**A PROJECT REPORT ON**

**DETECTING MOBILE MALICIOUS WEB PAGES IN REAL TIME**

Submitted in Partial Fulfilment of the Requirement for the Award of

# BACHELOR OF TECHNOLOGY

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By**

1. **NITHIN 16S11A05F0**

**UNDER THE GUIDANCE OF**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# MALLA REDDY INSTITUTE OF TECHNOLOGY AND SCIENCE

Permanently affiliated to JNTUH and approved by AICTE, New Delhi

NAAC & NBA Accredited, ISO 9001:2015 certified, Approved by U.K Accreditation centre, Granted status of 2(f) and 12(b) under UGC act 1956, Govt. of India.

MAISAMMAGUDA, DHULAPALLY, SECUNDERABAD-500 100

**2016-2020**

**MALLA REDDY INSTITUTE OF TECHNOLOGY AND SCIENCE**

Permanently affiliated to JNTUH and approved by AICTE, New Delhi

NBA & NAAC Accredited, ISO 9001:2015 certified, Approved by U.K Accreditation centre, Granted status of 2(f) and 12(b) under UGC act 1956, Govt. of India.

MAISAMMAGUDA, DHULAPALLY, SECUNDERABAD-500 100

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

This is to certify that the mini project report entitled **“DETECTING MALICIOUS MOBILE WEB PAGES IN REAL TIME**” being submitted by **A.NITHIN (16S11A05F0),** in partial fulfilment of the degree of Bachelor of Technology in Computer Science and Engineering during the academic year 2019-2020.

Certified further, to the best of my knowledge, the work reported here is not a part of any other project on the basis of which a degree or an award has been given on an earlier occasion to any other candidate. The results have been verified and found to be satisfactory.

**Internal Guide Exteral Examiner**

**Head of the Department**

**Mr.S Md MUJEEB**

Associate professor

# ACKNOWLEDGEMENT

I express a whole hearted gratitude to **Dr. K. RAVINDRA**, Principal and Professor of Electronics and Communication Engineering Department, Malla Reddy Institute of Technology and Science for providing us the conducive environment for carrying our academic schedules and projects with ease**.**

I thank **Mr.S.MD.MUJEEB,** Associate Professor and Head, Department of Computer Science and Engineering for providing his seamless support and knowledge during our B.Tech course period and also for providing right suggestions at every phase of the development of our project.

I’m very much thankful and greatly indebted to our internal guide, for his guidance and encouragement throughout the preparation and completion of this project.

There is definitely a need to thank all staff members, friends and parents without whose support project would have been deferred.

# ABSTRACT

Mobile specific web pages differ significantly from their desktop counterparts in content, layout and functionality. Accordingly, existing techniques to detect malicious websites are unlikely to work for such web pages. In this paper, we design and implement kayo, a mechanism that distinguishes between malicious and benign mobile webpages. KAYO makes this determination based on static features of a webpage ranging from the number of iframes to the presence of known fraudulent phone numbers. First, we experimentally demonstrate the need for mobile specific techniques and then identify a range of new static features that highly correlate with mobile malicious webpages. We then apply kAYO to a dataset of over 350,000 known benign and malicious mobile webpages and demonstrate 90% accuracy in classification. Moreover, we discover, characterize and report a number of webpages missed by Google Safe Browsing and Virus Total, but detected by kAYO. Finally, we build a browser extension using kAYO to protect users from malicious mobile websites in real-time. In doing so, we provide the first static analysis technique to detect malicious mobile webpages.

# CONTENTS

**TOPICS Page No.**

**ACKNOWLEDGEMET ABSTRACT**

**LIST OF FIGURES**

**LIST OF TABLES**

**CHAPTER 1: INTRODUCTION 01-06**

* 1. How Data Mining works 01-02
  2. Five major elements of data mining: 02-03
  3. Characteristics of Data Mining 03-04
  4. Advantages of Data Mining 04-05

**CHAPTER 2: LITERATURE SURVEY 06-09**

**CHAPTER 3: SYSTEM ANALYSIS 10-13**

* 1. Existing System 10
  2. Proposed System 10
  3. Feasibility 11-12
     1. Economical Feasibility 11
     2. Technical Feasibility 12
     3. Social Feasibility 12
  4. Software & Hardware Requirements 13
     1. Hardware Requirements 13
     2. Software Requirements 13

**CHAPTER 4 SYSTEM DESIGN 14-22**

* 1. System Architecture 14
  2. System Flowchart 16-17
  3. UML Diagrams 18

|  |  |  |  |
| --- | --- | --- | --- |
|  | 4.3.1 Use Case Diagram | 18-19 |  |
| 4.3.2 Class Diagram | 20 |  |
| 4.3.3 Sequence Diagram  4.3.4 Activity Diagram | 21  22 |  |
| **CHAPTER 5** | **IMPLEMENTATION** | **23-27** |  |
|  | 5.1 Implementation Tools  5.1.1 The .NET framework | 23  23 |  |
| **CHAPTER 6** | **TESTING** | **28-31** |  |
|  | 6.1 Testing Methodologies | 28 |  |
|  | 6.2 Unit testing | 30 |  |
|  | 6.3 Integration Testing | 30 |  |
|  | 6.4 Acceptance Testing | 31 |  |
| **CHAPTER 7** | **CONCLUSION** | 32 |  |

**BIBILOGRAPHY 33**

**List of Figures**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.No.** | **Figure No.** | **Name of the figure** | **Page No.** |
| **1** | 1.1 | Normalized density curves of static features. | 01 |
| 2 | 4.1 | System Architecture | 14 |
| 3 | 4.2 | Flowchart | 17 |
| 4 | 4.3 | Use Case Diagram | 19 |
| 5 | 4.4 | Class Diagram | 20 |
| 6 | 4.5 | Sequence Diagram | 21 |
| 7 | 5.1 | Layered Representation | 27 |

**CHAPTER 1**

**INTRODUCTION**

Mobile devices are increasingly being used to access the web. However, in spite of significant advances in processor power and bandwidth, the browsing experience on mobile devices is considerably different. These differences can largely be attributed to the dramatic reduction of screen size, which impacts the content, functionality and layout of mobile web pages.

Content, functionality and layout have regularly been used to perform static analysis to determine malicious- ness in the desktop space [20], [37], [51]. Features such as the frequency of iframes and the number of redirec- tions have traditionally served as strong indicators of malicious intent. Due to the significant changes made to accommodate mobile devices, such assertions may no longer be true. For example, whereas such behavior would be flagged as suspicious in the desktop setting, many popular benign mobile webpages require multiple redirections before users gain access to content. Previous techniques also fail to consider mobile specific webpage elements such as calls to mobile APIs. For instance, links that spawn the phone’s dialer (and the reputation of the number itself) can provide strong evidence of the intent of the page. New tools are therefore necessary to identify malicious pages in the mobile web.

In this paper, we present kAYO, a fast and reliable static analysis technique to detect malicious mobileweb-pages. kAYO uses static features of mobile webpages derived from their HTML and JavaScript content, URL and advanced mobile specific capabilities. We first exper- imentally demonstrate that the distributions of identical static features when extracted from desktop and mobile webpages vary dramatically. We then collect over 350,000 mobile benign and malicious webpages over a period of three months. We then use a binomial classification tech- nique to develop a model for kAYO to provide 90% accuracy and 89% true positive rate. kAYO’s performance matches or exceeds that of existing static techniques used in the desktop space. kAYO also detects a number of malicious mobile webpages not precisely detected by existing techniques such as VirusTotal and Google Safe Browsing. Finally, we discuss the limitations of existing tools to detect mobile malicious webpages and build a browser extension based on kAYO that provides real- time feedback to mobile browser users.

We make the following contributions:

**Experimentally demonstrate the differences in the “security features” of desktop and mobile web- pages:** We experimentally demonstrate that the distributions of static features used in existing tech- niques (e.g., the number of redirections) are different when measured on mobile and desktop webpages. Moreover, we illustrate that certain features are in- versely correlated or unrelated to or non-indicative to a webpage being malicious when extracted from each space. The results of our experiments demon- strate the need for mobile specific techniques for detecting malicious webpages.

# Design and implement a classifier for malicious and benign mobile webpages: We collect over 350,000 benign and malicious mobile webpages. We then identify *new* static features from these web- pages that distinguish between mobile benign and in classification and shows improvement of two or- ders of magnitude in the speed of feature extraction over similar existing techniques. We further empir- ically demonstrate the significance of kAYO’s fea- tures. Finally, we also identify 173 mobile webpages implementing cross-channel attacks, which attempt to induce mobile users to call numbers associated with known fraud campaigns.

