A PROJECT REPORT ON

ANDROID SECURITY BASED TOOLS

SUBMITTED TO THE UNIVERSITY OF PUNE, PUNE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE OF

BACHELOR OF ENGINEERING (Computer Engineering)

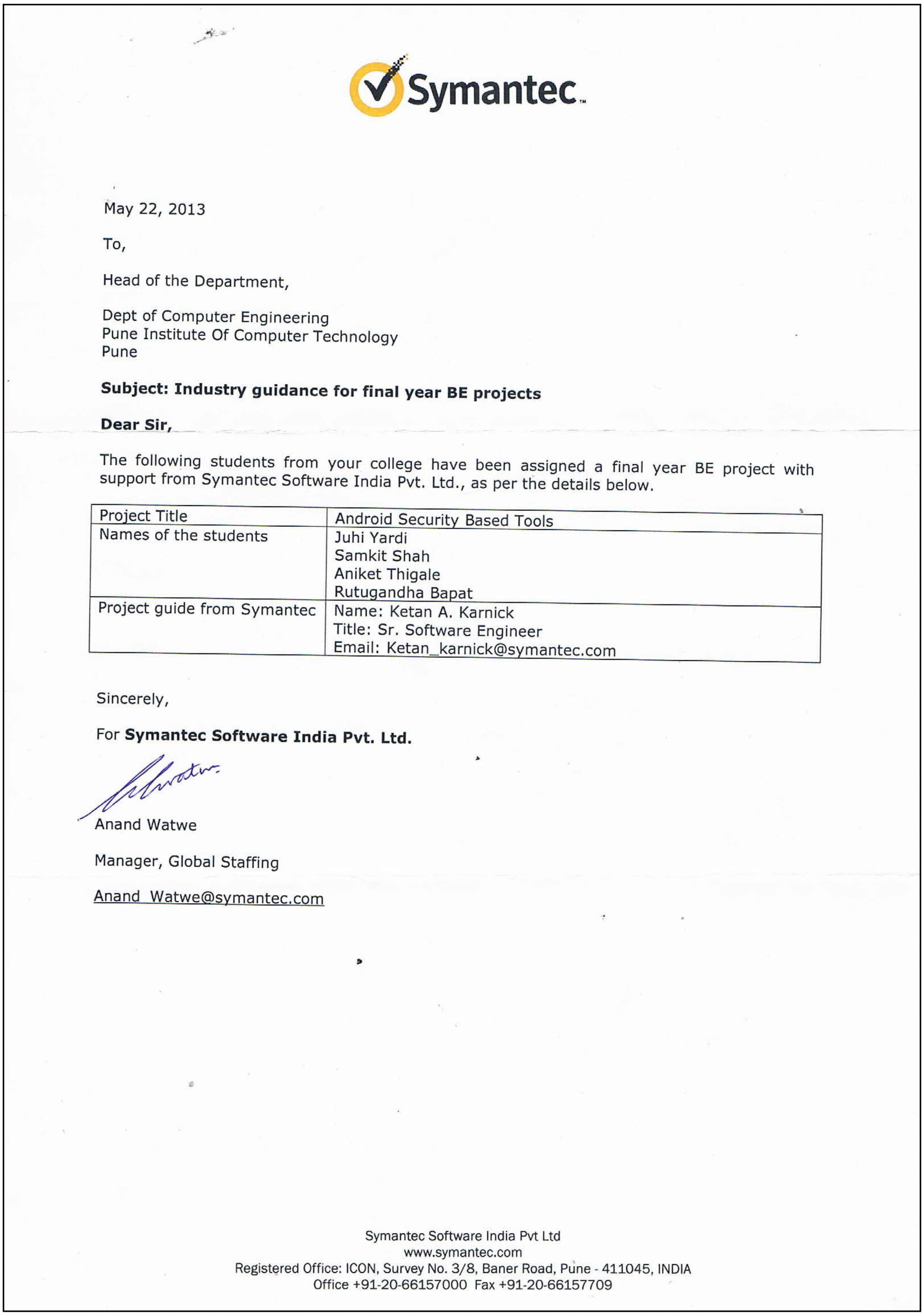
BY

Rutugandha Bapat	Exam No: 4104
Samkit Shah	Exam No: 4160
Aniket Thigale	Exam No: 4166
Juhi Yardi	Exam No: 4168

Under The Guidance of

Prof. M.S.Wakode

DEPARTMENT OF COMPUTER ENGINEERING
PUNE INSTITUTE OF COMPUTER TECHNOLOGY
Sr. No. 27, Pune Satara Road, Dhanakawadi
PUNE 411043



May 22, 2013

To,

Head of the Department,

Dept of Computer Engineering
Pune Institute Of Computer Technology
Pune

Subject: Industry guidance for final year BE projects

Dear Sir,

The following students from your college have been assigned a final year BE project with support from Symantec Software India Pvt. Ltd., as per the details below.

Project Title	Android Security Based Tools
Names of the students	Juhi Yardi Samkit Shah Aniket Thigale Rutugandha Bapat
Project guide from Symantec	Name: Ketan A. Karnick Title: Sr. Software Engineer Email: Ketan_karnick@symantec.com

Sincerely,

For **Symantec Software India Pvt. Ltd.**

[illegible]

Anand Watwe

Manager, Global Staffing

Anand_Watwe@symantec.com

Symantec Software India Pvt Ltd
www.symantec.com
Registered Office: ICON, Survey No. 3/8, Baner Road, Pune - 411045, INDIA
Office +91-20-66157000 Fax +91-20-66157709

PUNE INSTITUTE OF COMPUTER TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the preliminary project report entitled

“ANDROID SECURITY BASED TOOLS”

Submitted by

Rutugandha Bapat	Exam No: 4104
Samkit Shah	Exam No: 4160
Aniket Thigale	Exam No: 4166
Juhi Yardi	Exam No: 4168

is a bonafide work carried out by them under the supervision of Prof.M.S.Wakode and it is submitted towards partial fulfillment of the requirement of University of Pune, Pune, for the award of the degree of Bachelor of Engineering (Computer Engineering)

Prof. M.S.Wakode
Internal Guide

Name:
Signature:
External Examiner

Place: Pune
Date:

Head
Department of Computer Engineering
PICT Pune

ACKNOWLEDGEMENT

It gives us a great pleasure to present the project report for our project on —

"Android Security Based Tools"

We would like to take this opportunity to thank our internal guide **Mrs. M.S. Wakode** for giving us all the help and guidance we needed. We would like to express our sincere gratitude for her constant encouragement and valuable guidance for project work. We are really grateful to her for her support throughout the analysis and design phase.

We are also grateful to, **Prof.Girish Potdar**, Head of Computer Department, Pune Institute of Computer Technology and other staff members for giving important suggestions.

Our external guide Mr.Ketan Karnick has always been very prompt at helping and sharing his valuable technical know-how about the subject. The smooth completion of this project would not have been possible without their guidance.

Our special thanks to Library and the non-teaching staff of Pune Institute of Computer Technology who have provided us access to the lab facilities, from time to time. We would also like to thank the faculty and all the lab assistants of the Computer Department for their timely help.

We would like to express our gratitude towards our parents, who have always been a source of inspiration and motivation.

LIST OF DIAGRAMS

LIST OF TABLES

ABSTRACT

Smartphones are steadily gaining popularity, creating new application areas as their capabilities increase in terms of computational power, sensors and communication. Emerging new features of mobile devices give opportunity to new threats. While being based on a Linux kernel, Android has unique properties and specific limitations due to its mobile nature. This makes it harder to detect and react upon malware attacks if using conventional techniques.

This project aims at creating system which monitors the development as well as execution phase of each application for any security leaks & malicious behavior, using static and dynamic analysis techniques. Static analysis involves parsing of key components of android application to detect loopholes overlooked by the developer & also strengthens the application installation for end users. The goal of static analysis is to flag the vulnerabilities in the code to the developer to avoid capability leaks & privilege escalation attacks. Dynamic analysis will help us reason the behavior of the applications from information gathered at the run time. We will further investigate the behavior of malicious Android applications and present a simple and effective way to safely execute and analyze them.

Index

CHAPTER 1

INTRODUCTION

1.1 Project Idea:

Smartphone's are steadily gaining popularity, creating new application areas as their capabilities increase in terms of computational power, sensors and communication. The popularity and adoption of Smartphone's has greatly stimulated the spread of mobile malware, especially on the popular platforms such as Android. In light of their rapid growth, there is a pressing need to develop effective solutions. However, our defense capability is largely constrained by the limited understanding of these emerging mobile malware and the lack of timely access to related samples. Storage of personal data on phones, sensitive information like bank account details and 24*7 enabled social networking etc. makes a smart phone, an open treasure to malware developers.

