Masaryk University
Faculty of Informatics

Detecting Malicious URLs

Master's Thesis

Bc. Andrea Turiaková

Brno, Fall 2019

Masaryk University
Faculty of Informatics

Detecting Malicious URLs

Master's Thesis

Bc. Andrea Turiaková

Brno, Fall 2019

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Andrea Turiaková

Advisor: RNDr. Jan Sedmidubský, Ph.D.

Acknowledgements

Firstly, I would like to express my gratitude and sincere thanks to supervisor RNDr. Jan Sedmidubský, Ph.D. for his kind guidance, personal approach and useful comments for this thesis.

I would also like to thank my partner, for his endless support and motivation me to complete this work.

Last but not the least, I thank my family and friends who always supported me during my studies.

Abstract

Malicious URLs host various unsolicited content and can pose a high threat to potential victims. Therefore, a fast and efficient detection technique is needed. In this thesis, we focus on the problem of detecting malicious URLs based on the information obtained from URLs using machine-learning technologies. We proceeded in two steps.

First, we created and implemented a tool that extracts, processes, and store the URL features in such a way that machine-learning models can be trained on them. Second, we trained and evaluated models created with SVMlight and TensorFlow libraries on a real-life dataset of malicious and benign URLs, provided by ESET company. The efficiency and accuracy of the prediction of these models are measured and compared. The results are showing the ability and suitability of tested technologies to predict the probability of a URL being malicious.

Keywords

malicious, URL, detection, machine learning, SVM, TensorFlow, malware, classification, model

Contents

Introduction

Online services have become an irreplaceable part of today's businesses, schools, banking, or personal lives. With their increasing popularity, the number of malicious websites is growing. A malicious website contains some unsolicited content with a purpose to gather sensitive data or install malware onto a user's machine. Usually, some interaction from the user part is needed, but in the case of a drive-by download, malware is installed automatically without asking for permission.

Prevention from such attacks is complicated, as being cautious is sometimes not enough. Attackers can able to exploit vulnerabilities in web applications to insert malicious code without knowledge of the owner. According to the 2019 Webroot Threat Report [1], 40% of malicious URLs were found on good domains. On the first look, legitimate websites can become a potential threat for users. Browser and antiviruses re expected to prevent a user from accessing such websites.

The URL is usually the first and cheapest information we have about a website. Therefore it is a natural choice to develop techniques that would recognize a malicious URL from benign. Moreover, accessing and downloading the content of the website can be time-consuming and brings risks associated with downloading potentially harmful content.

The most common technique used to aim this problem is blacklisting. The list of 'bad URLs' is compiled, and the browser prevents the user from accessing them. The major problem with this technique is blacklist incompleteness, URL only appear there if it was already reported before. To solve this issue, we need a more proactive solution that can spot and match patterns in malicious URLs.

In this thesis, we focused on the problem of the detection of malicious URLs with machine learning techniques. We have chosen two libraries, SVM^{light} and TensorFlow, to train the prediction models. The goal is to determine if chosen algorithms can efficiently decide if the given URL on input is malicious or not. Based on their accuracy can be later used their estimated probability of URL being malicious as a relevant precondition for its further analysis.

Data samples for training the models and subsequent evaluation were provided by ESET, a security software company. Malicious samples contained the URLs, checked by their software, and detected as malware. In benign set were URLs that were evaluated as clean after the check.

The thesis is organized as follows: In chapter 2 we describe the term of malicious URL, its different types, and different features that can be extracted from it. Chapter 3 provides information about state-of-art techniques, with a focus on machine learning and needed features preprocessing. In chapter 4 we describe shortly proposed an approach, and then in chapter 5 we explain more in detail the design and implementation of processing and classifying URLs with selected libraries. The results of experimental evaluations are then presented in chapter 6.

1 Malicious URLs

URL is the abbreviation of Uniform Resource Locator, the global address of documents and other resources on the World Wide Web. In other words, it is a specific character string which constitutes a reference to an existing Internet resource.
A URL usually consist of three to five components:

scheme, identifies the protocol that should be used
host, specifies the IP address or registered domain name
path, identifies the location of a specific resource in the host, can be followed by port number
query string, a string of parameters and values, part of the URL coming after '?'
fragment, identification of a secondary resource, usually a subsection within the page

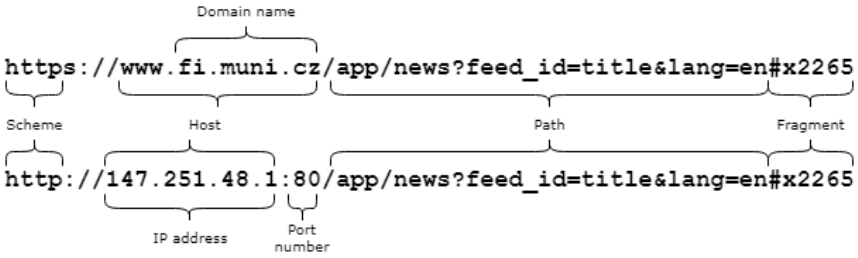


Figure 1.1: Examples of a Uniform Resource Locator (URL)

The term *Malicious URLs* is used for URLs that point to websites that host various unsolicited content. Based on this malicious content, they are used for different kinds of cyberattacks and present a danger to users who unsuspectingly visit them. These URLs can be newly generated, but they are also often compromised legitimate websites.

1.1 Types of malicious URLs

We can distinguish different types of malicious URLs, based on unsolicited content they host and the type of cyberattacks they are aimed for. The most common are spam, phishing, and drive-by download.

Each type can have its own distinguishing features, and a different methodology can be successful in detecting them.

Spam

Webspam refers to websites that are trying to cheat search engines to be ranked higher and increase such their traffic. URLs leading to such websites can be called spam URLs. Despite the higher rank, users will not find there legitimate content they searched for.

In general, there are two most common types of Webspam techniques: content spam and link spam.

- **Content spam** involves all techniques that modify the page content to be rank higher. This includes adding popular words to the actual content, modifying the parts that have a higher importance in a ranking, like a page title or anchor text and other. of the page
- **Link spam** takes advantage of link-based ranking algorithms. These algorithms give websites a higher ranking the more other highly ranked websites link to it.

Phishing

Phishing websites try to steal confidential information such as card numbers, bank account numbers, or passwords from users by tricking them to believe they are on a legitimate website. URLs and content of phishing websites resemble original content, making it hard for the victim to recognize the difference. Acquired sensitive information is then mostly used for stealing victims' identity or money.

Drive-by download

The drive-by download refers to the unintentional download of malicious code (malware) to a user's device after visiting compromised websites. A victim doesn't have to click on anything to initiate the download. Simply accessing an infected website can download and

install malware silently in the background without the user's knowledge. The installed malware then can be used to gain control over the compromised machine, steal passwords or other sensitive information, ransom payment, or other unwanted actions.

1.2 URLs features

Using the whole URL string as only information for detecting URLs may not be enough. Therefore necessarily first step is selecting and extracting useful features that sufficiently describe the URL. Working with features and not using URL as a whole string would allow creating more effective and successful detection algorithms. For each type of malicious URLs, a different set of features can bring better results. Papers addressing malicious URL detection have proposed several types of features that can be used. We can categorize these features into Blacklist, Lexical, Host-based, Content-based features, and Rank-based features [2]. The overview of the properties of those features can be found at the end of this chapter in Table 1.1.

1.2.1 Blacklist features

Blacklist is essentially a listing of URLs that were previously identified as malicious. It is continuously updating over time using a combination of automatic mechanisms and humans.

Using blacklist for identification of malicious URLs is considered as very fast and easy technique to implement with a very low false-positive[1] rate. Unfortunately, this technique fails to detect newly generated URLs, and therefore suffering from a high false-negative[2] rate [3].

However, instead of using blacklist as a solo identification tool, it can be a handy feature for more advanced techniques. Detection tools can use their own created blacklists, or because of high data storage capacity needed, one of many public blacklisting APIs such as from Google Safe Browsing [4] or PhishTank [5].

1. **False positive** – benign URL is misclassified as malicious
2. **False negative** – malicious URL is undetected.

1.2.2 Lexical features

Lexical features idea is based on the assumption that malicious websites look different and thus have distinguishable patterns in their URL text. In many cases is a user able to recognize suspicious URLs just from the look. This applies mostly for phishing URLs, where the URL tends to look similar as of the original page it claims to be. For example, taking the URLs: `https://www.paypal.com/` and `https://paypal.co.uk.b8wt.icu/n/`
The second URL has from the first sight too many domain names and looks suspicious.

