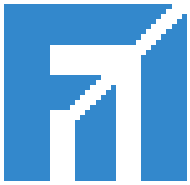
„ALEXANDRU IOAN CUZA" UNIVERSITY OF IAŞI

FACULTY OF COMPUTER SCIENCE



Bachelor Paper

Improving Protection against Internet Attacks

through

Contextual Feature Pairing

by

Georgiana Ingrid Stoleru

Session: *July, 2018*

Scientific coordinator

Assoc. Prof. PhD Dragoș Teodor Gavriluț

„ALEXANDRU IOAN CUZA" UNIVERSITY OF IAȘI

FACULTY OF COMPUTER SCIENCE

Improving Protection against Internet Attacks

through

Contextual Feature Pairing

Georgiana Ingrid Stoleru

Session: *July, 2018*

Scientific coordinator

Assoc. Prof. PhD Dragoș Teodor Gavriluț

Contents

1 Introduction

1.1 Overview

In recent years, the number of software vulnerabilities discovered has grown significantly, as miscreants continue to deliver a variety of malicious programs to hosts around the world. This creates the need for prioritizing the response to new disclosures by assessing which vulnerabilities are likely to be exploited further and by excluding the ones that are not exploited in the real world. One aspect that lays at the heart of the malware delivery techniques consists in the URLs hosting unsolicited content, which leads users to become victims of scams and causes losses of billions of dollars every year. There exists a wide variety of attacks types, such as explicit hacking attempts, Drive-by Exploits, Social Engineering, Phishing, Man-in-the Middle and SQL injections.

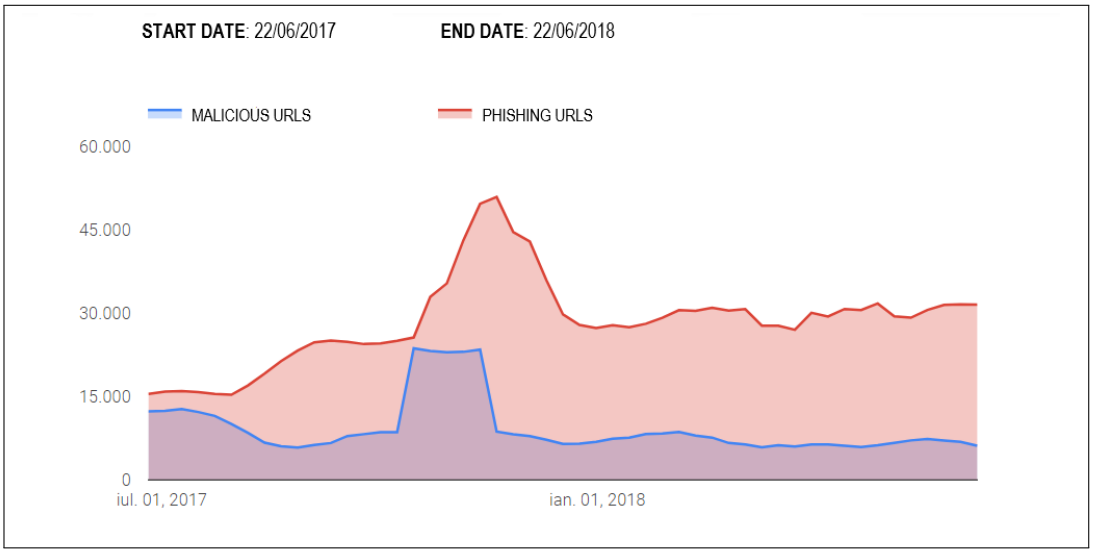


Figure 1: Weekly number of unsafe URLs detected [1]

For example, Safe Browsing service from Google examines billions of URLs and their content, in search of suspicious sites. According to their analysis, during the last year, the weekly number of blocked sites due to

[1]Source: `https://transparencyreport.google.com/safe-browsing/`

unsafe content reached to more than 45000 cases of phishing content and more than 15000 cases of malicious content, with a distribution as the one represented in Figure 1. Considering this aspect, it becomes imperative to design robust systems with a view to detecting cyber-security breaches.

URL is the abbreviation of Uniform Resource Locator, which is the global address of documents and other resources on the World Wide Web. As represented in Figure 2, the URL contains the following components: the protocol identifier, which indicates what protocol to use, the subdomain, main domain and top level domain, the folders, as well as the webpage name with keywords.

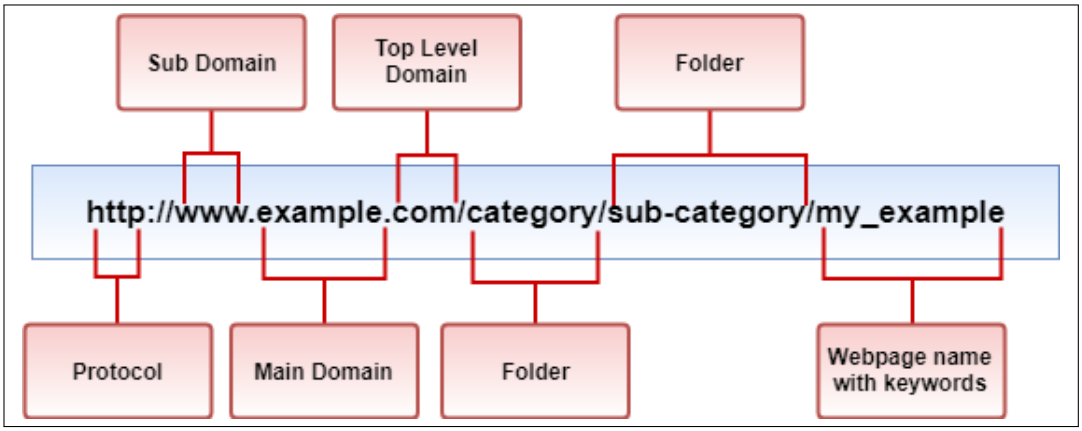


Figure 2: The structure and components of a URL

The compromised websites, which are used in cyber attacks, ar termed as malicious URLs. It has been stated that almost one-third of all websites are potentially malicious, which emphasizes the wide use of malicious URLs in what concerns the perpetuation of cyber-crimes [1].

Popular types of attacks using malicious URLs include: Phishing and Social Engineering, Spam and Drive-by-Download, which particularly refers to the (unintentional) download of malware upon just visiting a URL. These are usually carried out by exploiting vulnerabilities in plugins or inserting malicious code through JavaScript. Also, Phishing, as well as Social Engineering attacks trick the user into revealing sensitive information, by pretending to be regular web pages. Spam represents the usage of unsolicited messages for

the purpose of advertising or phishing.

While effective detection systems become imperatively mandatory, the most common way of detecting malicious URLs consists in blacklists, which are not exhaustive and lack the ability of detecting newly generated malicious URLs.

1.2 Motivation

According to the statistics made by Safe Browsing service from Google, during the last year, the weekly number of displayed warnings has increased by three times, as displayed in Figure 3:

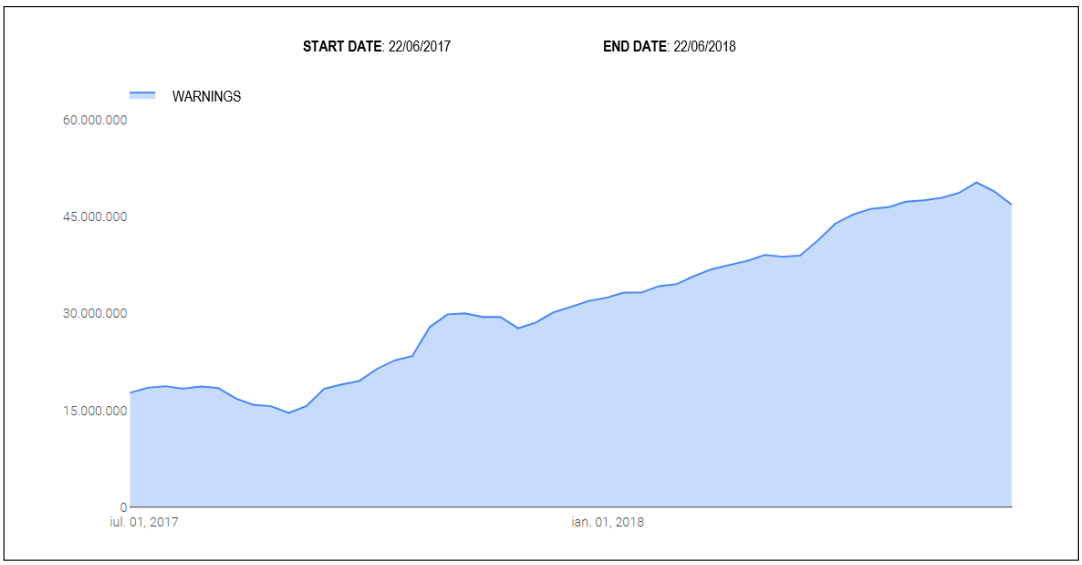


Figure 3: Weekly number of displayed warnings[2]

Furthermore, Proofpoint's Quarterly Threat Report, published on 26^{th} October 2017, shows that there has been an 85 percent rise in malicious phishing emails in the third quarter. Considering the increasing number of attacks, as well as the short lifetime of a URL, it is mandatory to determine a quick method of evaluating the potentially malicious content. The blacklists, which are esentially a database of URLs that have been confirmed to

[2]Source: `https://transparencyreport.google.com/safe-browsing/`

be malicious in the past, have a very low false-positive rate. Moreover, they have a high detection rate, although retroactive, as the detection of a sample requires the confirmation of an analysts that it is malicious. However, attackers use efficient techniques with a view to evading blacklists, especially by obfuscating the URL in order to appear legitimate. The malicious URL can be masked by obfuscating the Host with an IP or another domain, as well as by misspelling the Host. Also, attackers can automatically generate new URLs, which represents a high load for the database. Furthermore, they can launch more attacks, which alter the attack-signature, by making it undetectable by tools which focus exclusively on specific signatures. Taking into account the above mentioned aspects, many anti-malware researches have turned to machine-learning techniques in order to create mathematical models for proactively detecting malicious URLs.

The goal of this thesis is to determine the proper combination of features for a set of particular machine learning meta-heuristics, which would provide us with the best results, from the point of view of the detection rate and the resources needed for training. Starting from the features extracted from the URLs, we wanted to determine whether by adding features extracted from the afferent files, there can be obtained a better detection rate.

This thesis is further structured into the following sections: The *Preliminary* part presents the evolution of security attacks and defenses, as well as the increasing number of malicious URLs and the approaches used in order to detect them. The *Proposed Solution* section describes an approach based on subroutines, considering both their advantages and drawbacks. Furthermore, there are presented the machine learning meta-heuristics which have been used, the datasets, the obtained formal models and the results. Finally, in the last two sections there are presented the conclusions, as well as the future work related to the topic of the current thesis.

