

**DISSERTATION**

**ON**

**Phishing Sites Detection**

**SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS  
FOR THE AWARD OF DEGREE OF  
BACHELOR OF VOCATION  
IN**

**SOFTWARE DEVELOPMENT**



**SUBMITTED BY**

**Chetan Harde  
Prajwal Dhore  
Ritesh Kalarkar**

**UNDER THE GUIDANCE OF**

**Asst. Prof. A.P. Ramteke**

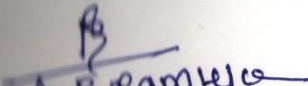
**DEPARTMENT OF BACHELOR OF VOCATION  
BHIWAPUR MAHAVIDYALAYA  
BHIWAPUR  
2022-2023**


**DEPARTMENT OF BACHELOR OF VOCATION  
BHIWAPUR MAHAVIDYALAYA  
BHIWAPUR, NAGPUR**

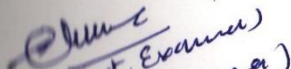



**CERTIFICATE**

This is to certify that the project work entitled Phishing Sites Detection, is a bonafide work done by Chetan Harde, Prajwal Dhore And Ritesh Kalarkar in the Software Development section of the Bachelor of Vocation, Bhiwapur Mahavidyalaya, Bhiwapur, Nagpur, in partial fulfillment of the requirement for the award of Bachelor of Vocation in Software Development.

  
A. P. Ramkulkarni  
Project Guide

  
Principal  
Bhiwapur Mahavidyalaya  
Bhiwapur.  
PRINCIPAL  
Bhiwapur Mahavidyalaya  
Bhiwapur, Dist. Nagpur

  
(Dr. S. K. Sharma)

  
Internal Ex  
(Dr. R. K. Dhurwadkar)

## ACKNOWLEDGEMENT

I wish to express my deepest sense of gratitude and obligation to my revered teacher and guide Asst. Prof. A.P. Ramteke, Software Development, Department of Bachelor of Vocation, Bhiwapur Mahavidyalaya, Bhiwapur, Nagpur for his/her inspirational guidance, suggestions, constructive criticism through out my graduate studies. I relied heavily on his professional judgment and encouragement, which benefited me immensely in carrying out this project.

I also express my sincere gratitude to Dr. Jobi George, Principal, Bhiwapur Mahavidyalaya, Bhiwapur, Nagpur, for his encouragement, and immense co-operation during my graduate studies at Bhiwapur Mahavidyalaya.

I wish to express my gratitude to my parents for sparing me to undertake this research project without any hindrances.

Chetan Harde *CH Harde*  
Prajwal Dhore *P.J. Dhore*  
Ritesh Kalarkar *R.G. Kalarkar*

**BHIWAPUR MAHAVIDYALAYA  
BHIWAPUR, NAGPUR**

**DECLARATION**

This Project work entitled Phishing Sites Detection Platform is my own work carried out under the guidance of Asst. Prof. Ashiwini Ramteke Assistant Professor in Bachelor of Vocation, Bhiwapur Mahavidyalaya, Bhiwapur, Nagpur. This work in the same form or in any other form is not submitted by me or by anyone else for the award of any degree.

Chetan Harde

Charde

Prajwal Dhore

P. J. Dhore

CERTIFICATE

Ritesh Kalarkar

R. U. Kalarkar

This is to certify that the Project work entitled Phishing Sites Detection is the bonafide work done by Chetan Harde, Prajwal Dhore And ritesh Kalarkar is submitted to Bhiwapur Mahavidyalaya, Bhiwapur, Nagpur, for the partial fulfillment of the requirements for the degree of Bachelor of Vocation in Software Development.



## **ABSTRACT**

Online phishing is one of the most epidemic crime schemes of the modern Internet. A common countermeasure involves checking URLs against blacklists of known phishing websites, which are traditionally compiled based on manual verification, and is inefficient. Thus, as the Internet scale grows, automatic URL detection is increasingly important to provide timely protection to end users. In this thesis, we propose an effective and flexible malicious URL detection system with a rich set of features reflecting diverse characteristics of phishing webpages and their hosting platforms, including features that are hard to forge by a miscreant. Using Random Forests algorithm, our system enjoys the benefit of both high detection power and low error rates. Based on our knowledge, this is the first study to conduct such a large-scale websites/URLs scanning and classification experiments taking advantage of distributed vantage points for feature collection. Experiment results demonstrate that our system can be utilized for automatic construction of blacklists by a blacklist provider.

# INDEX

CHPT NO.	TITLE	PAGE NO.
	<b>ACKNOWLEDGEMENT</b>	<b>iv</b>
	<b>ABSTRACT</b>	<b>v</b>
<b>1.</b>	<b>INTRODUCTION</b>	<b>2</b>
	1.1 Objectives	<b>3</b>
<b>2.</b>	<b>LITERATURE REVIEW</b>	<b>4</b>
	2.1 Problem Definition	<b>5</b>
	2.2 Proposed System	<b>6</b>
	2.3 Proposed Work	<b>7</b>
	2.4 Problem Scope	<b>8</b>
<b>3.</b>	<b>SYSTEM REQUIREMENT SPECIFICATION</b>	<b>9</b>
	3.1 Software Specification	<b>10</b>
	3.1.1 PyCharm	<b>10</b>
	3.1.2 Dataset	<b>10</b>
	3.2 Minimum Software and Hardware	<b>11</b>
	3.2.1 Minimum Software	<b>11</b>
	3.2.2 Minimum Hardware	<b>11</b>
<b>4.</b>	<b>SYSTEM DESIGN</b>	<b>12</b>
	4.1 Flowchart Diagram	<b>13</b>
	4.2 Creating a Phishing Dataset	<b>14</b>
	4.3 Implemented Features	<b>15</b>
	4.3.1 URL Based	
	4.4 DNS Based	<b>16</b>
	4.5 HTML Based	<b>17</b>