Under lexical features, we refer to all textual properties of URL itself, not including the content of the page it points to. Lexical features are attractive to use because they are fast to process, need low amounts of data storage, and can be obtained without having to call other services.

Common detection techniques usually treat lexical features of hostname and path separately, because of their different impact on classification [6]. According to Sahoo, Liu and Hoi[7] we can divide lexical features to *Traditional* and *Advanced* one.

Traditional lexical features

Traditional lexical features include all commonly used lexical features. Most of them are a combination of features motivated by McGrath and Gupta [8] and Kolari et al. [9].

This includes the length of the hostname, the length of the entire URL, as well as the frequencies of the characters in the URL. Further, the presence or order of tokens in the hostname (delimited by '.') and in the path (delimited by '.', '/', '?', '=', '-', ' ') are used.

Advanced lexical features

Advanced lexical features need more computation and are more complex than traditional ones. However, they are more successful in the detection of various obfuscation techniques[7].

Typical examples of obfuscation techniques are (a) host replaced with an IP address, (b) targeted domain in the path (c) large hostnames, additional domains (d) domain unknown or misspelled [10].

Le et al. [11] proposed advanced features such as the presence of an IP address, the number of special characters, or statistical properties about tokens such as their count, average, or maximal length.

A feature that was used by Daeef et al. [12] or Verma et al. [13] is the usage of n-grams, where instead of the presence of single words we identify the presence of n-characters sub-strings of tokens. This way, we can detect sub-words or misspelled words.

Verma and Dyer[14] used as one of the features Euclidean distance, to spot the differences between phishing URLs and standard English.

Kolmogorov Complexity (or called Kolmogorov entropy, algorithmic entropy) used by authors Pao et al. [15] can be considered as a more advanced feature. This method, in case of detection of malicious URLs, is deciding if a URL is benign or malicious by comparing if it has more similar character patterns to given benign or malicious URL database.

1.2.3 Host-based features

Host-based features describe properties that are identified by the hostname portion of the URL[16]. They allow us to approximate the location, owner, registration date, management style, and other properties of malicious hosts. Those features increase the overall accuracy of detection [17] but are all-time expensive, considering they are not derived from URL itself but gathered from third party services.
Typical host-based features used are:

WHOIS properties [3] – date of registration/update/expiration dates, registrars, and registrants of the domain name. According to Ma et al. [6], a malicious URL often has a registration date or update date in the recent past. Also, if a group of malicious URLs is registered by the same individual, it should be treated as a malicious feature.

Domain name properties – time-to-live(TTL) values, existence of pointer (PTR) record or if PTR record resolve one of the host's IP addresses. Furthermore we can include here features that can be considered also like lexical features such as presence of words 'server'

3. WHOIS is a public service used for retrieving information about IP addresses and domain names.

and 'client' or IP address in the host-name.

Geographic properties – physical geographic information - continent/country/city to which the IP address belongs.

Connection speed – speed of the uplink connection. Reputable websites tend to have faster connection speeds.[17]

1.2.4 Content-based features

Content-based features are those obtained after opening and downloading the website. Those features are data-intensive but provide the most accurate information about the content of the website. However, for the phishing websites, they may not be effective as a phishing website should have similar content as the original website it pretends to be. We can divide Content-based features into three sub-categories: HTML, JavaScript and Other Content-based features[18].

HTML features – mostly contain the statistical features of the HTML document, such as page length, the average length of the words, word count, distinct word count, or white space percentage. Furthermore, statistical features about certain HTML elements such as count the numbers of HTML tags, lines, hyperlinks, or iframes are also used.

JavaScript features – statistical properties of javascript code used on websites. Choi et al. [2] counted the number of seven suspicious native JavaScript functions: `escape()`, `eval()`, `link()`, `unescape()`, `exec()`, `link()`, and `search()`.

Other Content-based features – usually more specific and complex features such as visual features, that focus on finding similarity with other websites, or analysis of directory structure of the websites.

1.2.5 Rank-based features

Rank-based features are obtained from services that order or rank in some way domains or individual pages. This is done on the assumption that malicious websites should have a lower rank than the benign ones.

Alexa rank[4] is commonly used[17] for this purpose, calculating the popularity of the website by a combination of estimated traffic and visitor engagement over the last three months. Unfortunately, this ranking system is only domain-based, possibly resulting in link shorteners or web hosting websites, achieving a high score.

Another option is using the data of website popularity in search engines. The risk of this feature is that the content of the website may have been manipulated in a way to get a higher rank in search engines. A solution to this problem could be using a combination of more search engines that use different rank algorithms. For example Choi et al. [2] used for detection link popularity from the five different search engines: *Altavista*, *AllTheWeb*, *Google*, *Yahoo!* and *Ask*.

Type	Category	Properties				
		External dependency	Collection time	Processing time	Size	Risk
Blacklist	Blacklist	yes	medium	low	low	low
Lexical	Traditional	no	low	low	high	low
	Advanced	no	low	medium	low	low
Host	WHOIS	yes	high	low	high	low
	Domain name	yes	high	low	low	low
	Geographic	yes	high	low	mediu	low
	Connection speed	no	high	low	low	low
Content	HTML	no	low	high	low	high
	Javascript	no	low	medium	low	high
	Other	no	high	high	low	high
Rank	Rank	yes	medium	low	low	low

Table 1.1: Overview of features used for detection of malicious URLs

4. https://www.alexa.com/topsites

2 Malicious URLs detection techniques

This chapter describes the techniques used in the field of detection of Malicious URLs. The focus will be mainly on techniques using machine learning, considering they are achieving the best results.

2.1 Blacklisting

Blacklisting technique is the creation of a database of URLs that were previously identified as malicious. In order to check if the link is malicious, we will just look it up in this database. We considered this technique as a feature that can be used for other more complex detection algorithms in chapter 1.2.1.

A common problem for blacklists is finding a reliable source of new URLs and, subsequently, the way of confirming whether they are truly malicious.

Many of publicly accessible blacklists, such as PhishTank[5], URLhaus[19] or VirusTotal[20] get their URLs by manual submissions from users or from external sources. Subsequently, they verify the threat by inspecting it with multiple antivirus programs or other URL/domain blacklisting services.

On the other hand, Google Safe Browsing[4] or OpenPhish[21] are examples of blacklists that don't depend on external resources, and they developed their own analyzing algorithms.

2.2 Heuristic-based approach

Heuristic-based techniques identify and extract common malicious features from websites and use them to detect malicious URLs [22]. Instead of storing whole malicious links, we just store malicious features. This approach is similar to blacklisting but is able to detect threats also in new URLs. Prakash et al. [23] use various heuristic methods to create a predictive blacklist.

It is important to say that all Machine learning approaches involve at the beginning some form of heuristic approach to identify features with an impact on classifying.

As an example of heuristic-based rules one of those used by Solanki and Vaishnav[24]: [h]

```
if (Googlepage rank > 5) feature = Legitimate
elseif (Googlepage rank >= 3 & < 5) feature = Suspicious
else feature = phishing
```

2.3 Machine learning approach

Machine Learning approaches are using a list of URLs defined as a set of features and, based on them, train a prediction model to classify a URL as benign or malicious. This gives them the ability to detect also new potentially dangerous URLs.[7]

As already mentioned, this approach typically consists of two steps: the first one is the selection and appropriate representation of features, and the second one is using this representation to train a prediction mechanism. Machine learning can be classified as *supervised*, *unsupervised*, and *semi-supervised*, depending on whether the algorithm knows if URLs are malicious/benign or does not know it.

Based on how is the model accepting training data and learning on them, we can distinguish two basic groups: *Batch Learning* (learn over finite data group) and *Online Learning* (accepting and subsequently learning over data flowing in streams).

2.3.1 Data preparation

Data preparation is a technique that is used to convert the raw data into a clean data set. The first step is the selection and extraction of useful features that will sufficiently represent URLs as feature vectors. The selection of features is usually made by some heuristic techniques using previous knowledge about malicious URLs.

In the next step, those features vectors need to be transformed into a form that is accepted as input by the algorithm. Usually, if they do not transform data by themselves, they accept only numerical vectors, since the learning is based on mathematical equations and calculations. Moreover, some data sanitization and normalization are needed before feeding them into the model because the quality and the form of data can directly affect the model's ability to learn.

