A

Major Project Report on

**REAL TIME DETECTION OF MALICIOUS MOBILE WEBPAGES**

*Submitted for partial fulfilment of the requirements for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**by**

|  |  |
| --- | --- |
| **ABHISHEK TRIPATHI** | **18K85A0561** |
| **ANGAD SINGH** | **18K85A0566** |
| **ATULYA CHAUHAN**  **NITHIN BHARADWAJ** | **18K81A0569**  **18K81A0589** |

Under the Guidance of

**Dr . K. SRINIVAS**

**ASSOCIATE PROFESSOR**

**DEPARTMENT OF CSE**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**JULY - 2022**

|  |
| --- |
| **St. MARTIN'S ENGINEERING COLLEGE**  An Autonomous Institute  NBA & NAAC A+ Accredited  Dhulapally, Secunderabad - 500 100 |
| **CERTIFICATE** |

|  |  |  |
| --- | --- | --- |
| Logo, company name  Description automatically generated | **St. MARTIN'S ENGINEERING COLLEGE** | A picture containing text, clipart, gear  Description automatically generated |
| UGC Autonomous  Affiliated to JNTUH, Approved by AICTE Accredited by NBA & NAAC A+, ISO 9001:2008 Certified  Dhulapally, Secunderabad - 500 100 |

This is to certify that the project entitled **“REAL TIME DETECTION OF MALICIOUS MOBILE WEBPAGES”** is being submitted by **Mr. Abhishek Tripathi 18K81A0561**, **Mr. Angad Singh 18K81A0566, Ms. Atulya Chauhan 18K81A0569** and **Mr. Nithin Bharadwaj 18K81A0589** in fulfilment of the requirement for the award of degree of **BACHELOR IN COMPUTER SCIENCE AND ENGINEERING** is recorded of bonafide work carried out by them. The result embodied in this report have been verified and found satisfactory.

**Guide Head of the Department**

**Dr. K. Srinivas Dr. M. Narayanan**

**Associate Professor Professor & Head**

**Department of CSE Department of CSE**

**Internal Examiner** **External Examiner**

**Place:**

**Date:**

|  |  |  |
| --- | --- | --- |
| Logo, company name  Description automatically generated | **St. MARTIN'S ENGINEERING COLLEGE** | A picture containing text, clipart, gear  Description automatically generated |
| UGC Autonomous  Affiliated to JNTUH, Approved by AICTE Accredited by NBA & NAAC A+, ISO 9001:2008 Certified  Dhulapally, Secunderabad - 500 100 |

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

We, the students of ‘**Bachelor of Technology in Department of Computer Science and Engineering’**, session: 2018 - 2022**, St. Martin’s Engineering College, Dhulapally, Kompally, Secunderabad,** hereby declare that the work presented in this Major Project Work entitled **REAL TIME DETECTION OF MALICIOUS MOBILE WEBPAGES** is the outcome of our own bonafide work and is correct to the best of our knowledge and this work has been undertaken taking care of Engineering Ethics. This result embodied in this project report has not been submitted in any university for award of any degree.

|  |  |
| --- | --- |
| **Mr.Abhishek Tripathi** | **18K85A0561** |
| **Mr.Angad Singh** | **18K85A0566** |
| **Ms.Atulya Chauhan**  **Mr.Nithin Bharadwaj** | **18K81A0569**  **18K81A0589** |

**ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose encouragement and guidance have crowded our efforts with success.

We extend our deep sense of gratitude to Principal, **Dr. P. SANTOSH KUMAR** **PATRA**, St. Martin’s Engineering College Dhulapally, for permitting us to undertake this project.

We are also thankful to **Dr. M.NARAYANAN**, Head of the Department, Department of Computer Science and Engineering, St. Martin’s Engineering College, Dhulapally, Secunderabad for his support and guidance throughout our project.

We would like to thank our Project Coordinators **Dr. B.RAJALINGAM**, Associate Professor and **Dr. K.SRINIVAS**, Associate Professor, Department of Computer Science and Engineering for their valuable support.

We would like to express our sincere  gratitude  and  indebtedness to  our  project  supervisor, **Dr. K . SRINIVAS** , Associate Professor, Department of Computer Science and Engineering,, St. Martins Engineering College, Dhulapally, for his support and guidance throughout our project.

Finally, we express thanks to all those who have helped us successfully completing this project. Furthermore, we would like to thank our family and friends for their moral support and encouragement. We express thanks to all those who have helped us in successfully completing the project.

|  |  |
| --- | --- |
| Mr. Abhishek Tripathi | 18K85A0561 |
| Mr. Angad Singh | 18K85A0566 |
| Ms. Atulya Chauhan  Mr. Nithin Bharadwaj | 18K81A0569  18K81A0589 |

**ABSTRACT**

Mobile specific webpages differ significantly from their desktop counterparts in content, layout and functionality. Accordingly,existing techniques to detect malicious websites are unlikely to work for such webpages.

In this project, we design and implement kAYO, a mechanism that distinguishes between malicious and benign mobile webpages. kAYO makes this determination based on staticfeatures 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 mobilemalicious webpages. We then apply kAYO to a dataset of over 350,000 known benign and malicious mobile webpages and demonstrate90% accuracy in classification. Moreover, we discover, characterize and report a number of webpages missed by Google Safe Browsing and VirusTotal, 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**

|  |  |
| --- | --- |
| **TITLE** | **PAGE NO.** |
| **ACKNOWLEDGEMENT**  **ABSTRACT**  **LIST OF FIGURES**  **LIST OF TABLES**  **LIST OF ACRONYMS AND DEFINITION** | **i**  **ii**  **vi**  **vii**  **vii** |
| **CHAPTER 1: INTRODUCTION** | **01** |
| **CHAPTER 2: LITERATURE SURVEY**  **CHAPTER 3: SYSTEM ANALYSIS AND DESIGN**  3.1 Existing System  3.2 Proposed System  **CHAPTER 4: SYSTEM REQUIREMENTS & SPECIFICATION**  4.1 DESIGN  4.1.1 Architecture  4.1.2 Use-case Diagram  4.1.3 Class Diagram  4.1.4 Dataflow Diagram  4.1.5 Sequence Diagram  4.1.6 Component Diagram  4.1.7 Deployment Diagram  4.1.8 Activity Diagram  4.1.9 Collaboration Diagram  4.2 MODULES  4.7.1 Upload Dataset  4.7.2 Process Pages For Kayo Features  4.7.3 JavaScript Feature Set  4.7.4 HTML Feature Set  4.7.5 URL Feature Set  4.7.6 Mobile Feature Set  4.7.7 View Detection Classification  4.7.8 Detection  4.3 SYSTEM REQUIREMENTS  4.4 PROJECT PLANNING AND PROCESS MODEL  4.4.1 SDLC  4.4.2 Requirements Gathering Stage  4.4.3 Analysis Stage  4.4.4 Designing Stage  4.4.5 Development (Coding) Stage  4.4.6 Integration & Test Stage  4.4.7 Installation & Acceptance Test  4.4.8 Maintenance  4.4.9 Feasibility Study  4.5 IMPLEMENTATION  4.5.1 Introduction of Technologies Used  4.5.1.1 About Java  4.5.1.2 Importance of Java to Internet  4.5.1.3 Applications and Applets  4.5.1.4 Java Architecture  4.5.1.5 Compilation of Code  4.5.2 AWT and Swings  4.5.2.1 Graphical User Interface  4.5.2.2 Components  4.5.2.3 Containers  4.5.3 Creating a Frame  4.5.3.1 Method 1  4.5.3.2 Method 2  4.5.4 Types of Components  4.5.4.1 Labels  4.5.4.2 Buttons  4.5.4.3 Checkbox  4.5.4.4 Radio Button  4.5.4.5 Choice  4.5.4.6 List  4.5.4.7 Text Field  4.5.4.8 Text Area  4.5.5 Layout Manager  4.5.5.1 Flow Layout  4.5.5.2 Border Layout  4.5.5.3 Grid Layout  4.5.6 Swings  4.5.6.1 About Swings  4.5.6.2 Difference between Swings and AWT  4.5.6.3 Java Swing Class Hierarchy  4.5.6.4 Swing Components  4.5.6.5 JtabbedPane Class  4.5.6.6 Example Program  4.5.6.7 JMenuBar,JMenu,JMenuItem  4.6 TESTING  4.6.1 Implementation  4.6.2 Testing  4.6.3 System Testing  4.6.4 Module Testing  4.6.5 Integration Testing  4.6.6 Acceptance Testing  4.6.7 Test Cases  **CHAPTER 5 :** **SOURCE CODE**  **CHAPTER 6 :** **EXPERIMENTAL RESULTS**  **CHAPTER 7: CONCLUSION AND FUTURE ENHANCEMENT**  7.1 CONCLUSION  7.2 FUTURE SCOPE  **REFERENCES** | **03**  **09**  09  09  **11**  11  11  12  12  13  14  15  16  17  18  20  20  20  20  20  20  20  20  20  21  22  22  22  24  24  25  26  27  28  28  30  30  30  30  30  31  31  32  32  33  34  35  35  36  36  36  35  37  38  39  40  40  41  42  42  43  44  44  44  45  46  46  47  47  49  50  50  50  50  51  51  51  52  **54**  **77**  **87**  87  87  **87** |