1.3 Related work

Recent years have witnessed innovative applications of machine learning in cyber security, which discuss a variety of other cyber threats and do not only focus on malicious URLs detections. For example, [3] present a survey on the usage of machine learning and data mining techniques for Cyber Security intrusion detection. Also, [4] did an empirical analysis of different machine learning techniques for Malicious URL Detection in 2007, at a time when neither features nor machine learning models had been widely explored. While [5] provided us with a broad overview of Phishing, without an extensive survey of the feature representation or the learning algorithms, [6] focused especially on the feature selection for Malicious URL Detection.

As malicious URL Detection is closely related to the area of Spam Detection, it can be mentioned that [7] conducted a survey in 2012, where there have been identified several types of spam, consisting in Content Spam, Link Spam, Cloaking and Redirection, as well as Click Spam. The techniques used for countering them mainly refer to lexical features, such as Bag of Words and Natural Language Processing techniques.

A more comprehensive review on Malicious URL Detection using Machine Learning has been provided in [8]. The thesis also discusses two highly important topics: Feature Representation for Malicious URL Detection and Deep Learning, particularly for Natural Language Processing. Furthemore, McGrath and Gupta [9] analyzed the differences between normal URLs and phishing URLs concerning some features, such as the URL length, domain name length and number of dots in URLs.

Kan and Thi [10] also conducted URL classification based on lexical features only, but their work targets the assignment of categories such as news, business and sports, instead of the detection of malicious URLs. The URL patterns are very valuable as they can provide us with hints concerning the malicious activities. Ma [11] applied machine learning methods to construct classifiers for malicious URLs detection.

But besides the lexical features, he has also used the host-based features, which are related to the host information such as IP addresses, WHOIS data and the geographic properties of the URLs.

Concerning the algorithms used, the perceptron represents a simple, yet efficient algorithm, computing a linear combination of the feature vector and the set of weights. While it proved to generate really good results, it had the disadvantage of the low convergence rate. In [14], the authors proposed a distributed variant of the perceptron, splitting the training set into more shards, so that the training was done in parallel on each shard. At the end, the results were mixed. Also, Chu et al. have successfully tested the performance of various batch training algorithms [15]. There have also been done different trials concerning the machine learning algorithms, which would maximize the detection rate, minimizing the number of false positives. In [16], the authors evaluated different machine learning techniques with a view to reducing false positives, but they concluded that the perceptron algorithm can not be tuned to minimize false positives. Moreover, authors in [17] have analyzed whether the problem of reducing false positives can be solved by using support vector machines with content-specific misclassification costs. Finally, the target has been achieved by [18] by combining 53 spam filters.

2 Contribution

Considering the landscape of malicious threats previously described, it becomes imperative to identify the best detection techniques, which are reliable in terms of detection rate, response time and false positives. This paper presents different approaches for detecting malicious URLs, mainly focusing on machine learning algorithms.

There already exists a suite of standard detections, consisting of blacklists and heuristic approaches. However, the blacklists involve a verification process, based on the human feedback, which is time-consuming, even if highly-accurate. They represent a reactive model, not a proactive one, as they don't detect unknown menaces, such as zero-day threats. Also, blacklists can be

resource intensive, especially considering the need for considerable storing space. Heuristic approaches represent another solution, but they involve a high probability of creating false positives, which can lead to the instability of the analysed system or even to data loss.

In this context, the machine learning approaches seem to be the most effective, as they have the biggest generalization capability, being therefore difficult to bypass. In many cases, these techniques also involve a suite of disadvantages, mostly related to possible false positives. However, in real life situations it is mandatory to have a really low number of false positives, as detecting a clean file can lead to its removal from the system, making it unusable. Mostly, the cause of the false alarms is the fact that the content of the URL is not taken into account.

Therefore, the solution proposed by the current thesis consists in applying machine learning approaches on a set of features, which contain both lexical information extracted from the URLs and information extracted from the content. Because, as stated above, we can not afford false positives, there has been used an adapted version of the Perceptron, which is able to detect malicious samples, while training at a low rate of false positives. A further advantage of this algorithm consists in the fact that the mathematical model is simple, being therefore easy to integrate it in a security solution.

3 Preliminary

3.1 The evolution of security attacks and defenses

In order to understand the necessity of a performant security system from the point of view of the requested resources and the detection rate, we will firstly explore how malware has developed self-defense techniques and how these techniques have evolved as it has become more difficult for malware to survive. We will further provide an overview of the current situation. According to the AV-TEST Security Report, the overall number of malware programs for all the operating systems exceeds 780 million, data which has been recorded until 18^{th} June 2018. Furthermore, their analysis

systems record an average of 350,000 new malware programs per day, which is equivalent to an average of four new malware samples per second. The above image displays the increasing number of global malware recorded by the AV-TEST analysis systems during the last 10 years.

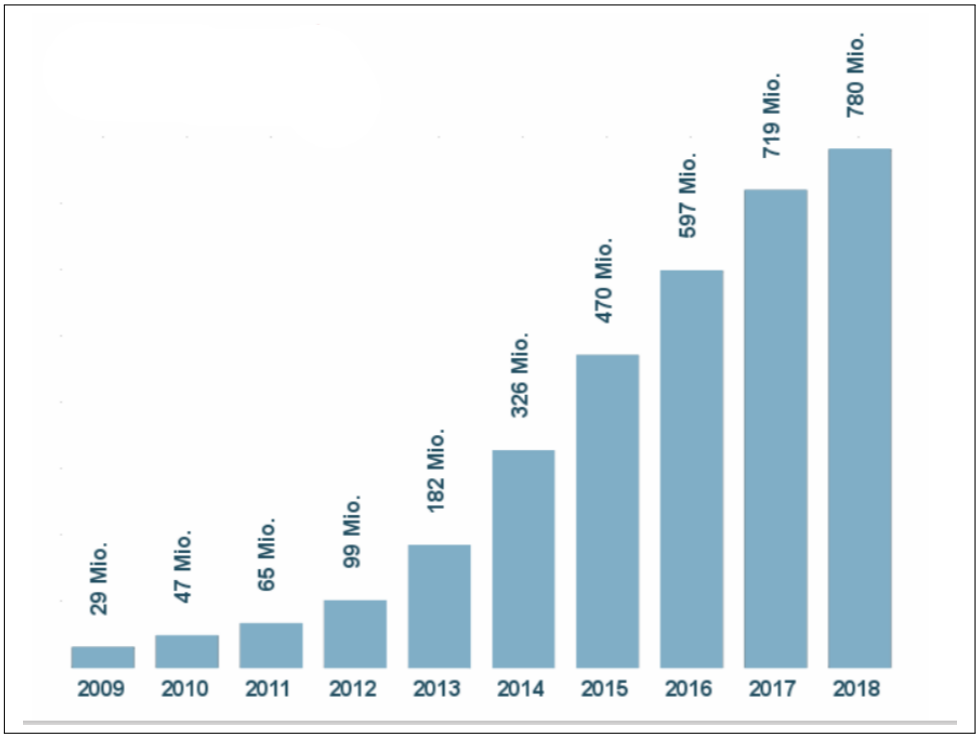


Figure 4: The number of malware samples recorded during the last 10 years[3]

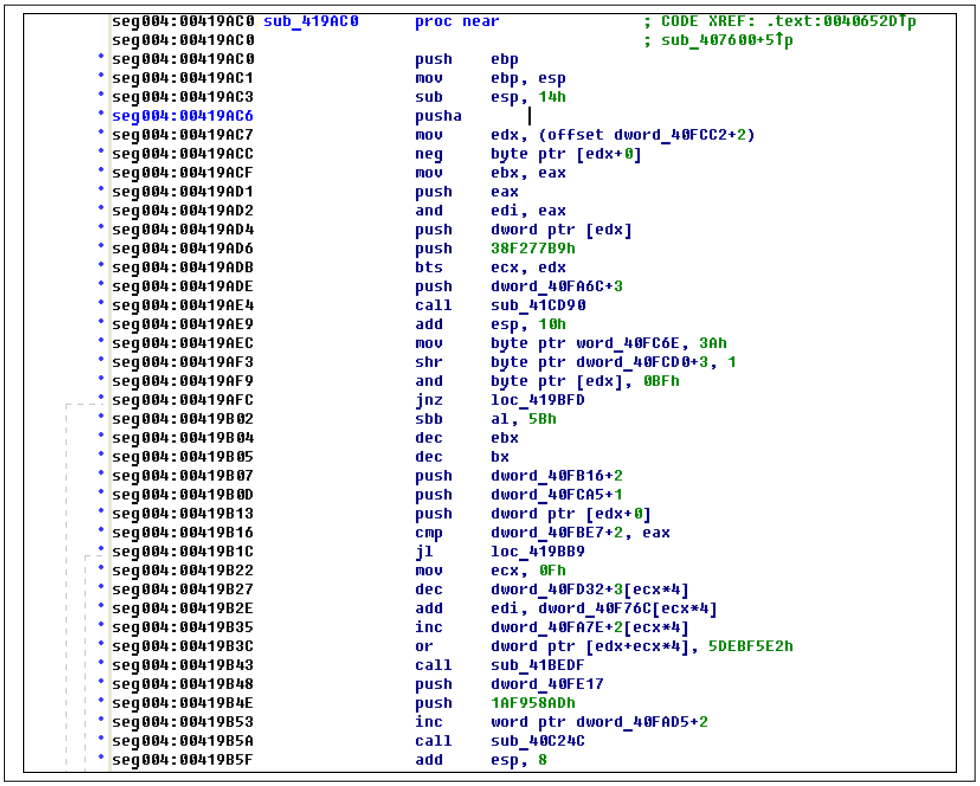
Firstly, the term "malware self-defense" refers to various means of modifying and packing code, in order to conceal the presence of malicious code in the system and to disrupt the functionality of the detection solution.

[3]Source: `https://www.av-test.org/de/statistiken/malware/`

We will consider only the malicious programs written for the Windows operating system (and its predecessor, DOS) due to the rarity and relatively small number of malicious programs for other platforms. Modification of malicious code is done with the view of evading detection and of making the process of reverse engineering more complicated.

While the history of malware began in the 1970s, the history of malware self-defense started in the late 1980s. The first virus that attempted to defend itself was the DOS virus Cascade, which was partially encrypting its own code. However, it has been detected by antivirus programs, as each new copy of the virus - despite being unique from previous copies - still contained an unaltered piece of code that made it possible to be detected every time.