Representing Text Numerically
Key part of feature processing is transforming non-numerical(text) values to numerical ones. We need to transform features containing single text value (host country, top domain) or vector of text values (tokens in the path). In the context of URLs, all of these values are discrete, categorical data.

At the beginning of every encoding technique, the vocabulary of all possible values(words) for certain features is built. And then using this vocabulary, we can do:

1. **Bag-of-Words**
   Bag of Words is a common technique of representing text data used in machine learning algorithms. It is popular because of its simplicity and easy implementation. After building the vocabulary, the simple presence of words is measured.

   Briefly, each text feature is transformed into a vector of 0 and 1 in the size of vocabulary, putting 1 for present value, 0 for not present. If a feature value can contain the same word multiple times, as in the case of tokens in the path, word frequencies can be saved instead of their simple presence.

```
url_one = {'country':'Russia', 'path_tokens': {'be', 'successful'}}
url_two = {'country':'Slovakia', 'path_tokens': {'think','happy', 'be', 'happy'}}

vocabulary_country = {'Russia', 'Slovakia'}
vocabulary_path_tokens = {'be', 'successful', 'think', 'happy'}

url_one_bow = {{1,0},{1,1,0,0}}
url_two_bow = {{0,1},{1,0,1,1}}
url_two_bow_freq = {{0,1},{1,0,1,2}}
```

   This method is called "bag" of words because any information about the order or structure of words is discarded, causing lost of words order or their similarity. [25]. Moreover, it is very heavy on size, with output as a sparse vector (most of the values are 0).

2. **One-Hot encoding**
   This method is similar to Bag-of-Words, but unlike it, it keeps the order of the words. We transform each word into a vector of

0 and 1. It means that a feature containing a vector of words will be transformed into a vector of vectors. So every word will be encoded to vector where only one value is 1 and rest are 0.

```
url_one_one_hot = {{1,0},{{1,0,0,0},{0,1,0,0}}}
url_two_one_hot = {{0,1},{{0,0,1,0},{0,0,0,1},{1,0,0,0},{0,0,0,1}}}
```

Because the output is a sparse vector, this method, similar to the previous one, is considered as inefficient.

3. **Encoding to unique numbers**
   With this approach, each unique word is encoded as a unique number. The order of the words is not lost, but as in the previous method, there is no relationship between the similarity of any two words. The output is a dense vector, and therefore this method is more efficient.

```
url_one_unique_num = {1,{1,2}}
url_two_unique_num = {2,{3,4,1,3}}
```

4. **Word embeddings**
   Word embedding is a representation of words in which similar words have similar encoding. The output is a dense vector of floating-point values. Values are not computed manually but are weights results from the machine learning model.

   This representation allow us to capture word/subword relation (*bank – banking*) or a close relationship from context (*pay – bank*).

   The most famous method for word embedding representation is **Word2vec** developed by Mikolov et al. [26]. Word2vec is not a single algorithm but a combination of two different techniques – Continuous Bag-of-Words(CBOW) and Skip-gram model.

   CBOW predicts the word given its context and Skip-gram predicts the context given a word [27]. Building of the vocabulary is much more complex, taking into account distances among each word

Normalize Data
Varying scales of numerical features may have a negative influence on the learning process in machine learning algorithms. Features with larger values can influence the result more even if they are not necessarily more important. Therefore normalizing the attributes to all have the same scale is needed.

There are different types of data normalization:

1. **Standardization (Zero mean normalization)**
   Standardization is a technique where values are transformed to have properties of the standard normal distribution, i.e., mean of 0 and a standard deviation of 1. Standardized values are usually calculated as follows:

   $$z_i = \frac{x_i - \mu}{\sigma}$$

   where x is original value, μ is the mean and σ is the standard deviation from the mean

2. **Min-Max scaling**
   In this approach, the data is rescaled to a fixed range – usually 0 to 1. Scaled values are achieved by this formula:

   $$z_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

   where x_{min} is a global minimum and x_{max} maximum

3. **Decimal Normalization**
   In this method we normalize by moving the decimal point of value.

   $$z_i = \frac{x_i}{10^j}$$

   where j is the minimal value such that $|z_{max}| < 1$

2.3.2 Batch learning

Batch based learning algorithms accept on the input and train their model over a batch of data. This model is then later used for classification/prediction. After some time model become outdated, and it needs to be retrained on a new batch of data.

Batch learning algorithms may not always be an efficient solution because of the frequent need to retrain, and the amount of data on which algorithm can train is resource-limited.

Support Vector Machine (SVM)
Support Vector Machine is a widely used batch learning algorithm. It was successfully used in the field of detection malicious URLs [6][16][9][15].

SVM constructs a set of hyperplanes in a high or infinite-dimensional space for classification. Then selects the hyperplane that has the largest distance between the nearest training sample and the hyperplane (so-called functional margin)[2]. That can be achieved by computing:

$$max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} y_i y_j \alpha_i \alpha_j K(x_i x_j)$$

subject to

$$\sum_{i=1}^{N} y_i \alpha_i = 0; \alpha_i \geq 0, i = 1, 2, ...N$$

where $K(x_i x_j)$ is a Kernel function used, and α_i and α_j are coefficients associated with examples x_i, x_j computed to maximize the margin of correct classification on the training set.

2.3.3 Online learning

Online learning algorithms train a model in rounds. In each learning round t, model receive x_t, denoting new URL represented as vector of features, and $y_t \in \{-1, 1\}$, true class label of URL, denoting if URL is malicious or benign.
In each round t model makes the prediction of URL's class label $f(\mathrm{x}_t) = w_t x_t$. Where w_t is a weight vector, initialized to 0 at the

time $t = 0$. After making prediction the model receive the true class label y_t, and based on whether its previous prediction was correct or not the model is updated.

Stochastic Gradient Descent (SGD)
Stochastic Gradient Descent is a simplified version of *Gradient Descent* algorithm. Both algorithms iteratively update the weight vector w_t to minimize the error rate with a purpose to produce a model that gives the most accurate predictions.
Gradient descent uses the whole batch sample in every iteration to estimate the update. Therefore, this method is computationally expensive and not suitable for large amounts of data with many features. Large data batch can cause a very long computation time of a single iteration.
Gradient descent updates the weight vector as follows:

$$w_{t+1} = w_t - \eta \nabla_w Q(w_t)$$

where η is the learning rate, and Q is some predefined error function.

Stochastic Gradient Descent computes gradient on a single randomly picked sample x_t instead of whole data set batch. The gradient can be estimated as follows:

$$w_{t+1} = w_t - \eta \nabla_w Q(w_t, x_t; y_t)$$

Because of a random choice of sample, Stochastic Gradient Descent does not always converge to minimum possible error rate, but with a large number of samples and iterations, it will be extremely close. Moreover, the advantage of SDG is much faster and efficient than the Gradient Descent algorithm.

3 Proposed Approach

The main scope of this thesis is to analyze the effectiveness of chosen machine learning classification technologies within the problem of malicious URL detection. Specifically, we are interested in results using only URL address itself without the need to download potentially risky content of the page.

We consider the problem of detecting malicious URLs as a binary classification problem, i.e., URL can be benign or malicious. Determining the type of malicious URL may be possible, but it is usually not the main goal of detection algorithms.

In the beginning, we analyzed and selected the set of features that may have a positive impact on the results. Due to a given fixed dataset, the features that could change over time cannot be used, e.g., host or rank features. In the next step, we extracted and preprocessed these features.

We decided to use SVM^{light}[28], an implementation of Support Vector Machine, as a representative for batch learning. And TensorFlow's implementation of the Keras API, a library for building and training deep learning models, as a representative for online learning approach.

The reason behind this choice is the success of using SVM^{light} [2],[29], and extremely easy and user-friendly use of TensorFlow.

At the end, we analyzed the results from the use of various settings, features sets, amounts of data, or data from different days.

3.1 Data

Our data samples were provided by ESET, IT security company, offering anti-virus and internet security products. URLs had been collected for 20 days, between 11. November 2018 and 20. November 2018.

The data contains two sets of URL, smaller set *Changed* and larger *Sample*. Each of those sets contains three different subsets, CLN, MAL, FMAL, furthermore broken by days when they were detected.