**LIST OF FIGURES**

|  |  |  |  |
| --- | --- | --- | --- |
| **Figure No.** | **Figure Title** | **Page No.** | |
| 4.1 | Architecture Diagram | 11 | |
| 4.2  4.3 | Use Case Diagram  Class Diagram | 12  13 | |
| 4.4 | Data Flow Diagram | 14 | |
| 4.5 | Sequence Diagram | 15 | |
| 4.6 | Component Diagram | 16 | |
| 4.7 | Deployment Diagram | 17 | |
| 4.8 | Activity Diagram | 18 | |
| 4.9 | Collaboration Diagram | 19 | |
| 4.10 | Software Development Life Cycle Model | 22 | |
| 4.11 | Requirement Gathering Stage | 23 | |
| 4.12 | Analysis Stage | 24 | |
| 4.13 | Designing Stage | 25 | |
| 4.14 | Development Stage | 25 | |
| 4.15 | Integration and Test Stage | 26 | |
| 4.16 | Installation and Acceptance Stage | 27 | |
| 4.17 | Compiling and interpreting java source code | | 31 |
| 4.18 | Types of Components | 33 | |
| 4.19 | Types of Containers | 34 | |
| 4.20 | Java Swing Class Hierarchy | 46 | |
| 6.1 | Run.bat File | 77 | |
| 6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9  6.10 | Upload Dataset  Dataset Loaded  Process pages for Kayo features  JavaScript Features Set  HTML Features Set  URL Features Set  Mobile Features Set  Detection Classification  Detection Chart | 78  79  80  81  82  83  84  85  86 | |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Table Name** | **Page No.** |
| 4.1 | Difference between Swings and AWT | 45 |
| 4.2 | Swing Components | 46 |
| 5.1 | Test Cases | 52 |

**LIST OF ACRONYMS AND DEFINITIONS**

|  |  |  |
| --- | --- | --- |
| **S No.** | **ACRONYM** | **DEFINITION** |
| 01. | API | Application Programming Interface |
| 02. | CSRF | Cross Site Request Forgery |
| 03. | DNS | Domain Name System |
| 04. | HTML | Hyper Text Markup Language |
| 05. | MAST | Mobile Application Security Triage |
| 06. | MCA | Multiple Correspondence Analysis |
| 07. | SSL | Secure Socket Layer |
| 08. | TLS | Transport Layer Security |
| 09. | URL | Uniform Resource Locator |

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 webpages.

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 redirections 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 project, we present kAYO1, 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 then collect over 350,000 mobile benign and malicious webpages over a period of three months. We then use a binomial classification technique to develop a model for kAYO to provide 90% ac- curacy 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 techniques (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 demonstrate 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 malicious Webpages. kAYO provides 90% accuracy in classification and shows improvement of two or- ders of magnitude in the speed of feature extraction over similar existing techniques. We further empirically demonstrate the significance of kAYO’s features. 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 extension 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, because 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. **LITERATURE SURVEY**

Porting browsers to mobile platforms may lead to new vulnerabilities whose solutions require careful balancing between usability and security and might not always be equivalent to those in desktop browsers. In this project, we perform the first large-scale security comparison between mobile and desktop browsers. We focus our efforts on display security given the inherent screen limitations of mobile phones. We evaluate display elements in ten mobile, three tablet and five desktop browsers. We identify two new classes of vulnerabilities specific to mobile browsers and demonstrate their risk by launching real-world attacks including display ballooning, login CSRF and clickjacking. Additionally, we implement a new phishing attack that exploits a default policy in mobile browsers. These previously unknown vulnerabilities have been confirmed by browser vendors. Our observations, inputs from browser vendors and the pervasive nature of the discovered vulnerabilities illustrate that new implementation errors leading to serious attacks are introduced when browser software is ported from the desktop to mobile environment. We conclude that usability considerations are crucial while designing mobile solutions and display security in mobile browsers is not comparable to that in desktop browsers [1].

Mobile browsers are increasingly being relied upon to perform security sensitive operations. Like their desktop counterparts, these applications can enable SSL/TLS to provide strong security guarantees for communications over the web. However, the drastic reduction in screen size and the accompanying reorganization of screen real estate significantly changes the use and consistency of the security indicators and certificate information that alert users of site identity and the presence of strong cryptographic algorithms. In this project, we perform the first measurement of the state of critical security indicators in mobile browsers. We evaluate ten mobile and two tablet browsers, representing over 90% of the market share, using the recommended guidelines for web user interface to convey security set forth by the World Wide Web Consortium (W3C). While desktop browsers follow the majority of guidelines, our analysis shows that mobile browsers fall significantly short. We also observe notable inconsistencies across mobile browsers when such mechanisms actually are implemented. Finally, we use this evidence to argue that the combination of reduced screen space and an independent selection of security indicators not only make it difficult for experts to determine the security standing of mobile browsers, but actually make mobile browsing more dangerous for average users as they provide a false sense of security [2].

The Domain Name System (DNS) is an essential protocol used by both legitimate Internet applications and cyber attacks. For example, botnets rely on DNS to support agile command and control infrastructures. An effective way to disrupt these attacks is to place malicious domains on a "blocklist" (or "blacklist") or to add a filtering rule in a firewall or network intrusion detection system. To evade such security countermeasures, attackers have used DNS agility, e.g., by using new domains daily to evade static blacklists and firewalls. In this project we propose Notos, a dynamic reputation system for DNS. The premise of this system is that malicious, agile use of DNS has unique characteristics and can be distinguished from legitimate, professionally provisioned DNS services. Notos uses passive DNS query data and analyzes the network and zone features of domains. It builds models of known legitimate domains and malicious domains, and uses these models to compute a reputation score for a new domain indicative of whether the domain is malicious or legitimate. We have evaluated Notos in a large ISP's network with DNS traffic from 1.4 million users. Our results show that Notos can identify malicious domains with high accuracy (true positive rate of 96.8%) and low false positive rate (0.38%), and can identify these domains weeks or even months before they appear in public blacklists [3].

Malware is a pressing concern for mobile application market operators. While current mitigation techniques are keeping pace with the relatively infrequent presence of malicious code, the rapidly increasing rate of application development makes manual and resource-intensive automated analysis costly at market-scale. To address this resource imbalance, we present the Mobile Application Security Triage (MAST) architecture, a tool that helps to direct scarce malware analysis resources towards the applications with the greatest potential to exhibit malicious behavior. MAST analyzes attributes extracted from just the application package using Multiple Correspondence Analysis (MCA), a statistical method that measures the correlation between multiple categorical (i.e., qualitative) data. We train MAST using over 15,000 applications from Google Play and a dataset of 732 known-malicious applications. We then use MAST to perform triage on three third-party markets of different size and malware composition---36,710 applications in total. Our experiments show that MAST is both effective and performant. Using MAST ordered ranking, malware-analysis tools can find 95% of malware at the cost of analyzing 13% of the non-malicious applications on average across multiple markets, and MAST triage processes markets in less than a quarter of the time required to perform signature detection. More importantly, we show that successful triage can dramatically reduce the costs of removing malicious applications from markets [4].

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 project, 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 [5].

Phishing websites, fraudulent sites that impersonate a trusted third party to gain access to private data, continue to cost Internet users over a billion dollars each year. In this project, we describe the design and performance characteristics of a scalable machine learning classifier we developed to detect phishing websites. We use this classifier to maintain Google’s phishing blacklist automatically. Our classifier analyzes millions of pages a day, examining the URL and the contents of a page to determine whether or not a page is phishing. Unlike previous work in this field, we train the classifier on a noisy dataset consisting of millions of samples from previously collected live classification data [6].

Porting browsers to mobile platforms may lead to new vulnerabilities whose solutions require careful balancing between usability and security and might not always be equivalent to those in desktop browsers. In this project, we perform the first large-scale security comparison between mobile and desktop browsers. We focus our efforts on display security given the inherent screen limitations of mobile phones. We evaluate display elements in ten mobile, three tablet and five desktop browsers [7].

Mobile browsers are increasingly being relied upon to perform security sensitive operations. Like their desktop counterparts, these applications can enable SSL/TLS to provide strong security guarantees for communications over the web. However, the drastic reduction in screen size and the accompanying reorganization of screen real estate significantly changes the use and consistency of the security indicators and certificate information that alert users of site identity and the presence of strong cryptographic algorithms [8].

The Domain Name System (DNS) is an essential protocol used by both legitimate Internet applications and cyber attacks. For example, botnets rely on DNS to support agile command and control infrastructures. An effective way to disrupt these attacks is to place malicious domains on a "blocklist" (or "blacklist") or to add a filtering rule in a firewall or network intrusion detection system. To evade such security countermeasures, attackers have used DNS agility, e.g., by using new domains daily to evade static blacklists and firewalls [9].

A novel technique is presented to identify the codec of a coded audio. The technique does not perform decoding, utilize any coding metadata, or assume information about the structure describing the bit stream format of a codec. The underlying idea of the technique is that the design choices governing the compression level, audio quality and complexity of a codec will reveal themselves on the coded audio. To exploit this, the technique samples 2–4 kilobytes of data from a coded audio and analyzes the randomness and chaotic nature of the sampled data to build statistical models that represent encoding process associated with different codecs [10].

The domain name service (DNS) plays an important role in the operation of the Internet, providing a two-way mapping between domain names and their numerical identifiers. Given its fundamental role, it is not surprising that a wide variety of malicious activities involve the domain name service in one way or another. For example, bots resolve DNS names to locate their command and control servers, and spam mails contain URLs that link to domains that resolve to scam servers. Thus, it seems beneficial to monitor the use of the DNS system for signs that indicate that a certain name is used as part of a malicious operation [11].

