. Formal verification of the data generation and model parameters, as well as full model evaluation using previously unseen data is necessary to confirm the preliminary result. However, provided that the results can be replicated, experiments from previous iterations can be repeated and model behaviour can be examined under the new findings.

While the four complete experiment iterations fail to train the model, they provide a solid basis for further experiments. Potentially resolving a model mis-configuration in the fifth unfinished iteration paves the way to recreating the previous experiments with new findings to arrive at a reasonably performing model.

7 Conclusion

During this project, we attempted to closer examine the hypothesis that monitoring online sources can be used to detect fresh security vulnerabilities. To prove or disprove this hypothesis we implemented a neural network model and trained it with historical data.

We established preliminaries and introduced some of the relevant data sources of Open-source Intelligence. We examined similar works and systems that gather, analyse or make predictions on cybersecurity data. Some of these works make their datasets available to the public, making them potentially useful for training our model.

In Chapter 4 we modelled a hypothetical end-to-end system to help us understand the requirements of vulnerability prediction. For the purposes of this system, we defined a *fresh* vulnerability using the vulnerability lifecycle model. We devised the high-level architecture of a supervised vulnerability detection model and defined how training data for this model should be represented.

In Chapter 5 we defined parameters for the training data and shortlisted suitable data sources according to these parameters. We collected data from these sources using automated data collection tools and APIs. Concurrently, we sourced several comprehensive datasets that could serve as training data for our model. Due to the amount and quality of the data, we selected the CrimeBB dataset [52] to be used as the primary source of the training data in our experiments

We implemented a data creation pipeline to generate training data for our model and implemented a model with six trainable layers in Keras. We adjusted the pipeline and the model in four experiment iterations but did not manage to obtain satisfactory results. In an unfinished fifth iteration we discovered a possible reason for the model non-performance and obtained promising preliminary results.

However, based on these experiments, we can not conclusively prove or disprove our initial hypothesis. We speculate that adjusting the classification threshold immediately operationalises the model. However, the quality of the underlying data still poses a challenge to further improving model performance. While the posts in the CrimeBB dataset are generally related to computing, they may not be directly useful for vulnerability prediction. The likely reason for this is the relative niche of forums frequented by professional hackers. It is in the interest of these forums to stay hidden, which is demonstrated by strong anti-crawl measures they deploy, as well as the rate in which they appear and disappear. Most of the hacker forums obtained during literature review from publications not older than 5 years could not be reached anymore. This makes creating a reliable dataset with sufficient training material extremely difficult.

Large-scale efforts that swipe across a significant portion of the available online sources, like the CrimeBB dataset do capture the scarce 'professional hacker' insights, but this useful data is still diluted in such datasets due to large amounts of other semi- or non-relevant material.

Future Work Based on the promising results in Iteration 5 the future work could verify the validity and performance of the configuration with an adjusted threshold. It would be desired to retrain the model using a larger amount of data and repeat the parameter fine-tuning based on the performance.

To increase the versatility of the model, data from sources other than only dark- and deep-web forums should be included in the training dataset. For some sources with good data collection infrastructure, such as Twitter, Reddit or Pastebin, building a dataset of sufficient size is straightforward and only limited by time and cost.

Exploring different options of filtering out irrelevant data might further help to increase the model performance. To improve the utility of the model to the final user, increasing the number of included software configurations (the dimensionality of target vector N) should be explored, as well as increasing the granularity of the predictions from *vendor* and *product* to *vendor*, *product* and *version*.

Provided a well-performing model can be developed and the system is deployed in practice, a malicious party may publish fake posts online, mentioning a made-up vulnerability, thus making the system issue false alarms to the operator. Resilience against such attacks might need to be researched in the future.

References

[1] Lillian Ablon and Andy Bogart. “Zero days, thousands of nights”. In: *RAND Corporation, Santa Monica, CA* (2017).

[2] Anna Sapienza et al. “Early warnings of cyber threats in online discussions”. In: *IEEE International Conference on Data Mining Workshops, ICDMW*. Vol. 2017-November. New Orleans, LA, USA: IEEE Computer Society, Dec. 2017, pp. 667–674. ISBN: 9781538614808. DOI: 10.1109/ICDMW.2017.94.