CLN – clean/benign URL

FMAL – URLs detected as malware/malicious

MAL – blocked/malicious URLs

Subset *Changed* contains URLs that during monitored days changed type between 'benign' ⇔ 'malicious'. In *Sample* subset, URLs can change type also, but the vast majority had just one status.

	CLN	FMAL	MAL
Changed	1 325 010	9 596	64 314
Sample	8 937 943	38 202	146 032

Table 3.1: Number of URLs in individual data sets.

The use of this given data set brings some advantages but also disadvantages with it. The biggest plus is that benign URLs samples are from the visual side very similar to malicious ones. This is a big issue of many papers, that benign samples are too perfect compared to malicious. As an example is use of URLs from random page generator from Yahoo [16], [24] or top 1000 pages ranked by Alexa [30].
The disadvantage is the limited amount of features we can use, as features coming from external resources can change value over time.

3.2 Features selection

The proper selection of features used is one of the key steps in machine learning. Features that have no or rather a small impact on results unnecessarily increase the size of the features vector. It is also worth considering using properties that are too time-consuming to obtain if their impact on performance is negligible compared to their contribution.

As stated before, due to a given fixed dataset, we couldn't use features that could change over time, as host or rank features. Therefore in our analysis, we used only lexical features like length, presence of the port, words used, etc.

In addition, we used a blacklisting feature and treated the URL from the MAL subset as blacklisted. Because we knew that it was blocked to access these URLs, therefore they were previously detected as malicious and are on some internal blacklist.

3.3 SVM^{light}

SVM^{light} library is a popular implementation of Support Vector Machine (SVM) developed by J. Thorsten [28]. The original implementation was in C, but since then, many wrapper libraries in other languages, such as Python, Ruby, or Java, were produced. Moreover, the data format, that SVM light accepts on input become widely accepted format by other libraries[31][32].

SVM^{light} has scalable memory requirements and can handle problems with many thousands of support vectors efficiently. Description of optimization techniques used can be found in [33].

3.4 TensorFlow

In our implementation we will use now recommended `tf.keras` module, TensorFlow's implementation of Keras API. This integration was introduced in TensorFlow 2.0 (September 2019)[1], and should provide better integration with TensorFlow features.

Keras[34] is a high-level deep neural network library, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. Keras was originally developed by Francois Chollet, and since its release in March 2015, it gained on popularity due to user-friendly API and support of fast experimentation and prototyping.

1. https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html

4 Implementation

In following chapter we describe the process of data preparation, training and evaluating models with both TensorFLow and SVM^{light} algorithms.

4.1 Data preparation

The correct way of data preparation is one of the most important parts of the machine learning process and leads to better results. The goal of this thesis is to train and evaluate results from different machine learning models. Therefore it's required to have prepared an easy to use tool for processing of raw input data, which can be executed multiple times with different inputs and configuration.

For this purpose, I developed the Feature Generator tool, an extendable, configurable application written in Python for processing input raw URL data. On output is a prepared set of features stored in the required format for both, TensorFlow and SVM^{light} algorithms.

4.1.1 Feature Generator design

The entire data flow consists of four stages: Preprocessing, Feature Generating, Data Storage, and Postprocessing. Each stage has a configurable number of components that can be added or removed based on desired output needs. Components are dynamically loaded based on application configuration. Dependent on the component's specific implementation, multithreading can be supported in order to make processing faster. Complete design of Feature Generator is shown visually in figure 4.1.

Preprocessing
Preprocessing is the first stage, where basic information about input data is retrieved and stored. This include all information that are necessary for further features processing, such as dictionary, or values for data normalization.

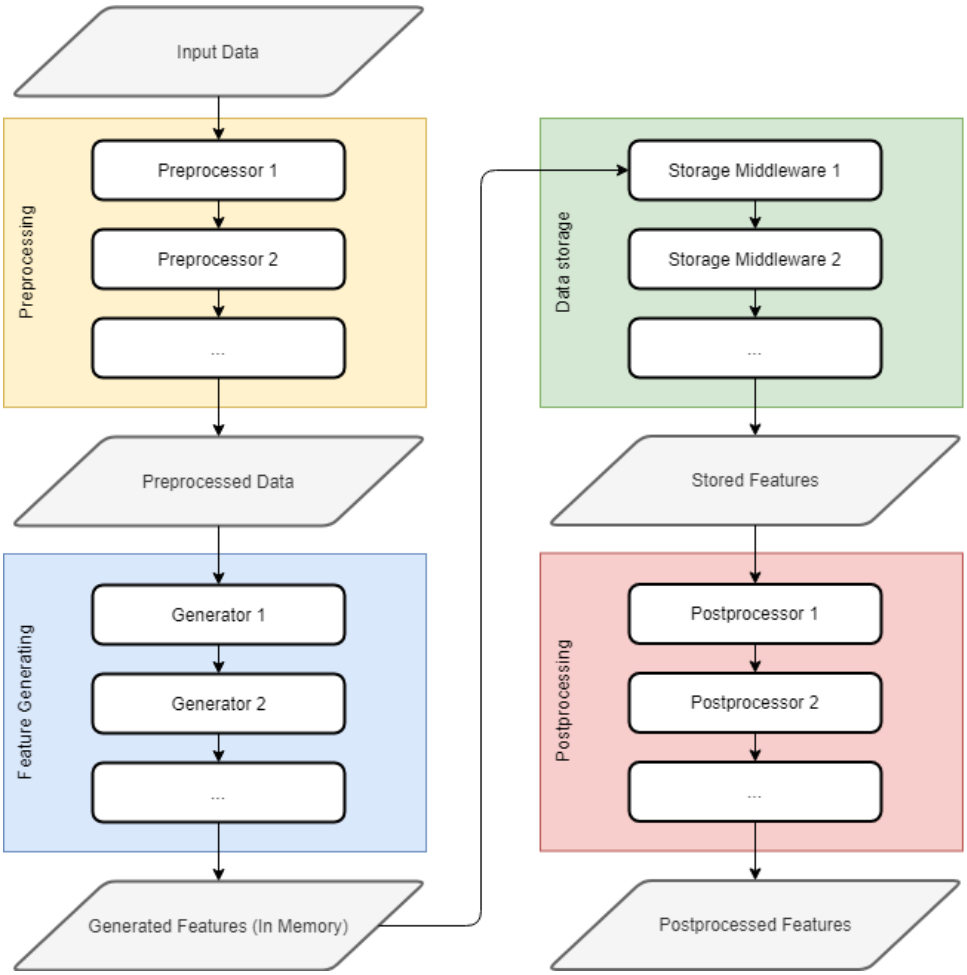


Figure 4.1: Design of Features Generator tool

Feature generating
Feature generating is the main stage of data processing, where data features are extracted and stored for the next stages. The output is a list of extracted features with values.

Data storage
Data storage is a stage where features are transformed and stored into a specific format required by a particular machine learning algorithm.

Postprocessing
The last stage is data post processing. Postprocessor's main purpose is to ensure that output data is finally prepared for evaluation experiments by machine learning algorithms.

Configuration
The important part is the configuration file named config.py containing information about components, their namespace, and specific settings. The core application ensures that each component has specific settings passed as an argument in the constructor, and only enabled ones are used in a data flow. Also, there are global settings used across the entire application.

```
globalSettings = {
    "globalSettingsKey1": "globalSettingsValue1"
}
preprocessors = [
    (
        "preprocessors.preprocessorFileName1.PreprocessorClassName1",
        {
            globalVar.enabledKey: False,
            "preprocessorSettingsKey1": "preprocessorSettingsValue1"
        }
    )
]
generators = [
    (
        "generators.generatorFileName1.GeneratorClassName1",
        {
            globalVar.enabledKey: True,
            "generatorSettingsKey1": "generatorSettingsValue1"
        }
    )
]
                            .
                            .
                            .
```

Figure 4.2: Proposed design of config

4.1.2 Feature Generator implementation and configuration

There are many ways how Feature Generator application can be configured and used. In this part, I describe how this tool is used to meet our requirements. The diagram 4.4 shows which particular components are used in parts of Feature Generator application.