As soon as a phishing website is broadcast through fraudulent email messages the first systems to visit it are typically mobile devices.This makes sense since mobile users are “always on” and are most likely to read email messages as soon as they arrive. Meanwhile, desktop users only read messages when they have access to their computer. Also most fraudulent emails call for immediate action. For example, they usually claim that suspicious activity has been detected in the user’s account and that immediate action is required. Most victims who fall for this ploy will visit the phishing site quickly [12].

Web access on mobile platforms already constitutes a significant (> 20%) share of web traffic. Furthermore, this share is projected to even surpass access from laptops and desktops . In conjunction with this growth, user expectations for the performance of mobile applications and websites is also growing rapidly . Surveys show that 71% of users expect websites to load almost as quickly as their desktops and 33% of annoyed users are likely to go to a competitor’s site leading to loss of ad- and click-based revenue streams [13].

Malicious web pages that host drive-by-download exploits have become a popular means for compromising hosts on the Internet and, subsequently, for creating large-scale botnets. In a drive-bydownload exploit, an attacker embeds a malicious script (typically written in JavaScript) into a web page. When a victim visits this page, the script is executed and attempts to compromise the browser or one of its plugins. To detect drive-by-download exploits, researchers have developed a number of systems that analyze web pages for the presence of malicious code. Most of these systems use dynamic analysis. That is, they run the scripts associated with a web page either directly in a real browser (running in a virtualized environment) or in an emulated browser, and they monitor the scripts’ executions for malicious activity [14].

Malware is a pressing concern for mobile application market operators. While current mitigation techniques are keeping pace with the relatively infrequent presence of malicious code, the rapidly increasing rate of application development makes manual and resource-intensive automated analysis costly at market-scale. To address this resource imbalance, we present the Mobile Application Security Triage (MAST) architecture, a tool that helps to direct scarce malware analysis resources towards the applications with the greatest potential to exhibit malicious behavior [15].

The fluidity of application markets complicate smartphone security. Although recent efforts have shed light on particular security issues, there remains little insight into broader security characteristics of smartphone applications. This project seeks to better understand smartphone application security by studying 1,100 popular free Android applications. We introduce the ded decompiler, which recovers Android application source code directly from its installation image. We design and execute a horizontal study of smartphone applications based on static analysis of 21 million lines of recovered code [16].

Modern browsers and smartphone operating systems treat applications as mutually untrusting, potentially malicious principals. Applications are (1) isolated except for explicit IPC or inter-application communication channels and (2) unprivileged by default, requiring user permission for additional privileges. Although inter-application communication supports useful collaboration, it also introduces the risk of permission redelegation. Permission re-delegation occurs when an application with permissions performs a privileged task for an application without permissions. This undermines the requirement that the user approve each application’s access to privileged devices and data. We discuss permission re-delegation and demonstrate its risk by launching real-world attacks on Android system applications; several of the vulnerabilities have been confirmed as bugs [17].

Identity theft through phishing attacks has become a ma- jor concern for Internet users. Typically, phishing attacks aim at luring the user to a faked web site to disclose per- sonal information. Existing solutions proposed against this kind of attack can, however, hardly counter the new genera- tion of sophisticated malware phishing attacks, e.g., pharm- ing Trojans, designed to target certain services. This project aims at making the first steps towards the design and imple- mentation of a security architecture that prevents both clas- sical and malware phishing attacks. Our approach is based on the ideas of compartmentalization for isolating applica- tions of different trust level, and a trusted wallet for stor- ing credentials and authenticating sensitive services [18].

Phishing is form of identity theft that combines social engineering techniques and sophisticated attack vectors to harvest financial information from unsuspecting consumers. Often a phisher tries to lure her victim into clicking a URL pointing to a rogue page. In this project, we focus on studying the structure of URLs employed in various phishing attacks. We find that it is often possible to tell whether or not a URL belongs to a phishing attack without requiring any knowledge of the corresponding page data. We describe several features that can be used to distinguish a phishing URL from a benign one [19].

Client-side attacks are on the rise: malicious websites that exploit vulner-

abilities in the visitor’s browser are posing a serious threat to client security, compro-

mising 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 in-

troduce 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, Internet-

scale analysis tool to build a database of threats found in the wild.

Client-side attacks are on the rise: malicious websites that exploit vulner-

abilities in the visitor’s browser are posing a serious threat to client security, compro-

mising 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 in-

troduce 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, Internet-

scale analysis tool to build a database of threats found in the wild.

Client-side attacks are on the rise: malicious websites that exploit vulner-

abilities in the visitor’s browser are posing a serious threat to client security, compro-

mising 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 in-

troduce 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, Internet-

scale analysis tool to build a database of threats found in the wild.

Client-side attacks are on the rise: malicious websites that exploit vulner-

abilities in the visitor’s browser are posing a serious threat to client security, compro-

mising 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 in-

troduce 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, Internet-

scale analysis tool to build a database of threats found in the wild.

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 [Iki]. Utilizing it as a client honeypot, we portray the challenge in such an approach and evaluate our system as a high-speed, Internet-scale analysis tool to build a database of threats found in the wild [20].

**3.SYSTEM ANALYSIS AND DESIGN**

**3.1 Existing System**

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 webpages. Content, functionality and layout have regularly been used to perform static analysis to determine maliciousness in the desktop space.

Features such as the frequency of iframes and the number of redirections 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.

Disadvantages of Existing System

1. Previous techniques also fail to consider mobile specific webpage elements such as calls to mobile APIs.

**3.2 Proposed System**

**Proposed System**

In this project, we present kAYO, a fast and reliable static analysis technique to detect malicious mobile webpages. 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 then collect over 350,000 mobile benign and malicious webpages over a period of three months. We then use a binomial classification technique to develop a model for kAYO to provide 90% accuracy and 89% true positive rate.

Advantages of Proposed System

1. kAYO’s performance matches or exceeds that of existing static techniques used in the desktop space.
2. kAYO also detects a number of malicious mobile webpages not precisely detected by existing techniques such as VirusTotal and Google Safe Browsing.

**4. SYSTEM REQUIREMENTS AND SPECIFICATIONS**

**4.1 DESIGN**

**4.1.1 Architecture**

Diagram

Description automatically generated

Fig.4.1 : Architecture Diagram

* In the figure 4.1 we can see a browser extension based on kAYO adds value for two reasons. First, the mobile specific design of kAYO enables detection of new threats previously unseen by existing services (e.g., pages including spam phone numbers).
* Building an extension allows immediate use of our technique
* Architecture of the mobile browser extension based on kAYO. User enters the URL he wants to visit in the extension toolbar and receives a response in real-time from our backend server about the maliciousness of the URL.
* If the URL is benign according to kAYO, the page of interest is rendered in the browser. Otherwise, the user is shown a warning message to not visit the URL.
* Users of the extension will browse both mobile specific and desktop webpages since not all websites offer a mobile specific version. Recall that being a mobile specific technique, kAYO does not perform well on desktop webpages.
* Consequently, processing all pages of interest through kAYO might output incorrect results for desktop webpages.
* We performed manual analysis of 100 randomly selected URLs (90 benign and 10 malicious) from our test dataset and measured the performance of kAYO in realtime. On an average, an output was rendered in 829 ms on average from the time the user entered a URL in kAYO’s toolbar

**4.1.2 Use-case Diagrams**

A **use case diagram** at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

This project includes the following Actors and Use Cases as illustrated in Fig. 4.2.

• User: A User can perform various functions like upload database, exit etc.

• The User using various functions the user can extract the results from the image.

Java script Features set

HTML feature set

URL feature set

User

Process pages for Kayo Features

Mobile feature set

View detection classification

Upload Mobile WebPages

Detection chart

Fig.4.2: Use Case Diagram

**4.1.3 Class Diagrams**

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

* The upper part holds the name of the class
* The middle part contains the attributes of the class
* The bottom part gives the methods or operations the class can take or undertake
* This project includes the following Class diagram illustrated in Fig. 4.3:
* The User is the class here and below are various methods or the functionalities that the user can use to get the annotations .



Fig.4.3: Class Diagram

**4.1.4 Dataflow Diagrams**

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest.

This project includes the following Class diagram illustrated in Fig. 4.4.

* Here the user uploads the image i.e. the Step 1 in the Data Flow diagram, then the consecutive steps will take place and from this we can get an idea of how each step is being carried out according to the need of the user.

Data Owner

* 2. Dataset loaded
* 4. processing features prepared
* 1. Upload Mobile WebPages 6. JavaScript tags displayed
* 3. Process pages for kayo features 8. Number of redirection pages displayed
* 5. Java Script Feature set 10. Number of sub domains displayed
* 7. HTML Feature set 12. Number of mobile API
* 9. URL Features set 14. Predict webpages are malicious or not
* 11. Mobile Feature set 16. Malicious & benign page count displayed
* 13. View Detection Classification
* 15. Detection Chart

Fig.4.4: Data flow Diagram

**4.1.5 Sequence Diagrams**

A **sequence diagram** 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. A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called **event diagrams**, **event scenarios**, and timing diagrams.

This project includes the following Class diagram illustrated in Fig. 4.5

* Here for every action by the user there is a sequence in which the next step should be performed.
* First the user uploads an image and next the user Runs the Existing Technique and gets the
* Annotation , after this the user Runs Extension Technique to get more number of Annotations.
* After that the user Views the Comparison Graph and gets a detailed analysis between the existing and the extension Technique.
* After that the User Exits the Application.