<b>CHPT NO.</b>	<b>TITLE</b>	<b>PAGE NO.</b>
<b>5.</b>	<b>IMPLEMENTATION</b>	<b>18</b>
	5.1 Sample Code	19
	5.2 Screenshots	27
<b>6.</b>	<b>ADVANTAGES</b>	<b>30</b>
<b>7.</b>	<b>APPLICATIONS</b>	<b>32</b>
<b>8.</b>	<b>CONCLUSION</b>	<b>34</b>
	9.1 Work Done	35
	9.2 Future Enhancement	36
<b>9.</b>	<b>REFERENCES</b>	<b>37</b>

## LIST OF FIGURES

<b>Figure No.</b>	<b>Caption / Title</b>	<b>Page No.</b>
4.1.1	App Flowchart	18
4.1.2	Database Flow Diagram	19
5.2.1	Screenshot 1	27
5.2.2	Screenshot 2	27
5.2.3	Screenshot 3	28



**1.**

# **INTRODUCTION**

# INTRODUCTION

Online phishing is one of the most epidemic crime schemes of the modern Internet. A common countermeasure involves checking URLs against blacklists of known phishing websites, which are traditionally compiled based on manual verification, and is inefficient. Thus, as the Internet scale grows, automatic URL detection is increasingly important to provide timely protection to end users. In this thesis, we propose an effective and flexible malicious URL detection system with a rich set of features reflecting diverse characteristics of phishing webpages and their hosting platforms, including features that are hard to forge by a miscreant. Using Random Forests algorithm, our system enjoys the benefit of both high detection power and low error rates. Based on our knowledge, this is the first study to conduct such a large-scale websites/URLs scanning and classification experiments taking advantage of distributed vantage points for feature collection. Experiment results demonstrate that our system can be utilized for automatic construction of blacklists by a blacklist provider.

When a client tries to connect with a server through HTTPS, a TLS handshake is performed. As the first step, the server needs to send over an X.509 certificate signed by a CA. Afterward, this certificate is used by the client to identify and authenticate the server against the X.509 trust chain until the root CA of the certificate is found in the so-called “root certificate store” of the client’s computer.

## 1.1 Objective

The project's objectives are as follows:

- To study various automatic phishing detection methods
- To identify the appropriate machine learning techniques and define a solution using the selected method
- To select an appropriate dataset for the problem statement
- To apply appropriate algorithms to achieve the solution to phishing attacks
- Trying to get unsuspecting users to give up their money, credentials or privacy is a particularly insidious form of social engineering that can have disastrous effects on people's lives.
- The lure is what entices the user to click on a link. It can be advertising a way to get easy money, obtain an illicit product, or a warning that a user's account has been compromised or blocked in some fashion.
- The hook is often a website that is designed to mimic a legitimate website of a reputable organization such as a bank or other financial institution.
- The hook is used to trick the user into entering and submitting their credentials such as user-name, password, credit card number, etc.
- The catch is when the user has submitted private information and the malicious owner of the website collects and uses this information to exploit the user and his account.
- Phishing attacks continue to be of persistent and critical concern to users, online businesses, and financial institutions.
- A phishing website lures users into divulging their sensitive information such as passwords, pin numbers, personal information, and credit card numbers, and uses such information for financial gains.
- According to current industry estimates, the annual financial losses due to phishing attacks across different economies surpasses \$3 billion

## **2. LITERATURE REVIEW**

## LITERATURE REVIEW

Phishing attacks classify as social engineering attack. In this kind of attack, the adversary does not necessarily look for a vulnerability in the system but also, looks for unaware users to lure them. For example, an attacker creates a webpage similar to a login page of a well-known email provider and sends the link to the users and asks them to log in. In this example, there is not any security concern relates to the Email provider. If the end-user does not aware of the potential threats, they may be fooled by the attacker. During last decade, different researchers tried to come up with different approaches. From a higher perspective, we categorize all of these efforts in two major categories. In the first category, we discuss the approaches that try to address the problem in a human-based manner. The approaches in this category increase the knowledge of end-users and help them to make a good decision when they face a suspicious websites. In the second category, we study the software-based approaches. In this approach, different techniques adapt to distinguish between legitimate websites and phishing ones and takes them down without considering end-users. The result of this category may also be fed to the first category to help end-users.

Our work in this thesis focuses mainly on detecting phishing websites with machine learning. There has been quite some effort regarding similar topics such as malicious domain blacklisting and email spam filtering. Furthermore, it is increasingly popular to utilize machine learning in these areas. Existing malicious websites detection approaches<sup>16–28</sup> can be mainly divided into two categories based on the features leveraged: static feature based approaches<sup>16–23</sup> and dynamic feature based approaches <sup>24–28</sup>. Static feature based approaches<sup>16–23</sup> rely on features extracted from the URL, page content, HTML DOM structure, domain-based information (such as WHOIS and DNS records) and so on. Alternatively, dynamic feature based solutions <sup>24–28</sup> primarily focus on analyzing behaviors captured when the page is loaded and rendered, or investigating system logs when some scripts are executed. In this thesis, we concentrate on exploiting static features.

## 2.1 Problem Definition

The problem is derived after making a thorough observation and study about the method of classification of phishing websites that makes use of machine learning techniques. We must design a system that should allow us to:

- Accurately and efficiently classify the websites into legitimate or phishing.
- Time consumed for detection should be less and should be cost effective.

We focus on the problem of determining if a target website is a phishing one or not, based on the information provided on the website. We consider the standard definitions of a phishing website from literature. Typically, the content of a phishing website is textually and visually similar to some legitimate website. We focus on characterizing the nature of such websites using only the information from the website and training a machine learning classifier to distinguish between phishing and legitimate websites.