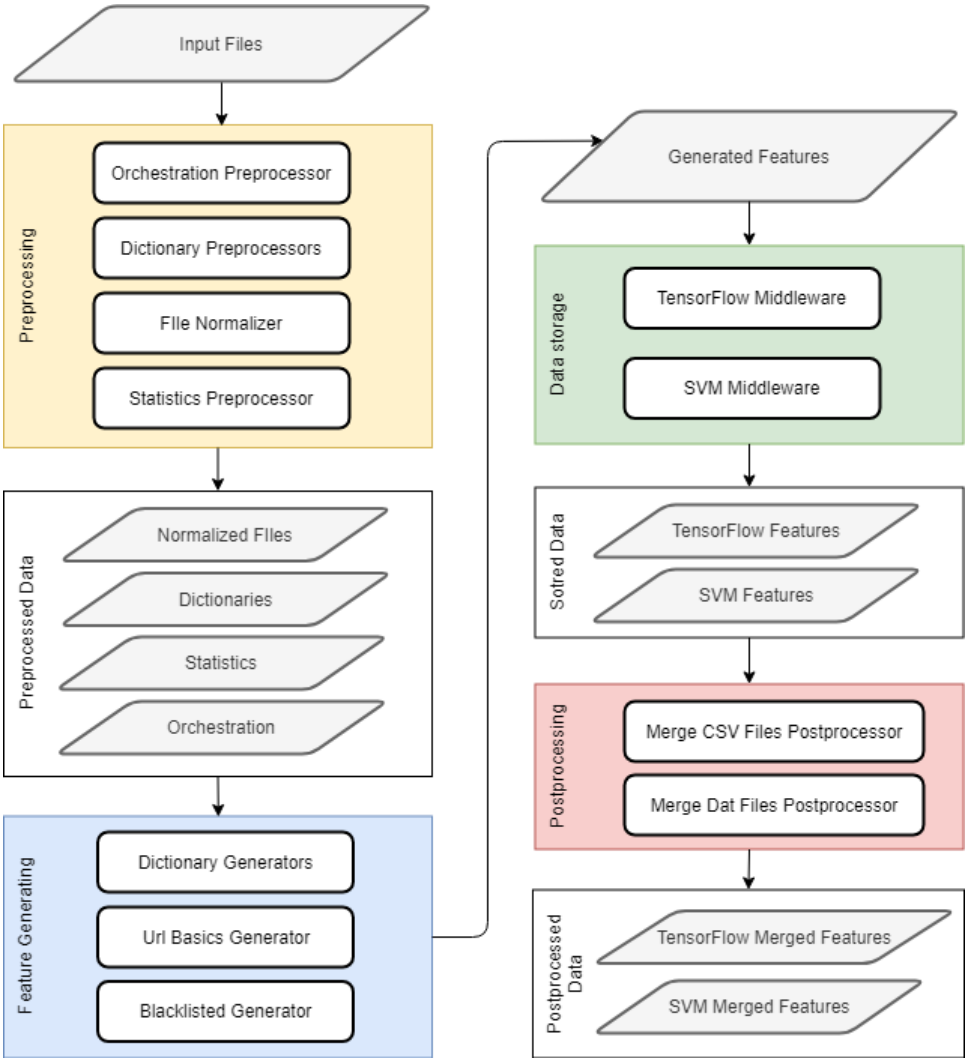


Figure 4.3: Concrete components of Features Generator tool

Orchestration preprocessor
Orchestration preprocessor is the first preprocessor analyzing all input files. The result is the list in size of number of files to be processed, containing basic information about a particular file. All this information is serialized into JSON string and stored in a file named `orchestration.json` by using the default configuration.


```
{"_default": {}, "input_files": {"1": {"name": "sample_cln_2018-11-11.txt",
    "malicious": false, "blacklisted": false, "date": "2018-11-11", "count": 493661}}}
```


Figure 4.4: Example of orchestration.json file

Dictionary preprocessors
Dictionary preprocessors collect words from all URLs in processed files. The result is a vocabulary of unique words with a number of occurrences across all files. Based on the configuration, we can omit rare words. Each word has assigned unique numeric encoding used as a numeric word identifier. There exist two versions of this component: Dictionary Preprocessor, Dictionary Split Preprocessor.

Dictionary Preprocessor creates exactly one vocabulary containing words from the entire URL string. This preprocessor is used primarily for the TensorFlow algorithm, which uses the encoding of words capable of managing words order in URL.

Dictionary Split Preprocessor creates two vocabularies, one is generated from the host part of the URL, and the second one from the path. Dictionary Split Preprocessor is used for the SVM^{light} algorithm that accepts text encoded in a modified version of one-hot-encoding. Therefore if we want to differentiate the position of words in the host or path, we need to create two separate vocabularies and one-hot vectors.

Both preprocessors support multithreading, where each thread processes exactly one file of URLs. Vocabularies are stored in a binary serialized format in files named dictionary.dat, respectively dictionary_domain.dat and dictionary_args.dat by using a default configuration.

File normalizer

In our input files, there is any inconsistency between the amount of malicious and clean URLs. We have significantly more benign samples, but this situation is also expected in the real world. Nevertheless, in order to get more reliable machine learning results, it's recommended to use an equal number of positive and negative samples for training the model.

File Normalizer recognizes whether in a file are malicious or benign URLs by keyword in the file name. The output of File Normalizer is one file per day of a collection containing the same amount of malicious and benign URLs shuffled to random order.


```
{"_default": {"1": {"url": "google.com", "malicious": false, "blacklisted": false},
    "2": {"url": "malicious.com", "malicious": true, "blacklisted": false}}}
```


Figure 4.5: Example of normalized file

Statistics preprocessor

Statistics preprocessor retrieves basic statistics information across all processed files, such as minimum or maximum length of URLs or number of words. This information is necessary for later data normalization. The output is serialized binary file 'stastistics.dat' containing all necessary global statistics used in later calculations.

Blacklisted generator

The blacklisted generator determines whether the URL is blacklisted or not. The blacklisted feature was extracted to a separate generator to be easily enabled or disabled based on application configuration.

URL basics generator

Basic generators extract simple statistical features from URLs such as length of URL, number of words in the host or path, and occurrence of the port. Basic URL Generator uses preprocessed statistics calculations from Statistics Preprocessor in order to normalize data.

Dictionary generator
Dictionary Generator transform URL into an encoded format. The result is URL encoded as an ordered sequence of encoded words. Words are encoded by a unique numeric identifier of a word in the vocabulary.

Dictionary Generator generates exactly one feature with an entire transformed URL while Dictionary Split Generator generates two features with separated encoding for host and path.


```
[[1, 2, 3, 4], [3, 3, 1]], "dictDomain": [[1, 2], [2, 2]], "dictArgs": [[1, 2], [1]]}
```


Figure 4.6: Example of features generated by both Dictionary Generator and Dictionary Split Generator for two URLs

TensorFlow middleware
TensorFlow Middleware is a component, which takes a list of features and its values on input and transforms it into the desired form for TensorFlow stored in CSV format as output. Example of URLs transformed for TensorFlow can be seen in figure 4.7.


```
malicious,blacklisted,length,hasport,hostTokens,pathTokens,dict
1,0,0.020508,0,0.1,0.00694,99 100 7 101 102 103 104 105
0,0,0.005743,0,0.1,0,99 119 7
```


Figure 4.7: Example of encoded two URLs stored in CSV file for TensorFlow

SVM middleware
SVM Middleware transforms a list of features into SVM^{light} specific format and stores it in .dat files. SVM^{light} application requires to have features in ascending order, and they must be unique. For this reason, we cannot use the same storage method as for TensorFlow, and moreover, with words in ascending order, we lose information about the position of words in URL.

Feature in SVM^{light} syntax has two parts separated by a colon, where the first part stands for feature index, and the second part is the value of a feature. For dictionary features, unlike in TensorFlow, every word is considered as a separate feature. Words not present in the URL are skipped and not included in the feature list.

```
1 1:0 2:0.020508 3:0 4:0.1 5:0.00694 12:1  104:1  105:1  106:1  107:1  108:1
109:1  110:1 81882:1  81893:1  81894:1 91288:1  91289:1  91290:1  91291:1  91292:1

-1 1:0 2:0.005743 3:0 4:0.1 5:0.0 12:1  104:1  124:1 81882:1  81893:1  81896:1
```

Figure 4.8: Same example of two URLs encoded for SVM^{light}

CSV file merge postprocessor
CSV File Merge Postprocessor is a post-processing component for merging multiple CSV files created by TensorFlow Middleware into one bigger CSV file containing all data of being merged files. The output file contains precisely one header. The number of files being concatenated is a configurable parameter.

Dat file merge postprocessor
Dat File Merge Postprocessor is a similiar component as CSV File Merge Postprocessor, but is used to merge multiple .dat files.

