

UNIVERSITY OF TARTU
Institute of Computer Science
Cyber Security Curriculum

Ahmed Nafies Okasha Mohamed
A New Heuristic Based Phishing Detection Approach Utilizing Selenium Web-driver

Master's Thesis (30 ECTS)

Supervisor(s):

Dr. Olaf Manuel Maennel

Dr. Raimundas Matulevicius

A New Heuristic Based Phishing Detection Approach Utilizing Selenium Web-driver

Abstract

Phishing is a nontrivial problem involving deceptive emails and webpages that trick unsuspecting users into willingly revealing their confidential information. In this paper, we focus on detecting login phishing pages, pages that contain forms with email and password fields to allow for authorization to personal/restricted content. We present the design, implementation, and evaluation of our phishing detection tool “SeleniumPhishGuard”, a novel heuristic-based approach to detect phishing login pages. First, the finest existing technologies or techniques that have used similar heuristics we will be discussed and evaluated. The methodology introduced in our paper identifies fraudulent websites by submitting incorrect credentials and analyzing the response. We have also proposed a mechanism for analyzing the responses from server against the submissions of all those credentials to determine the legitimacy of a given website. The application was implemented in python programming language by utilizing Selenium web testing library, hence “Selenium” is used in the name of our tool. To test the application, a dataset from Alexa top 500 and Phishtank was collected. All pages with login forms from the Alexa 500 and Phishtank were analyzed. The application works with any authentication technologies which are based on exchange of credentials. Our current prototype is developed for sites supporting both HTTP and HTTPS authentication and accepting email and password pair as login credential. Our algorithm is developed as a separate module which in future can be integrated with browser plug-ins through an API. We also discuss the design and evaluation of several URL analysis techniques we utilized to reduce false positives and improve the overall performance. Our experiments show that SeleniumPhishGuard is excellent at detecting login phishing forms, correctly classifying approximately 96% of login phishing pages.

Keywords:

Phishing detection, Heuristics, URL analysis, Login pages, Selenium, DOM, White-list, Web security

CERCS: P170, Computer science, numerical analysis, systems, control

Uus heuristikal põhinev õngitsemise avastamine Selenium Webdriveriga

Lühikokkuvõte

Õngitsemine on oluline probleem, mis hõlmab endas petlike meilide ja veebilehtede kasutamist, tüsates pahaaimamatuid kasutajad vabatahtlikult avaldama konfidentsiaalset informatsiooni. Antud uurimustöö põhifookuseks on avastada õngitsemise veebilehti, mis kasutavad identifitseerimiseks meili ja salasõna, et pääseda ligi personaalsele või piiratud sisule. Töös esitletakse SeleniumPhishGuard rakenduse kasutusmugavust ning analüüsitakse selle uude heuristilise lähenemisega programmi võimalusi ja tulemusi õngitsemise lehekülgede tuvastamisel. Esmalt hinnatakse ning diskuteeritakse olemasolevate parimate tehnoloogiliste lahenduste ning meetodite üle, mis kasutavad sarnast heuristikat. Selles magistritöös on kasutatud meetodikat, mis identifitseerib võltsveebilehed, sisestades vormi vigased andmed ning analüüsides saadud vastust. Lisaks serverist saadud andmevahetusele pakume meetodikat, mis määrab veebilehe legitiimsuse teiste põhimõtete järgi. Rakendus on realiseeritud Pythoni programmeerimiskeele, kasutades Selenium veebi testimise raamatukogu. Sellest tulenevalt on ka programmi nimes viidatud Seleniumile. Rakenduse testimiseks on kasutatud Alexa top 500 ja Phistank andmebaase. Kõiki sisselogimise vormiga veebilehti Alexa 500 ja Phistank andmebaasides töödeldi ja analüüsiti kasutades antud rakendust. Rakendus töötab kõikide identifitseerimistehnoloogiatega, mis põhinevad isikuandmete vahendamisel. Praegune prototüüp on välja töötatud lehtedele, mis toetavad nii HTTP kui ka HTTPS autentimist ning aktsepteerivad isikuandmetena meili ja parooli. Algoritm on välja töötatud iseseisva moodulina ning tulevikus on võimalik seda integreerida veebilehitseja lisana läbi API. Lisaks olemasolevale meetodikale on hinnatud ja uuritud erinevate URL analüüsise tehnikaid, mida kasutati vale positiivse info vähendamiseks ning soorituse parandamiseks. Katsetused näitasid, et SeleniumPhishGuard rakendus on hiilgav tööriist avastamiseks õngitsemise vorme. Rakendus suutis tuvastada ligikaudu 96% sisselogimisega õngitsemislehtedest

Võtmesõnad:

õngitsemise avastamine, heuristika, URL analüüs, sisselogimise lehekülg, Selenium, DOM, valge-nimekiri, veebi turvalisus

CERCS: P170, Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Acknowledgments

I would like to thank Tallinn University of Technology and University of Tartu for the opportunity to study masters of cyber security with a tuition weaver. I also feel grateful to Swedbank for the internship opportunity where I grasped a lot of new skills that helped me create the tool discussed in this paper. As well, I am quite grateful to Skype Estonia for awarding me with the Skype award for outstanding students on my first year which has helped me during my financial hardship.

I am also grateful to my supervisor Dr. Olaf Manuel Maennel for his patience and support in guiding me through my research. I would also like to thank my dear girlfriend Maryna Kovalenko for her support and help with writing and reviewing my paper. As well, many thanks for Mari-Liis Ling for translating the abstract and title to Estonian language.

I would like to thank my friends for accepting nothing less than excellence from me. Last but not the least, I would like to thank my family: my parents and to my brothers and sister for supporting me spiritually throughout writing this thesis and my life in general.

Table of Contents

Abstract	2
Lühikokkuvõte	3
Acknowledgments	4
1 Introduction	10
1.1 Phishing Life-cycle	10
2 Background	13
2.1 History of Phishing	13
2.2 Significance of Phishing	13
2.3 Motives for Phishing	15
3 Literature Review	17
3.1 Existing Mitigation Methodologies	17
3.1.1 Blacklists	17
3.1.2 Visual Similarity	17
3.1.3 Machine Learning Approaches	18
3.1.4 Phishing Detection by Heuristics	20
4 Methodology and Implementation	24
4.1 Methodology	24
4.2 URL And Domain Analysis Module	25
4.3 Phishing Identification Module	27
5 Data Collection Process	31
5.1 Phishing Pages' Scrapper	31
5.2 Legitimate Pages' Scrapper	32
5.3 Data Sets	33
6 Evaluation Metrics	35
7 Development Environment, Tools and System Usage	37
7.1 Development Environment	37
7.2 Testing Environment	38
7.3 System Usage	38
8 Results	41
8.1 Phishing Identification Module Results:	41
8.2 Comparison Between Related Work and Our Application:	45
9 Conclusion	46
Future Work: Enhancements	46
10 References	49

Appendix	51
I. Glossary.....	51
II. Previous Work (Discontinued).....	54
Previous Methodology:	54
URL and DNS Matching Module	55
III. Phishtank Scrapper.....	59
Phishtank Scrapper.....	59
IV. Alexa Top 500 Scrapper.....	60
Alexa Scrapper Code.....	60
V. URL and DNS Matching Module	63
URL and DNS Matching Module code:.....	63
VI. Phishing Identification Module.....	64
Phishing Identification Module Code:	64
VII. URL and Domain Analysis Module.....	69
VIII. Database Functions	70
IX. Index.....	72
X. License	73

Table of Figures

Figure 1. Phishing life cycle.....	10
Figure 2. Login page with email and password fields	11
Figure 3. Unique Phishing sites detected October 2015 - March 2016 from APWG Trends report	14
Figure 4. Phishing reports received January - March 2016 APWG Trends report	14
Figure 5. Number of unique phishing sites detected worldwide from 3rd quarter 2013 to 2nd quarter 2016	15
Figure 6. Methodology.....	24
Figure 7. URL and Domain analysis module activity diagram.....	27
Figure 8. HTML elements with input tags example	28
Figure 9. example of XPath of a password field	28
Figure 10. Sample of email list used for testing.....	28
Figure 11. Phishing Identification module sequence diagram	29
Figure 12. Phishing Identification module activity diagram.....	30
Figure 13. Phishtank page layout	31
Figure 14. Sample of Data scrapped from Phishtank website in JSON format	32
Figure 15. Phishtank scrapper	32
Figure 16. Google Scrapper	33
Figure 17. Sample output of the URL list exported by Alexa scrapper	33
Figure 18. Selenium user prompt	38
Figure 19. Testing real-time logs	39
Figure 20. Selenium filling Facebook login form	39
Figure 21. Page response after form submission	39
Figure 22. Test successfully finished	40
Figure 23. Grafana visualizing False positives and True Positives	40
Figure 24. Graph showing the threshold effect on accuracy.....	42
Figure 25. Comparison between system results with and without URL and Domain Analysis Module	44
Figure 26. Script based login form.....	47
Figure 27. Form with captcha	48
Figure 28. Preliminarily methodology	55
Figure 29. URL and DNS module activity diagram	56
Figure 30. Sample output of URL and DNS module	57
Figure 31. Sample of IP mismatch for legitimate domains.....	57
Figure 32. Sample of false positives by the URL and DNS matching module.....	58

Table of Tables

Table 1. Heuristics for the URL and domain analysis module	26
Table 2. Dataset (1) - extracted on 21/02/2017	34
Table 3. Dataset (2) – extracted on 16/03/2017	34
Table 4. Dataset (3) – extracted on 22/03/2017	34
Table 5. Dataset (4) - extracted on 02/04/2017	34
Table 6. Languages associated with datasets	41
Table 7. Heuristics weights	42
Table 8. Results of phishing identification module	43
Table 9. Phishing Identification with URL analysis Module results	44
Table 10. Table of comparison.....	45
Table 11. DNS and URL matching module results.....	56

Table of Equations

Equation 1. Simplified Classifier Score Function	25
Equation 2. Heuristic weight calculation function	26
Equation 3. True Positive Rate (TPR).....	35
Equation 4. False Positive Rate (FPR)	35
Equation 5. False Negative rate(FNR)	35
Equation 6. True Negative Rate (TNR)	36
Equation 7. Overall Accuracy (A)	36

1 Introduction

Detection and prevention of phishing attacks are big challenges as the phisher performs attacks to bypass the existing anti-phishing techniques. An educated and experienced user may still fall this attack. The attacker makes a fake yet similar webpage by copying or making a little change in the legitimate page, so that an internet user will not be able to differentiate between the real and the phished one. One of the effective solutions to prevent a phishing attacks is to integrate security features with the web browser to raise alerts whenever a phishing site is accessed by an internet user. Generally, web browsers provide security against phishing attacks with the help of list-based solutions.

The list-based solutions contain either black-list or white-list. These solutions match the requested domain with the domains present in a list and take suitable decision. A combination of technical experts and security software verify when a new domain needs to be added in this list. Security software checks the various features of a webpage to verify its legitimacy.

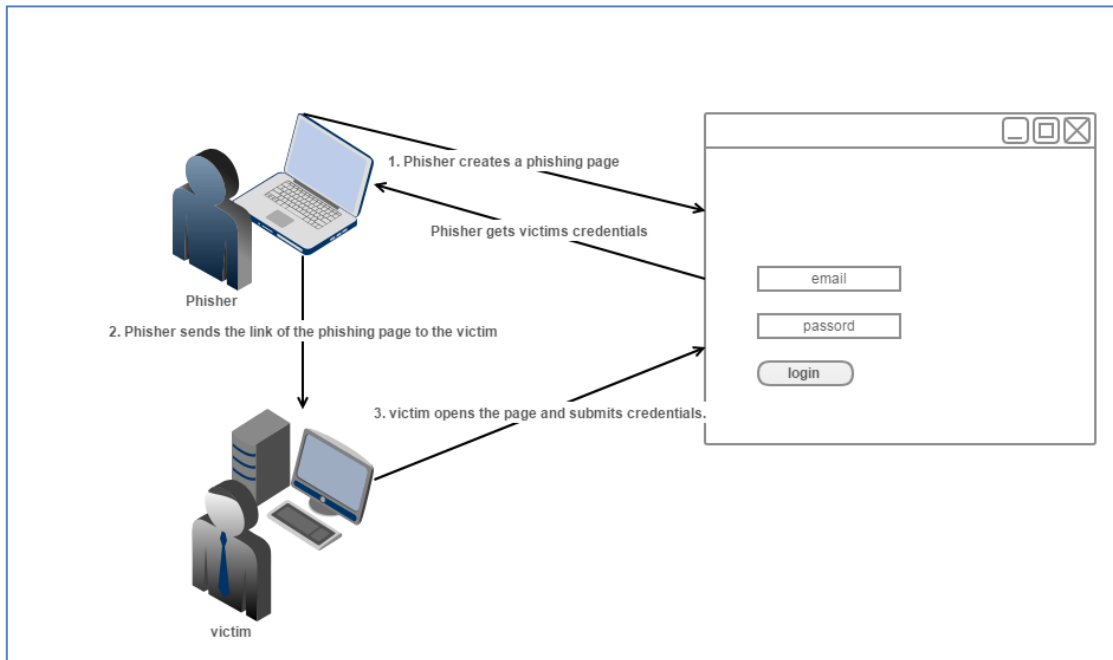


Figure 1. Phishing life cycle

1.1 Phishing Life-cycle

1. The phisher clones the content from the website of a legitimate company or a bank and generates a phishing website. The phisher tries to keep the visual similarity of the phishing website to the corresponding legitimate website to trick more users (see Figure 1).
2. The phisher sends an email including the link of the phishing website to it to his victims. In the case of spear phishing, a mail is sent to individual targeted victims.
3. When the victim opens the email, and visits the phishing website, the phishing website prompts the victim to insert private data, for example, if the phisher copycats the

phishing website of a famous organization, then the users of organization are expected to willingly reveal their private credentials to the phishing website.

4. The phisher receives private data of the victim via the phishing website and utilizes this data for financial or some other benefits which will be discussed in detail in the background section.

In order to provide dataset for our tests, Alexa database, which contains the top 500 visited pages, was used to collect legitimate pages and Phishtank was used to collect phishing pages. During analysis of dataset (1) shown in Table 2 in the data collection section. We observe that 56% of the Alexa database domains contains pages with a login form. Phishers mimic the pages with login forms to steal credentials for financial gain or identity theft. The significance of phishing websites is shown as 60% of the dataset of live phishing pages collected in this research contain login forms.

In this paper, we focus on finding new techniques to detect login phishing pages. Login pages are pages which contain a form with an email/user password combination input fields as shown in Figure 2. Once the form is submitted the credentials are transferred to the backend servers to provide authentication. This login page can be significantly important if it is the gateway to a bank account, online wallet or just an email which can be used as an identity as described before.

The nature of phishing relies on the naivety of computer users in accordance to their dealings with electronic communication channels, for instance web browsing, e-mail and so on. Due to its nature, it is a nontrivial task to be solved eternally and hence, the entire existing technology can only attempt to diminish the impact of phishing assaults. Phishing is a language-based attack, which utilizes communication channels to convey content in human readable languages. Computers have an enormous complication in precisely understanding human readable natural languages. Phisher introduces new phishing techniques that cannot be either detected by humans or software. Hence, the challenge here is that the mitigation techniques must always be improved.

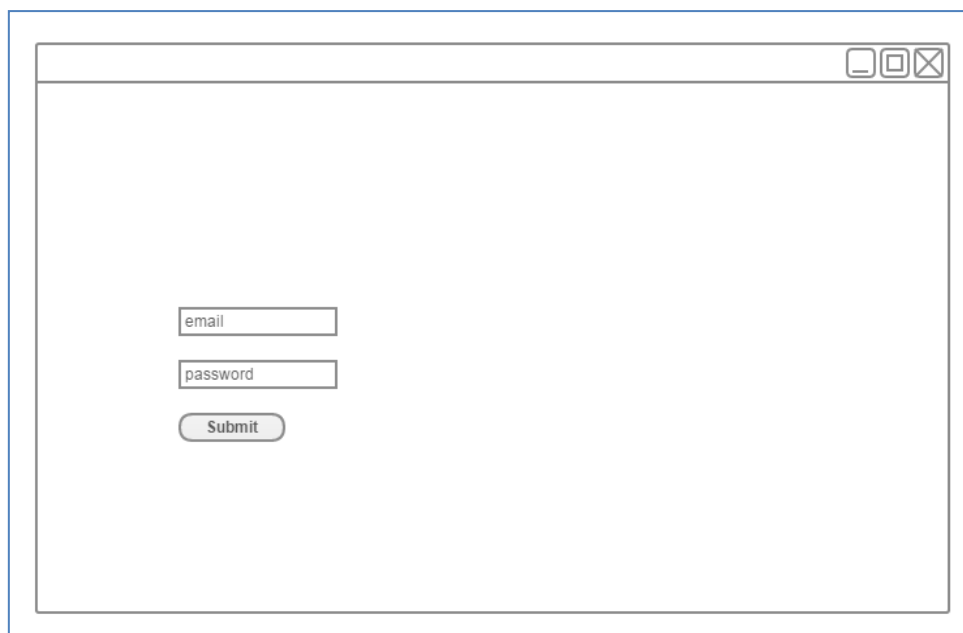


Figure 2. Login page with email and password fields

One of the biggest challenges in the security field is the zero-hour phishing attack. A zero-hour vulnerability denotes to a gaping hole in anti-phishing technique that is still unknown to the vendor. This security gaping hole is then exploited by attackers before the vendor detects the vulnerability and rushes to fix it. Embedded objects: The legitimate webpage is downloaded to create the phishing webpage which mimic a genuine webpage only in appearance. Hackers obfuscate the address bar by using an image or script which makes the victim trust that they are viewing the legitimate website. Phishers similarly utilize the embedded objects (flash, images, etc.) instead of HTML codes to avoid phishing detection techniques.