In this context, the rise of polymorphic viruses can be seen as virus writers' response to the increasing expertise of virus scanners. Since the detection rate has been considerably increasing, the only possible manner to evade detection consisted in designing viruses that change their code, thus making recognition by particular strings impossible. Polymorphic viruses employ code alteration and encryption to hide themselves from scanners. The typical strategy consists in encrypting the main part of the code with a variable key and leaving only the decryption executor unencrypted. The decryption code is altered during every infection to prevent detection with a search string. An example of polymorphic virus is Polip, its polymorphic code being displayed in the following image:



```
seg004:00419AC0 sub_419AC0      proc near               ; CODE XREF: .text:0040652D↑p
seg004:00419AC0                                         ; sub_407600+5↑p
seg004:00419AC0                 push    ebp
seg004:00419AC1                 mov     ebp, esp
seg004:00419AC3                 sub     esp, 14h
seg004:00419AC6                 pusha
seg004:00419AC7                 mov     edx, (offset dword_40FCC2+2)
seg004:00419ACC                 neg     byte ptr [edx+0]
seg004:00419ACF                 mov     ebx, eax
seg004:00419AD1                 push    eax
seg004:00419AD2                 and     edi, eax
seg004:00419AD4                 push    dword ptr [edx]
seg004:00419AD6                 push    38F277B9h
seg004:00419ADB                 bts     ecx, edx
seg004:00419ADE                 push    dword_40FA6C+3
seg004:00419AE4                 call    sub_41CD90
seg004:00419AE9                 add     esp, 10h
seg004:00419AEC                 mov     byte ptr word_40FC6E, 3Ah
seg004:00419AF3                 shr     byte ptr dword_40FCD0+3, 1
seg004:00419AF9                 and     byte ptr [edx], 0BFh
seg004:00419AFC                 jnz     loc_419BFD
seg004:00419B02                 sbb     al, 5Bh
seg004:00419B04                 dec     ebx
seg004:00419B05                 dec     bx
seg004:00419B07                 push    dword_40FB16+2
seg004:00419B0D                 push    dword_40FCA5+1
seg004:00419B13                 push    dword ptr [edx+0]
seg004:00419B16                 cmp     dword_40FBE7+2, eax
seg004:00419B1C                 jl      loc_419BB9
seg004:00419B22                 mov     ecx, 0Fh
seg004:00419B27                 dec     dword_40FD32+3[ecx*4]
seg004:00419B2E                 add     edi, dword_40F76C[ecx*4]
seg004:00419B35                 inc     dword_40FA7E+2[ecx*4]
seg004:00419B3C                 or      dword ptr [edx+ecx*4], 5DEBF5E2h
seg004:00419B43                 call    sub_41BEDF
seg004:00419B48                 push    dword_40FE17
seg004:00419B4E                 push    1AF958ADh
seg004:00419B53                 inc     word ptr dword_40FAD5+2
seg004:00419B5A                 call    sub_40C24C
seg004:00419B5F                 add     esp, 8
```

Figure 5: The polymorphic code of Polip

Because it usually takes considerable programming skill in order to design a polymorphic virus, there have been written and distributed polymorphic generators. These are represented by routines which can be linked to existing viruses, with the purpose of hiding actual viruses under the cloak of polymorphism. The first all-purpose polymorphic generator was the Mutation Engine(MtE), which was capable of billions of different permutations, linkable to any virus.

Today, there are 33 different viruses which are known to use the MtE. Another polymorphic generator is represented by TridenT Polymorphic Engine, which is capable of producing a smaller number of different permutations that the MtE, but it creates more generic decryptors. These generators are

typically distributed via underground networks, virus exchange BBSs [4] and private areas in the internet.

From the point of view of the operating principles, polymorphic generators are code modules which a programmer can incorporate into a program, through a process which is called linking. Once a generator is linked to a virus, it becomes an intrinsec part of that particular virus. When the virus is infecting a program file, it requests the generator to create an encrypted copy of the virus code and the generator itself. Besides performing the encryption, the generator also creates a decryptor - a routine which is able to undo the encryption applied to the actual virus code. Since the encryption key is change during every execution, the results of a polymorphic generator are measured by its ability to change the decryption routine.

The basic idea consists in making the binary image of the decryption routine totally different between different infections. This makes it impossible to search for the decryption routine with fixed search string, since there is no search string that could always be found in infections made by a polymorphic virus.

However, there are some limitations, consisting in the necessity of some programming experience in order to link the generator to the program to be encrypted. Furthermore, the generators increase the size of viruses, making it difficult to link them to boot sector viruses, which have limited code space.

Despite their impressive ability of hiding the code through encryption, polymorphic viruses have been detected by the security systems with the support of two major type of methods: the algorithmic methods and the checksum.

Algorithmic methods take into account the fact that however much a generator changes the decryption routine, there will still exist a suite of structures that can be regonized by the antivirus program. There should exist one detection per each generator, as they are different one from each other.

[4] A BBS(Bulletin Board System) is a computer server running software that allows users to connect to the system using a terminal program. Once logged in, the user can perform functions such as uploading and downloading software and data, reading news and bulletins, etc.

Furthermore, as the obfuscation can involve random program structures, the detection system will be prone to a large variety of false positives, as the common structures can be similar to the ones existing in legitimate files.

The second method of detecting polymorphic viruses is represented by checksums, which refer to the values calculated from the executable in a system. Since this strategy leads to the detection of every change in the system, by comparing the current checksums with the original ones in the database, the mutability of polymorphic viruses will be easily identified. However, in order not to provide warnings for legitimate programs, the checksummers should contain an exclusion list. Because the detection methods have evolved concerning this type of self-defense malware, polymorphism and related technologies are not commonly used nowadays anymore.

Afterwards, the malicious programs which needed to be embedded in another samples, have been replaced by fully independent malicious programs. The main methods of propagating the infection consisted in the hard disks and floppy disks, which were small, leading to the necessity of reducing the size of the viruses. That was when virus writers began to use packers, with a view to compressing and archiving files. The fact that the use of packers made it more difficult for antivirus programs to detect malicious samples lead to a new vulnerability which could be exploited. Since this point, the traditional signature-based detecting method was not enough anymore.

Considering that the process of packing changes only the appearance of a sample, without changing the execution semantics, there became mandatory for antivirus solutions to be able to unpack and inspect payloads belonging to packed samples. There have been used three major techniques for unpacking: manual unpacking, static unpacking and generic unpacking. Even if manual unpacking has been widely used, it is time consuming and requires a deep understanding of assembly code, making it possible only for the high-skilled ones to understand the code. The static unpackers refer to dedicated routine for decompressing executables, which are very efficient, but can be bypassed by attackers. Finally, generic unpacking methods consist in programs, such as IDA Pro, which execute and emulate the unknown packed executables until there are decrypted in memory.

As the AV products evolve in what concerns the amount of detections for malicious packed samples, the attackers are continuously trying to evade the detections, trend which will most likely continue in the future. In the following images, there care be observed the difference between an UPX-packed sample and the unpacked version of it:

Figure 6: Comparison between an UPX packed sample and its unpacked version[5]

Finally, rootkits represent a new category of malicious programs called stealthing programs. They generally succeed in gaining full rights on a system, through a vulnerability, making it possible to modify the system in order to use its undetected resources. Once they got into the system, the rootkits delete all the logs, in order not to raise suspicions. Afterwards, the installation process is developed, various user-level programs being modified. Therefore, the malicious program gains administrator rights, enabling access to any resource of the system, without being detected.

There exist various AV solutions, some of which relate to signatures that attempt to detect certain rootkits. Furthermore, there has been provided by the AV products a suite of heuristic detections which are trying to inspect

[5]Size of packed file: 38912 bytes and size of unpacked file: 68096 bytes

abnormal behaviour of a system, represented by some odd output, altered features, suspicious files and traffic. Also, there are detected any attempts of modifying particular elements of a system, services, utilities and deamons. As the threat landscape is becoming more complex, it becomes imperative to have a quick and exhaustive detection system, in order to face security attacks.

3.2 Statistical indicators concerning malicious URLs

In the context described above, malicious URLs themselves represent a considerable threat for the security of a system. As an additional motivation, we can say that in the current year, 2018, there has been identified a new in-the-wild method for spreading mobile malware on Android devices. Once the routers were hacked and the settings of the DNS were modified, the users were redirected to IP addresses, where they were prompted to download malware disguised as browser updates.

If during the 90s, the malicious code was distributed via e-mail, nowadays it is mostly propagated using websites. According to Sophos Labs, there are on average 30000 new websites identified each day, which are distributing malicious code to users passing by.

Furthermore, Sitelock has published its Q4 2017 Website Security Insider analysis of malware and it shows an increase of 20% in what concerns the number of infected websites over Q3 2017. At the same time, it appears that the total number of attacks actually decreased by 20%, which is a proof that it takes a smaller number of attacks in order to compromise the websites, meaning that it becomes more difficult to decode malicious samples and URLs. Further statistical indicators related to the types of vulnerabilities recorded by Sitelock per URL, during this period of time, are as it follows:

Vulnerability Type	Number
Cross-Site Scripting Vulnerabilities	414 pages/site
SQL Injection Vulnerabilities	959 pages/site
Cross-Site Request Forgery Vulnerabilities	414 pages/site

Table 1: Average number of vulnerabilities recorded per site

Also, according to 2018 June data from the AntiPhishing Working Group, the average phishing web site is online for about 54 hours. However, some sites managed to remain online for more than two weeks before being shut down or abandoned. Also, according to the same data, there were 1422 separate phishing scams in June, which represents a 52 percent increase compared to the reports of May. Needless to say, speed is critical to the success of the phishing operations and consequently, to their detection efectiveness. As they mostly migrate from one server to another with a view to increasing the life span of the URL, there exists one scam documented by APWG, which operated on seven servers, during 12 days.

Considering the large number of malicious URLs and their considerably short life span, it becomes mandatory to pay a special amount of attention to the different technologies used for their detection and how they can be improved, with the view of obtaining a higher detection rate, with minimal resources.

3.3 Approaches for malicious URLs detection

3.3.1 Standard detection technologies

The **blacklists** of malicious URLs represent a standard detection technique, which is based on the human feedback, that is time-consuming, even if highly accurate. Also, blacklisting involves no false positives, yet it can not detect unknown malicious URLs. There are several blacklists available, which can be queried before visiting a page. Phishtank is one example, the data being supplied by the public and then verified whether it contains malicious content.

The main disadvantage from this point of view is that the verification process can be slow. Consequently, there is a chance that the new malicious pages will not be on the blacklists, if still waiting for verification. Other methods for feeding the blacklist with data include web crawlers and heuristic content analysis of particular webpages. However, these are not so accurate as they can label the URLs coming from known companies as benign, making them much more difficult to detect. Another disadvantage consists in the fact that the exact match in the blacklist can be easily evaded. For example, as shown in the Table 2, the URL `"www.ywvcomputerprocess.info/errorreport/ty5ug6h4ndma4/"` is detected as Phishing and Malicious by 10 out of 67 AV products, as a result of an analysis request to Virus Total. If we apply a slight modification to the URL, such as adding a white space to the third split, which is "ty5ug6h4ndma4", only 7 engines would still detect that particular URL. Furthermore, if we also change the previous split, "errorreport", by adding one single white space or one character, the URL will be detected by only 6 engines.