4.2 Training and evaluating models

Training and evaluating models can be time-consuming, especially in cases of a large amount of data. In the previous section, we described how we are preparing the input data for both models to work more efficiently. In this section, we focus on training and evaluating models with our two selected libraries, TensorFlow and SVM^{light}.

4.2.1 SVM^{light}

For batch machine learning we've chosen binaries of SVM^{light} in version 6.02 for 64 bit Windows platform downloaded from official page[1].

1. http://svmlight.joachims.org

There exists some officially supported Python wrappers for SVM^{light}, however, we had some issues with them and binary worked the best for us at the end.

SVM^{light} consists of two executables: *svm_learn* and *svm_classify*. svm_learn is the module used for training and takes two mandatory and optional amounts of option arguments. An interesting option to change is $-t$, the kernel option. However, in our case change of kernel function didn't bring significant improvement, and so we used by default linear kernel function.
Module is called with following parameters:

```
svm_learn [option] example_file model_file
```

Where *example_file* is in the format explained in previous part and *model_file* is an output file.
svm_classify is the module used for model evaluating and predictions. It has three mandatory and optional amounts of options parameters. However, for module classify, only the amount and format of output information is possible to set. Module is called with following parameters:

```
svm_classify [options] example_file model_file output_file
```

Where *example_file* is in the format explained in previous part and *model_file* is our previously trained model.

4.2.2 TensorFlow

A second machine learning algorithm used was TensorFlow in the version of 2.0 as Python library[2]. We used `th.keras` module, TensorFlow's implementation of the Keras API[3]. For model creation we used its Functional API, as it offer possibility to define our own complex model. We created a Python script, which is responsible for data loading, model creation, iterative or batch learning, and evaluation.

2. https://pypi.org/project/tensorflow/2.0.0/
3. https://keras.io/

Data loading

The result of data preparation for TensorFlow is a CSV file containing one feature per column, including a special dictionary column. In the script, the CSV file is loaded into memory using the Python *Pandas* library[4], which allows easy manipulation of table data. The first column always targets the 'Malicious' column with binary values, true or false. Dictionary column 'dict' is extracted and all rest columns are considered as numeric columns with values from 0 to 1.

Model creation

The model consists of two input layers created by the concatenation of the embedding layer [5] for dictionary and layer of numeric features. There is another dense layer[6] with the ReLU activation function[7] and last dense layer with a sigmoid activation function.
The model is built using Adam optimizer and binary cross-entropy loss function[8].

Learning

There are two possible ways how to give data to created model.

Batch learning – takes data from one file, creates model and trains exactly once.

Iterative learning – Created script supports iterative learning reusing same model saved on file system in order to train with new data. This can significantly decrease the time required for calculation, because there is no need to train on entire datasat again.

4. https://pandas.pydata.org
5. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding
6. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
7. https://www.tensorflow.org/api_docs/python/tf/nn/relu
8. https://keras.io/losses/

Evaluation
Based on configuration, evaluation of dataset from specific file using already created model saved on file system can be executed.

5 Evaluation of the results

The last chapter is dedicated to analysis and evaluation of the results of classification malicious URLs with proposed methods.

We decided to focus on 4 different issues, specifically study the impact of selection different features, set of data, learning method or different amount of sample data.

For evaluation of models we used two metrics, accuracy and loss. **Accuracy** is the ratio of the correct predictions to the total number of test samples.
For binary classification, accuracy can be calculated as follows:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Where TP = True Positive, TN = True Negative, FP = False Positive, FN = False Negative. For using accuracy metrics it is very important to have balanced data set (ratio between data with different labels is the same). Predominance of one dataset can cause false high accuracy, and for this reason we always used 50% of malicious and 50% benign URLs.

A **loss** value indicates how good is the model in terms of being able to predict expected value. If predicted value deviate too much from actual one, the loss value will increase. The goal of machine learning algorithms is to minimize this value.

5.1 Comparison of different features set

The proper selection of features is a important factor in machine learning approach. Different features sets can lead to significant difference in results. Moreover it is interesting to analyse how can one property impact the efficiency of the model.

The initial selection of features was based on previous works in field of detection malicious URLs and on some heuristic approach (statistical analysis of features).

As explained in implementation section we extracted some lexical features from URLs. Specifically we divided these lexical features to two groups, dictionary and numerical. Into dictionary features we

included presence of words in URL, host or path. Numerical features tracked length of URL, count of words in host or path and presence of host in URL. Moreover we used blacklist feature and labeled URLs in MAL subset as blacklisted.

Test Case	1.	2.	3.	4.	5.	6.
Blacklist	x	x		x	x	
Presence of words	x					
Presence of words in host & path		x	x	x		
Numerical features	x	x	x		x	x

Table 5.1: Features selected in different test cases

We decided to split features for test cases into 4 groups: *blacklist*, *presence of words* (we do not differentiate between words in host or path), *Presence of words in host & path* – we distinguish if word is in host or path, *Numerical features* – all lexical features together, as influence of single feature is very small.

Test cases	SVM^{light}		TensorFlow	
	Accuracy	Loss	Accuraccy	Loss
1.	97.19%	0.15486	-	-
2.	97.39%	0.1406	97.34%	0.0447
3.	94.41%	0.3054	94.34%	0.1346
4.	97.31%	0.1415	97.63%	0.0648
5.	Failed		87.85%	0.216
6.	Failed		67.27%	0.5872

Table 5.2: Evaluation of models with different features set

In table 1.1 is summary of evaluation of six test cases with different sets of features. Every model was trained on 352 096 samples (50% malicious, 50% benign), URLs from first 19 days of collection. It was then evaluated on 16 372 different samples, from the last day of collec-

tion. Dictionary in this evaluation contained 1 032 730 different words (86 754 different words in host, 949 619 in path).

From the above table is obvious that the biggest impact on the results had dictionary features, SVM^{light} was not even able to finish the learning without them (test cases 5. 6.). Needless to say, that information about position of the word in host or path (test cases 1.2), did not significantly influenced the results. However we tested this test case only with SVM^{light} model, as for TensorFlow it would require significant rebuild of model.

Together with dictionary features just blacklist had noticeable impact. Test cases without blacklist had several times worse loss value. On the other side numerical values seems to have small or rather none impact on the result.

It is worth noting that the results SVM^{light} and TensorFlow are at the end very similar.

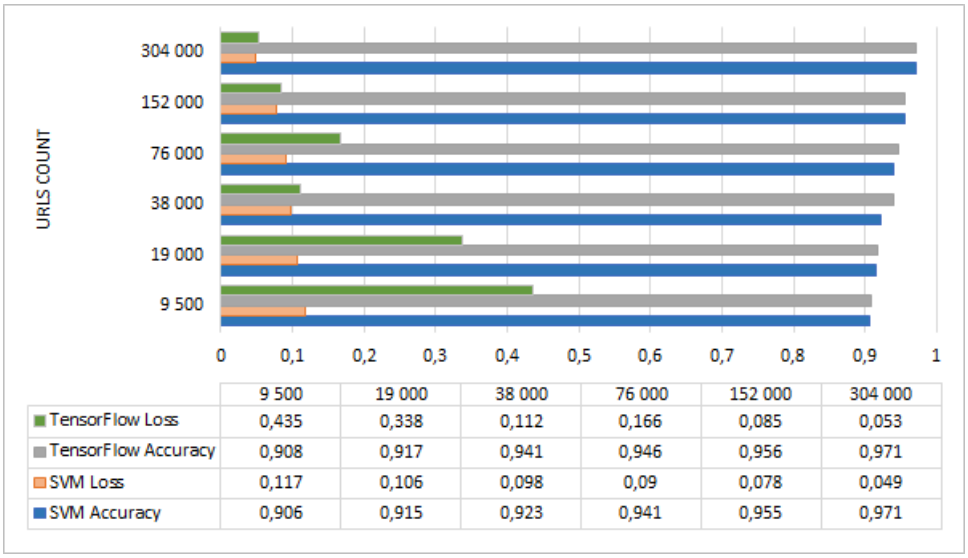
5.2 Influence of data sets size and time

In a real world new malicious URLs are daily generated and detected. They potentially may share some features with already previously detected malicious URLs. And therefore it is very important to look at how the size of the sample data and its age influence the results.