[3] Eric Nunes et al. “Darknet and deepnet mining for proactive cybersecurity threat intelligence”. In: *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data, ISI 2016*. Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 7–12. ISBN: 9781509038657. DOI: 10.1109/ISI.2016.7745435.

[4] Palash Goyal et al. “Discovering Signals from Web Sources to Predict Cyber Attacks”. In: (June 2018). URL: http://arxiv.org/abs/1806.03342.

[5] R Shirey. *Internet Security Glossary, Version 2*. Tech. rep. 4949. RFC Editor, Aug. 2007. URL: http://www.rfc-editor.org/rfc/rfc4949.txt.

[6] Inc. FIRST.org. *CVSS v3.1 Specification Document*. 2019. URL: https://www.first.org/cvss/v3.1/specification-document.

[7] Ashish Arora et al. “An Empirical Analysis of Software Vendors’ Patch Release Behavior: Impact of Vulnerability Disclosure”. eng. In: *Information Systems Research* 21.1 (2010), pp. 115–132. DOI: 10.1287/isre.1080.0226.

[8] H C Joh et al. *A Framework for Software Security Risk Evaluation using the Vulnerability Lifecycle and CVSS Metrics*. Tech. rep. Fort Collins, CO USA: Colorado State University, 2010. URL: http://www.honeynor.no/research/time2exploit/.

[9] Jeb Su. “Why Zerodium Will Pay $2.5 Million For Anyone Who Can Hack Android But Only $2 Million For An iPhone”. In: *Forbes.com* (Sept. 2019). URL: https://www.forbes.com/sites/jeanbaptiste/2019/09/04/why-zerodium-will-pay-2-5-million-for-anyone-who-can-hack-android-but-only-2-million-for-an-iphone/#17f3d432716b.

[10] Sudip Mittal et al. “CyberTwitter: Using Twitter to generate alerts for cybersecurity threats and vulnerabilities”. In: *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016*. Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 860–867. ISBN: 9781509028467. DOI: 10.1109/ASONAM.2016.7752338.

[11] Su Zhang, Xinming Ou, and Doina Caragea. “Predicting Cyber Risks through National Vulnerability Database”. In: *Information Security Journal: A Global Perspective* 24.4-6 (Dec. 2015), pp. 194–206. ISSN: 1939-3555. DOI: 10.1080/19393555.2015.1111961. URL: http://www.tandfonline.com/doi/full/10.1080/19393555.2015.1111961.

[12] Mike Isaac and Sydney Ember. *For Election Day Influence, Twitter Ruled Social Media.* New York, Nov. 2016. URL: https://www.nytimes.com/2016/11/09/technology/for-election-day-chatter-twitter-ruled-social-media.html.

[13] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. "Earthquake Shakes Twitter Users: Real-Time Event Detection by Social Sensors". In: *Proceedings of the 19th International Conference on World Wide Web.* WWW '10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 851–860. ISBN: 9781605587998. DOI: 10.1145/1772690.1772777. URL: https://doi.org/10.1145/1772690.1772777.

[14] Onook Oh, Manish Agrawal, and H. Raghav Rao. "Information control and terrorism: Tracking the Mumbai terrorist attack through twitter". In: *Information Systems Frontiers* 13.1 (Mar. 2011), pp. 33–43. ISSN: 13873326. DOI: 10.1007/s10796-010-9275-8.

[15] Devin Gaffney and J. Nathan Matias. "Caveat emptor, computational social science: Large-scale missing data in a widely-published reddit corpus". In: *PLoS ONE* 13.7 (July 2018). ISSN: 19326203. DOI: 10.1371/journal.pone.0200162.

[16] Adi Robertson. *Reddit has broadened its anti-harassment rules and banned a major incel forum.* Sept. 2019. URL: https://www.theverge.com/2019/9/30/20891920/reddit-harassment-bullying-threats-new-policy-change-rules-subreddits.

[17] Dennis Mirante and Justin Cappos. *Understanding password database compromises.* Tech. rep. New York: Dept. of Computer Science and Engineering Polytechnic Inst. of NYU, 2013.

[18] Srdjan Matic et al. "Peering into the muddy waters of pastebin". In: *ERCIM News: Special Theme Cybercrime and Privacy Issues* (2012), p. 16.

[19] Michael Chertoff and Tobby Simon. *The Impact of the Dark Web on Internet Governance and Cyber Security.* Tech. rep. Global Commission on Internet Governance, Feb. 2015.