****

Fig. 4.5: Sequence Diagram

**4.1.6 Component Diagram**

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems

Components in the figure 4.6 are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

Data Owner

Upload mobile

WebPages

Process pages for

Kayo features

JavaScript

Feature set

URL feature set

Mobile feature

Set

View detection

Classification

Detection chart

Fig.4.6: Component Diagram

**4.1.7 Deployment Diagrams**

A **deployment diagram** in the figure 4.7 Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

System

Process Kayo

And JavaScript

Features

HTML and URL

And Mobile

Feature set

Detect

Classification

And chart

Fig.4.7: Deployment Diagram

**4.1.8 Activity Diagrams**

Activity diagram in the figure 4.8 is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.

So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

Upload Mobile WebPages

Process pages for kayo features

Java Script Feature set

HTML Feature set

UML Features set

Mobile Feature set

View Detection Classification

Detection Chart

Fig.4.8: Activity Diagram

**4.1.9 Collaboration Diagram**

A collaboration diagram in the figure 4.9 describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

****

Fig.4.9: Collaboration Diagram

**4.2 MODULES**

1. Upload Dataset
2. Process Pages for Kayo Features
3. Javascript Feature Set
4. HTML Feature Set
5. URL Feature Set
6. Mobile Feature Set
7. View Detection Classification
8. Detection

**4.2.1 Upload Dataset**

Upload Dataset which contains various benign and malicious webpages data.

**4.2.2 Process Pages for Kayo Features**

Click on Process Pages for Kayo Features to get the classifies information from the dataset.

**4.2.3 Javascript Feature Set**

Click on Javascript feature Set to get the classifies information from the dataset.

**4.2.4 HTML Feature Set**

Click on HTML feature Set to get the classifies information from the dataset.

**4.2.5 URL Feature Set**

Click on URL feature Set to get the classifies information from the dataset.

**4.2.6 Mobile Feature Set**

Click on Mobile feature Set to get the classifies information from the dataset.

**4.2.7 View Detection Classification**

Click on View Detection Classification to get the classifies information from the dataset.

**4.2.8 Detection**

Click on Detection to get the classifies information from the dataset.

**4.3 REQUIRMENTS**

**4.3.1 Minimum Hardware Requirements**

• Processor : i5

• RAM : 8GB

• Space on Hard Disk : 1 TB

**4.3.2 Minimum Software Requirements**

• Programming language : Java 8.0

• Operating Systems. : Windows 10

**4.4 PROJECT PLANNING AND PROCESS MODEL**

**4.4.1 SDLC (Umbrella Model)**

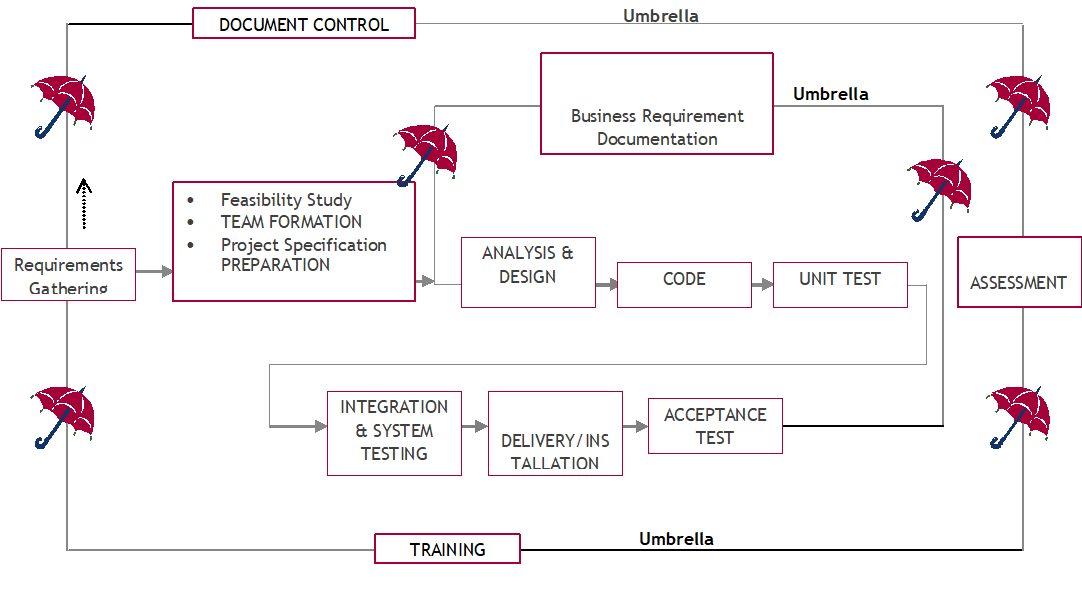
****

Fig.4.10: Software Development Life Cycle Model

SDLC is nothing but Software Development Life Cycle. It is a standard which is used by software industry to develop good software. The above Fig 4.10 depicts the SDLC.

Following are the stages in SDLC

* Requirement Gathering
* Analysis
* Designing
* Coding
* Testing
* Maintenance

**4.4.2 Requirements Gathering** **stage**

The requirements gathering process takes as its input the goals identified in the high-level requirements of the project plan. Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description. The following Fig 4.11 depicts the requirement gathering stage.



Fig.4.11: Requirement Gathering Stage

These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan. The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title. In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

* Feasibility study is all about identification of problems in a project.
* No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
* Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator.

**4.4.3 Analysis Stage**

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches. The following Fig 4.12 depicts the requirement gathering stage.



Fig.4.12: Analysis Stage

The most critical stage of the project plan is a listing of high-level product requirements, also referred to as goals. All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals. The minimum information for each goal consists of a title and textual description, although additional information and references to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan, and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

**4.4.4 Designing Stage**

The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input. The following Fig 4.13 depicts the requirement gathering stage.

  
Fig.4.13: Designing Stage

When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement. The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

**4.4.5 Development (Coding) Stage**

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, and data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software. The following Fig 4.14 depicts the requirement gathering stage.



Fig.4.14: Development Stage

The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software, an implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

**4.4.6 Integration & Test Stage**

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles. The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan. The following Fig 4.15 depicts the requirement gathering stage.

  
 Fig.4.15: Integration & Test Stage

The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

**4.4.7 Installation & Acceptance Test**

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.

After the Final delivery of the software the customer uses the software and runs the various functionalities that the customer is in need of and checks that everything is as per the required form as required. The following Fig 4.16 depicts the requirement gathering stage.



Fig.4.16: Installation & Acceptance Test

The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labor data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

**4.4.8 Maintenance**

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

**4.4.9 FEASIBILITY STUDY**

Feasibility studies aim to objectively and rationally uncover the strengths and weaknesses of the existing system or proposed venture. In its simplest term, the two criteria to judge feasibility are cost required and value to be attained. As such, a well-designed feasibility study should provide historical background of the project. Generally, feasibility studies precede technical development and project implementation. The assessment of feasibility study is based on the following factors:

1. Economic Feasibility
2. Operational Feasibility
3. Technical Feasibility

**1.** Economic Feasibility

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economical feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, There is nominal expenditure and economical feasibility for certain.

2. Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization’s operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

3. Technical Feasibility

Earlier no system existed to cater to the needs of ‘Secure Infrastructure Implementation System’. The current system developed is technically feasible. It is a web based user interface for audit workflow at NIC-CSD. Thus it provides an easy access to .the users. The database’s purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security.

**4.5 IMPLEMENTATION**

**4.5.1 Introduction of technologies used**

**4.5.1.1 About Java**

Initially the language was called as “oak” but it was renamed as “java” in 1995.The primary motivation of this language was the need for a platform-independent (i.e. architecture neutral) language that could be used to create software to be embedded in various consumer electronic devices.

• Java is a programmer’s language

• Java is cohesive and consistent

• Except for those constraint imposed by the Internet environment. Java gives. the programmer, full control

Finally Java is to Internet Programming where c was to System Programming.

**4.5.1.2 Importance of Java to the Internet**

Java has had a profound effect on the Internet. This is because; java expands the Universe of objects that can move about freely in Cyberspace. In a network, two categories of objects are transmitted between the server and the personal computer. They are passive information and Dynamic active programs. in the areas of Security and probability. But Java addresses these concerns and by doing so, has opened the door to an exciting new form of program called the Applet.

**4.5.1.3 Applications and applets**

An application is a program that runs on our Computer under the operating system of that computer. It is more or less like one creating using C or C++ .Java’s ability to create Applets makes it important. An Applet I saw application, designed to be transmitted over the Internet and executed by a Java-compatible web browser. An applet I actually a tiny Java program, dynamically downloaded across the network, just like an image. But the difference is, it is an intelligent program, not just a media file. It can be react to the user input and dynamically change.

**4.5.1.4 Java Architecture**

Java architecture provides a portable, robust, high performing environment for development. Java provides portability by compiling the byte codes for the Java Virtual Machine, which is then interpreted on each platform by the run-time environment. Java is a dynamic system, able to load code when needed from a machine in the same room or across the planet.

**4.5.1.5 Compilation of code**

When you compile the code, the Java compiler creates machine code (called byte code)for a hypothetical machine called Java Virtual Machine(JVM). The JVM is supposed t executed the byte code. The JVM is created for the overcoming the issue of probability. The code is written and compiled for one machine and interpreted on all machines .This machine is called Java Virtual Machine.

Compiling and interpreting java source code is shown in figure 4.17.

****

Fig 4.17: Compiling and interpreting java source code

During run-time the Java interpreter tricks the byte code file into thinking that it is running on a Java Virtual Machine. In reality this could be an Intel Pentium windows 95 or sun SPARCstation running Solaris or Apple Macintosh running system and all could receive code from any computer through internet and run the Applets.