The focus of this project is to develop an "Application Monitor" for Android which monitors the complete life cycle of an application on the Android phone i.e. right from the installation of application by performing static analysis prior the application installation & then dynamic analysis during run time of the application in order to closely determine its behaviour. Also an application can be coded by the developer in such a way that it becomes difficult for a third party application to exploit it. A developer's tool will hence be provided which performs static taint checking to track the data flow in the application to trace out data paths that represent capability leaks which points out to the developers the particular modules where he can strengthen the code.

1.2 Motivation:

The popularity and adoption of Smartphone's in various sensitive fields like online banking, personal and company data, has greatly simulated the spread of mobile malware on Android. Since its establishment, the Android applications market has been infected by a proliferation of malicious applications. Recent studies show that rogue developers are injecting malware into legitimate market applications which are then installed on open source sites for consumer uptake**.** While being based on a Linux kernel, Android has unique properties and specific limitations due to its mobile nature. This makes it harder to detect and react upon malware attacks if using conventional techniques. Current steps towards security in Android involves standardizations of permission requests, development formats & certification etc. are still not strong enough to extrude such malwares due to following reasons,

i) It is an Open source Technology, having millions of developers worldwide.

ii) Sources for obtaining applications are varied and endless.

iii) There are increasing number of Android users & users belong to various age groups and professions.

iv) Most of the users are least bothered about the permissions an application might require & are likely to be allowing all of them.

v) Lack of an Application monitor in Android.

1.3 LITERATURE SURVEY

Most of the security measures that exist right now are in form of applications in the android phone**.** Results indicate that the traditional anti-virus mechanisms running currently at application level are not able to correctly identify malicious Android applications [3]. The widespread use of smartphones has provided a new way for malware authors to propagate infectious software. The application market has the highest percentage of malicious software is the Android market. This is parly due to the loose permission granting structure currently in place for Android applications, as demonstrated in [1], [3] and [5].
Similar to desktop malware, there exist two common methods to investigate the structure and behaviour of Android malware: static analysis and dynamic analysis [3]. In the research literature on malware targeting desktops and laptops, one important approach has been to identify behavioural patterns [6] which then help to classify malware in the hope of better understanding its aims and so more quickly devising counter-measures. We identify patterns of behaviour in both benign and malicious applications which can distinguish one application from another. To our knowledge, the work in this system is to first perform the permission mapping which is also the direction of the system. . The category- permissions map gives us the detailed view of the permissions an application demands when it first establishes itself in an android environment. Our dataset comprises of samples that we collected from publicly available sources. Every application is hence monitored to check for its extra demands. Although applications are limited to perform actions that are allowed to be performed by the users at the install time, they could acquire extra capabilities which are abilities to perform various actions by exploiting interfaces by more capable applications. The idea is to also prevent such attacks.

CHAPTER 2

PROBLEM DEFINITION AND SCOPE

PROBLEM DEFINITION AND SCOPE

2.1 Problem Statement

Developing a system for the Android Platform that monitors the life cycle of each application for any unauthorised activity, intent and malicious behaviour using static and dynamic analysis and flags it to the user with the severity level in order to perform necessary actions.

2.2 Goals and objectives

1. Ensuring security of the Android mobile System towards the malicious activities.
2. Securing private & personal data of the mobile user.
3. Detecting & restricting Privilege Escalation attacks.
4. Detection of malware using machine learning algorithms & user notification regarding the same.
5. A developer’s tool to prevent permissions re-delegation and detect capability leaks in the developed application.
6. Dynamic and static security handling provisions for faster and complete detection.

2.3 Statement of scope

Scope of the system encompasses:

1. Permission Extraction from the Android Package.
2. Comparing the permissions extracted to the database containing Category→permission mapping.
3. Verification of the inter-process communications against a set of pre-defined security policies.
4. Static analysis of the android application before installation of application.
5. Building an Android Loadable Kernel Module for catching system calls.
6. Detection of phone calls made or SMS sent by the application.
7. Creation of feature vector and its MD5 Checksum and sending it the server for processing it.
8. Running algorithms on the feature vector and classifying the application.
9. Updation of malware database.
10. Notification to the user about every possible or suspicious security breach.

11. Creation of a tool for developers to detect taint checking and possible capability leaks.
12. Graphical and tabular module wise results for developers.

2.4 Major constraints based on SWOT Analysis

2.4.1 Strength:

1. Security issues regarding authorized access & scope of application are handled in a single system (a service).
2. User friendly system that requires very little intervention from the user.
3. The system takes into account that the client is not aware of intricate details of the security issues concerning android.
4. Results are presented in simple graphical and tabular form to be understood by the user so that further user action becomes easier.
5. User is informed of every possible security threat from permission escalation and miss-handling to malware attacks.
6. The system is scalable in terms of number of applications running on phone if the system is deployed on remote server.

2.4.2 Weakness:

1. Tool is not backward compatible.
2. Internet speed affects the communication between system on device & database.
3. Accuracy of the classification algorithms depends on the number of vectors in the training set on server.
4. Dependency of the system on the categorization provided by the user.

2.4.3 Opportunities:

1. System can be deployed on the cloud akin to the Google Play Store for distributed use.

2.4.4 Threats:

1. Security of the systems database.
2. Required internet speed not available.
3. User does not report suspicious behavior to the central database and ask for update.

2.5 Technology and associated platform:

Hardware Resources Required

1. Android Devices – testing done on HTC One
2. Android OS v4.0 ICS
3. ARMv7 Processor
4. 768MB RAM
5. WiFi connection

Software Resources Required

1. Ubuntu 12.10 64 bit
2. Minimum Req. of Android OS : ICS
3. Rooted phone/ Android SDK Emulator
4. Play Framework 2.2.0
5. Goldfish Kernel Source 2.6.29
6. Android Source Code - Platform v.4.0.3_r1
7. Android SDK Eclipse Bundle
8. JDK 1.6.0_45 on the Linux machine
9. MySQL WorkBench v5.2.40
10. Android NDK r9c Toolchain
11. Scala Plugin for Eclipse
12. WEKA API

2.6 Area of Project: Android, Security, Access Control, Pattern matching, Data Mining

2.7 Technical Keywords (Refer ACM Keywords):

- D.4.6 Security and Protection
  - Authentication
  - Invasive Software
  - Security Kernels
  - Access Control
  - Verification
- H.2.8 Database Applications
  - Data Mining
- H.3.3 Information Search and Retrieval
  - Clustering
  - Relevance Feedback

- E.1 Data Structures
  - Graphs
  - Trees

CHAPTER 3

SOFTWARE PROJECT PLAN

3.1 Introduction

3.1.0 Problem Definition

Developing a system for the Android Platform that monitors the life cycle of each application for any unauthorised activity, intent and malicious behaviour using static and dynamic analysis and flags it to the user with the severity level in order to perform necessary actions. Making an application secure is also the responsibility of the developer. The system also provides a developer's tool which uses the inter-procedural control flow graph searching coupled with taint checking to detect capability leaks and provides the developers a graphical and tabular result which allows him to analyse the modules in order to strengthen his code.

3.1.1 Purpose of the Document:

This document seeks to provide the Software Requirements Specifications for the project, *"Android Security Based Tools"*, to be developed as a part of Final Year Engineering Project at Pune Institute of Computer Technology with purpose of obtaining a degree in Computer Science at the Pune University. This document will be used by faculty of Computer Engineering department and the external guide. This document provides detailed description of software context, its usage scenario, data model and description, data object and their relationships, functional model and description, performance issues, software design constraints, software interfacing description, limitations and validation criteria. This document describes the function and performance considerations of operating systems. It gives general background and reference information. The document gives the complete problem definition, project scope requirements, utility and functionality of our system.

3.2 Project Estimates

3.2.1 Historical data used for estimates

The performance of the similar systems available is aggregated from various published works.

3.2.2 Estimation techniques applied and results

3.2.2.1 Estimation technique : COCOMO

Basic Cocomo formula:

Effort Applied (E) = $a_b(\text{KLOC})^{b}{}_{b}$ [person-months]

Development Time (D) = $c_b(\text{Effort Applied})^{d}{}_{b}$ [months]

People required (P) = Effort Applied / Development Time [count]

The Intermediate Cocomo formula:

E=ai(KLoC)^(bi).EAF

Software project	**a_i**	**b_i**
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

Table 3.1 COCOMO Value Reference

The EAF is calculated from the following cost driver table

	Ratings					
Cost Drivers	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65

Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	

Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

Table 3.2 EAF Value

3.2.2.2 Estimate for technique COCOMO

Software Development (Elaboration and Construction)

Effort = 19.7 Person-months
Schedule = 9.8 Months
Cost = $3931

Total Equivalent Size = 3334 SLOC

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.2	1.2	1.0	$236
Elaboration	4.7	3.7	1.3	$944
Construction	14.9	6.1	2.4	$2988
Transition	2.4	1.2	1.9	$472

Table 3.3 Acquisition Phase Distribution

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.6	1.5	0.3
Environment/CM	0.1	0.4	0.7	0.1
Requirements	0.4	0.8	1.2	0.1
Design	0.2	1.7	2.4	0.1
Implementation	0.1	0.6	5.1	0.4
Assessment	0.1	0.5	3.6	0.6
Deployment	0.0	0.1	0.4	0.7

Table 3.4 Software Effort Distribution for RUP/MBASE (Person-Months)

3.2.2.3 Estimation Technique - Function Points

As defined in requirements engineering, functional requirements specify particular results of a system. This should be contrasted with non-functional requirements which specify overall characteristics such as cost and reliability. Functional requirements drive the *application architecture* of a system, while non-functional requirements drive the *technical architecture* of a system.