[20] David Lacey and Paul M. Salmon. "It's dark in there: Using systems analysis to investigate trust and engagement in dark web forums". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics).* Vol. 9174. Springer Verlag, 2015, pp. 117–128. ISBN: 9783319203720. DOI: 10.1007/978-3-319-20373-7{_}12.

[21] Jana Shakarian, Andrew T. Gunn, and Paulo Shakarian. "Exploring malicious hacker forums". In: *Cyber Deception: Building the Scientific Foundation.* Springer International Publishing, Jan. 2016, pp. 259–282. ISBN: 9783319326993. DOI: 10.1007/978-3-319-32699-3{_}11.

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router.* Tech. rep. Naval Research Lab Washington DC, 2004.

[23] Matthias Schafer et al. "BlackWidow: Monitoring the Dark Web for Cyber Security Information". In: *International Conference on Cyber Conflict, CYCON.* Vol. 2019-May. NATO CCD COE Publications, May 2019. ISBN: 9789949990443. DOI: 10.23919/CYCON.2019.8756845.

[24] George Kadianakis and Karsten Loesing. "Extrapolating network totals from hidden-service statistics". In: *Research rep* (2015), p. 1.

[25] Virgil Griffith, Yang Xu, and Carlo Ratti. "Graph Theoretic Properties of the Darkweb". In: (Apr. 2017). URL: http://arxiv.org/abs/1704.07525.

[26] John Robertson et al. *Darkweb Cyber Threat Intelligence Mining*. New York, NY, USA: Cambridge University Press, 2017. ISBN: 1107185777.

[27] Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing*. Chapman and Hall/CRC, 2010.

[28] C. Silva and B. Ribeiro. "The importance of stop word removal on recall values in text categorization". In: *Proceedings of the International Joint Conference on Neural Networks, 2003.* Vol. 3. Portland, OR, USA: IEEE, July 2003, pp. 1661–1666. ISBN: 0-7803-7898-9. DOI: 10.1109/IJCNN.2003.1223656. URL: http://ieeexplore.ieee.org/document/1223656/.

[29] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[30] Guillaume Lample et al. "Neural architectures for named entity recognition". In: *arXiv preprint* (2016).

[31] David M Blei, Andrew Y Ng, and Michael I Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[32] Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3 (Dec. 2015), pp. 211–225. ISSN: 2307-387X. DOI: 10.1162/tacl{_}a{_}00134.

[33] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: (Jan. 2013). URL: http://arxiv.org/abs/1301.3781.

[34] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (May 2019). URL: http://arxiv.org/abs/1810.04805.

[35] Jieh-Sheng Lee and Jieh Hsiang. "PatentBERT: Patent Classification with Fine-Tuning a pre-trained BERT Model". In: (May 2019). URL: http://arxiv.org/abs/1906.02124.

[36] Ashutosh Adhikari et al. "DocBERT: BERT for Document Classification". In: (Apr. 2019). URL: http://arxiv.org/abs/1904.08398.

[37] Iz Beltagy, Kyle Lo, and Arman Cohan. "SciBERT: A Pretrained Language Model for Scientific Text". In: (Mar. 2019). URL: http://arxiv.org/abs/1903.10676.

[38] Andrew M. Dai, Christopher Olah, and Quoc V. Le. "Document Embedding with Paragraph Vectors". In: (July 2015). URL: http://arxiv.org/abs/1507.07998.

[39] Jey Han Lau and Timothy Baldwin. "An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation". In: (July 2016), pp. 78–86. URL: http://arxiv.org/abs/1607.05368.

[40] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to Machine Learning*. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-15728-9. DOI: 10.1007/978-3-030-15729-6. URL: http://link.springer.com/10.1007/978-3-030-15729-6.

[41] Ethem Alpaydin. *Introduction to machine learning*. 2nd ed. Cambridge, Massachusetts: MIT press, 2010. ISBN: 978-0-262-01243-0.

[42] Zhongqing Wang and Yue Zhang. "DDoS Event Forecasting Using Twitter Data". In: *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. IJCAI'17. AAAI Press, 2017, pp. 4151–4157. ISBN: 9780999241103.

[43] Miroslav Kubat. *An Introduction to Machine Learning*. Springer International Publishing, Sept. 2017, pp. 1–348. ISBN: 9783319639130. DOI: 10.1007/978-3-319-63913-0.

