Optimal Distributed Malware Defense in Mobile Networks with Heterogeneous Devices

A M M A R Y A S I R A L I
CWID : 893253526

D R . S H A W N W A N G

(CPSC 597-13 13361)
Spring 2016
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
CALIFORNIA STATE UNIVERSITY FULLERTON

CALIFORNIA STATE UNIVERSITY
FULLERTON™

Department of Computer Science

This project has been satisfactorily demonstrated and is of suitable form.

This project report is acceptable in partial completion of the requirements for the Master of Science degree in Computer Science.

OPTIMAL DISTRIBUTED MALWARE DEFENSE SYSTEM

Project Title

AMMAR YASIR ALI

Student Name

DR. SHAWN WANG

Advisor's Name

Advisor's signature Date

DR. BIN CONG

Reviewer's name

Reviewer's signature Date

CPSC 597-13 13361

"OPTIMAL DISTRIBUTED MALWARE DEFENSE SYSTEM"

By:

Ammar Yasir Ali (893253526)

CALIFORNIA STATE UNIVERSITY
FULLERTON

Dr. Shawn Wang

College of Engineering & Computer Science
California State University, Fullerton

Spring 2016

ACKNOWLEDGEMENT

First of all, I am greatly thankful to ALMIGHTY GOD, the Most Merciful and Beneficent, who gave me strength, patience, courage, forbearance, spirit and integrity to finish this project. This project could not have been possible without the kind help of many individuals. I would like to thank my Project Adviser **Dr. Shawn Wang** and my Project Reviewer **Dr. Bin Cong**. They constantly helped me in making this project successful with their best suggestions. I also express my heartiest thanks to my parents and siblings for all their support in my educational endeavors and also too all my friends for encouraging me all the time. Finally, a special thanks to California State University, Fullerton. Studying and completing my Master's Degree at university were special.

May God Mighty Always Bless, Cherish and Prosper Us All Under His Protection.

AMMAR YASIR ALI

ABSTRACT

Nowadays, malware attacks are becoming much more frequent in mobile networks. I shall deploy an efficient defence system to protect the mobile network against these infections and to help the already infected nodes to recover which is important in prevention of serious spreading and outbreaks. The technological challenges nowadays are that the mobile devices are heterogeneous concerning the terms of operating systems. The malware infects any targeted system in any opportunistic fashion by the means of local and global connectivity, while the to-be-deployed defence system would be usually resource limited on the other hand. In this project, I will investigate the optimal solution to distribute the content-based signatures of malware, which will help to detect the corresponding malware and disable further propagation, which will help in minimizing the number of infected nodes. I will model the defence system with practical assumptions addressing all the above challenges that might have not been addressed in the previous analytical work. Based on the framework/infrastructure of optimizing the system welfare utility, which is usually weighted on summation of the individual utility depending on the final number of infected nodes by the means of the signature allocation, I shall be using an encounter-based distributed algorithm which will be based on Metropolis sampler. Through theoretical analysis and simulations with both synthetic and realistic mobility traces, I will show that the distributed algorithm achieves the optimal solution, and performs efficiently in realistic environments.

Keywords : Security threat, mobile malware, distributed algorithm, heterogeneous mobile networks.

TABLE OF CONTENTS

Chapter 1: INTRODUCTION 1
1. INTRODUCTION 2
1.1 PROBLEM STATEMENT 5
1.1.1 DISADVANTAGES OF EXISTING SYSTEM: 6
1.2 AIM, SCOPE, GOALS AND OBJECTIVE 6
1.2.1 ADVANTAGES OF PROPOSED SYSTEM: 7
1.3 LITERATURE SURVEY 7
1.3.1 Supporting Anonymous Location Queries in Mobile Environments With Privacy Grid 7
1.3.2 On The Value of a Random Minimum Weight Steiner Tree 8
1.3.3 Random Key Pre-distribution Schemes for Sensor Networks 8
1.3.4 Enhancing Base Station Security in Wireless Sensor Networks 9
1.3.5 Intrusion Tolerance and Anti-Traffic Analysis Strategies for Wireless Sensor Networks 10
1.4 PROJECT ACTIVITIES 10
1.5 METHODOLOGIES 11
1.6 DEVELOPMENT ENVIRONMENT 12
1.6.1 Hardware 12
1.6.2 Software 12
1.7 OPERATIONAL ENVIRONMENT 13
1.7.1 Software 13
1.7.2 Hardware 13
Chapter 2: REQUIREMENTS DESCRIPTION 14
2. REQUIREMENTS DESCRIPTION 15

2.1 Relevant Technologies & Software Environment 15
2.1.1 Android Platform 15
2.1.2 Java Technology 16
2.1.3 The Java Programming Language 16
2.1.2 Android Studio 18
Project Structure 19
Gradle Build System 20
Debug and Profile Tools 21
Inline debugging 21
Memory and CPU monitor 22
Heap dump 22
Allocation tracker 23
Data file access 24
Code inspections 25
Annotations in Android Studio 26
Inferring Nullability 27
Validating annotations 27
Log messages 28
2.1 Functional Requirements 28
2.1.1 User Use Case #1: Create Account 28
2.1.2 User Use Case #2: Login 28
2.1.3 User Use Case #3: Logout 29
2.1.4 User Use Case #4: Scan Device for Malwares 29
2.1.5 User Use Case #5: Check the Battery Health 29
2.1.6 User Use Case #6: Memory Booster 29
2.2 User Interface Specification 30

2.3 Non-functional Requirements 30
2.3.1 Product Non-functional Requirements 30
2.3.2 Process Non-functional Requirements 30
2.4 System Study 30
2.4.1 FEASIBILITY STUDY 30
2.4.2 Economical Feasibility 31
2.4.3 Technical Feasibility 31
2.4.4 Social Feasibility 31
Chapter 3: DESIGN DESCRIPTION 33
3. DESIGN DESCRIPTION 34
3.1 SYSTEM DESIGN 34
3.1.1 ACTIVITY LIFE CYCLE 34
3.1.2 DATA FLOW DIAGRAM 34
3.2 UML DIAGRAMS 36
GOALS: 36
3.2.1 USE CASE DIAGRAM: 38
Use Case Description: 39
3.2.2 Class Diagram: 43
3.2.3 Sequence Diagram: 43
3.2.4 Collaboration Diagram 44
3.2.5 System Architecture: 45
Chapter 4: IMPLEMENTATION 47
4. IMPLEMENTATION 48
4.1 Tools & Technology Used 48
4.1.1 Android Studio 48
4.1.2 Extensible Markup Language (XML) 48

4.1.3 Java Programming Language 48
4.1.4 SQLite Database 49
4.2 INPUT DESIGN 49
4.2.1 OBJECTIVES 50
4.3 OUTPUT DESIGN 51
4.4 User Interface (GUI) Design 52
4.5 MODULES 61
4.6 MODULE DESCRIPTION: 61
4.6.1 Malware signature finder and Spreading Model: 61
4.6.2 Problem Formulation and Centralized Algorithm: 61
4.6.3 The Metropolis Sampler: 62
4.6.4 Performance Evaluation: 62
Chapter 5: TESTING & INTEGRATION 63
5 TESTING AND INTEGRATION 64
5.1 SYSTEM TESTING 64
5.2 TYPES OF TESTS 65
5.2.1 Unit Testing 65
5.2.2 Integration Testing 65
5.2.3 Functional Test 65
5.2.4 System Test 66
5.2.5 White Box Testing 66
5.2.6 Black Box Testing 66
5.3 Unit Testing 67
5.3.1 Test Strategy and Approach 67
5.4 Integration Testing 67
5.5 Acceptance Testing 68

Chapter 6: INSTALLATION & OPERATING INSTRUCTIONS.....69

6 INSTALLATION & OPERATING INSTRUCTIONS 70

6.1 Installation Instruction (Users) 70

6.1.1 Setting Up Your Device 70

6.2 Connecting Device to Computer 71

6.2.1 Connecting Android 2.0 – 3.0 (Ginger Bread – Honey Comb) 71

6.2.2 Connecting Android 4.0 – 4.1.2 (Ice Cream Sandwich) 72

6.2.3 Connecting Android 4.2 or Higher 73

6.3 Transfer APK File Between Your Computer and Android Device 75

6.4 Installing APK File On Android Device 75

6.5 Installation Instruction (Developers) 75

Chapter 7: FUTURE WORK & RECOMMENDATIONS 79

7 FUTURE WORK & RECOMMENDATIONS FOR ENHANCEMENTS ... 80

Chapter 8: RESULTS 81

Chapter 9: REFERENCES & BIBLIOGRAPHY 85

8. REFERENCES & BIBLIOGRAPHY 86

LIST OF FIGURES

Figure 1. Malware Sample 5

Figure 2: Approach 7

Figure 3. Waterfall Model 11

Figure 4: Android Software Environment 15

Figure 5: JVM 16

Figure 6: Java Program 17

Figure 7: Java API Model 17

Figure 8: Java SDK & JRE 18
Figure 9: Android Studio Project Structure 19
Figure 10: Monitor Memory and CPU Usage 22
Figure 11: HPROF viewer with Heap Dump 23
Figure 12: Allocation Tracker 24
Figure 13: Inspections Configurations 26
Figure 14: Activity Life Cycle 34
Figure 15: Data Flow Diagram 35
Figure 16: UML Diagram Types 37
Figure 17: Use Case Diagram 38
Figure 18: Class Diagram 43
Figure 19: Sequence Diagram 44
Figure 20: Collaboration Diagram 45
Figure 21: System Architecture Diagram 46
Figure 22: User Interface (Loading Window) 52
Figure 23: User Interface (Login Window) 53
Figure 24: User Interface (Registration Window) 54
Figure 25: User Interface (Forgot Password) 55
Figure 26: User Interface (Home Screen) 56
Figure 27: User Interface (Scanner Screen) 57
Figure 28: User Interface (Resolved Screen) 58
Figure 29: User Interface (Memory Booster Screen) 59
Figure 30: User Interface (Battery Information Screen) 60
Figure 31: Android Testing 64
Figure 32: Setting Up Device 70
Figure 33: USB Connected (Android 2.0 – 3.0) 71

Figure 34: Connect USB Storage (Android 2.0 – 3.0) 72
Figure 35: Connect USB Storage (Android 4.0 – 4.1.2) 73
Figure 36: USB Debugging (Android 4.2 or Higher) 74
Figure 37: Connect USB Storage (Android 4.0 – 4.1.2) 74
Figure 38: Android Studio Dialog 76
Figure 39: Android Studio Open File or Project Dialog 76
Figure 40: Android Studio Open Project from The File Menu Dialog 77
Figure 41: Android Studio Open File or Project Dialog 78
Figure 42: Frequent API Parameters in Malware 83
Figure 43: Performance of API-Based Models 83
Figure 44: Model Comparison 84
Figure 45: Distribution of DroidAPIMiner Processing Time 84

List of Tables

Table 1: Hardware Specification for the Development Environment 12
Table 2: Software Specification for the Development Environment 13
Table 3: Software Specification for the Operational Environment 13
Table 4: Hardware Specification for the Operational Environment 13
Table 5: Use Case Description (Registration) 39
Table 6: Use Case Description (Login) 40
Table 7: Use Case Description (Memory Booster) 41
Table 8: Use Case Description (Scanner) 41
Table 9: Use Case Description (Battery Information) 42

Chapter 1:

INTRODUCTION

1. INTRODUCTION

The target landscape for malware attacks (i.e., viruses, spam bots, worms, and other malicious software) has moved considerably from the large-scale Internet to the growingly popular mobile networks (P. Wang, M. Gonzalez, C. Hidalgo, and A. Barabasi, 2009), with a total count of more than one million known mobile malware instances reported last year (2015) (G Data Software, 2015). This is mainly because of two reasons. One is the emergence of powerful mobile devices, such as the iPhone, Android, and Windows devices, and increasingly diversified mobile applications, such as multimedia messaging service (MMS), mobile games, and peer-to-peer file sharing. The other reason is the emergence of mobile Internet, which indirectly induces the malware. Malware residing in the wired Internet can now use mobile devices and networks to propagate. The potential effects of malware propagation on mobile users and service providers would be very serious (G. Lawton, 2009).

Understanding the behaviors and damages of mobile malware, and designing an efficient detection and defense system are necessary to prevent large-scale outbreaks (M. Khouzani, S. Sarkar, and E. Altman, 2010), (Z. Zhu, G. Cao, S. Zhu, S. Ranjan, and A. Nucci, 2009); and it should be an urgent and high-priority research agenda. Currently, mobile malware can propagate through two different dominant approaches. Via MMS, a malware may send a copy of itself to all devices whose numbers are found in the address book of the infected handset. This kind of malware propagates in the social graph formed by the address books, and can spread very quickly without geographical limitations. The other approach is to use the short-range wireless media such as Bluetooth to infect the devices in proximity as “proximity malware.” Recent work of Wang et al. has investigated the proximity malware propagation features, and finds that it spreads slowly because of the human mobility, which offers ample opportunities to deploy the

defense system. However, the approach for efficiently deploying such a system is still an ongoing research problem. In this paper and project, I shall be addressing the challenges of designing a defense system for both MMS and proximity malware. I shall introduce an optimal distributed solution to efficiently avoid malware spreading and to help infected nodes to recover. Consider a mobile network where a portion of the nodes are infected by malware.

This research problem is to focus and implementing an efficient defense system to help infected nodes to recover and prevent healthy nodes from further infection. Typically, we disseminate the content-based signatures of known malware to as many nodes as possible (G. Zyba, G. Voelker, M. Liljenstam, A. Me´hes, and P. Johansson, 2009). Consequently, distributing these signatures into the whole network while avoiding unnecessary redundancy is my optimization goal. However, to address the above problem in the realistic mobile environment is challenging for several reasons. First, typically we cannot rely on centralized algorithms to distribute the signatures because the service infrastructure is not always available. Therefore, a sensible way for signature distribution is to use a distributed and cooperative way among users. Second, mobile devices in general have limited resources, i.e., CPU, storage, and battery power. Although their storage and CPU capacity has been increasing rapidly, it is still very resource-limited compared with PCs / desktops. Hence, in the to-be-deployed defense system, I shall adequately consider the limitation of resources, especially the memory capacity to store the defense software and signatures. Finally, the mobile devices are heterogeneous in terms of operating systems (OS), and different malware targets different systems. These heterogeneous features as well as the propagation via both local and global connectivity should be taken into consideration in the design of defense system for real use.

In this project, I will develop an android application based on an optimal signature distribution scheme by considering the following realistic modeling assumptions:

1. the network contains heterogeneous devices as nodes,
2. different types of malware can only infect the targeted systems, and
3. the storage resource of each device for the defense system is limited. These assumptions are usually not addressed in previous analytical works for simplicity reasons.

My contributions are summarized as follows:

- Formulate the optimal signature distribution problem with the consideration of the heterogeneity of mobile devices and malware, and the limited resources of the defense system. Moreover, formulated model is suitable for both the MMS and proximity malware propagation.
- Use a centralized greedy algorithm for the signature distribution problem. Prove that the proposed greedy algorithm obtains the optimal solution for the system, which provides the benchmark solution for distributed algorithm design.
- Use an encounter-based distributed algorithm to disseminate the malware signatures using Metropolis sample. It only relies on local information and opportunistic contacts. Through theoretical proof and extensive real and synthetic traces driven simulations, show that the distributed algorithm approaches the optimal system performance.

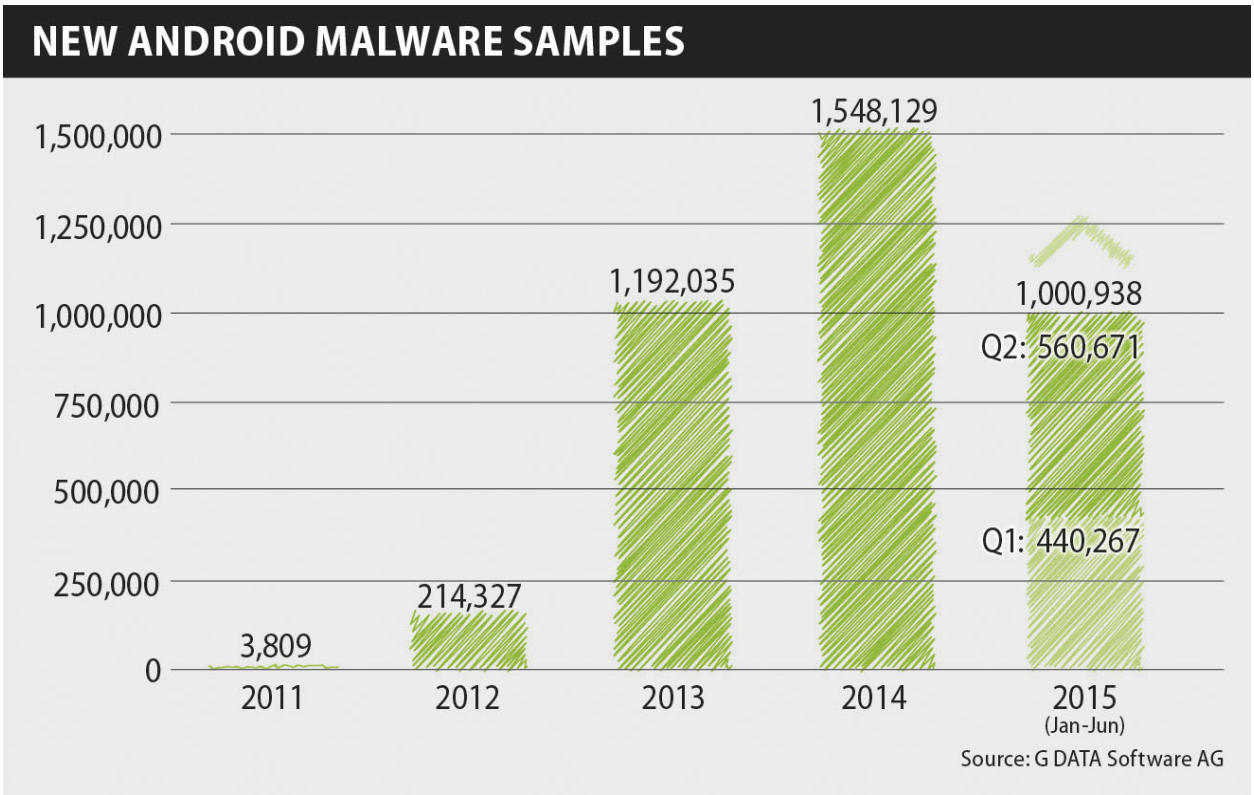


Figure 1. Malware Sample

1.1 PROBLEM STATEMENT

Mobile malware can propagate through two different dominant approaches.

1. Via MMS, a malware may send a copy of itself to all devices whose numbers are found in the address book of the infected handset. This kind of malware propagates in the social graph formed by the address books, and can spread very quickly without geographical limitations.
2. The other approach is to use the short-range wireless media such as Bluetooth to infect the devices in proximity as "proximity malware."

Recent work of Wang et al. has investigated the proximity malware propagation features, and finds that it spreads slowly because of the human mobility, which offers ample opportunities

to deploy the defense system. However, the approach for efficiently deploying such a system is still an ongoing research problem.

1.1.1 DISADVANTAGES OF EXISTING SYSTEM:

- There is a problem for optimal signature distribution to defend mobile networks against the propagation of both proximity and MMS-based malware.
- The existing system offers only protection against only one attack at a time.

1.2 AIM, SCOPE, GOALS AND OBJECTIVE

- To Design a defense system for both MMS and proximity malware. My research problem is to deploy an efficient defense system to help infected nodes to recover and prevent healthy nodes from further infection.
- I will formulate the optimal signature distribution problem with the consideration of the heterogeneity of mobile devices and malware, and the limited resources of the defense system. Moreover, my formulated model is suitable for both the MMS and proximity malware propagation.
- I will use a centralized greedy algorithm (Li, Hui, Jin, Su & Zeng, 2014) for the signature distribution problem. I will prove that the proposed greedy algorithm obtains the optimal solution for the system, which provides the benchmark solution for my distributed algorithm design.
- I will use an encounter-based distributed algorithm (Li, Hui, Jin, Su & Zeng, 2014) to disseminate the malware signatures using Metropolis sampler (Johnson & Flegal, 2014). It only relies on local information and opportunistic contacts.

1.2.1 ADVANTAGES OF PROPOSED SYSTEM:

- The system provides optimal signature distribution to defend mobile networks against the propagation of both proximity and MMS-based malware.
- The proposed system offers protection against both MMS based attack and Bluetooth based attack at the same time.