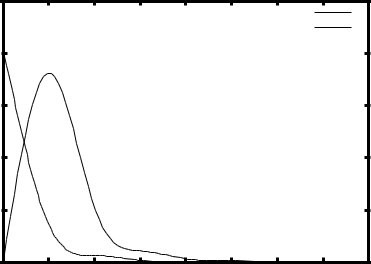
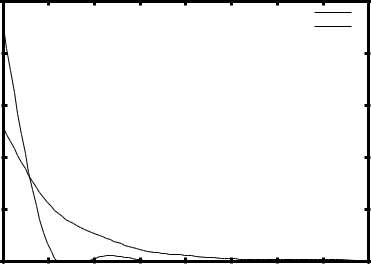
# Implement a browser extension based on kAYO: To the best of our knowledge kAYO is the first technique that detects mobile specific malicious webpages by static analysis. Existing tools such as Google Safe Browsing are not enabled on the mobile versions of browsers, thereby precluding mobile users. Moreover, the mobile specific design of kAYO enables detection of malicious mobile webpages missed by existing techniques. Finally, our survey of existing extensions on Firefox desktop browser suggests that there is a paucity of tools that help users identify mobile malicious webpages. To fill this void, we build a Firefox mobile browser ex- tension using kAYO, which informs users about the maliciousness of the webpages they intend to visit in real-time. We plan to make the extension publicly available post publication.

We note that we define maliciousness broadly, as is done in the prior literature on the static detection in the desktop space However,drive-by-downloads are not at all common in the mobile space at the time of writing, the overwhelming majority of detected pages are related to phishing

**1.1 Motivation**

Static analysis techniques to detect malicious websites of- ten use features of a webpage such as HTML, JavaScript and characteristics of the URL. Usually, these features are fed to machine learning techniques to classify benign and malicious webpages. These techniques are predicated on the assumption that the features are distributed differently across benign and malicious webpages. Accordingly, any changes in the distribution of static features in benign and/or malicious webpages impact

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  |  |  |  |  | Desktop |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| webpages |  |  |  |  |  |  | Mobile |  |  |  |
| 0.8 |  |  |  |  |  |  |  |  |  |
| 0.6 |  |  |  |  |  |  |  |  |  |
| of total |  |  |  |  |  |  |  |  |  |
| 0.4 |  |  |  |  |  |  |  |  |  |
| Fraction | 0.2 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
|  | 0 |  |



iframes per webpage

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | (a) |  |  |  |  |  |
|  | 1 |  |  |  |  |  | Desktop |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| webpages |  |  |  |  |  |  | Mobile |  |  |  |
| 0.8 |  |  |  |  |  |  |  |  |  |
| 0.6 |  |  |  |  |  |  |  |  |  |
| of total |  |  |  |  |  |  |  |  |  |
| 0.4 |  |  |  |  |  |  |  |  |  |
| Fraction | 0.2 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
|  | 0 |  |

Redirections per webpage

(c)

**Fig. 1.1: Normalized density curves of static features.**

There is substantial difference between the distributions of the number of (a)iframes (b)Javascript and (c) redirections when measured on mobile and desktop versions of the same websites, whereas, the distribution of the number of (d) IP addresses is similar Successful, these static analysis techniques have been used exclusively for desktop webpages [20], [51], [59].Mobile websites are significantly different from their desktop counterparts in content, functionality and layout. Consequently, existing tools using static features to detect malicious desktop webpages are unlikely to work for mobile webpages. We explain four factors that motivate building separate static analysis techniques to detect malicious mobile webpages.

**1.Differences in content**: Mobile websites are often simpler than their desktop counterparts. Therefore, the distribution of content-based static features (such as the number of JavaScripts) on mobile webpages differs from that of desktop webpages. For example, Figure 1 (a) and Figure 1 (b) show the normalized density of the number of iframes and the number of Javascript found in mobile2 and the corresponding desktop versions of the top-level webpage of the 10,000 most popular websites from Alexa [11]. Approximately 90% of mobile webpages do not have any iframes, whereas the corresponding desktop webpages have multiple iframes. Desktop web- pages have more Javascripts than mobile webpages.Due to the simplicity of mobile webpages, the majority of other content related static features used in exist- ing techniques including, the number of images, page length, the number of hidden elements, and the number of elements with a small area also differ in magnitude.We describe the method used to define and identify mobile in mobile and desktop webpages

**2.Infrastructure:** Website providers use JavaScript or user agent strings to identify and then redirect mobile users to a mobile specific version. Figure 1 (c) shows the normalized density of the number of redirection steps taken by the desktop and mobile versions of the top 10,000 websites on Alexa before landing on the final URL3. Even the most popular mobile websites show multiple redirects, which has traditionally been a prop- erty of desktop websites hosting malware [51]. However, multiple redirects does not necessarily indicate bad be- havior for mobile websites due to the characteristics of their hosting infrastructure.

We note that not all static features used in existing techniques differ when measured on mobile and desktop webpages. For example, the number of IP addresses re- turned by DNS servers for mobile and desktop versions of the same sites are comparable. Mobile websites appear to share their hosting infrastructure with the correspond- ing desktop websites [36]. We used seven public (Google, OpenDNS, UltraDNS, Norton, DynDNS, Level3, and Scrubit) DNS servers to obtain the IP addresses returned in the DNS A records of mobile and corresponding desktop URLs of Alexa top 10,000 websites. As seen in Figure 1 (d), the distributions of the number of IP addresses returned by the seven DNS servers are similar

We use the term final URL to denote the URL that is rendered in the browser after redirections (if any) from the seed URL. The final1536-123w3 (ecb) 2p0a1g6 eIEsEEin. Pderesotanaill uisne iSs epcertmioitnted4,.b2u.t republication/redistribution requires IEEE permissionU. RSeLe hmttpa:/y/wcwhwa.ineege.eorbg/apsuebdlicaotinontsh\_setanbdraorwds/speurb’lsicautisoenrs/raigghetsn/itn.dex.html for more information for mobile and desktop websites.

**3.Impact of screen size:** The screen size of a mobile phone is significantly smaller that that of a desktop computer. Therefore, a mobile user only sees a part of the URL of a webpage. Intuitively, the author of a mobile phishing webpage may only need to include misleading words at the beginning of the URL and a short URL might suffice to trick a user.

Mobile specific functionality: Mobile websites enable access to a user’s personal information and advanced capabilities of mobile devices through web APIs. Existing static analysis techniques do not consider these mobile specific functionalities in their feature set. We argue and later demonstrate that accounting for the mobile specific functionalities helps identify new threats specific to the mobile web. For example, the presence of a known ‘bank’ fraud number on a website might indicate that the webpage is a phishing webpage imitating the same bank [16].

**4.Limitations of existing techniques:** These discrepancies between mobile and desktop webpages demand inves- tigation. Existing static analysis techniques and tools for detecting malicious webpages are focused on desktop webpages. Therefore, they are unable to detect mobile specific threats with high accuracy.4 Secondly, several webpages built specifically for mobile, return empty pages when rendered in a desktop browser. Thus, even existing dynamic analysis techniques that execute websites in desktop browsers on virtual machines, are ineffective on such mobile websites. Finally, signature based tools such as Google Safe Browsing currently only work with desktop browsers. We manually visited five mobile spe- cific known malicious webpages collected from Phish- Tank [6], from the Google Chrome mobile browser. We observed that these webpages are flagged as malicious on the Chrome desktop browser, but not on the Chrome mobile browser whose users are the real targets of the mobile malicious webpages. Although enabling Google Safe Browsing in mobile Chrome is an engineering effort, we argue and later demonstrate that a mobile specific static technique can also detect new threats previously unseen by such services.

**CHAPTER 2**

**LITERATURE SURVEY**

**1)**Detecting malicious websites with low-interaction honeyclients

**AUTHORS:** A. Ikinci, T. Holz, and F. Freiling.

Monkey-spider: Client-side attacks are on the rise: malicious websites that exploit vulnerabilities in the visitor’s browser are posing a serious threat to client security, compromising innocent users who visit these sites without having a patched web browser. Currently, there is neither a freely available comprehensive database of threats on the Web nor sufficient freely available tools to build such a database. In this work, we introduce the Monkey-Spider project [Mon]. Utilizing it as a client honeypot, we portray the challenge in such an approach and evaluate our system as a high-speed, Internetscale analysis tool to build a database of threats found in the wild. Furthermore, we evaluate the system by analyzing different crawls performed during a period of three months and present the lessons learned.