[44] J. A.K. Suykens and J. Vandewalle. "Least squares support vector machine classifiers". In: *Neural Processing Letters* 9.3 (1999), pp. 293–300. ISSN: 13704621. DOI: 10.1023/A:1018628609742.

[45] Filip Adamik. *Improving Search in Health Forums with a Neural Network Internship Report*. Tech. rep. Copenhagen: Aalborg University Copenhagen, 2019.

[46] Felix A. Gers, Jurgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: *IEE Conference Publication*. Vol. 2. 470. IEE, 1999, pp. 850–855. ISBN: 0852967217. DOI: 10.1049/cp:19991218.

[47] Jonathan D Cryer and Kung-Sik Chan. *Time Series Analysis: With Applications in R*. New York, NY: Springer, 2008, p. 487. ISBN: 9780387759586. DOI: 10.1007/978-0-387-75959-3. URL: https://link.springer.com/book/10.1007/978-0-387-75959-3.

[48] Nathanael A Heckert et al. *Handbook 151: NIST/SEMATECH e-Handbook of Statistical Methods*. Tech. rep. NIST/SEMATECH, 2002. URL: ttp://www.itl.nist.gov/div898/handbook/.

[49] Hengqing Tong, T. Krishna Kumar, and Yangxin Huang. *Developing Econometrics*. John Wiley and Sons, Oct. 2011. ISBN: 9780470681770. DOI: 10.1002/9781119954231.

[50] Shalini Ghosh et al. "ATOL: A framework for automated analysis and categorization of the Darkweb Ecosystem". In: *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*. 2017. URL: www.aaai.org.

[51] Tianjun Fu, Ahmed Abbasi, and Hsinchun Chen. "A focused crawler for Dark Web forums". In: *Journal of the American Society for Information Science and Technology* 61.6 (June 2010), n/a–n/a. ISSN: 15322882. DOI: 10.1002/asi.21323. URL: http://doi.wiley.com/10.1002/asi.21323.

[52] Sergio Pastrana et al. "CrimeBB: Enabling Cybercrime Research on Underground Forums at Scale". In: *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*. New York, New York, USA: Association for Computing Machinery (ACM), Apr. 2018, pp. 1845–1854. DOI: 10.1145/3178876.3186178. URL: http://dl.acm.org/citation.cfm?doid=3178876.3186178.

[53] Sagar Samtani et al. "AZSecure Hacker Assets Portal: Cyber threat intelligence and malware analysis". In: *IEEE International Conference on Intelligence and Security Informatics: Cybersecurity and Big Data, ISI 2016*. Tucson, AZ, USA: Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 19–24. ISBN: 9781509038657. DOI: 10.1109/ISI.2016.7745437.

[54] Po Yi Du et al. "Identifying, collecting, and presenting hacker community data: Forums, IRC, carding shops, and DNMs". In: *2018 IEEE International Conference on Intelligence and Security Informatics, ISI 2018*. Institute of Electrical and Electronics Engineers Inc., Dec. 2018, pp. 70–75. ISBN: 9781538678480. DOI: 10.1109/ISI.2018.8587327.

[55] Gwern Branwen et al. *Dark Net Market archives, 2011-2015*. July 2015. URL: https://www.gwern.net/DNM-archives.

[56] P Maciołek and G Dobrowolski. "Cluo: web-scale text mining system for open source intelligence purposes". English. In: *Computer Science* Vol. 14.1 (2013), pp. 45–62.

[57] Elvis Pontes et al. "Applying multi-correlation for improving forecasting in cyber security". In: *2011 6th International Conference on Digital Information Management, ICDIM 2011*. Melbourn, QLD, Australia: IEEE, Sept. 2011, pp. 179–186. ISBN: 9781457715389. DOI: 10.1109/ICDIM.2011.6093323.

[58] Yang Liu et al. "Cloudy with a Chance of Breach: Forecasting Cyber Security Incidents". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 1009–1024. ISBN: 978-1-939133-11-3. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/liu.

[59] M. Lisa Mathews et al. "A collaborative approach to situational awareness for cybersecurity". In: *CollaborateCom 2012 - Proceedings of the 8th International Conference on Collaborative Computing: Networking, Applications and Worksharing*. Pittsburgh, PA, USA: IEEE, Nov. 2012, pp. 216–222. ISBN: 9781936968367. DOI: 10.4108/icst.collaboratecom.2012.250794.