URL	**Number of detections**
`www.ywvcomputerprocess.info/errorreport/ty5ug6h4ndma4/`	10/67
`www.ywvcomputerprocess.info/errorreport/ty5ug6h4ndm%20a4/`	7/67

www.ywvcomputerprocess.info/errorre%20port/ty5ug6h4ndm%20a4/	6/67

Table 2: Number of detections after modifying a malicious URL

As mentioned above, another vulnerability is represented by the cases in which the attacker infects a legitimate URL. FireEye has done an analysis from this point of view. They state that according to Alexa, the site "http://www.hmall.com" is being ranked 8,055 worldwide, so a URL reputation database would probably ignore the risk it represents in case of an infection. Usually, when a site is hacked, the attacker sticks in a malicious iframe that does the actual exploiting. According to FireEye, in the current case, the hacker stuck the whole exploit on the legitimate `"http://www.hmall.com"` webserver, which made detection more difficult. Also, the secondary download was on `"http://222.233.53.126/H.exe"`, which at the time of the infection was recognized as malware by 9 AV products on Virus Total.

A further example is represented by the site `"http://www.woodmed.com/Osteopathic%\20Medicine.html"`, which had been hacked to redirect to malware. "Woodmed.com" appeared to be a legitimate medical site and consequently, it would have been allowed by a standard URL categorizing system. In this particular case, according to FireEye, the sever had been directly hacked and there had been added a 302 redirect to some malicious samples. It is possible that a standard categorizing system, either a blacklist or a signature-based one, would provide some protection if it detected the page to which the user is redirected. However, if the attacker would have loaded the malware directly on the medical site, the detection wouldn't have worked anymore.

3.3.2 Machine learning meta-heuristics

As it can be noticed from the above described context, the issue of malicious URLs is a considerable one, especially since the number of attacks has been increasing by 20% during a single quarter last year and the life span of a malicious URL is at most two weeks, before being shut down or abandoned. Needless to say, attackers will be trying to obfuscate the code as to prevent signature based tools from detecting them. They would also target legitimate sites for hosting malware, in order to evade detection, since these are less likely to be identified by a system based on blacklists. However, the blacklists wouldn't detect new suspicious URLs, which represents a highly important issue, as each day there are submitted only on Phishtank more than 1000 new phishing URLs.

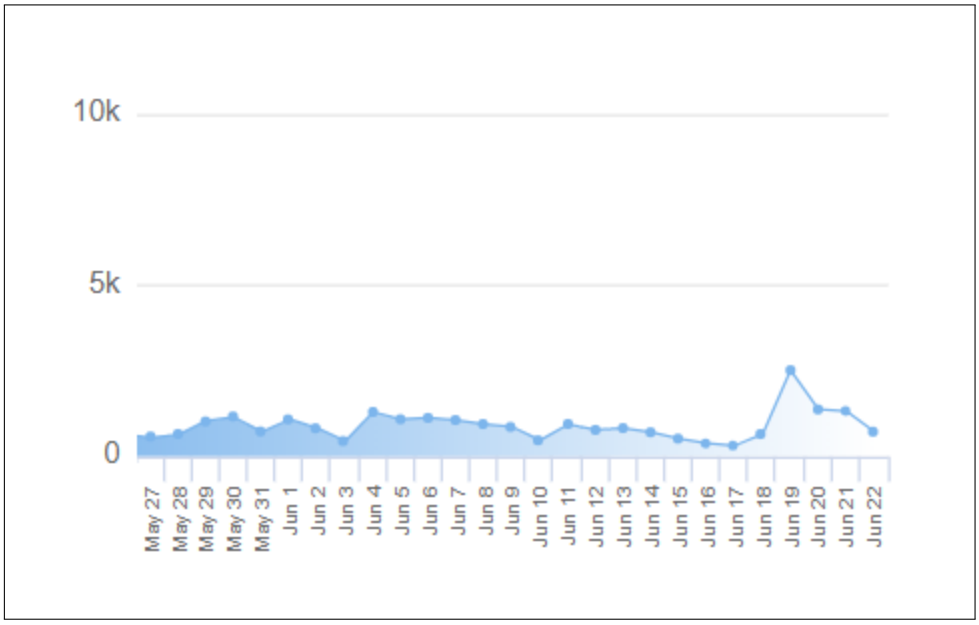


Figure 7: Daily phishing URLs submitted on Phishtank[6]

In order to overcome these issues, the researchers have applied a variety of machine learning techniques for malicious URLs detection. The common aspect of these approaches consists in the fact that they all use a set of URLs

[6]Source: `https://www.phishtank.com/stats.php`

as training data and based on their features, the algorithms learn a prediction function in order to classify the URL as malicious or benign. Consequently, unlike the blacklisting methods, this provides the ability of detecting new malicious URLs as well.

Consider that a label describes the category to which a URL belongs, so whether the URL is malicious or benign. A simplified taxonomy of machine learning approaches would correspond to supervised, unsupervised and semi-supervised algorithms, which correspond to having the labels for the training data, not having the labels and having labels just for a limited fraction of the training data. Once the training data is collected, the next step consists in the process of feature extraction, which refers to the extraction of distinctive properties of input patterns, that help in differentiating between the various categories of input patterns. Isabelle Guyon and André Elisseeff state that we can decompose the problem of feature extraction in two steps, consisting in feature construction and feature selection [12]. The feature construction can be integrated in the modeling process or can correspond to the preprocessing stage. In the step of feature creation, we will consider x the pattern vector, components of which represent the original features. If the features refering to similar objects have different scales, standardization may be applied. The second step is represented by the feature selection, which is usually applied for general data reduction, performance improvement and a better data undestanding. In case there is a large feature set, we would want to limit the amount of storage needed, in order to increase the algorithm speed and to gain a better predictive accuracy.

Needless to say, using the URL just as a string won't be enough in order to learn an effective prediction model, as this would reduce the model to a blacklist method. Consequently, the suitable features should be extracted based on a suite of principles and heuristics. The obtained feature representation will therefore contain lexical features, such as statistical properties of the URL string (Table 3) and host-based features, such as WHOIS info and geo-location properties of a host (Table 4).

Lexical features
Length of URL
Tokens
Number of delimiters
Directory structure
Entropy of the domain name

Table 3: Types of lexical features

Host-based features
WHOIS data
IP address information
Domain name registration

Table 4: Types of host-based features

The quality of the chosen features will influence the efectiveness of the model learned by the machine learning algorithm. Concerning the training itself, there are a variety of classification algorithms that can be used such as Naïve Bayes, Support Vector Machine and Logistic Regression.

However, the large size of the database, which might contain millions of entries, as well as the great amount of time required for the preprocessing of the features can make the training time too high for the model to be used in practice. The URL Classification System described above can be resumed as in the following image:

The perceptron represents a solution for the above mentioned issues, as it provides qualitative results with a small model, which can be easily integrated into a security solution.

The perceptron algorithm is used for learning a binary classifier, so it is a function that maps its input x to an output value f(x). Considering that w is a vector of real-valued weights and b is the bias, the function can be expressed as it follows:

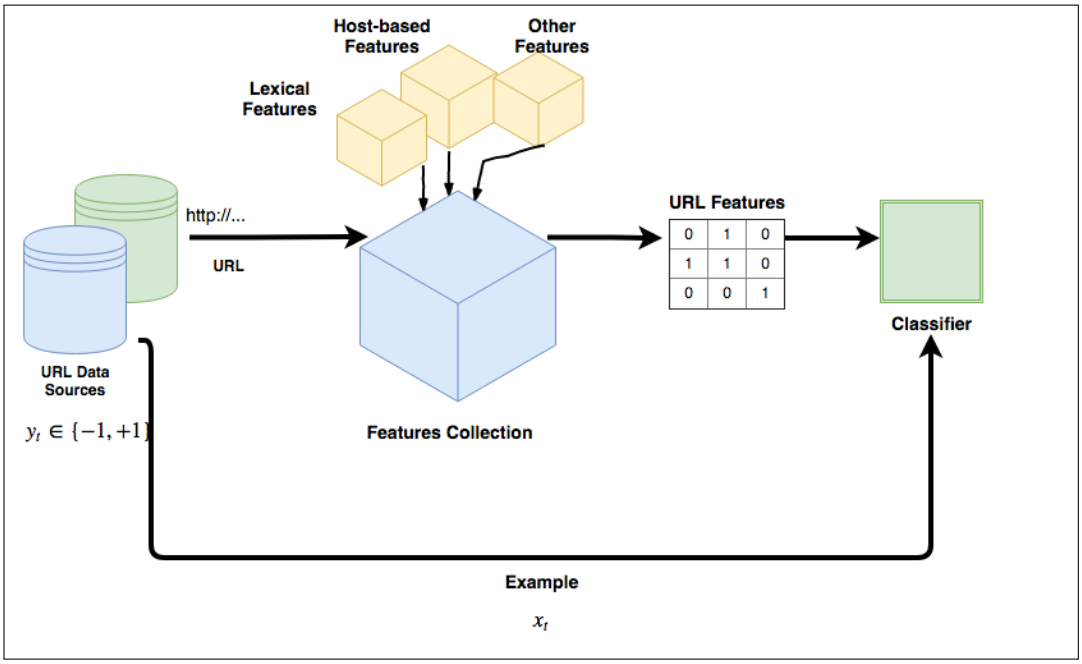


Figure 8: URL Classification System

$$f(x) = \begin{cases} 1, & \text{if} \quad w * x + b > 0^{7} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The bias does not depend on the input and expresses the shift of the decision boundary from the origin. Furthermore, the value of f(x), which is 0 or 1, is used to classify x as either a positive or a negative instance, in case of binary classification problems. In order for the algorithm to terminate, it is mandatory that the learning set is linearly separable.

The simplicity and efectiveness of the perceptron make it suitable for actual security technologies. However, a further issue that occurs at the level of the AV products is that considering the large size of the dataset, as well as the imperative need of having a high detection rate, the resulted models are producing many false positives. In this context, Dragoş Gavriluţ et al.

[7] $w \times x = \sum_{i=1}^{m} w_i \times x_i$

have proposed a modified version of the perceptron algorithm, which is able to detect malicious samples while training at a low rate of false positives [13].

In real life situations, it is mandatory to have a really low number of false positives, as detecting a clean file can lead to its removal from the system, making it unusable. According to the above mentioned thesis, the AV products usually combine more detection mechanisms in order to obtain a high detection rate. This is why a medium detection rate provided by a machine learning algorithm can be considered to be used in practice, as long as there are no false alarms. Gavriluţ et al. state that the simplest step consisted in modifying the bias so that all the clean files were correctly classified. This strategy led to an extremely low detection rate.

The reason why the perceptron was chosen out of the various algorithms which can be used in order to obtain zero false positives was that the mathematical model is small, containing only a linear combination, thus it can be easily integrated in an antivirus product. However, an algorithm used especially for getting zero false alarms should be used in an ensemble with another algorithms in order to achieve a high detection rate.

The strategy used in the above mentioned thesis consists in eliminating, on each iteration, as many of the false negatives as possible, without giving any false positive. An important observation of the authors is that limiting the number of false alarms can considerably contribute to the detection of noise in the database, related to the existance of clean files marked as malicious. The features have boolean values, which signal whether a file is packed, executable, if it downloades other files or injects itself in other processes, etc.