PhishGuard is a phishing detection tool optimized to detect phishing login pages by injecting fake credentials and analyzing the HTTP response from the server. The main challenge for PhishGuard was classifying HTTPS pages. Normal HTTP pages reply to request with what is called HTTP server status codes. For example, 200 status code is returned when successful and 404 is returned when page requested is not found. There are codes that are special for HTTP authentication, if user is authenticated, the server returns 200 ok, if no successful authentication then server code 401 will be returned. In case of HTTPSs, the behavior is different. The server usually return status code 200 ok, In case of authentication success or failure.

In this paper, we describe some common characteristics of recent web phishing attacks and the recent effective heuristics used by detection tools. Moreover, we are proposing a new methodology to detect phishing login webpages using heuristics similar to PhishGuard, SeleniumPhishGuard uses domain name, URL, link, and tests the login form shown in “Figure 2” to evaluate the likelihood that a given page is part of a phishing attack. For example, a page with a URL such as “http://suspicious.URL.com@127.0.0.1/phish.asp”.

The URL will be tested against some heuristics rules specified in this paper and the will be given a certain score. If the score is higher than the threshold then the URL will be classified as phishing. If the score is lower than if the page contains login form then fake credentials will be injected and form will be submitted for an n number of times. SeleniumPhishGuard will classify the page depending on the response. If the credentials are rejected then the website will be classified as legitimate, if not, then the website will be classified as phishing.

2 Background

2.1 History of Phishing

Phishing according to the APWG – (Anti phishing working group) in the phishing activity trends report - 1st quarter of 2016, which was published on May 23, 2016 is a “criminal mechanism employing both social engineering and technical subterfuge to steal consumer’s personal identity data and financial account credentials. Social engineering schemes use spoofed e-mails purporting to be from legitimate businesses and agencies, designed to lead consumers to counterfeit websites that trick recipients into exposing financial data such as usernames and passwords. Technical phishing schemes plant crime-ware onto PCs to steal credentials directly, often using systems to intercept consumers online account user names and passwords. Moreover, phishing schemes corrupt local navigational infrastructures to misdirect consumers to counterfeit websites (or authentic websites through phisher-controlled proxies used to monitor and intercept consumers’ keystrokes)” [1].

Phishing scams commonly use spoofed websites and emails as decoy to prompt people to willingly hand over sensitive information such as login credentials or bank information. The term “phishing” is frequently used to express these ploys. Fishing is an activity to try catching fish by using bait. However, “ph.” used in place of the “f” in the spelling of the term as some of the first hackers were recognized as phreaks [1]. Phreaking refers to the examination, experimenting and learning of telecom systems. Furthermore, both Phreaks and hackers have always been closely linked. The “ph.” spelling was used to link phishing scams with these underground communities [1].

According to Phishing.org, January 2, 1996 was the earliest time that the term “phishing” was used. The state occurred in a UseNet newsgroup called alt.online-service.America-online. It is fitting that it was made there too; AOL or America Online is where the initial boost of what would become the major cybercriminal problem ever to occur. Back then when America Online (AOL) was the top provider of Internet access, enormous amount of people logged on to the service each day. In addition, its popularity made it a natural choice for those who had less than pure motives. From the start, hackers and others who traded pirated software used the service to communicate with one another [2].

Phishing has developed dramatically since its America online show day. Phishers started paying attention to online payment systems. Although the first attack in June 2001, which was on E-Gold, was not measured to be victorious, it was the building block to what came later. In the last quarter of 2003, phishers registered hundreds of domains that appear to be legitimate sites like eBay and PayPal. Phishers used adopted worm programs to spread spoofed emails to PayPal or eBay customers. Victims were redirected to spoofed sites and prompted to update their credentials, credit card information and other identifying information [2].

2.2 Significance of Phishing

The APWG or Anti-Phishing Working Group reported that there are more phishing attacks in the first quarter of the year 2016 as more than in any other quarter ever since it started tracking and reporting data in the year of 2004, according to the anti-cybercrime coalition's first quarter phishing activity trends report. In keeping with those statistics, the APWG reported that the amount of phishing websites it detected rose hysterically by 250 percent within the period of just October 2015 to March 2016 (see Figure 3) [1].

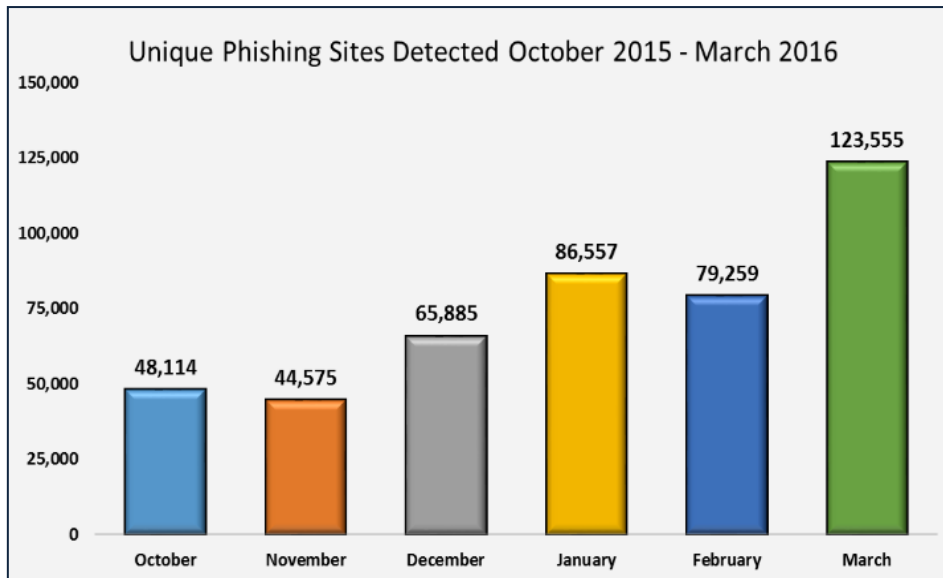


Figure 3. Unique Phishing sites detected October 2015 - March 2016 from APWG Trends report

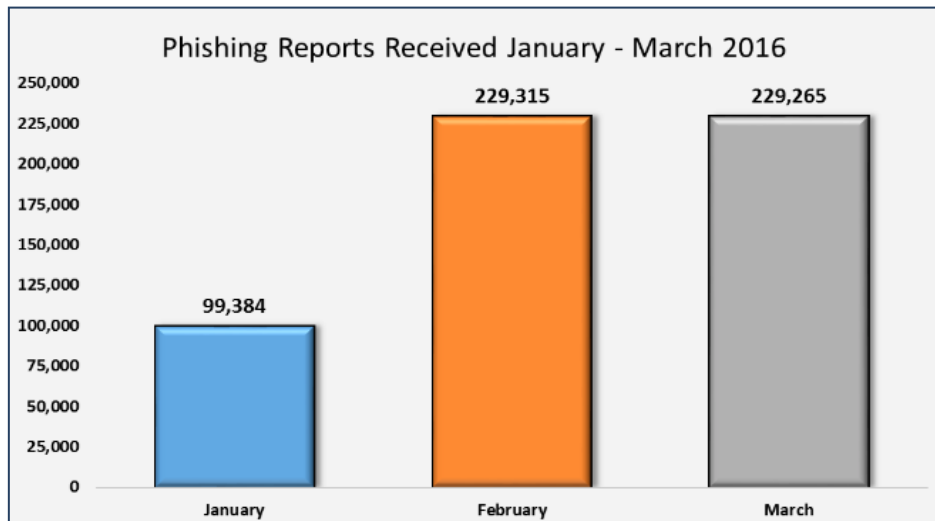


Figure 4. Phishing reports received January - March 2016 APWG Trends report

The amount of zero-day phishing reports submitted to APWG during last quarter of 2016 was 557,964. The number of unique phishing reports submitted to APWG saw a dramatic raise of almost 130,000 in precisely two months' period (See Figure 4) [1].

In order to have a broader perspective, statistics from Statistica.com which is online statistics, market research and business intelligence portal, shows a statistic that provides data on the amount of international unique phishing domain names as of the second quarter of 2016 (see Figure 5). As of the last tracked quarter, 466,065 unique phishing sites were detected, up from 289,371 zero-day sites in the preceding quarter [3].

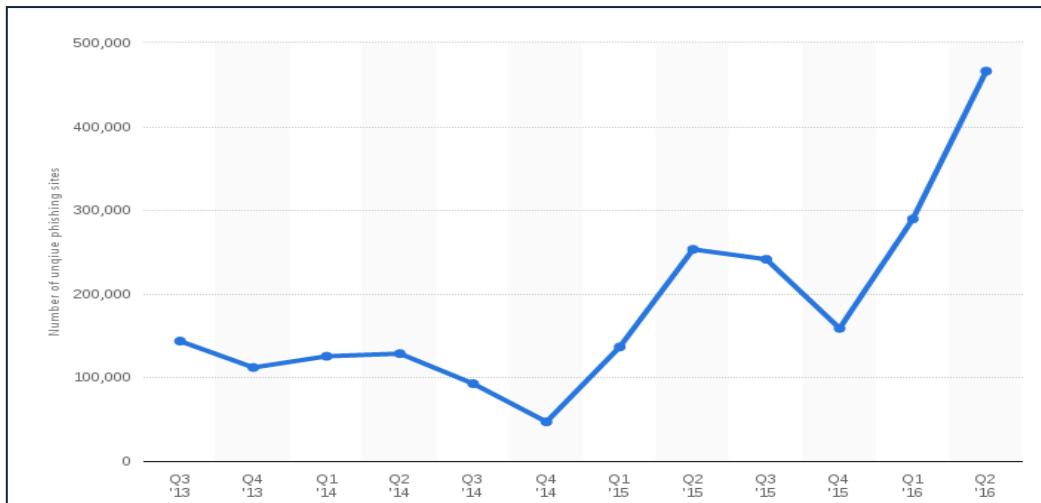


Figure 5. Number of unique phishing sites detected worldwide from 3rd quarter 2013 to 2nd quarter 2016

It is apparent from Statistica's graph above, that the general number of phishing websites. In fact, the number of unique zero-day phishing websites is increasing dramatically [3]. Around 80% of people who are exposed to phishing fall victims for it according to a study to CBS News when it teamed up with Intel Security. The test was intended to examine their capability to detect phishing emails designed to steal their information [4]. More than 19,000 individuals around the globe from 143 different countries have taken the test. Intel's test displayed 10 real emails sent to inboxes and extracted by analysts at McAfee Labs, which was division of Intel's Security. A handful of emails were legitimate correspondences from global companies, while many others were just phishing emails that look tremendously realistic and convincing. However, from all the 19,458 individuals who have taken the test, unfortunately the greater part, around 80% fell for no less than one of the phishing email from the ones they were presented. Only 3% has achieved an ideal score [4]. When compared to the earlier version of the test, where around 97% of participants opened at least one phishing email, 80% is not a striking decline, nevertheless certainly improved [5].

2.3 Motives for Phishing

According to S. Sharma et. al. [6], the primary motives behind phishing attacks, from an attacker's perspective are:

Financial Gain:

Sharma indicated that financial gain is the leading motive regarding phishing as different researches indicated that main victims of phishing attacks were the financial institutions. A phishing attack against any money related institution involves destroying the brand of the association. According to Shivangi, the widely-used technique is developing a phishing website page where phishers ask authorization to access the account details of a victim.

Identity Hiding:

Phishers steal identities and either commit a fraud related crime by means of these identities or sell them for financial gain to criminals who acquire and utilize stolen identities to hide their own.

Fame and Notoriety:

Peer recognition in this case is the primary motive where phishing attacks are initiated by persons who mainly want to gain recognition and acknowledgment among their colleagues. This is a tremendously psychologically driven aspect of phishing, wherein data is phished not for financial benefit, but rather just with the end goal of picking up consideration and greatness in the online group.

Malware Distribution:

This assault occurs by distributing malware by means of phishing messages that are sent in bulk and therefore, zombie networks are the most suitable to transmit large phishing assaults. These messages enclose malicious links which, when clicked by an inexperienced customer, results into malware spreading over the victim's machine.

Harvesting Passwords:

Phishers perform this raid using diverse methods. For instance, key loggers and additional malware like Man in the Browser (MITB) attack. Data gathered from client is either used once more for financial benefit, identity hiding, fraud or sold to attracted parties for fiscal gain.

3 Literature Review

3.1 Existing Mitigation Methodologies

The broader perspective to view the phishing problem mitigation techniques will provide us with only two options, end-user awareness and education where end-users are trained on how to correctly identify phishing to improve one's own detection level to appropriately identify and report phishing and the other solution where software is utilized to classify and identify phishing with a little to no human interaction. In this paper, the focus will be primarily on software solutions.

The phishing detection literature survey which was published in IEEE Communications Surveys & Tutorials in 2013[7] highlighted that there are classically 4 major software approaches to mitigate phishing by the aid of software:

- Blacklists
- Heuristics
- Visual similarity
- Machine learning

3.1.1 Blacklists

Phishing detection by blacklists is the oldest and most widely-used phishing detection mechanism till this very day. Typically, it is client-server application where blacklist database is hosted on the server and connected to a client application which queries the database whenever the user opens a URL. The major drawbacks of blacklists are privacy and inability to detect unique zero-hour phishing websites. However, blacklists are considered to have faster detection rate than heuristics, visual similarity and machine learning and have a lower false positives rate than heuristics [8]. Steve Sheng in his study called 'An Empirical Analysis of Phishing Blacklists' [8] where among his team, they collected 191 zero-hour phishing pages that were live in less than 30 minutes. Blacklists were considered unsuccessful when defending end-users primarily, as most of them detected less than 20% of phishing websites at zero-hour. The study [8] also highlights an enormous delay of 12 hours before 47% to 83% of phishing URLs were blacklisted. The common life time for 63% of phishing campaigns before it ends is two hours, which makes this enormous delay is a momentous concern [8].

3.1.2 Visual Similarity

In [9] and [10] phishing detections approaches based on heuristics check common properties of phishing sites such as unique keywords used in URLs or web pages to identify zero-day phishing websites. Nevertheless, these types of heuristics will be effortlessly bypassed by attackers once their mythology is exposed. Visual similarity-based detection techniques have been proposed to circumvent this limitation. Since phishing web pages must imitate victim sites, visual similarity between phishing sites and their target sites is alleged to be an inherent and not simply concealable property. However, these techniques require images of real target sites for detection. In [9] it was proposed to use a phishing detection mechanism based on visual similarity among phishing sites that imitate the same target website. It was claimed that Just by analyzing visual similarity among web pages without a previous information, the method automatically extracts 224 different web page layouts imitated by 2,262 phishing sites. However, it achieves a detection rate around 80 % while maintaining the false-positive rate to 17.5 %.

3.1.3 Machine Learning Approaches

Phishing detection by Machine learning approaches will be discussed in this section. It will not be covered in detail, but more of a comparative analysis of most existing approaches that uses heuristics similar to the ones that are used in this research.

a. *CANTINA+*

CANTINA+ is a proposed a page content-based anti-phishing technique which calculates webpage content's (TF-IDF) or term frequency-inverse document frequency [11]. *CANTINA+* is an upgraded version of *CANTINA* which as well utilizes extra features from URL, HTML DOM (Document object model), third party services, search engine and trained these features using SVM (Support vector machine) to detect phishing attack. True positive rate of *CANTINA+* is 92% and low false positive rate is 0.4% [12].

b. *Associative Classification data mining*

The approach proposed by Neda Abdelhamid et al [13] using Multi-Label Classifier based Associative Classification (MCAC) method for website phishing. Associate classification is successfully detecting phishing websites with high accuracy. Furthermore, MCAC is utilized to produce novel rules and it also improves its classifiers anticipation performance. Neda relies on various rules related to URL analysis, redirect, DNS records, Age of domain and website traffic According to Neda, The accuracy ranges between 94%-95%

c. *Classification mining techniques*

Maher Aburrous presented a methodology based on Classification Data Mining (DM) for detection on e-banking phishing websites. Aburrous has implemented six different classification algorithms (C4.5, JRip, PART, PRISM, CBA and MCAR) to measure the performance and accuracy on each of algorithm, however the downside of this algorithm is that the false positive rate is very high (13%) [14].

d. *SVM-based techniques to detect phishing URLs*

H Huang [15] et proposed an approach based on the URL based features. They have taken 23 features from URLs and train the system using SVM. System takes decision based on lexical and brand name features of URL by comparison with the top 10 brand name websites. It is claimed that this approach has an accuracy of 99% on average when tested with URLs downloaded from PhishTank database.

V. Ramanathan [16] presents a strong technique to detect phishing websites by means of semantic analysis a topic modeling technique, a natural language processing technique Latent Dirichlet Allocation, and AdaBoost used for classification. The mere advantage of using this methodology is that it is both device and language independent. The technique uses a web-crawler which utilizes Google's language translator to translate pages to English. Topic model is created by means of the translated contents of desktop and mobile clients.

In addition, the web-crawler impersonates a regular human behavior using the browser. The classifier for phishing websites is developed using distribution probabilities for the topics found as features using LDA or Latent Dirichlet Allocation and AdaBoost voting methodology. Tests were carried out on 47500 phishing websites and 52500 legitimate websites. Results have shown phishing detection accuracy of 99%.

Garera et al. [17], presented a methodology based on phishing URLs discussed the four different kinds of obfuscation techniques of phishing URLs.

I. Obfuscating the Host with an IP address.

Hostname is swapped with an IP address, and typically the party being phished is placed in the path. Currently the IP address expressed in decimal or hex rather than the dotted quad form.

II. Obfuscating the Host with another Domain.

URL's hostname contains a valid looking domain name; however, the path includes the party being phished. This type of assault often aims to mimic URLs accommodating a redirect so that it seems legit.

III. Domain unknown or misspelled.

In this case, there is no obvious connection to the association being phished or the domain name is mistyped.

IV. Obfuscating with large host names.

This type of assault uses the party being phished in the host however; it adds a long string of domain and words after the host name.

In the presented work, a range of URL features and suspicious keywords found in URL were extracted along with some addition and modification of this technique. The average accuracy for this technique was 97.31%.