1. A guided approach to finding malicious web pages

**AUTHORS:** L. Invernizzi, S. Benvenuti, M. Cova, P. M. Comparetti, C. Kruegel, and G. Vigna. Evilseed

Malicious web pages that use drive-by download attacks or social engineering techniques to install unwanted software on a user's computer have become the main avenue for the propagation of malicious code. To search for malicious web pages, the first step is typically to use a crawler to collect URLs that are live on the Internet. Then, fast prefiltering techniques are employed to reduce the amount of pages that need to be examined by more precise, but slower, analysis tools (such as honey clients). While effective, these techniques require a substantial amount of resources. A key reason is that the crawler encounters many pages on the web that are benign, that is, the "toxicity" of the stream of URLs being analyzed is low. In this paper, we present EVILSEED, an approach to search the web more efficiently for pages that are likely malicious. EVILSEED starts from an initial seed of known, malicious web pages. Using this seed, our system automatically generates search engines queries to identify other malicious pages that are similar or related to the ones in the initial seed. By doing so, EVILSEED leverages the crawling infrastructure of search engines to retrieve URLs that are much more likely to be malicious than a random page on the web. In other words EVILSEED increases the "toxicity" of the input URL stream. Also, we envision that the features that EVILSEED presents could be directly applied by search engines in their prefilters. We have implemented our approach, and we evaluated it on a large-scale dataset. The results show that EVILSEED is able to identify malicious web pages more efficiently when compared to crawler-based approaches.

**3)** Blog identification and splog detection.

**AUTHORS:** P. Kolari, T. Finin, and A. Joshi. Svms for the blogosphere:

Weblogs, or blogs have become an important new way to publish information, engage in discussions and form communities. The increasing popularity of blogs has given rise to search and analysis engines focusing on the 'blogosphere'. A key requirement of such systems is to identify blogs as they crawl the Web. While this ensures that only blogs are indexed, blog search engines are also often overwhelmed by spam blogs (splogs). Splogs not only incur computational overheads but also reduce user satisfaction. In this paper we first describe our experiments on blog identification using Support Vector Machines (SVM). We compare results of using different feature sets and introduce new features for blog identification. We then report preliminary results on splog detection and identify future work.

**4)** Phishdef: Url names say it all.

**AUTHORS:** A. Le, A. Markopoulou, and M. Faloutsos.

Phishing is an increasingly sophisticated method to steal personal user information using sites that pretend to be legitimate. In this paper, we take the following steps to identify phishing URLs. First, we carefully select lexical features of the URLs that are resistant to obfuscation techniques used by attackers. Second, we evaluate the classification accuracy when using only lexical features, both automatically and hand-selected, vs. when using additional features. We show that lexical features are sufficient for all practical purposes. Third, we thoroughly compare several classification algorithms, and we propose to use an online method (AROW) that is able to overcome noisy training data. Based on the insights gained from our analysis, we propose PhishDef, a phishing detection system that uses only URL names and combines the above three elements. PhishDef is a highly accurate method (when compared to state-of-the-art approaches over real datasets), lightweight (thus appropriate for online and client-side deployment), proactive (based on online classification rather than blacklists), and resilient to training data inaccuracies (thus enabling the use of large noisy training data).

**5)** The core of the matter: Analyzing malicious traffic in cellular carriers.

**AUTHORS:** C. Lever, M. Antonakakis, B. Reaves, P. Traynor, and W. Lee

Much of the attention surrounding mobile malware has focused on the in-depth analysis of malicious applications. While bringing the community valuable information about the methods used and data targeted by malware writers, such work has not yet been able to quantify the prevalence with which mobile devices are actually infected. In this paper, we present the first such attempt through a study of the hosting infrastructure used by mobile applications. Using DNS traffic collected over the course of three months from a major US cellular provider as well as a major US noncellular Internet service provider, we identify the DNS domains looked up by mobile applications, and analyze information related to the Internet hosts pointed to by these domains. We make several important observations. The mobile malware found by the research community thus far appears in a minuscule number of devices in the network: 3,492 out of over 380 million (less than 0.0009%) observed during the course of our analysis. This result lends credence to the argument that, while not perfect, mobile application markets are currently providing adequate security for the majority of mobile device users. Second, we find that users of iOS devices are virtually identically as likely to communicate with known low reputation domains as the owners of other mobile platforms, calling into question the conventional wisdom of one platform demonstrably providing greater security than another. Finally, we observe two malware campaigns from the upper levels of the DNS hierarchy and analyze the lifetimes and network properties of these threats. We also note that one of these campaigns ceases to operate long before the malware associated with it is discovered suggesting that network-based countermeasures may be useful in the identification and mitigation of future threat.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1 Existing System**

* A popular approach in detecting malicious activity on the web is by leveraging distinguishing features between malicious and benign DNS usage.
* Both passive DNS monitoring and active DNS probing methods have been used to identify malicious domains. While some of these efforts focused solely on detecting fast flux service networks, another can also detect domains implementing phishing and drive-by-downloads.
* The best-known non-proprietary content-based approach to detect phishing webpages is Cantina

**Disadvantages**

* Existing tools such as Google Safe Browsing are not enabled on the mobile versions of browsers, thereby precluding mobile users.
* DNS based mechanisms do not provide deeper understanding of the specific activity implemented by a webpage or domain.
* Downloading and executing each webpage impacts performance and hinders scalability of dynamic approaches.
* URL-based techniques usually suffer from high false positive rates.
* Cantina suffers from performance problems due to the time lag involved in querying the Google search engine. Moreover, Cantina does not work well on webpages written in languages other than English.
* Finally, existing techniques do not account for new mobile threats such as known fraud phone numbers that attempt to trigger the dialer on the phone.

**3.2 Proposed System**

* In this paper, we present kAYO, a fast and reliable static analysis technique to detect malicious mobile web-pages. kAYO uses static features of mobile webpages derived from their HTML and JavaScript content, URL and advanced mobile specific capabilities.
* We first experimentally demonstrate that the distributions of identical static features when extracted from desktop and mobile webpages vary dramatically
* We experimentally demonstrate that the distributions of static features used in existing techniques (e.g., the number of redirections) are different when measured on mobile and desktop webpages. Moreover, we illustrate that certain features are inversely correlated or unrelated to or non-indicative to a webpage being malicious when extracted from each space.

**Advantages**:

* kAYO also detects a number of malicious mobile webpages not precisely detected by existing techniques such as VirusTotal and Google Safe Browsing.
* The results of our experiments demonstrate the need for mobile specific techniques for detecting malicious webpages.
* To the best of our knowledge kAYO is the first technique that detects mobile specific malicious webpages by static analysis.
* Moreover, the mobile specific design of Kayo enables detection of malicious mobile webpages missed by existing techniques.
* Finally, our survey of existing extensions on Firefox desktop browser suggests that there is a paucity of tools that help users identify mobile malicious webpages.

**3.3 Feasibility**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* Economical Feasibility
* Technicial Feasibility
* Social Feasibility

**3.3.1 Economical Feasibility**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### 3.3.2 Technical Feasibility

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

**3.3.3 Social Feasibility**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

**3.4 System Requirements:**

**3.4.1 Hardware Requirements:**

* System : Pentium IV 2.4 GHz.
* Hard Disk : 40 GB.
* Floppy Drive : 1.44 Mb.
* Monitor : 15 VGA Colour.
* Mouse : Logitech.
* Ram : 512 Mb.

**3.4.2 Software requirements:**

* Operating system : Windows XP/7.
* Coding Language : JAVA/J2EE
* IDE : Netbeans 7.4
* Database : MYSQL

**CHAPTER 4**

**SYSTEM DESIGN**

**Input Design**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

* What data should be given as input?
* How the data should be arranged or coded?
* The dialog to guide the operating personnel in providing input.
* Methods for preparing input validations and steps to follow when error occur.

**Objectives**

1.Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

1. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

**Output Design**

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system’s relationship to help user decision-making.

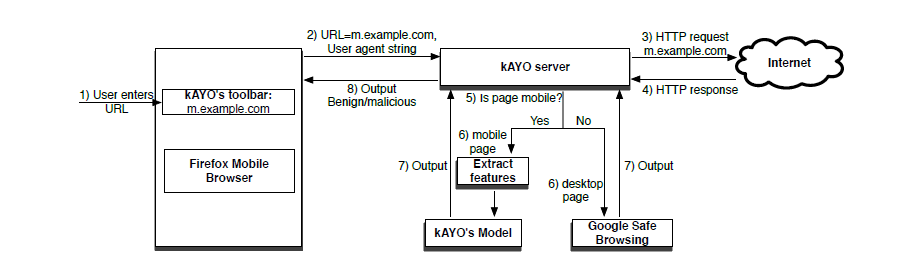
1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.
2. Select methods for presenting information.