Simple

Java was designed to be easy for the Professional programmer to learn and to use effectively. If you are an experienced C++ Programmer, learning Java will oriented features of C++. Most of the confusing concepts from C++ are either left out of Java or implemented in a cleaner, more approachable manner. In Java there are a small number of clearly defined ways to accomplish a given task.

### Object oriented

Java was not designed to be source-code compatible with any other language. This allowed the Java team the freedom to design with a blank state. One outcome of this was a clean usable, pragmatic approach to objects. The object model in Java is simple and easy to extend, while simple types, such as integers, are kept as high-performance non-objects.

### Robust

The multi-platform environment of the web places extraordinary demands on a program, because the program must execute reliably in a variety of systems. The ability to create robust programs was given a high priority in the design of Java? Java is strictly typed language; it checks your code at compile time and runtime.

Java virtually eliminates the problems of memory management and deal location, which is completely automatic. In a well-written Java program, all run-time errors can and should be managed by your program.

**4.5.2 AWT and Swings**

**4.5.2.1 Graphical User Interface**

The user interface is that part of a program that interacts with the user of the program. GUI is a type of [user interface](http://en.wikipedia.org/wiki/User_interface) that allows [users](http://en.wikipedia.org/wiki/User_(computing)) to [interact](http://en.wikipedia.org/wiki/Human-computer_interaction) with electronic devices with images rather than text commands. A class library is provided by the Java programming language which is known as Abstract Window Toolkit (AWT) for writing graphical programs. The Abstract Window Toolkit (AWT) contains several graphical widgets which can be added and positioned to the display area with a layout manager.

As the Java programming language, the AWT is not platform-independent. AWT uses system peers object for constructing graphical widgets. A common set of tools is provided by the AWT for graphical user interface design. The implementation of the user interface elements provided by the AWT is done using every platform's native GUI toolkit. One of the AWT's significance is that the look and feel of each platform can be preserved.

**4.5.2.2 Components**

A graphical user interface is built of graphical elements called components. A *component* is an object having a graphical representation that can be displayed on the screen and that can interact with the user. Components allow the user to interact with the program and provide the input to the program. In the AWT, all user interface components are instances of class Component or one of its subtypes. Typical components include such items as buttons, scrollbars, and text fields.

Types of Components is shown in figure 4.18:

**Diagram

Description automatically generated**

Fig 4.18: Types of Components

Before proceeding ahead, first we need to know what containers are. After learning containers we learn all components in detail.

**4.5.2.3 Containers**

Components do not stand alone, but rather are found within containers. In order to make components visible, we need to add all components to the container. Containers contain and control the layout of components. In the AWT, all containers are instances of class Container or one of its subtypes. Components must fit completely within the container that contains them. For adding components to the container we will use add() method.

Types of containers is shown in figure 4.19:

**Diagram

Description automatically generated**

Fig 4.19: Types of Containers

Basic GUI Logic:

The GUI application or applet is created in three steps. These are:

* Add components to Container objects to make your GUI.
* Then you need to setup event handlers for the user interaction with GUI.
* Explicitly display the GUI for application.

A new thread is started by the interpreter for user interaction when an AWT GUI is displayed. When any event is received by this new thread such as click of a mouse, pressing of key etc then one of the event handlers is called by the new thread set up for GUI. One important point to note here is that the event handler code is executed within the thread.

**4.5.3 Creating a Frame**

**4.5.3.1 Method1**

In the first method we will be creating frame by extending Frame class which is defined in java.awt package. Following program demonstrate the creation of a frame.

import java.awt.\*;

public class FrameDemo1 extends Frame

{

FrameDemo1()

{

setTitle("Label Frame");

setVisible(true);

setSize(500,500);

}

public static void main(String[] args)

{

new FrameDemo1 ();

}

}

In the above program we are using three methods:

setTitle: For setting the title of the frame we will use this method. It takes String as an argument which will be the title name.

SetVisible: For making our frame visible we will use this method. This method takes Boolean value as an argument. If we are passing true then window will be visible otherwise window will not be visible.

SetSize: For setting the size of the window we will use this method. The first argument is width of the frame and second argument is height of the frame.

**4.5.3.2 Method 2**

In this method we will be creating the Frame class instance for creating frame window. Following program demonstrate Method2.

import java.awt.\*;

public class FrameDemo2

{

public static void main(String[] args)

{

Frame f = new Frame();

f.setTitle("My first frame");

f.setVisible(true);

f.setSize(500,500);

}

}

**4.5.4 Types of Components**

**4.5.4.1 Labels**

This is the simplest component of Java Abstract Window Toolkit. This component is generally used to show the text or string in your application and label never perform any type of action.

Label l1 = new Label("One");

Label l2 = new Label("Two");

Label l3 = new Label("Three",Label.CENTER);

In the above three lines we have created three labels with the name “one, two, three”. In the third label we are passing two arguments. Second argument is the justification of the label. Now after creating components we will be adding it to the container.

add(l1);

add(l2);

add(l3);

We can set or change the text in a label by using the **setText( )** method. You can obtain the current label by calling **getText( )**. These methods are shown here:

void setText(String *str*)

String getText( )

**4.5.4.2 Buttons**

This is the component of Java Abstract Window Toolkit and is used to trigger actions and other events required for your application. The syntax of defining button is as follows :

Button l1 = new Button("One");

Button l2 = new Button("Two");

Button l3 = new Button("Three");

We can change the Button's label or get the label's text by using the Button.setLabel(String) and Button.getLabel() method.

**4.5.4.3 CheckBox**

A check box is a control that is used to turn an option on or off. It consists of a small boxthat can either contain a check mark or not. There is a label associated with each check box that describes what option the box represents. You change the state of a check box by clicking on it. The syntax of the definition of Checkbox is as follows :

Checkbox Win98 = new Checkbox("Windows 98/XP", null, true);

Checkbox winNT = new Checkbox("Windows NT/2000");

Checkbox solaris = new Checkbox("Solaris");

Checkbox mac = new Checkbox("MacOS");

The first form creates a check box whose label is specified in first argument and whose group is specified in second argument*.* If this check box is not part of a group, then *cbGroup* must be **null**. (Check box groups are described in the next section.) The value truedetermines the initial state of the check box is checked. The second form creates a check box with only one parameter.

To retrieve the current state of a check box, call **getState( )**. To set its state, call **setState( )**. You can obtain the current label associated with a check box by calling **getLabel( )**. To set the label, call **setLabel( )**. These methods are as follows:

boolean getState( )

void setState(boolean *on*)

String getLabel( )

void setLabel(String *str*)

Here, if *on* is **true**, the box is checked. If it is **false**, the box is cleared. The string passed in *str* becomes the new label associated with the invoking check box.

**4.5.4.4 Radio Button**

This is the special case of the Checkbox component of Java AWT package. This is used as a group of checkboxes which group name is same. Only one Checkbox from a Checkbox Group can be selected at a time. Syntax for creating radio buttons is as follows:

CheckboxGroup cbg = new CheckboxGroup();

Checkbox Win98 = new Checkbox("Windows 98/XP", cbg , true);

Checkbox winNT = new Checkbox("Windows NT/2000",cbg, false);

Checkbox solaris = new Checkbox("Solaris",cbg, false);

Checkbox mac = new Checkbox("MacOS",cbg, false);

For Radio Button we will be using CheckBox class. The only difference in Checkboxes and radio button is in Check boxes we will specify null for checkboxgroup but whereas in radio button we will be specifiying the checkboxgroup object in the second parameter.

**4.5.4.5 Choice**

The Choice class is used to create a pop-up list of items from which the user may choose. Thus, a Choice control is a form of menu. Syntax for creating choice is as follows:

Choice os = new Choice();

/\* adding items to choice \*/

os.add("Windows 98/XP");

os.add("Windows NT/2000");

os.add("Solaris");

os.add("MacOS");

We will be creating choice with the help of Choice class. Pop up list will be creating with the creation of object, but it will not have any items. For adding items we will be using add() method defined in Choice class.

To determine which item is currently selected, you may call either **getSelectedItem( )** or **getSelectedIndex( )**. These methods are shown here:

String getSelectedItem( )

int getSelectedIndex( )

The **getSelectedItem( )** method returns a string containing the name of the item. **getSelectedIndex( )** returns the index of the item. The first item is at index 0. By default, the first item added to the list is selected.

**4.5.4.6 List**

List class is also same as choice but the only difference in list and choice is, in choice user can select only one item whereas in List user can select more than one item. Syntax for creating list is as follows:

List os = new List(4, true);

First argument in the List constructor specifies the number of items allowed in the list. Second argument specifies whether multiple selections are allowed or not.

/\* Adding items to the list \*/

os.add("Windows 98/XP");

os.add("Windows NT/2000");

os.add("Solaris");

os.add("MacOS");

In list we can retrieve the items which are selected by the users. In multiple selection user will be selecting multiple values for retrieving all the values we have a method called getSelectedValues() whose return type is string array. For retrieving single value again we can use the method defined in Choice i.e. getSelectedItem().