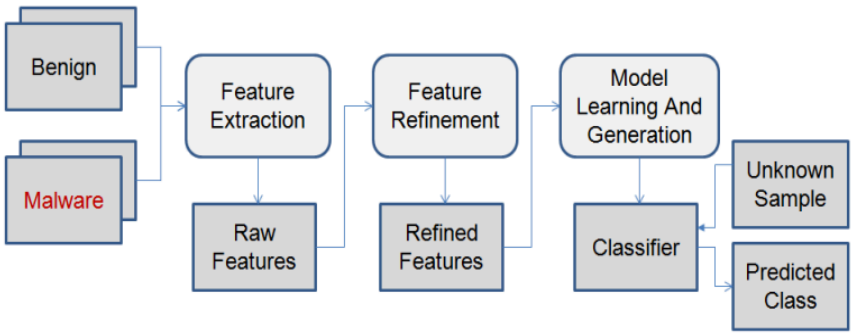


Figure 2: Approach

1.3 LITERATURE SURVEY

1.3.1 SUPPORTING ANONYMOUS LOCATION QUERIES IN MOBILE ENVIRONMENTS WITH PRIVACY GRID

Authors: B. Bamba, L. Liu, P. Pesti, and T. Wang

This paper presents PrivacyGrid - a framework for supporting anonymous location-based queries in mobile information delivery systems. The PrivacyGrid framework offers three unique capabilities. First, it provides a location privacy protection preference profile model, called location P3P, which allows mobile users to explicitly define their preferred location privacy requirements in terms of both location hiding measures (e.g., location k-anonymity and location l-diversity) and location service quality measures (e.g., maximum spatial resolution and maximum temporal resolution). Second, it provides fast and effective location cloaking

algorithms for location k-anonymity and location l-diversity in a mobile environment. We develop dynamic bottom-up and top-down grid cloaking algorithms with the goal of achieving high anonymization success rate and efficiency in terms of both time complexity and maintenance cost. A hybrid approach that carefully combines the strengths of both bottom-up and top-down cloaking approaches to further reduce the average anonymization time is also developed. Last but not the least, PrivacyGrid incorporates temporal cloaking into the location cloaking process to further increase the success rate of location anonymization. We also discuss PrivacyGrid mechanisms for supporting anonymous location queries. Experimental evaluation shows that the PrivacyGrid approach can provide close to optimal location k-anonymity as defined by per user location P3P without introducing significant performance penalties.

1.3.2 On The Value of a Random Minimum Weight Steiner Tree

Authors: B. Bollobas, D. Gamarnik, O. Riordan, and B. Sudakov

Consider a complete graph on n vertices with edge weights chosen randomly and independently from an exponential distribution with parameter 1. Fix k vertices and consider the minimum weight Steiner tree which contains these vertices. We prove that with high probability the weight of this tree is $(1 + o(1))(k - 1)(\log n - \log k)/n$ when $k = o(n)$ and $n \to \infty$. 1.

1.3.3 Random Key Pre-distribution Schemes for Sensor Networks

Authors: H. Chan, A. Perrig, and D. Song

Key establishment in sensor networks is a challenging problem because asymmetric key cryptosystems are unsuitable for use in resource constrained sensor nodes, and also because the nodes could be physically compromised by an adversary. We present three new mechanisms for key establishment using the framework of pre-distributing a random set of keys to each node. First, in the q-composite keys scheme, we trade off the unlikeliness of a large-scale network

attack in order to significantly strengthen random key pre-distribution's strength against smaller-scale attacks. Second, in the multipath-reinforcement scheme, we show how to strengthen the security between any two nodes by leveraging the security of other links. Finally, we present the random-pairwise keys scheme, which perfectly preserves the secrecy of the rest of the network when any node is captured, and also enables node-to-node authentication and quorum-based revocation.

1.3.4 Enhancing Base Station Security in Wireless Sensor Networks

Authors: **J. Deng, R. Han, and S. Mishra**

Wireless sensor networks that are deployed in applications such as battlefield monitoring and home sentry systems face acute security concerns, including eavesdropping, forgery of sensor data, denial of service attacks, and the physical compromise of sensor nodes. Sensor networks are often organized hierarchically, with a base station serving as a gateway for collecting data from a multi-hop network of resource constrained sensor nodes. Prior work that has focused on securing the routing between sensor nodes has assumed that the base station is sufficiently powerful to defend itself against security threats. This paper considers strategies for securing the sensor network against a variety of threats that can lead to the failure of the base station, which represents a central point of failure. First, multipath routing to multiple destination base stations is analyzed as a strategy to provide tolerance against individual base station attacks and/or compromise. Second, confusion of address and identification fields in packet headers via hashing functions is explored as a technique to help disguise the location of the base station from eavesdroppers. Third, relocation of the base station in the network topology is studied as a means of enhancing resiliency and mitigating the scope of damage.

1.3.5 Intrusion Tolerance and Anti-Traffic Analysis Strategies for Wireless Sensor Networks

Authors: J. Deng, R. Han, and S. Mishra

Wireless sensor networks face acute security concerns in applications such as battlefield monitoring. A central point of failure in a sensor network is the base station, which acts as a collection point of sensor data. In this paper, we investigate two attacks that can lead to isolation or failure of the base station. In one set of attacks, the base station is isolated by blocking communication between sensor nodes and the base station, e.g. by DOS attacks. In the second attack, the location of the base station is deduced by analyzing data traffic towards the base station, which can lead to jamming and/or discovery and destruction of the base station. To defend against these attacks, two secure strategies are proposed. First, secure multi-path routing to multiple destination base stations is designed to provide intrusion tolerance against isolation of a base station. Second, anti-traffic analysis strategies are proposed to help disguise the location of the base station from eavesdroppers. A performance evaluation is provided for a simulated sensor network, as well as measurements of cryptographic overhead on real sensor nodes.

1.4 PROJECT ACTIVITIES

The objectives are achieved in the following way:

- Functional Models (Activity Diagram, Use-Case Diagram, Use-Case Descriptions)
- Structural Model (Class Responsibility Collaborator (CRC) Cards, Class and Object Diagram)
- Behavioral Models (Sequence Diagram, Communication Diagram).
- Designing Database and User Interface.
- Coding and Unit Testing for:
    - Common module: Login/Register/Logout, etc.

- Testing Plan and Testing.

1.5 METHODOLOGIES

The methodology that is used in development of this software is the Waterfall model. With this model, I was able to design the processes and developed it from one phase to the next and it also made possible to go backward such as from design back to analysis.

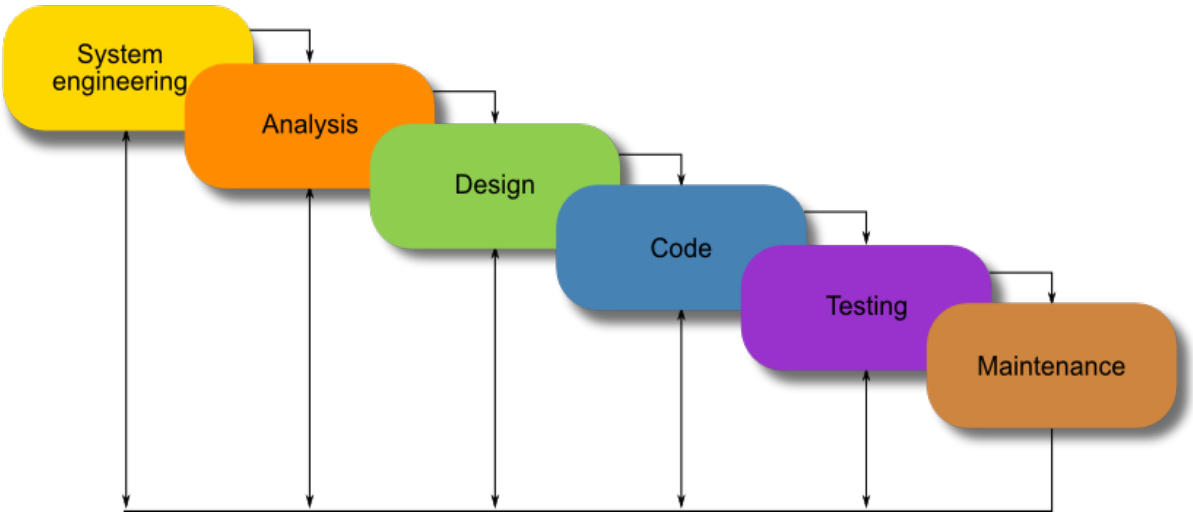


Figure 3. Waterfall Model

This methodology has the following phases:

- **System Engineering:** An interdisciplinary approach and means to enable the realization of successful systems. Successful systems must satisfy the needs of their customers, users and other stakeholders.
- **Requirement Gathering and Analysis:** The analyst gathers specific data, all features and details are required to build the system.
- **System Design:** Designing all features that related to the project: logical design and physical design.

- **Code:** The programmer will develop a program to unit test.
- **Integration and Testing:** After testing each unit; all units developed in the previous stage are joined to the system.
- **Deployment of the System:** The system will be released for the user.
- **Maintenance:** This phase handles errors corrections that are related to tools and components that used in the project.

1.6 DEVELOPMENT ENVIRONMENT

1.6.1 HARDWARE

Hardware	Description
Manufacturer	Apple Inc.
Model	MacBook Pro (Retina, 13 inch, Late 2013)
Processor	2.4 GHz Intel Core i5
RAM	4 GB 1600 MHz DDR3
Hard Disk Drive	120 GB Flash Storage
Operating System	OS X El Capitan (Version 10.11.4)
Graphics	Intel Iris 1536 MB

Table 1: Hardware Specification for the Development Environment

1.6.2 SOFTWARE

Type	Software
Integrated Development Environment (IDE)	Android Studio
Programming / Coding Languages	XML, Java, SQLite

Software Development Kit (SDK)	Android SDK
Development Kit	Java Development Kit (JDK) Java Runtime Environment (JRE)
Databases	MySQL, Firebase Database
Application Program Interfaces (APIs) Used	Google Droid API Miner, Application-specific resources APIs, Android framework resources APIs, DVM related resources APIs, System resources APIs Utilities APIs

Table 2: Software Specification for the Development Environment

1.7 OPERATIONAL ENVIRONMENT

1.7.1 SOFTWARE

Software	Requirements
Operating System	Android OS 2.3 or Above

Table 3: Software Specification for the Operational Environment

1.7.2 HARDWARE

Hardware	Requirements
Smartphone	Android Smartphone

Table 4: Hardware Specification for the Operational Environment

Chapter 2:

REQUIREMENTS DESCRIPTION

2. REQUIREMENTS DESCRIPTION

2.1 Relevant Technologies & Software Environment

2.1.1 Android Platform

Android is an open operating mobile phone system based on Linux platform and launched by Google in 2007. It allows any mobile terminal manufacturers to join Android alliance. (Android, 2016)

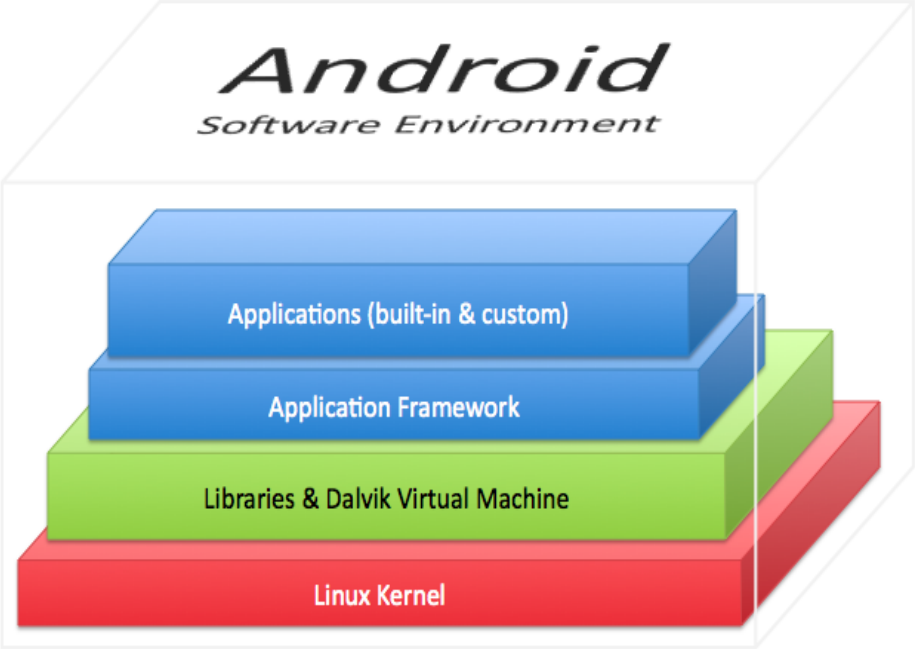


Figure 4: Android Software Environment

Android platform is composed of operating system, middleware, and user interface and application software. There are four layers of Android system architecture. From Figure 2.1, they are application, application framework, libraries and Dalvik Virtual Machine and Linux Kernel from the highest to the lowest layer. Android owns four developing components. Activity applies for expression; service is used for backgrounder and not provides interface render; broadcast

receiver receives broadcast; content provider supports storage and reading data in many applications, and it can be seen as database. (Android, 2016)

2.1.2 Java Technology

Java technology is both a programming language and a platform. Java was introduced by Sun Microsystems in 1995 and instantly created a new sense of the interactive possibilities of the Web. Both of the major Web browsers include a Java virtual machine. Almost all major operating system developers (IBM, Microsoft, and others) have added Java compilers as part of their product offerings. (Oracle Java, 2016)

2.1.3 The Java Programming Language

Java is a programming language expressly designed for use in the distributed environment of the Internet. It was designed to have the "look and feel" of the C++ language, but it is simpler to use than C++ and enforces an object-oriented programming model. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. It can also be used to build a small application module or applet for use as part of a Web page. Applets make it possible for a Web page user to interact with the page. (Oracle Java, 2016)

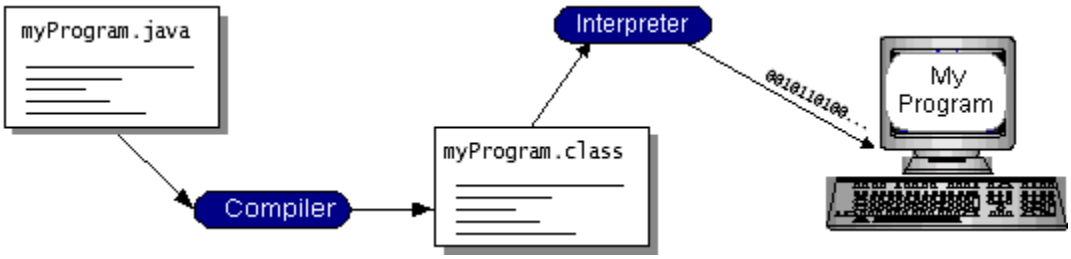


Figure 5: JVM

The major characteristics of Java are:

The programs you create are portable in a network. Your source program is compiled into what Java calls bytecode, which can be run anywhere in a network on a server or client that has a

Java virtual machine. The Java virtual machine interprets the bytecode into code that will run on the real computer hardware. This means that individual computer platform differences such as instruction lengths can be recognized and accommodated locally just as the program is being executed. Platform-specific versions of your program are no longer needed. (Oracle Java, 2016)

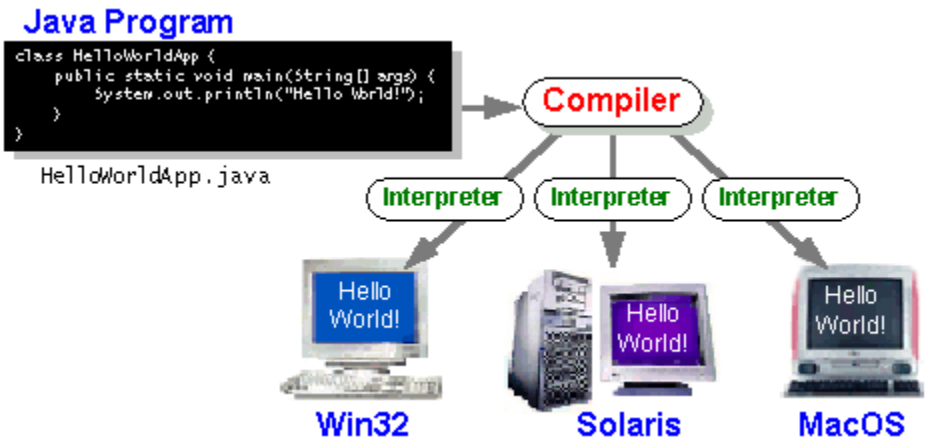


Figure 6: Java Program

- The code is robust, here meaning that, unlike programs written in C++ and perhaps some other languages, the Java objects can contain no references to data external to themselves or other known objects. This ensures that an instruction cannot contain the address of data storage in another application or in the operating system itself, either of which would cause the program and perhaps the operating system itself to terminate or "crash." The Java virtual machine makes a number of checks on each object to ensure integrity. (Oracle Java, 2016)

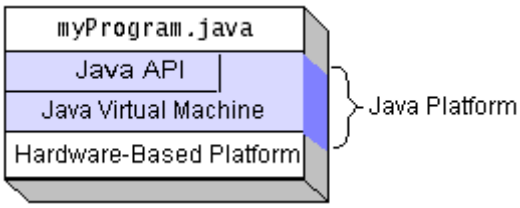


Figure 7: Java API Model

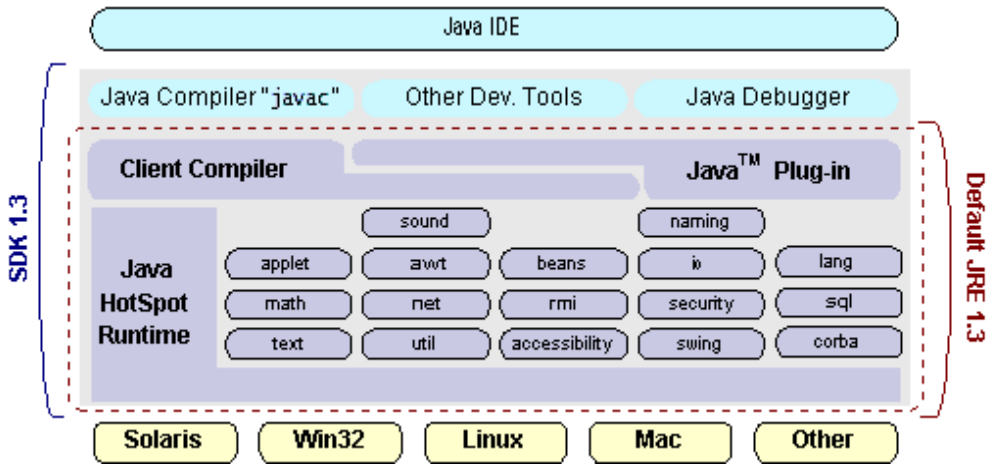


Figure 8: Java SDK & JRE

- Java is object-oriented, which means that, among other characteristics, an object can take advantage of being part of a class of objects and inherit code that is common to the class. Objects are thought of as "nouns" that a user might relate to rather than the traditional procedural "verbs." A method can be thought of as one of the object's capabilities or behaviors. (Oracle Java, 2016)
- In addition to being executed at the client rather than the server, a Java applet has other characteristics designed to make it run fast. (Oracle Java, 2016)

The Java virtual machine includes an optional just-in-time compiler that dynamically compiles bytecode into executable code as an alternative to interpreting one bytecode instruction at a time. In many cases, the dynamic JIT compilation is faster than the virtual machine interpretation. (Oracle Java, 2016)

2.1.2 Android Studio

Android Studio is the official IDE for Android app development, based on IntelliJ IDEA. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more

features that enhance your productivity when building Android apps, such as: (Android Studio, 2016)

- A flexible Gradle-based build system
- Build variants and multiple APK file generation
- Code templates to help you build common app features
- A rich layout editor with support for drag and drop theme editing
- Lint tools to catch performance, usability, version compatibility, and other problems
- Code shrinking with ProGuard and resource shrinking with Gradle
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Project Structure

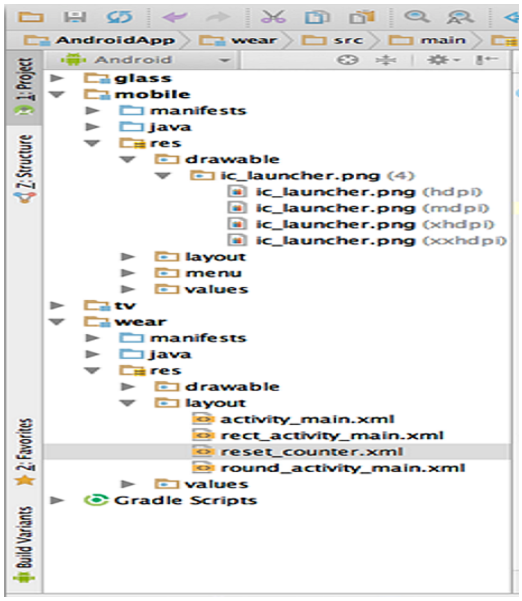


Figure 9: Android Studio Project Structure

Each project in Android Studio contains one or more modules with source code files and resource files. Different types of modules include:

- Android app modules
- Test modules

- Library modules
- App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 2.2. This view is organized by modules to provide quick access to the key source files of your project. (Android Studio, 2016)

All the build files are visible at the top level under Gradle Scripts and each app module contains the following three elements:

- manifests: Manifest files.
- java: Source code files.
- res: Resource files.

The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select Project from the Project drop-down (in figure 1, it's showing as Android). (Android Studio, 2016)

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the Problems view of your project displays links to the source files containing any recognized coding and syntax errors, such as missing an XML element closing tag in a layout file. (Android Studio, 2016)

GRADLE BUILD SYSTEM

Android Studio uses Gradle as the foundation of the build system, with more Android-specific capabilities provided by the Android Plugin for Gradle. This build system runs as an integrated tool from the Android Studio menu and independently from the command line. You can use the features of the build system to: (Android Studio, 2016)

- Customize, configure, and extend the build process.
- Create multiple APKs for your app with different features using the same project and modules.
- Reuse code and resources across source sets.