Gowtham et al [18] proposed using heuristics on 15 extracted features from Webpages. Results were used as an input to a trained machine learning algorithm to identify phishing sites. Prior to deploying heuristics to these webpage's, two main classifying modules were utilized in this system. The first module checks site identifier and Webpages against a white-list. The second module is a Login Form Finder that extracts the HTML DOM from the page, arranges and divides web pages as legitimate when there are no login forms found. These modules used to decrease the unnecessary computations by the system and hence minimizing the rate of false positives without affecting the rate the false negatives. This technique identifies web pages with a 0.4% of false positive rate and 99.8% overall precision.

There are other more techniques regarding machine learning yet they are out of scope or the interest of this research since the heuristics or methodologies used by these machine learning techniques are not closely related to our tool.

3.1.4 Phishing Detection by Heuristics

In [7], Phishing detection by heuristics is defined as Software which is deployed on the server or client side to inspect payloads of different protocols via diverse algorithms. Protocols include HTTP, SMTP, POP3 or any arbitrary protocol. Moreover, Algorithms could be any method to identify or stop phishing assaults. Furthermore, Phishing heuristics are properties that are considered to exist in phishing assaults in real life, nevertheless these properties are not always assured to exist in such attacks. Hence, if a set of universal heuristic examinations are recognized, it might detect zero-hour phishing attacks, which is an advantage against blacklists. Since blacklists require exact matches to identify phishing websites, the precise same phishing attacks need to be examined first to blacklist them. Nevertheless, such widespread heuristics also carry the risk of misidentifying legitimate websites (False Positives). Recently, Global mail clients and web browsers are developed with phishing protection technologies, such as heuristic based detectors that help at identifying phishing attacks. Furthermore, the clients include but not limited to Internet Explorer, MS Outlook, Mozilla Firefox and Mozilla Thunderbird. In addition, phishing detection heuristics is included in Anti Viruses (i.e. claimAV¹) [7].

a. *Spoof Guard:*

Possibly one of the closest heuristic based techniques to the one used in this paper and thus will be discussed in detail. Spoof-Guard² a web browser add-on build by Stanford University, identifies HTTP/HTTPS based phishing attempts as a web browser plug-in, by measuring assured anomalies found in the HTML content against a defined threshold value. [19] The plug-in screens and filters a user's Internet activity, calculates a spoof index, and alerts the user if the index exceeds a certain level adjusted by the user. The current level of detection accuracy and precision may be adequate to assist unsophisticated web users.

Spoof-Guard analyses domain name, URL, link, and image checks to compute the likelihood that a current page is part of a phishing attack. Spoof-Guard as well utilizes user search history, such as if the user has visited the domain before and if the referring page was from an email site such as Gmail or Hotmail. Spoof Guard intercepts and computes user posts considering related history and the spoof index of an HTML form submitting page. After, it will examine post data 'user name' or 'email' and 'password' fields and compares posted data against previously entered passwords from different domains [19]. This technique alerts the user when sending his/her password to a site with a logo but outside the domain, for example. In addition, passwords matching is carried out using a cryptographically secure hash, and thus passwords in plaintext are by no means stored by Spoof-Guard [19].

Since distinct sites have special input field names for user ids and passwords, 20 usernames and 10 passwords combinations are predefined in Spoof-Guard. These combinations are utilized to detect sensitive data in the obtained post data structure. These predefined names are utilized by various major bank forms, commercial sites such as Amazon and eBay. Spoof-Guard at present cannot differentiate username and password combinations with a different input field names [19].

¹ www.claimav.net

² <https://crypto.stanford.edu/SpoofGuard/>

b. PhishGuard

The closest technique to the one proposed in this research. The proposed algorithm in [20] was considered as a novel algorithm at that time. The methodology's main target is to detect a phishing website by injecting fake credentials before the user inputs correct credentials in a login form of a website. In addition, it was also presented to utilize a mechanism for screening the server response against the submissions of these submitted credentials to decide if the web page is legitimate or not. Although the idea is nonspecific and works with any authentication technologies that are based on submission of any credentials, the current prototype is developed for sites utilizing HTTP Digest authentication and accepting user-id and password combination as credential. Furthermore, method is built within a browser as plug-in for Mozilla Firefox

The proposed methodology in [20] works as follows:

1. The user visits a page with a login form.
2. User submits his/her login credentials.
3. PhishGuard will trap the credentials and will send fake credentials instead for a n random number of times
4. If the page responded with '*HTTP 200 OK message*', then it means the page is a phishing page, and is simply returning fake authentication success messages.
5. If the page responded with '*HTTP 401 Unauthorized message*', then will possibly be:
 - i. Legitimate website.
 - ii. Phishing websites and blindly replies *HTTP 401 Unauthorized message*
6. To detect if the website is legitimate or not, PhishGuard sends the correct credentials to the website for the $n + 1$ time.
7. If the server responds with a '*HTTP 200 OK message*' after the login form submission,
 - i. Then the site is considered legitimate.
8. If the server responds with a '*HTTP 401 Unauthorized message*', then it leads to two possibilities:
 - i. The web page is a phishing page that indiscriminately replies with failure authentication messages. The drawback here is that the correct login credentials of the user were submitted to the phisher. Obviously, this method only prevents password theft for a subset of phishing websites.
 - ii. The user submitted the wrong password.
9. PhishGuard came up with an idea to guarantee that the submitted password is not a wrongly mistyped password by the user, PhishGuard stores password hashes and validates future login against it:

- i. If the hash of the password submitted matches any hash value previously stored by phish guard, then the password was correct, and
- ii. The site is considered as a phishing website.
- iii. If no match was found, then PhishGuard concludes the user mistyped the password, and an alert will be generated to correct the informing the user that the submitted was not correct.

The drawback of this methodology is when tested with secured site (i.e. HTTPS) using user-id/password as credential as that the response to authentication failure message is as same as “200 OK” along with a redirected page with appropriate information alerting user of authentication failure. In this paper, a methodology to overcome this problem will be proposed.

c. PhishWish

Although, PhishWish is used to detect spam emails, yet the approach is like the one used in this research, therefore it is covered in this section. In [21], only 11 heuristic rules were presented to determine whether an incoming email is a phishing message.

The presented solution aims toward providing:

- Far better protection against zero-hour attacks than blacklists while utilizing relatively negligible resources (11 rules)
- URL that fall within the email’s body and email headers are analyzed

The email is considered as phishing if:

1. If a URL is a page with a login form that is not a business’ real login page, the result is positive. PhishWish will utilize search engines and to get the business’ real login page.
2. If the email has HTML content, and the included URL uses (TLS) Transport Layer Security, whilst the authentic (HREF) attribute - Hypertext Reference does not employ TLS.
3. If the host-name part of a URL is an IP address.
4. If an organization’s name (e.g. Amazon, PayPal) is in the URL path but does not exist in the domain name.
5. If the HREF attribute contains a different domain name than the displayed domain name
6. If the SMTP header received does not contain the organization’s domain name
7. If a non-image URL’s domain part has obvious inconsistencies.
8. If deviations are found in WHOIS records from non-image URL’s domain part.

9. If deviations are found in image URL's domain part.

10. If deviations are found in WHOIS records of image URL's domain part

11. If the page is not accessible or down.

The weighted mean of all the 11 rules is calculated as the score which is used to foresee a class for the email depending on the score against a threshold. For example, let's assume that all rules have equal weights, if the score of a given e-mail is $\geq 50\%$ then it is predicted to be phishing, or legitimate if otherwise.

4 Methodology and Implementation

Our approach is to build a client-side application with automatic real-time phishing detection mechanism based on white-lists as shown in Figure 6. The application will inject fake credentials to login pages and check the response of the website. This approach is quite similar to the PhishGuard's [20] approach which was discussed in section 3 "literature view". Phishing login pages are designed to lure victims into willingly giving their credentials. However, Phishing websites has no information regarding the victim's real credentials. Hence, it is expected to have one of the following scenarios:

- a) Phishing website shows a success message.
- b) Phishing website redirects to another website.
- c) Phishing website shows a failure message.
- d) Phishing website shows the same login page again.

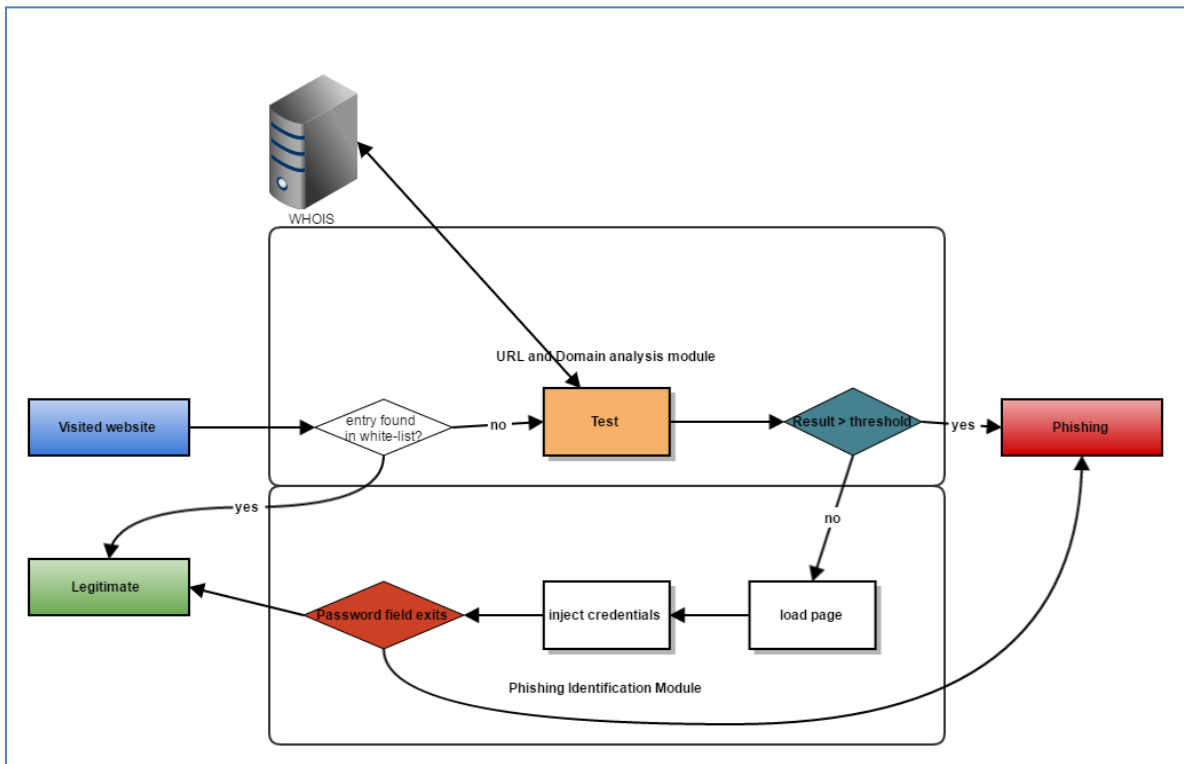


Figure 6. Methodology

4.1 Methodology

1. User visits a website as shown in Figure 6.
2. URL and Domain analysis module checks if the website is in the white-list.
3. If the domain name is found, then the website is legitimate.
4. If no domain is found in the white-list, the URL and Domain analysis module will check the following properties of the URL:
 - a. Domain creation date < 365 days

- b. Domain expiry date < 180 days
 - c. No domain exists in WHOIS
 - d. Number of Dots in the URL > 5
 - e. Special character “@” in the URL.
5. Each property will have a certain weight, for simplicity let’s assume all properties have equal weight of value 1.
 6. The sum weights will be computed, assume that all properties exist, then sum will range from 0 to 5 depending in which property exists.
 7. The sum of property weights will be compared with a specified threshold let’s assume 3 and if sum is greater than 3 then URL will be classified as phishing.
 8. The Phishing identification module will inject n number of fake email password combinations.
 - a. The phishing detection module will detect all the input text fields with type attribute (email) or name attribute (email, user, username, Id and userid).
 - b. The module will inject the input text fields with fake credentials and wait till page loads.
 - i. If the page loads with no password fields then it is considered as phishing.
 - ii. If the page redirects to another website then it is considered as phishing.
 - iii. If the page reloads with an input text field of type password, then the application will inject more fake credentials for n number of times.
 - iv. If the password field still exists after n number of trials, then the page will be considered as legitimate.

4.2 URL And Domain Analysis Module

The purpose of the URL and domain analysis module is to minimize the rate of false positives and reduce analysis time. The URL and domain analysis module act as a filter for URLs that are clearly not legitimate before testing with the phishing identification module. Our filter consists of heuristics presented in the Table 1 below. These heuristics were used by CANTINA [11].

There are many algorithms available to determine the best weights for the heuristics shown in the table below. However, for simplicity forward linear model will be used.

Equation 1. Simplified Classifier Score Function

$$C = th_x \left(\sum w_{e_i} \times h_{e_i} \right) \quad (1)$$

$h_{e_i} \rightarrow$ Heuristic variable, integer {0 or 1}

$we_i \rightarrow$ Weight of a certain heuristic, integer

$th_x \rightarrow$ threshold function where "x" is the threshold value , binary

$C \rightarrow$ Our classifier function, binary

Table 1. Heuristics for the URL and domain analysis module

Heuristic	Suspected phishing?
Domain creation date	≤ 365 days
Domain expiry date	≤ 180 days
'@' in URL	≥ 1
'-' in URL	≥ 1
Dots in URL	≥ 5
WHOIS	No entry for domain

The next step is to calculate the weight for each heuristic. Fundamentally, the more effective the heuristic, the higher the weight it acquires. Preferably, the heuristic with high weights have high accuracy in detecting phishing sites while also having a low false positive rate.

To measure the effect of a heuristic, the difference between true positives and false positives is calculated. This is a straightforward approach which is like the one used in another report on anti-phishing toolbars [22]. We calculate each weight proportionally, that is:

Equation 2. Heuristic weight calculation function

$$we_i = \text{round}\left(\frac{TPR_i - FPR_i}{10}\right) \quad (2)$$

Equations 1, 2 were used to determine the best weights for each heuristic. We used 100 phishing URLs chosen from PhishTank and 100 legitimate pages from Alexa top 500 database. The 200 URLs are different languages sites to ensure integrity. different threshold values were used for testing. The highest achieved accuracy 96.66% occurred when threshold value $x = 8$.

The URL analysis module checks a URL using the heuristics. The module will parse a given URL and extract the domain. WHOIS database will be queried using the extracted domain for the domain creation date and expiry date. Each heuristic rule will be checked and the sum of their weights will be calculated. The classifier will compare the sum with the threshold of value $x = 8$, $C = 1$ if $x > 8$ and in this case the page is classified as phishing and $C = 0$ if $x > 8$ and in this case, the page is classified as legitimate (see Figure 7).

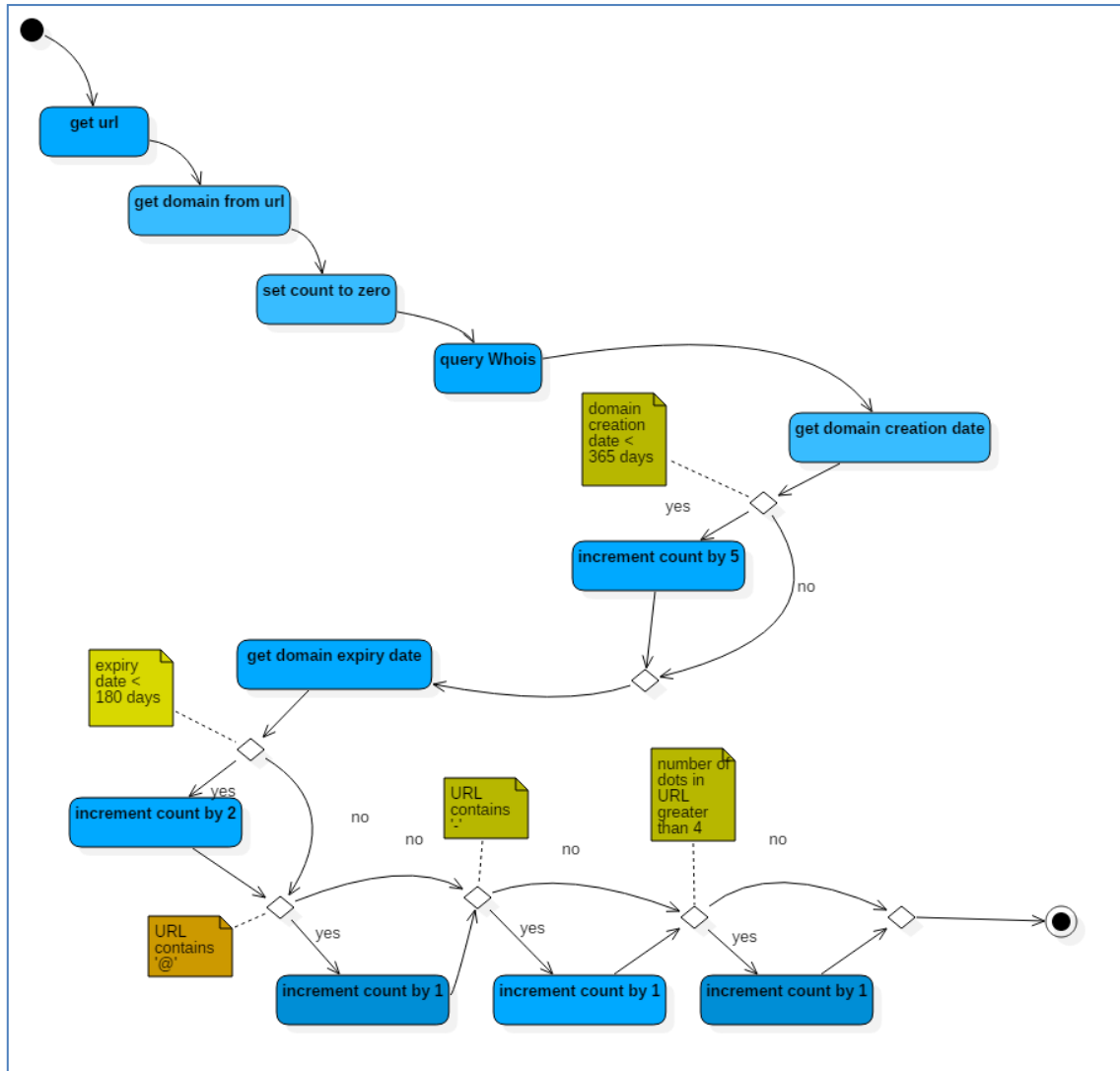


Figure 7. URL and Domain analysis module activity diagram

4.3 Phishing Identification Module

Webpages are being fully tested and classified by the phishing identification module. The phishing identification module tests login pages by filling the login fields with fake credentials multiple times and based on the response, the page will be classified as phishing or legitimate. The phishing detection module is written in Python and utilizes Selenium web driver. Selenium is used to simulate normal user behavior (button clicks, text insertion, form submission, drag and drop). In addition, Selenium renders the webpage by utilizing many browsers (Firefox, Chrome). This is useful since tests are performed on browsed fully rendered pages. The module will use Selenium to test a given URL. However, before Selenium starts with testing the webpage, the application will check if the domain is in the white-list by querying our stored white-list in MongoDB³ database. If the database has no entry for the current domain, it will continue testing the URL.