As mentioned above, it is extremely important for a security software to reduce the number of false positives to 0. Even if in theory the perceptron can split the data into two classes if the data is linearly separable, in practice some records are incorrectly classified, due to the time limitations and database noises. This thesis proposes a batch perceptron, enforcing a maximum number of elements which remain incorrectly classified for each class.

Consequently, there has been derived from the batch perceptron a new algorithm, called "One side class-bias", which modifies the bias, so that the

number of clean files wouldn't exceed a certain imposed limit. According to the results, even if the detection rate is lower compared to the one of the batch perceptron, the obtained model is more practical when used for malware due to the low rate of false positives. Furthermore, because the data has been processes in a parallel manner, the training of the model has been done in a small amount of time, making it suitable for a security solution.

Another machine learning algorithm that can be used with the purpose of classifying the websites into two classes, malicious or benign, is represented by Naïve Bayes. This classifier is a probabilistic model based on Bayesian theorem. Even if it is simple, it usually outperforms many other classifiers, especially if trained using supervised learning techniques. Considering that C represents the class to which the sample belongs and F represents a certain feature, then the conditional probability of C given F is provided by the equation:

$$P_r\left(C \mid F\right) = \frac{P_r\left(F \mid C\right) \times P_r(C)}{P_r(F)} \tag{2}$$

In a more natural language, this would be equivalent to:

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}} \tag{3}$$

As the dominator does not depend on C, we are only interested in the numerator of the fraction, which is equivalent to the join probability model, which can be rewritten using the chain rules, as it follows:

$$\begin{aligned} p\left(C_k \mid x_1, ..., x_n\right) = p\left(x_1 \mid x_2, ..., x_n, C_k\right) \times \\ p\left(x_2 \mid x_3, ..., x_n, C_k\right) ... \times p\left(x_{n-1} \mid x_n, C_k\right) \end{aligned} \tag{4}$$

Finally, according to the "naive" conditional independence assumptions, the joint model can be expressed as it follows:

$$p\left(C_k \mid x_1, ..., x_n\right) = p(C_k) \prod_{i=1}^{n} p\left(x_i \mid C_k\right) \tag{5}$$

Once defined the Bayes classifier, there should be chosen a decision rule. The most common of all is to pick the most probale hypothesis, which is the *maximum a posteriori* or *MAP* decision rule. Consequently, the Bayes classifier will asign a class label y for some k as it follows:

$$y = \text{argmax} \quad p(C_k) \prod_{i=1}^{n} p\left(x_i \mid C_k\right), \tag{6}$$

$$where \quad k \in \{1, 2, ..., K\} \tag{7}$$

4 Proposed solution

Taking into account the increasing number of malicious URLs in the past years, as well as the considerable short life span of a suspicious URL, there has become imperative to determine various effective detection techniques, as a countermeasure to the current security threats. As mentioned above, the standard detection methods related to blacklists do not usually detect the new samples, making it easy for the attackers to evade detection by slightly changing the content of the URL. Furthermore, even if the search in the database is quite quick, this approach requires an ever-growing need for extra storage space. However, even if the blacklists can be easily bypassed, due to their simplicity, this technology is highly used by the security solutions nowadays.

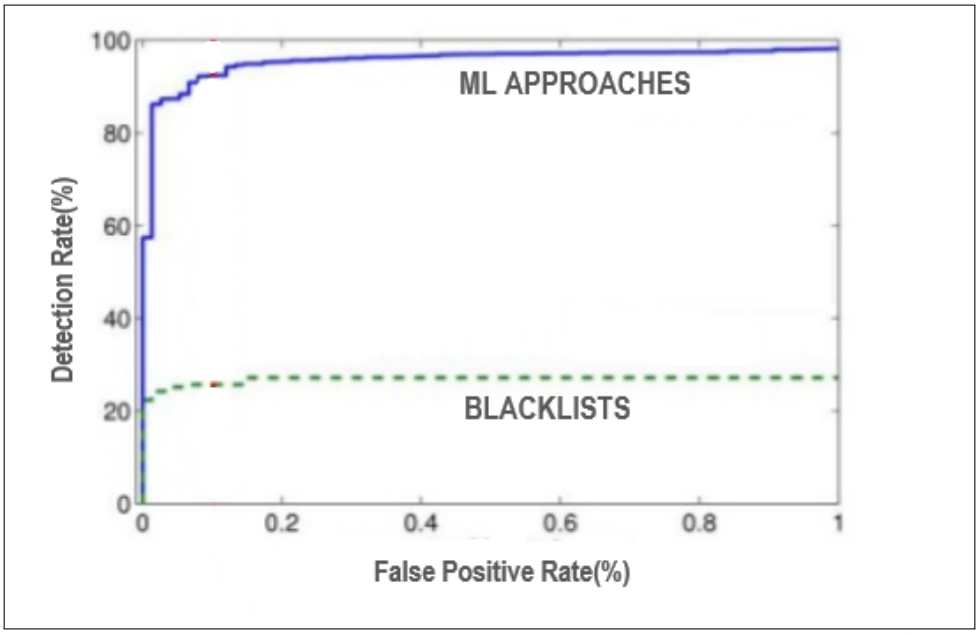


Figure 9: Detection rates for Machine Learning technologies and Blacklists[8]

According to an analysis made by Yahoo and PhishTank, the detection rate for the machine learning algorithms(with the entire collection of features) is 5 times higher than the detection rate obtained using blacklists, as shown in the above chart.

Heuristic approaches represent a further solution proposed by the anti-virus systems. They consist in a suite of signatures, which are meant to identify a common behaviour of the attackers. As each of them is assigned to a certain type of attack, it is still difficult to create an exhaustive list of signatures. However, they have the ability of detecting newly generated malicious URLs, which provides them with a higher generalization capability. These types of signatures can be bypassed mostly through obfuscation. However, they can be improved through the particular analysis of the execution dynamics of a webpage. A drawback from this point of view is the case in which the execution of the malicious code is not done immediately, so that the attack may not be detected.

[8]Source: `https://pdfs.semanticscholar.org/32a1/94ae204549d3d8cb9980f24e0b3ce98a0add.pdf`

The machine learning approaches seem to be the most effective technique of detecting malware nowadays, as they have the biggest generalization capability, being therefore more difficult to bypass. A representative algorithm from this point of view would analyse the information provided by a particular URL, would afterwards extract the features and would train a prediction model on a particular database, consisting of both malicious and benign URLs.

Taking the above mentioned aspects into account, the current thesis aims to make a comparison between the heuristic approaches and the machine learning techniques for the detection of malicious URLs, with a view to obtaining the best detection rate, with the minimal number of false positives and of required amount of resources.

We further want to determine whether we can obtain a better detection rate for the malicious URLs by adding a particular context to the detection, which is done in practice by bringing external information, related to the files which are downloaded from that particular URLs. The algorithm we use from this point of view is the one side class perceptron, introduced in the theoretical part. This choice has been made due to the fact that the mathematical model is simple and it can therefore be integrated in a security solution. Moreover, the training is done at a low rate of false positives, which is crucial for the anti-virus products, as detecting a clean file can lead to the program becoming unusable.

4.1 Subroutines

The first category of malware detection approaches proposed by the current thesis consists in the standard **subroutines**, which are composed of various sets of rules, each of which identifies a specific family of malicious URLs. The particularity of this approach consists in the fact that the rules are manually specified by the analysts, which involves a considerable amount of effort and time.

This method proves to be considerably effective in a suite of attacks, such as the URLs whose content is a PNG file with malicious payload hidden in it. For example, the URL: `http://www.example.com/image.png` has as content the file *"image.png"*, which actually conceals malicious code. Needless to say, there is a high probability for each and every URL which contains an image with suspicious code embedded to be malicious.

Let us further consider the malicious URL: `"http://www.example.com/malware.exe"`. A straightforward subroutine created with the purpose of detecting this single URL could check, for example, if the content of the URL is executable and in case it is, if its name is *"malware.exe"*.

Needless to say, the immediate advantage which derives from this approach is the certainty that the URL will be detected. However, similarly to blacklists, if the name of the file slightly changes to *"malwaree.exe"*, so just by adding one single character, the attacker will easily manage to evade detection.

Consequently, in order to have an increasing detection rate, it is mandatory to have a **less restrictive** set of rules. For this particular example, that could mean that instead of checking whether the name of the file is exactly *"malware.exe"*, we just check if the string *"malware.exe"* occurs in the filename. So any URL whose file content contains *"malware"* in its name would be detected as malicious.

As a result, the rate of false positives(benign URLs classified as malicious) will considerably increase: for example an URL which downloads the tool *"malware-removal"* would be detected as suspicious. Therefore, this approach solves the limitation of the blacklists, but it increases the potential number of false positives. The flow described above is further described:

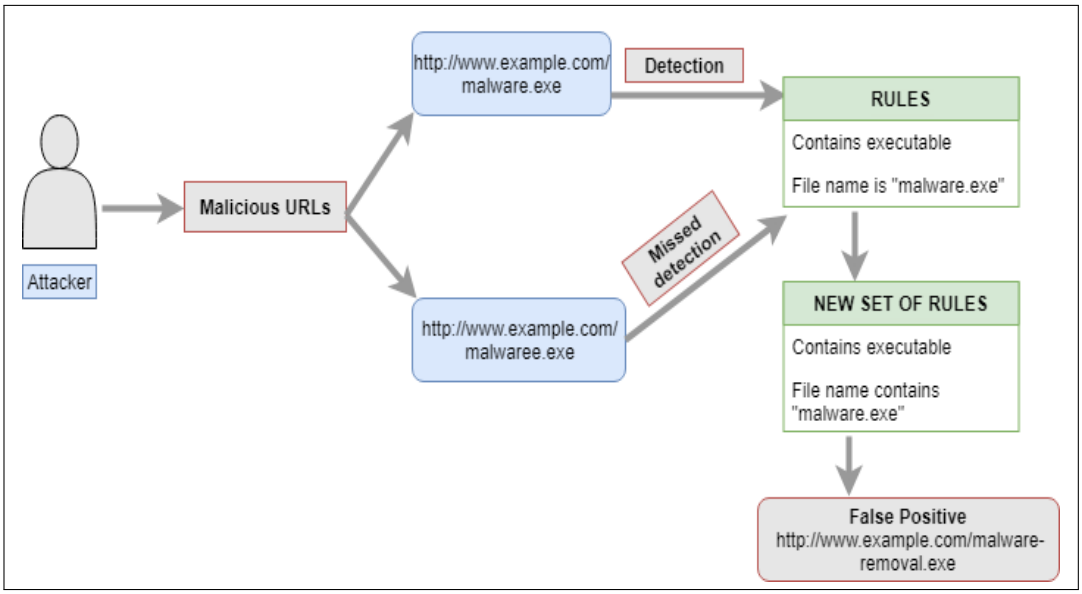


Figure 10: Malicious URL Attack and Subroutine Defense flow

4.2 Contextual detections using one side perceptrons

4.2.1 A ML approach connecting disparate features

We formulate **the problem of malicious URLs detection** as a binary classification task for two-class prediction, consisting in "malicious" and "benign". We will further analyze the steps of the binary classification task, according to the presentation made by Sahoo et al. in [8].