**4.5.4.7 Text Field**

Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys. TextField is a subclass of TextComponent. Syntax for creating list is as follows:

TextField tf1 = new TextField(25);

TextField tf2 = new TextField();

In the first text field we are specifying the size of the text field and the second text field is created with the default value. **TextField** (and its superclass **TextComponent**) provides several methods that allow you to utilize a text field. To obtain the string currently contained in the text field, call **getText( )**. To set the text, call **setText( )**. These methods are as follows:

String getText( )

void setText(String *str*)

We can control whether the contents of a text field may be modified by the user by calling **setEditable( )**. You can determine editability by calling **isEditable( )**. These methods are shown here:

boolean isEditable( )

void setEditable(boolean *canEdit*)

**isEditable( )** returns **true** if the text may be changed and **false** if not. In **setEditable( )**, if *canEdit* is **true**, the text may be changed. If it is **false**, the text cannot be altered.

There may be times when we will want the user to enter text that is not displayed, such as a password. We can disable the echoing of the characters as they are typed by calling **setEchoChar( )**.

**4.5.4.8 Text Area**

TextArea is a multiple line editor. Syntax for creating list is as follows:

TextArea area = new TextArea(20,30);

Above code will create one text area with 20 rows and 30 columns. **TextArea** is a subclass of **TextComponent**. Therefore, it supports the **getText( )**, **setText( )**, **getSelectedText( )**, **select( )**, **isEditable( )**, and **setEditable( )** methods described in the preceding section.

**TextArea** adds the following methods:

void append(String *str*)

void insert(String *str*, int *index*)

void replaceRange(String *str*, int *startIndex*, int *endIndex*)

The **append( )** method appends the string specified by *str* to the end of the current text. **insert( )** inserts the string passed in *str* at the specified index. To replace text, call **replaceRange( )**. It replaces the characters from *startIndex* to *endIndex*–1, with the replacement text passed in *str.*

**4.5.5 Layout Managers**

A layout manager automatically arranges controls within a window by using some type of algorithm. Each **Container** object has a layout manager associated with it. A layout manager is an instance of any class that implements the **LayoutManager** interface. The layout manager is set by the **setLayout( )** method. If no call to **setLayout( )** is made, then the default layout manager is used. Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it. The **setLayout( )** method has the following general form:

void setLayout(LayoutManager *layoutObj*)

Here, *layoutObj* is a reference to the desired layout manager. If you wish to disable the layout manager and position components manually, pass **null** for *layoutObj.* If we do this, you will need to determine the shape and position of each component manually, using the setBounds( ) method defined by Component.

Void setBounds(int x , int y , int width, int length)

In which first two arguments are the x and y axis. Third argument is width and fourth argument is height of the component.

Java has several predefined **LayoutManager** classes, several of which are described next. You can use the layout manager that best fits your application.

**4.5.5.1 FlowLayout**

**FlowLayout** is the default layout manager. This is the layout manager that the preceding examples have used. **FlowLayout** implements a simple layout style, which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component, above and below, as well as left and right. Here are the constructors for **FlowLayout**:

FlowLayout( )

FlowLayout(int *how*)

FlowLayout(int *how*, int *horz*, int *vert*)

The first form creates the default layout, which centers components and leaves five pixels of space between each component. The second form lets you specify how each line is aligned. Valid values for *how* are as follows:

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

These values specify left, center, and right alignment, respectively. The third form allows you to specify the horizontal and vertical space left between components in *horz* and *vert,* respectively.

**4.5.5.2 BorderLayout**

THE JAVA LIBRARYThe **BorderLayout** class implements a common layout style for top-level windows. It has four narrow, fixed-width components at the edges and one large area in the center. The four sides are referred to as north, south, east, and west. The middle area is called the center. Here are the constructors defined by **BorderLayout**:

BorderLayout( )

BorderLayout(int *horz*, int *vert*)

The first form creates a default border layout. The second allows you to specify the horizontal and vertical space left between components in *horz* and *vert,* respectively. **BorderLayout** defines the following constants that specify the regions:

BorderLayout.CENTER BorderLayout.SOUTH

BorderLayout.EAST BorderLayout.WEST

BorderLayout.NORTH

When adding components, you will use these constants with the following form of **add( )**, which is defined by **Container**:

void add(Component *compObj,* Object *region*);

Here, *compObj* is the component to be added, and *region* specifies where the component will be added.

**4.5.5.3 GridLayout**

**GridLayout** lays out components in a two-dimensional grid. When you instantiate a **GridLayout**, you define the number of rows and columns. The constructors supported by **GridLayout** are shown here:

GridLayout( )

GridLayout(int *numRows*, int *numColumns* )

GridLayout(int *numRows*, int *numColumns*, int *horz*, int *vert*)

The first form creates a single-column grid layout. The second form creates a grid layout with the specified number of rows and columns. The third form allows you to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively. Either *numRows* or *numColumns* can be zero. Specifying *numRows* as zero allows for unlimited-length columns. Specifying *numColumns* as zero allows for unlimited-length rows.

**4.5.6 Swings**

**4.5.6.1 About Swings**

Swing is important to develop Java programs with a graphical user interface (GUI). There are many components which are used for the building of GUI in Swing. The Swing Toolkit consists of many components for the building of GUI. These components are also helpful in providing interactivity to Java applications. Following are components which are included in Swing toolkit:

* list controls
* buttons
* labels
* tree controls
* table controls

All AWT flexible components can be handled by the Java Swing. Swing toolkit contains far more components than the simple component toolkit. It is unique to any other toolkit in the way that it supports integrated internationalization, a highly customizable text package, rich undo support etc. Not only this you can also create your own look and feel using Swing other than the ones that are supported by it. The customized look and feel can be created using Synth which is specially designed. Not to forget that Swing also contains the basic user interface such as customizable painting, event handling, drag and drop etc.

The Java Foundation Classes (JFC) which supports many more features important to a GUI program comprises of Swing as well. The features which are supported by Java Foundation Classes (JFC) are the ability to create a program that can work in different languages, the ability to add rich graphics functionality etc.

There are several components contained in Swing toolkit such as check boxes, buttons, tables, text etc. Some very simple components also provide sophisticated functionality. For instance, text fields provide formatted text input or password field behavior. Furthermore, the file browsers and dialogs can be used according to one's need and can even be customized.

**4.5.6.2 Difference between Swings and AWT**

Table 4.1: Difference between Swings and AWT

|  |  |
| --- | --- |
| **Swings** | **AWT** |
| Swings are the light weight components. | AWTs are the heavy weight components. |
| Swings are developed by using pure java language. | AWTs are developed by using C and C++. |
| We can have different look and feel in swings. | This feature is not available in awt. |
| Swing has many advanced features like JTabel, JTabbedPane and JTree | This is not available in AWT. |

**4.5.6.3 Java Swing Class Hierarchy**

Diagram

Description automatically generated

Fig 4.20: Java Swing Class Hierarchy

**4.5.6.4 Swing Components**

All the components which are supported in AWT same components are also supported in Swings with a slight change in their class name.

Table 4.2: Swing Components

|  |  |
| --- | --- |
| **AWT Components** | **Swing Components** |
| Label | JLabel |
| TextField | JTextField |
| TextArea | JTextArea |
| Choice | JComboBox |
| Checkbox | JCheckBox |
| List | JList |
| Button | JButton |
| - | JRadioButton |
| - | JPasswordField |
| - | JTable |
| - | JTree |
| - | JTabbedPane |
| MenuBar | JMenuBar |
| Menu | JMenu |
| MenuItem | JMenuItem |
| - | JFileChooser |
| - | JOptionPane |

We will discuss only those components which are not discussed in AWT chapter.

**4.5.6.5 JTabbedPane class**

  The JTabbedPane container allows many panels to occupy the same area of the interface, and the user may select which to show by clicking on a tab.

Constructor

JTabbedPane tp = new JTabbedPane();

## Adding tabs to the JTabbedPane

Add tabs to a tabbed pane by calling addTab and passing it a String title and an instance of a class which should be called when we pressed a tab. That class should be a subclass of JPanel.

addTab(“String”,instance);

**4.5.6.6 Example program**

import javax.swing.\*;

import java.awt.\*;

public class TabbedPaneDemo extends JFrame

{

TabbedPaneDemo()

{

setLayout(new FlowLayout(FlowLayout.LEFT));

setTitle("Tabbed Demo");

setVisible(true);

setSize(500,500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

JTabbedPane pane = new JTabbedPane();

pane.addTab("Countries",new Count());

pane.addTab("Cities",new Cit());

add(pane);

}

public static void main(String a[])

{

new TabbedPaneDemo();

}

}

class Count extends JPanel

{

Count()

{

JButton b1 = new JButton("India");

JButton b2 = new JButton("SriLanka");

JButton b3 = new JButton("Australia");

add(b1);

add(b2);

add(b3);

}

}

class Cit extends JPanel

{Cit()

{ JCheckBox cb1 = new JCheckBox("Hyderabad");

JCheckBox cb2 = new JCheckBox("Banglore");

JCheckBox cb3 = new JCheckBox("Pune");

add(cb1);

add(cb2);

add(cb3);

}}

**4.5.6.7 JMenuBar, JMenu, JMenuItem**

A top-level window can have a menu bar associated with it. A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu. This concept is implemented in Java by the following classes: JMenuBar, JMenu, and JMenuItem. In general, a menu bar contains one or more JMenu objects. Each JMenu object contains a list of JMenuItem objects. Each JMenuItem object represents something that can be selected by the user. To create a menu bar, first create an instance of JMenuBar. This class only defines the default constructor. Next, create instances of JMenu that will define the selections displayed on the bar. Following are the constructors for Menu:

JMenu( )

JMenu(String *optionName*)

Here, *optionName* specifies the name of the menu selection. The first form creates an empty menu. Individual menu items are of type MenuItem.