SeleniumPhishGuard in our case utilized Mozilla Firefox web browser for rendering and testing pages. After the webpage is requested and fully loaded, the web driver will start

³ <https://www.mongodb.com/>

searching for all fields with input tag name. For example, the Figure 8. contains an HTML two input element fields with attribute type email and password which we are looking for in a page.

```
<input type="text" type="email" name="email">  
<input type="text" type="email" name="password">
```

Figure 8. HTML elements with input tags example

In case if there are no fields with input tag name, the page is not considered for this test and the URL is logged to our logging database “InfluxDB⁴”. On the other hand, if the input tag name exists, the application will then filter and extract all input tag names with attribute type or name that contains ‘email’, ‘user’, ‘username’, ‘userid’ and ‘password’ using their XPath.

Here is an example of the XPath of HTML input tag names with type attribute set to password. Any password field in any webpage will have the same XPath (see Figure 9).

```
"//input[@type='password']"
```

Figure 9. example of XPath of a password field

In case if there is no password field or input field in general, the current page will be considered as page with no login form and will not eligible for this test. The application will use a list of hardcoded predefined emails (see Figure 10) and passwords to test the URL with the following list of fake emails and some generic random password.

```
emails = ['first.last@name.com', 'bla@bla.com', 'python@great.com', 'happy@best.com']
```

Figure 10. Sample of email list used for testing

As shown in Figure 11. the web driver will insert the first email ‘first_email@domain.com’ and password and will simulate the Enter Key stroke as if performed by the user. The application will then wait for the server response and will check if a password field exists, if it exists then web driver will continue with submitting the form with the next fake email address in the list ‘next_email@domain.com’ and so on until the list finishes. In every iteration SeleniumPhishGuard will check if the login form exists, if password field no longer exists during or after the test (see Figure 12). The current domain will be considered as phishing as the normal behavior for a legitimate website with a login page which submitted with fake credentials is to re-display the login form again to the user with an error message and thus a password field must exist always for a webpage to be considered as legitimate.

⁴ <https://docs.influxdata.com/influxdb/v1.2/>

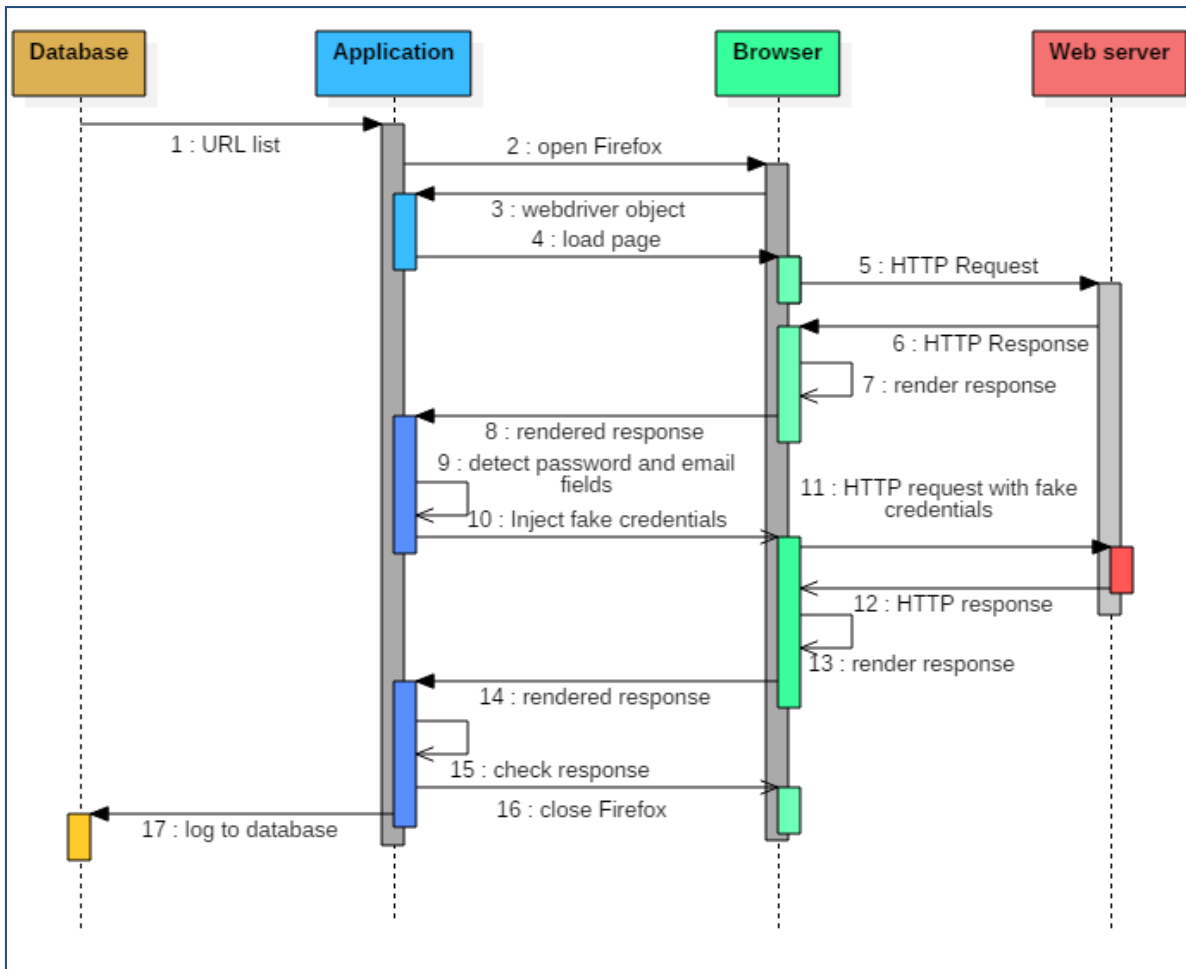


Figure 11. Phishing Identification module sequence diagram

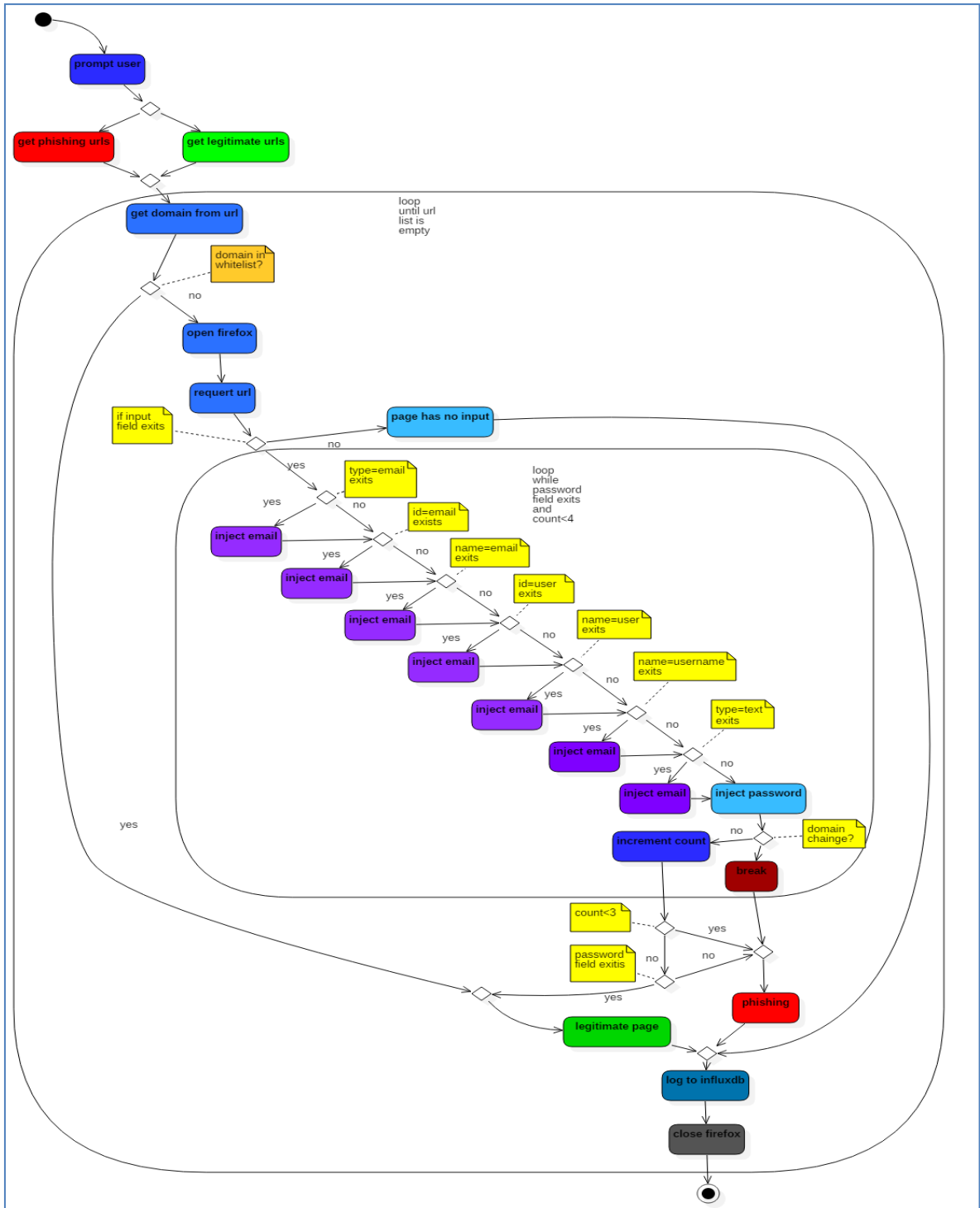
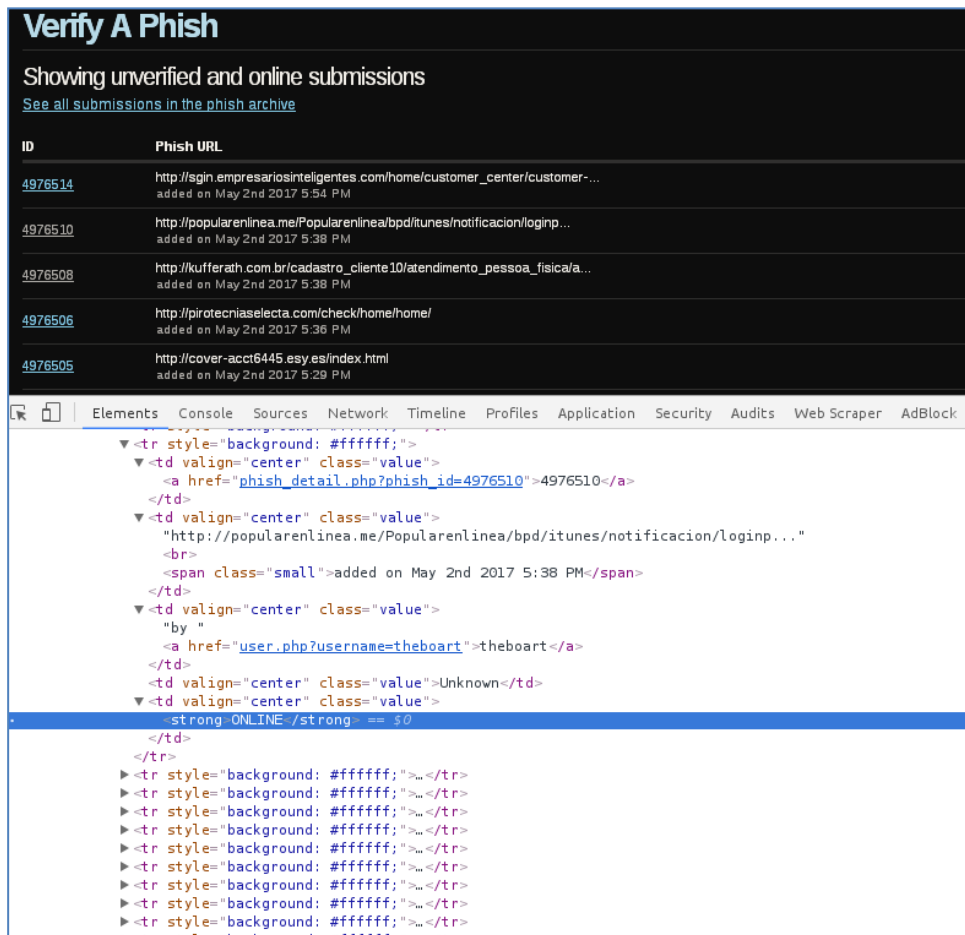


Figure 12. Phishing Identification module activity diagram

5 Data Collection Process

5.1 Phishing Pages' Scrapper

Two different scrapers were used to collect dataset for our tests. Ruby based scraper for scrapping phishing pages is implemented using Marlonso's⁵ Phishtank scraper ruby module. The scrapper will fetch and extract data from the first 50 pages from Phishtank. Phishtank displays data in form of HTML table as shown in Figure 13. The scrapper will only scrap data if the last cell in the row contains 'ONLINE' as HTML element content.



The screenshot shows the Phishtank website interface with a table of phishing submissions. Below the table, the browser's developer tools are open, displaying the HTML structure of the table rows. The HTML shows columns for ID, Phish URL, and a status field containing 'ONLINE'.

ID	Phish URL	
4976514	http://sgin.empresariosinteligentes.com/home/customer_center/customer-... added on May 2nd 2017 5:54 PM	
4976510	http://popularenlinea.me/Popularenlinea/bpd/itunes/notificacion/loginp... added on May 2nd 2017 5:38 PM	
4976508	http://kufferath.com.br/cadastro_cliente10/atendimento_pessoa_fisica/a... added on May 2nd 2017 5:38 PM	
4976506	http://pirotecniaselecta.com/check/home/home/ added on May 2nd 2017 5:36 PM	
4976505	http://cover-acct6445.esy.es/index.html added on May 2nd 2017 5:29 PM	ONLINE

Figure 13. Phishtank page layout

The scraper will extract all cells (id, URL, created_at, submitted_by, valid and online) and export it to JSON format (see Figure 14) which will be later used in our tests (see Figure 15). The scraper will scrap Phishtank website however, we are only interested in phishing pages with login forms. This will be filtered in Phishing identification module. This scraper was used to scrape dataset (1) shown in Table 2.

⁵ https://github.com/marlonoso/phishtank_scraper

```
[
  {
    "id": "4904749",
    "URL": "http://cafeim.co.kr/wp/caixa.gov.br/pages/inter/index.php",
    "created_at": "added on Mar 27th 2017 5:52 PM",
    "submitter": "anafeijo",
    "valid": "",
    "online": "ONLINE"
  },
  {
    "id": "4904745",
    "URL": "http://www.emapasgep.com/plugins/content/sub/.drpbxauh.php",
    "created_at": "added on Mar 27th 2017 5:42 PM",
    "submitter": "balomish",
    "valid": "",
    "online": "ONLINE"
  }
]
```

Figure 14. Sample of Data scrapped from Phishtank website in JSON format

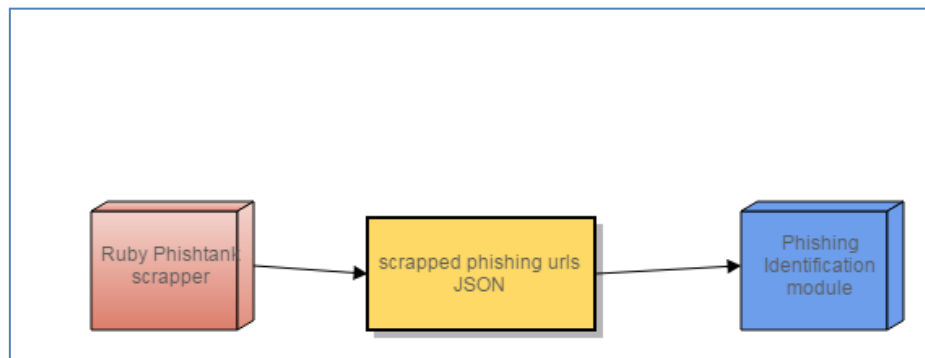


Figure 15. Phishtank scrapper

5.2 Legitimate Pages' Scrapper

The legitimate pages' scrapper is based on 2 different scrappers. The first scrapper is used to scrap Alexa database for the top 500 domains. This scrapper is based on a vivekpatani's⁶ Python library and adds it to a domain list. Then the second scrapper NikolaiT⁷ Google scraper will scrap Google using for URLs containing keywords 'login' and the domain name. The results will be exported to JSON file (see Figure 17) for future use by the phishing identification module (see Figure 16). This scraper is used to scrape dataset (1) shown in table 2.