The flexibility of Gradle enables you to achieve all of this without modifying your app's core source files. To build an Android Studio project, see Building and Running from Android Studio. To configure custom build settings in an Android Studio project, see Configuring Gradle Builds. (Android Studio, 2016)

Debug and Profile Tools

Android Studio assists you in debugging and improving the performance of your code, including inline debugging and performance analysis tools. (Android Studio, 2016)

Inline debugging

Use inline debugging to enhance your code walk-throughs in the debugger view with inline verification of references, expressions, and variable values. Inline debug information includes:

- Inline variable values
- Referring objects that reference a selected object
- Method return values
- Lambda and operator expressions
- Tool tip values

To enable inline debugging, in the Debug window click the Settings icon and select the check box for **Show Values In Editor**. (Android Studio, 2016)

MEMORY AND CPU MONITOR

Android Studio provides a memory and CPU monitor view so you can more easily monitor your app's performance and memory usage to track CPU usage, find deallocated objects, locate memory leaks, and track the amount of memory the connected device is using. With your app running on a device or emulator, click the **Android** tab in the lower left corner of the runtime window to launch the Android runtime window. Click the **Memory** or **CPU** tab. (Android Studio, 2016)

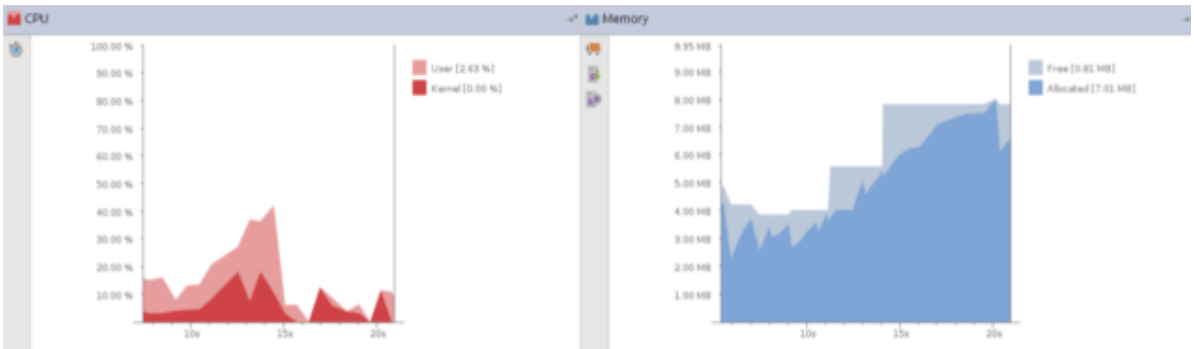


Figure 10: Monitor Memory and CPU Usage

HEAP DUMP

When you're monitoring memory usage in Android Studio you can, at the same time, initiate garbage collection and dump the Java heap to a heap snapshot in an Android-specific HPROF binary format file. The HPROF viewer displays classes, instances of each class, and a reference tree to help you track memory usage and find memory leaks. (Android Studio, 2016)

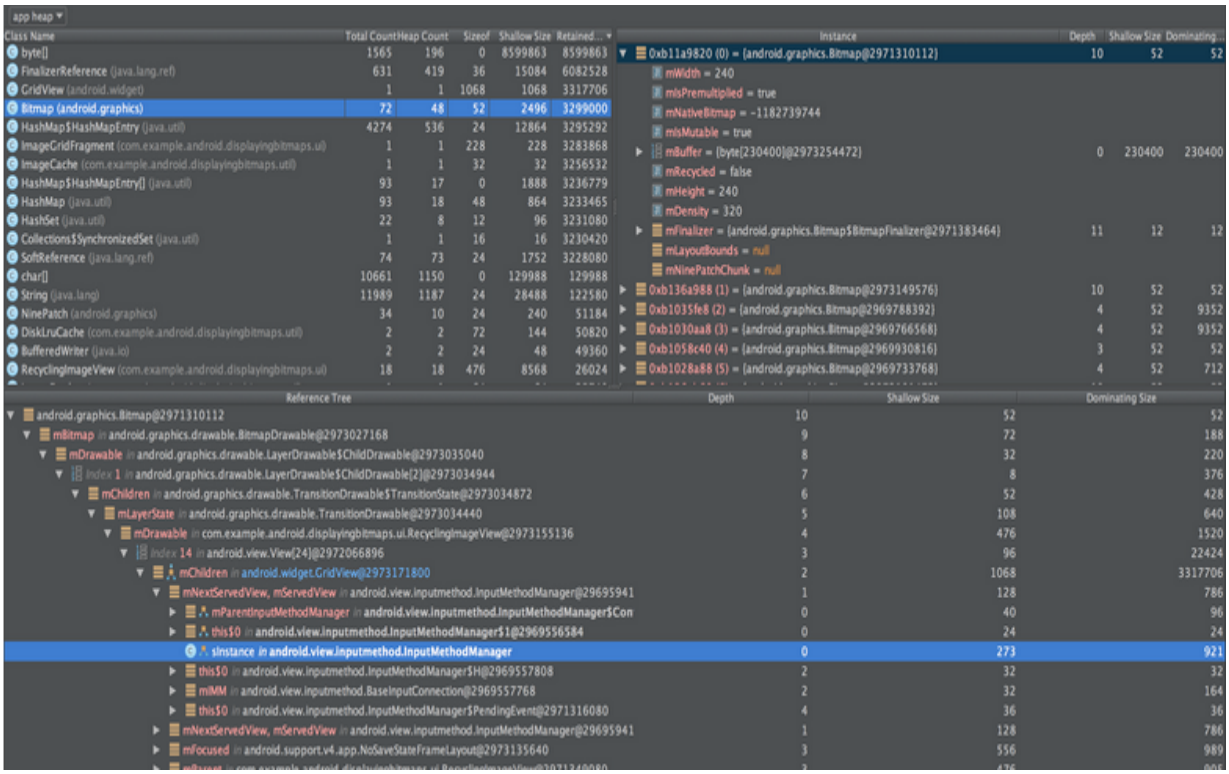


Figure 11: HPROF viewer with Heap Dump

To create a snapshot of the Android app heap memory, click the Dump Java Heap icon ( ) in the Memory Monitor. Android Studio creates the heap snapshot file with the filename `Snapshot-yyyy.mm.dd-hh.mm.ss.hprof` in the *Captures* tab. Double-click the heap snapshot file to open the *HPROF* viewer. (Android Studio, 2016)

To convert a heap dump to standard HPROF format in Android Studio, right-click a heap snapshot in the *Captures* view and select **Export to *standard.hprof.*** (Android Studio, 2016)

ALLOCATION TRACKER

Android Studio allows you to track memory allocation as it monitors memory use. Tracking memory allocation allows you to monitor where objects are being allocated when you

perform certain actions. Knowing these allocations enables you to adjust the method calls related to those actions to optimize your app's performance and memory use. (Android Studio, 2016)

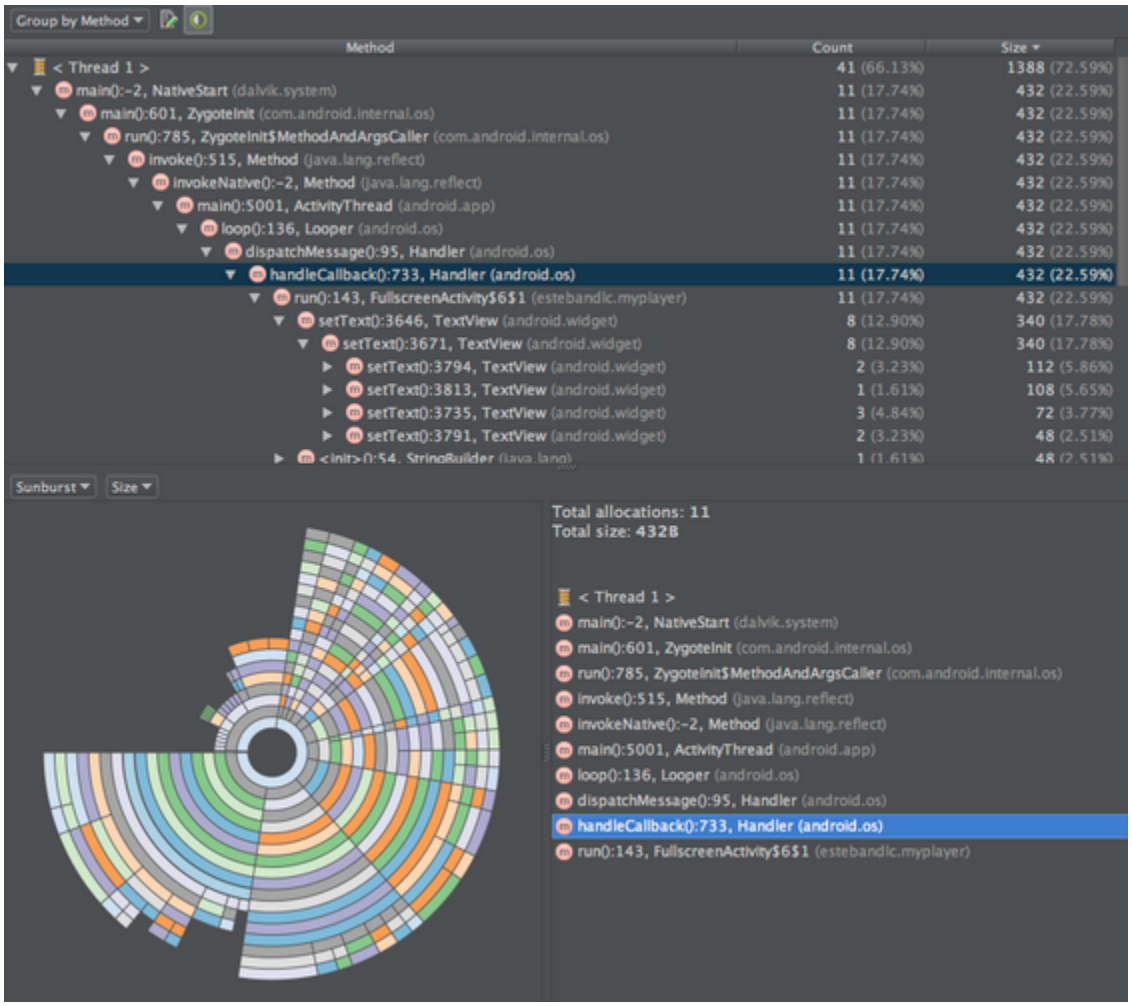


Figure 12: Allocation Tracker

DATA FILE ACCESS

The Android SDK tools, such as Systrace, logcat, and Traceview, generate performance and debugging data for detailed app analysis. (Android Studio, 2016)

To view the available generated data files, click **Captures** in the left corner of the runtime window. In the list of the generated files, double-click a file to view the data. Right-click any .hprof files to convert them to a standard .hprof file format. (Android Studio, 2016)

CODE INSPECTIONS

In Android Studio, the configured lint and other IDE inspections run automatically whenever you compile your program. (Android Studio, 2016)

Android Studio enables several lint checks to ensure:

- Cipher.getInstance() is used with safe values
- In custom Views, the associated declare-styleable for the custom view uses the same base name as the class name
- Security check for fragment injection
- Where ever property assignment no longer works as expected
- Gradle plugin version is compatible with the SDK
- Right to left validation
- Required API version
- many others

With Android Studio, you can also run lint inspections for a specific build variant, or for all build variants. You can configure the lint inspections that run by adding a lintOptions property to the Android settings in the build.gradle file. (Android Studio, 2016)

You can also manage inspection profiles and configure inspections within Android Studio. Choose **File > Settings >**, expand the **Editor** options, and select **Inspections**. The *Inspection Configuration* page appears with the supported inspections. (Android Studio, 2016)

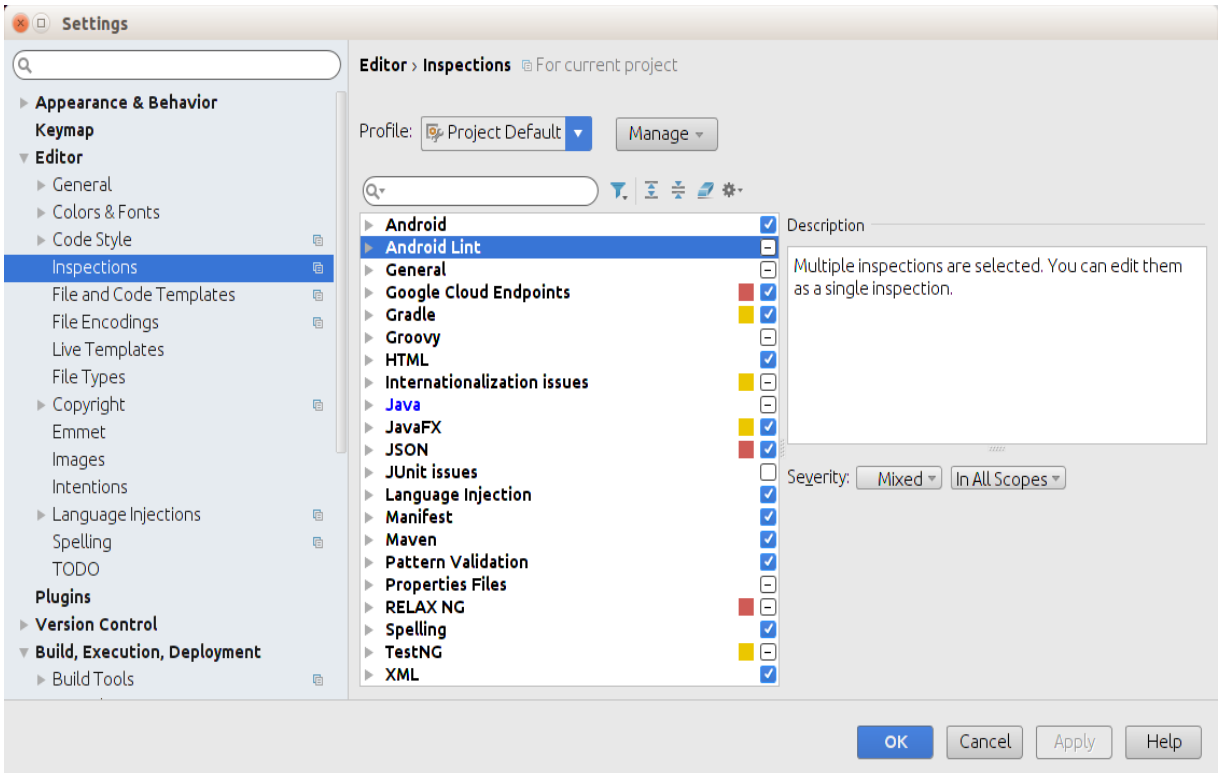


Figure 13: Inspections Configurations

To manually run inspections in Android Studio, choose **Analyze > Inspect Code**. The *Inspections Scope* dialog appears so you can specify the desired inspection profile and scope. (Android Studio, 2016)

ANNOTATIONS IN ANDROID STUDIO

Android Studio supports annotations for variables, parameters, and return values to help you catch bugs, such as null pointer exceptions and resource type conflicts. The Android SDK Manager packages the Support-Annotations library in the Android Support Repository for use with Android Studio. Android Studio validates the configured annotations during code inspection. (Android Studio, 2016)

To add annotations to your code in Android Studio, first add a dependency for the Support-Annotations library:

1. Select **File > Project Structure**.
2. In the *Project Structure* dialog, select the desired module, click the **Dependencies** tab.
3. Click the + icon to include a **Library dependency**.
4. In the *Choose Library Dependency* dialog, select support-annotations and click **Ok**.

The build.gradle file is updated with the support-annotations dependency. (Android Studio, 2016)

Inferring Nullability

A nullability analysis scans the contracts throughout the method hierarchies in your code to detect:

- Calling methods that can return null
- Methods that should not return null
- Variables, such as fields, local variables, and parameters, that can be null
- Variables, such as fields, local variables, and parameters, that cannot hold a null value

The analysis then automatically inserts the appropriate null annotations in the detected locations.

To run a nullability analysis in Android Studio, select the **Analyze > Infer Nullity** menu option. Android Studio inserts the Android @Nullable and @NonNull annotations in detected locations in your code. After running a null analysis, it's good practice to verify the injected annotations. (Android Studio, 2016)

Validating annotations

You can also manually add nullability, resource, and enumerated annotations throughout your code to perform validations for a variety of reference values, such as R.string resources, Drawable resources, Color resources, and enumerated constants. (Android Studio, 2016)

Run **Analyze > Inspect Code** to validate the configured annotations.

For a complete list of the supported annotations, either use the auto-complete feature to display the available options for the import android.support.annotation statement or view the contents of the Support-Annotations library. (Android Studio, 2016)

LOG MESSAGES

When you build and run your app with Android Studio, you can view **ADB** and device log messages (logcat) by clicking **Android** at the bottom of the window. (Android Studio, 2016)

If you want to debug your app with the **Android Debug Monitor**, you can launch it by clicking **Monitor** in the toolbar. The Debug Monitor is where you can find the complete set of DDMS tools for profiling your app, controlling device behaviors, and more. It also includes the Hierarchy Viewer tools to help optimize your layouts. (Android Studio, 2016)

2.1 FUNCTIONAL REQUIREMENTS

Functional requirements are services that are going to be provided to the user. The main type of user is the regular user. Administrators are not needed in this system because it should be self-sufficient by the regular user/customer

2.1.1 USER USE CASE #1: CREATE ACCOUNT

Brief Description:

1. User goes to the New User login screen.
2. The user enters desired Username and Password into form boxes.
3. The user will be redirected to the User Login Screen if successful.

2.1.2 USER USE CASE #2: LOGIN

Brief Description:

1. Enter Username and Password into the Login Screen
2. Click Send
3. User will be presented with the homepage.

2.1.3 User Use Case #3: Logout

Brief Description:

1. Click Exit or Switch Users on the Home screen.

2.1.4 User Use Case #4: Scan Device for Malwares

Brief Description:

1. Users Clicks Scanner Button
2. Scanner automatically checks and cleans the smartphone from all malicious software and activities.

2.1.5 User Use Case #5: Check the Battery Health

Brief Description:

1. Users Clicks Battery Info Button
2. System automatically displays the Battery's Health, Level, Plugged, Present, Status, Technology, Temperature, Voltage.

2.1.6 User Use Case #6: Memory Booster

Brief Description:

1. Users Clicks Memory Booster Button
2. System automatically checks and displays all the Apps that are significantly using the memory.
3. User Clicks the Kill All Task Button.

4. System safely kills/stop all the memory taking tasks.

2.2 User Interface Specification

The users using this system need only minimum computer knowledge. The system is designed to be an easy to use application for anyone that uses the any Internet Browser and Android Operating System on a mobile device.

2.3 Non-functional Requirements

This section of the document will cover those requirements that are not the functional requirements of the system. Such requirements are the constraints of the system as it pertains to performance relative to a given system.

2.3.1 Product Non-functional Requirements

The performance of the application will rely heavily on the system on which the system is used. Older computers and Android phones may take long in processing the complete scan of files and folders.

2.3.2 Process Non-functional Requirements

Currently, the system is being developed using the Android Studio. This Application is built with the Java Programming Language and Extensible Markup Language (XML). Both Android and the Java Application are constructed using Java. JDBC will be used to interact with the SQLite database.

2.4 System Study

2.4.1 FEASIBILITY STUDY

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the

feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

2.4.2 Economical Feasibility

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

2.4.3 Technical Feasibility

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

2.4.4 Social Feasibility

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the

users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

Chapter 3:

DESIGN DESCRIPTION

3. DESIGN DESCRIPTION

3.1 SYSTEM DESIGN

3.1.1 ACTIVITY LIFE CYCLE

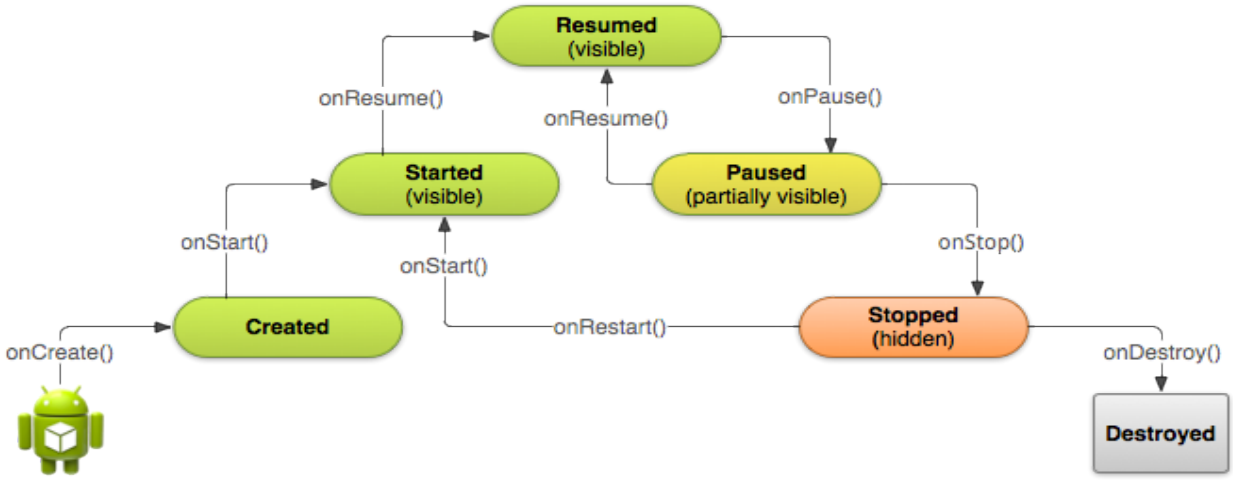


Figure 14: Activity Life Cycle

3.1.2 DATA FLOW DIAGRAM

1. A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.
2. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
3. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by

the process, an external entity that interacts with the system and the information flows in the system.

4. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
5. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

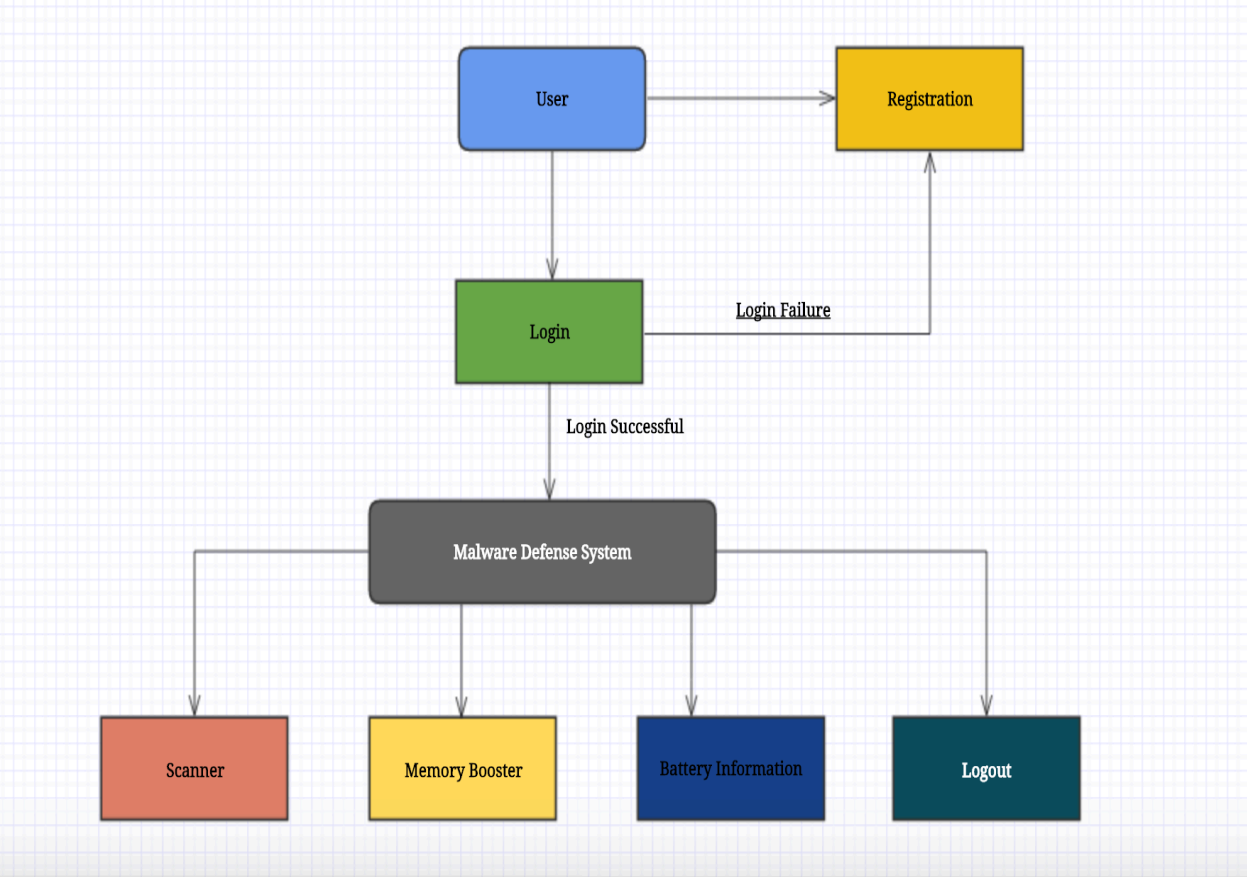


Figure 15: Data Flow Diagram

3.2 UML DIAGRAMS

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

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

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

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

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

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.

4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of Object Oriented tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

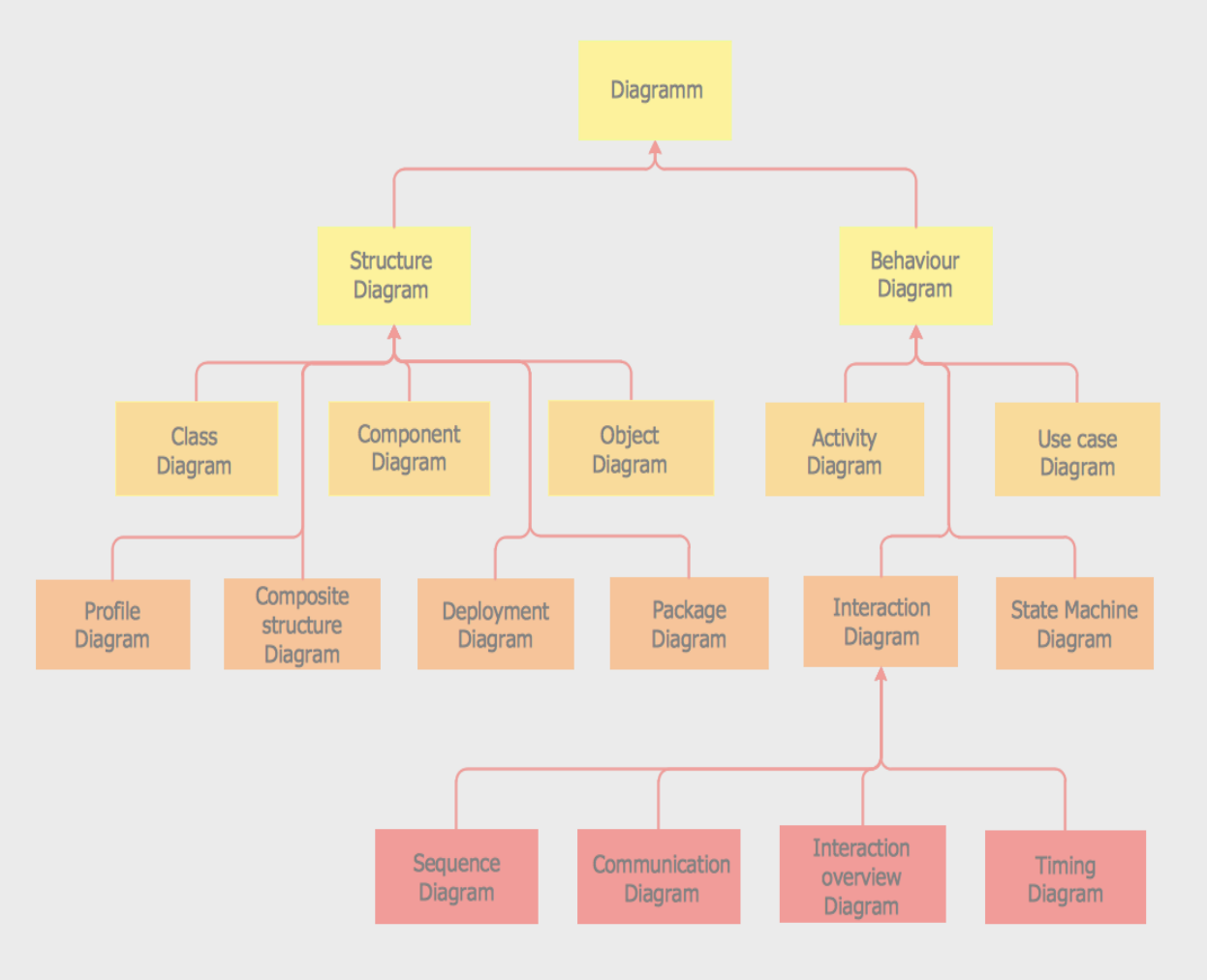


Figure 16: UML Diagram Types

3.2.1 USE CASE DIAGRAM:

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

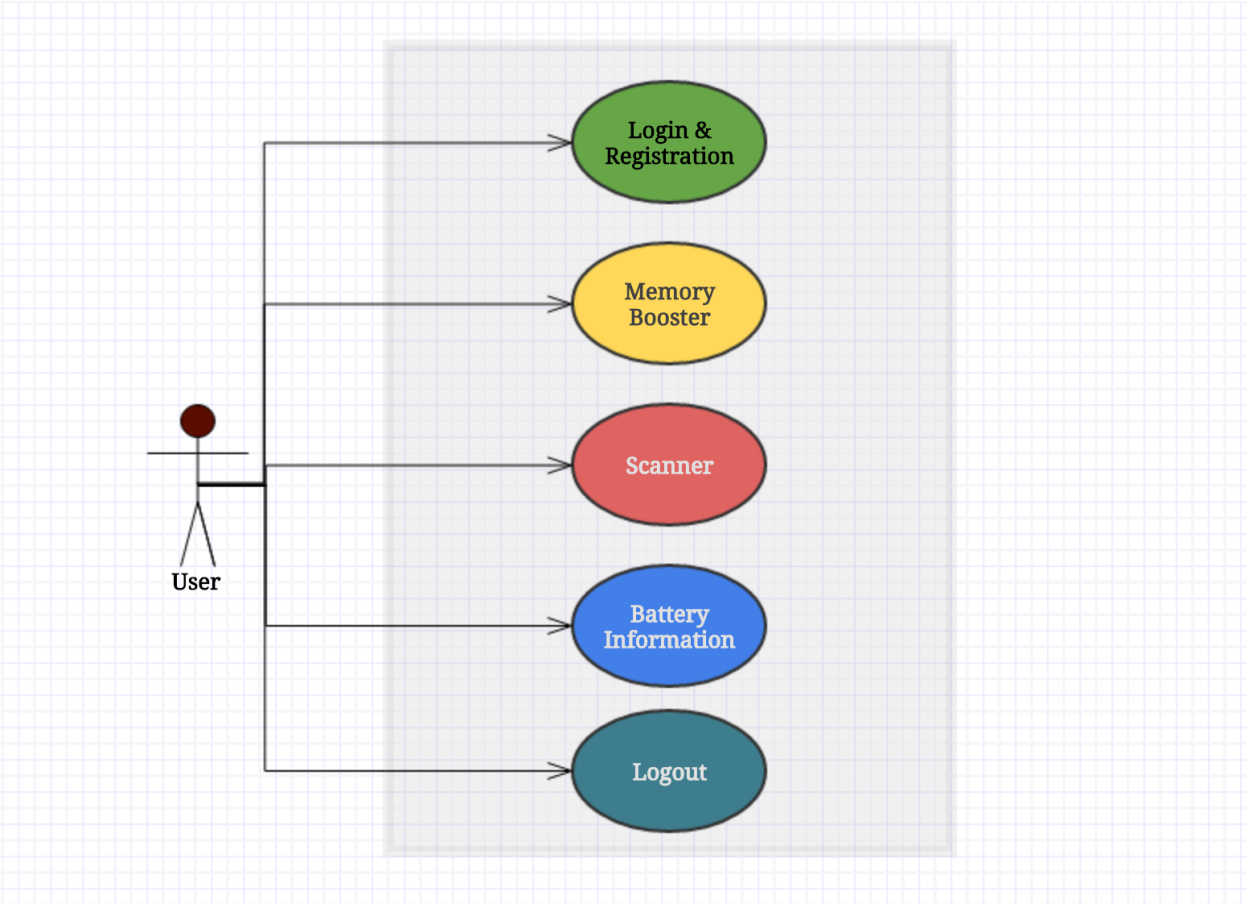


Figure 17: Use Case Diagram

USE CASE DESCRIPTION:

Use Case Section	Comment
ID	UC001
Use Case Name	Registration
Goals	User registers an account successfully
Actor	User
Precondition(s)	None
Trigger	User clicks register button
Post-condition	The system creates new account successfully
Main scenario (1)	User Clicks Register Button. The system displays Registration Form. User enters data and click submit. The system creates new account.
Extensions	If user enters invalid data, the system displays error message.

Table 5: Use Case Description (Registration)

Use Case Section	Comment
ID	UC002
Use Case Name	Login
Goals	User Logs-In successfully
Actor	User
Precondition(s)	User Must be Registered

Trigger	User clicks Login button
Post-condition	The system Logs-In to account successfully
Main scenario (1)	User Fill the Input Fields by User-ID and Password Clicks Login Button. The system displays Home Page.
Main scenario (2)	User Click Social Media Login Button Fills the Social Media Login Info System Displays the Home Page
Extensions	If user enters invalid data, the system displays error message.

Table 6: Use Case Description (Login)

Use Case Section	Comment
ID	UC003
Use Case Name	Memory Booster
Goals	Kill all the task using significant memory
Actor	User
Precondition(s)	User must be Logged-In
Trigger	User clicks Memory Booster Button
Post-condition	The system Kills all the task using significant memory
Main scenario	User Clicks Memory Booster Button. The system displays all the Apps using significant memory. User clicks clear memory.

	The system clears the memory.
Extensions	None

Table 7: Use Case Description (Memory Booster)

Use Case Section	Comment
ID	UC004
Use Case Name	Scanner
Goals	Check and Clear all Potential Threats
Actor	User
Precondition(s)	User must be Logged-In
Trigger	User clicks Scanner Button
Post-condition	The system Clear all Potential Threats
Main scenario	User Clicks Scanner Button. The system checks for all the potentially malicious Apps. User clicks resolve all button. The system clears and delete all the malicious data.
Extensions	None

Table 8: Use Case Description (Scanner)

Use Case Section	Comment
ID	UC005
Use Case Name	Battery Information
Goals	Show user the phone battery status

Actor	User
Precondition(s)	User must be Logged-In
Trigger	User clicks Battery Info Button
Post-condition	The system displays the Battery Information
Main scenario	User Clicks Battery Info Button. The system displays all information related to phones battery.
Extensions	None

Table 9: Use Case Description (Battery Information)

3.2.2 Class Diagram:

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

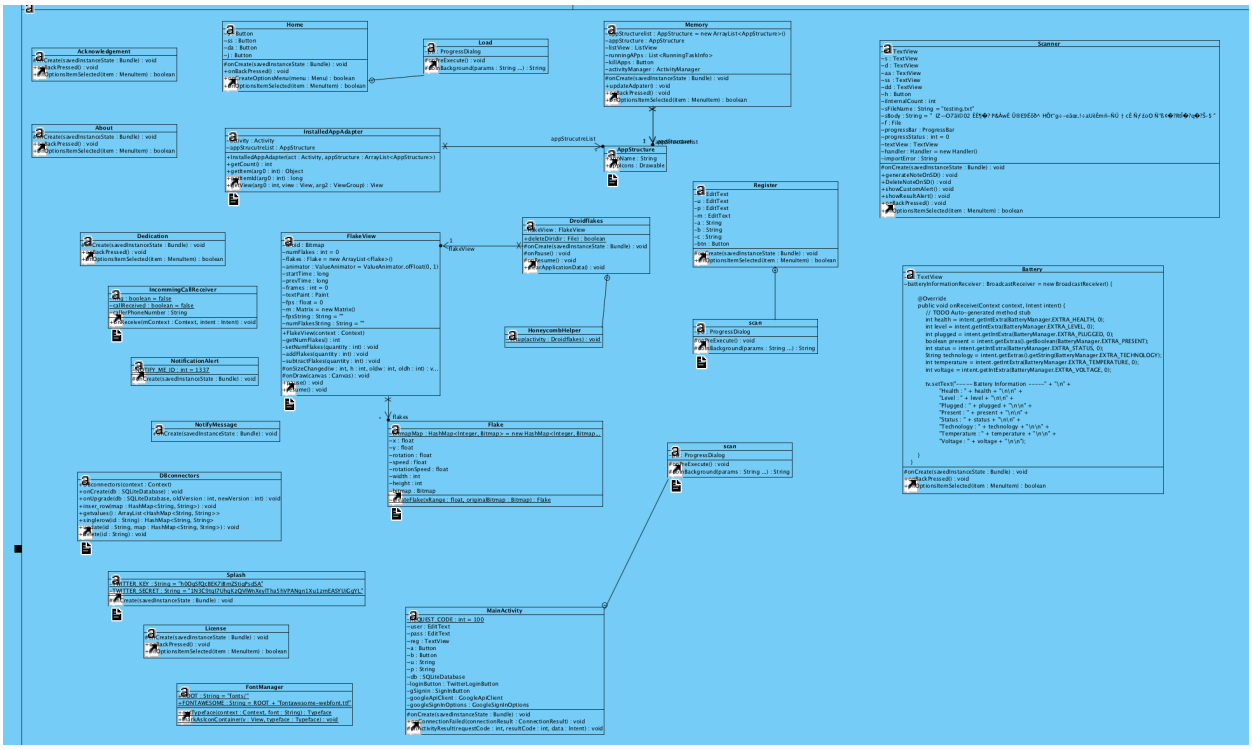


Figure 18: Class Diagram

3.2.3 Sequence Diagram:

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

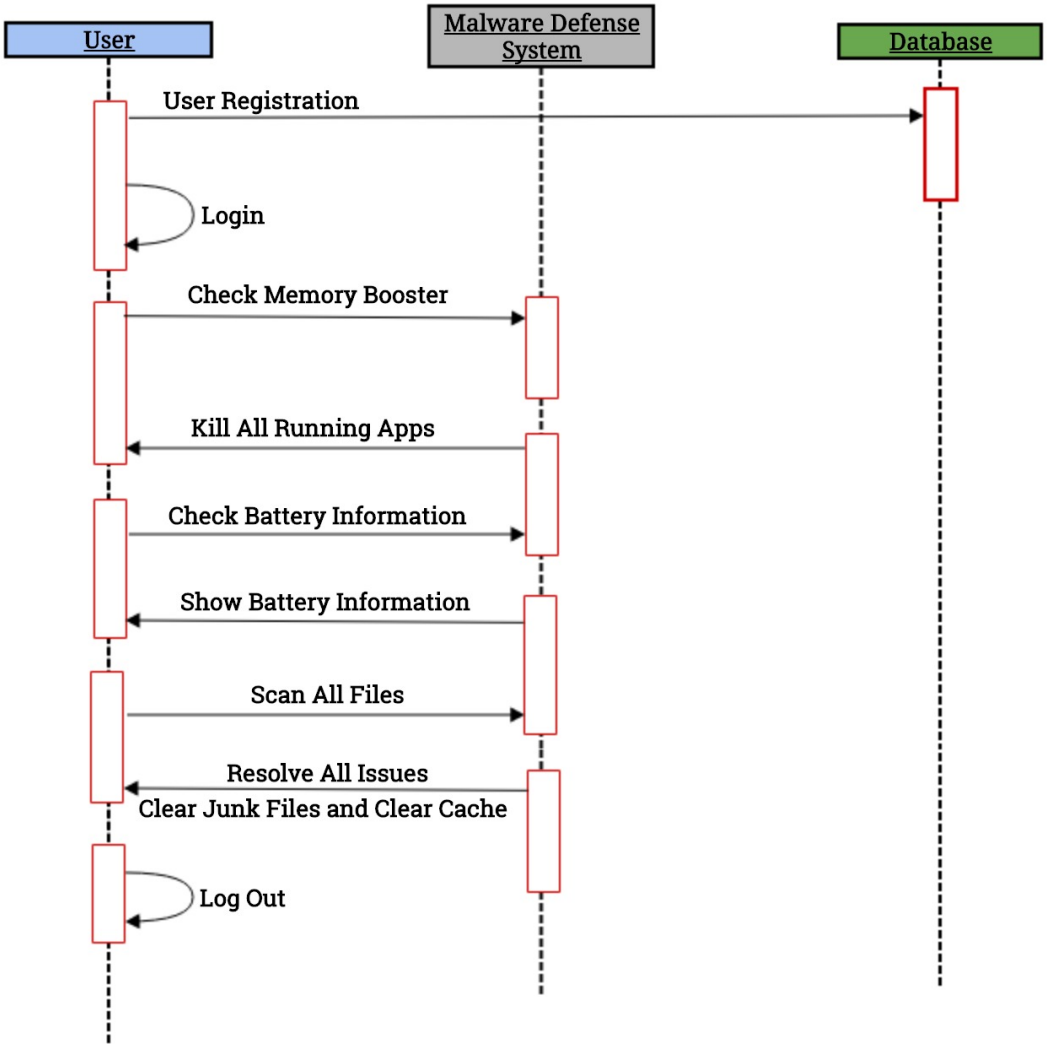


Figure 19: Sequence Diagram

3.2.4 Collaboration Diagram

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

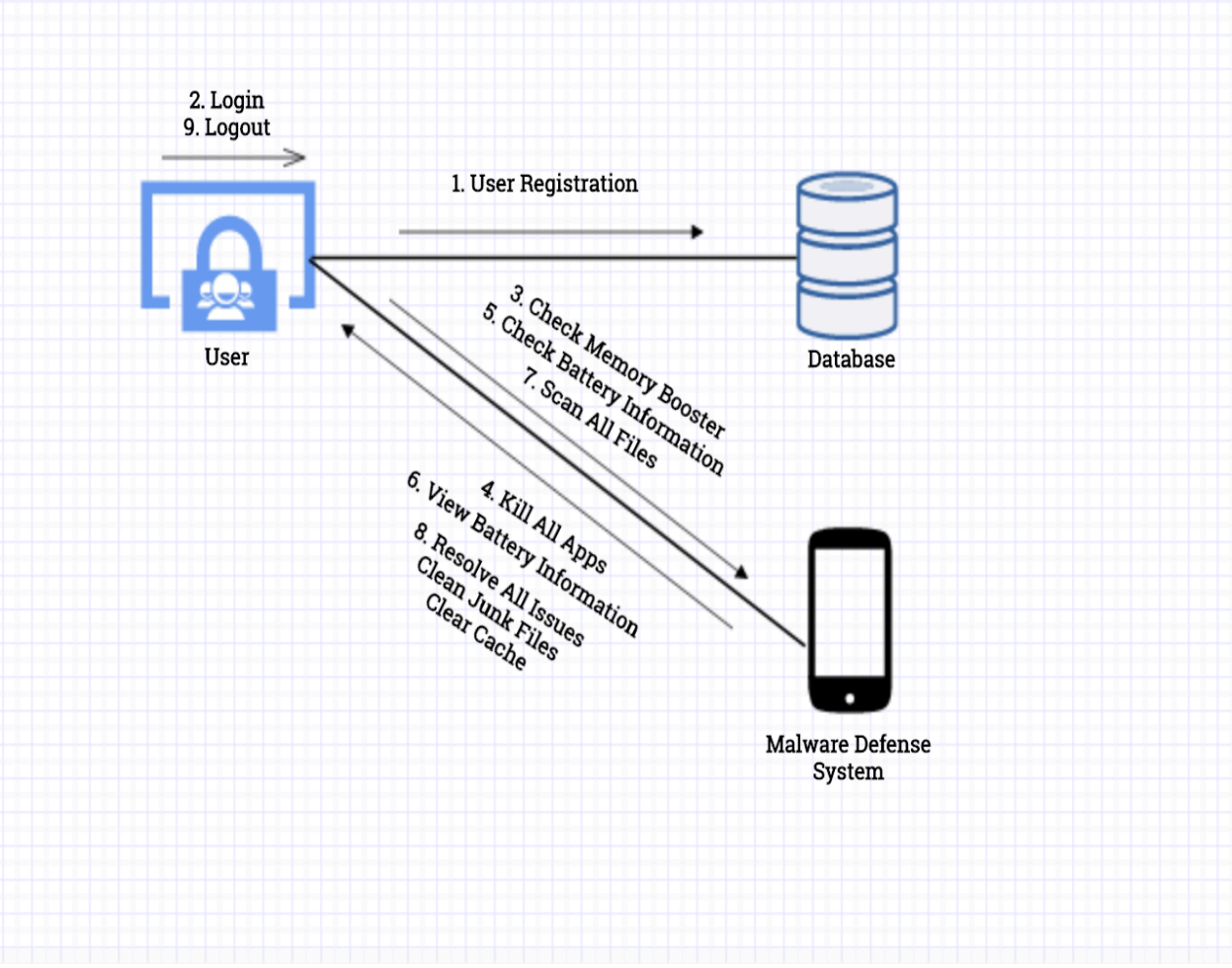


Figure 20: Collaboration Diagram

3.2.5 System Architecture:

A **system architecture** or **systems architecture** is the conceptual model that defines the structure, behavior and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