[60] Sagar Samtani, Ryan Chinn, and Hsinchun Chen. "Exploring hacker assets in underground forums". In: *2015 IEEE International Conference on Intelligence and Security Informatics: Securing the World through an Alignment of Technology, Intelligence, Humans and Organizations, ISI 2015*. Baltimore, MD, USA: Institute of Electrical and Electronics Engineers Inc., July 2015, pp. 31–36. ISBN: 9781479998883. DOI: 10.1109/ISI.2015.7165935.

[61] Nazgol Tavabi et al. “Darkembed: Exploit prediction with neural language models”. In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

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

[63] Sehun KIM et al. “Hybrid Intrusion Forecasting Framework for Early Warning System”. In: *IEICE Transactions on Information and Systems* E91.D.5 (May 2008), pp. 1234–1241. ISSN: 0916-8532. DOI: 10.1093/ietisy/e91-d.5.1234.

[64] Keunsoo Lee et al. “DDoS attack detection method using cluster analysis”. In: *Expert Systems with Applications* 34.3 (Apr. 2008), pp. 1659–1665. ISSN: 09574174. DOI: 10.1016/j.eswa.2007.01.040.

[65] Ian Perry et al. “Differentiating and Predicting Cyberattack Behaviors Using LSTM”. In: *DSC 2018 - 2018 IEEE Conference on Dependable and Secure Computing*. Institute of Electrical and Electronics Engineers Inc., Jan. 2019. ISBN: 9781538657904. DOI: 10.1109/DESEC.2018.8625145.

[66] Bowen Baker et al. “Designing Neural Network Architectures using Reinforcement Learning”. In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (Nov. 2016). URL: http://arxiv.org/abs/1611.02167.

[67] Peter D. Turney and Patrick Pantel. “From Frequency to Meaning: Vector Space Models of Semantics”. In: *Journal of Artificial Intelligence Research* 37 (Mar. 2010), pp. 141–188. DOI: 10.1613/jair.2934. URL: http://arxiv.org/abs/1003.1141%20http://dx.doi.org/10.1613/jair.2934.

[68] Katrin Erk. “Vector Space Models of Word Meaning and Phrase Meaning: A Survey”. In: *Language and Linguistics Compass* 6.10 (2012), pp. 635–653. DOI: 10.1002/lnco.362. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/lnco.362.

[69] M. Baroni, G. Dinu, and G. Kruszewski. “Don’t count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors”. In: *52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014 - Proceedings of the Conference*. Vol. 1. 2014, pp. 238–247. ISBN: 9781937284725. DOI: 10.3115/v1/p14-1023.

[70] Yuji Roh, Geon Heo, and Steven Euijong Whang. “A Survey on Data Collection for Machine Learning: A Big Data - AI Integration Perspective”. In: *IEEE Transactions on Knowledge and Data Engineering* (Oct. 2019), pp. 1–1. ISSN: 1041-4347. DOI: 10.1109/tkde.2019.2946162.

[71] Brant A Cheikes, David Waltermire, and Karen Scarfone. *Common Platform Enumeration: Naming Specification Version 2.3*. Tech. rep. Gaithersburg, MD: National Institute of Standards and Technology, Aug. 2011. DOI: 10.6028/NIST.IR.7695. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7695.pdf.

[72] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 08997667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://www.mitpressjournals.org/doix/abs/10.1162/neco.1997.9.8.1735.

[73] Gabriel Pierobon. *Visualizing intermediate activation in Convolutional Neural Networks with Keras*. Nov. 2018. URL: https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0.

[74] cantordust. *neural networks - How to choose an activation function?* June 2018. URL: https://ai.stackexchange.com/questions/7088/how-to-choose-an-activation-function.

[75] missinglink.ai. *7 Types of Activation Functions in Neural Networks: How to Choose?* 2020. URL: https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/.

[76] Santanu_Pattanayak. *optimization - Guidelines for selecting an optimizer for training neural networks*. 2017. URL: https://datascience.stackexchange.com/questions/10523/guidelines-for-selecting-an-optimizer-for-training-neural-networks.