⁶ <https://github.com/vivekpatani/alexa-scraper>

⁷ <https://github.com/NikolaiT/GoogleScraper>

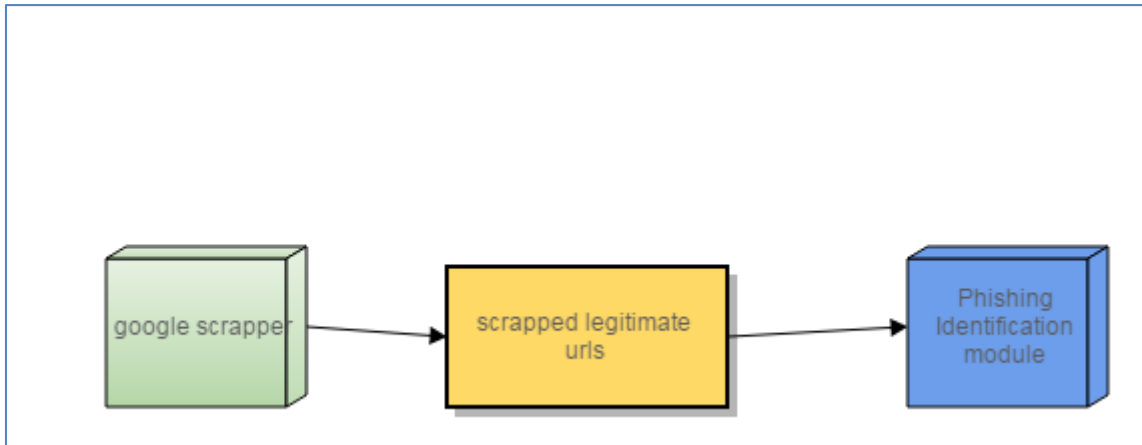


Figure 16. Google Scrapper

```

"results": [
  {
    "domain": "myaccount.payoneer.com",
    "id": "648",
    "link": "https://myaccount.payoneer.com/",
    "link_type": "results",
    "rank": "1",
    "serp_id": "199",
    "snippet": "Payoneer",
    "title": "Payoneer",
    "visible_link": "https://myaccount.payoneer.com"
  },
  {
    "domain": "teach.mapnwea.org",
    "id": "649",
    "link": "https://teach.mapnwea.org/",
    "link_type": "results",
    "rank": "2",
    "serp_id": "199",
    "snippet": "{{copyright}} ... {{copyright}}",
    "title": "NWEA UAP Login",
    "visible_link": "https://teach.mapnwea.org"
  }
]
  
```

Figure 17. Sample output of the URL list exported by Alexa scrapper

5.3 Data Sets

Data scrapped from Phishtank and Alexa are filtered. The total number of URLs scrapped from Alexa database is 500 and from Phishtank 1020. However, 649 pages only were available and online and the rest were either not found or unreachable as shown Dataset (1) shown in Table 2. Since phishing pages have a very limited time online, more phishing pages were scrapped from Phishtank and filtered only for pages with login forms resulting in 258. In addition, Alexa URLs from Dataset (1) were filtered for pages with login form resulting in 284 pages as shown in Table 3.

Dataset (3) shown in Table 4 used to determine the heuristic weights for the URL and domain analysis module, it is the first 100 phishing pages and the first 100 legitimate pages

from dataset (2). Dataset (4) uses the same 284 legitimate pages from Dataset (2) and newly scrapped URLs from Phishtank and filtered for login pages resulting in 421 URLs as shown in Table 5.

Table 2. Dataset (1) - extracted on 21/02/2017

Database	Number of URLs	Phishing/Legitimate
Phishtank	649	Phishing
Alexa	500	Legitimate

Table 3. Dataset (2) – extracted on 16/03/2017

Database	Number of URLs	Phishing/Legitimate
Phishtank	258	Phishing
Alexa	284	Legitimate

Table 4. Dataset (3) – extracted on 22/03/2017

Database	Number of URLs	Phishing/Legitimate
Phishtank	100	Phishing
Alexa	100	Legitimate

Table 5. Dataset (4) - extracted on 02/04/2017

Database	Number of URLs	Phishing/Legitimate
Phishtank	421	Phishing
Alexa	284	Legitimate

6 Evaluation Metrics

Evaluation metrics presented in this section will be used in proceeding sections. In any binary classification problem like the problem discussed in this paper, where the goal is to detect phishing pages in a dataset with a mixture of phishing and legitimate pages, there are only four classification possibilities exist. These possibilities are: true positive rate, false positive rate, true negative rate, false negative rate, and accuracy of our phishing detection mechanism. These are considered as standard metrics to moderate any type of phishing detection system. N_P represents the total number of phishing websites and N_L represents the total number of legitimate websites.

- $N_{P \rightarrow P}$ phishing websites classified as phishing
- $N_{P \rightarrow L}$ phishing websites classified as legitimate
- $N_{L \rightarrow P}$ legitimate websites classified as phishing
- $N_{L \rightarrow L}$ legitimate websites classified as legitimate

Performance measurements of a phishing detection mechanism is assessed in the following approach.

True positive rate (TPR):

- True positive rate is the rate of phishing websites classified as phishing out of the total phishing websites.

Equation 3. True Positive Rate (TPR)

$$TPR = \frac{N_{P \rightarrow P}}{N_P} \times 100 \quad (3)$$

- False positive rate (FPR): false positive rate is the rate of legitimate websites classified as phishing out of the total legitimate websites.

Equation 4. False Positive Rate (FPR)

$$FPR = \frac{N_{L \rightarrow P}}{N_L} \times 100 \quad (4)$$

- False negative rate (FNR): false negative rate is the rate of phishing websites classified as legitimate out of the total phishing websites.

Equation 5. False Negative rate(FNR)

$$FNR = \frac{N_{P \rightarrow L}}{N_P} \times 100 \quad (5)$$

- True negative rate (TNR): true negative rate is the rate of legitimate websites classified as legitimate out of the total legitimate websites.

Equation 6. True Negative Rate (TNR)

$$TNR = \frac{N_{L \rightarrow L}}{N_L} \times 100 \quad (6)$$

- Accuracy (A) measures the rate of phishing and legitimate websites which are identified correctly with respect to all the websites.

Equation 7. Overall Accuracy (A)

$$Accuracy = \frac{N_{L \rightarrow L} + N_{P \rightarrow P}}{N_L + N_p} \times 100 \quad (7)$$

7 Development Environment, Tools and System Usage

To ensure that our application was not compromised during analyzing phishing web pages, the development environment and testing environment were separated. The whole project was developed, Debian platform (development environment) and then deployed and tested on an Ubuntu virtual environment. The Ubuntu virtual environment is based on virtual screenshots or states, which means that every time the virtual environment is started with only libraries and tools installed. The code from the last run will be deleted.

7.1 Development Environment

The Operating system Debian⁸ GNU/Linux with release version number '8.7' was chosen as our development environment. The reasons for choosing Debian Linux is that it is one of the most widely used operating systems for servers due it is robustness, stability and availability. Moreover, debian.org claims that Debian has the best packing system in the world. Furthermore, the system is quite memory efficient specially when compared to Linux distributions. A Pentium 4, 1GHz system is the minimum required for a desktop system. Depending on the architecture, it is expected to install Debian from 20MB for s390 to 60MB for amd64. Debian as well supports all the packages and Python modules needed for developing our tool. This is considered as a great aspect since all packages were signed and installed via package manager. No packages were compiled from source code.

Python⁹ programming language with interpreter version 3.5 was chosen for the development of 'SeleniumPhishGuard'. Professionally, Python is great for backend web development, artificial intelligence, data analysis, data miming and scientific computing. Numerous programmers have also utilized Python to build productivity applications, games, and desktop tools, and thus there are enormous amount of resources and libraries to help bootstrap our development. Moreover, Python is a cross platform and works on all the popular operating systems. Furthermore, almost all Linux distributions come with Python installed by default which makes the deployment of our tool.

The Database used for storing the white-listed URLs is MongoDB¹⁰ version 3.4.3. MongoDB is a document-oriented database which saves data in collections made from separate documents, instead of storing data in tables like made from separate rows, like relational databases do. In MongoDB, a document is a big JSON file with no specific arrangement or schema. This makes the extensibility of the database quite easy specially in case if more relative data (e.g. IP address) must be added in the future.

Selenium Webdriver¹¹ is an open source software-testing framework for web applications which provides a playback tool for tests too. One of its options is Selenese - a test domain-specific language to write tests in Python and other programming languages. The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and OS X platforms.

Git¹² is a version control system (VCS) for tracking changes in code development, files and coordinating work on those files among multiple people if needed. It is principally used for software development but it can be used to follow changes in any files. Moreover, as a distributed revision control system it is intended to have support for distributed, non-linear

⁸ <https://www.debian.org/>

⁹ <https://www.Python.org/>

¹⁰ <https://www.mongodb.com/>

¹¹ <http://Selenium-Python.readthedocs.io/index.html>

¹² <https://github.com/>

workflows and robust data integrity. Version 2.12.2 was utilized in the development of our tool since it was the latest and the most stable version and connected to our remote repository on github.com.

7.2 Testing Environment

Ubuntu LTS (Long Term Support) version 16.04 was utilized as the testing environment. It is one of the most widely used Linux distributions. Ubuntu was installed within a virtual environment by utilizing Oracle VM VirtualBox¹³. The main reason for choosing Ubuntu is that there are many readymade live virtual-box images available for free. Thus, there is no need to install and configure it, it is plug and play on virtual box. The minimum requirements for Ubuntu desktop version are 700 MHz processor Intel Celeron or better, system memory of 512 MiB RAM, hard-drive space of 5 GB, VGA capable of 1024x768 screen resolution, either a CD/DVD drive or a USB port for the installer media and Internet access.

The same Python interpreter of version 3.5 was installed on Ubuntu along with Selenium Webdriver module. InfluxDB¹⁴ version 1.2 was utilized for logging purposes. It is created to store time-series data. Relational databases can handle time-series but weren't designed specifically for that objective. InfluxDB is made to work with a large size of time-series data and perform real-time analysis on those data, rapidly. This database was used for logging test results, errors and bugs with respect to time. It also identifies that schema preferences may change over time. In InfluxDB, there is no need to define schemas at start. Data points can have one of the fields on a measurement, all fields on a measurement, or any number in-between. New fields can be added to a measurement basically by writing a point for that new field.

Grafana¹⁵ 4.2.0 was utilized for data visualization and monitoring. Grafana is an open source metric analytics and visualization suite. It is frequently used for visualizing time series data for infrastructure and application analytics but numerous developers use it in other domains including weather, industrial sensors, process control and home automation. Grafana was specifically used since it compatible with InfluxDB and contains powerful real time visualization tools.

7.3 System Usage

User will be prompt as shown in Figure 18, by inputting “l”, legitimate pages will be tested. If the user inputs “p”, phishing pages will be tested.

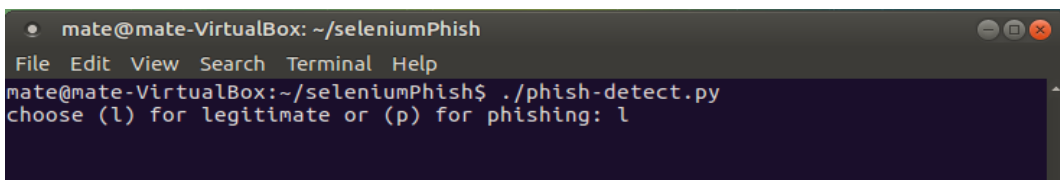


Figure 18. Selenium user prompt

Figure 19 shows the log output by SeleniumPhishGuard during testing.

¹³ <http://docs.grafana.org/>

¹⁴ Data visualization & Monitoring

¹⁵ <http://docs.grafana.org/>

```

File Edit View Search Terminal Help
test started with 276 pages
https://www.facebook.com
testing ... www.facebook.com
domain is legit and in whitelist
https://wordpress.com/wp-login.php
testing ... wordpress.com
domain is legit and in whitelist

```

Figure 19. Testing real-time logs

Firefox will find and inject email and password in their corresponding fields. Then it will submit form by simulating “enter” key (See Figure 20).

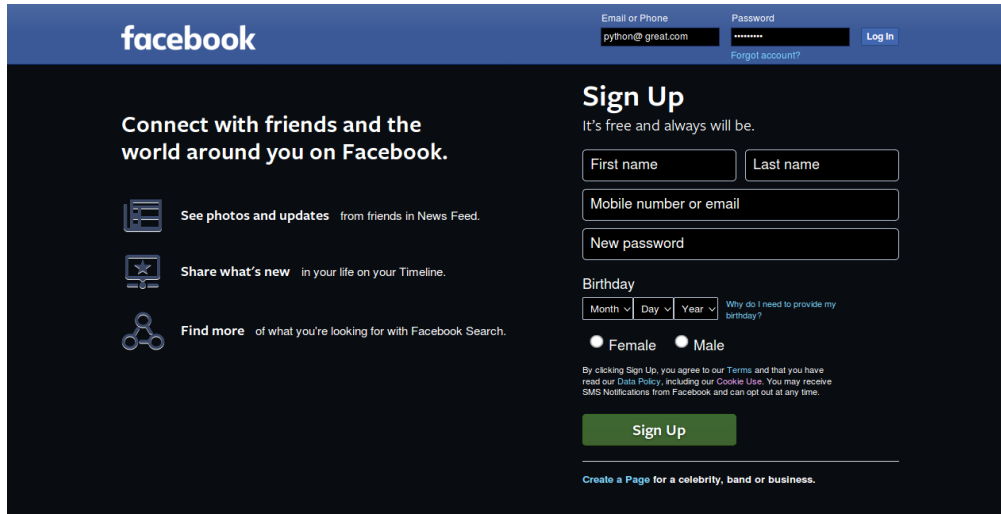


Figure 20. Selenium filling Facebook login form

Web-driver will continue to inject email and password and submit the form for n number of times. After that it will check for the presence of the password field (See Figure 21). If the password still exists then the webpage will be considered as legitimate, else classified as phishing. Test results are shown in the output terminal as shown in Figure 22.

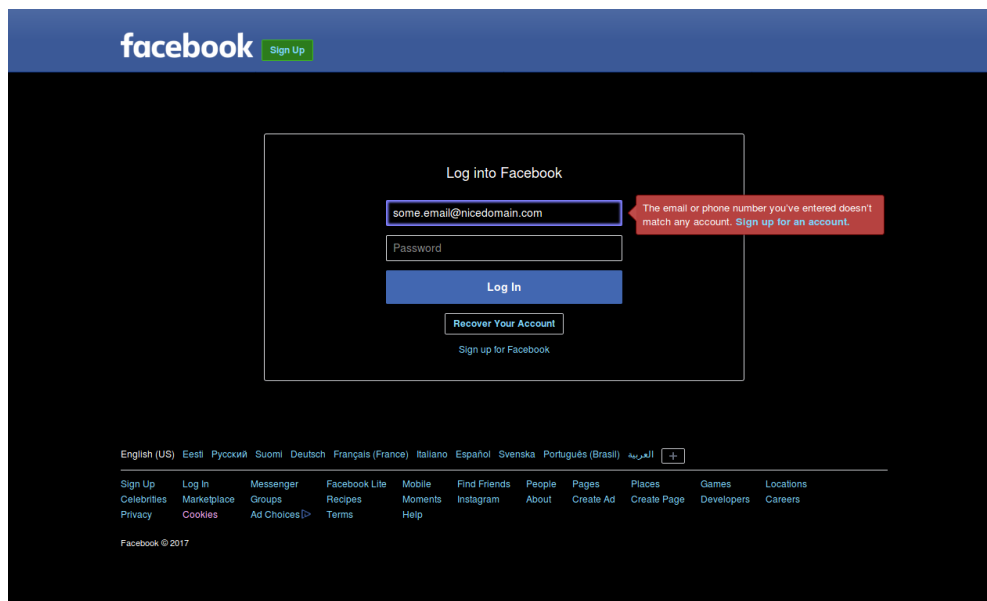


Figure 21. Page response after form submission

```
File Edit View Search Terminal Help
this page has no login
http://kocasoy.com.tr/secure-files/www.wellsfargo.com-online/
this is a phishing website domain analysis
http://dr0pb0x.uploadedfiles.com.rudolphson.com/MyDBoxFile/d40679fcec005d0bd68e6
f1a826b65162017/login.php?cmd=login_submit&id=f271d7a1e816617834c2cb0d090481
42f271d7a1e816617834c2cb0d09048142&session=f271d7a1e816617834c2cb0d09048142f
271d7a1e816617834c2cb0d09048142
testing ... dr0pb0x.uploadedfiles.com.rudolphson.com
webdriver exception hereMessage: Element is not visible

http://dr0pb0x.uploadedfiles.com.rudolphson.com/MyDBoxFile/d40679fcec005d0bd68e6
f1a826b65162017/login.php?cmd=login_submit&id=122bbfbe78118880d3c907e84629e6
54122bbfbe78118880d3c907e84629e654&session=122bbfbe78118880d3c907e84629e6541
22bbfbe78118880d3c907e84629e654
testing ... dr0pb0x.uploadedfiles.com.rudolphson.com
webdriver exception hereMessage: Element is not visible

http://dr0pb0x.uploadedfiles.com.rudolphson.com/MyDBoxFile/d40679fcec005d0bd68e6
f1a826b65162017/
testing ... dr0pb0x.uploadedfiles.com.rudolphson.com
webdriver exception hereMessage: Element is not visible

test sucessfully finished
mate@mate-VirtualBox:~/seleniumPhish$
```

Figure 22. Test successfully finished

Real time data visualization was achieved by using Grafana. Grafana is a graphical tool integrated with our influx time series logging database to show results in real-time (See Figure 23).



Figure 23. Grafana visualizing False positives and True Positives

8 Results

8.1 Phishing Identification Module Results:

The results and evaluation of our tool “SeleniumPhishGuard” will be discussed in this section. To ensure that our tool is language independent datasets collected from different languages websites as shown in Table 6.

Table 7. shows the results used to calculate heuristic weights for the URL and domain analysis module according to their effectiveness (See sub section 4.2). According to our test results, creation date was the most effective heuristic with 85% True positive rate. Dots in URL was effective as a heuristic by detecting 44% of phishing webpages.