Consider a data set with T URLs $(u_1, y_1), ..., (u_T, y_T)$, where u_t for t = 1, .., T represents a URL from the training data and $y_t \in \{1, -1\}$ is the corresponding label, where $\boldsymbol{y_t = 1}$ represents a **malicious** URL and $\boldsymbol{y_t = -1}$ represents a **benign** one. The automatic machine learning detection would consist of two steps:

1) **Feature Representation**: The extraction of features, which can be formalized as it follows: $u_t -> x_t$, where $x_t \in \mathbb{R}^d$ is a d-dimensional feature vector representing the URL;

2) **Machine Learning**: Learning a prediction function $\mathbf{f} : \mathbb{R}^d -> \mathbb{R}$, which predicts the class assignment for any URL instance x using its feature representation.

Let there be u an URL and f the corresponding downloaded file. The first step consists in converting u and f into two feature vectors, which will afterwards be merged. This means that all the relevant information regarding the two samples will be gathered into a d-dimensional feature vector x ($x \in \mathbb{R}^d$). The features corresponding to the URL belong to the lexical category, consisting in information such as the length of an URL, number of splits, length of primary domain, longest split length, etc.

The next step consists in learning the prediction function $f : \mathbb{R}^d -> \mathbb{R}$, which is usually formalized as an optimization problem: the detection rate is maximized or alternately, a loss function is minimized.

The definition of the function usually requires a further d-dimensional vector w, called the weight vector, such that $f(x) = (w^T \times x)$. We denote by $\hat{y}_t = sign(f(x_t))$ the class label predicted by the function f. The number of mistakes made by the prediction model on the entire collection of training data is: $\sum_{t=1}^{T} I_{\hat{y}_t != y_t}$, where I is an indicator which evaluates to 1 if the condition is true and 0 otherwise.

The following figure illustrates the arhitecture used for solving the problem of malicious URLs detection using machine learning, as described above:

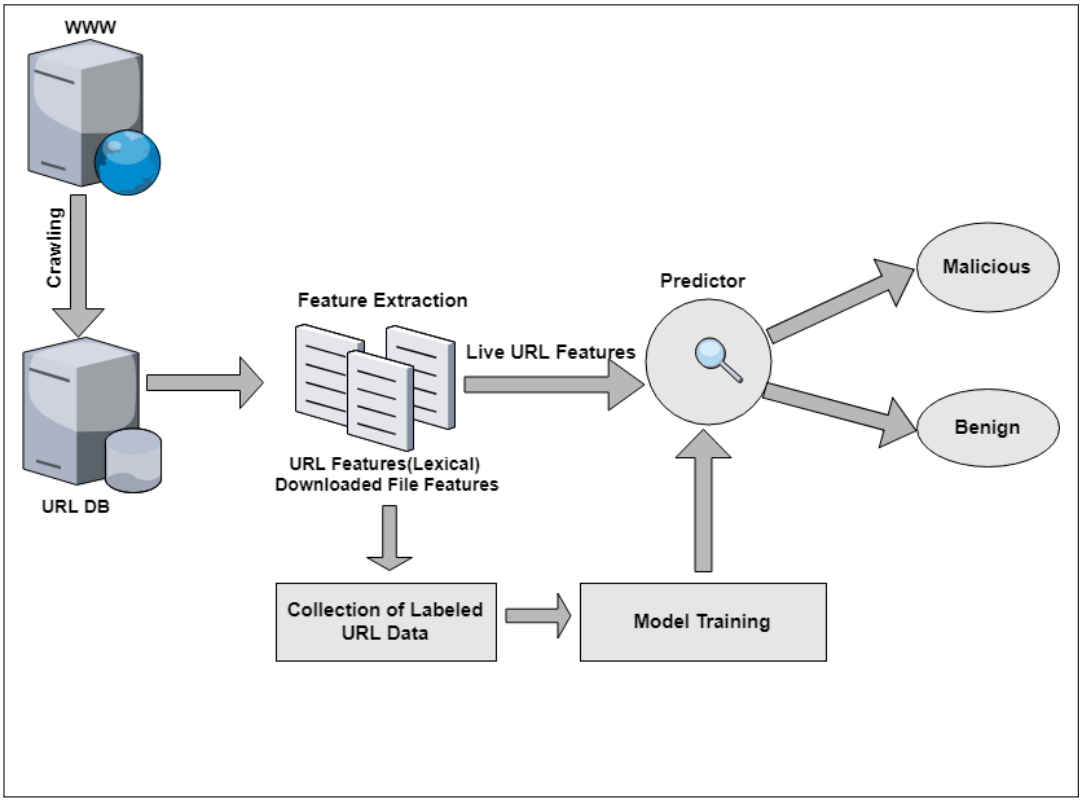


Figure 11: The processing framework for Malicious URL Detection using Machine Learning approaches

4.2.2 Features extraction

The research has been done on a database, composed of 1 million URLs, which have been provided by the Bitdefender Cyber Threat Intelligence Lab. In order to specify the manner in which the data has been retrieved, we firstly need to define the concept of web scraping, also known as crawling. Web scraping represents the process by which an automatic bot or tool extracts particular data from a website. Using this technique, we are able to create custom plugins that follow a well defined protocol for URLs extraction.

Consequently, considering that we want to retrieve data from a particular website, we create a program which starts from the main page, retrieves its HTML files and afterwards searches, using a regular expression, the category, subcategory and the product links. After each step is completed, the program continues, until it acquires the download link of the product. After the sites are scraped, the contents are downloaded by a multi-threaded script and moved to the storage location. Through this process, the database has been populated with the URLs and the corresponding downloaded files, which have been further used for the training process.

Needless to say, the efectiveness of the machine learning approach depends on the quality of the data representation, which is domain-specific. In the current thesis, the process of feature construction belongs to the preprocessing stage. According to [8], the goal of feature representation is to find a mapping $g : U- > \mathbb{R}^d$, such that $g(u)- > x$, where $x \in \mathbb{R}^d$ is the d-dimensional feature vector, that can be fed into the machine learning model.

Concerning this aspect, we firstly extract only the **lexical features** of the URL. The initial assumption is that the properties of the URL as a string can represent a clue whether this is malicious or benign. The first category of lexical features which we extract is represented by the statistical ones, such as the length of the URL, the length of each of the components of the URL(Hostname, Top Level Domain, Primary Domain, etc.), the number of special characters, etc. We define a number of real intervals and we set a boolean flag if one value belongs to one of the intervals. This leads to the discretization of the continuous features mentioned above, so that the

collection of labeled URL Data is fed just with boolean features.

We use boolean features because the storage of the dataset can be highly demanding. If we were to record a real-value for each feature, then we would need 64 bits for each of them. However, using boolean values means that we store only one bit, which translates to a considerable smaller memory needed. Furthermore, considering the dynamic scenario of the ever changing malicious URLs, the boolean features are more resilient to change than the discrete ones.

Therefore, the URL is split by the special character "/". Each of the obtained components is stored by setting a global variable, as the order of the words which appear in the URL needs to be preserved. For example, we need to clearly distinguish between the presence of "com" in the top-level domain vs other parts of the URL. The first component obtained provides us with information regarding the **protocol** which has been used, either "HTTP" or "HTTPS". Afterwards, there are processed the **primary domain** and **the top level domain**(tld).

There are set features which indicate if the domain is short or large, by actually checking if the length of the domain belongs to a particular real interval. Also, there are set features which indicate the frequency of a certain tld, whether it is rarely or oftenly used. For this features to be set, we store a list of highly used and one of rarely used top level domains and search the tld of the current URL in these predefined lists. For example, "bz" and "kz" are rarely used, while "tk", "com" and "net" are frequently used. We continue parsing the URL, until we get to the special character "?", which indicates the fact that we got to the parameters part.

An approach which is widely used by the security solutions for the detection of malicious URLs consists in using the **bag-of-words** features for the machine learning techniques. This is actually a machine learning compatible blacklist. Basically, this method consists in the construction of a dictionary based on all the different types of words in all the URLs. Therefore, each component of a sample URL becomes a feature. If the word is present in the URL, then the value of the feature is set to 1, and 0 otherwise. However, attackers can evade the detection by generating malicious URLs algorithmi-

cally. The **Domain Generation Algorithm**(DGA) is a relevant example from this point of view.

It is well-known the fact that most of the current malware contacts the command and control server(C&C) after being installed on a victim's computer. Also, by design, IP addresses are hardcoded into malware and a higher number of IP addresses increases the probability of infection. The reasoning consists in the fact that one single IP is able to host many base domains, which can house various subdomains, considerably increasing the chance of a successful infection. Furthermore, when one IP is taken down, the C&C server will contact a further one on the list. A visual representation of the previously exposed flow is displayed in the following image:

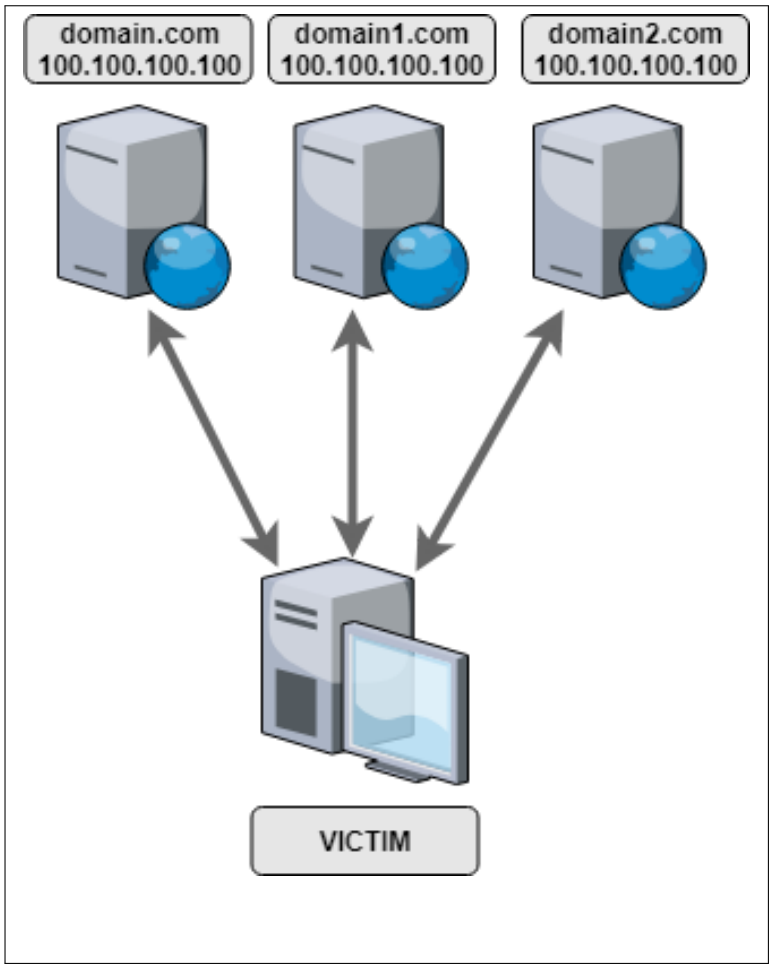


Figure 12: Representation of suspicious connection attempts

Using the date and a seed number, a DGA is capable of generating domains to which an infected computer can connect. A wide list of domains can be generated every day, all with unique, pseudo-random alphanumeric names. The threat actors will select one of the above generated domain names and will set it up either for malicious services or C&C. During the time required for investigation, attackers might already switch to a new domain before the first one being taken down. Taking into account the poor performance of the bag-of-words approach, as algorithmically generated URLs can produce words never seen before, the features collection proposed by my approach does not take into account this category of features.