[77] Chengwei Zhang. *Quick Notes on How to choose Optimizer In Keras*. Mar. 2018. URL: https://www.dlology.com/blog/quick-notes-on-how-to-choose-optimizer-in-keras/.

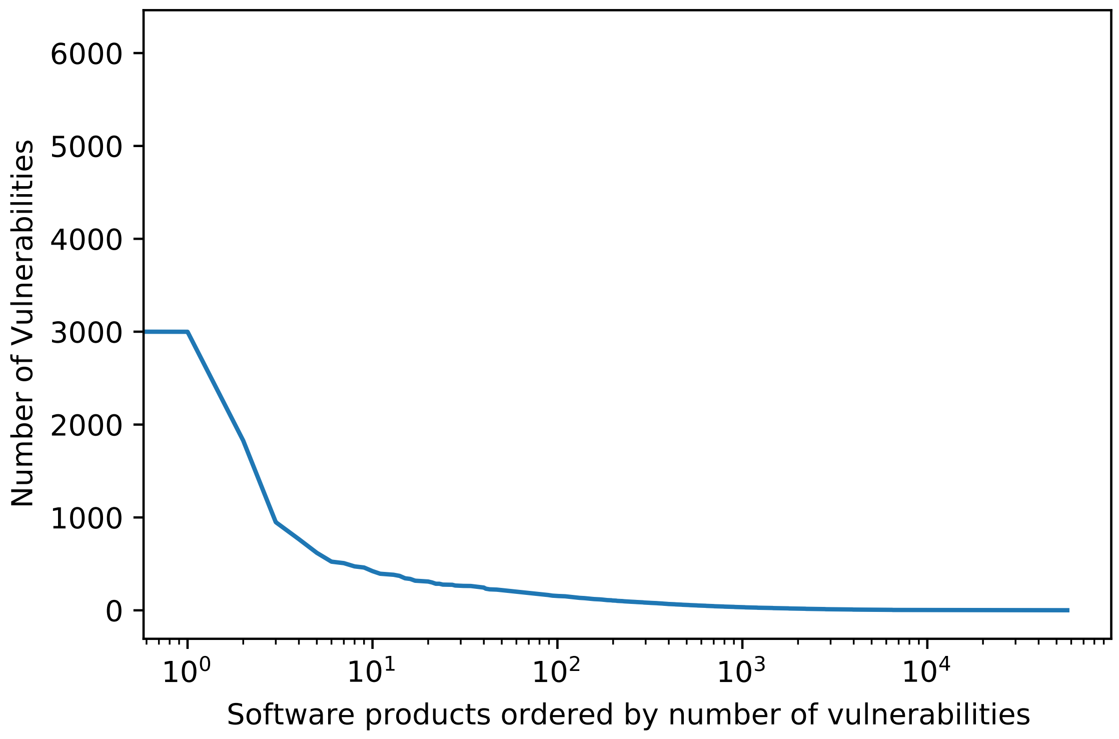
[78] Shiva Verma. *Multi-Label Image Classification with Neural Network | Keras*. Sept. 2019. URL: https://towardsdatascience.com/multi-label-image-classification-with-neural-network-keras-ddc1ab1afede.

[79] Raúl Gómez. *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*. May 2018. URL: https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[80] frankyjuang. *python - How does Keras handle multilabel classification?* May 2017. URL: https://stackoverflow.com/questions/44164749/how-does-keras-handle-multilabel-classification.

[81] *NVD - Data Feeds*. 2020. URL: https://nvd.nist.gov/vuln/data-feeds.

A Vulnerability distribution by software product



Distribution of vulnerabilities across software products. *Software product* is a particular family of software by a single vendor (e.g. Google Chrome). Different versions of the same software are grouped together (e.g. Google Chrome 61 and Google Chrome 62 are grouped under Google Chrome). Table 8 lists top software products with most vulnerabilities. Reported vulnerabilities between 2002 and 2019, based on data from NVD [81].

X-axis: Position of software product in number of vulnerabilities, relative to other software products (products with most vulnerabilities are on the left). Logarithmic scale.

Y-axis: Total number of vulnerabilities in the software product.