The functional user requirements of the software are identified and each one is categorized into one of five types: outputs, inquiries, inputs, internal files, and external interfaces. Once the function is identified and categorized into a type, it is then assessed for complexity and assigned a number of function points. Each of these functional user requirements maps to an end-user business function, such as a data entry for an Input or a user query for an Inquiry.

3.2.2.4 Estimate for Technique - Functional Points

Software Development (Elaboration and Construction)

Effort = 27.8 Person-months
Schedule = 11.0 Months
Cost = $5554
Total Equivalent Size = 4560 SLOC

Phase	Effort (Person-months)	Schedule (Months)	Average Staff	Cost (Dollars)
Inception	1.7	1.4	1.2	$333
Elaboration	6.7	4.1	1.6	$1333
Construction	21.1	6.9	3.1	$4221
Transition	3.3	1.4	2.4	$667

Table 3.5 Acquisition Phase Distribution(FP)

Phase/Activity	Inception	Elaboration	Construction	Transition
Management	0.2	0.8	2.1	0.5
Environment/CM	0.2	0.5	1.1	0.2
Requirements	0.6	1.2	1.7	0.1
Design	0.3	2.4	3.4	0.1
Implementation	0.1	0.9	7.2	0.6
Assessment	0.1	0.7	5.1	0.8
Deployment	0.0	0.2	0.6	1.0

Table 3.6 Software Effort Distribution for RUP/MBASE (Person-Months)(FP)

3.2.3 Reconciled Estimate

Software Development (Elaboration and Construction)

Effort = 23.75 Person-months

Schedule = 10.4 Months

Cost = $4742

Total Equivalent Size = 3947 SLOC

3.2.4 Project Resources

Project resources [People, Hardware, Software, Tools and other resources] based on Memory Sharing, IPC, and Concurrency derived using appendices to be referred.

3.2.4.1 Project Resources

Hardware Resources Required:

1. Android Devices – testing done on HTC One V
2. Android OS v4.0 ICS
3. ARMv7 Processor
4. 768MB RAM
5. Wi-Fi connection

Software Resources Required:

1. Ubuntu 12.10 64 bit
2. Minimum Req. of Android OS : ICS
3. Rooted phone/ Android SDK Emulator
4. Play Framework 2.2.0
5. Goldfish Kernel Source 2.6.29
6. Android Source Code - Platform v.4.0.3_r1
7. Android SDK Eclipse Bundle
8. JDK 1.6.0_45 on the Linux machine
9. MySQL WorkBench v5.2.40
10. Android NDK r9c Toolchain
11. Scala Plugin for Eclipse
12. WEKA API

3.3 Risk Management

We have classified the risks into different categories:

- Development Risks
- Business Risks
- Project Risks
- Requirements Risks

3.3.1 Risk Table

Catastrophic: Failure to meet the requirement would result in project failure

Critical: Failure would degrade performance; project success is Questionable.

Marginal: Failure would result in degradation of secondary project.

Negligible: Failure would have slight impact

No.	Risks	Category	Probability
1	Lack of training on tools or knowledge on different domains	Development	20%
2	Delivery deadline will be tightened	Business	20%
3	Requirement will change	Requirements	10%

3.3.2 Risk Mitigation, Monitoring, Management

It will help the team to study the major risks in the initial stages. The requirement specification and software specification will be reviewed and studied. All the risks though making negligible impact will be taken into consideration. A proactive approach will be practiced. The risks will be recorded and a best course of action will be planned to avoid the risks. As all risks cannot be avoided, performing risk management, we can attempt to ensure that the right risks are taken at right time.

No.	Risks	Mitigation	Monitoring	Management
1	Lack of training on tools or knowledge on the various domains	The tool manuals and tutorials are frequently practiced and the topics are researched upon and discussed frequently among team members	Done by members of the team	Assure foolproof understanding each time
2	Requirement will change	Study and predict what requirements are and what they can be in the near future.	Done by administrator and members	Frequent Discussion with the team members.
3	Ambiguity in data structures written by developers.	Carry out unit tests on minimal set of rules	By members	Change the rules as and when if at all ambiguity occurs
4	Poor documentation.	The code is documented to reflect the meaning to the observer.	By other members of the team	Any change is properly documented

Table 3.3.2 Risk Mitigation, Monitoring and Management Table

3.4 Project Schedule

Refer Annexure C

3.4.1 Project task set

Refer Annexure C

3.4.2 Task network

Refer Annexure C

3.4.3 Timeline chart

Refer Annexure C.

3.5 Staff Organization

3.5.1 Team structure

The project is being worked upon by a team of 5 people (1 project internal guide and 4 project developers). Each project developer is aware of the entire working of the project. This is possible due to the fact that the project group is small. Thus distribution of work is according to the need of the hour. It is decided to keep the team structure highly flexible throughout the project. Each individual shall contribute equally through all the phases of the project namely Problem Definition, Requirements Gathering and Analysis, Design, Coding, Testing and Documentation.

Role	Responsibilities	Participant(s)
Project Sponsor	Ultimate decision-maker and tie-breaker Provide project oversight and guidance Review/approve some project elements	Symantec Guide : Mr. Ketan Karnick
Project Guide	Manages project in accordance to the project plan Provide overall project direction Direct/lead team members toward project objectives Handle problem resolution	Prof. M.S.Wakode
Project Participants	Communicate project goals, status and progress throughout the project to personnel in their area Review and approve project deliverables Coordinates participation of work groups, individuals and stakeholders . Assure quality of products that will meet the	Aniket Thigale Rutugandha Bapat Samkit Shah Juhi Yardi

Role	Responsibilities	Participant(s)
	project goals and objectives . Identify risks and issues and help in resolutions	

Table 3.5.1 Team Structure

3.5.2 Management reporting and communication

- ❖ The communication began in June with initial assignments based on studying the Android Architecture perfectly. Meetings were held to discuss our findings and the direction in which to proceed ahead.
- ❖ In September, the focus was on finding problem definition and related Literature Survey. We had an average of one meeting per week.
- ❖ Minutes of the meeting were recorded after each meeting with our external guide.
- ❖ November onwards module by module completion was achieved.
- ❖ In March the different modules were integrated to form a full-fledged system.

3. 6 Tracking and Control Mechanisms

3.6.1 Quality assurance and control

Software quality is defined as conformance to explicitly state functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed softwares. Software quality assurance (SQA) is an activity that is applied throughout the software process. It involves various steps like:

- ❖ Create a set of activities that will help ensure that every software engineering work product exhibits high quality.
- ❖ Perform quality assurance activities on every software products.
- ❖ Use matrix to develop strategies for improving your software process and as a consequence the quality of the product.
- ❖ Everyone involved in the software engineering process is responsible for the quality of product.

Documentation

As a part of ensuring software, project following documents should be prepared

1. Software Requirement Specification
2. Software Design Specification
3. Risk Mitigation, Monitoring and Management Plan

Standards, Practices and Conversions

All documents are written in Microsoft Word in Font Times New Roman and size 12. The headings are bold with font size ranging from 12 to 14. The documents are fully justified.

Coding Standards

The code for various modules is written in Java.

Developer's tool as is hosted on website needs usage of common web based languages like Jsp, servlets, PhP.

CSS has been used for styling.

Review Guidelines

The team will update the documents on regular basis and send the updated documents to team members as well as external guide for critical reviews and assessment. Suggested improvements and changes will be incorporated promptly.

Metrics

Following metrics will be used to ensure software quality:

1) Team meeting every day to discuss progress.

2) Meeting with internal guide every week to analyze the progress over the week

3) Meeting with external guide once in a week to discuss the progress till date, Suggestion of changes for improvement and deciding the future course of Action.

4) Logging of daily work by every team member.

5) Use of group mailing list to manage work done by every team member.

6) Periodic analysis of the man-hours being put in by each team member

CHAPTER 4

SOFTWARE REQUIREMENT SPECIFICATION

4.1 Introduction

This document accounts the detailed logical, behavioural, physical and process level detailed functionality and description of the system.