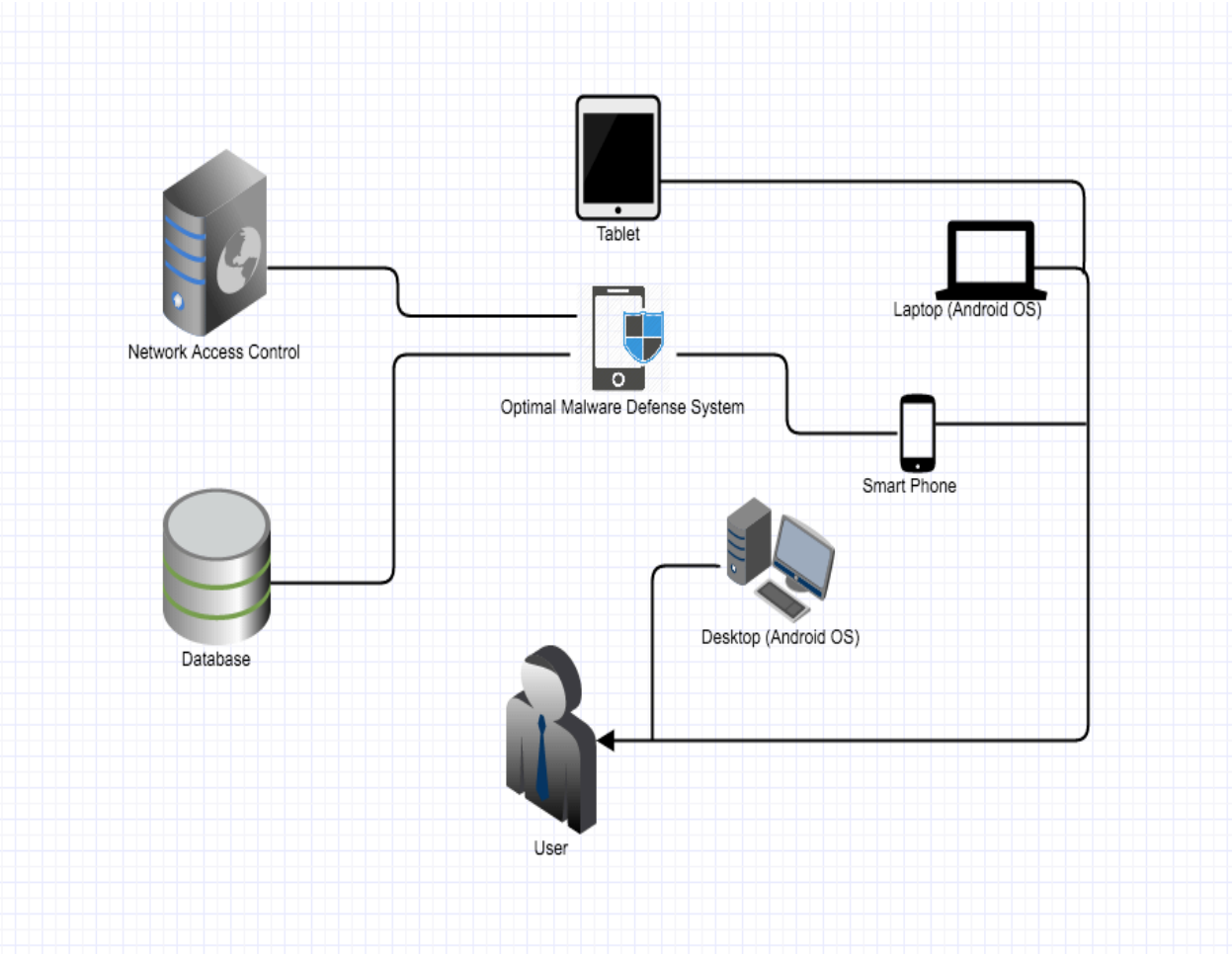


Figure 21: System Architecture Diagram

Chapter 4:

IMPLEMENTATION

4. IMPLEMENTATION

4.1 Tools & Technology Used

4.1.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Android platform development.

It was announced on May 16, 2013 at the Google I/O conference. Android Studio is freely available under the Apache License 2.0.

It is based on JetBrains' IntelliJ IDEA software, Android Studio is designed specifically for Android development.[7] It is available for download on Windows, Mac OS X and Linux, and replaced Eclipse Android Development Tools(ADT) as Google's primary IDE for native Android application development.

4.1.2 Extensible Markup Language (XML)

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format which is both human-readable and machine-readable. It is defined by the World Wide Web Consortium (W3C) XML 1.0 Specifications and by several other related specifications, all of which are free open standards.

The design goals of XML emphasize simplicity, generality and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures such as those used in web services.

4.1.3 Java Programming Language

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as

possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine(JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

4.1.4 SQLITE DATABASE

SQLite is a relational database management system contained in a C programming library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

SQLite is ACID-compliant and implements most of the SQL standard, using a dynamically and weakly typed SQL syntax that does not guarantee the domain integrity.

SQLite is a popular choice as embedded database software for local/client storage in application software such as web browsers. It is arguably the most widely deployed database engine, as it is used today by several widespread browsers, operating systems, and embedded systems, among others. SQLite has bindings to many programming languages.

4.2 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to

read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

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

4.2.1 OBJECTIVES

1. Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

4.3 OUTPUT DESIGN

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

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

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

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

4.4 User Interface (GUI) Design

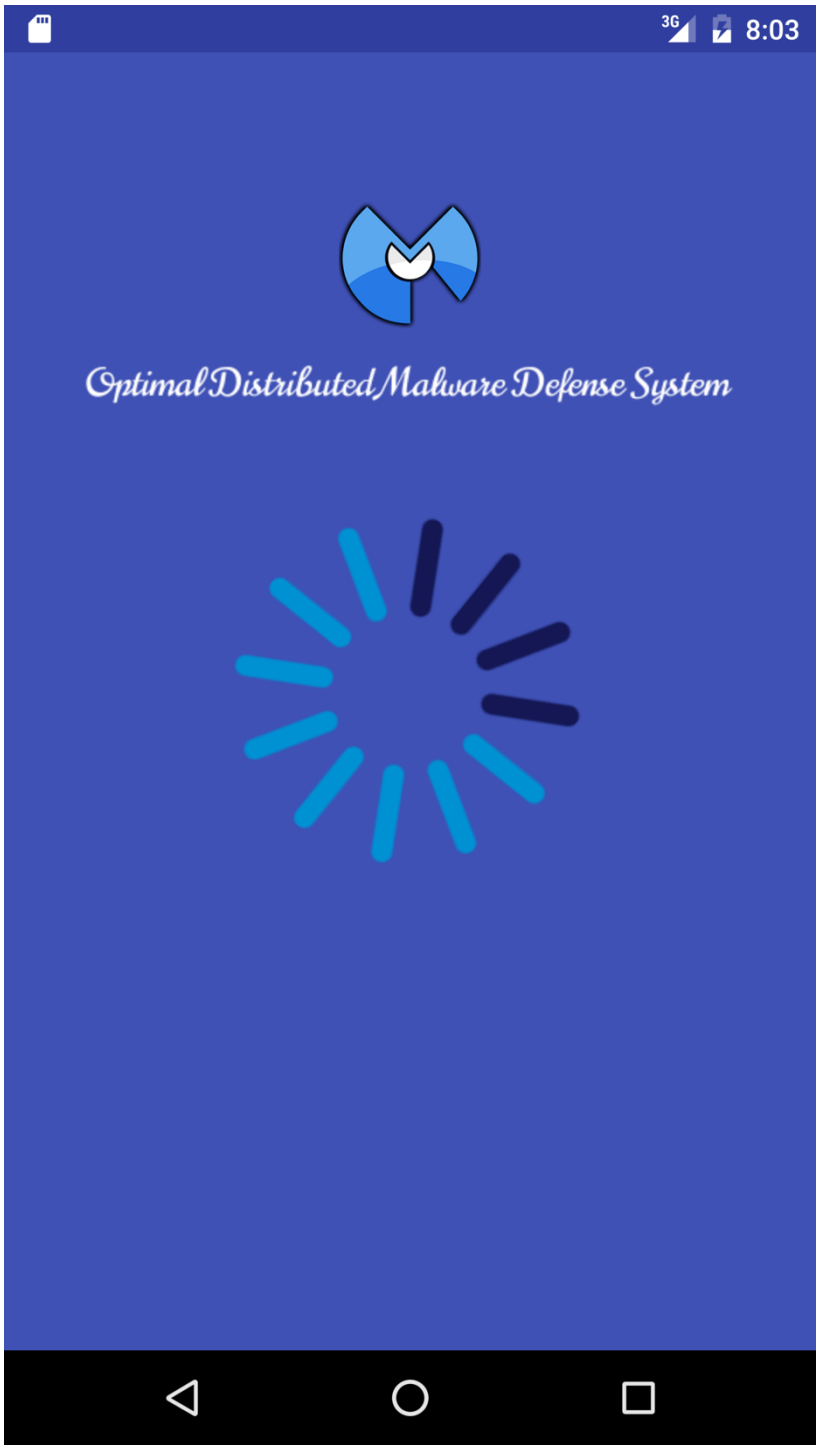


Figure 22: User Interface (Loading Window)

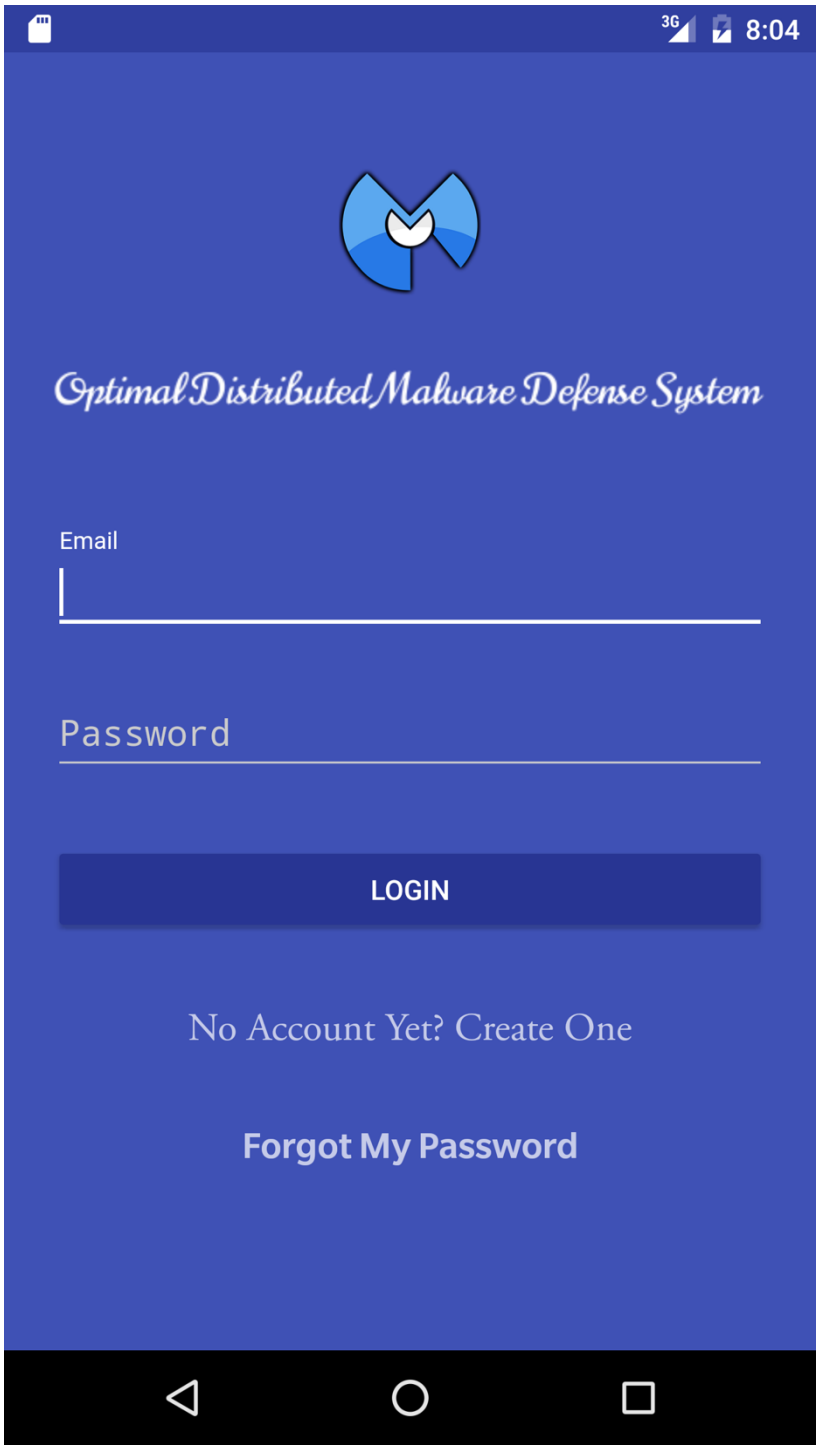


Figure 23: User Interface (Login Window)

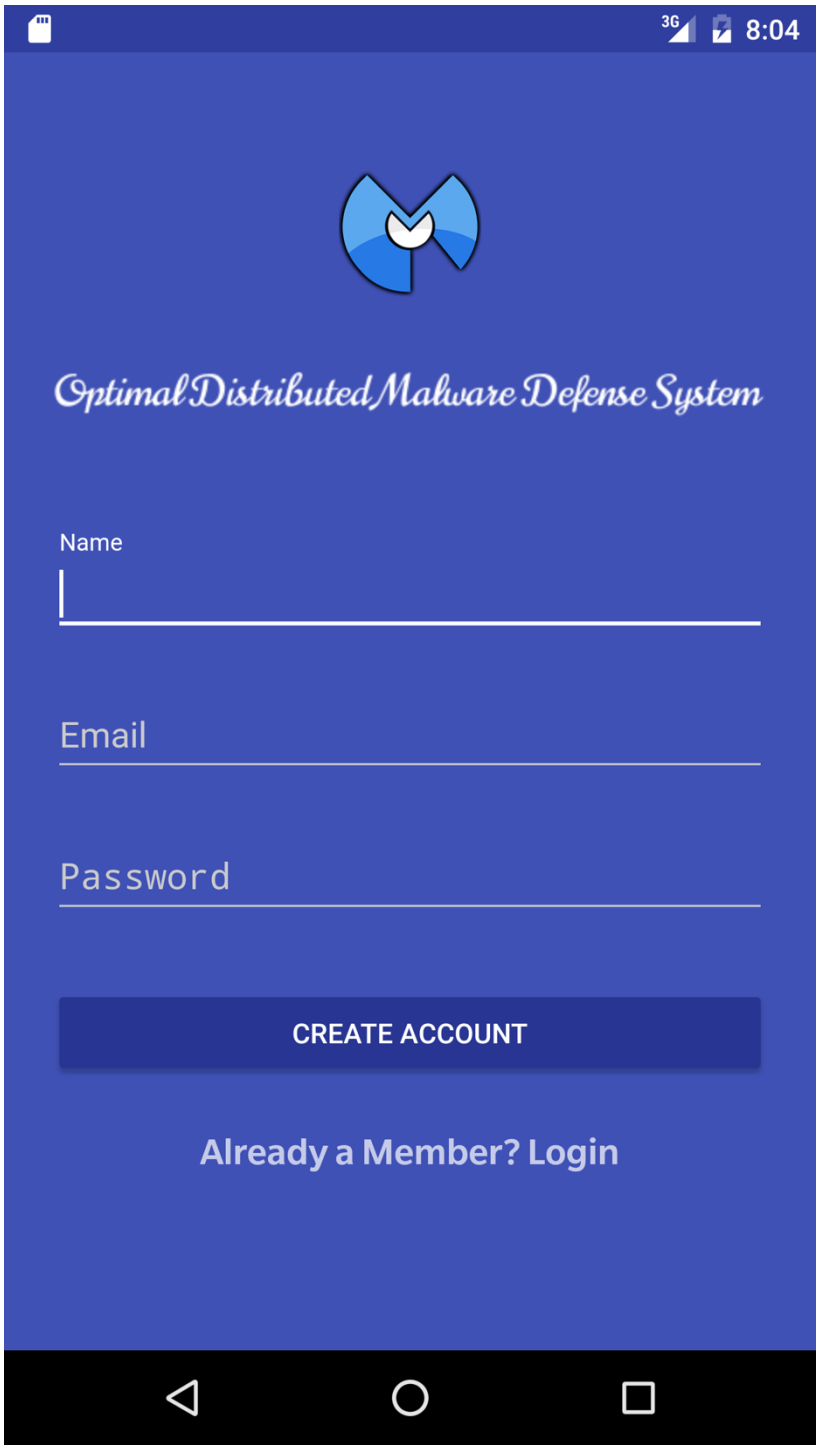


Figure 24: User Interface (Registration Window)