**4.6 PROJECT TESTING**

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

## **4.6.1 Implementation**

## The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifies as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

## **4.6.2 Testing**

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

### **4.6.3 System Testing**

Testing has become an integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to user the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

**4.6.4 Module Testing**

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

**4.6.5 Integration Testing**

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

**4.6.6 Acceptance Testing**

When that user fined no major problems with its accuracy, the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

**4.6.7 TEST CASES**

Table 4.3: Test Cases

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case Id** | **Test Case Name** | **Test Case Desc.** | **Test Steps** | | | **Test Case Status** | **Test Priority** |
| **Step** | **Expected** | **Actual** |
| 01 | upload Mobile WebPages | Verify Mobile WebPages are uploaded or not | If user may not upload | we cannot do further operations | we can do further operations | High | High |
| 02 | process pages for kayo features | Verify either pages are process for kayo features or not | If WebPages may not loaded | He cannot process pages for kayo features | prepare processing features | High | High |
| 03 | JavaScript feature set | verify either the JavaScript feature set processed or not | If processing features may not prepare | we cannot process JavaScript feature set | the number of JavaScript tags displayed | High | High |
| 04 | HTML Feature set | verify either the HTML feature set processed or not | If processing features may not prepare | we cannot process HTML feature set | the number of redirection pages displayed | High | High |
| 05 | URL feature set | verify either the URL feature set processed or not | If processing features may not prepare | we cannot process URL feature set | Number of sub domains displayed | High | High |
| 06 | Mobile Feature set | verify either the Mobile feature set processed or not | If processing features may not prepare | we cannot process Mobile  feature set | number of mobile APIs displayed | High | High |
| 07 | view detection classification | Verify the both users logout or not | If they are not logout | They cannot complete the process | predicted WebPages are displayed | High | High |
| 08 | Detection Chart | verify the detection chart displayed or not | if the classification data may not save | detection chart may not displayed | the detection chart displayed successfully | High | High |

**5. SOURCE CODE**

package com;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Container;

import java.awt.Font;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JButton;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;

import javax.swing.UIManager;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import javax.swing.JScrollPane;

import javax.swing.JFileChooser;

import java.io.File;

import org.jfree.ui.RefineryUtilities;

public class KayoMaliciousDetection extends JFrame{

JPanel p1,p2;

JLabel l1;

JButton b1,b2,b3,b4,b5,b6,b7,b8;

Font f1,f2;

DefaultTableModel dtm;

JScrollPane jsp;

JTable table;

JFileChooser chooser;

File file;

static int malicious,benign;

public KayoMaliciousDetection(){

setTitle("Kayo Mobile Malicious Web Pages Detection");

f1 = new Font("Courier New",Font.BOLD+Font.ITALIC,18);

p1 = new JPanel();

l1 = new JLabel("<HTML><BODY><CENTER>Detecting Mobile Malicious Webpages in Real Time</CENTER></BODY></HTML>".toUpperCase());

l1.setFont(this.f1);

l1.setForeground(new Color(125,254,120));

p1.add(l1);

p1.setBackground(new Color(100,30,40));

f2 = new Font("Courier New",Font.BOLD,14);

chooser = new JFileChooser(new File("."));

chooser.setFileSelectionMode(chooser.DIRECTORIES\_ONLY);

p2 = new JPanel();

p2.setLayout(null);

b1 = new JButton("Upload Mobile Webpages");

b1.setFont(f2);

b1.setBounds(100,50,300,50);

p2.add(b1);

b1.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

clearTable();

int option = chooser.showOpenDialog(KayoMaliciousDetection.this);

if(option == chooser.APPROVE\_OPTION){

file = chooser.getSelectedFile();

JOptionPane.showMessageDialog(KayoMaliciousDetection.this,"Dataset Loaded");

}

}

});

b2 = new JButton("Process Pages For Kayo Features");

b2.setFont(f2);

b2.setBounds(450,50,300,50);

p2.add(b2);

b2.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

clearTable();

File list[] = file.listFiles();

for(int i=0;i<list.length;i++){

Object row[] = {(i+1),list[i].getName(),(list[i].length()/1000)+" KB"};

dtm.addRow(row);

}

}

});

b3 = new JButton("Javascript Features Set");

b3.setFont(f2);

b3.setBounds(100,120,300,50);

p2.add(b3);

b3.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ExtractFeatures.readPage(file);

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Javascript Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Keyword Count");

vf.dtm.addColumn("External Count");

vf.dtm.addColumn("Noscript Count");

for(int i=0;i<ExtractFeatures.javascript.size();i++){

JavascriptFeatures jf = ExtractFeatures.javascript.get(i);

Object row[] = {jf.getPage(),jf.getKeywordCount(),jf.getExternalCount(),jf.getNoscriptCount()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b4 = new JButton("HTML Features Set");

b4.setFont(f2);

b4.setBounds(450,120,300,50);

p2.add(b4);

b4.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View HTML Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Redirection Count");

for(int i=0;i<ExtractFeatures.html.size();i++){

HTMLFeatures hf = ExtractFeatures.html.get(i);

Object row[] = {hf.getPage(),hf.getRedirection()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b5 = new JButton("URL Features Set");

b5.setFont(f2);

b5.setBounds(100,190,300,50);

p2.add(b5);

b5.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View URL Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Subdomain Count");

for(int i=0;i<ExtractFeatures.url.size();i++){

URLFeatures uf = ExtractFeatures.url.get(i);

Object row[] = {uf.getPage(),uf.getSubdomain()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b6 = new JButton("Mobile Features Set");

b6.setFont(f2);

b6.setBounds(450,190,300,50);

p2.add(b6);

b6.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Mobile Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("SMS Count");

vf.dtm.addColumn("MMS Count");

vf.dtm.addColumn("Telephone Count");

for(int i=0;i<ExtractFeatures.mobile.size();i++){

MobileFeatures mf = ExtractFeatures.mobile.get(i);

Object row[] = {mf.getPage(),mf.getSmsCount(),mf.getMmsCount(),mf.getTelephoneCount()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b7 = new JButton("View Detection Classification");

b7.setFont(f2);

b7.setBounds(100,260,300,50);

p2.add(b7);

b7.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

malicious = 0;

benign = 0;

Classification.classify();

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Detection Classification");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Detection");

for(int i=0;i<Classification.malicious.size();i++){

Object row[] = {Classification.malicious.get(i),"Malicious Page"};

vf.dtm.addRow(row);

malicious = malicious + 1;

}

File list[] = file.listFiles();

for(int i=0;i<list.length;i++){

if(!Classification.malicious.contains(list[i].getName())){

Object row[] = {list[i].getName(),"Benign Page"};

vf.dtm.addRow(row);

benign = benign + 1;

}

}

vf.setVisible(true);

vf.setSize(600,600);

}

});

b8 = new JButton("Detection Chart");

b8.setFont(f2);

b8.setBounds(450,260,300,50);

p2.add(b8);

b8.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

Chart chart1 = new Chart("Detection Chart");

chart1.pack();

RefineryUtilities.centerFrameOnScreen(chart1);

chart1.setVisible(true);

}

});

dtm = new DefaultTableModel(){

public boolean isCellEditable(){

return false;

}

};

dtm.addColumn("Page No");

dtm.addColumn("Webpage Name");

dtm.addColumn("Webpage Size");

table = new JTable(dtm);

table.getTableHeader().setFont(new Font("Courier New",Font.BOLD,15));

table.setFont(new Font("Courier New",Font.BOLD,14));

table.setRowHeight(30);

jsp = new JScrollPane(table);

jsp.setBounds(10,330,900,300);

p2.add(jsp);

getContentPane().add(p1, "North");

getContentPane().add(p2, "Center");

}

public void clearTable(){

for(int i=dtm.getRowCount()-1;i>=0;i--){

dtm.removeRow(i);

}

}

public static void main(String a[])throws Exception{

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

KayoMaliciousDetection kayo = new KayoMaliciousDetection();

kayo.setVisible(true);

kayo.setExtendedState(JFrame.MAXIMIZED\_BOTH);

}

}

**MobileFeatures.java**

package com;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Container;

import java.awt.Font;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JButton;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.SwingUtilities;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;

import javax.swing.UIManager;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

import javax.swing.JScrollPane;

import javax.swing.JFileChooser;

import java.io.File;

import org.jfree.ui.RefineryUtilities;

public class KayoMaliciousDetection extends JFrame{

JPanel p1,p2;

JLabel l1;

JButton b1,b2,b3,b4,b5,b6,b7,b8;

Font f1,f2;

DefaultTableModel dtm;

JScrollPane jsp;

JTable table;

JFileChooser chooser;

File file;

static int malicious,benign;

public KayoMaliciousDetection(){

setTitle("Kayo Mobile Malicious Web Pages Detection");

f1 = new Font("Courier New",Font.BOLD+Font.ITALIC,18);

p1 = new JPanel();

l1 = new JLabel("<HTML><BODY><CENTER>Detecting Mobile Malicious Webpages in Real Time</CENTER></BODY></HTML>".toUpperCase());

l1.setFont(this.f1);

l1.setForeground(new Color(125,254,120));

p1.add(l1);

p1.setBackground(new Color(100,30,40));

f2 = new Font("Courier New",Font.BOLD,14);

chooser = new JFileChooser(new File("."));

chooser.setFileSelectionMode(chooser.DIRECTORIES\_ONLY);

p2 = new JPanel();

p2.setLayout(null);

b1 = new JButton("Upload Mobile Webpages");

b1.setFont(f2);

b1.setBounds(100,50,300,50);

p2.add(b1);

b1.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

clearTable();

int option = chooser.showOpenDialog(KayoMaliciousDetection.this);

if(option == chooser.APPROVE\_OPTION){

file = chooser.getSelectedFile();