Table 6. Languages associated with datasets

Language
English
German
Russian
Arabic
Japanese
Korean
French
Italian
Portuguese
Spanish
Polish
Czech
Mandarin
Hindi
Urdu

Table 7. Heuristics weights

Heuristic	True positive	False positive	Effect	weight
Domain creation date	85%	32%	53	5
Domain expiry date	23%	5%	18	2
'@' in URL	10%	0%	10	1
'-' in URL	15%	4%	11	1
Dots in URL	44%	5%	39	4
Domain inWHOIS	9%	7%	2	0

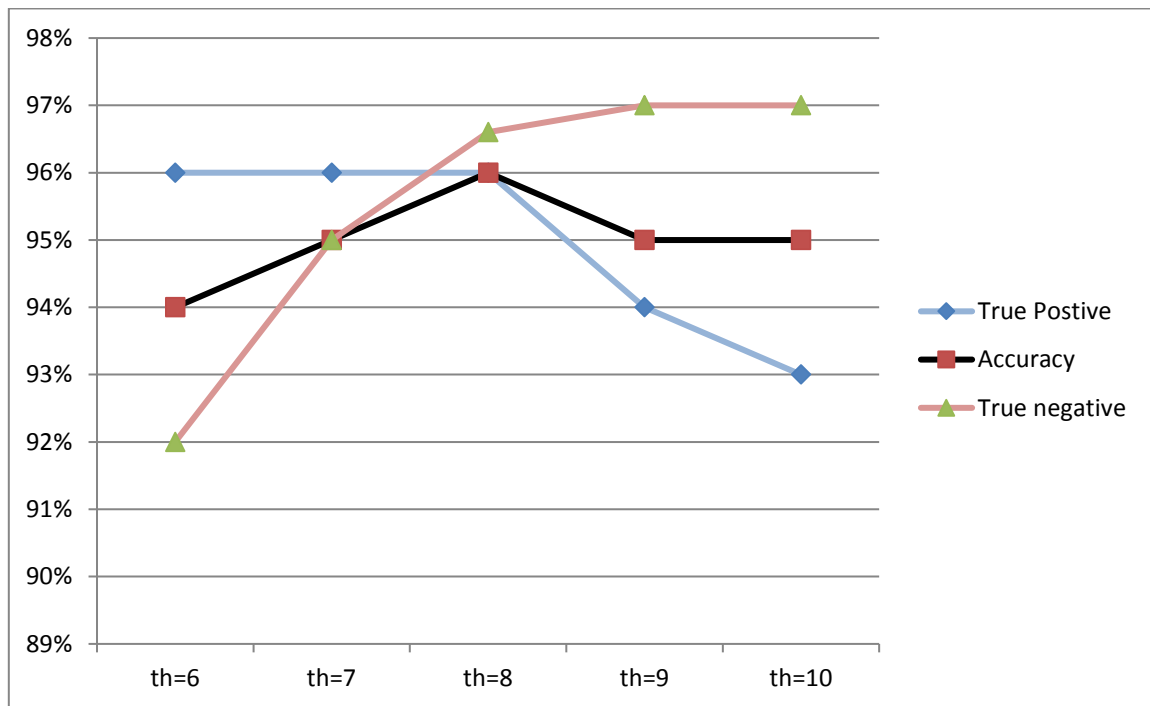


Figure 24. Graph showing the threshold effect on accuracy

To choose a value “ x ” for threshold “ th_x ” function in equation no.1, the system was tested 5 times against 100 phishing URLs and 100 legitimate URLs dataset (3) shown in Table 4. The limits for the choice of threshold values was not chosen randomly. Domain creation date has a heuristic weight of 5 and it does not make sense to have a threshold value “ x ” of 5 since the classification will be based only on this heuristic. Therefore, lowest threshold value in our test is 6, as the highest weight. The highest x value for this test is 10, since its corresponding true positive rate is equal 93% same as using the phishing identification

module unaccompanied. Therefore, higher testing with higher threshold values was not considered. Figure 24 shows how different threshold values affect the true negative rate, true positive rate and thus affects the overall accuracy.

When $x = 6$ the URL and domain analysis module had a lower true negative rate, which means a higher false positive rate or classifying more legitimate pages as phishing as expected. As shown in Figure 24 the increase in the threshold value results in a higher true negative rate. However, as x increases beyond 8, the true positive rate decreases as URL and domain analysis module classify more URLs legitimate.

Table 8 below shows results when the phishing identification module was tested solely against dataset (2) shown in Table 3. The True negative rate is 96,83% which is quite high as expected. Since a legitimate website must always show a password field if the login form is submitted with incorrect credentials injected.

Table 8. Results of phishing identification module

Total tested URLs	542
Phishing	258
Legitimate	284
True positives	241
False Negative	17
False Positive	9
True Negative	275
True Positive Rate	93,42%
False Negative Rate	6,58%
True Negative Rate	96,83%
False Positive Rate	3,16%
Overall Accuracy	95,2%

The whole system (URL and domain analysis module with the phishing identification module) is tested against dataset (4) Table 5. The URL and domain analysis module improved the overall performance to 96.66% as shown in Table 9 due to lowering the rate of false positives from 3.16% to 2.65% as shown in Figure 25.

Table 9. Phishing Identification with URL analysis Module results

Phishing	421
Legitimate	284
True positives	405
False Negative	16
False Positive	8
True Negative	276
True Positive Rate	96,2%
False Negative Rate	3,80%
True Negative Rate	97,34%
False Positive Rate	2,65%
Overall Accuracy	96,66%

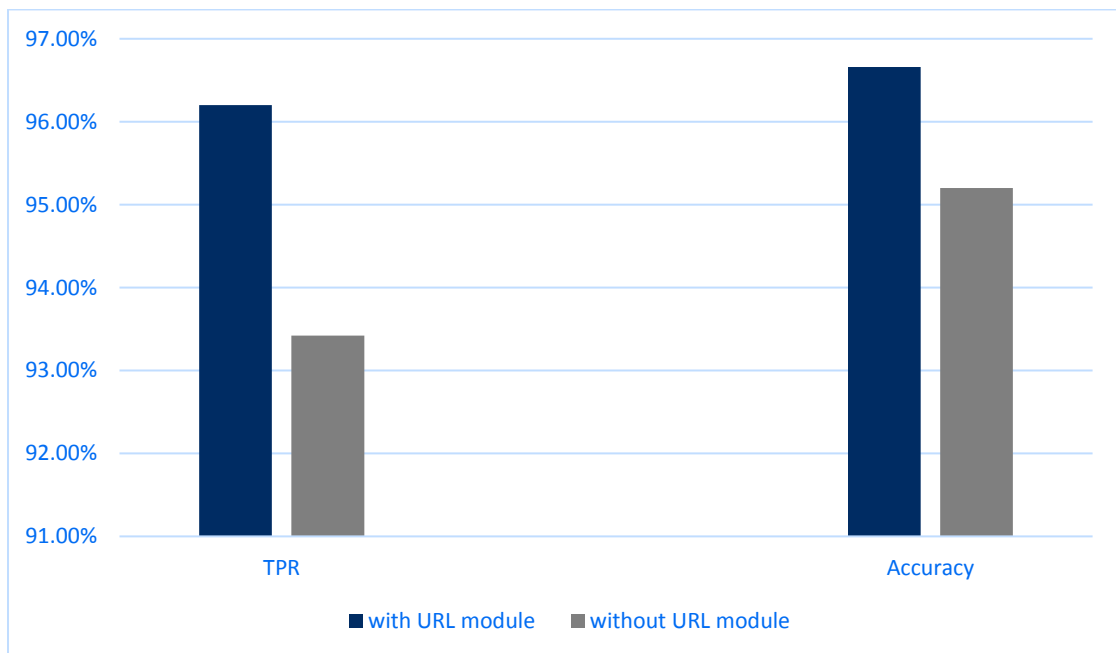


Figure 25. Comparison between system results with and without URL and Domain Analysis Module

8.2 Comparison Between Related Work and Our Application:

Comparison between SeleniumPhishGuard and other related tools is presented in Table 10. All tools are designed to work as real time applications and detect zero-day phishing pages. SeleniumPhishGuard is language independent along with PhishGuard and Spoof Guard. However, CANTINA only works with English language. SeleniumPhishGuard correctly tests and classifies HTTPS pages along with CANTINA and Spoof guard where PhishGuard fails since it depends on HTTP server status codes. SeleniumPhishGuard tests and classifies script based pages since the tool utilizes Firefox to render pages. However, the rest of the other related tools work on the HTML source. The biggest challenge for SeleniumPhishGuard is testing pages with AJAX content and it is a challenge for the rest of the tools as well. In addition, our tool is the slowest amongst the rest of the tools since WHOIS queries and page rendering requires a lot of time.

Table 10. Table of comparison

	Spoof Guard	PhishGuard	CANTINA	SeleniumPhishGuard
Detecting script based pages	No	No	No	Yes
HTTP	yes	Yes	Yes	Yes
HTTPS	yes	no	Yes	Yes
Language independency	yes	yes	No	yes
real-time	yes	yes	yes	Yes
Pages with AJAX content	No	No	No	No
Zero-day detection	yes	yes	Yes	Yes
Pages without login	Yes	Yes	Yes	No
Time efficient	Yes	Yes	Yes	No

9 Conclusion

The APWG reported that the amount of phishing websites it detected rose hysterically by 250 percent within the period of just October 2015 to March 2016 [1]. Just in March 2016, 123555 new unique phishing websites were detected. Phishing websites mimic legitimate websites to trick victims into submitting credentials to access private content. According to our analysis, around 50% to 60% of the pages extracted for testing contains a login form with email/username and password combination. Thus, the focus of this paper is to introduce a new phishing detection tool that is effective in detecting phishing login pages.

In this paper, overview of most of the existing phishing detection techniques were discussed. Effective heuristic-based methodologies were discussed more and our new method was proposed. Our approach is to detect phishing login pages by injecting fake credentials and analyzing response. Usually, Phishing pages does not check submitted password and will show success message or redirect to another website. Detecting the response from the server is quite achievable in case of HTTP authentication as the server replies with codes to represent if the authentication was successful (200 OK) or not authorized (401). The challenge was detection of server response in case of HTTPS authentication, since the server response is usually the same in both cases (Authentication success or failure). Our approach is to inject the credentials for n number of times and check if the password input field. If the password input field exists then this is considered as authentication failure and considered as authentication successful otherwise.

A tool called “SeleniumPhishGuard” was built based on our approach. The tool utilizes Selenium web-driver to render pages, inject credentials and analyze response. The URL and DNS matching module which was introduced in [23] was used in the first version of this tool to minimize the rate of false positives, however it was discarded in the current version due to high false negative rate.

URL analysis module was introduced to minimize the phishing detection rate and improve overall accuracy. We have devised a small set of weighted rules to detect some characteristics of phishing URLs. This module tests a given URL against the weighted rules and computes a total score for this URL and if the score exceeds the threshold the website is phishing. If not then Selenium will open the page and test it.

We have presented a detailed algorithm and shown that our solution correctly identifies Phishing login pages with a low false positive rate. Our results indicate that our rules are quite efficient. In addition, SeleniumPhishGuard can detect zero-day attacks that are gone unnoticed by existing phishing detection mechanisms. Finally, besides the benefit of increased accuracy, SeleniumPhishGuard requires little configuration when compared to other filters. SeleniumPhishGuard can only be used as a stand-alone filter yet in future, An API for browser plugins will be introduced. Currently, SeleniumPhishGuard is tested on either HTTP or HTTPS pages that contains a login form.

We continue to further experiment with the current heuristics, particularly by testing and updating our tool against different language and technology based websites.

Future Work: Enhancements

The challenge that other content based phishing detection methodologies had to face is login pages that generate or obfuscate content dynamically with JavaScript to elude web crawlers that look for and process static HTML only. Because we render the page and thus JavaScript code will be executed, we can see through this obfuscation. Manual identification showed 12 suspected phishing pages that used JavaScript that we find in rendered page but

not in HTML alone: our application found all 12 of them. As well, since the module is language independent and it only relies of the presence or absence of the password field, it managed to detect phishing in many different languages that other heuristics based phishing detection tools will not.

However, there are some limitations in Selenium that create a challenge for our classifier to successfully classify pages as legitimate or phishing. One of the properties of Selenium web-driver, is that the page must be fully loaded before the test is executed. In addition, analysis is complicated when AJAX is implemented. To overcome this issue, specific waiting time was added in case if Selenium throws an exception that the page is not fully loaded, for instance Google accounts use AJAX with in forms (See Figure 26). It will forcibly stop page and inject the credentials to the password and email fields then refresh the page with the injected credentials. However, it is not guaranteed that it will work permanently.

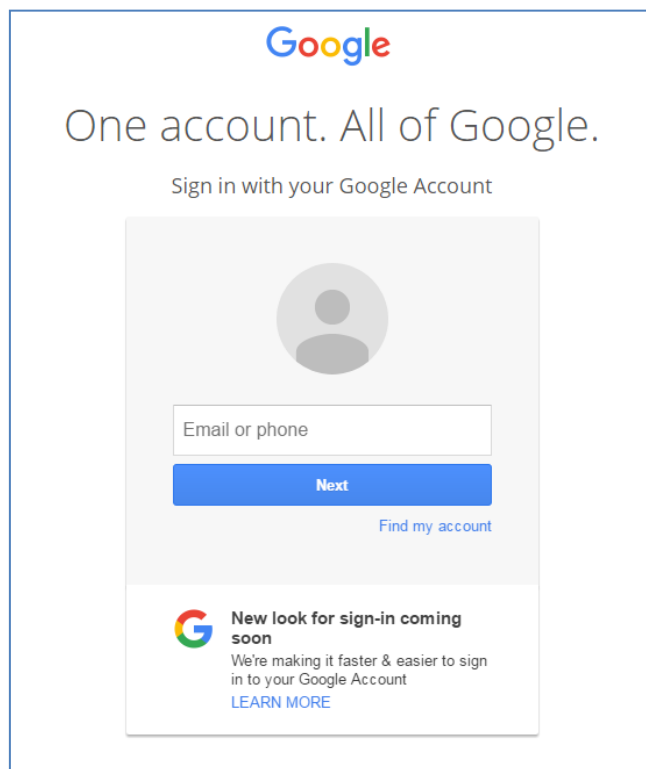


Figure 26. Script based login form

The image shows a web form titled "Register" with a blue header. The form contains the following fields and elements:

- Name:** Input field containing "It Solution Stuff".
- E-Mail Address:** Input field containing "admin@gmail.com".
- Password:** Input field containing six asterisks "*****".
- Confirm Password:** Empty input field.
- Captcha:** A reCAPTCHA widget showing a green checkmark, the text "I'm not a robot", and the reCAPTCHA logo with "reCAPTCHA" and "Privacy - Terms" links.
- Register Button:** A blue button with a white document icon and the text "Register".

Figure 27. Form with captcha

Another challenge was captchas. Selenium cannot solve captchas as shown in Figure 27, and it was not intended to be implemented in our application, since this is a phishing detection tool not a hacking tool. In case of a captcha existing in the form, it completely depends on the type of captcha and the response of the website. For some websites, captcha exits and it seems to Selenium that form is submitted and in this case, there is a big possibility that the website will be considered as legitimate regardless. The other case, where the form will not be submitted, the website will be classified as incomplete test. Since our application is white-list based, URLs with incomplete tests will not be added to the white-list and hence if classified as legitimate it will be considered as a false positive.

Time taken for testing ranges from 0.1 sec to 4.9 seconds per page depending on the page being tested due to WHOIS queries and page rendering. Threading is one of the possible enhancements, which reduces the computational time. Our application can be used as a background script for browser plug-ins and thus an API should be implemented. It could be used to identify phishing URLs in web server referrer logs, or it could be used by web hosting providers to check for phishing sites on their servers.

Classifying mainly by injecting credentials and analyzing response might create an opportunity for attackers to circumvent our application by always rendering a page with a password field. New heuristics must be introduced.

10 References

- [1] APWG, "Phishing Activity Trends Report", APWG, 2016.
- [2] "History of Phishing | Phishing.org", Phishing.org, 2017. [Online]. Available: <http://www.phishing.org/history-of-phishing/>. [Accessed: 11- Feb- 2017].
- [3] Statista-cybercrime, "Number of global phishing sites 2013-2016 | Statistic", Statista, 2017. [Online]. Available: <https://www.statista.com/statistics/266155/number-of-phishing-domain-names-worldwide/>. [Accessed: 11- Feb- 2017]
- [4] "Majority of Americans fall for email phishing scams", Cbsnews.com, 2017. [Online]. Available: <http://www.cbsnews.com/news/majority-of-americans-fall-for-email-phishing-scams-cbs-intel-security-quiz/>. [Accessed: 14- Feb- 2017].
- [5] "97% Of People Globally Unable to Correctly Identify Phishing Emails | McAfee Online Newsroom", Newsroom.mcafee.com, 2017. [Online]. Available: <http://newsroom.mcafee.com/press-release/97-people-globally-unable-correctly-identify-phishing-emails>. [Accessed: 14- Feb- 2017].
- [6] S. Sharma and S. Karla, "A Comparative Analysis of Phishing Detection and Prevention Techniques", International Journal of Grid and Distributed Computing, vol. 9, no. 8, pp. 371-384, 2016.
- [7] M. Khonji, Y. Iraqi and A. Jones, "Phishing Detection: A Literature Survey", IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2091-2121, 2013.
- [8] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong and C. Zhang, "An Empirical Analysis of Phishing Blacklists", Proceedings of Sixth Conference on Email and Anti-Spam (CEAS)., vol. 16-17, 2009.
- [9] M. Hara, A. Yamada, and Y. Miyake, "Visual similarity-based phishing detection without victim site information," in IEEE Symposium on Computational Intelligence in Cyber Security, 2009. CICS '09, pp. 30 – 36, 2009
- [10] A. Y. Fu, L. Wenyin, and X. Deng, "Detecting phishing web pages' visual similarity assessment based on earth mover's distance (emd)," IEEE Trans. Dependable Secure. Comput. vol. 3, no. 4, pp. 301–311, Oct. 2006.
- [11] Y. Zhang, J. Hong, and L. Cranor, "Cantina: a content-based approach to detecting phishing web sites," in Proceedings of the 16th international conference on World Wide Web, 2007.
- [12] G. Xiang, J. Hong, C. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," ACM Transactions on Information and System Security, vol. 14, no. 2, 2011.
- [13] Neda Abdelhamid, Aladdin Ayesh, Fadi Thabtah, "Phishing detection based Associative Classification data mining," Expert Systems with Applications, Volume 41, Issue 13, pp. 5948-5959, 2014
- [14] M. Aburrous, MA Hossain, K Dahal, T. Fadi, "Predicting phishing websites using classification mining techniques," Seventh international conference on information technology, Las Vegas, Nevada, USA, 2010.