3.Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

* Convey information about past activities, current status or projections of the
* Future.
* Signal important events, opportunities, problems, or warnings.
* Trigger an action.
* Confirm an action.

**4.1 System Architecture**



**Fig 4.1** System architecture

**4.2 kAYO Feature Set**

A webpage has several components including HTML and JavaScript code, images, the URL, and the header. Mobile specific webpages also access applications run-ning on a user’s device using web APIs (e.g., the dialer). We extract structural, lexical and quantitative properties of such components to generate kAYO’s feature set. We focus on extracting mobile relevant features that take minimal extraction time. Our hypothesis is that such features are strong indicators of whether a webpage has been built for assisting a user in their web browsing experience or for malicious purposes.

Our feature set consists of 44 features, 11 of which are new and not previously identified or used. We describe these new features in detail. A subset of features in kAYO have been used by other authors in static inspection of desktop webpages in the past.5 However, it is important to note that these features in mobile webpages and desktop webpages differ in magnitude (e.g., number of iframes) and show varying correlation with the nature of the webpage (i.e., malicious/benign).

We divide kAYO’s 44 features into four classes: mobile specific-, JavaScript, HTML and URL features. To the best of our knowledge, we are the first to use these mobile specific features, and do not claim novelty on using subsets of other previously identified features. Table 1 summarizes the 8 mobile, 10 JavaScript, 14 HTML and 12 URL features. We empirically illustrate the effectiveness of each of the features in Section 5.2.

**4.2.1 Mobile specific features**

We collect eight mobile specific features to capture the advanced capabilities of mobile webpages. Mobile websites enable access to personal data from a user’s phone, an experience not offered by desktop websites. For example, mobile web APIs such as tel: and sms: spawn the dialer and the SMS applications respectively on a mobile device. In order to characterize the behavior of mobile API calls, we extracted the number of API calls tel:, sms:, smsto:, mms: and mmsto: from each mobile webpage. We further extracted the target phone numbers from these API calls. We ran the commercially available Pindrop Security Phone Reputation System (PRS) [7] on each phone number. Based on the results of the PRS, we gave the score of 1/0 (known fraud/benign) to each phone number scraped from the mobile API calls, and added the score as a feature in kAYO. We only extracted phone numbers with API prefixes that could trigger an application installed on a user’s phone.

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Features | Total # of features |  |
|  |  |  |  |
| Mobile specific | # of API calls to tel:, sms:, smsto:, mms:, mmsto:, geolocation; # of apk, # of ipa | 8 |  |
| JavaScript | presence of JS, noscript, internal JS, external JS, embedded JS; | 10 |  |
| # of JS, noscript, internal JS, external JS, embedded JS |  |
|  |  |  |
|  | presence of internal links, external links, images; # of internal links, external links, images |  |  |
| HTML | # of cookies from header, secure and HTTPOnly cookies, | 14 |  |
| presence of redirections and iframes, # of redirects and iframes, |  |
|  |  |  |
|  | whether webpage served over SSL, % of white spaces in the HTMl content |  |  |
|  | # of misleading words in the URL such as login and bank , length of URL |  |  |
| URL | # of forward slashes and question marks, digits, dots, hyphens and underscores, | 12 |  |
| # of equal signs and ampersand, subdomains, two letter subdomains, semicolons, |  |
|  |  |  |
|  | presence of subdomain, % of digits in hostname |  |  |
|  | Total: | 44 |  |

**TABLE 1:** The 44 features of kAYO from four categories. The significance of both new mobile and prior features res is evaluated in Section 5.2 did not consider phone number strings simply listed on webpages without an API prefix.

We argue that due to the popularity of application markets such as Google play and iTunes, a website hosting its own mobile application binary (e.g., .apk or .ipa files) possibly suggests bad behavior. If we found more than a threshold (in the few hundreds) of apk/ipa files on the same webpage, we assumed that the webpage was a third-party app store (of which there are many) and was unlikely to be malicious.

**4.2.2 JavaScript features**

JavaScript enables client-side user interaction, asynchronous communication with servers, and modification of the DOM objects of webpages on the fly. We extract 10 features that capture the JavaScript relevant static behavior of a webpage, two of which are new. All the features are faster to extract than the features based on JavaScript deobfuscation.

JavaScript found on malicious webpages can be obfuscated. Instead of deobfuscating every JavaScript, we ex-tract simple JavaScript related features from a webpage. The primary reason in choosing this approach is that a large number of benign webpages include potentially dangerous JavaScript code as shown by Yue et al. [57]. For example, 44.4% of the top 6,805 websites from Alexa use the potentially dangerous eval function. These observations invalidate the assumption made in existing techniques [20]; that potentially dangerous JavaScript keywords are more frequently used in malicious web-pages. Secondly, external JavaScript can be very large, sometimes of the order of a few megabytes. Our goal is to build a real-time browser extension based on kAYO. Accordingly, we avoided using features that would slow down the feature extraction process.

Webpages generally include three types of JavaScript: internal, external and embedded. An internal JavaScript is one hosted on the same domain as that of its parent webpage, whereas, an external JavaScript’s domain is different from its host’s domain. Since mobile web-pages are often simpler than desktop webpages and phishing is the biggest threat on mobile webpages at present, we expect that benign webpages will include more external JavaScript for advertisements and analytics purposes, whereas malicious webpages will have a lower number of external JavaScript. Accordingly, we determine whether a webpage holds external and inter-nal JavaScript, and then extract the number of internal and external JavaScript from a webpage. Unlike internal and external JavaScript, embedded JavaScript code is contained in the webpage. If the number of lines of JavaScript is relatively small, a webpage with embedded

JavaScript loads faster than pages that must reference external code. This is because, as the web browser loads the page and encounters the reference to the external code, it must make a separate request to the web server to fetch the code. Webpages built for performance often use a number of embedded JavaScript. Performance is critical in the mobile web since it impacts revenue and user interest [50]. Therefore, we determine whether a webpage hosts embedded JavaScript and then calculate the number of embedded JavaScript in a webpage. Our hypothesis is that on average, benign webpages will have more embedded JavaScript. Finally, we determine whether JavaScript is present at all on a webpage, and measure the total number of JavaScript on the webpage including embedded, internal and external. Note that we believe that this feature is indicative, but not an alarm by itself as malicious pages could also seek to gain revenue from advertisement related to the latest breaking news. Accordingly, we first determine whether a webpage has any images, internal and external HTML links. We then extract the number of internal links, external links and images from a webpage as features of kAYO.

Malicious webpages (especially those implementing drive-by-downloads and clickjacking) include links to bad content in iframes [51]. Recall that the distribution of iframes on mobile webpages is different as compared to that on desktop webpages. However, we do not rule out the possibility of a mobile malicious webpage including malicious content in iframes and consider the presence and number of iframes in a webpage as features in kAYO. Past research also shows that malicious websites take several redirections before leading the user to the target webpage to avoid DNS based detection [51]. Recall that mobile webpages generally take at least one or more redirections since both desktop and mobile versions of the webpage share hosting infrastructure. Therefore, we determine whether a webpage was redirected and then measure the number of redirections the user experiences before landing on the final URL. Finally, we extract other features such as the percentage of white spaces in the HTML content, the number of cookies from the header, the number of secure and HTTPOnly cookies, and whether the webpage is served over an SSL connection. Readers are encouraged to refer to prior literature [20], [41], [59] for more information on the usefulness of these HTML features

**4.2.3 URL features**

Structural and lexical properties of a URL have been used to differentiate between malicious and benign web-pages. However, using only URL features for such differentiation leads to a high false positive rate. We extract 12 URL features in total.

Authors of phishing webpages often exploit the familiarity of users to a webpage [22] by including words in the URL that can mislead a user into believing that the phishing webpage is the legitimate webpage. Words such as login and bank are commonly used in the URL of the login webpage for benign websites that are highly prone to imitation. Only a part of the URL is visible to the user of a mobile phone due to the small screen [13]. Therefore, intuitively, the author of a phishing webpage will include misleading words at the beginning of the URL. We consider the presence of such words in the URL as a new feature in kAYO.