As seen above, the extraction of traditional lexical features did not require much computational power. However, the machine learning approach would not be as effective, as in the case in which more complex features would be extracted. From this point of view, [19] proposed a suite of advanced features which are obfuscation resistant. According to their classification, we can identify URL related features (keywords, length, etc), domain features (length of domain name, whether there exists an IP address as domain name), directory related features (length of directory, number of subdirectory tokens, etc.), file name features (length of filename, number of delimiters, etc.) and argument features (length of an argument, number of variables, etc.).

Concerning the **URL related features**, there is extracted information regarding the length of the URL, the number of splits, the protocol used("http" or "https"), the hostname, etc. Furthermore, there are also extracted **domain based features**: if the domain of the URL is actually an IP address, if it is using a non standard port, if the Top Level Domain is common(".net", ".com") or specific to a country, if the domain name is randomly generated, etc.

Regarding the **directory related features**, we extract information related to the number of subdirectory tokens, their length, the existence of random words, of small or one letter words, etc. Also, we are interested in the **content related features**, which indicate whether the content might be an executable(if it has an ".exe" or ".scr" or ".dll" extension embedded in the URL) or a document(PDF, DOC, XLS). Finally, the **argument fea-**

tures tell us whether the URL contains parameters and in case it does, if the parameters can indicate log-in information, such as the user name or the password, etc.

Further features extracted from the URLs consist in the domain-based ones, as there is evaluated whether the domain is present in the *white domain table*. Because converting the URL in a feature vector can become very resource intensive, it is not practical to consider a large number of features. Also, there has been done a trade off between the usefulness of a feature, its dimensionality and the difficulty of retrieving the information. Taking these aspects into account, there have been extracted **148 features from the URLs**.

Considering the malicious URL `"http://cdn.discordapp.com/attachments/402490727474528267/407242837365751809/d.exe"`, some of the features which will be set to 1 are exemplified in the following table:

Category	**Feature**	**Description of the feature**
Content related	URL-IS-EXEC	The content is an executable file
Content related	FILENAME-IS-ALPHANUMERIC	The downloaded file name contains alphanumeric symbols
Content related	KNOWN-EXTENSION	The extension of the downloaded file belongs to a predefined list
URL related	HTTP-PROTOCOL	The protocol used is "http"
Domain related	KNOWN-TLD	The tld is common
Directory related	PREV-DIGIT	The last but one split contains only digits
Directory related	PREV-SHORT	The last but one split has a small length

Table 5: A part of the boolean features set to 1 for a malicious URL

However, the attacker may try to obfuscate a malicious URL, making it look similar to benign samples, in order to evade detection. A countermeasure from this point of view consists in using the above extracted lexical features in conjunction with another ones. Many security solutions consider using host-based features in order to improve the performance of the model. However, the current thesis investigates the efectiveness of the detection system after adding further external information to the data which feeds the model. Consequently, there has been downloaded the corresponding file for each URL and there have been extracted its features, therefore adding a particular context to the detection. Out of the samples in the database, there have been chosen only the URLs whose corresponding downloaded files are **executables** (they have a ".exe", ".scr" or ".dll" extension embedded in the URL).

The file features have been extracted using an internal tool, based on various malware specific information, such as the characteristic behaviour in protected environments, the file format from the geometrical point of view, indicators which state whether the file is packed or obfuscated, etc. Furthermore, there have been extracted features related to a file's imports, exports and directory resources, as well as to the different strings which reside in its data section.

The total number of file features extracted as defined above reached the value 8413. After the feature discretization, 2 feature selection algorithms have been applied, based on the F2-Score, respectively the Conditional Mutual Information maximization criterion, which resulted in **256 features** extracted from the corresponding downloaded **files**. Just like in the case of the URLs, only boolean features are generated, due to the same memory and resiliency reasons mentioned above. There are further represented some of the extracted file features before the discretization process took place:

Feature	Value	Description of the feature
IS-MSIL	1(True)	The sample is a MSIL file [9]
SET-STARTUP	1(True)	The program adds itself to startup
IS-PACKED	0(False)	The program is packed with a known packer
RANDOM-WORDS	2	There have been identified two random words in the file
NUMBER-CLASSES	4	The program contains four classes
NUMBER-RESOURCES	1	The program contains one resource
NUMBER-ICONS	0	There are 0 icons in the resources section

Table 6: Part of the extracted features for the corresponding file

The features obtained from both the URLs and the corresponding downloaded files are paired and the obtained collection is further used in the training part. Consequently, there are added various content-related features to the already existing lexical features, previously extracted from the URLs.

[9]The MSIL file extension is associated with Microsoft Intermediate Language, that is used in .NET framework to compile source code (C#, C++, Viusal Basic etc.) into universal language. The .msil files contain code written with the MSIL language.

4.2.3 Formal model

The most common machine learning algorithm that can be used is the perceptron algorithm. Not only does it provide effective results, but it is also convenient from the computational point of view, as it only requires a suite of multiplications and one addition. I have used its derived version (**OSC** [13]), because it is adjusted for a low number of false positives, which facilitiates the integration into a security solution. This approach involves adding a secondary phase after each training step. At this phase, the previously obtained model is modified such that all the elements from the set of benign URLs are correctly classified. This ensures a low number of false positives, at least in the training phase. The underlying base algorithm is presented in Algorithm 1. Consider that there has been denoted by $R_i.Label$ the label of the i-th record and by $R_i.Feature_j$ the j-th feature of the i-th record.

The main advantage of this algorithm consists in the fact that testing if an URL is malicious or benign is done in linear time, which is of particular interest for security technologies, where there needs to be provided a fast response.

Algorithm 1 One Side Perceptron Algorithm(0 False Positives)

1: *// Training*
2: **while** (mist = False) or (iteration <maxNumberIterations) **do**
3: $mist = False$
4: **for** i = 1 to recordsNumber **do**
5: **if** $R_i.Label \times ((\sum_{j=1}^{m} S_i.F_j \times w_j) + b) <= 0$ **then**
6: $beta = beta + learningRate \times R_i.Label$
7: **for** j = 1 to featuresNumber **do**
8: $delta_j = delta_j + R_i.Label \times learningRate \times R_i.Feature_j$
9: **end for**
10: $mist = True$
11: **end if**
12: **end for**

13: **for** j = 1 to featuresNumber **do**
14: $w_j = w_j + delta_j$
15: **end for**
16: $b = b + beta$
17: *// End of training step*
18: *// Class minimize Step*
19: **Train until all the elements in the benign class are correctly classified**
20: *// End of Class minimize Step*
21: $iteration = iteration + 1$
22: **end while**

Furthermore, as stated in the section related to the feature extraction, there are 8413 file features. With a view to reducing the overfitting of learning methods, as well as increasing the computation speed of the prediction, 2 **feature selection algorithms** have been applied in order to select only 256 features from the initial collection.

The first of them is based on the **F2-Score**. The first parameter considered by it is the **precision**, which is the number of correct positive results, divided by the number of all positive results returned by the classifier. The second parameter is represented by the **recall**, defined by the number of correct positive results divided by the number of all relevant samples. Consequently, the F2-Score is a harmonic average of the precision and recall, which reaches its best value at 1 and the worst at 0.

Consequently, we define the following statistical indicators:

Outcome	**Description**
TP(True Positive)	The object belongs to the class **positive** and it is classified as **positive**

FP(False Positive)	The object belongs to the class **negative** and it is classified as **positive**
TN(True Negative)	The object belongs to the class **negative** and it is defined as **negative**
FN(False Negative)	The object belongs to the class **positive** and it is defined as **negative**

Table 7: Possible outcomes and their decription

Considering that the *precision* is defined as:

$$precision = \frac{TP}{TP + FP} \tag{8}$$

And the *recall* is represented by:

$$recall = \frac{TP}{TP + FN} \tag{9}$$

Then, the F2-Score will have the following value:

$$F2 = 5 \times \frac{precision \times recall}{4 \times precision + recall} \tag{10}$$

The F2-Score weighs recall higher than precision, which leads to placing more emphasis on false negatives, therefore increasing the detection rate. The results are further sorted in descending order by their F2-score and the top 256 features are selected.

However, this score is a **uni-variate feature selection method**, as it scores each of the features individually, without considering that a feature may improve in combination with another.

The second algorithm used for feature selection is based on the **Conditional Mutual Information Maximization criterion** [20]. According to the author of the above cited paper, the particularity of this approach is the fact that it does not select a feature similar to already picked ones, even if it is individually powerful, because it does not carry additional information about the class to predict.

This algorithm has been chosen because, according to the experiments done by the author of the paper, it outperforms the other feature selection methods. Furthermore, a naive Bayesian classifier based on features chosen with this criterion achieves error rates similar or lower than AdaBoost or SVMs.

4.2.4 Results

The dataset, consisting of URLs and the corresponding downloaded files, has been, as mentioned in the section related to feature extraction, provided by the Bitdefender Cyber Threat Intelligence Lab. We denote by one sample a pair $< URL, File >$, where File is the content downloaded from the URL at the collection moment. Out of the initial number of 1 million samples, there have been removed all the URLs whose content were not executable files. Furthermore, we noticed that the URLs having identical labels and domains tend to be very similar from the lexical point of view. Therefore, for cases in which the number of URLs which had these similarities was large enough, we only kept a relatively small amount.

Furthermore, the benign dataset consisted in URLs that are hosted on domains, which are often accessed by users. In order to minimize the number of potentially malicious URLs which have been wrongfully labeled as benign, we only kept the URLs which had a longer lifespan than the average for a malicious URL. After these prefilter rules had been applied, the entire collection of files consisted in 98163 malicious samples and 234574 benign ones.

The next step consisted in the feature extraction phase. As mentioned above, there have been 148 features extracted from URLs and 8413 from

files. After applying feature selection on the last ones, there only remained 256 features extracted from files.