Software product	**Vulnerability Count**
cisco ios	6154
google chrome	2999
linux linux_kernel	1827
imagemagick imagemagick	949
cisco ios_xe	766
cisco nx-os	619
cisco adaptive_security_appliance_software	524
digium asterisk	508
btiteam xbtitracker	473
nginx nginx	461
ponsoftware explzh	422
phpmyadmin phpmyadmin	394
maradns maradns	387
moinejf abcm2ps	383
oracle mysql	371
apache tomcat	345
postgresql postgresql	338
google v8	319
debian dpkg	315
perl perl	314
samba samba	310
wordpress wordpress	300
typo3 typo3	286
adobe flash_player	286
gnu gnutls	277
libpng libpng	276
gitlab gitlab	276
php php	275
google chrome_os	267
ibm websphere_application_server	265
debian apt	264
asterisk open_source	263
cisco ios_transmission_control_protocol	263
git_for_windows_project git_for_windows	262

Table 8: Top 35 software products with most reported security vulnerabilities. Data from [81]

B List of forums

Forum Name	Forum location
Torum	`http://torum6uvof666pzw.onion/`

Torigon Cyber Security	`http://torigonsn3d63cldhr76mkfdzo`
	`3tndnl2tftiek55i2vilscufer6ryd.onion/`

CryptBB	`http://cryptbb2gezhohku.onion/`
Hack Forums	`https://hackforums.net/index.php/`
Offensive Community	`http://www.offensivecommunity.net/`
0day Forum	`http://mvfjfugdwgc5uwho.onion/`

Table 9: List of deep- and dark-web locations to be crawled.

C List of Pastebin search keywords

zeroday, 0day, exploit, vulnerability, hack, cve, java, json, python, security, patch, overflow, execution, root, privilege, escalation, javascript, unauthorised, unauthorized, rootkit

D Experiment Configurations

D.1 Iteration 2

Parameter	**Value**
Data Generation	
# vulnerabilities	**106**
# rows per vulnerability	20
Window size W	7 days
# posts per day	3
Ttarget vector length N	50
Maximum sentence length (`max_length`)	25
Maximum # sentences per post (`max_sentences`)	15
Data from year	2016
Model hyperparameters	
`post_highlights_convolution` # filters	**80**
`post_highlights_convolution` Kernel	**5**
`post_highlights_convolution` Stride size	1
`post_highlights_convolution` Activation function	**Leaky ReLU**
`shared_post_maxpool` Pool size	10
`shared_post_maxpool` Stride size	5
`day_highlights_convolution` # filters	**80**
`day_highlights_convolution` Kernel	**3**
`day_highlights_convolution` Stride size	1
`day_highlights_convolution` Activation function	**Leaky ReLU**
`shared_day_maxpool` Pool size	5
`shared_day_maxpool` Stride size	3
LSTM # units	**150**
Output layer activation function	Sigmoid
Optimiser	Adam
Learning rate	**0.1**
Loss Function	Binary cross-entropy
Notes	
Increase the size of the dataset to include more vulnerabilities and test out different adjustments to the model hyperparameters.	

Table 10: Overview of data creation and model parameters in experiment Iteration 2. Bold items highlight change from last iteration.

D.2 Iteration 3

Parameter	**Value**
Data Generation	
# vulnerabilities	106
# rows per vulnerability	20
Window size W	**21 days**
# posts per day	**5**
Target vector length N	50
Maximum sentence length (`max_length`)	25
Maximum # sentences per post (`max_sentences`)	15
Data from year	2016
Model hyperparameters	
`post_highlights_convolution` # filters	40
`post_highlights_convolution` Kernel	4
`post_highlights_convolution` Stride size	1
`post_highlights_convolution` Activation function	ReLU
`shared_post_maxpool` Pool size	10
`shared_post_maxpool` Stride size	5
`day_highlights_convolution` # filters	40
`day_highlights_convolution` Kernel	1
`day_highlights_convolution` Stride size	1
`day_highlights_convolution` Activation function	ReLU
`shared_day_maxpool` Pool size	5
`shared_day_maxpool` Stride size	3
LSTM # units	50
Output layer activation function	Sigmoid
Optimiser	Adam
Learning rate	0.01
Loss Function	Binary Cross-entropy
Notes	
By increasing the time window W and number of posts per day we include more vulnerabilities in each training row.	

Table 11: Overview of data creation and model parameters in experiment Iteration 3. Bold items highlight change from last iteration.

CVE-2018-????? A remote attacker can inject arbitrary text into public-facing pages via the comments box. xkcd.com/1957/