Currently, a lot of existing tools, encapsulated in browsers, search engines or applications, such as Safe Browsing from Google<sup>9</sup> and SmartScreen<sup>10</sup> from Microsoft, try to inform a user that a specific URL the user is about to visit has been identified as unsafe or malicious. This is realized by matching the URL being visited with blacklists constructed by the security community. Those blacklists are accumulated using various techniques, ranging from user reporting to web crawlers with site content analysis to automatic classification based on heuristics or machine learning classifiers. However, many malicious websites can still sneak through such protection systems, which can be the consequence of a number of reasons:

1. The website is too new and thus has not been scanned or analyzed by any mechanisms yet.
2. The website has been incorrectly analyzed, either due to the imperfection of mechanisms or the countermeasures against detection taken by the attackers, e.g. abusing the legal short URL services.

## 2.2 Proposed System

The state-of-the-art machine learning approaches for phishing detection can be broadly classified as email, content, and URL based. The email-based approaches focus on analyzing emails based on various features. However, there has been the considerable evolution of phishing emails against such approaches, which makes them inadequate for current day context. This is shown by the relatively high rate of success of spear phishing emails attacks compared to other phishing methods. The content-based approaches perform in-depth analysis of content and build classifiers to detect phishing websites. These works rely on features extracted from the page content as well as from third-party services like search engines, and DNS servers. However, these approaches are not efficient due to a large number of training features and the dependence on third-party servers.

Using third-party servers violates user privacy by revealing the user's browsing history. More importantly, several features used in these approaches, like URL related features, do not accurately model the phishing phenomenon. Furthermore, in most of these approaches, except there is a critical issue of using biased datasets and the design of features that seem to work well for such datasets. Approaches like examine DOM content of the pages looking for the similarity of attacks. But, with the advent of newer attacks like that closely mimic legitimate websites, such approaches will be ineffective. The URL-based approaches analyze various features based on the target URL such as length of the URL, page rank of the URL, number of dots in the URL, presence of special characters, hostname features like IP address, domain age, DNS properties, and geographic properties, among other features. While the intuition in these approaches is sound, i.e., the URL is a good indicator of phishing attacks, the structural changes of modern-day URLs negates several lexical features identified by these approaches. For instance, these days, the URLs generated by websites like Google and Amazon, are long and contain many non-alphabetic characters, which dilute the lexical similarity of legitimate URLs. For this reason, the URL based approaches inadvertently tend to be biased towards the datasets being used and are likely to be ineffective in the future. A few hybrid detection mechanisms combine content and URL features, but suffer from the same problems as described. The work in also discusses features based on the Fully Qualified Domain Name of the phishing website. However, their approach depends on the results of search engines and incurs a significant delay.



## 2.3 Proposed Work

Machine learning algorithms have been proven to have the ability to discover complex correlation among different data items of similar nature. Many algorithms consist of two steps: learning and testing. In the learning step, the algorithms try to learn from supporting examples and in the testing phase, the researchers evaluate the accuracy of the algorithms. Attackers often use email to send out phishing URLs to the victim. Consequently, detecting potentially dangerous emails helps lead to prevent users to be caught in the phishing website. There is a wide literature on automating detection for phishing email by looking at the context of the email. used 16 features to detect phishing email. While they use email messages as a source to extract the features, we only focus on the website itself rather than how the attacker tries to tempt the users. described an approach based on URL classification using statistical methods to discover the lexical and host-based properties of malicious web site URLs.

They use lexical properties of URLs and registration, hosting, and geographical information of the corresponding hosts to classify malicious web pages at a larger scale. These methods are able to learn highly predictive models by extracting and automatically analyzing tens of thousands of features potentially indicative of suspicious URLs. The resulting classifiers obtain 95-99% accuracy, detecting large numbers of malicious websites by just using their URLs. However, their approach requires a large feature set and extracts host information with the help of third-party servers. We discussed why using URL-based features and third-party services lead to a biased dataset. Provided an overview of nine different machine learning techniques, including Support Vector Machine, Random forests, Neural Networks, Naive Bayes, and Bayesian Additive Regression Trees. They analyzed the accuracy of each classifier on the dataset, a state of the art dataset, and achieved a maximum accuracy of 91.34% using AdaBoost. They used a wide range of classifiers but based on adaptive nature of these attacks and not using an updating dataset cannot guarantee the resiliency of the solution.

## **2.4 Problem Scope**

The system will get HTML source code and URL of input webpage first. URL features normally just check internal and external links from HTML source code based on domain name. In HTML source code, there normally are four types of features that will be investigated and extracted, namely login forms, hyperlinks, CSS and JavaScript, and web identity features. The system will extract a different set of features into a common feature vector. Then, it will train, test, and validate with a specific classifier such as a random forest classifier. A content-based framework, uses features based on capturing various characteristics of legitimate web applications as well as their underlying web infrastructures. focus on the fundamental characteristics of phishing web sites and decompose the classification task for a phishing web site into URL classifier and content-based classifier. Their classifier does not need periodic retraining. Additionally, propose to extract features from URLs and webpage links to detect phishing website. Also analyze the hyperlinks found in the HTML source code of the website. Propose a stacking model to detect phishing webpages using URL and HTML features.

# **3. SYSTEM REQUIREMENT SPECIFICATION**

## **SYSTEM REQUIREMENT SPECIFICATION**

### **3.1.1 PyCharm**

PyCharm is the most popular IDE used for Python scripting language. This chapter will give you an introduction to PyCharm and explains its features.

PyCharm offers some of the best features to its users and developers in the following aspects –

- Code completion and inspection
- Advanced debugging
- Support for web programming and frameworks such as Django and Flask

### **3.1.2 Dataset**

Dataset in Python is mostly used for manipulation of Gifs and other custom data which frames the entire dataset as per requirement. It helps in maintaining the order and simplifying the complex data structure for further manipulation or enhancement. Dataset in any format is mostly used for many other necessities that streamline the process. Dataset in Python has a lot of significance and is mostly used for dealing with a huge amount of data. These datasets have a certain resemblance with the packages present as part of Python 3.6 and more. Python datasets consist of dataset object which in turn comprises metadata as part of the dataset. Querying to these datasets may include dataset objects to return the required index based on rows and columns. The dataset object comes into the picture when the data gets loaded initially that also comprise the metadata consisting of other important information.