4.1.1 Purpose and Scope of Document:

The purpose of the SRS is to model the Android Security Based Tool System in accordance to the Software Development Life Cycle. The SRS enlists the detailed internal and external components of the system along with their functionality in detail to successfully complete the design phase of the project.
The SRS covers the following:

1. Prerequisites
2. Internal Components of System
3. External Components of the system
4. Functionality of each component in the system
5. Data Flow / Sequence Flow
6. System Activities
7. Development environments required in order to deploy applications
8. Functionality executed by the software
9. Limitations and validation criteria
10. Performance issues
11. Software design constraints

4.1.2 Overview of responsibilities of Developer

Project Related responsibilities of the developer are:
To understand and Research on -

1. Application development in Android
2. Android Kernel
3. Dalvik Virtual Machine and .dex File Formats
4. Inter-process Communication in android
5. Security Model of Android
6. Permission and Privilege Escalation Attacks

7. Malware Attacks in Android and the various machine learning algorithms like K – Nearest Neighbor and J48 Decision Tree Algorithms.

8. Package installation process

9. Control flow graph searching coupled with taint checking

Other general requirements that a developer must abide to are:

1. To carefully understand the requirements.
2. To complete the project successfully and scale it depending on time
3. To carefully follow the software engineering practices.

4.2 Product Overview

Developing a system for the Android Platform that monitors the life cycle of each application for any unauthorised activity, intent and malicious behaviour using static and dynamic analysis and flags it to the user with the severity level in order to perform necessary actions. Making an application secure is also the responsibility of the developer. Our system also provides a developer’s tool which uses the inter-procedural control flow graph searching coupled with taint checking to detect capability leaks and provides the developers a graphical and tabular result which allows him to analyse the modules in order to strengthen his code.

4.3 Usage scenario

This section provides a usage scenario for the software. It is the organized information collected during requirements elicitation into use-cases.

4.3.1 User profiles (Actors)

Functionalities of various users in the system are:

1. Client: Any peer present in the system that interacts with the security tool for instructing directions and probable actions to be undertaken by the security tool on notifying user any possible security breach .Here in the system, Client is any Android Device User.

2. Android Application: It is any application (developed to be executed on Android Platform: APK) which is to be downloaded, installed or updated on Android System. Its change in state (from downloaded to installed or installed to updated or inactive to running) will trigger the security tool to monitor its behavior for probable security threats.

3. Android System: It includes the entire Android kernel, along with higher application layer services that provide support for deployment of android applications and services. It is deployed on the Phone / any device that satisfies the hardware requirements. It provides platform for further development at kernel or application layer both.
4. Category Server: It is repository for all category types in android application development. It helps to classify an application into one of the categories for further monitoring for privilege escalation or permission based malware detection.
5. Malware Detection Server: It contains a database which is checked to see if app is a malware. It also runs various algorithms on the feature vector to classify the application.
6. Android Loadable Kernel Module: It catches the system calls made by an application and logs their frequency. Other data regarding files used by the application can also be caught here.
7. Developer: He is the user of the developer’s tool. He is any individual who has developed an application and wishes to check its vulnerability.

4.3.2 Use-Cases

All use-cases for the software are presented. Description of all main Use cases using use case template is provided.

USE-CASE 1:

Sr. No	1
Use Case Name	General System Overview
Primary Actor	Client/User , System-Android
Secondary Actor	Security Tool
Pre-Condition	Android System Boot Completed System is in running state
Main Flow	1. Android Boot Completed intent starts the security tool 2. System continuously runs in the background 3. It monitors the Android System for: 3.1 Android Application Activity (download, install, update) 3.2 Privilege escalation attack 3.3 Malware Attack 4. Stop Service on system shutdown.
Post Condition	Detection and notification of any malicious behaviour by android applications to the user.

USE-CASE 2:

Sr. No	2
Use Case Name	Malware Detection
Primary Actor	Android System
Secondary Actor	Security Tool , Category and Malware Signature server
Pre-Condition	Android System Boot Completed System is in running state Android Applications in execution User launches tool for malware detection
Main Flow	1. Android Service continuously runs in the background 2. User Selects Static analysis for malware detection 2.1 Decompilation of Android Package is done 2.2 System uses Categorization and Pattern matching tools for static analysis 2.3 Report user of results. 2.4 Follow user directions 3. User Selects Dynamic Analysis for malware detection 3.1 Execute new application for 60 seconds. 3.2 Android LKM logs frequency of system calls made. 3.3 Generation of feature vector. 3.4 Feature vector sent to server side 3.5 User is notified. 4. Stop Service on system shutdown.
Post Condition	Android System is secured from malware threat. Alert Generation Malware database is updated.

USE-CASE 3:

Sr. No	3
Use Case Name	Dynamic Analysis for malware detection
Primary Actor	Android Application(to be tested)
Secondary Actor	Security Tool , Category and Malware Signature servers, Android System
Pre-Condition	Android System Boot Completed System is in running state Android Applications in execution User launches tool for malware detection
Main Flow	1. Android service continuously runs in the background 2. User Selects Dynamic Analysis for malware detection 2.1 Execute Application for minimum 60 sec. 2.2 Loadable Kernel Module (LKM) intercepts the System Calls and logs their frequency. 2.3 Detection of phone calls made and SMS sent 2.4 Generation of feature vector. 2.5 Vector sent to malware detection server. 2.6 Malware DB checked. 2.7 Algorithms run on the vector to classify the application. 2.8 Malware DB updated. 2.9 Report user of results. 2.10 Follow user directions. 3. Stop Service on system shutdown.
Post Condition	Dynamic analysis completed. Alert generation Malware database is updated.

USE-CASE 4.1:

Sr. No	4
Use Case Name	Static Analysis for Malware detection
Primary Actor	Android System, Client
Secondary Actor	Category and Malware Signature servers , Internal database
Pre-Condition	Android System Boot Completed System is in running state Android Applications in execution User launches tool for malware detection
Main Flow	1. System continuously runs in the background 2. User Selects Static analysis for malware detection 2.1 Decompilation of Android Package is done 2.2 The Category and permissions granted to the application is fetched from the internal database of the user phone. 2.3 Extract features from the classes.dex , res/ and androidmanifest.xml for static analysis. 2.4 Encrypt the features in binary / hex format or as the signature encryption. 2.5 Pattern matching with the Malware Signature server. 2.6 Report user of results. 2.7 Follow user directions 4. Shutdown the sandbox. 5. Stop Service on system shutdown.
Post Condition	Static Analysis completed. Alert generation Malware database is updated.
Sr. No	4.2
Use Case Name	Static Analysis for privilege escalation attack and capability leak detection
Primary Actor	Developer, security tool, databases
Secondary Actor	Category servers , Internal database

Pre-Condition	Developer has logged into the system Developer uploads the (.apk) package Package file has been reverse engineered to obtain the AndroidManifest.xml and source files Security tool has been launched
Main Flow	For Privilege escalation detection: Refer Use-Case 6 For capability leak detection Refer Use-Case 7
Post Condition	Static Analysis for developer completed. Alert generation for developer

USE-CASE 5:

Sr. No	5
Use Case Name	Detailed Overview of Security tool
Primary Actor	Android System, Client
Secondary Actor	Category and Malware Signature servers , Internal database
Pre-Condition	Android System Boot Completed System is in running state Android Applications in execution User launches tool for malware detection
Main Flow	1. System continuously runs in the background 2. For static Analysis : Refer Use-Case 3 and 4 3. For dynamic analysis : Refer Use-Case 2 and 4. 4. Handle privilege escalation attacks by monitoring APK Function calls and intercepting inter-process communication. 5. Stop Service on system shutdown.
Post Condition	Android System is secured from malware threat. User is notified of malicious activities. Malware database is updated.

USE-CASE 6:

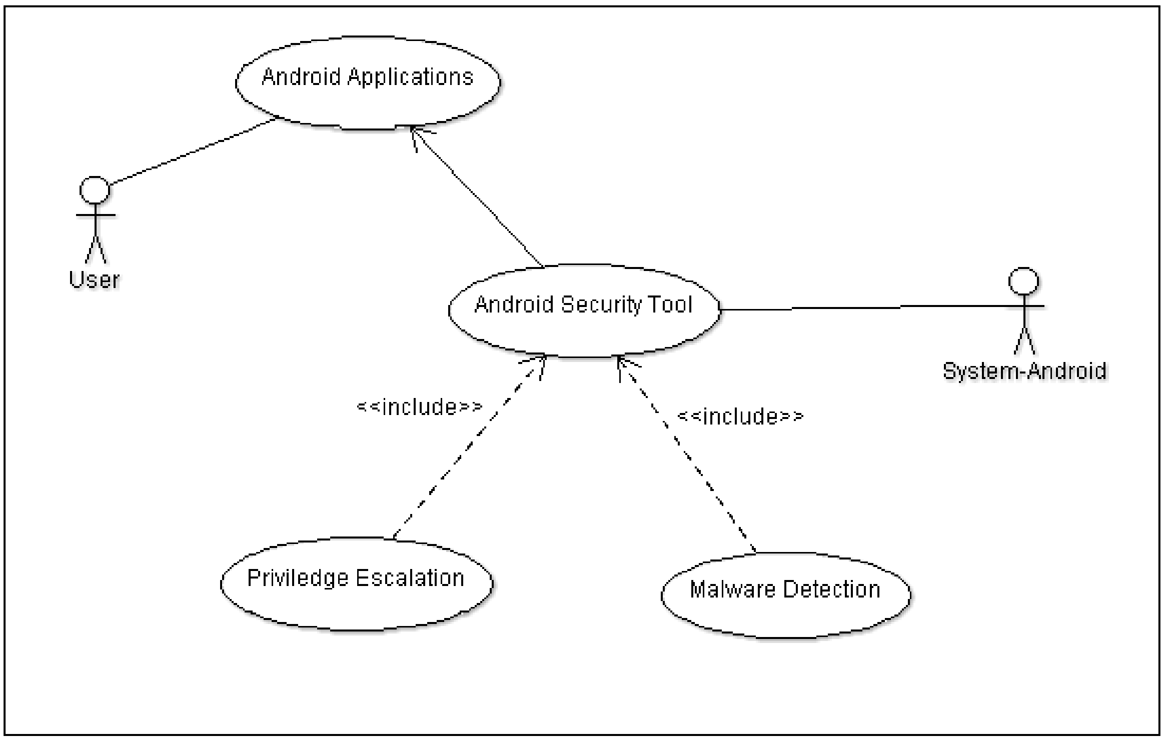
Sr. No	6
Use Case Name	Privilege Escalation attack
Primary Actor	Android App developer
Secondary Actor	Category servers , Internal database, Security tool
Pre-Condition	Developer has logged into the system Developer uploads the (.apk) package Package file has been reverse engineered to obtain the AndroidManifest.xml Security tool has been launched
Main Flow	1. Security tool continuously parses the AndroidManifest.xml 2. Checks for presence of uses permissions tag 3. Start Analysis 1) Gets in the value of the permissions present. 2) For every activity and service- a. check for permissions tag b. check if <exported> is true or false c. check for intent filter d. for every intent filter check action and data
Post Condition	Application’s vulnerability is tested Individual modules strength against privilege escalation attack is detected If prone to attack the particular code is highlighted to developer in order to perform necessary action

USE CASE 7:

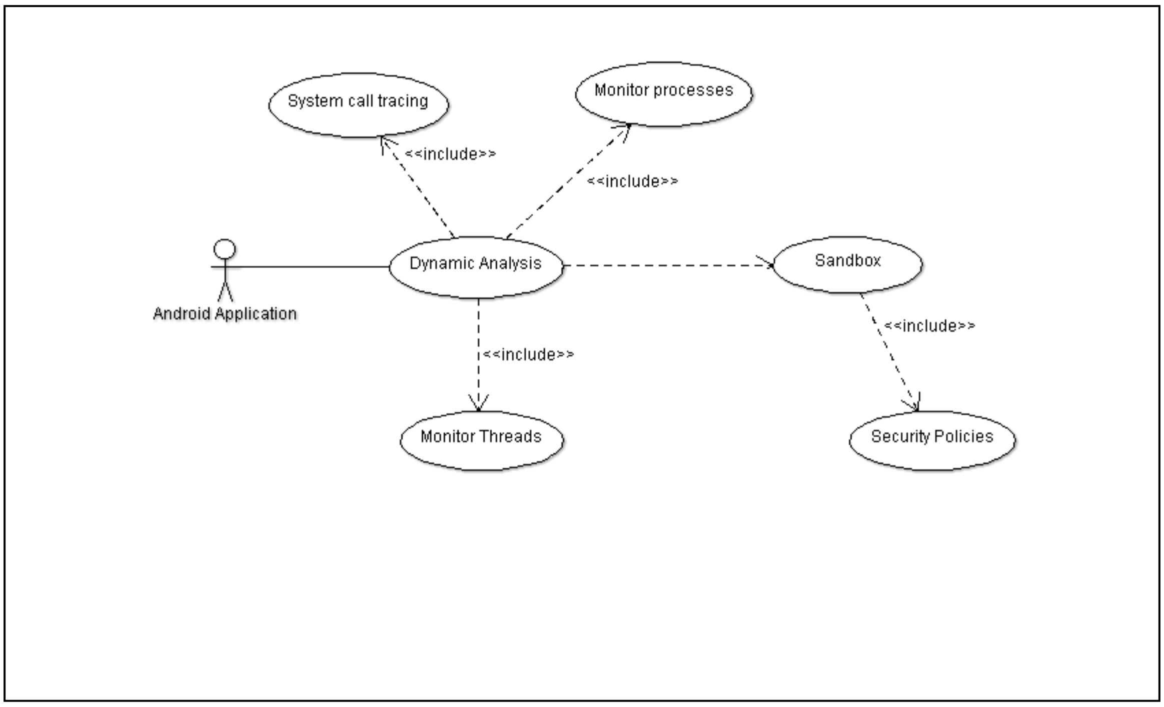
Sr. No	7
Use Case Name	Capabilities Leak Detection
Primary Actor	Android App developer
Secondary Actor	Category servers , Internal database, Security tool
Pre-Condition	Developer has logged into the system Developer uploads the (.apk) package Package file has been reverse engineered to obtain the source files. Security tool has been launched
Main Flow	1. Security tool reads the source files 2. Checks for presence of uses permissions tag 3. Start Analysis 4. Detect components a. for every component generate control flow graphs 5. Visit every node in the graph a. for every node detect probable sink b. check for tainted source
Post Condition	Application's vulnerability is tested If prone to attack the particular code is highlighted to developer in order to perform necessary action

4.3 Use Case View

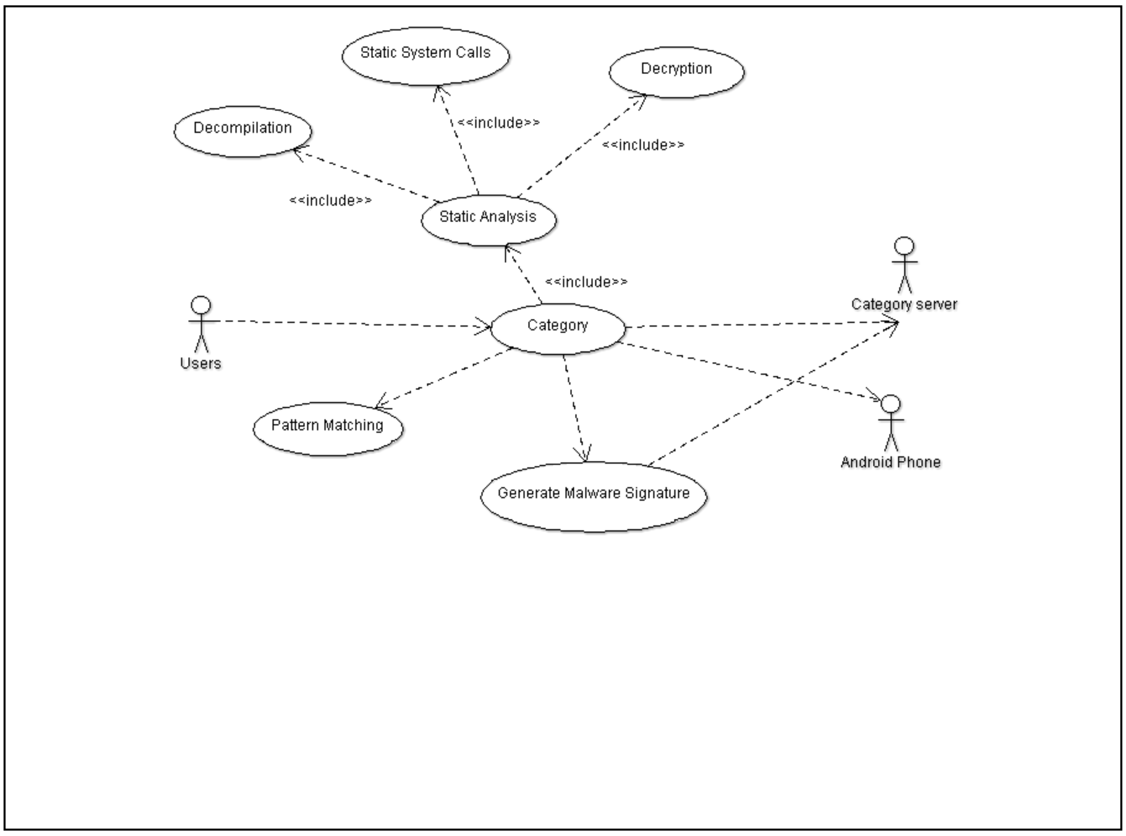
Use Case 1:



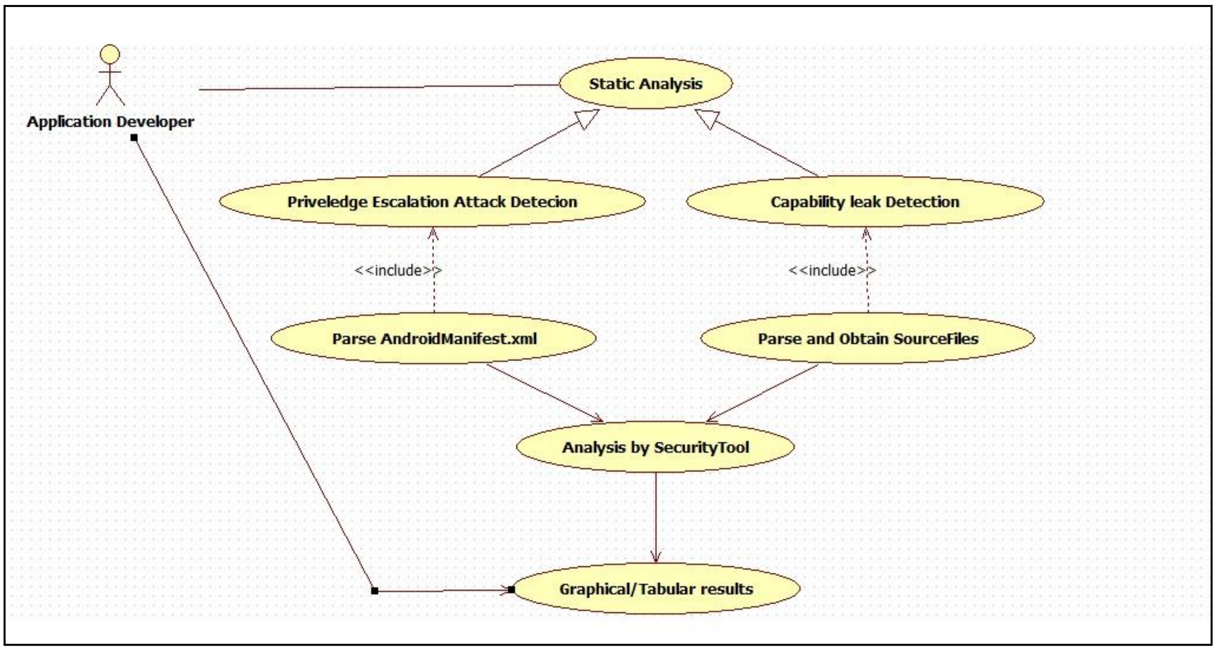
Use Case 2:



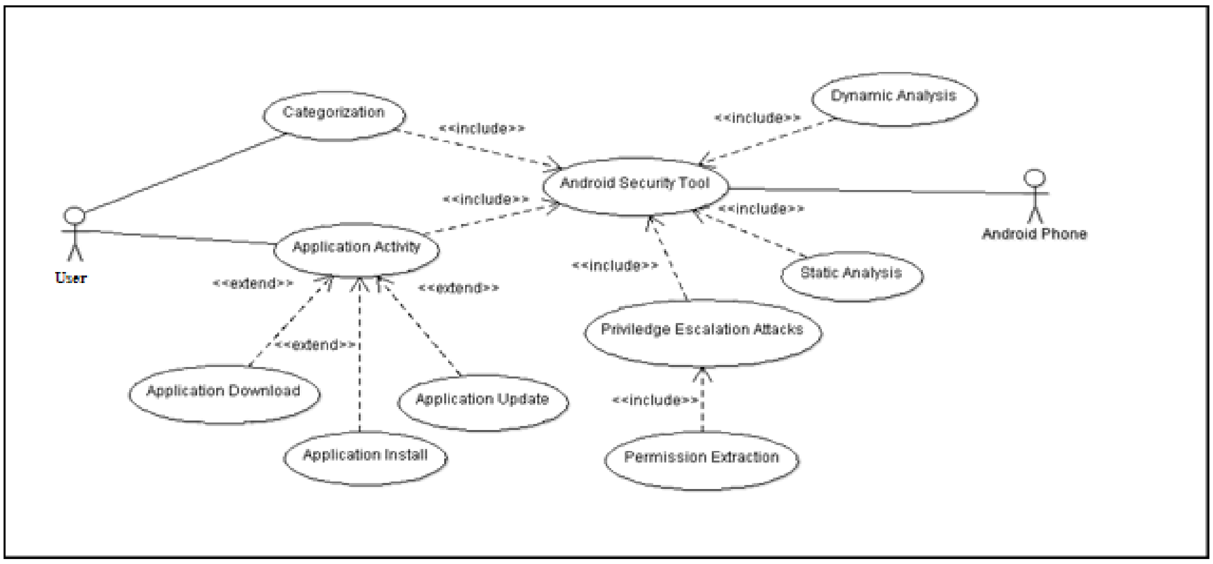
Use case 3:



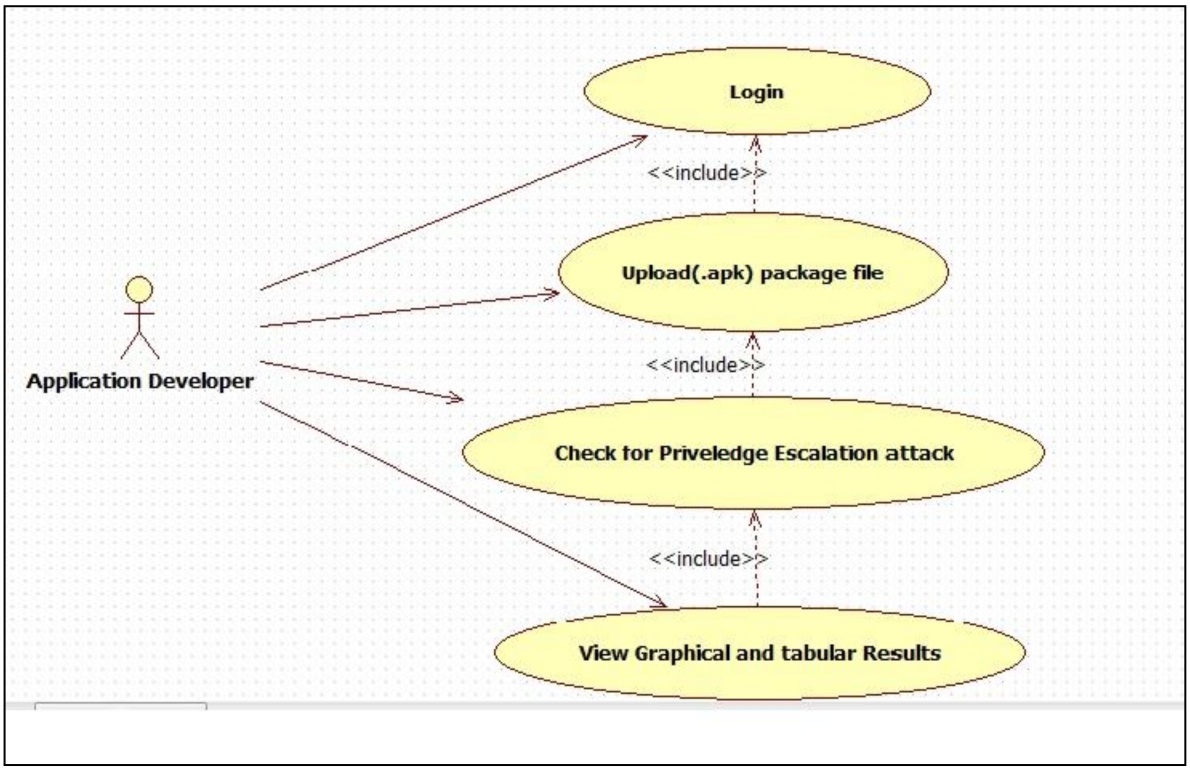
Use case 4:


Application Developer
Static Analysis
Priveledge Escalation Attack Detecion
Capability leak Detection
<<include>>
<<include>>
Parse AndroidManifest.xml
Parse and Obtain SourceFiles
Analysis by SecurityTool
Graphical/Tabular results