In the table 5.1 we can see that the size of data-set had an impact on the final evaluation. Model accuracy had for both techniques rising tendency, and conversely loss value descending. However we should not compare the loss values of SVM models against TensorFlow loss values, as they use different loss functions.

The loss value of TensorFlow was in one case higher than in cases with less training data. As the data were always randomly selected, it is possible that smaller set with the good representing examples can have potentially better results.

Training data for models were evenly merged from the first 19 days of collection and then evaluated on a daily proportionally equal set from 20th day. Dataset collections again contained 50% of malicious and 50% of benign URLs.



	9 500	19 000	38 000	76 000	152 000	304 000
TensorFlow Loss	0,435	0,338	0,112	0,166	0,085	0,053
TensorFlow Accuracy	0,908	0,917	0,941	0,946	0,956	0,971
SVM Loss	0,117	0,106	0,098	0,09	0,078	0,049
SVM Accuracy	0,906	0,915	0,923	0,941	0,955	0,971

Figure 5.1: The impact of the amount of data samples

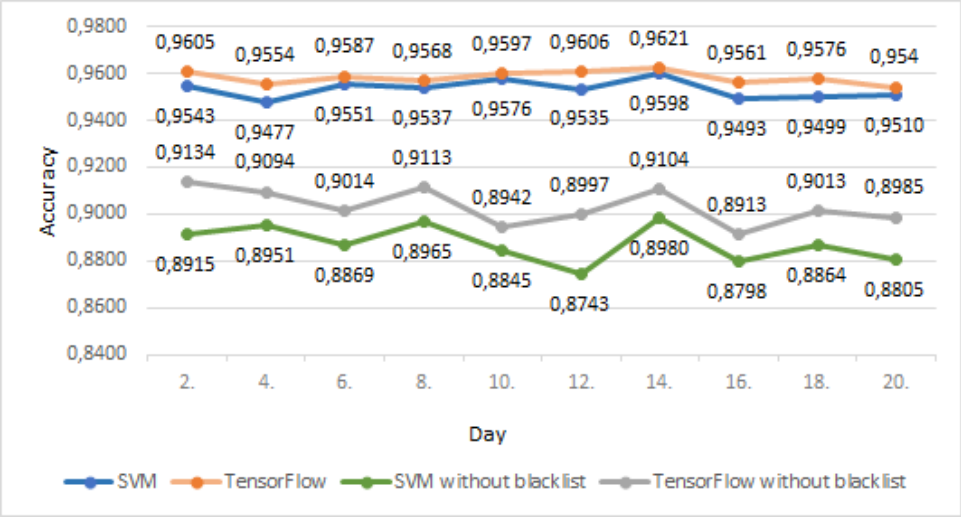


Figure 5.2: Evaluation of old model on new datasets

In Figure 5.2 is shown classification of new URLs by the model trained on data from the first day of collection (23 480 URLs). There is very little almost none difference between evaluation on data from second and from the last day. This can indicate, that malicious websites in our dataset are available and spread for longer period of time.

Slightly bigger difference is between evaluation on datasets without blacklist feature. It follows that this feature positively impacted results evaluated on the old model. This effect can be caused by fact that our blacklist feature contains by itself some new updated information (it was blacklisted at the time of collection).

Nevertheless, the resulting differences are still very small, pointing out that the models are not out-dating rapidly.

5.3 Efficiency of selected methods

So far both selected algorithms, SVM^{light} and TensorFlow, achieved very similar results. In this section we focus on their difference in handling new data, specifically on time and space efficiency.

In the figure 5.3 is shown time needed for adjusting the model to new data on daily basis. All experiments were performed on an Intel Xeon E3-1241 v3 @ 3.50GHz with 16 GB of RAM working under Windows 10 pro system.

We can see that SVM^{light} is much faster on smaller amount of data, but due to the fact that model need to be retrained daily, time consumption rapidly grow. On the other side, TensorFlow time consumption is consistent every day, as he is just adjust model with new data. Because of this TensorFlow is more time efficient solution in the long run, for dataset expected to grow.

Moreover, not just time efficiency but algorithm, that needs to load all data into memory is not suitable solution. Our dataset of 350 000 URLs vectors of features had the size around 70 MB. Working with huge dataset of URLs, thousand of features would significantly increase memory requirements.

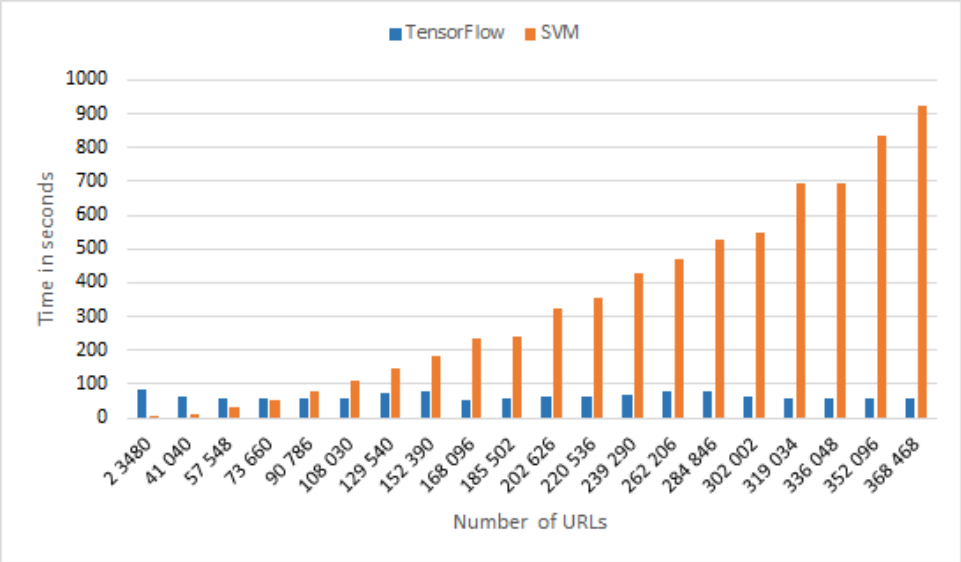


Figure 5.3: Measured training time by amount of data

5.4 Influence of changed data type

Our testing data sets originally consisted of two subsets. The first smaller subset contained URLs that changed their status over time. The second subset, that we primary used because of its size, contained also URLs with changed status, but in a negligible amount.

Our hypothesis is that a model trained on a set with a changing status should achieve worse results. This is based on the assumption that the second time the model receives the same input URL, it predicts its class incorrectly.

However when we run our analysis, we discovered, that this is not fully true. In the case of TensorFlow, there were no significant differences in results between the model trained on changed and normal dataset. But the model trained by SVM^{light} on changed dataset had a noticeably lower accuracy.

Loss value from TensorFlow was reaching on some days high peaks. With changed dataset it would make sense, as the model is expected to guess incorrectly samples that changed status. Unfortunatelly we didn't find the cause of this behavior on normal set.

We simulated daily training of the models on the data available up to that day, and evaluated the models on samples from the next day. Dataset containing URLs with changing status had almost 150 000 samples, the maximum amount of URLs possible to have balanced malicious/benign URLs amount. The second subset contained similar amount of URLs.

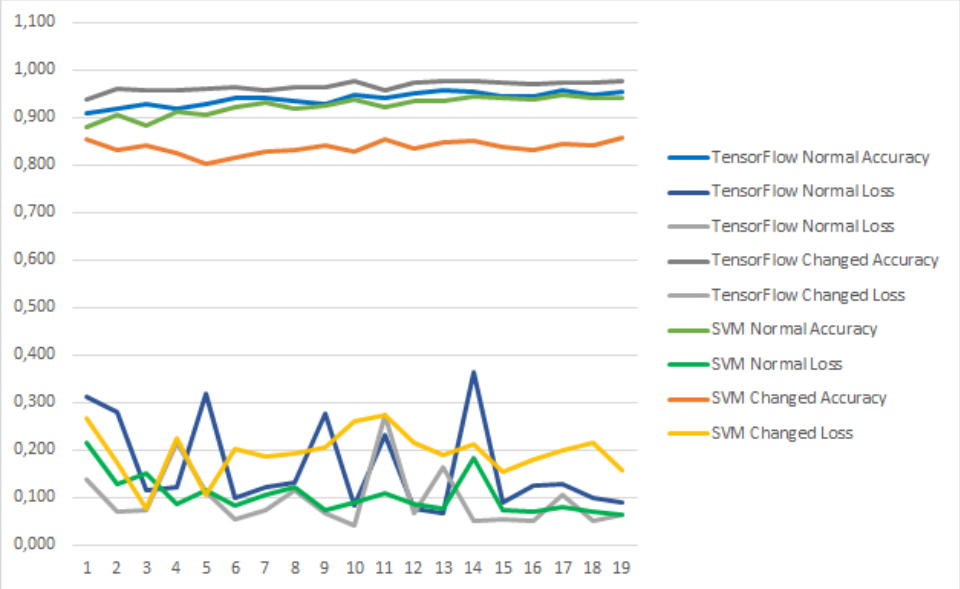


Figure 5.4: Comparison of evaluation models day by day on changing and 'normal'(not-changing) data

6 Conclusions

This thesis focus on the problem of detecting malicious URLs based on the information obtained from URLs string without the need to download page content. We presented and described a variety of features that can be extracted to represent a URL effectively. Then we explained the problem of appropriate representation of features and their possible necessary normalization.