# MINIMUM SOFTWARE REQUIREMENTS

## 3.2.1 Minimum Software Requirements

- PyCharm
- Database: Datasets

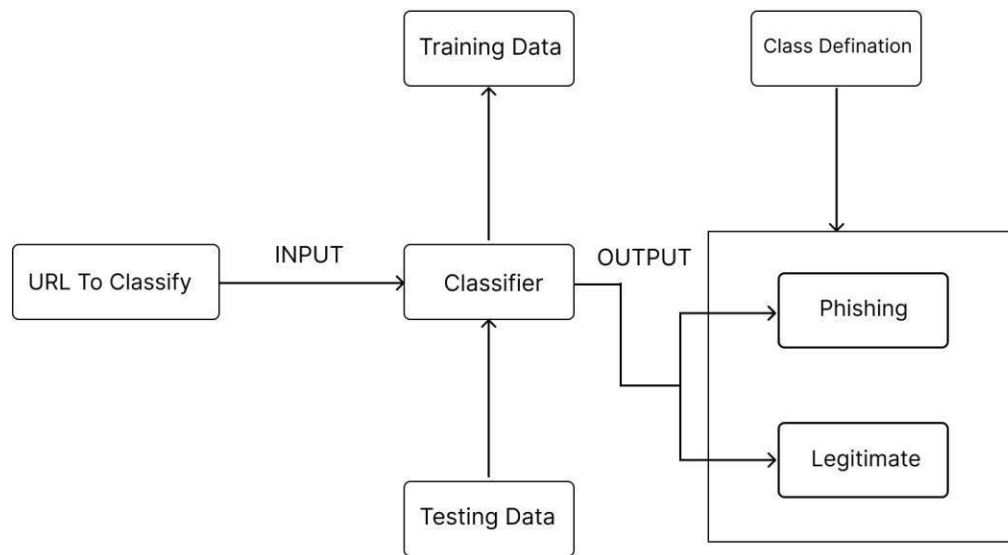
## 3.2.2 Language Used

- Machine Learning
- Python

## **4. System Design**

## SYSTEM DESIGN

### 4.1 Flowchart Diagram



### 4.2 Creating a Phishing Dataset

The dataset based on features of websites on the internet quickly become out of date and stale. We built a framework that can address this problem. Using that, it is possible to add/remove a feature to/from the dataset. In addition, the user can redo the extracting step to get the updated values for currently defined features. In the initial attempt, we use features that defined, but we implement them in the Python. To create our dataset, we scanned the top 3000 sites in the Alexa.com database and 3000 online phishing websites obtained from phishtank.com. We made two assumptions here: first, all of the websites gotten from Alexa.com are legitimate websites. We believe this to be a valid assumption because of the ephemeral nature of phishing websites, they tend to pop in and out of existence (as is evidenced by the short domain registration times) to evade being blocked or tagged as phishing.



The top sites ranked in Alexa.com must be popular and have been around for a longer period of time to attain this ranking. Second, we assumed that websites found on the Phishtank.com were phishing websites. PhishTank.com incorporates a community of registered users who report sites as phishing. Each member is ranked by the community and builds a good reputation by correctly reporting if a website is phishing or not. Since it is a very well-known repository for phishing websites, we can trust its decision for labeling a website as a phishing one.

### **4.3 Implemented Features**

We used 29 different features to create their dataset and we used their definitions to create our own dataset. These features can be categorized into five categories: URL based, DNS based, External statistics, HTML based, and JavaScript based.

#### **4.3.1 URL Based**

URL based features are based on some aspect of the URL of the website. Attackers try to use the URL to deceive users by obfuscating it in some fashion. For example, URLs that have an IP address, an 'at' symbol (@), double slash, contain a prefix or suffix are all methods employed to disguise a URL. Other notable methods are the length of URL, whether the website has a subdomain, uses a shortening service or uses a non-standard port.

**1. Having IP Address:** If an IP address is used as an alternative of the domain name in the URL, such as "http://125.98.3.123/fake.html", users can be sure that someone is trying to steal their personal information. In a Python script, we checked that if the website URL is in the form of an IP, we assume it as a phishing website otherwise it is legitimate.

**2. URL Length:** To ensure the accuracy of our study, we calculated the length of URLs in the data set and produced an average URL length. The results showed that if the length of the URL is greater than or equal 54 characters then the URL classified as phishing. By reviewing our dataset, we were able to find 1220 URLs whose lengths equal to 54 or more which constitute 48.8% of the total dataset size.

**3. Shortening Service:** URL shortening is a method on the web in which a URL may be made considerably smaller in length and still lead to the same webpage. This is accomplished by means of an "HTTP Redirect" on a domain name that is short, which links to the webpage that has a long URL. For example, TinyURL is a service that makes the URL shorter. The URL like "http://portal.hud.ac.uk/" can be shortened to "bit.ly/19DXSk4" using this service. If it used TinyURL, we will assume it as a phishing, otherwise, it is a legitimate website.

**4. Having At (@) Symbol:** A URL that contains a "@" symbol is not trusted as the browser generally ignores everything proceeding the "@". If the URL contains the "@" sign we marked it as phishing.

**5. Double Slash Redirecting:** URLs that contain "://" are marked as phishing as the double slash is used to redirect users to another site. Phishing URLs employ this method to hide their real URL. An example is <http://www.colostate.edu/http://www.phishing.com>.

**6. Prefix Suffix:** The dash symbol is rarely used in legitimate URLs. Phishers tend to add prefixes or suffixes separated by (-) to the domain name so that users feel that they are dealing with a legitimate webpage. For example, http://www.Confirme-paypal.com/. In our framework, we check whether that website uses a "-" in the name of URL or not. If it is used, we assume it as a phishing website.