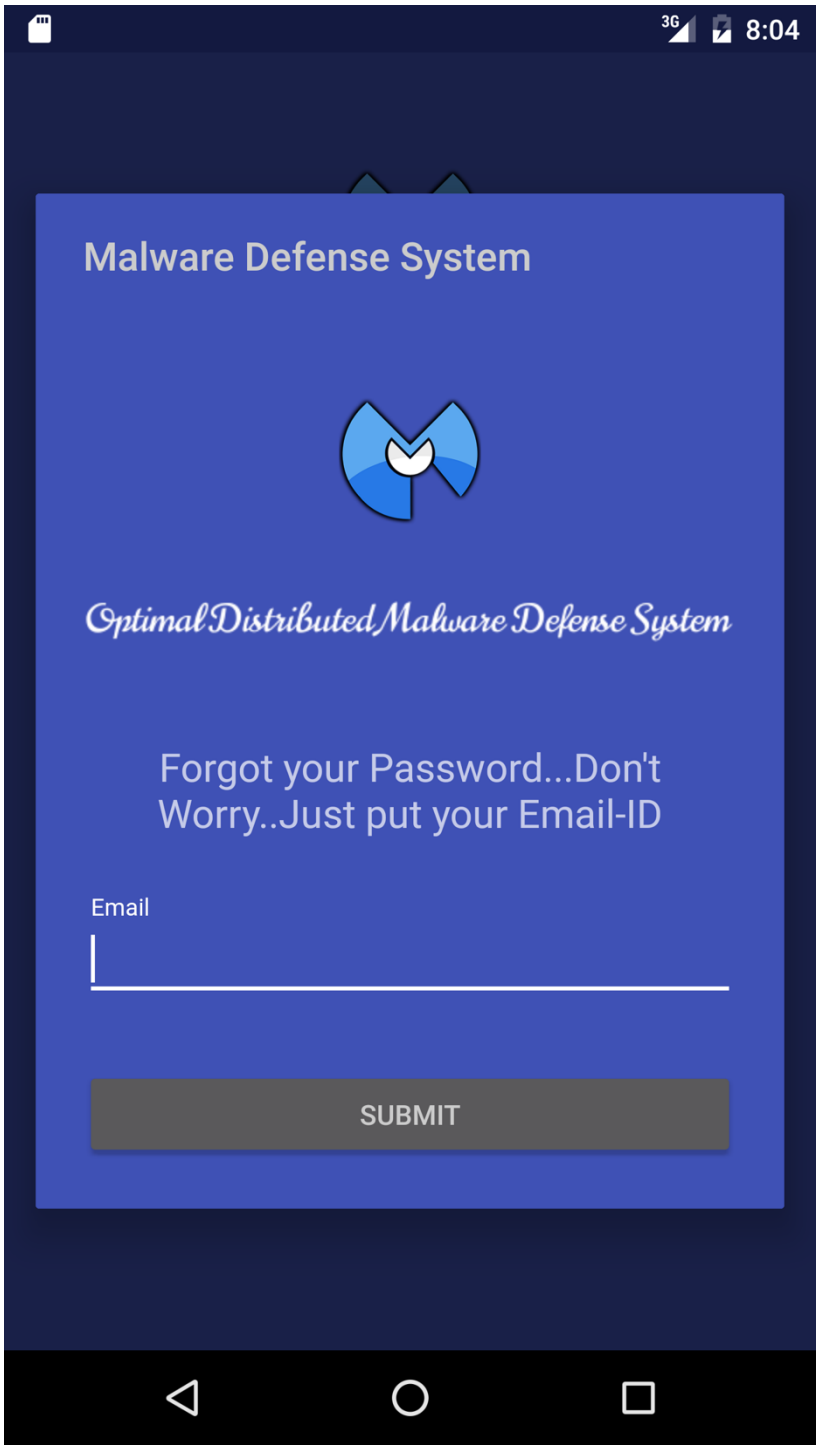


Figure 25: User Interface (Forgot Password)

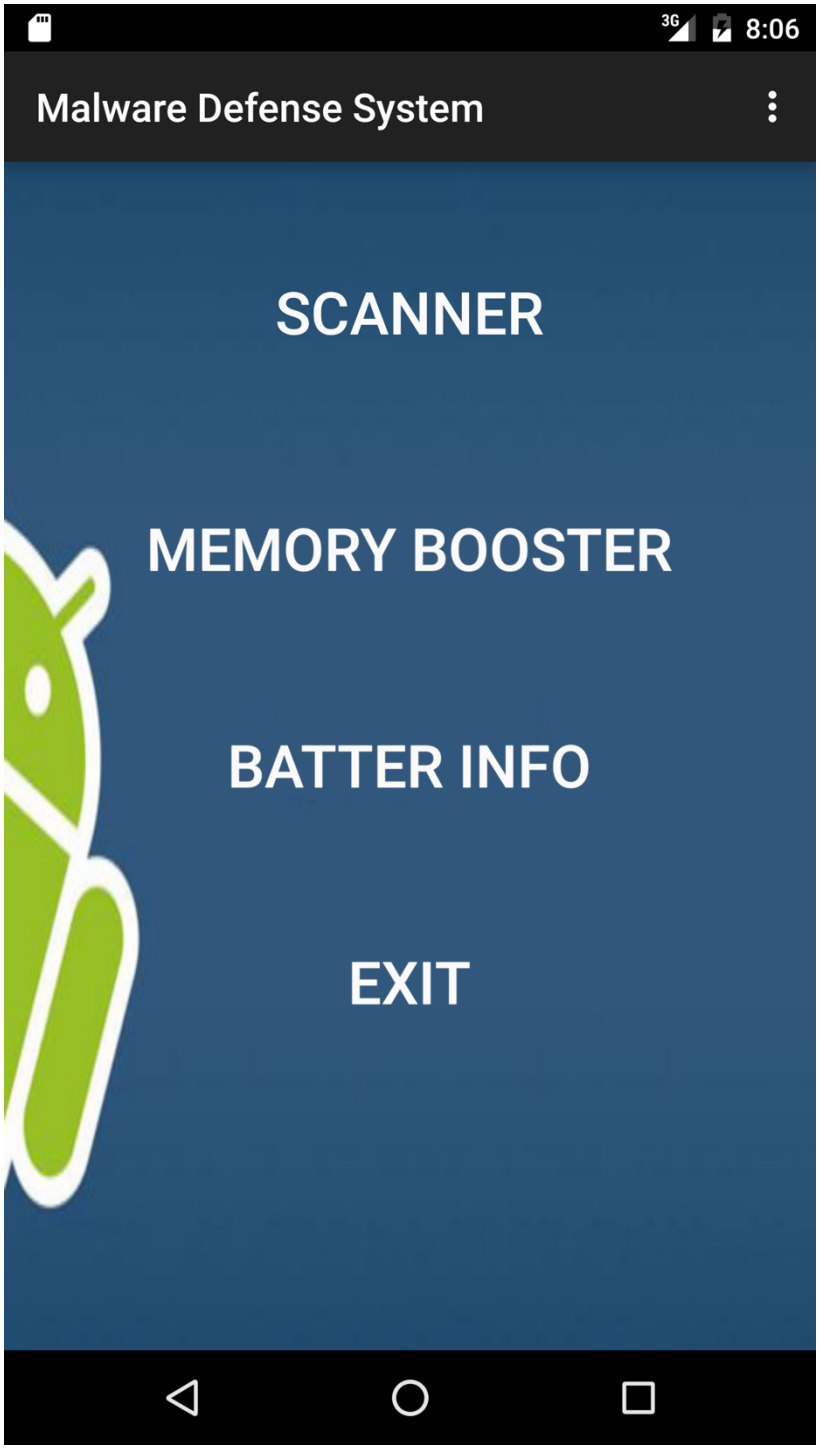


Figure 26: User Interface (Home Screen)

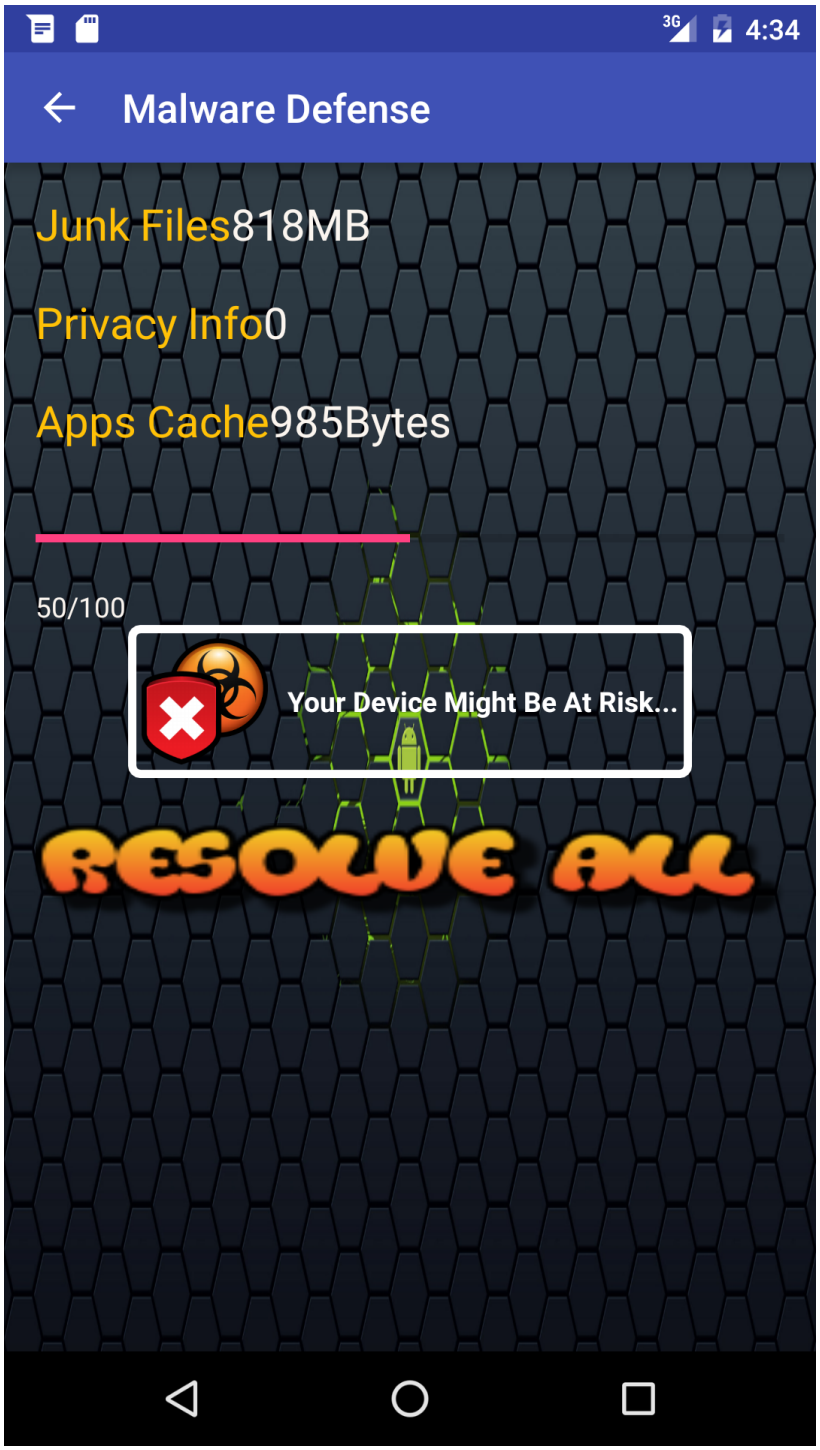


Figure 27: User Interface (Scanner Screen)

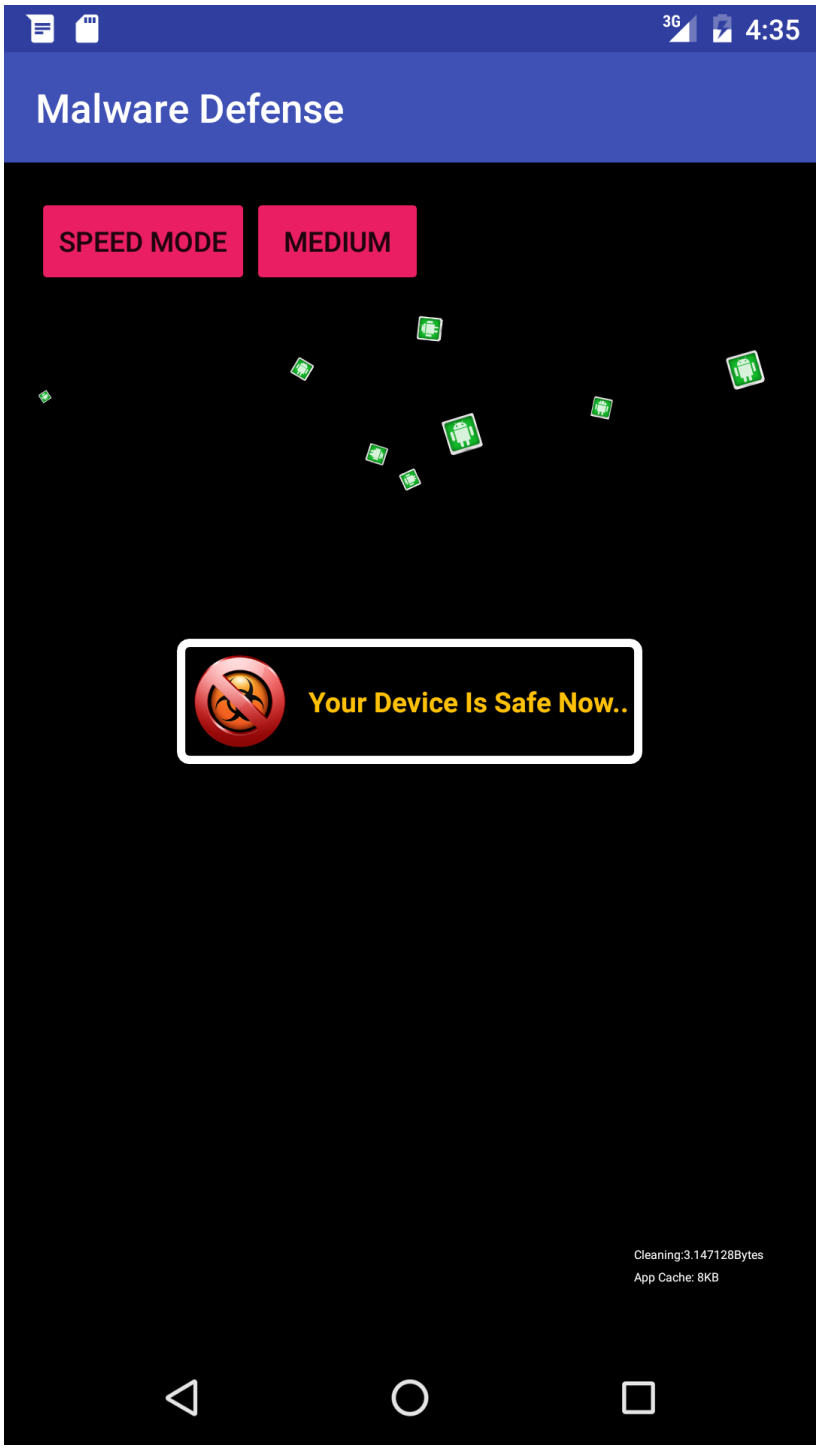


Figure 28: User Interface (Resolved Screen)

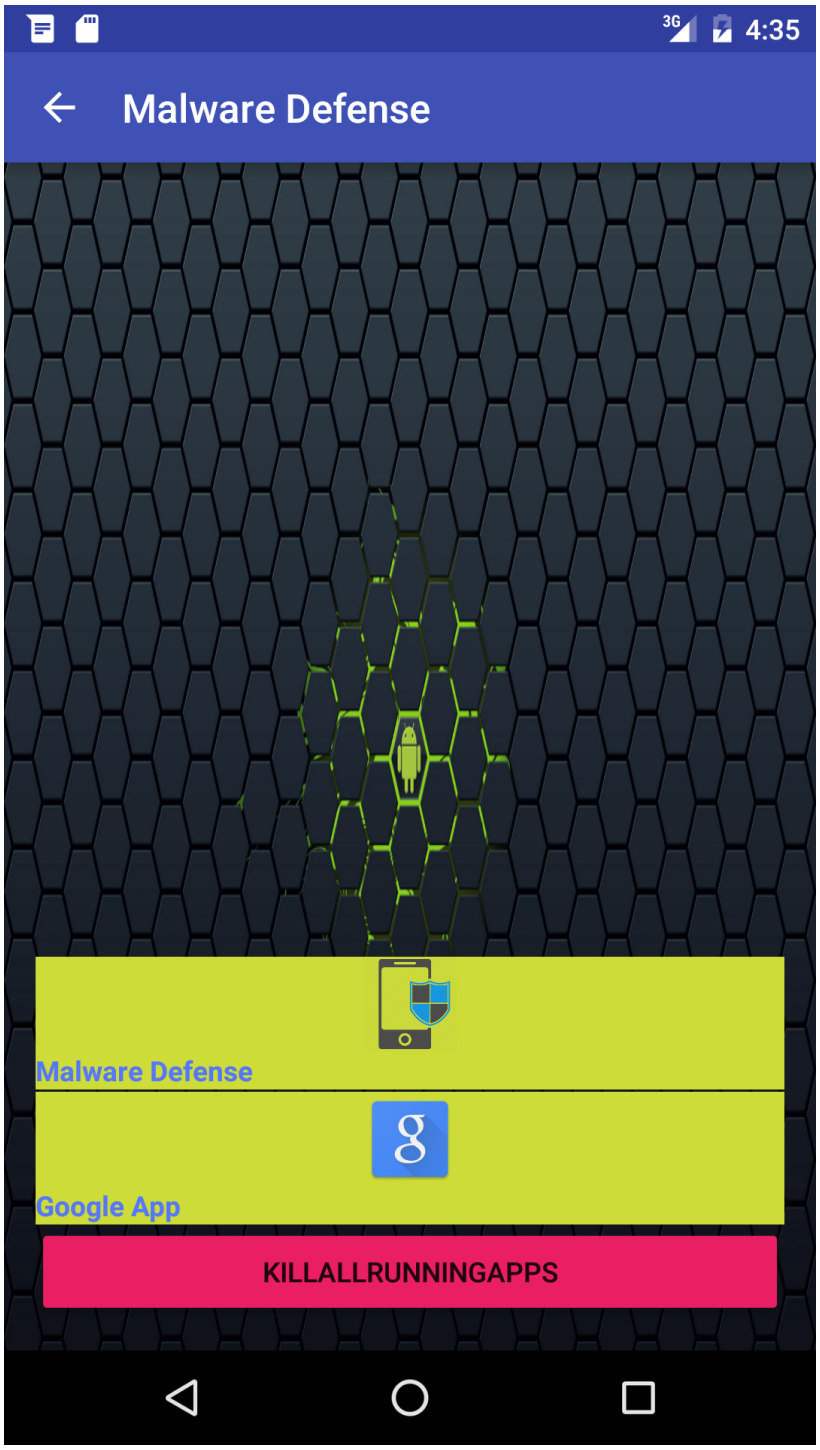


Figure 29: User Interface (Memory Booster Screen)

Figure 30: User Interface (Battery Information Screen)

4.5 MODULES

- Malware signature finder and Spreading Module
- Problem Formulation and Centralized Algorithm
- The Metropolis Sampler.
- Performance Evaluation

4.6 MODULE DESCRIPTION:

4.6.1 MALWARE SIGNATURE FINDER AND SPREADING MODEL:

In this module, malware signature will be analyzed and distributed over connected node. We consider a system of N heterogeneous wireless nodes belonging to K types (e.g., type of OS), which can be infected by K types of malware, denoted by set IK. In the defense system, we assume that there are S helpers, denoted by set SS, storing the signatures to help other nodes with detecting the malware.

4.6.2 PROBLEM FORMULATION AND CENTRALIZED ALGORITHM:

Based on the malware spreading model, we first formulate the problem, and then give a greedy algorithm to achieve the optimal signature distribution. Now, we validate the proposed malware spreading model expressed, which is based on the epidemic model for malware spreading and the fluid model in DTN. Since our model characterizes the fraction of the malware infected nodes, we simulate the malware spreading, and compare the simulation results of infected ratio with that obtained by the model. As we have claimed that this model characterizes the MMS and proximity malware spreading, we validate the malware spreading in both the proximity and MMS scenarios.

4.6.3 The Metropolis Sampler:

In this module we develop the distributed algorithm for the signature distribution problem. The designed algorithm is based on a simulated annealing technique called Metropolis sampler. In the following sections, we first describe the basic notions and the framework of Metropolis sampler, then design the distributed algorithm based on simulated annealing with the Metropolis sampler, and finally prove that the proposed algorithm converges to the optimal performance.

4.6.4 Performance Evaluation:

We present numerical results with the goal of demonstrating that our greedy algorithm for the signature distribution, denoted OPT, achieves the optimal solution and yields significant enhancement on the system welfare compared with prior heuristic algorithms. Related to the heuristic algorithms, we consider 1) Important First (IF), which uses as many helpers as possible to store the signature of the most popular malware, 2) Uniform Random (UR), where each helper randomly selects the target signatures to store, and 3) Proportional Allocation (PA), which is a heuristic policy that assigns signatures with the uniform distribution proportional to the market sharing and the weights of different malware.