The improper representation of data was the reason we did not initially achieve good and desired results. We proposed and implemented a solution that automatically extracts, normalize, and prepare the input data for different machine learning libraries.

This implementation is fully modular and can be expanded to support additional features, normalization techniques, or machine learning libraries.

The main objective of this thesis was to assess whether machine-learning classification technologies are a suitable solution and can give us a reliable prediction of whether URL is malicious or not.

We executed and evaluated our experiments on the models created by SVM^{light} and TensorFlow library, one representative of batch learning, and one of the online learning approach. We were interested in differences in results, but also in their efficiency and time-consumption. Surprisingly, both approaches achieved very similar results evaluated on the same input data. When we used all features and a training set of size 352 096 URL samples, both achieved accuracy more than 97.3%.

The presence of the word proved to be a feature that can significantly influence the success of the prediction. Model evaluation without this feature was even not successful or had a bad accuracy.

We concluded that SVM^{light} is probably not an appropriate solution due to its exponentially growing time demand. Even though the analysis has shown that both models can successfully detect new threats after a long period of time, and therefore, the daily re-train of the model is not necessarily needed.

6.1 Future work

Due to the use of static datasets for testing and analysis, we could not include the host or rank properties in the list of used features. These properties may change over time, and thus their extraction from the URL after a long time may not reflect the real state at the time of collection. As future work, a tool or way for dynamic data reception can be added, allowing to extract and use more features.

Even though the machine-learning technologies we used achieved very good results, comparing multiple models and settings can bring more interesting insights and results.

Bibliography

1. INC., Webroot. *2019 Webroot Threat Report*. Available also from: https://www-cdn.webroot.com/9315/5113/6179/2019_Webroot_Threat_Report_US_Online.pdf.
2. CHOI, Hyunsang; ZHU, Bin B.; LEE, Heejo. Detecting Malicious Web Links and Identifying Their Attack Types. In: *Proceedings of the 2Nd USENIX Conference on Web Application Development*. 2011.
3. SINHA, S.; BAILEY, M.; JAHANIAN, F. Shades of grey: On the effectiveness of reputation-based "blacklists". In: *2008 3rd International Conference on Malicious and Unwanted Software (MALWARE)*. 2008.
4. GOOGLE. *Google Safe Browsing - Blacklist service provided by Google*. Available also from: https://safebrowsing.google.com/.
5. MALWAREURL. *Phishing blacklist operated by OpenDNS*. Available also from: https://www.malwareurl.com/.
6. MA, Justin; SAUL, Lawrence; SAVAGE, Stefan; VOELKER, Geoffrey. Beyond blacklists: learning to detect malicious Web sites from suspicious URLs. In: 2009, pp. 1245–1254.
7. SAHOO, Doyen; LIU, Chenghao; HOI, Steven C. H. *Malicious URL Detection using Machine Learning: A Survey*. 2017.
8. MCGRATH, D.; GUPTA, Minaxi. Behind Phishing: An Examination of Phisher Modi Operandi. In: 2008.
9. KOLARI, Pranam; FININ, Tim; JOSHI, Anupam. SVMs for the Blogosphere: Blog Identification and Splog Detection. In: 2006, pp. 92–99.
10. GARERA, Sujata; PROVOS, Niels; CHEW, Monica; RUBIN, Aviel D. A Framework for Detection and Measurement of Phishing Attacks. In: *Proceedings of the 2007 ACM Workshop on Recurring Malcode*. 2007.
11. LE, Anh; MARKOPOULOU, Athina; FALOUTSOS, Michalis. PhishDef: URL Names Say It All. 2010.

12. YAHYA, Ammar; AHMAD, R.Badlishah; MOHD YACOB, Yasmin; MOHD WARIP, Mohd Nazri Bin. Lightweight phishing URLs detection using N-gram features. 2016, vol. 8, pp. 1563–1570.

13. VERMA, Rakesh; DAS, Avisha. What's in a URL: Fast Feature Extraction and Malicious URL Detection. In: 2017, pp. 55–63.

14. VERMA, Rakesh; DYER, Keith. On the Character of Phishing URLs: Accurate and Robust Statistical Learning Classifiers. *CODASPY 2015 - Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. 2015.

15. PAO, H.; CHOU, Y.; LEE, Y. Malicious URL Detection Based on Kolmogorov Complexity Estimation. In: *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. 2012.

16. MA, Justin; SAUL, Lawrence K.; SAVAGE, Stefan; VOELKER, Geoffrey M. Identifying Suspicious URLs: An Application of Large-scale Online Learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009.

17. ALTHOBAITI, Kholoud; RUMMANI, Ghaidaa; VANIEA, Kami. A Review of Human-and Computer-Facing URL Phishing Features. In: 2019.

18. MANAN, Wan Nurulsafawati Wan; AHMED, Abdul Ghani Ali; KAHAR, Mohd Nizam Mohmad. Characterizing Current Features of Malicious Threats on Websites. In: 2019.

19. URLHAUS. *Malware blacklist operated by abuse.ch*. Available also from: `https://urlhaus.abuse.ch/`.

20. CHRONICLE. *VirusTotal is an online service that analyzes files and URLs*. Available also from: `https://www.virustotal.com`.

21. OPENPHISH. *Automated self-contained phishing blacklist*. Available also from: `https://www.openphish.com/`.

22. LEE, Jin-Lee; DONG-HYUN, Kim; CHANG-HOON, Lee. Heuristic based Approach for Phishing Site Detection Using URL Features. In: 2015, pp. 131–135.

23. PRAKASH, P.; KUMAR, M.; KOMPELLA, R. R.; GUPTA, M. PhishNet: Predictive Blacklisting to Detect Phishing Attacks. In: *2010 Proceedings IEEE INFOCOM*. 2010, pp. 1–5.

24. SOLANKI, J.; VAISHNAV, R. G. Website phishing detection using heuristic based approach. In: *Proceedings of the third international conference on advances in computing, electronics and electrical technology*. 2015.

25. BROWNLEE, J. *Deep Learning for Natural Language Processing: Develop Deep Learning Models for your Natural Language Problems*. Machine Learning Mastery, 2017.

26. MIKOLOV, Tomas; CORRADO, G.s; CHEN, Kai; DEAN, Jeffrey. Efficient Estimation of Word Representations in Vector Space. In: 2013, pp. 1–12.

27. MA, L.; ZHANG, Y. Using Word2Vec to process big text data. In: *2015 IEEE International Conference on Big Data (Big Data)*. 2015, pp. 2895–2897.

28. THORSTEN, Joachims. *SVM light*. Available also from: `http://svmlight.joachims.org/`.

29. DAVUTH, N.; KIM, S.-R. Classification of malicious domain names using support vector machine and bi-gram method. 2013, vol. 7, pp. 51–58.

30. CHONG, Christophe; LIU, Daniel; LEE, Wonhong. *Malicious URL Detection*. 2009.

31. COURNAPEAU, David. *Search Results Web result with site links scikit-learn: machine learning in Python*. Available also from: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_svmlight_file.html`.

32. CORPORATION, Microsoft. *Machine Learning Studio - Convert to SVMLight*. Available also from: `https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/convert-to-svmlight`.

33. JOACHIMS, Thorsten. *Learning to Classify Text Using Support Vector Machines – Methods, Theory, and Algorithms*. Kluwer Academic Publishers, 2002.

34. CHOLLET, François et al. *Keras* [https://keras.io]. 2015.