- [15] H. Huang, L. Qian, Y. Wang, "A SVM-based technique to detect phishing URLs," *Inf.Technol. J.* vol. 11, no. 7, pp. 921–925, 2012.
- [16] V. Ramanathan, H. Wechsler, "Phishing website detection using Latent Dirichlet Allocation and AdaBoost," *IEEE International Conference on Intelligence and Security Informatics. Cyberspace, Border, and Immigration Securities*, Piscataway, NJ, pp. 102–107, 2012.
- [17] S. Garera, N. Provos, M. Chew, and A. D. Rubi, "A Framework for Detection and Measurement of Phishing Attacks," *WORM Proceedings of the 2007 ACM workshop on Recurring malware*, pp.1-8,2007.
- [18] R. Gowtham, Ilango Krishnamurthi, "A comprehensive and efficacious architecture for detecting phishing webpages," *Computers & Security*, Vol. 40, pp. 23-37, 2014.
- [19] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell, "Client-side defense against web-based identity theft," in *NDSS. The Internet Society*, 2004.
- [20] P. Likarish, D. Dunbar, and T. E. Hansen, "Phishguard: A browser plug-in for protection from phishing," in *2 nd International Conference on Internet Multimedia Services Architecture and Applications*, 2008. *IMSAA 2008*, 2008, pp. 1 – 6.
- [21] D. L. Cook, V. K. Gurbani, and M. Daniluk, "Phishwish: A stateless phishing filter using minimal rules," in *Financial Cryptography and Data Security*, G. Tsudik, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 182–186
- [22] A. Jain and B. Gupta, "A novel approach to protect against phishing attacks at client side using auto-updated white-list", *EURASIP Journal on Information Security*, vol. 2016, no. 1, 2016. , pp. 5–9
- [23] 3Sharp, 3Sharp Study finds Internet Explorer 7 Edges Out Netcraft As Most Accurate for Anti-Phishing Protection. 2006.
<http://www.3sharp.com/projects/antiphishing/>

Appendix

I. Glossary

Abbreviation	Description
APWG	Anti-phishing working group An international consortium that brings together businesses affected by phishing attacks, security products and services companies, law enforcement agencies, government agencies, trade association, regional international treaty organizations and communications companies.
AOL	America Online An American multinational mass media corporation based in New York, a subsidiary of Verizon Communications
URL	Uniform Resource Locator A URL is the address of a specific webpage or file on the Internet.
DNS	Domain Name Server/Service Domain names serve as memorable names for websites and other services on the Internet. DNS translates domain names into IP addresses, allowing you to access an Internet location by its domain name.
API	Application programming Interface An API is a set of commands, functions, protocols, and objects that programmers can use to create software or interact with an external system.
HTTP	Hypertext transfer protocol HTTP is the protocol used to transfer data over the web. It is part of the Internet protocol suite and defines commands and services used for transmitting webpage data.
HTTPS	Hypertext transfer protocol secure HTTPS is the same thing as HTTP, but uses a secure socket layer (SSL) for security purposes.
OS	Operating System A software that communicates with the hardware and allows other programs to run. It is comprised of system software, or the fundamental files your computer needs to boot up and function.

IEEE	<p>Institute of Electrical and Electronics Engineers</p> <p>The IEEE is a professional association that develops, defines, and reviews electronics and computer science standards.</p>
HTML	<p>Hyper Text Markup Language</p> <p>HTML is the language used to create webpages.</p>
DOM	<p>Document Object Model</p> <p>A cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.</p>
IP	<p>Internet Protocol</p> <p>It allows devices running on different platforms to communicate with each other as long as they are connected to the Internet.</p>
SMTP	<p>Simple Mail Transfer Protocol</p> <p>A protocol used for sending e-mail over the Internet.</p>
POP	<p>Post Office Protocol</p> <p>A simple, standardized method of delivering e-mail messages.</p>
AV	<p>Anti-Virus</p> <p>Antivirus software is a type of utility used for scanning and removing viruses from your computer.</p>
JSON	<p>JavaScript Object Notation</p> <p>JSON is a text-based data interchange format designed for transmitting structured data.</p>
AJAX	<p>Asynchronous JavaScript and XML</p> <p>AJAX is a combination of Web development technologies used for creating dynamic websites.</p>
TLS	<p>Transport Layer Security</p> <p>Transport layer security (TLS) is a protocol that provides communication security between client/server applications that communicate with each other over the Internet.</p>
HREF	<p>Hypertext Reference</p>

	Reference to data that the reader can directly follow either by clicking, tapping, or hovering
LDA	Latent Dirichlet Allocation Generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar
TF-IDF	Term frequency/inverse document frequency A numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus.
SVM	Support Vector Machine Supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis
MCAC	Multi-label Classifier based Associative Classification

II. Previous Work (Discontinued)

Previous Methodology:

1. User visits a website as shown in Figure 28.
2. URL and DNS module checks if the website is in the white-list which updates from Google public DNS.
3. If the domain name is found and matched the IP, then the website is legitimate.
4. If domain is found in the white-list however, the IP was not matched, then this is a phishing website.
5. If no domain is found in the white-list, will be checked by the phishing identification module.
6. If Google open DNS has no info about the website then the application will test it.
7. The Phishing identification module will inject n number of fake email password combinations.
 - I. The phishing detection module will detect all the input text fields with type attribute (email) or name attribute (email, user, username, Id and userid).
 - II. The module will inject the input text fields with fake credentials and wait till page loads.
 - a. If the page loads with no password fields then it is considered as phishing.
 - b. If the page redirects to another website then it is considered as phishing.
 - c. If the page reloads with an input text field of type password, then the application will inject more fake credentials for n number of times.
 - d. If the password field still exists after n number of trials, then the page will be considered as legitimate.

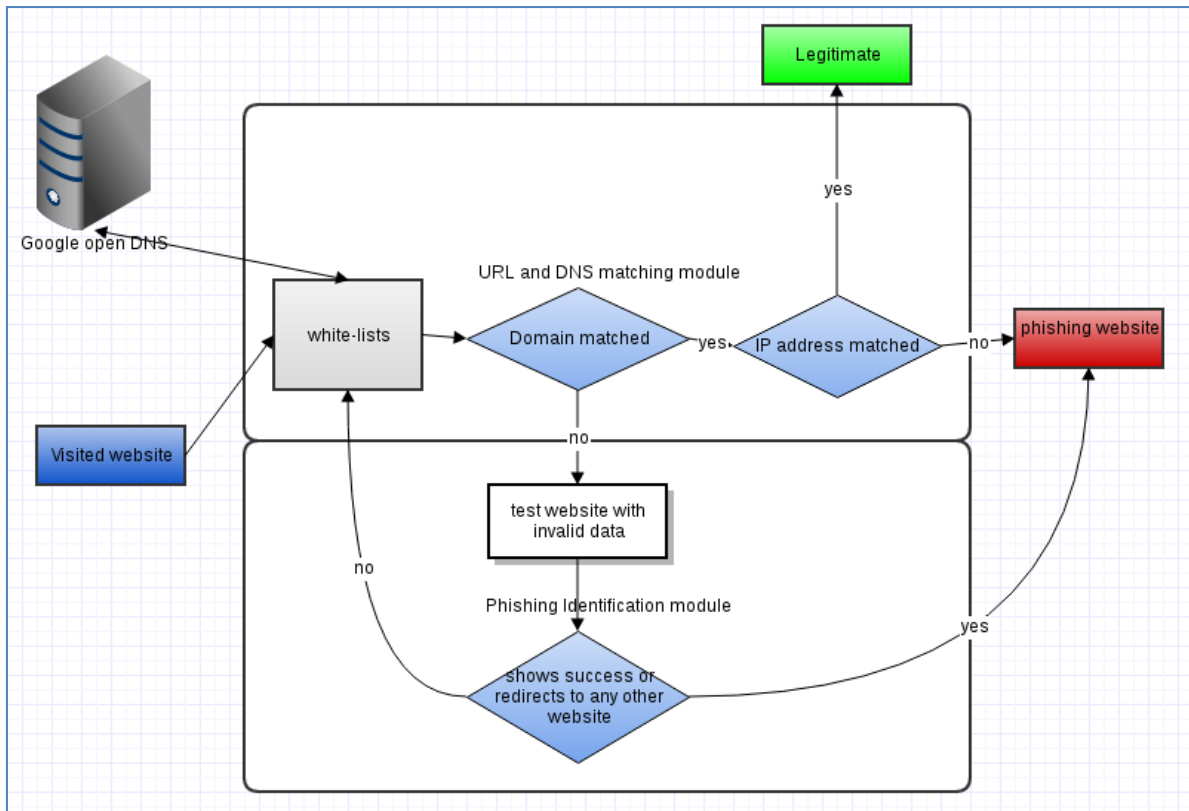


Figure 28. Preliminary methodology

URL and DNS Matching Module

In this paper, we are utilizing the same the URL and DNS matching module which was introduced in [23]. The purpose of the URL and DNS module which accommodates white-list is to minimize the false positive rate. In addition, the white-list is composed of two parameters, domain name and its corresponding IP address. When a user requests access to website (see Figure 28), the application extracts the domain name of the current website from the URL and checks if it exists in the white-list. If the domain of the current website is found in the white-list, then the application checks the IP address before decision making. If the accessed website is currently residing in the white-list, then the matching module matches IP address to the corresponding domain to check for DNS poisoning attack. The white-list starts empty at the beginning; which means that at the beginning, there is no domain in the list and the white-list starts populating once a user accesses the fresh web pages. Moreover, whenever a user accesses a website, there are two chances, either it is the first-time user accesses the website or it was visited previously by the user. If the user is accessing the website for the first time, the domain of the website will not be present in the white-list. Google public DNS will be queried to check if there were entries for that domain. If no domain exists, then the second module starts operating. The second module is the phishing identification module, which checks whether a webpage is legitimate or phishing (See Figure 29). This module was discontinued due high false positive rate as shown in Table 11 and was replaced by URL and domain analysis module discussed in section 4.2.

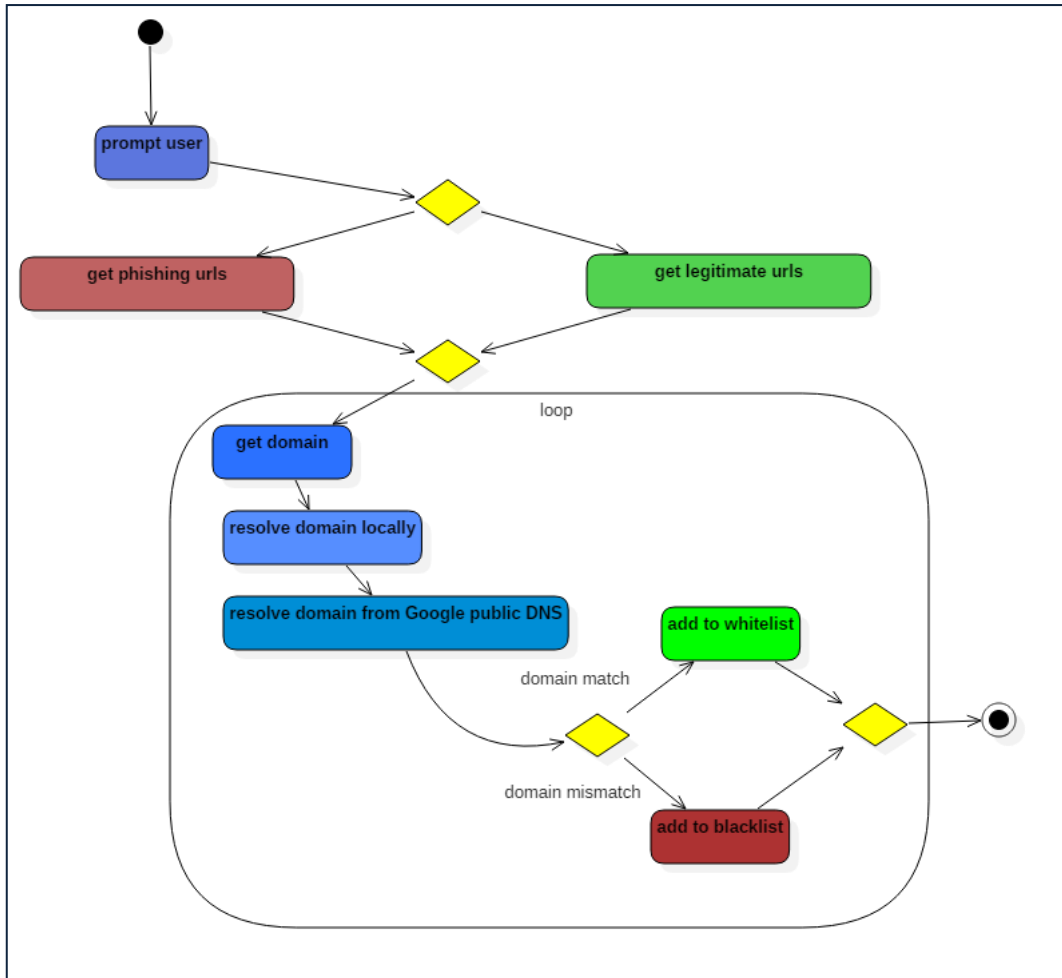


Figure 29. URL and DNS module activity diagram

Table 11. DNS and URL matching module results

Total tested URLs	1149
Phishing	649
Legitimate	500
True Positives	25
False Negative	624
False Positive	64
True Negative	436
True Positive Rate	3,85%
False positive Rate	12,8%


```
File Edit View Search Terminal Help
['52.54.4.253']
domain match
Politico.com
['52.3.120.217']
['52.3.120.217']
domain match
Mysql.com
['137.254.60.6']
['137.254.60.6']
domain match
Dell.com
['143.166.135.105', '143.166.147.101']
['143.166.135.105', '143.166.147.101']
domain match
Themegrill.com
['109.73.226.14']
['109.73.226.14']
domain match
Safedog.cn
['110.86.5.91']
['110.86.5.91']
domain match
Oecd.org
['78.41.128.140']
```

Figure 30. Sample output of URL and DNS module

The DNS module as shown in the Figure 30, checks all IP addresses in both the local DNS and if results do not exactly match, the URL and DNS module will classify the whole domain as phishing. If we take a closer look at the results that our URL and DNS module has classified as phishing as shown in Figure 31 and Figure 32 we clearly deduce that some of the top websites are considered as DNS poisoning or phishing as they have different IP addresses in the local DNS and Google public DNS. Major website domains like Facebook.com, Twitter.com, Instagram.com and Google.de. Therefore, this module had to be discarded and replaced by the URL analysis module instead.

```
Twitter.com
['104.244.42.129', '104.244.42.65']
['104.244.42.193', '104.244.42.65']
Domain mismatch
Google.com
['216.58.211.142']
['216.58.209.142']
Domain mismatch
Youtube.com
['216.58.211.142']
['216.58.209.110']
```

Figure 31. Sample of IP mismatch for legitimate domains

```
File Edit View Search Terminal Help
> select * from phishing
name: phishing
time          browser url
----          -
1491857867836599504 firefox Facebook.com
1491857867898472439 firefox Twitter.com
1491857868089682099 firefox Instagram.com
1491857873532151650 firefox Blogspot.com
1491857873871603254 firefox Youtu.be
1491857874073899811 firefox Goo.gl
1491857875082962777 firefox Blogger.com
1491857875147548853 firefox T.co
1491857875836026493 firefox Addthis.com
1491857876059953198 firefox Bluehost.com
1491857876257482806 firefox Google.de
1491857876428524122 firefox Soundcloud.com
1491857876891695772 firefox Feedburner.com
1491857876936958685 firefox Digg.com
1491857880138127976 firefox Weibo.com
1491857880893661169 firefox Google.co.jp
1491857881225735918 firefox Yelp.com
1491857885426138822 firefox Telegraph.co.uk
1491857885784849369 firefox Google.co.uk
```

Figure 32. Sample of false positives by the URL and DNS matching module.

III. Phishtank Scrapper

Phishtank Scrapper

Scraper code is written in ruby language

The scraper will mainly crawl the first 50 pages of phishtank.com and will scrape and filter results that are valid and online.