A significant number of phishing domain names are simply IP addresses of machines hosting them [28],

Therefore, we calculated the number of digits in a URL and the percentage of digits in the hostname. Phishing webpage developers usually create a number of subdomains to include deceptive keywords such as paypal as a subdomain. This might increase the length of phishing URLs [40]. Therefore, we include the length of a URL, whether the URL contains a subdomain,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Top Level Domain |  | .mobi | | |  |
| Subdomain |  | m., mobile., touch., 3g., sp., | | |  |
|  | s., mini., mobileweb., t. | | |  |
|  |  |  |
| URL Path Prefix |  | /mobile, /mobileweb, /m, /mobi, | | |  |
|  | /?m=1, /mobil, /m home | | |  |
|  |  |  |
|  |  |  |  |  |  |

TABLE 2: Indicators of mobile specific webpages ex-tracted by manual analysis of the top-level mobile and desktop webpages of the 1,000 most popular websites on Alexa. We identified one top-level domain (TLD), nine subdomains and seven URL path prefixes.

as features. Our URL feature set also contains the number of semicolons, equal signs and ampersand symbols, hyphens and underscores, forward slashes and question marks. Interested readers are referred to prior literature [28], [34], [39] for details on the importance of these URL features.

Note that the HTML, JavaScript and URL features are not specific to mobile and can be used for analyzing desktop webpages as well. However, the mobile features derived from mobile applications such as dialer and SMS do not apply to desktop webpages.

**4.3 Data Collection**

Our data gathering process included accumulating labeled benign and malicious mobile specific webpages. First, we describe an experiment that identifies and defines ‘mobile specific webpages’. We then conduct the data collection process over three months in 2013. We use these crawls specifically because they are close to the publication of the related work, making them as close to equivalent as possible.

Identification of mobile specific webpages: We crawled the top-level webpage of the 1,000 most popular websites from Alexa.com [11] using the Android mobile and desktop Internet Explorer (IE) browsers. We used Android mobile version 4.0 and IE desktop version 9.0 for Windows 7. We then manually analyzed each pair of final URLs for the same seed URL when crawled from each browser. Before classifying a URL as mobile specific, we confirmed that the final URLs for desktop and mobile were different for the same seed URL. We also compared the contents of each pair of desktop and mobile webpages, and ensured that the two contents were different. We ignored all the seed URLs that led to an identical final URL when crawled from the desktop and the mobile browser. Our analysis identified nine subdomains (e.g., m.) and seven URL path prefixes (e.g., /mobile) in the URLs of popular websites to represent their mobile specific webpages. Additionally, we considered all URLs with the ‘.mobi’ Top Level Domain (TLD) to be mobile sites [12]. We defined a mobile specific webpage as one containing Alexa rank.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| URLs | 1200 |  |  |  |  |  |  |
| 1000 |  |  |  |  |  |  |
| webpages/10k |  |  |  |  |  |  |
| 800 |  |  |  |  |  |  |
| 600 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| # of mobile | 400 |  |  |  |  |  |  |
| 200 |  |  |  |  |  |  |
|  | 0 | 200000 | 400000 | 600000 | 800000 | 1e+06 |  |
|  | 0 |  |

**Fig. 2:** Number of mobile specific websites found in every 10,000 websites in the Alexa top 1M.

differences in content from the corresponding desktop webpage. Table 2 summarizes the mobile indicators.

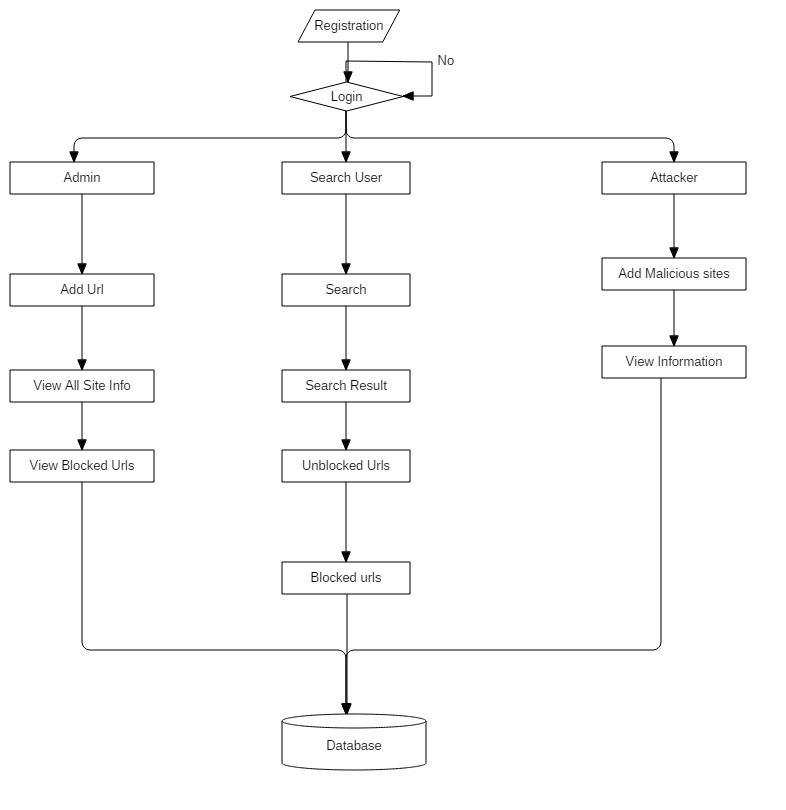
Building the dataset: To generate training data for our model, we statically crawled the top-level webpage of the top 1,000,000 most popular websites from Alexa from an Android mobile browser. We then extracted the mobile specific webpages using the algorithm described above. Figure 2 shows the number of top-level mobile specific webpages found in the dataset. 1,244 out of the first 10,000 most popular websites offer a mobile specific version and 763 maintain mobile specific webpages in the 10,000-20,000 range. From 20,000 onwards upto one million, the number of mobile specific webpages found using our algorithm is largely constant. We observed that 485 out of the top one million Alexa websites have the ‘.mobi’ TLD. Using the 17 mobile indicators defined in Table 2, we collected 53,638 mobile specific URLs at the top-level by statically crawling each website in Alexa from an Android mobile browser. We then crawled each of the 53,638 mobile specific websites two levels deep. Interestingly, we found links to several non-mobile URLs on the mobile specific webpages. We discarded non-mobile webpages and were left with 295,512 mobile URLs at depth two. In total, we derived 349,150 mobile URLs from the Alexa one million websites.

Gathering data for malicious mobile URLs was chal-lenging since the mobile web is still evolving and new threats are emerging. We monitored several public blacklists [2], [3], [5] continually for three months and extracted mobile specific URLs from the blacklists. We set up a continuous feed from two public blacklists and crawled newly uploaded malicious URLs every two seconds. We also monitored PhishTank’s [6] online dataset for mobile specific phishing URLs. After moni-toring these sources for three months, we gathered data from 531 top-level and 4,681 depth two mobile specific malicious URLs. Note that our dataset also contains mobile URLs that were submitted to the blacklists before 2013, but were live at the time of crawling.

The Google Safe Browsing tool performs both static and dynamic analysis on webpages [51]. It first discards benign webpages identified using static analysis and then performs dynamic analysis on the webpages tagged as malicious. VirusTotal queries 41 different malware de-tection tools based on dynamic analysis, crowd sourcing and signatures. To be conservative, we labeled a URL as malicious only when Google Safe Browsing tagged a URL as malicious, or four or more tools queried by VirusTotal labeled the URL as malicious. We also performed manual inspection if necessary. For example, the URLs from PhishTank are crowdsourced, and Google Safe Browsing and VirusTotal do not detect all valid URLs from PhishTank as malicious. We manually visited such URLs to ensure that they are phishing webpages. Our final dataset consisted of 349,137 benign URLs and 5,231 malicious URLs. We used this dataset to train kAYO’s model. We note that we waited for a number of months to determine if many of our pages were ever classified by this engine, so as to give other detection tools time to discover the candidate sites [43]. Finally, we used the lifetime of pages that were clear scams (e.g., banking pages that disappeared within 24 hours) to judge a small subset of pages.

**4.4 System Flowchart**

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.