**7. Having Subdomain:** Let us assume we have the following link: http://www.hud.ac.uk/students/. A domain name might include the countrycode top-level domains (ccTLD), which in our example is "UK". The "ac" part is shorthand for "academic", the combined "ac.uk" is called a second-level domain (SLD) and "hud" is the actual name of the domain. To produce a rule for extracting this feature, we first have to omit the (www.) from the URL which is, in fact, a subdomain in itself. Then, we have to remove the (ccTLD) if it exists. Finally, we count the remaining dots. If the number of dots is greater than one, then the URL is classified as "Suspicious" since it has one subdomain. However, if the dots are greater than two, it is classified as "Phishing" since it will have multiple subdomains. Otherwise, if the URL has no subdomains, we will assign "Legitimate" to the feature. We calculated the number of dots in a URL. If it is more than, we classify as phishing otherwise it is a legitimate website.

**8. Unusual Port:** Most legitimate websites use ports 80 for unencrypted traffic and port 443 for encrypted traffic. We mark the sites that use other ports as phishing.

#### **4.4 DNS Based**

DNS based features use information of the domain such as when the domain was first registered and how long the registration is valid.

**1. Domain Update Date:** This feature gets data of "Update Field" from WHOIS. This field demonstrates the latest time that domain owner updated the DNS record on the WHOIS database. The legitimate websites updated their information on the WHOIS database more often than the phishing website. If the updated date is less than half of a year, we mark this site as legitimate.

**2. HTTPS Token:** Phishing URLs will often try to make it look like the URL uses HTTPS. They will include HTTPS as part of the URL, for example, `http://https-colostate.edu`. We mark this URL as phishing.

**3. Age of Domain:** This feature can be extracted from WHOIS database. Most phishing websites live for a short period of time. By reviewing our dataset, we find that the minimum age of the legitimate domain is 6 months. Rule: If the age of domain is greater than 6 months, we will assume it as legitimate otherwise we will assume it as phishing.

**4. DNS Record:** This feature can be extracted from WHOIS database. For phishing sites, either the claimed identity is not recognized by the WHOIS database or the record of the host-name is not founded. If the DNS record is empty or not found then the website is classified as phishing, otherwise, it will classify as legitimate. We implement a Python script which gets DNS information from `www.WHOISXMLAPI.com` and check if the DNS record is empty or not. 3.2.3

#### **4.5 HTML Based**

The HTML served by a website contains many valuable features used to determine if the site is phishing or not. Examples of these features include whether the website has a favicon and if the images and JavaScript have the same source URL as the serving website.

Other HTML based features are whether the site implements iFrames, how many links point outside the serving domain, etc.

**1. Favicon:** A favicon is a graphic image (icon) associated with a specific webpage. Many existing user agents such as graphical browsers and newsreaders show favicon as a visual reminder of the website identity in the address bar. If the favicon is loaded from a domain other than that shown in the address bar, then the webpage is likely to be considered a phishing website. For this attribute, we checked the HTML code of each website and found where the Favicon is loading from. If it is loaded from a foreign domain, we assume that website is a phishing.

**2. Request URL:** This feature examines whether the external objects contained within a webpage such as images, videos and sounds are loaded from another domain. In legitimate webpages, the webpage address and most of the objects embedded within the webpage are obtained the same domain. We implemented a Python script which looks at all of the addresses and marks them as domain-inside or domain-outside. If more than half of addresses are domain-outside, we will mark the site as phishing otherwise it is a legitimate one.

**3. URL of Anchor:** This feature looks at the links in the website. If the links in the website point to a domain different from the domain of the website more than 50% of the time, then the site is marked as phishing.

**4. Links in Tags:** This feature looks at the domain in the tags of the header such as

# **5. IMPLEMENTATION**

## IMPLEMENTATION

### 5.1 Sample Code

#### Admin Page Code

```
def domainAge(domain_name):
    creation_date = domain_name.creation_date
    expiration_date = domain_name.expiration_date
    if (isinstance(creation_date,str) or isinstance(expiration_date,str)):
        try:
            creation_date = datetime.strptime(creation_date,'%Y-%m-%d')
            expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
        except:
            return 1
    if ((expiration_date is None) or (creation_date is None)):
        return 1
    elif ((type(expiration_date) is list) or (type(creation_date) is list)):
        return 1
    else:
        ageofdomain = abs((expiration_date - creation_date).days)
        if ((ageofdomain/30) < 6):
            age = 1
        else:
            age = 0
    return age

# 14.End time of domain: The difference between termination time and current time
(Domain_End)
def domainEnd(domain_name):
    expiration_date = domain_name.expiration_date
    if isinstance(expiration_date,str):
        try:
            expiration_date = datetime.strptime(expiration_date,"%Y-%m-%d")
        except:
            return 1
    if (expiration_date is None):
        return 1
    elif (type(expiration_date) is list):
        return 1
    else:
        today = datetime.now()
        end = abs((expiration_date - today).days)
        if ((end/30) < 6):
            end = 0
        else:
```

```

# importing required packages for this section
import requests

# IFrame Redirection (iFrame)
def iframe(response):
    if response == "":
        return 1
    else:
        if re.findall(r"<iframe>|<frameBorder>]", response.text):
            return 0
        else:
            return 1

# Checks the effect of mouse over on status bar (Mouse_Over)
def mouseOver(response):
    if response == "":
        return 1
    else:
        if re.findall("<script>.+onmouseover.+</script>", response.text):
            return 1
        else:
            return 0

# Checks the status of the right click attribute (Right_Click)
def rightClick(response):
    if response == "":
        return 1
    else:
        if re.findall(r"event.button ?== ?2", response.text):
            return 0
        else:
            return 1

# Checks the number of forwardings (Web_Forwards)
def forwarding(response):
    if response == "":
        return 1
    else:
        if len(response.history) <= 2:
            return 0
        else:
            return 1