Then the application will export the output to file in JSON format.

```
require 'phishtank_scraper'
require 'json'

# get the contents of pages 0-50
# export these contents as JSON in 'links.json' file
pt_scraper = PhishtankScraper.new
submissions = pt_scraper.page_scrape((0..50), {active: "y", valid: "y"})
submissions_json = submissions.to_json
begin
  file = File.open("links.json", "w")
  file.write(submissions_json)
rescue IOError => e
  #some error occur, dir not writable etc.
ensure
  file.close unless file.nil?
end
```

IV. Alexa Top 500 Scrapper

Alexa Scrapper Code

This scrapper class returns an array of the Alexa top 500 websites.

```
def scrape(n=50, sub_dir="topsites", local="global", sub_local=""):
    """
    Scrape the given number of pages from alexa website
    """
    n = int(n)
    if (n > 500):
        print("Alexa Top 500 has at most 500 Links.")
        n = 500

    # Converting each letter after / to be uppercase, otherwise link will result
    in no response
    for each_sub in range(len(sub_local)):
        if (sub_local[each_sub - 1] == '/' or each_sub == 0):
            new_sub_local += sub_local[each_sub].upper()
        else: new_sub_local += sub_local[each_sub]

    sub_local = "Top/" + new_sub_local

    # Additional Headers to make it more human.
    request_headers = {
        "Accept-Language": "en-US,en;q=0.5",
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/50.0",
        "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8",
        "Connection": "keep-alive" }

    # To calculate time for programme to run
    start = time.clock()

    # Get the number of pages we need to scrape
    num_of_pages = calc_number_of_pages(n)
    final_list = []

    for page_num in range(num_of_pages):

        # Generate the complete URL based on input
        full_URL = "http://www.alexa.com/" + sub_dir + "/" + local + ";" + str(pa
ge_num) + "/" + str(sub_local)

        # Collect page data for current page
        page = ""

        # Try Making a request
```

```

try:
    request = Request(full_URL, headers=request_headers)
except e:
    print(e.reason)

try:
    response = URLOpen(request)
except e:
    print(e.reason)

for line in response:
    line = line.decode('utf-8', 'ignore')
    page += line

# Create a new parser for n links
parser = htmlparser(n=n)

# Reduce total by 25 links, since there are 25 links on each page
n -= 25

# Parse the collected page feed
parser.feed(page)

# Append the links to the final list
final_list += parser.links

# Print them in a readable way
print_top(final_list)

# print(str(time.clock() - start))
return final_list

```

```

def main():

    # In case if user does not provide a parameter, 50 is defaulted
    if (len(sys.argv) == 1):
        scrape()

    # When user provides n
    elif (len(sys.argv) == 2):
        try:
            num = int(sys.argv[1])
        except ValueError:
            print("Not an Integer")
        scrape(n=num)

    # When user provides by local and sub local
    elif (len(sys.argv) == 4 and (sys.argv[2] == 'category' or sys.argv[2] == 'countries')):
        num, local, sub_local = sys.argv[1:]
        try:
            num = int(sys.argv[1])
        except ValueError:
            print("Not an Integer")
        scrape(n=num, local=local, sub_local=sub_local)

    # When user gives an unknown format of unknown, show them option
    else:
        print("Maybe you missed a parameter\n \
List of Valid Command Formats: \n \
1.) python script-name.py \n \
2.) python script-name.py number-of-links \n \
3.) python script-name.py number-of-links countries country-code \n \
4.) python script-name.py number-of-links category sub-category1/sub-
category2/sub-categoryn/")

if __name__ == "__main__":
    main()

```

V. URL and DNS Matching Module

URL and DNS Matching Module code:

```
# resolve Ip from domain by querying the local Dns
# using python socket library
def get_ips_for_host(host):

    ip_list = []
    try:
        ips = socket.gethostbyname_ex(host)
        ip_list = ips[2]
        ip_list.sort()
        print(ip_list)
    except socket.gaierror:
        print('error')
    return ip_list

# Query google open dns to get ip of the same domain
def get_Info_from_googleOpenDns(domain):
    res = resolver.Resolver()
    res.nameservers = ['8.8.8.8']
    answers = res.query(domain)

    index = 0
    ip_list = []

    for rdata in answers:
        ip_list.append(rdata.address)
        ip_list.sort()

    print(ip_list)
    return ip_list

def main(URL):
    try:
        domain = get_domain_from_uri(URL)
        ip_local = get_ips_for_host(domain)
        ip_google_dns = get_Info_from_googleOpenDns(domain)
        if ip_local == ip_google_dns:
            print('domain match')
            to_influx_database(domain,0)
        else:
            print('Domain mismatch')
            to_influx_database(domain,1)
    except:
        continue
```

VI. Phishing Identification Module

Phishing Identification Module Code:

Main function works as follows

```
# this is our main function
# user will be prompt to input whether he wants to test legitimate or phishing page
# the function will extract domain from url and check if it is in white list
# if domains exists the webdriver will close the firefox window session
# if domain does not exist then it will be tested by 'full_test' function
# then it will be logged
def main():
    user_input = input("choose (1) for legitimate or (p) for phishing: ")
    if user_input == 'p':
        link_array = get_phishing_pages()
    elif user_input == '1':
        link_array = get_legitimate_pages()
    else:
        link_array = get_phishing_pages()
    print('test started with '+str(len(link_array))+ ' pages')
    old_link = ''
    for link in link_array:
        print(link)
        analysis = url_analysis.check(link)
        if analysis >= 7 :
            print('this is a phishing website domain analysis')
            to_influx_database(link, 1)
            continue

    if link != old_link:
        if link is not None and link != '':
            driver = webdriver.Firefox()
            url = link
            domain = get_domain_from_uri(url)
            try:
                driver.get(url)
                print('testing .... '+domain)
                domain_in_whitelist = check_domain_in_white_list(domain)
                if domain_in_whitelist != None:
                    print('domain is legit and in whitelist')
                    to_influx_database(link, 0)
                else:
                    WebDriverWait(driver,2)
                    result = full_test(driver, domain, url)
                    to_influx_database(url, result)

            except TimeoutException as error:
                print('there is a timeout exception here'+str(error))
                to_influx_database(link, -1)
                pass
            except WebDriverException as error:
                print('webdriver exception here'+str(error))
                pass
            except ValueError or TypeError or UnicodeDecodeError as error:
                print ('value error or type error: ' +str(error))
                continue
            except Exception as error:
                print('total random exception: '+str(error))
                continue

        driver.quit()
        old_link = link
    print('test sucessfully finished')
```



```

# this function returns a list of fake emails for testing
def test_email_list():
    emails = ['some_email@nicedomain.com', 'happy.forever@best.com', 'python@grea
t.com', 'first.last@name.com']
    return emails

#selenium webdriver has no builtin function to check whether an element exists or
not
# this is why this function was implemented
# it tries to find an element by xpath
# if the element does not exist
# NoSuchElementException will be raised and the function will catch that exceptio
n and
# and return false

def check_exists_by_xpath(driver, xpath):
    try:
        driver.find_element_by_xpath(xpath)
    except NoSuchElementException :
        return False
    return True

# this function will check the existance of all of the elements specified
# and will return the ones that exists for testing
def email_and_password_exits(driver):
    input_tag = check_exists_by_xpath (driver, input_tag_xpath)
    text_type = check_exists_by_xpath(driver, text_type_xpath)
    email_type = check_exists_by_xpath(driver, email_type_xpath)
    email_id = check_exists_by_xpath(driver, email_id_xpath)
    email_name = check_exists_by_xpath(driver, email_name_xpath)
    user_id = check_exists_by_xpath(driver, userId_xpath)
    user_name = check_exists_by_xpath(driver, user_name_xpath)
    username_name = check_exists_by_xpath(driver, username_name_xpath)
    passwd = check_exists_by_xpath(driver, passwd_xpath)

    return input_tag, text_type, email_type, email_id, email_name, \
        user_id, user_name, username_name, passwd

#----- testing part -----
-#

# this function will take fake credentials (email, username .. etc)
# and will first clear the text input field then add
# the fake email
def test_fake_credentials(driver,test_email,xpath,count):
    try:
        user = driver.find_element_by_xpath(xpath)
        if count > 0:
            try:
                # user.clear()
                user.send_keys(Keys.CONTROL + "a")
                user.send_keys(Keys.DELETE)
            except:
                pass
        user.send_keys(test_email) # Your email_id
    except ElementNotVisibleException:
        pass
    except NoSuchElementException:
        pass

```

```

# this function will get the password type field and will input a fake password
def test_fake_password(driver, count):
    passwd = driver.find_element_by_xpath(passwd_xpath)
    if count > 0:
        try:
            passwd.clear()
            passwd.send_keys(Keys.CONTROL + "a")
            passwd.send_keys(Keys.DELETE)
        except:
            pass
    passwd.send_keys("Hello12345") # Your password
    passwd.send_keys(Keys.RETURN)
    WebDriverWait(driver, 2)
    WebDriverWait(driver, 5).until(EC.staleness_of(passwd))

# here is our main testing function
def full_test(driver, domain_name, url):

    # returns all elements that we are interested in
    input_tag, text_type, email_type, email_id, email_name, \
    user_id, user_name, username_name, \
    password = email_and_password_exits(driver)

    # getting the fake email list
    count = 0
    email_list = test_email_list()

```

```

try:
    while input_tag and count < 3:

        if password:
            domain = get_domain_from_uri(driver.current_url)

        if email_type:
            test_fake_credentials(driver, email_list[count],email_type_xpath, count)

        elif email_id:
            test_fake_credentials(driver, email_list[count],email_id_xpath, count)

        elif email_name:
            test_fake_credentials(driver, email_list[count],email_name_xpath, count)

        elif user_id:
            test_fake_credentials(driver, email_list[count],userId_xpath, count)

        elif user_name:
            test_fake_credentials(driver, email_list[count],user_name_xpath, count)

        elif username_name:
            test_fake_credentials(driver, email_list[count],username_name_xpath, count)

        elif text_type:
            test_fake_credentials(driver, email_list[count],text_type_xpath, count)

        test_fake_password(driver, count)
# if selenium injects the website with an email and password
# and a redirect occurs this is for sure a phishing website
newDomain = get_domain_from_uri(driver.current_url)

        if newDomain != domain:
            print('this is a phishing website because of redirect')
            return 1
        else:
            count += 1
            #recheck-
ing again what are the elements exist in the page before the next iteration
            input_tag,text_type, email_type, email_id, email_name, \
            user_id, user_name, username_name, \
            password = email_and_password_exists(driver)

    else:
        break

```

```

except TimeoutException as error:
    if password and count > 0:
        print('this is a legitimate page')
        to_mongodb(domain_name, url)
        return 0
        #if this was the first iteration and there is not login fields
# then this page has no login
    if count < 1 and (not email_type or not email_id or not email_name) and not p
assword:
        print('this page has no login')
        return 2

# if after a couple of emails there are no more password field exists?
# then the website is a phishing one
elif not password and count < 2:
    if not email_type or not email_id or not email_name:
        print('this is a phishing website due to test')
        return 1
# if the loop continued till the end and password still exists
# this website passes the test and is considered as legit
elif password and count >=2 :
    print('this is a legitimate page')
    to_mongodb(domain_name,url)
    return 0

else:
    # if count > 1 and password:
    #     print('this is a legitimate page')
    #     to_mongodb(domain_name, url)
    #     return 0
    # else:
    print('unknown error')

```

VII. URL and Domain Analysis Module

```
# check the expiry date and creation date of domain
def check_whois(domain):
    WHOIS_query = WHOIS.query(domain)
    creation_date = WHOIS_query.creation_date
    expiration_date = WHOIS_query.expiration_date
    return creation_date,expiration_date

#compute difference between expiration_date and creation_date
def check_date_difference(cer_date,exp_date):
    todays_date = datetime.date.today()
    days_since_creation = (todays_date - cer_date.date()).days
    days_till_expiration = (exp_date.date() -todays_date).days
    print(days_since_creation)
    print(days_till_expiration)

def url_contain_symbols(url):
    dots = url.count('.')
    dashes = url.count('-')
    ats = url.count('@')
    return dots, dashes, ats

def check(url):
    count = 0
    domain = get_domain_from_uri(url)
    try:
        cer_date, exp_date = check_whois(domain)
        cer_days, exp_days = check_date_difference(cer_date, exp_date)
        if cer_days < 365:
            count += 5
        if exp_days < 180:
            count += 2
    except:
        count = 6
    dots,dashes,ats = url_contain_symbols(url)
    if dots >= 5:
        count +=4
    if dashes > 0:
        count +=1
    if ats > 0:
        count += 3
    return count
```

VIII. Database Functions

```
# this function fetchs the mongodb white list for domains
# to check if it already resides in the whitelist
def check_domain_in_white_list(domain):
    db_client = MongoClient()
    db = db_client.phishing
    cursor = db.whitelist.find_one({'legitimate.domain_name': domain})
    return cursor

# get phishing pages from the ruby scraper
def get_phishing_pages():
    jsonFile = open('scraper/phishing_links.json', 'r')
    data = json.load(jsonFile)
    jsonFile.close()

    link_array = []

    for index in data:
        link_array.append(index['URL'])

    print(len(link_array))
    return link_array

# get legit pages from python scraper
# data is saved as JSON and this function will parse JSON
# and will extract the domains
def get_legitimate_pages():
    jsonFile = open('scraper/legitimate.json', 'r')
    data = json.load(jsonFile)
    jsonFile.close()
    link_array = []
    for index in data:
        if len(index['results']) != 0:
            results = index['results']
            for result in results:
                link_array.append(result['link'])
    return link_array

# strip protocol header and path
# extract the URL
def get_domain_from_uri(uri):
    domain_name = urlparse(uri).hostname.split('.')
    domain = domain_name[-2] + '.' + domain_name[-1]
    return domain

# this function is called whenever a legitimate website
# needs to be added to our whitelist which is hosted in this mongodb
# the domain will be added under the collection *legitimate* in phishing database
name
def to_mongodb(domain,url):
    db_client = MongoClient()
    db = db_client.phishing
    db.whitelist.insert_one(
        {
            "legitimate": {
                "domain_name": domain,
                "url": url
            }
        }
    )
```

```

# Influx_db is a time series no sql databases which has the first field always as
# the current time the log
# in our case it is used because of the ease of use
#we have mainly 4 measurements
# phishing for websites that are detected as phishing
# legitimate for the websites that are detected as legitimate
# if the website has no login or considered as neutral
# if the website contains login yet for some reason the test could not complete
# it is considered as incomplete_test
# this function takes the url and sends it to the influxdb under its correspondin
# g measurement
def to_influx_database(url, res):
    if res == 1:
        result = "phishing"
    elif res == 0:
        result = "legitimate"
    elif res == -1:
        result = "incomplete_test"
    else:
        result = "neutral"

    points = [

        {
            "measurement": result,
            "tags": {
                "browser": "firefox"
            },
            "fields": {
                "url": url
            }
        }
    ]

    try:
        db_client = InfluxDBClient('localhost', '8086',
                                   'root', 'root', 'phishing_db')
        db_client.create_database('phishing_db')
        db_client.write_points(points)

    except IOError as error:
        print(str(error))

```

IX. Index

- 200 status code, 8, 20, 21, 26, 49
- 401 status code, 8, 20, 49
- 404 status code, 8
- Accuracy, 37, 45, 46
- Acknowledgments, 4
- AdaBoost, 17, 53
- AJAX, 47, 50, 55
- Alexa, 7, 26, 32, 33, 34, 39, 64
- Amazon, 19, 21
- America Online, 10, 54
- APWG, 10, 11, 49, 52, 54
- Blacklists, 6, 15, 52
- CANTINA, 16, 25, 47
- CANTINA+, 16
- Chrome, 27
- claimAV, 18
- computer, 7
- credentials, 7, 9, 10, 19, 20, 23, 24, 27, 28, 45, 49, 50, 51, 58
- credit card, 10
- cybercriminal, 10
- Debian, 38
- DNS, 16, 49, 54, 58, 59, 60, 61, 62, 67
- domain, 6, 8, 12, 16, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 32, 38, 43, 45, 52, 58, 59, 61, 73
- eBay, 10, 19
- E-Gold, 10
- email, 6, 7, 8, 12, 19, 21, 22, 24, 28, 41, 49, 50, 52, 58
- Firefox, 18, 19, 27, 28, 41, 47
- Git, 38
- Google, 17, 32, 33, 50, 58, 59, 61
- Grafana, 39, 42
- Heuristic weight calculation function, 26
- heuristics, 8, 9, 15, 16, 18, 25, 26, 49, 50, 51
- HTML, 8, 16, 18, 19, 21, 28, 31, 47, 50, 55
- HTTP, 8, 18, 19, 20, 47, 49, 54
- HTTPS, 8, 19, 21, 47, 49, 54
- Identity hiding, 13
- IEEE, 15, 52, 53, 55
- InfluxDB, 28, 39
- Intel, 12
- JSON, 31, 32, 55, 63
- Latent Dirichlet Allocation, 17, 53, 56
- machine learning, 15, 18, 53
- Machine learning, 15, 16
- Malware, 13
- Man in the Browser (MITB) attack, 13
- MCAC, 16, 56
- McAfee, 12, 52
- mitigation, 7, 15
- MongoDB, 27, 38, 39
- password, 7, 8, 19, 20, 21, 24, 28, 41, 45, 49, 50, 51, 58
- PayPal, 10, 21
- PC, 10
- PhishGuard, 8, 19, 20, 23, 47
- phishing, 2, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 31, 32, 33, 34, 36, 37, 38, 40, 41, 43, 44, 45, 47, 49, 50, 51, 52, 53, 54, 58, 59, 61
- Phishing life-cycle, 6
- Phishing pages' scrapper, 31
- Phishing.org, 10, 52
- PhishTank, 17, 26
- PhishWish, 21
- phreaks, 10
- Python, 27, 38, 39
- Ruby, 40
- Selenium, 27, 38, 39, 41, 49, 51
- SeleniumPhishGuard, 9, 28, 41, 43, 47, 49
- Simplified Classifier Score Function, 25
- Social engineering, 10
- Spoof gurad, 18
- spoofed e-mails, 10
- Spoof-Guard, 19
- Statistica, 12
- SVM, 16, 17, 53, 56
- Ubuntu, 38, 39
- URL, 8, 9, 15, 16, 17, 18, 19, 21, 22, 24, 25, 26, 27, 28, 31, 33, 43, 44, 45, 46, 49, 54, 58, 59, 61, 62, 67, 73
- Visual similarity, 15, 16, 52
- web browsers, 6
- web-crawler, 17
- white-list, 6, 18, 24, 27, 51, 53, 58, 59
- WHOIS, 21, 22, 24, 25, 26, 47, 51
- zero-hour, 8, 15, 18, 21

X. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Ahmed Nafies Okasha Mohamed,

(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
 - 1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
 - 1.2. make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

A New Heuristic Based Phishing Detection Approach Utilizing Selenium Web-driver,

(title of thesis)

supervised by Dr.Olaf Manuel Maennel

Dr.Raimundas Matulevičius

(supervisor's name)

2. I am aware of the fact that the author retains these rights.
3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **19.05.2017**