Afterwards, there have been removed all the inconsistencies, mostly caused by the malicious samples which generated the same sequence of features as the clean ones. Taken into account the fact that it is mandatory for a security solution to reduce the false positives rate, we eliminated only the sequences of features related to the malicious samples. Finally, the duplicate entries have been removed, therefore obtaining 11107 unique sequences from the malicious samples and 31247 unique sequences from the benign ones. The first training has been realised with only 148 features, all of them extracted from the URLs. The algorithm ran for 2000 epoches, providing a model, let it be OSC-U, with the following results:

Se[10]	Sp[11]	Tn[12]	Tp[13]	Acc[14]	Med
44.002%	100%	31,247	4,887	85.31%	72.001%

Table 8: 148 features — 2000 epochs

We noticed that the Sensitivity is less than 50%. In order for a model to be considered to be implemented in practice the detection rate should be more than 75%. Consequently, we added the features coming from the corresponding downloaded files to the formal model. Firstly, we applied the **F2-Score** for the process of feature selection (only on the features extracted from files), which resulted into 148 features extracted from URLs and 256 features extracted from files. The algorithm ran for 2000 epochs and the resulted model, let it be OSC-UF, provided the following results:

[10]sensitivity - how many positive samples are correctly classified
[11]specificity - how many negative samples are correctly classified
[12]true negative - the total number of correctly classified negative samples
[13]true positive - the total number of correctly classified positive samples
[14]accuracy - (Tn + Tp) / (Tn + Tp + Fn + Fp)

Se	Sp	Tn	Tp	Acc	Med
72.57%	100%	31,247	8,060	92.8%	86.285%

Table 9: 404 features — 2000 epochs

Needless to say, the detection rate has considerably improved, but the results were not as good as expected. One reason might consists in the fact that the F2-Score is a uni-variate feature selection method, scoring each of the features individually, without considering that a feature might improve in combination with another. Therefore, the next feature selection algorithm applied on the entire collection of features extracted from the files was the **Conditional Mutual Information Maximization criterion**. As this approach selects only the features which carry additional information about the class to predict, we expect it to provide improved results. After running the algorithm for 2000 epochs, there has been obtained a third model, let it be OSC-UFF, which provided the following results:

Se	Sp	Tn	Tp	Acc	Med
94.34%	100%	31,247	10,478	98.5%	97.17%

Table 10: 404 features — 2000 epochs

As expected, there has been recorded a remarkable improvement of the detection rate. Another approach of improving the model is discussed in "Malware Detection Using Perceptrons and Support Vector Machines"[21]. According to Dragoş Gavriluţ et al., the concept of *mapping* denotes the process of taking the features to a new dimension, with a view of making them linearly separable. Therefore, from a feature space with m features, there is obtained a new feature space with m(m+1)/2 features. These are obtained applying logical conjunction between the initial features.

By applying this method, we obtained 81,810 features. After running the algorithm for 2000 epochs, there has been obtained a further model, generically named OSC-CM, which provided the following results:

Se	Sp	Tn	Tp	Acc	Med
95.26%	100%	31,247	10,580	98.75%	97.67%

Table 11: 81,810 features — 2000 epochs

Furthermore, the results slightly improved by applying the above mentioned approach for 10,000 epochs. However, in this case, the overfitting needs to be taken into account as well. We called generically the last model OSC-CM1. The results were the following:

Se	Sp	Tn	Tp	Acc	Med
96.60%	100%	31,247	10,729	99.10%	98.3%

Table 12: 81,810 features — 10,000 epochs

The last model is tested at this moment in Bitdefender's technologies in order to receive actual data from its clients and check it for false positives. We expect the model to provide correct results after three months from the moment it has been integrated with the Bitdefender technologies. Currently, it has been tested for one month and from the data classified during this period there have been selected 50,000 URLs, out of which 15,273 malicious and 34,727 clean.

FP + TN	FP	FP rate	FN + TP	TP	TP rate
34,727	52	0.0015%	15,273	12,140	79.49%

Table 13: Real world detection data

As seen above, we obtained a better detection rate, by adding external information related to the files which are downloaded from that particular URLs. Furthermore, the detection rate obtained after applying feature selection through an approach based on the *Conditional Mutual Information Maximization criterion* was considerably higher than the one for which there has been applied the F2-score for the feature selection step. As expected, by mapping the features or by increasing the number of the algorithm's epochs, further improvements were recorded (as seen in the following table)

Approach	Epochs number	Se	Sp	Tn	Tp	Acc	Med
OSC-U	2000	44.002%	100%	31,247	4,887	85.31%	72.001%
OSC-UF	2000	72.57%	100%	31,247	8,060	92.8%	86.285%
OSC-UFF	2000	94.34%	100%	31,247	10,478	98.5%	97.17%
OSC-CM	2000	95.26%	100%	31,247	10,580	98.75%	97.67%
OSC-CM1	10000	96.60%	100%	31,247	10,729	99.10%	98.3%

Table 14: Comparative results

5 Conclusion

As the number of malicious attacks over the Internet has grown significantly in recent years, so did the necessity of prioritizing the countermeasures taken by the security solutions. It is well-known the fact that there exists a suite of standard detection techniques, mostly consisting in blacklists and heuristic approaches.

As seen in the first part of the thesis, the blacklists represent a reactive method, not a proactive one, as they don't detect unknown menaces, such as zero-day threats. Also, they can be resource intensive, especially considering the need for considerable storing space.

Heuristic approaches represent a further solution, but their implementations in many cases is not very successful or reliable. Also, they involve a high probability of creating false positives, which can lead to the instability of the analysed system or even to data loss.

Considering the above mentioned aspects, malware detection has started to increasingly rely on machine learning techniques, which use different feature collections in order to make a clear distinction between the malware samples and the benign ones. However, this approaches often involve an increasing number of false positives, mainly because the content of the URL is not taken into account.

The solution proposed in the current thesis consists in pairing the lexical features extracted from the URLs with the features extracted from the corresponding downloaded samples and further applying machine learning techniques on the obtained collection of features. Considering the high probability of false alarms concerning the URLs, due to the increasing number of legitimate sites which are hacked, there has been used an adapted version of the perceptron, able to detect malicious samples, while training at a low rate of false positives.

The results prove that the above presented models can be successfully used in detecting malicious URLs with a low rate of false positives and a reasonable amount of storage space required for a large dataset. Consequently, training on a collection of paired features extracted from both the URLs and the corresponding content, will provide us with a much more effective model, which can be easily integrated in a security solution. The next step consisted in applying the mapped version of the above mentioned perceptron, which proved to be better than the first one, as the accuracy has been improved by 2%, leading to a value of 96%.

6 Future directions

As a future work, this machine learning approach could be extended for different specific protocols, also considering further categories of features, both for the URLs and the corresponding downloaded files. From this point of view, one example is represented by the host-based features extracted from the URLs. These would allow us to know the location of malicious hosts, their identity, management style and properties. For example, the phishers oftenly exploit the Short URL services, which is valuable information to take into account at the stage of feature extraction.

Furthermore, the algorithms could be ported on the GPU of the clients and the processing of the data could be done in the cloud. Running the computationally intensive algorithms on GPUs over CPUs would drastically minimize their processing time. Furthermore, processing data in the cloud would increase the level of reliability, leading to higher performance with less resources.

Moreover, this approach could take into account the reputation of a particular analysed sample. The simplest technique for computing a reputation score consists in recording the number of clients on the computer of whom that sample had been recorded. Considering a lot of 10 samples, 9 of which are detected, it can be stated that there exists a 90% detection rate. However, if each of the detected samples infected one single computer, while the last one infected 1000 computers, then the above detection rate is not relevant anymore. Consequently, there needs to be taken into account the number of different IPs which hit an URL. This number is more relevant than the total number of hits of that particular URL, as an attacker could generate many hits per URL just with the view of misleading the malware analysts.

References

[1] B. Liang, J. Huang, F. Liu, D. Wang, D. Dong, and Z. Liang. "Malicious web pages detection based on abnormal visibility recognition in *E-Business and Information System Security, 2009. EBISS09. International Conference on.* IEEE, 2009, pp. 15.

[2] Takashi Mitsuhashi. "Impact of feature extraction to accuracy of machine learning based hotspot detection in *in Proceedings Volume 10451, Photomask Technology; 104510C (2017)* Event: SPIE Photomask Technology and EUV Lithography, 2017, Monterey, California, United States.

[3] A. L. Buczak and E. Guven. "A survey of data mining and machine learning methods for cyber security intrusion detection", *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 11531176, 2016.

[4] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair. "A comparison of machine learning techniques for phishing detection in *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit.* ACM, 2007, pp. 6069.

[5] M. Khonji, Y. Iraqi, and A. Jones. "Phishing detection: a literature survey in *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 20912121, 2013.

[6] H. Zuhair, A. Selamat, and M. Salleh. "Feature selection for phishing detection: a review of research, *International Journal of Intelligent Systems Technologies and Applications.*, vol. 15, no. 2, pp. 147162, 2016.

[7] N. Spirin and J. Han. "Survey on web spam detection: principles and algorithms, *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 5064, 2012.

[8] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. "Malicious URL Detection using Machine Learning: A Survey" (2017).

[9] D. K. McGrath and M. Gupta. "Behind phishing: an examination of phisher modi operandi" in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 4:14:8, Berkeley, CA, USA, 2008. USENIX Association.

[10] M.-Y. Kan and H. O. N. Thi. "Fast webpage classification using url features" in *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM 05, pages 325326, New York, NY, USA, 2005. ACM.

[11] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. "Beyond blacklists: learning to detect malicious web sites from suspicious urls" in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD09, pages 12451254, New York, NY, USA, 2009. ACM.

[12] Isabelle Guyon and André Elisseeff. "An Introduction to Feature Extraction" in *IBM Research GmbH*, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland and *ClopiNet, 955 Creston Rd.*, Berkeley, CA 94708, USA.

[13] Dragoş Gavriluţ, Răzvan Benchea and Cristina Vătămanu. "Optimized zero false positives perceptron training for malware detection" in *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timişoara, September 2012.

[14] R. McDonald, K. Hall, and G. Mann. "Distributed training strategies for the structured perceptron" in *HLT '10 Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 456464, 2002.

[15] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, and A. Y. Ng. "Map-reduce for machine learning on multicore" in *Advances in Neural Information Processing Systems 19*, pp. 281288, 2006.

[16] K. Tretyakov. "Machine learning techniques in spam filtering" in *Data Mining Problem-oriented Seminar, MTAT 3*, pp. 6079, 2004.

[17] A. Kolcz and J. Alspector. "Svm-based filtering of e-mail spam with content-specific misclassification costs" in *Proceedings of the TextDM01 Workshop on Text Mining - held at the 2001 IEEE International Conference on Data Mining*, 2001.

[18] T. R. Lynam, G. V. Cormack, and D. R. Cheriton. "On-line spam filter fusion" in *SIGIR 06 Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 123-130, 2006.

[19] A. Le, A. Markopoulou, and M. Faloutsos. Phishdef: Url names say it all, in *INFOCOM, 2011 Proceedings IEEE*, 2011, pp. 191195.

[20] François Fleuret. Fast Binary Feature Selection with Conditional Mutual Information, in *Journal of Machine Learning Research*, 2004.

[21] Dragoş Gavriluţ, Mihai Cimpoeşu, Dan Anton and Liviu Ciortuz. Malware Detection Using Perceptrons and Support Vector Machines, in *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, 2009.