Chapter 5:

TESTING & INTEGRATION

5 TESTING AND INTEGRATION

5.1 SYSTEM TESTING

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

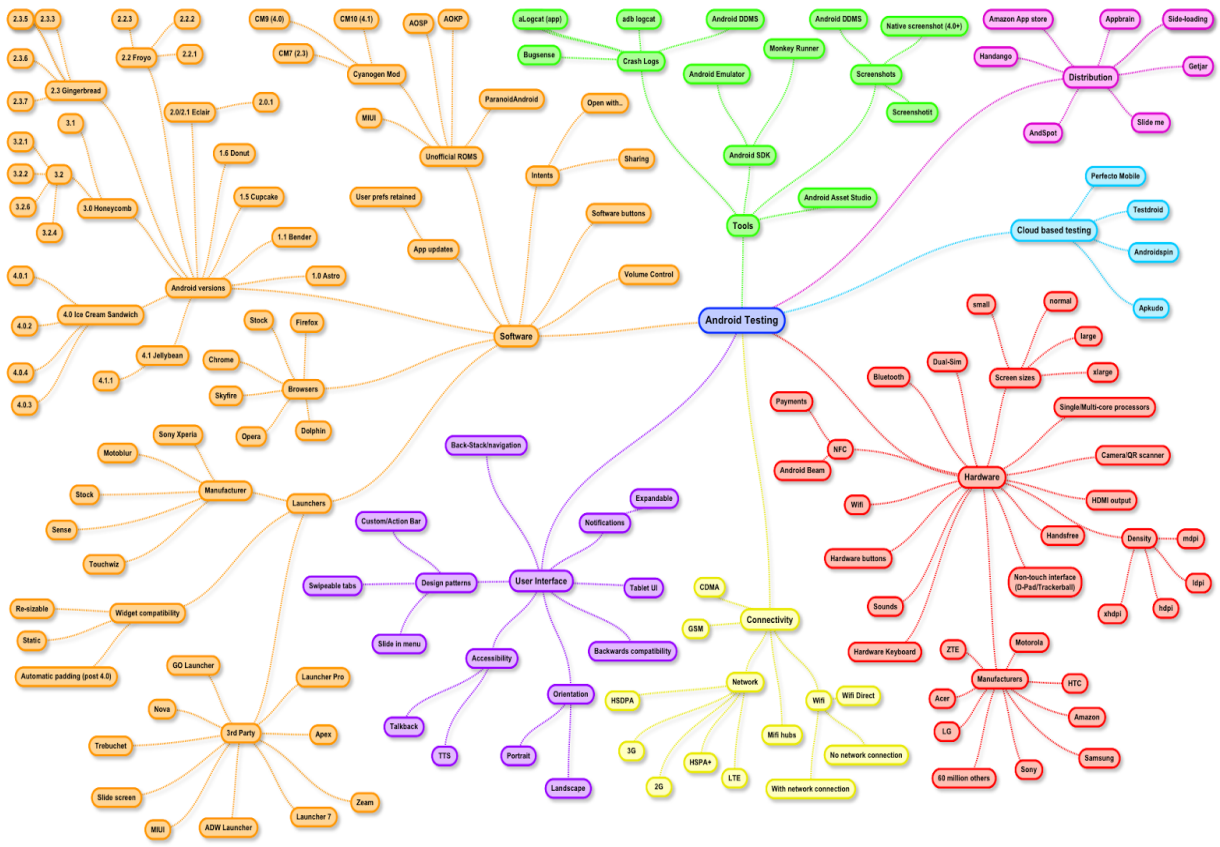


Figure 31: Android Testing

5.2 TYPES OF TESTS

5.2.1 UNIT TESTING

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

5.2.2 INTEGRATION TESTING

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

5.2.3 FUNCTIONAL TEST

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

Functional testing is centered on the following items:

Valid Input	:	Identified classes of valid input must be accepted.
Invalid Input	:	Identified classes of invalid input must be rejected.

Functions	:	Identified functions must be exercised.
Output	:	Identified classes of application outputs must be exercised.
Systems/Procedures	:	Interfacing systems or procedures must be invoked.

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

5.2.4 System Test

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

5.2.5 White Box Testing

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

5.2.6 Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software

under test is treated, as a black box, you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

5.3 Unit Testing

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

5.3.1 Test Strategy and Approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

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

Features to be tested

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

5.4 Integration Testing

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

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

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

5.5 Acceptance Testing

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

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

Chapter 6:

INSTALLATION

&

OPERATING INSTRUCTIONS

6 INSTALLATION & OPERATING INSTRUCTIONS

6.1 Installation Instruction (Users)

6.1.1 Setting Up Your Device

From your smartphone or tablet running Android 4.0 or Higher, go to Settings, scroll down to Security, and select Unknown sources. Selecting this option will allow you to install apps outside of the Google Play store. Depending on your device, you can also choose to be warned before installing harmful apps. This can be enabled by selecting the Verify apps option in the Security settings.

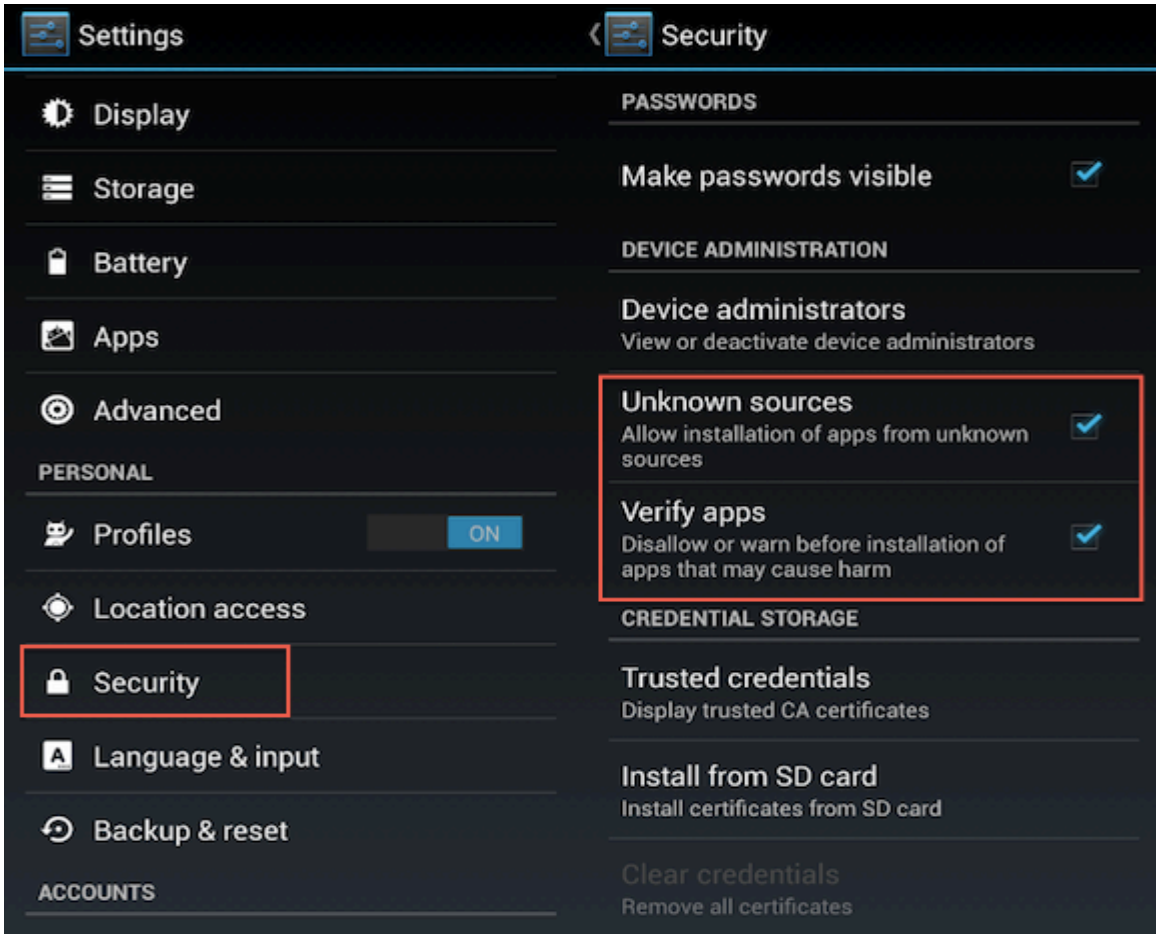


Figure 32: Setting Up Device

* On devices running an earlier version of Android, go to Settings, open the Applications option, select Unknown sources, and click OK on the popup alert.

6.2 Connecting Device to Computer

6.2.1 Connecting Android 2.0 – 3.0 (Ginger Bread – Honey Comb)

Step 1: Turn on USB debugging by

click **Menu > Setting > Applications > Development > USB debugging**

Step 2: Plug the supplied USB cable into your PC and connect it to your device. A USB icon will appear in the status bar when the device has been recognized.

Step 3: In the main screen of your Android device, use your finger to pull down the top status bar. Then touch **USB Connected**.

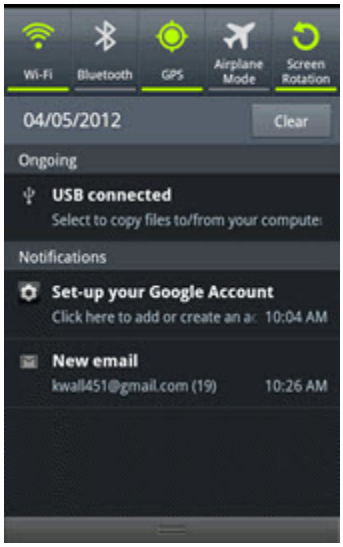


Figure 33: USB Connected (Android 2.0 – 3.0)

Step 4. Touch **Connect USB storage**.

Figure 34: Connect USB Storage (Android 2.0 – 3.0)

Then tap "**Ok**". When the green Android icon turn orange, the phone is now in USB Mass Storage mode and should now appear as USB disk drives in "Devices with Removable Storage" and be assigned drive letters.

6.2.2 Connecting Android 4.0 – 4.1.2 (Ice Cream Sandwich)

Step 1: Open the "**Setting**" > "**More...**"

Step 2: Search then Tap on "**USB utilities**"

Step 3: Check the option: "**Connect Storage to PC**"

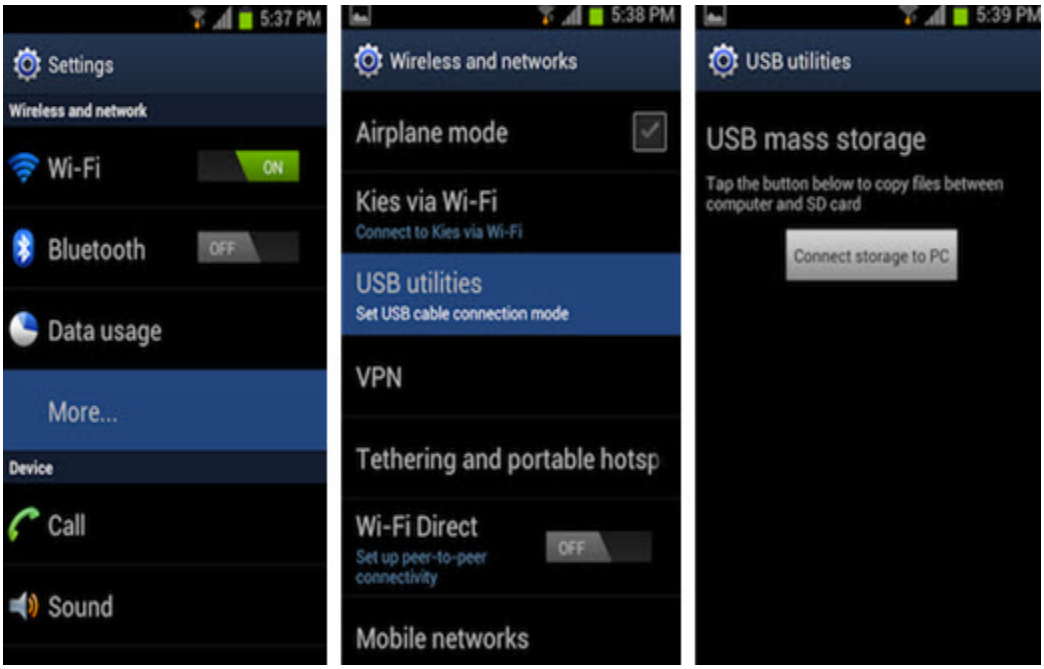


Figure 35: Connect USB Storage (Android 4.0 – 4.1.2)

Then plug the USB cable into the device and connect to your computer. The "**USB connected**" screen with the large green Android icon will appear. Tap on "**Connect USB storage**". A confirmation dialog box will appear. Tap "**OK**". The green Android icon will turn orange, indicating that the phone is now in USB Mass Storage mode and should now appear as USB disk drives in "**Devices with Removable Storage**" and be assigned drive letters.

6.2.3 Connecting Android 4.2 or Higher

You will be dismayed to see that there is no USB Debugging option in Android 4.2 or higher smartphones or tablets. Here is a way to turn it on since it is very well hidden!

Step1: Go to **Settings -> About Phone/Tablet**.

Step 2: Go to "**Build Number**" at the end of the Scroll list.

Step 3: Tap on "**Build Number**" repeatedly (**7 times**). On your third tap you should see message indicating that you only have 4 more taps to go to 'become a developer'. Keep tapping until you are a developer.

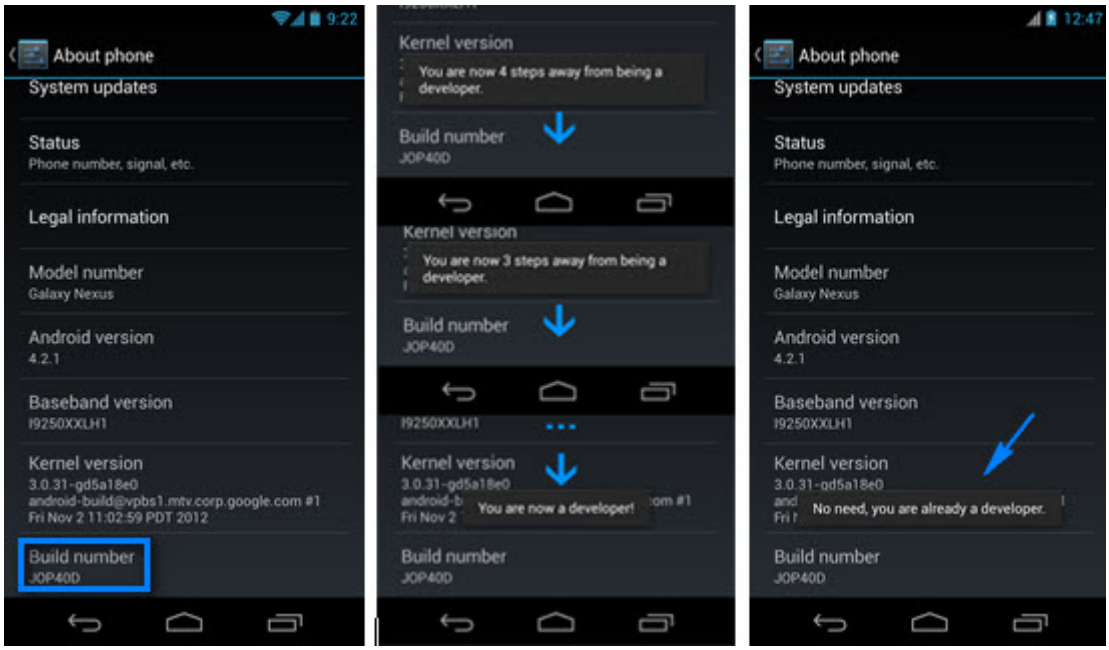


Figure 36: USB Debugging (Android 4.2 or Higher)

Step 4. Now Developer Option will be included in setting list of yours now. Go to "**Develop Options**" to check **USB debugging > OK. Done**.

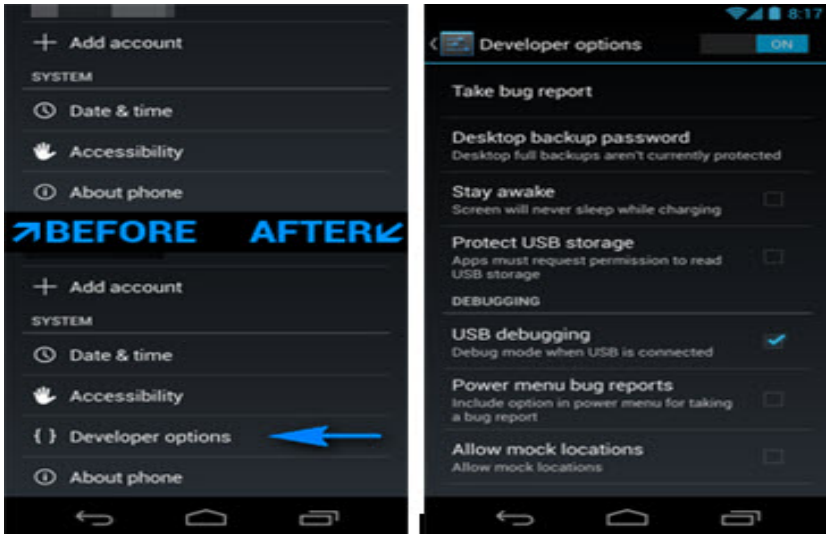


Figure 37: Connect USB Storage (Android 4.0 – 4.1.2)

After connecting your devices to PC with USB Mass Storage mass, or connect the memory card through your computer's card reader. Then you can recover deleted photos, videos from devices.

6.3 Transfer APK File Between Your Computer and Android Device

Step 1: Go to the folder **-> /MalwareDefense/app**

Step 2: Copy "**app-release.apk**" file to your android device.

6.4 Installing APK File On Android Device

Step 1: Open FileManager app on Device and click on the "**app-release.apk**" file.

Step 2: It will ask you whether to install this app or not. Click Yes or OK.

6.5 Installation Instruction (Developers)

Step 1: Download and Install Android Studio from Developers Website.

Step2: Download and Install the Latest Version of JAVA Development Kit (JDK) from Oracle's Website.

Step 3: Open Android Studio

From The Studio Project Dialog

This is the screen that appears the first time that you open Android Studio. You will also see it if you close your current project(s).

- From the project dialog choose the second option: **Open an existing Android Studio Project**

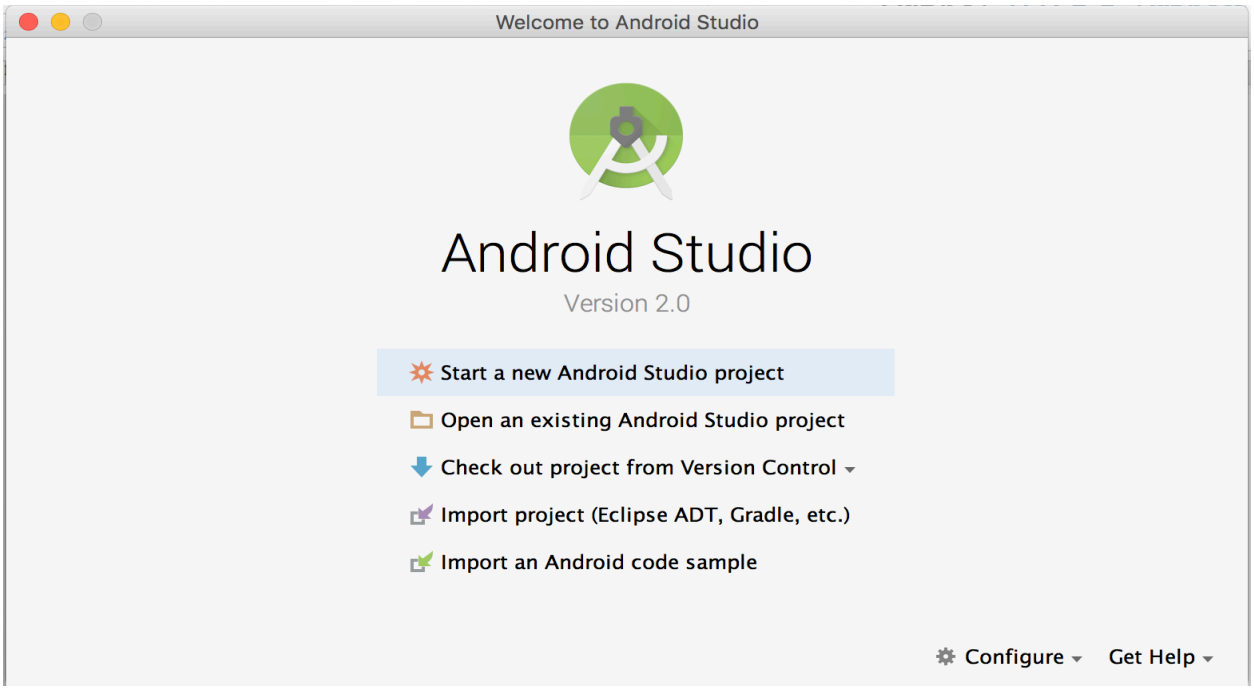


Figure 38: Android Studio Dialog

In the file dialog that appears, navigate into the project directory of the project (the one that contains the app folder)

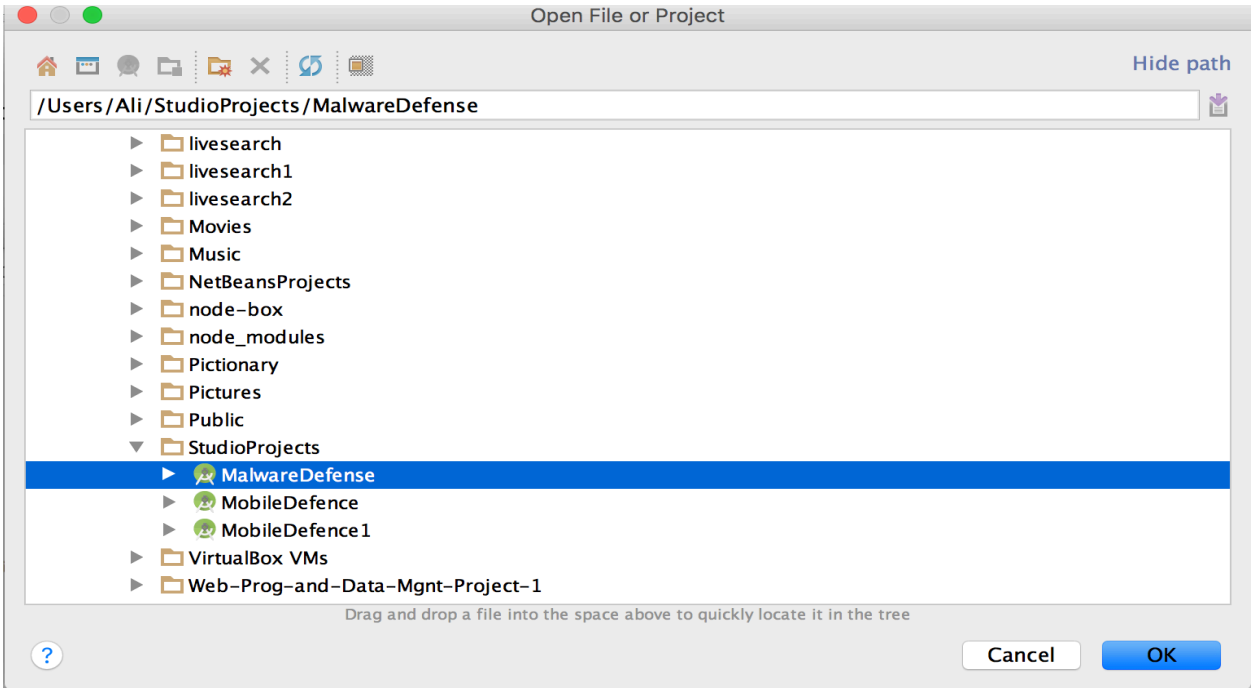


Figure 39: Android Studio Open File or Project Dialog

- Click Choose/OK

From The File Menu

Use this method if you already have a project open.