**Fig 4.2** Flowchart

**4.5 UML Diagrams**

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

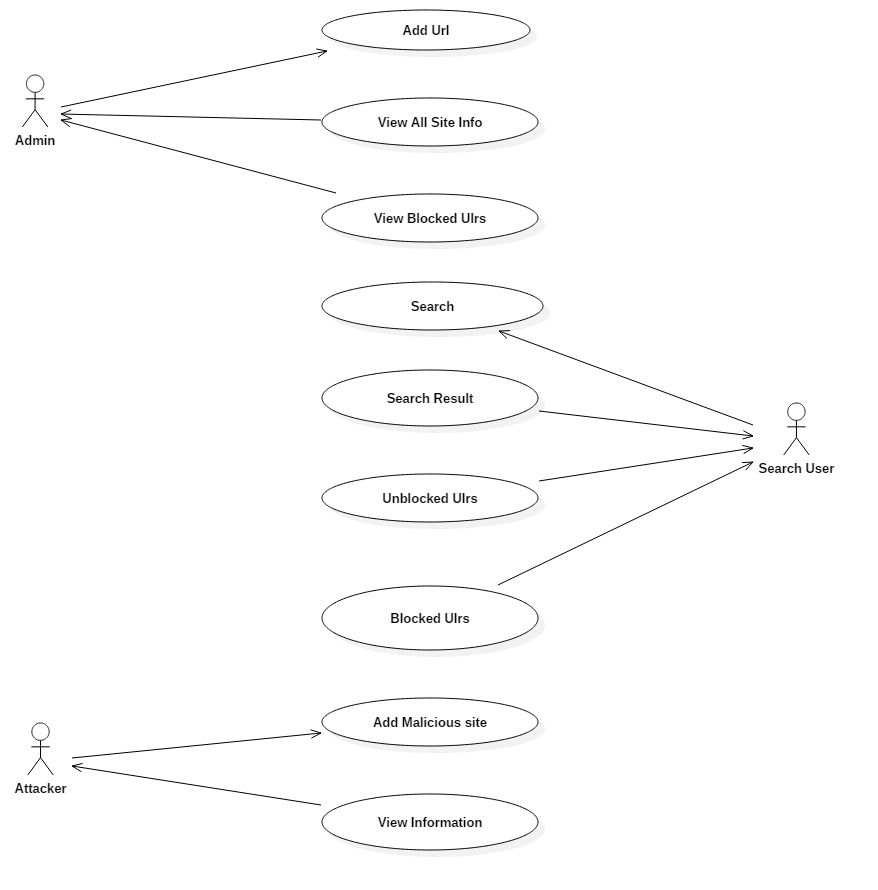
The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

**4.5.1 Use Case Diagram**

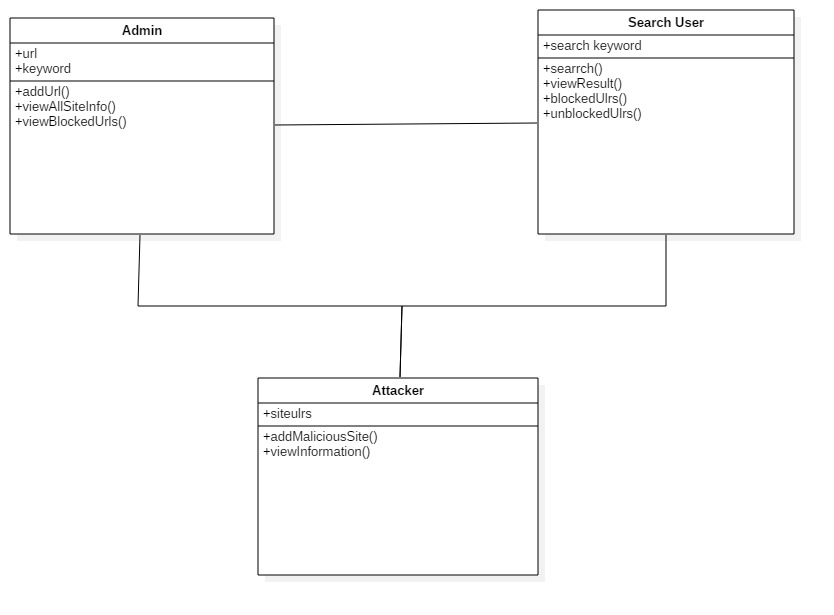
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



**Fig 4.3** Use Case Diagram

**4.5.2 Class Diagram**

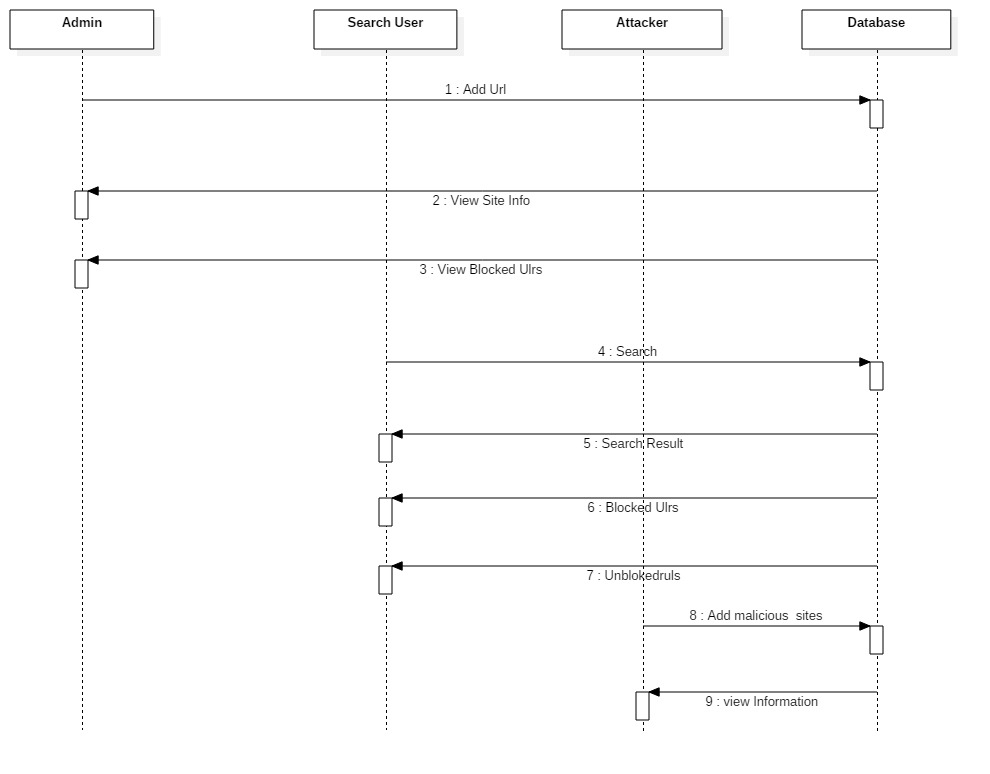
In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