```



```

# Function to extract features
def featureExtraction(url):
    features = []
    getDomain(url)
    # Address bar based features (10)
    features.append(havingIP(url))
    features.append(haveAtSign(url))
    features.append(getLength(url))
    features.append(getDepth(url))
    features.append(redirection(url))
    features.append(httpDomain(url))
    features.append(tinyURL(url))
    features.append(prefixSuffix(url))

    # Domain based features (4)
    dns = 0
    try:
        domain_name = whois.whois(urlparse(url).netloc)
    except:
        dns = 1

    features.append(dns)
    #features.append(web_traffic(url))
    features.append(1 if dns == 1 else domainAge(domain_name))
    features.append(1 if dns == 1 else domainEnd(domain_name))

    # HTML & Javascript based features (4)
    try:
        response = requests.get(url)
    except:
        response = ""
    features.append(iframe(response))
    features.append(mouseOver(response))
    features.append(rightClick(response))
    features.append(forwarding(response))
    #features.append(la)

    return features
'''
d='http://www.abxchvina.cn/'
fe = featureExtraction(d)
print(fe)

feature =[]
for i in range(0, len(data)):
    print(data[i])
    feature.append(featureExtraction(data[i],l[i]))

```

```

#converting the list to dataframe
feature_names = ['Have_IP', 'Have_At', 'URL_Length', 'URL_Depth', 'Redirection',
                 'https_Domain', 'TinyURL', 'Prefix/Suffix', 'DNS_Record',
                 'Domain_Age', 'Domain_End', 'iFrame', 'Mouse_Over', 'Right_Click',
                 'Web_Forwards', 'Label']

urls = pd.DataFrame(feature, columns= feature_names)

urls.to_csv("", index= False)

from flask import Flask, render_template, request
import dbn as d
import feature_extraction as fe

app = Flask(__name_)

@app.route('/', methods=["GET", "POST"])
def get_bot_response():
    if request.method == "POST":
        data = request.form.get('msg')
        features = []

        features.append(fe.featureExtraction(data))

        finalOutput_DBN, reconstructedOutput_DBN = d.dbn.dbn_output(features)

        yhat = d.clf.predict(finalOutput_DBN)

        if (yhat == 1):
            return render_template("phishing.html")
        else:
            return render_template("legitimate.html")
    return render_template("index.html")

if __name__ == '__main__':
    app.run()

train_index = range(0, len(train_X))
test_index = range(len(train_X), len(train_X)+len(test_X))

print(train_index)
print(test_index)

```

```

train_X = pd.DataFrame(data=train_X, index=train_index)
train_Y = pd.Series(data=train_Y, index=train_index)

test_X = pd.DataFrame(data=test_X, index=test_index)
test_Y = pd.Series(data=test_Y, index=test_index)

print(train_X.describe())

class RBM(object):

    def __init__(self, input_size, output_size,
                 learning_rate, epochs, batchsize):
        # Define hyperparameters
        self._input_size = input_size
        self._output_size = output_size
        self.learning_rate = learning_rate
        self.epochs = epochs
        self.batchsize = batchsize

        # Initialize weights and biases using zero matrices
        self.w = np.zeros([input_size, output_size], dtype=np.float32)
        self.hb = np.zeros([output_size], dtype=np.float32)
        self.vb = np.zeros([input_size], dtype=np.float32)

    # forward pass, where h is the hidden layer and v is the visible layer
    def prob_h_given_v(self, visible, w, hb):
        return tf.nn.sigmoid(tf.matmul(visible, w) + hb)

    # backward pass
    def prob_v_given_h(self, hidden, w, vb):
        return tf.nn.sigmoid(tf.matmul(hidden, tf.transpose(w)) + vb)

    # sampling function
    def sample_prob(self, probs):
        return tf.nn.relu(tf.sign(probs - tf.random.uniform(tf.shape(probs))))

    def train(self, X):
        tf.compat.v1.disable_eager_execution()
        _w = tf.compat.v1.placeholder(tf.float32, [self._input_size, self._output_size])
        _hb = tf.compat.v1.placeholder(tf.float32, [self._output_size])
        _vb = tf.compat.v1.placeholder(tf.float32, [self._input_size])

        prv_w = np.zeros([self._input_size, self._output_size], dtype=np.float32)
        prv_hb = np.zeros([self._output_size], dtype=np.float32)
        prv_vb = np.zeros([self._input_size], dtype=np.float32)

```

```

cur_w = np.zeros([self._input_size, self._output_size], dtype=np.float32)
cur_hb = np.zeros([self._output_size], dtype=np.float32)
cur_vb = np.zeros([self._input_size], dtype=np.float32)

v0 = tf.compat.v1.placeholder(tf.float32, [None, self._input_size])

# v0 = tf.keras.Input(shape=[None, self._input_size], dtype=tf.float32)
h0 = self.sample_prob(self.prob_h_given_v(v0, _w, _hb))
v1 = self.sample_prob(self.prob_v_given_h(h0, _w, _vb))
h1 = self.prob_h_given_v(v1, _w, _hb)
# To update the weights, we perform constrastive divergence.
positive_grad = tf.matmul(tf.transpose(v0), h0)
negative_grad = tf.matmul(tf.transpose(v1), h1)

update_w = _w + self.learning_rate * (positive_grad - negative_grad) /
tf.cast(tf.shape(v0)[0], float)
update_vb = _vb + self.learning_rate * tf.reduce_mean(v0 - v1, 0)
update_hb = _hb + self.learning_rate * tf.reduce_mean(h0 - h1, 0)
# We also define the error as the MSE
err = tf.reduce_mean(tf.square(v0 - v1))

error_list = []

"""Once we call sess.run, we can feed in batches of data to begin the training.
During the training, forward and backward passes will be made, and the RBM
will update weights based on how the generated data compares to the original input.
We will print the reconstruction error from each epoch"""
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    for epoch in range(self.epochs):
        for start, end in zip(range(0, len(X), self.batchsize), range(self.batchsize, len(X),
self.batchsize)):
            batch = X[start:end]
            cur_w = sess.run(update_w, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
            cur_hb = sess.run(update_hb, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
            cur_vb = sess.run(update_vb, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
            prv_w = cur_w
            prv_hb = cur_hb
            prv_vb = cur_vb
            error = sess.run(err, feed_dict={v0: X, _w: cur_w, _vb: cur_vb, _hb: cur_hb})
            print('Epoch: %d' % epoch, 'reconstruction error: %f' % error)
            error_list.append(error)
        self.w = prv_w
        self.hb = prv_hb