- Choose File/Open from the menu

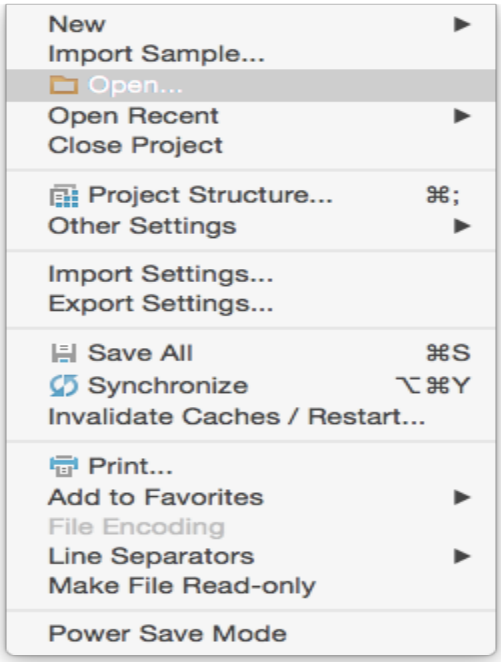


Figure 40: Android Studio Open Project from The File Menu Dialog

- In the file dialog that appears, navigate into the project directory of the project (the one that contains the app folder)

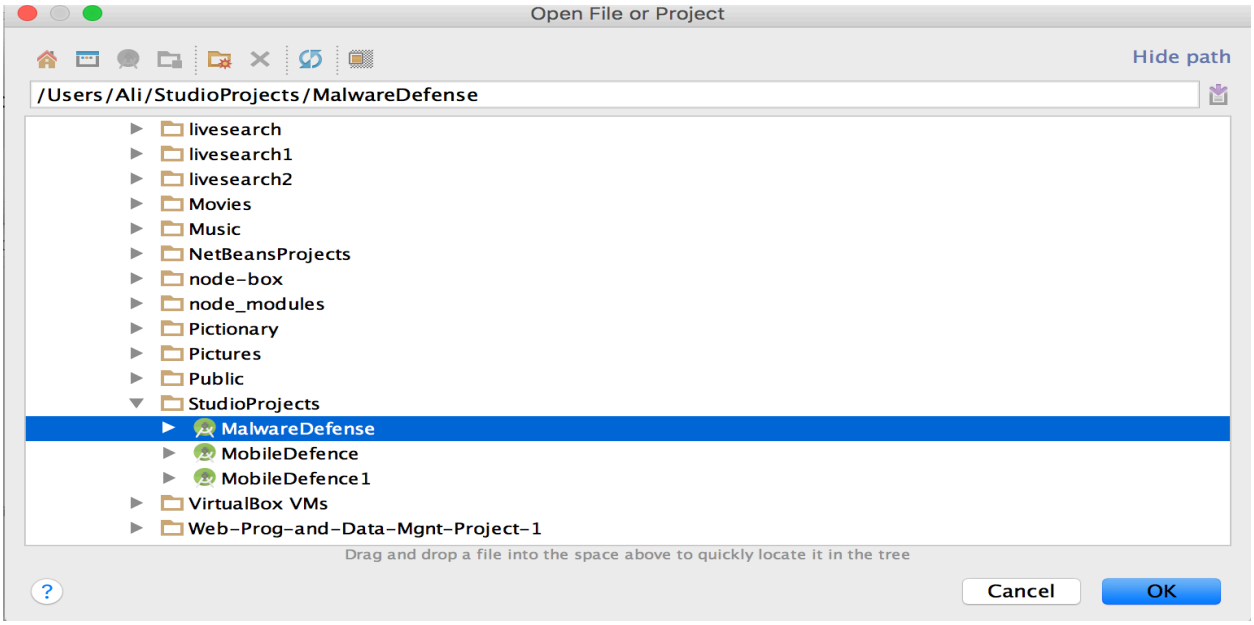


Figure 41: Android Studio Open File or Project Dialog

- Click Choose/OK

Chapter 7:

FUTURE WORK & RECOMMENDATIONS FOR ENHANCEMENTS

7 FUTURE WORK & RECOMMENDATIONS FOR ENHANCEMENTS

It is possible to add more functionality for the application. This application has many deficiencies because of limited time. In the future, it is possible to have a better art design (GUI) for more fresh experience. Also create a list of top developers, rate projects and developer's activity according to their input to development. Copyright policy must be defined in advance. Organize private groups of developers and idea makers. Add other social network support. Create one of the web sources for example a web page which can be accessed from other devices with different OS.

Implementing and Deploying the Malware Defense System for Apple iPhone iOS Devices.

Registering application with Google Services and uploading it into Google Play Store.

Develop an application for Mobile Operator Servers.

Deploy the application to the Main Servers of Telecom Operators.

Chapter 8:

RESULTS

Some Frequent API Parameters in Malware

Class	Method	Parameter type	Difference (%)
StringBuilder	append	Dangerous command	35.95
ContentResolver	query	SMS or MMS	23.65
StringBuilder	append	Unicode string	23.6
StringBuilder	init	Dangerous command	23.07
DataOutputSream	writebytes	Reads from process	21.80
DataOutputSream	init	Reads from process	21.62
runTime	exec	Dangerous command	21.27
InetSocketAddress	init	Port 80	19.91
StringBuilder	append	Compressed file	19.58
DataInputStream	init	Reads from connection	19.27
String	valueOf	Unicode string	18.05
StringBuilder	append	File manupilation	17.79
File	init	Accesses external storage	16.92
InetSocketAddress	init	Explicit IP	14.87
String	getBytes	URL manupilation	14.05
Intent	setFlags	SendTo	12.94
Intent	setFlags	Call	11.67
ContentResolver	query	Telephony	10.88
Intent	setFlags	Send	10.47
ContentResolver	query	Call_log	10.12

Figure 42: Frequent API Parameters in Malware

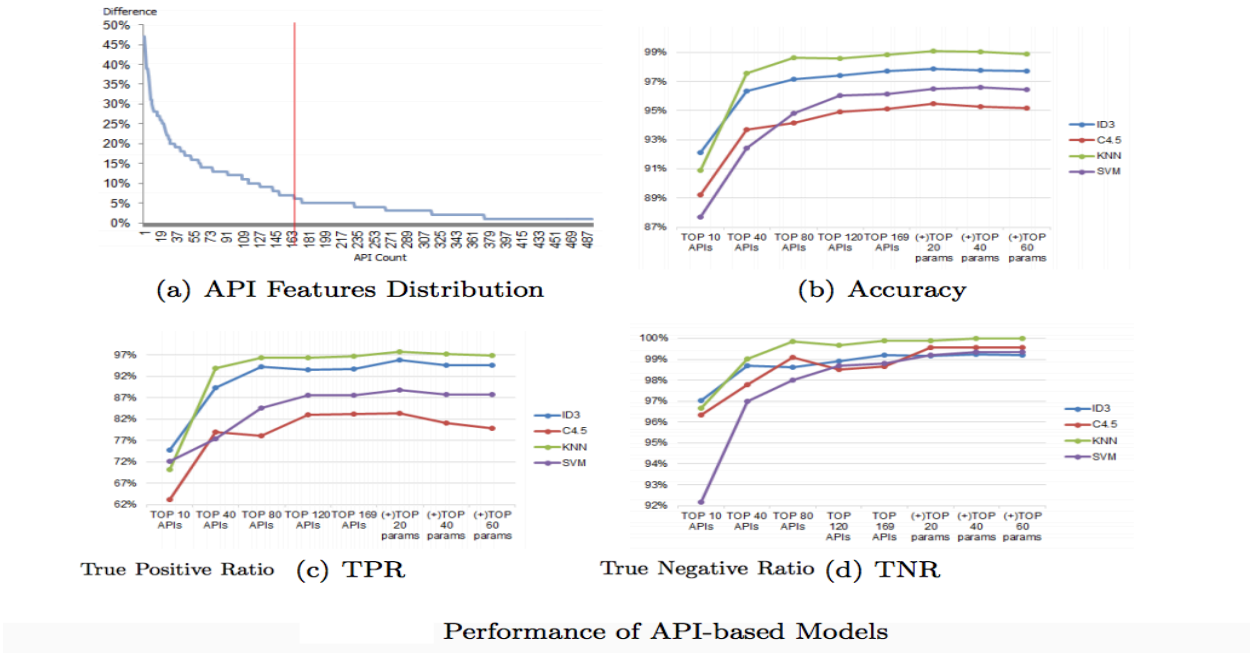


Figure 43: Performance of API-Based Models

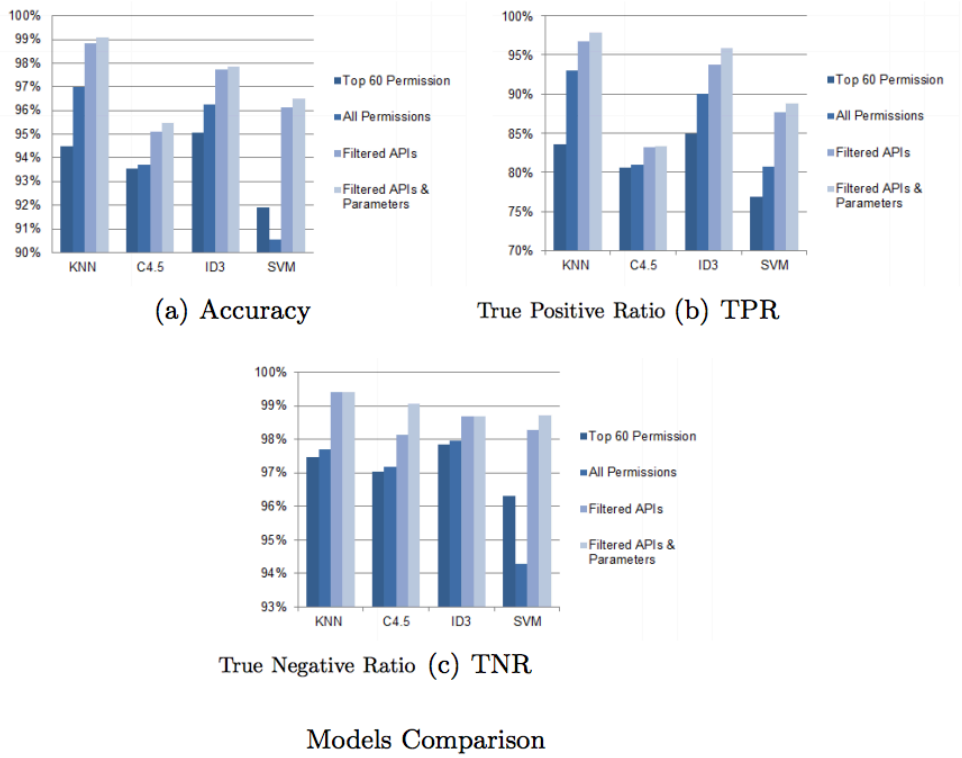
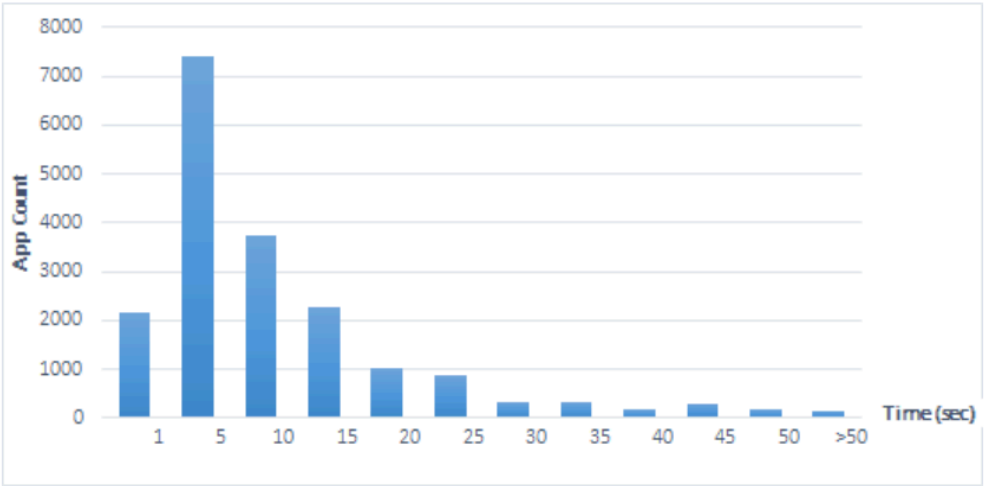


Figure 44: Model Comparison



Distribution of DroidAPIMiner Processing time

Figure 45: Distribution of DroidAPIMiner Processing Time

Chapter 9:

REFERENCES

&

BIBLIOGRAPHY

8. REFERENCES & BIBLIOGRAPHY

[1]. Android N Developer Preview. (n.d.). Retrieved April 18, 2016, from http://developer.android.com/index.html

[2]. Android StudioThe Official IDE for Android. (n.d.). Retrieved April 18, 2016, from http://developer.android.com/sdk/index.html

[3]. Android. (n.d.). Retrieved April 18, 2016, from http://www.android.com/

[4]. B. Wu, J. Wu, E. Fernandez, and S. Magliveras, "Secure and efficient key management in mobile ad hoc networks," in Proc. IEEE IPDPS, 2005.

[5]. C. Hsu and C. Lin, "A comparison of methods for multiclass support vector machines," IEEE transactions on Neural Networks, vol. 13, no. 2, pp. 415–425, 2002.

[6]. C. van Rijsbergen, Information Retrieval, Second Edition. Butterworths, 1979.

[7]. C.-C. Chang and C.-J. Lin, LIBSVM: a library for support vector machines, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[8]. D. Faria and D. Cheriton, "Detecting identity-based attacks in wireless networks using signalprints," in Proceedings of the ACM Workshop on Wireless Security (WiSe), September 2006.

[9]. D. Madigan, E. Elnahrawy, R. Martin, W. Ju, P. Krishnan, and A. S. Krishnakumar, "Bayesian indoor positioning systems," in Proc. IEEE INFOCOM,

[10]. E. Elnahrawy, X. Li, and R. P. Martin, "The limits of localization using signal strength: A comparative study," in Proc. IEEE SECON, Oct. 2004.

[11]. F. Ferreri, M. Bernaschi, and L. Valcamonici, "Access points vulnerabilities to dos attacks in 802.11 networks," in Proceedings of the IEEE Wireless Communications and Networking Conference, 2004.

[12]. F. Guo and T. Chiueh, "Sequence number-based mac address spoof detection," in Recent Advances in Intrusion Detection, 2006, pp. 309–329.

[13]. G. Zhou, T. He, S. Krishnamurthy, and J. A. Stankovic, "Models and solutions for radio irregularity in wireless sensor networks," ACM Transactions on Sensor Networks, vol. 2, pp. 221–262, 2006.

[14]. Goldsmith, Wireless Communications: Principles and Practice. New York, NY, USA: Cambridge University Press, 2005.

[15]. Google Developers. (n.d.). Retrieved April 18, 2016, from https://developers.google.com/

[16]. J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions," in Proceedings of the USENIX Security Symposium, 2003, pp. 15 – 28.

[17]. J. Yang and Y. Chen, "A theoretical analysis of wireless localization using RF-based fingerprint matching," in Proceedings of the Fourth International Workshop on System Management Techniques, Processes, and Services (SMTPS), April 2008.

[18]. J. Yang, Y. Chen, and W. Trappe, "Detecting spoofing attacks in mobile wireless environments," in Proc. IEEE SECON, 2009.

[19]. K. Wang, "Estimating the number of clusters via system evolution for cluster analysis of gene expression data," Computer Science Department, Xidian University, P.R.China, Technical Report NO. 2007-258, 2007.

[20]. L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis. Wiley Series in Probability and Statistics, 1990.

[21]. L. Sang and A. Arora, "Spatial signatures for lightweight security in wireless sensor networks," in The 27th Conference on Computer Communications, INFOCOM 2008., 2008, pp. 2137–2145.

[22]. L. Xiao, L. J. Greenstein, N. B. Mandayam, and W. Trappe, "Fingerprints in the ether: using the physical layer for wireless authentication," in Proceedings of the IEEE International Conference on Communications (ICC), June 2007, pp. 4646–4651.

[23]. Li, Y. , Hui, P. , Jin, D. , Su, L. , & Zeng, L. (2014). Optimal distributed malware defense in mobile networks with heterogeneous devices. IEEE Transactions on Mobile Computing, 13(2), 377-391.

[24]. M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. New York: Courier Dover, 1965.

[25]. M. bohge and W. Trappe, "An authentication framework for hierarchical ad hoc sensor networks," in Proceedings of the ACM Workshop on Wireless Security (WiSe), 2003, pp. 79–87.

[26]. N. Cristianini and J. Shawe-Taylor, An introduction to support Vector Machines: and other kernel-based learning methods. Cambridge Univ Pr, 2000.

[27]. P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in Proc. IEEE INFOCOM, 2000.

[28]. P. Enge and P. Misra, Global Positioning System: Signals, Measurements and Performance. Ganga-Jamuna Pr, 2001.

[29]. P. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," Journal of Computational and Applied Mathematics, vol. 20, no. 1, pp. 53–65, November 1987.

[30]. Q. Li and W. Trappe, "Relationship-based detection of spoofing-related anomalous traffic in ad hoc networks," in Proc. IEEE SECON, 2006.

[31]. T. Fawcett, "An introduction to ROC analysis," Pattern Recognition Letters, Elsevier, vol. 27, pp. 861–874, 2006.

[32]. T. He, C. Huang, B. Blum, J. A. Stankovic, and T. Abdelzaher, "Range-free localization schemes in large scale sensor networks," in Proceedings of the Ninth Annual ACM International Conference on Mobile Computing and Networking (MobiCom'03), 2003.

[33]. T. Sarkar, Z. Ji, K. Kim, A. Medouri, and M. Salazar-Palma, "A survey of various propagation models for mobile communication," Antennas and Propagation Magazine, IEEE, vol. 45, no. 3, pp. 51–82, June 2003.

[34]. Twitter Developers. (n.d.). Retrieved April 18, 2016, from https://dev.twitter.com/

[35]. UML Web Site! (n.d.). Retrieved April 18, 2016, from http://www.uml.org/

[36]. V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in Proceedings of the 14th ACM international conference on Mobile computing and networking, 2008, pp. 116–127.

[37]. V. Franc and V. Hlavác, "Multi-class support vector machine," in Proc. ICPR, vol. 16, 2002, pp. 236–239.

[38]. W3C. (n.d.). Retrieved April 18, 2016, from https://www.w3.org/

[39]. Wool, "Lightweight key management for ieee 802.11 wireless lans with key refresh and host revocation," ACM/Springer Wireless Networks, vol. 11, no. 6, pp. 677–686, 2005.

[40]. Y. Chen, J. Francisco, W. Trappe, and R. P. Martin, "A practical approach to landmark deployment for indoor localization," in Proc. IEEE SECON, September 2006.

[41]. Y. Chen, K. Kleisouris, X. Li, W. Trappe, and R. P. Martin, "The robustness of localization algorithms to signal strength attacks: a comparative study," in Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS), June 2006, pp. 546–563.

[42]. Y. Chen, W. Trappe, and R. Martin, "Attack detection in wireless localization," in Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), April 2007.

[43]. Y. Chen, W. Trappe, and R. P. Martin, "Detecting and localizing wireless spoofing attacks," in Proc. IEEE SECON, May 2007.

[44]. Y. Sheng, K. Tan, G. Chen, D. Kotz, and A. Campbell, "Detecting 802.11 MAC layer spoofing using received signal strength," in Proc. IEEE INFOCOM, April 2008.

[45]. Z. Yang, E. Ekici, and D. Xuan, "A localization-based anti-sensor network system," in 26th IEEE International Conference on Computer Communications. INFOCOM 2007, 2007, pp. 2396–2400.

[46]. Johnson, A. , & Flegal, J. (2014). A modified conditional metropolis-hastings sampler. Computational Statistics & Data Analysis, 78, 141-152.