****

**Fig 4.4** Class Diagram

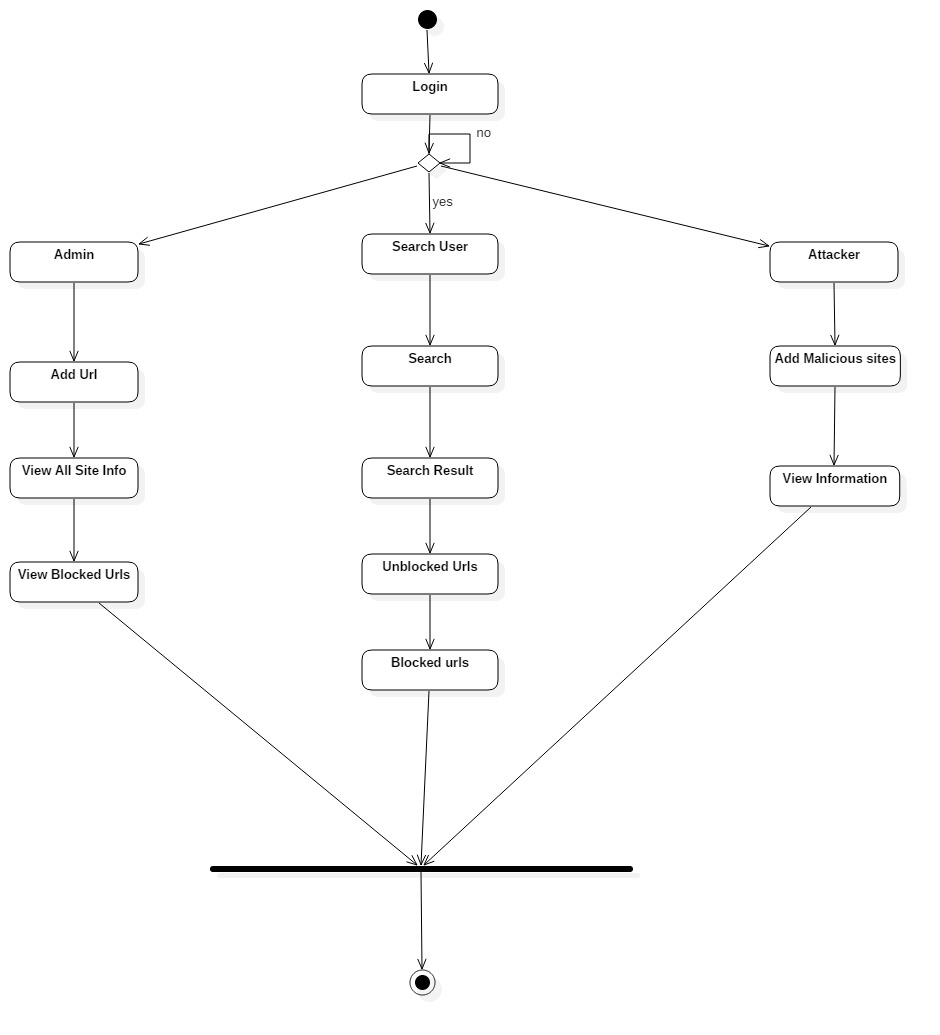
**4.5.3 Sequence Diagram**

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

****

**Fig 4.5** Sequence Diagram

**4.5.4 Activity diagrams**

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components.****

**Fig 4.5** Diagram Activity Diagram

**CHAPTER 5**

**IMPLEMENTATION**

**5.1 Implementation Tools**

Microsoft .NET is a set of Microsoft software technologies for rapidly building and integrating XML Web services, Microsoft Windows-based applications, and Web solutions. The .NET Framework is a language-neutral platform for writing programs that can easily and securely interoperate. There’s no language barrier with .NET: there are numerous languages available to the developer including Managed C++, C#, Visual Basic and Java Script. The .NET framework provides the foundation for components to interact seamlessly, whether locally or remotely on different platforms. It standardizes common data types and communications protocols so that components created in different languages can easily interoperate.

“.NET” is also the collective name given to various software components built upon the .NET platform. These will be both products (Visual Studio.NET and Windows.NET Server, for instance) and services (like Passport, .NET My Services, and so on).

**5.1.1 The .NET Framework**

The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).

2. A hierarchical set of class libraries.

The CLR is described as the “execution engine” of .NET. It provides the environment within which programs run. The most important features are

* Conversion from a low-level assembler-style language, called Intermediate Language (IL), into code native to the platform being executed on.
* Memory management, notably including garbage collection.
* Checking and enforcing security restrictions on the running code.
* Loading and executing programs, with version control and other such features.
* The following features of the .NET framework are also worth description:

**Managed Code**

The code that targets .NET, and which contains certain extraInformation - “metadata” - to describe itself. Whilst both managed and unmanaged code can run in the runtime, only managed code contains the information that allows the CLR to guarantee, for instance, safe execution and interoperability.

**Managed Data**

With Managed Code comes Managed Data. CLR provides memory allocation and Deal location facilities, and garbage collection. Some .NET languages use Managed Data by default, such as C#, Visual Basic.NET and JScript.NET, whereas others, namely C++, do not. Targeting CLR can, depending on the language you’re using, impose certain constraints on the features available. As with managed and unmanaged code, one can have both managed and unmanaged data in .NET applications - data that doesn’t get garbage collected but instead is looked after by unmanaged code.

**Common Type System**

CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling. As well as ensuring that types are only used in appropriate ways, the runtime also ensures that code doesn’t attempt to access memory that hasn’t been allocated to it.

**Common Language Specification**

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

**The Class Library**

.NET provides a single-rooted hierarchy of classes, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System. Object. As well as objects, there are value types. Value types can be allocated on the stack, which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary.

The set of classes is pretty comprehensive, providing collections, file, screen, and network I/O, threading, and so on, as well as XML and database connectivity.

The class library is subdivided into a number of sets (or namespaces), each providing distinct areas of functionality, with dependencies between the namespaces kept to a minimum.

**Languages Supported by .NET**

The multi-language capability of the .NET Framework and Visual Studio .NET enables developers to use their existing programming skills to build all types of applications and XML Web services. The .NET framework supports new versions of Microsoft’s old favorites Visual Basic and C++ (as VB.NET and Managed C++), but there are also a number of new additions to the family.

Visual Basic .NET has been updated to include many new and improved language features that make it a powerful object-oriented programming language. These features include inheritance, interfaces, and overloading, among others. Visual Basic also now supports structured exception handling, custom attributes and also supports multi-threading.

Visual Basic .NET is also CLS compliant, which means that any CLS-compliant language can use the classes, objects, and components you create in Visual Basic .NET.

Managed Extensions for C++ and attributed programming are just some of the enhancements made to the C++ language. Managed Extensions simplify the task of migrating existing C++ applications to the new .NET Framework.

C# is Microsoft’s new language. It’s a C-style language that is essentially “C++ for Rapid Application Development”. Unlike other languages, its specification is just the grammar of the language. It has no standard library of its own, and instead has been designed with the intention of using the .NET libraries as its own.

Microsoft Visual C# .NET provides the easiest transition for Java-language developers into the world of XML Web Services and dramatically improves the interoperability of Java-language programs with existing software written in a variety of other programming languages.

Active State has created Visual Perl and Visual Python, which enable .NET aware applications to be built in either Perl or Python. Both products can be integrated

Other languages for which .NET compilers are available include

* FORTRAN
* COBOL
* Eiffel

|  |  |
| --- | --- |
| ASP.NET  XML WEB SERVICES | Windows Forms |
| Base Class Libraries | |
| Common Language Runtime | |
| Operating System | |

**Fig 5.1 Layered representation**

C#.NET is also compliant with CLS (Common Language Specification) and supports structured exception handling. CLS is set of rules and constructs that are supported by the CLR (Common Language Runtime). CLR is the runtime environment provided by the .NET Framework; it manages the execution of the code and also makes the development process easier by providing services.

C#.NET is a CLS-compliant language. Any objects, classes, or components that created in C#.NET can be used in any other CLS-compliant language. In addition, we can use objects, classes, and components created in other CLS-compliant languages in C#.NET .The use of CLS ensures complete interoperability among applications, regardless of the languages used to create the application.

**CHAPTER 6**

### SYSTEM TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

**6.1 Testing Methodologies**

**Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

**Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program.Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of

**Functional testing**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.

Invalid Input : identified classes of invalid input must be rejected.

Functions : identified functions must be exercised.

Output : identified classes of application outputs must be exercised.

Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows;0 data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

**System Testing**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

**White Box Testing**

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

**Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

**6.1 Unit Testing:**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

**Test objectives**

* All field entries must work properly.
* Pages must be activated from the identified link.
* The entry screen, messages and responses must not be delayed.

**Features to be tested**

* Verify that the entries are of the correct format
* No duplicate entries should be allowed
* All links should take the user to the correct page.

# 6.2 Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**6.3 Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

**CHAPTER 7**

**CONCLUSION**

Mobile webpages are significantly different than their desktop counterparts in content, functionality and layout. Therefore, existing techniques using static features of desktop webpages to detect malicious behavior do not work well for mobile specific pages. We designed and developed a fast and reliable static analysis technique called KAYO that detects mobile malicious webpages. KAYO makes these detections by measuring 44 mobile relevant features from webpages, out of which 11 are newly identified mobile specific features. KAYO provides 90% accuracy in classification, and detects a number of malicious mobile webpages in the wild that are not detected by existing techniques such as Google Safe Browsing and VirusTotal. Finally, we build a browser extension using KAYO that provides real-time feedback to users. We conclude that KAYO detects new mobile specific threats such as websites hosting known fraud numbers and takes the first step towards identifying new security challenges in the modern mobile web.

**BIBILOGRAPHY**

[1] Gnu octave: high-level interpreted language. http://www.gnu.org/software/octave/.

[2] hphosts, a community managed hosts file. http://hphosts.gt500.org/hosts.txt.

[3] Joewein.de LLC blacklist. http://www.joewein.net/dl/bl/dom-bl-base.txt.

[4] Lookout. https://play.google.com/store/apps/details?hl=en&id=com.lookout.

[5] Malware Domains List. http://mirror1.malwaredomains.com/files/domains.txt.

[6] Phishtank. http://www.phishtank.com/.

[7] Pindrop phone reputation service. http://pindropsecurity.com/phone-fraud-solutions/phone reputation service prs/.

[8] Scrapy — an open source web scraping framework for python. http://scrapy.org/.

[9] VirusTotal. https://www.virustotal.com/en/.

[10] Google developers: Safe Browsing API. https://developers.google.com/safe-browsing/, 2012.

[11] Alexa, the web information company. http://www.alexa.com/topsites,

2013.

[12] dotmobi. internet made mobile. anywhere, any device. http://dotmobi.

com/, 2013.

[13] C. Amrutkar, K. Singh, A. Verma, and P. Traynor. VulnerableMe: Measuring

systemic weaknesses in mobile browser security. In Proceedings of the

International Conference on Information Systems Security (ICISS), 2012.