```

```

self.vb = prv_vb
return error_list

# function to generate new images from the generative model that the RBM has learned
def rbm_output(self, X):

input_X = tf.constant(X)
    _w = tf.constant(self.w)
    _hb = tf.constant(self.hb)
    _vb = tf.constant(self.vb)
    out = tf.nn.sigmoid(tf.matmul(input_X, _w) + _hb)
    hiddenGen = self.sample_prob(self.prob_h_given_v(input_X, _w, _hb))
    visibleGen = self.sample_prob(self.prob_v_given_h(hiddenGen, _w, _vb))
    with tf.compat.v1.Session() as sess:
        sess.run(tf.compat.v1.global_variables_initializer())
        return sess.run(out), sess.run(visibleGen), sess.run(hiddenGen)

inputX = np.array(train_X)
inputX = inputX.astype(np.float32)

# Create list to hold our RBMs
rbm_list = []

# Define the parameters of the RBMs we will train
rbm_list.append(RBM(15, 12, 1.0, 50, 100))
rbm_list.append(RBM(12, 8, 1.0, 50, 100))
rbm_list.append(RBM(8, 5, 1.0, 50, 100))

outputList = []
error_list = []

# For each RBM in our list
for i in range(0, len(rbm_list)):
    print('RBM', i + 1)
    # Train a new one
    rbm = rbm_list[i]
    err = rbm.train(inputX)
    error_list.append(err)
    outputX, reconstructedX, hiddenX = rbm.rbm_output(inputX)
    outputList.append(outputX)
    inputX = hiddenX
'''
i = 1
for err in error_list:

```

```

plt.xlabel("Epoch")
plt.ylabel("Reconstruction Error")
plt.show()
i += 1

class DBN(object):
    def __init__(self, original_input_size, input_size, output_size,
                 learning_rate, epochs, batchsize, rbmOne, rbmTwo, rbmThree):
        # Define hyperparameters
        self._original_input_size = original_input_size
        self._input_size = input_size
        self._output_size = output_size
        self.learning_rate = learning_rate
        with tf.compat.v1.Session() as sess:
            sess.run(tf.compat.v1.global_variables_initializer())
            for epoch in range(self.epochs):
                for start, end in zip(range(0, len(X), self.batchsize), range(self.batchsize, len(X),
self.batchsize)):
                    batch = X[start:end]
                    cur_w = sess.run(update_w, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
                    cur_hb = sess.run(update_hb, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
                    cur_vb = sess.run(update_vb, feed_dict={v0: batch, _w: prv_w, _hb: prv_hb, _vb:
prv_vb})
                    prv_w = cur_w
                    prv_hb = cur_hb
                    prv_vb = cur_vb
                    error = sess.run(err, feed_dict={v0: X, _w: cur_w, _vb: cur_vb, _hb: cur_hb})
                    print('Epoch: %d' % epoch, 'reconstruction error: %f' % error)
                    error_list.append(error)
                self.w = prv_w
                self.hb = prv_hb
                self.vb = prv_vb
            return error_list

# function to generate new images from the generative model that the RBM has learned
def rbm_output(self, X):