JOptionPane.showMessageDialog(KayoMaliciousDetection.this,"Dataset Loaded");

}

}

});

b2 = new JButton("Process Pages For Kayo Features");

b2.setFont(f2);

b2.setBounds(450,50,300,50);

p2.add(b2);

b2.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

clearTable();

File list[] = file.listFiles();

for(int i=0;i<list.length;i++){

Object row[] = {(i+1),list[i].getName(),(list[i].length()/1000)+" KB"};

dtm.addRow(row);

}

}

});

b3 = new JButton("Javascript Features Set");

b3.setFont(f2);

b3.setBounds(100,120,300,50);

p2.add(b3);

b3.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ExtractFeatures.readPage(file);

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Javascript Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Keyword Count");

vf.dtm.addColumn("External Count");

vf.dtm.addColumn("Noscript Count");

for(int i=0;i<ExtractFeatures.javascript.size();i++){

JavascriptFeatures jf = ExtractFeatures.javascript.get(i);

Object row[] = {jf.getPage(),jf.getKeywordCount(),jf.getExternalCount(),jf.getNoscriptCount()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b4 = new JButton("HTML Features Set");

b4.setFont(f2);

b4.setBounds(450,120,300,50);

p2.add(b4);

b4.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View HTML Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Redirection Count");

for(int i=0;i<ExtractFeatures.html.size();i++){

HTMLFeatures hf = ExtractFeatures.html.get(i);

Object row[] = {hf.getPage(),hf.getRedirection()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b5 = new JButton("URL Features Set");

b5.setFont(f2);

b5.setBounds(100,190,300,50);

p2.add(b5);

b5.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View URL Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Subdomain Count");

for(int i=0;i<ExtractFeatures.url.size();i++){

URLFeatures uf = ExtractFeatures.url.get(i);

Object row[] = {uf.getPage(),uf.getSubdomain()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b6 = new JButton("Mobile Features Set");

b6.setFont(f2);

b6.setBounds(450,190,300,50);

p2.add(b6);

b6.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Mobile Features Set");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("SMS Count");

vf.dtm.addColumn("MMS Count");

vf.dtm.addColumn("Telephone Count");

for(int i=0;i<ExtractFeatures.mobile.size();i++){

MobileFeatures mf = ExtractFeatures.mobile.get(i);

Object row[] = {mf.getPage(),mf.getSmsCount(),mf.getMmsCount(),mf.getTelephoneCount()};

vf.dtm.addRow(row);

}

vf.setVisible(true);

vf.setSize(600,400);

}

});

b7 = new JButton("View Detection Classification");

b7.setFont(f2);

b7.setBounds(100,260,300,50);

p2.add(b7);

b7.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

malicious = 0;

benign = 0;

Classification.classify();

ViewFeatures vf = new ViewFeatures();

vf.setTitle("View Detection Classification");

vf.dtm.addColumn("Page Name");

vf.dtm.addColumn("Detection");

for(int i=0;i<Classification.malicious.size();i++){

Object row[] = {Classification.malicious.get(i),"Malicious Page"};

vf.dtm.addRow(row);

malicious = malicious + 1;

}

File list[] = file.listFiles();

for(int i=0;i<list.length;i++){

if(!Classification.malicious.contains(list[i].getName())){

Object row[] = {list[i].getName(),"Benign Page"};

vf.dtm.addRow(row);

benign = benign + 1;

}

}

vf.setVisible(true);

vf.setSize(600,600);

}

});

b8 = new JButton("Detection Chart");

b8.setFont(f2);

b8.setBounds(450,260,300,50);

p2.add(b8);

b8.addActionListener(new ActionListener(){

public void actionPerformed(ActionEvent ae){

Chart chart1 = new Chart("Detection Chart");

chart1.pack();

RefineryUtilities.centerFrameOnScreen(chart1);

chart1.setVisible(true);

}

});

dtm = new DefaultTableModel(){

public boolean isCellEditable(){

return false;

}

};

dtm.addColumn("Page No");

dtm.addColumn("Webpage Name");

dtm.addColumn("Webpage Size");

table = new JTable(dtm);

table.getTableHeader().setFont(new Font("Courier New",Font.BOLD,15));

table.setFont(new Font("Courier New",Font.BOLD,14));

table.setRowHeight(30);

jsp = new JScrollPane(table);

jsp.setBounds(10,330,900,300);

p2.add(jsp);

getContentPane().add(p1, "North");

getContentPane().add(p2, "Center");

}

public void clearTable(){

for(int i=dtm.getRowCount()-1;i>=0;i--){

dtm.removeRow(i);

}

}

public static void main(String a[])throws Exception{

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

KayoMaliciousDetection kayo = new KayoMaliciousDetection();

kayo.setVisible(true);

kayo.setExtendedState(JFrame.MAXIMIZED\_BOTH);

}

}

* 1. **EXPERIMENTAL RESULTS**

To execute project double click on ‘run.bat’ file to get below screen. Then there will a window similar to figure 6.1. There will 8 buttons each having its own functionality. Before proceeding with any option the user has to upload the required dataset that needs to be processed.

Graphical user interface, text, application, email

Description automatically generated

Fig 6.1: Run.bat File

Click on ‘Upload Mobile Webpages’ as shown in Figure 6.2 to upload entire dataset folder. This folder contains 12 web pages that have several benign and malicious elements in it.We upload the dataset as image shown below.

Graphical user interface, application, Word

Description automatically generated

Fig 6.2 : Upload Dataset

After uploading the document we get a message telling the dataset has been loaded as shown in the Figure 6.3 below. This means the dataset is loaded successfully and the user can carry on the further process.

Graphical user interface

Description automatically generated

Fig 6.3 : Dataset loaded

Now click on “Process Pages For Kayo Features’ as shown in Figure 6.4 to read dataset and prepare processing features, this prepares the webpages for further processing. This basically gives the tabular notation of the webpages with their relative sizes. From that the user can see how much size each webpage is carrying.

Graphical user interface, application, table

Description automatically generated

Fig 6.4 : Process pages for Kayo features

Now click on ‘Javascript Features Set’ to check for no of javascript tags used in web pages by web designer. This is one of the feature set that the user gets to examine with the kayo. In the figure 6.5 we can see the keyword count, external count and nonscript count of every webpage.

Table

Description automatically generated

Fig 6.5 : Javascript Features set

Now click on ‘HTML features set’ to get no of redirection pages used by web designer in webpage. From the figure 6.6 we can see every webpage has a redirection count associated with it. From it we can see how many times a particular is being redirected. This indirectly describes the malicious nature of the webpage.

Graphical user interface, table

Description automatically generated

Fig 6.6 : HTML Features set

Now click on ‘URL features set’ to get no of sub domains used by web designer in web page. The Figure 6.7 Shows the subdomain count of each webpage. Which describes how many subdomains that particular page contains. The subdomains differs from one page to another. Some of the webpages contain multiple Subdomains.

Table

Description automatically generated

Fig 6.7 : URL Features Set

Now click on ‘Mobile features set’ to get no of time mobile api (mobile api such as sms, smsto, mms etc) used by web designer inside web page. The Figure 6.8 shows the tabular denotion of each webpage which shows the sms count, mms count and telephone count. More number of counts means the webpage is on the verge of becoming a malicious webpage.

Table

Description automatically generated

Fig 6.8: Mobile Features Set

Now click on ‘View Detection Classification’ button to predict weather web page is malicious or not by analyzing above features count. The Figure 6.9 denotes the Detection Classification. If count > threshold then that page consider as malicious otherwise consider as benign.

Table

Description automatically generated

Fig 6.9: Detection Classification

Now click on ‘Detection Chart’ shown in 6.10 button to display graph for no of malicious and benign pages. This depicts the total number of Malicious page count and the total number of Benign Page count. From our dataset we see that the number of Malicious web page count is more than the number of Benign web page count.

Graphical user interface, application, table

Description automatically generated

Fig 6.10: Detection Chart

1. **CONCLUSION AND FUTURE SCOPE**

**7.1 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 behaviour 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.

**7.2 FUTURE SCOPE**

The Future enhancement of this project will be to take the dynamic websites into consideration and perform the same processes which we performed on the existing technique.

In Dynamic websites the data will be vast data and from that we’ll classify the webpages into malicious and benign in real time. The kAYO extension will process the webpage and in realtime it’ll assist the user which will save the user from malicious webpages and help to secure the confidential data of the user.

**REFERENCES**

[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]. 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.

[7]. 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.

[8]. 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.

[9]. 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.

[10]. 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.

[11]. 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.

[12]. M. Boodaei. Mobile users three times more vulnerable to phishing attacks. http://www.trusteer.com/blog /mobile -users-threetimes -more-vul nerable-to-phishing-attacks, 2011.

[13. ]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 14th Workshop on Mobile Computing Systems and Applications (HotMobile), 2013

 [14]. 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

[15]. S. Chakradeo, B. Reaves, P. Traynor, and W. 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.

 [16]. W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In Proceedings of the 20th USENIX Security Symposium, 2011.

 [17]. 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.

[18]. S. Gajek, A.-R. Sadeghi, C. Stuble, 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.

[19]. 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.

[20]. A. Ikinci, T. Holz, and F. Freiling. Monkey-spider: Detecting malicious websites with low-interaction honeyclients. In Proceedings of Sicherheit, Schutz und Zuverlassigkeit, 2008.