[14] C. Amrutkar, P. Traynor, and P. C. van Oorschot. Measuring SSL indicators

on mobile browsers: Extended life, or end of the road? In Proceedings of the

Information Security Conference (ISC), 2012.

[15] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building

a dynamic reputation system for DNS. In Proceedings of the 19th USENIX

Conference on Security (SECURITY), 2010.

[16] V. A. Balasubramaniyan, A. Poonawalla, M. Ahamad, M. T. Hunter, and

P. Traynor. Pindr0p: using single-ended audio features to determine call

provenance. In Proceedings of the 17th ACM Conference on Computer and

Communications Security (CCS), 2010.

[17] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. EXPOSURE : Finding

malicious domains using passive DNS analysis. In Proceedings of the 18th

Annual Network and Distributed System Security Symposium (NDSS), 2011.

[18] M. Boodaei. Mobile users three times more vulnerable to phishing attacks.

http://www.trusteer.com/blog/mobile-users-threetimes-

more-vulnerable-to-phishing-attacks, 2011.

[19] M. Butkiewicz, Z. Wu, S. Li, P. Murali, V. Hristidis, H. V. Madhyastha,

and V. Sekar. Enabling the transition to the mobile web with websieve. In

Proceedings of the 14thWorkshop on Mobile Computing Systems and Applications

(HotMobile), 2013.

[20] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: a fast filter for

the large-scale detection of malicious web pages. In Proceedings of the 20th

International Conference on World Wide Web (WWW), 2011.

[21] S. Chakradeo, B. Reaves, P. Traynor, andW. Enck. MAST: Triage for Marketscale

Mobile Malware Analysis. In Proceedings of the ACM Conference on

Security and Privacy in Wireless and Mobile Networks (WiSec), 2013.

[22] R. Dhamija, J. D. Tygar, and M. Hearst. Why phishing works. In Proceedings

of the SIGCHI Conference on Human Factors in Computing Systems, 2006.

[23] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android

application security. In Proceedings of the 20th USENIX Security Symposium,

2011.

[24] B. Feinstein and D. Peck. Caffeine monkey: Automated collection, detection

and analysis of malicious javascript. In Proceedings of the Black Hat Security

Conference, 2007.

[25] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions

demystified. In Proceedings of the 18th ACM conference on Computer and

communications security, 2011.

[26] A. P. Felt and D. Wagner. Phishing on mobile devices. In Web 2.0 Security

and Privacy (W2SP), 2011.

[27] A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission

re-delegation: attacks and defenses. In Proceedings of the 20th USENIX

conference on Security, 2011.

[28] I. Fette, N. Sadeh, and A. Tomasic. Learning to detect phishing emails. In

Proceedings of the 16th International Conference on World Wide Web (WWW),

2007.

[29] S. Gajek, A.-R. Sadeghi, C. St ¨ uble, and M. Winandy. Compartmented

security for browsers or how to thwart a phisher with trusted computing.

In Second International Conference on Availability, Reliability and Security

(ARES), 2007.

[30] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection

and measurement of phishing attacks. In Proceedings of the ACM workshop

on recurring malcode, 2007.

[31] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling. Measuring and detecting

fast-flux service networks. In Proceedings of the Network and Distributed

System Security Symposium (NDSS), 2008.

[32] A. Ikinci, T. Holz, and F. Freiling. Monkey-spider: Detecting malicious

websites with low-interaction honeyclients. In Proceedings of Sicherheit,

Schutz und Zuverlassigkeit, 2008.

[33] L. Invernizzi, S. Benvenuti, M. Cova, P. M. Comparetti, C. Kruegel, and

G. Vigna. Evilseed: A guided approach to finding malicious web pages. In

Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012.

[34] P. Kolari, T. Finin, and A. Joshi. Svms for the blogosphere: Blog identification

and splog detection. In Proceedings of AAAI Spring Symposium on

Computational Approaches to Analysing Weblogs, 2006.

[35] A. Le, A. Markopoulou, and M. Faloutsos. Phishdef: Url names say it all.

In Proceedings of IEEE International Conference on Computer Communications

(INFOCOM), 2011.

[36] C. Lever, M. Antonakakis, B. Reaves, P. Traynor, and W. Lee. The core of

the matter: Analyzing malicious traffic in cellular carriers. In Proceedings

of Network and Distributed System Security Symposium (NDSS), 2013.

[37] P. Likarish, E. Jung, and I. Jo. Obfuscated malicious javascript detection

using classification techniques. In Proceedings of Malicious and Unwanted

Software (MALWARE), 2009.

[38] C. Ludl, S. Mcallister, E. Kirda, and C. Kruegel. On the effectiveness of

techniques to detect phishing sites. In Proceedings of the 4th International

Conference on Detection of Intrusions and Malware, and Vulnerability Assessment

(DIMVA), 2007.

[39] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning

to detect malicious web sites from suspicious URLs. In Proceedings of the

SIGKDD Conference, 2009.

[40] D. K. McGrath and M. Gupta. Behind phishing: an examination of phisher

modi operandi. In Proceedings of the 1st Usenix Workshop on Large-Scale

Exploits and Emergent Threats (LEET), 2008.

[41] E. Medvet, E. Kirda, and C. Kruegel. Visual-similarity-based phishing

detection. In Proceedings of International Conference on Security and Privacy

in Communication Netowrks (SecureComm), 2008.

[42] Y. min Wang, D. Beck, X. Jiang, R. Roussev, C. Verbowski, S. Chen, and

S. King. Automated web patrol with strider honeymonkeys: Finding web

sites that exploit browser vulnerabilities. In Proceedings of the Networking

and Distributed Systems Security (NDSS), 2006.

[43] A. Mohaisen and O. Alrawi. AV-Meter: An Evaluation of Antivirus Scans

and Labels. In Proceedings of the 4th International Conference on Detection of

Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2014.

[44] T. Moore, R. Clayton, and H. Stern. Temporal correlations between spam

and phishing websites. In Proceedings of the 2nd USENIX conference on Largescale

exploits and emergent threats: botnets, spyware, worms, and more (LEET),

2009.

[45] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A crawler-based

study of spyware on the web. In Proceedings of Network and Distributed

System Security Symposium (NDSS), 2006.

[46] J. Nazario. Phoneyc: a virtual client honeypot. In Proceedings of the

2nd USENIX conference on Large-scale Exploits and Emergent Threats: botnets,

spyware, worms, and more (LEET), 2009.

[47] J. Nazario and T. Holz. As the net churns: Fast-flux botnet observations. In

Proceedings of 3rd International Conference on Malicious and Unwanted Software

(MALWARE), 2008.

[48] E. Passerini, R. Paleari, L. Martignoni, and D. Bruschi. Fluxor: Detecting

and monitoring fast-flux service networks. In Proceedings of the 5th International

Conference on Detection of Intrusions and Malware, and Vulnerability

Assessment (DIMVA), 2008.

[49] R. Perdisci, I. Corona, D. Dagon, and W. Lee. Detecting malicious flux

service networks through passive analysis of recursive DNS traces. In

Proceedings of Annual Computer Security Applications Conference (ACSAC),

2009.

[50] G. Podjarny. Mobile web performance optimization.

http://www.slideshare.net/blazeio/

mobile-web-performance-optimization-tips-and-tricks.

[51] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your

iframes point to us. In Proceedings of the 17th USENIX conference on Security

(SECURITY), 2008.

[52] L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation

coefficient. The American Statistician, 1988.

[53] C. Seifert, I. Welch, and P. Komisarczuk. Identification of malicious web

pages with static heuristics. In Telecommunication Networks and Applications

Conference, 2008.

[54] F. Weimer. Passive DNS replication. 2005.

[55] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification

of phishing pages. In Proceedings of the Networking and Distributed Systems

Security (NDSS), 2010.

[56] G. Xiang, J. Hong, C. P. Rose, and L. Cranor. Cantina+: A feature-rich

machine learning framework for detecting phishing web sites. ACM

Transactions on Information and System Security (TISSEC), 14(2), Sept. 2011.

[57] C. Yue and H. Wang. Characterizing insecure javascript practices on the

web. In Proceedings of the 18th international conference on World Wide Web

(WWW), 2009.

[58] B. Zdrnja, N. Brownlee, and D. Wessels. Passive monitoring of DNS

anomalies. In Proceedings of the 4th International Conference on Detection of

Intrusions and Malware, and Vulnerability Assessment (DIMVA), 2007.

[59] Y. Zhang, J. I. Hong, and L. F. Cranor. Cantina: a content-based approach

to detecting phishing web sites. In Proceedings of the 16th international

conference on World Wide Web (WWW), 200