```

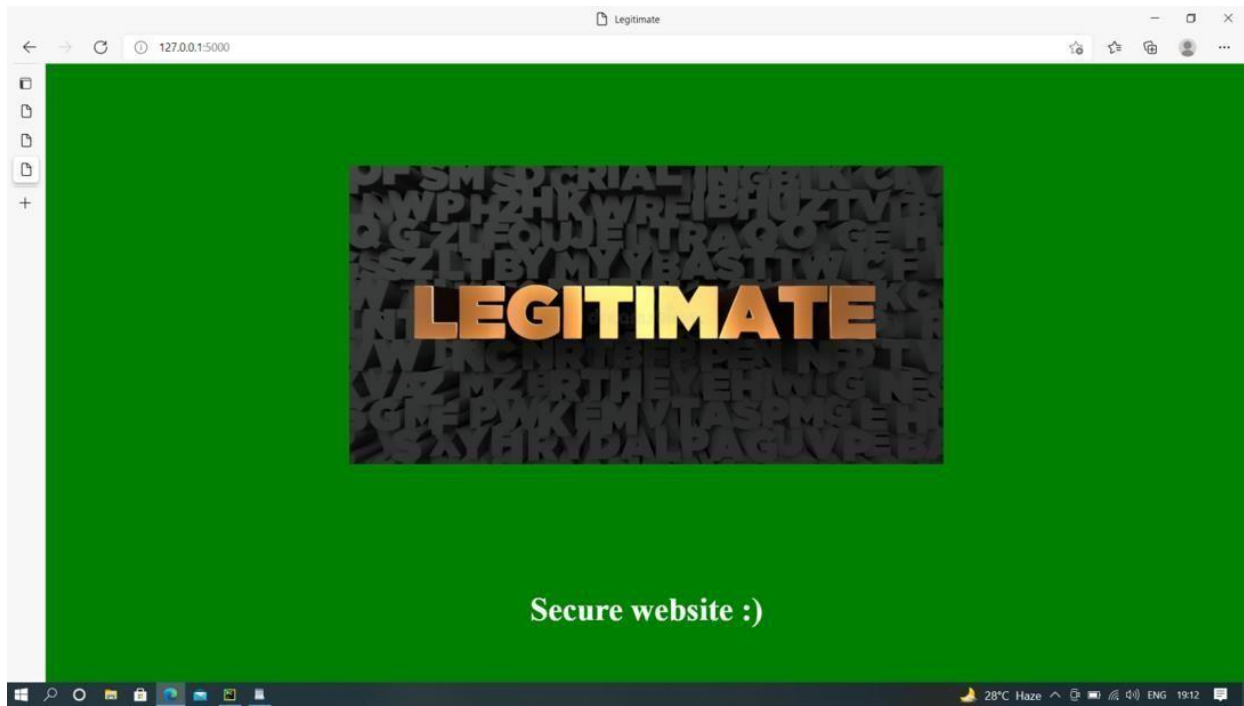
## 5.2 Screenshots



Fig.5.2.1 Homepage



Fig.5.2.2 Phishing Detection



**Fig.5.2.3 Trusted Site**



## **6. ADVANTAGES**

## ADVANTAGES

Improved Approach: Domain Name Based Features With the Fresh-Phish framework, we learned several lessons and were able to get a better understanding of the features that need to be used for phishing detection. For instance, one lesson we learned is that the problem of phishing cannot be viewed in a statistical way and needs to consider the intent of the attackers for fooling the users. Therefore, the features should be finally chosen for phishing detection, should reflect this intuition. Another important lesson we learned was there is a strong correlation between the domain name and the nature of the websites. Based on these observations, we now describe our solution. Our work is the first solution to be entirely focused on the domain name of the phishing website. In our work, the domain name is the string before the top-level domain identifier, e.g., for the URL google.co.uk, the domain name is google. We only concern ourselves with examining the landing page of this website and with the information that can be extracted from this page without the help of third-party servers, search engines or DNS servers. Our approach is based on the intuition that the domain name of the phishing websites is a key indicator of a phishing attack. We design several features that are based on the domain name and train a machine learning classifier based on sample data. The trained classifier is used to test a suspicious website against these features. In the following, we describe the key challenges in our proposed approach and our solutions to these challenges.

## **7. APPLICATIONS**

## APPLICATIONS

Phishing campaigns usually work based on a three-part scheme. The first part is using email or some form of communication to lure users and redirect them to a phishing page. The fake page closely mimics a trustworthy site. Finally, the user enters their information, which is captured by the adversary. In our proposed approach, instead of defining features that group the phishing websites together, we relate a suspicious phishing website to its target and define features based on the similarity of a given suspicious website and its target. Reach more customers.

Phishing attacks are constantly evolving and the cyber world is hit by new types of attacks often. Hence a particular detection approach or algorithm cannot be tagged as the best one giving exact results. Through the literature survey, it is evidently visible that Random Forest gives better results in most scenarios. But then the performance of each algorithm varies depending on the dataset used, train-test split ratio, feature selection techniques applied etc. Researchrs prefer to create machine learning models that perform phishing detection with best value for evaluation parameters and least training time. Therefore, the future works should focus on these aspects of phishing detection.

## **8. CONCLUSION**

## CONCLUSION

The goal of this thesis was to explore the possibility of using machine learning to detect phishing URLs without performing additional scraping of the URL itself. The first chapter overviewed the recent trends in phishing, listed the general types of internet-based phishing, and focused on describing the main techniques of creating a deceptive URL. The second chapter provided a brief high-level overview of the machine learning landscape as a whole. A description of the available data and the methods of its collections were presented in the third chapter. Chapter four outlined the basic principles upon which each of the selected machine learning algorithms works and highlighted their main strengths and weaknesses. The fifth chapter explains the methodology used for the conducted experiment. Chapter six informed on the process of the model training phase and the final choices of hyper parameters for the individual algorithms. Finally, the seventh chapter provided an evaluation of models' performance, summarized the results of the experiment, and discussed the limitations and possible improvements to the approach. The results of this experiment show that we are able to detect phishing URLs with a precision of 96% (for the positive class) and classify URL as benign or phishing with an accuracy of 95% using Random Forest algorithm. All of the other used algorithms performed rather similarly except for Naive Bayes, which showed slightly worse results. The primary output of this thesis is the in-depth description of the possible approach to tackling the phishing detection problem without any web scraping. The secondary output is a standalone machine learning project that allows for future replications or modifications of the performed experiment.

## **9. REFERENCES**

## REFERENCES

1. SHENG, Steve; WARDMAN, Brad; WARNER, Gary; CRANOR, Lorrie; HONG, Jason; ZHANG, Chengshan. An Empirical Analysis of Phishing Blacklists. 2009.
2. Phishing Activity Trends Report Q4 2018 [online]. APWG [visited on 2020-04-13]. Available from: [https://docs.apwg.org/reports/apwg\\_trends\\_report\\_q4\\_2018.pdf](https://docs.apwg.org/reports/apwg_trends_report_q4_2018.pdf).
3. GUARNIERI, Claudio. The Year of the Phish [online]. Nex [visited on 2020-04-12]. Available from: <https://nex.sx/blog/2019/12/15/the-year-of-the-phish.html>.
4. Phishing Activity Trends Report [online] [visited on 2020-04-13]. Available from: <https://apwg.org/trendsreports/>.
5. Uniform Resource Identifier (URI): Generic Syntax [online]. IETF [visited on 2020-04-18]. Available from: <https://tools.ietf.org/html/rfc3986>.
6. KOZA, John R.; BENNETT, Forrest H.; ANDRE, David; KEANE, Martin A. Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming. In: Artificial Intelligence in Design '96. 1996, pp. 151–170. ISBN 978-94-009-0279-4. Available also from: [https://doi.org/10.1007/978-94-009-0279-4\\_9](https://doi.org/10.1007/978-94-009-0279-4_9).
7. MITCHELL, Tom M. Machine Learning. In: McGraw-Hill, 1997, p. 2. ISBN 978-0-07-042807-2.
8. GERÓN, Aurelién. Hands-On Machine Learning with ScikitLearn, Keras, and TensorFlow. In: O'Reilly Media, Inc., 2017, chap. 1. ISBN 978-1-49-203264-9.
9. Transport Layer Security (TLS) Extensions [online]. IETF [visited on 2020-04-18]. Available from: <https://www.rfc-editor.org/info/rfc3546>.
10. GERÓN, Aurelién. Hands-On Machine Learning with ScikitLearn, Keras, and TensorFlow. In: O'Reilly Media, Inc., 2017, chap. 6. ISBN 978-1-49-203264-9