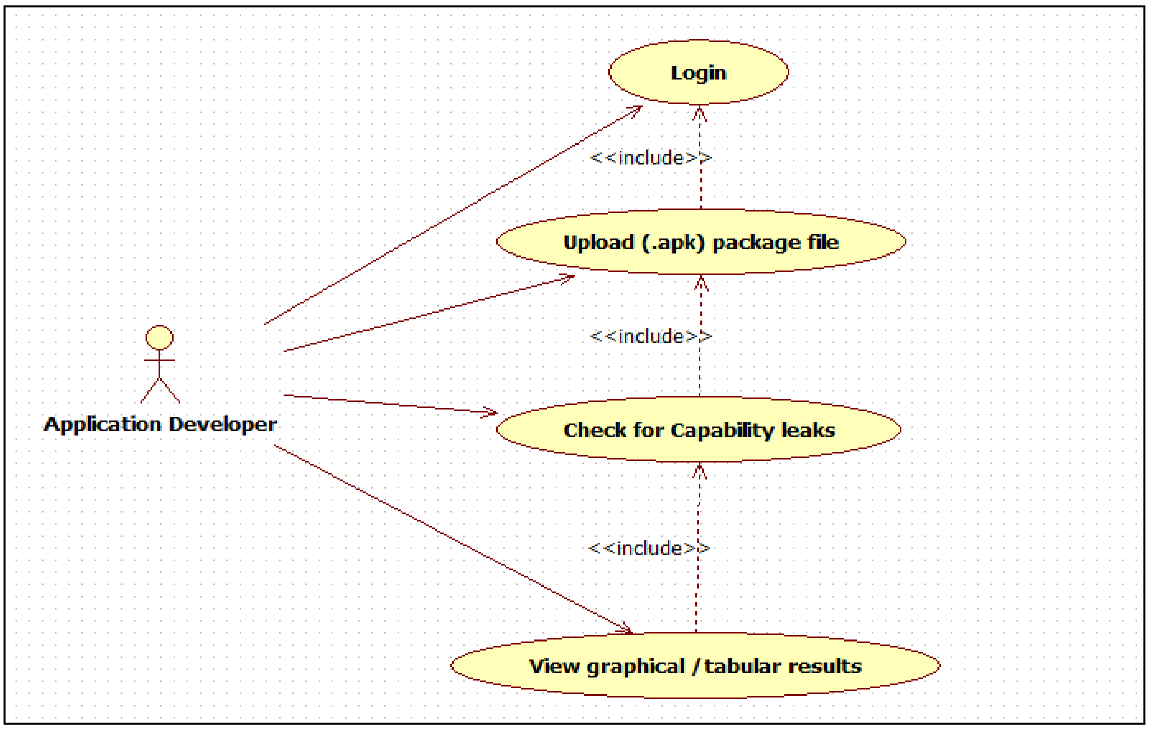
Use case 5:



Use case 6:


Login
<<include>>
Upload(.apk) package file
Application Developer
<<include>>
Check for Priveledge Escalation attack
<<include>>
View Graphical and tabular Results

Use case 7:


Login
<<include>>
Upload (.apk) package file
<<include>>
Application Developer
Check for Capability leaks
<<include>>
View graphical / tabular results

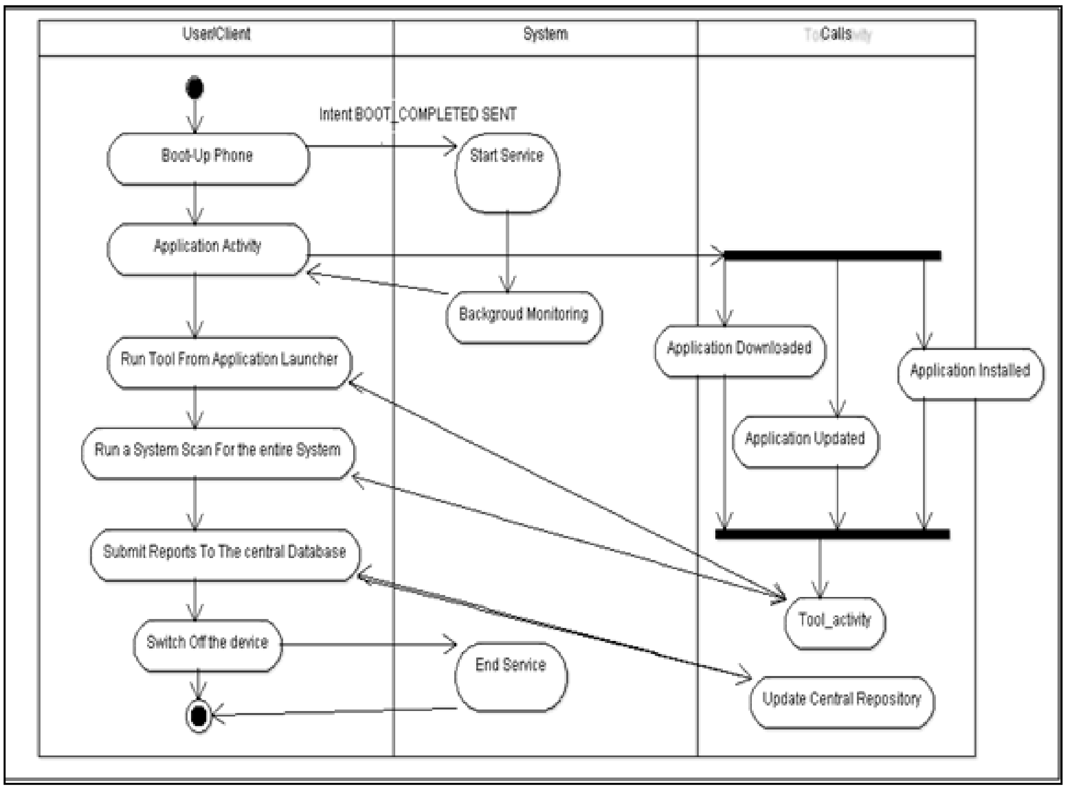
4.5 Functional Model and Description

A description of each major software function, along with **data flow** (structured analysis) **is presented.**

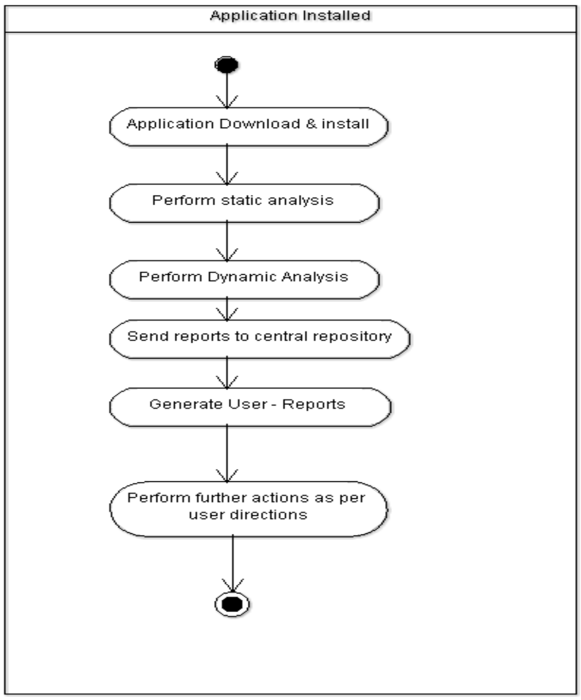
4.5.1 Flow diagram

A diagram showing the flow of information through the function and the transformation it undergoes is presented.

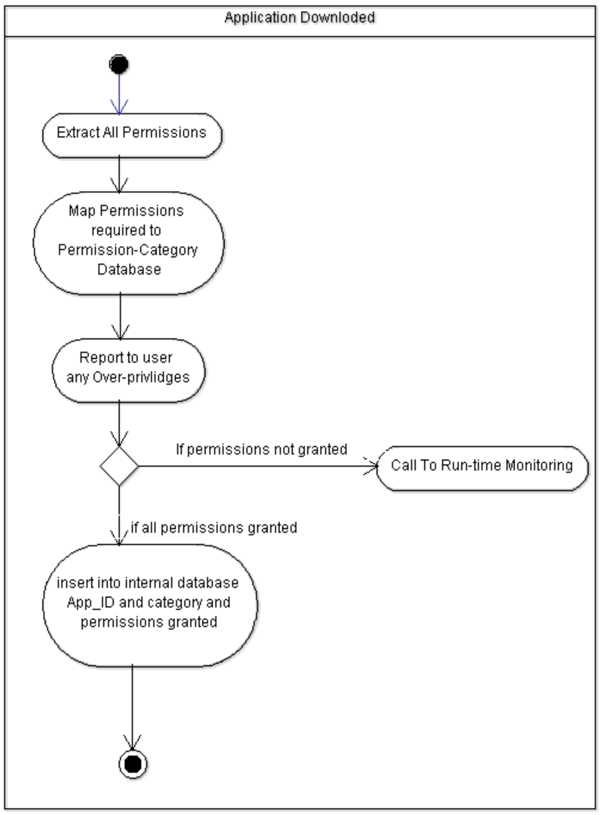
Flow Diagram 1:



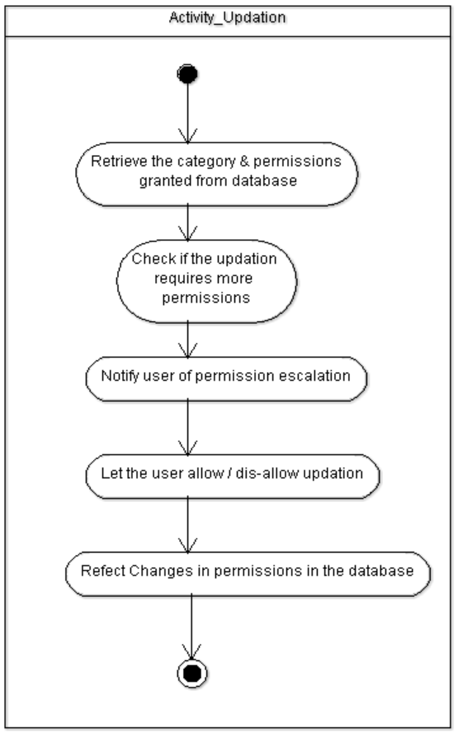
Flow Diagram 2:


Application Installed
Application Download & install
Perform static analysis
Perform Dynamic Analysis
Send reports to central repository
Generate User - Reports
Perform further actions as per user directions

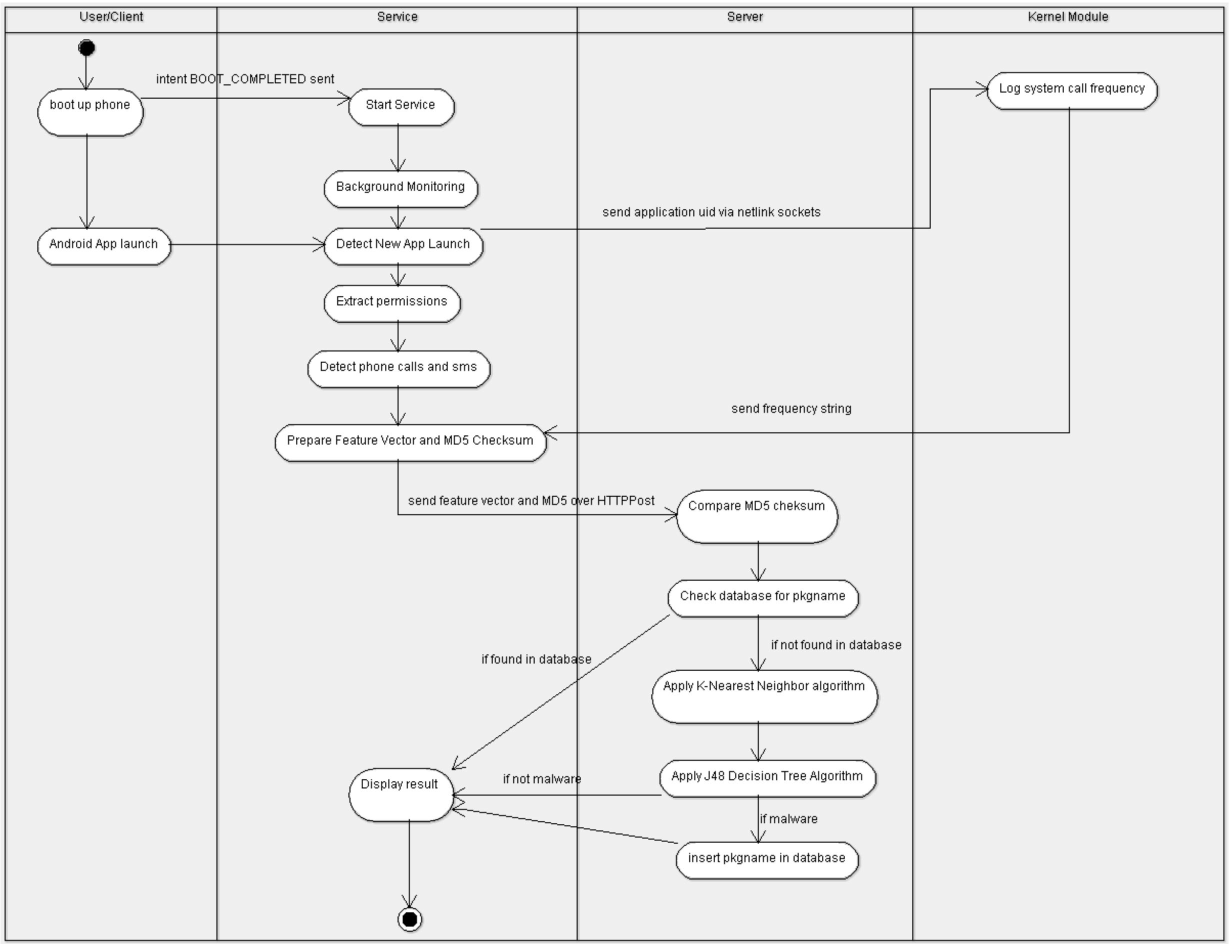
Flow Diagram 3:


Application Downloded
Extract All Permissions
Map Permissions required to Permission-Category Database
Report to user any Over-privlidges
If permissions not granted
Call To Run-time Monitoring
if all permissions granted
insert into internal database App_ID and category and permissions granted

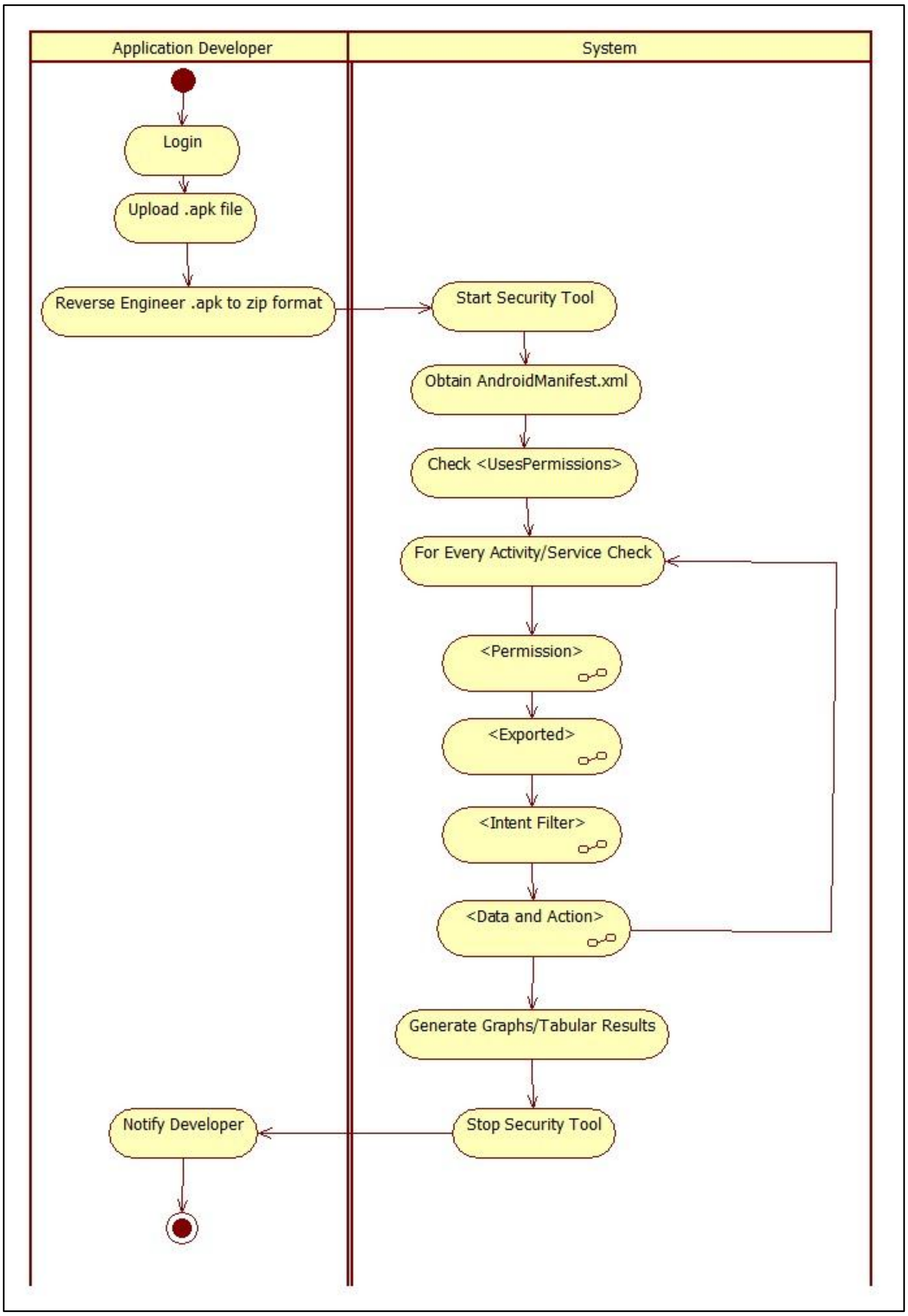
-Flow Diagram 4:


Activity_Updation
Retrieve the category & permissions granted from database
Check if the updation requires more permissions
Notify user of permission escalation
Let the user allow / dis-allow updation
Refect Changes in permissions in the database

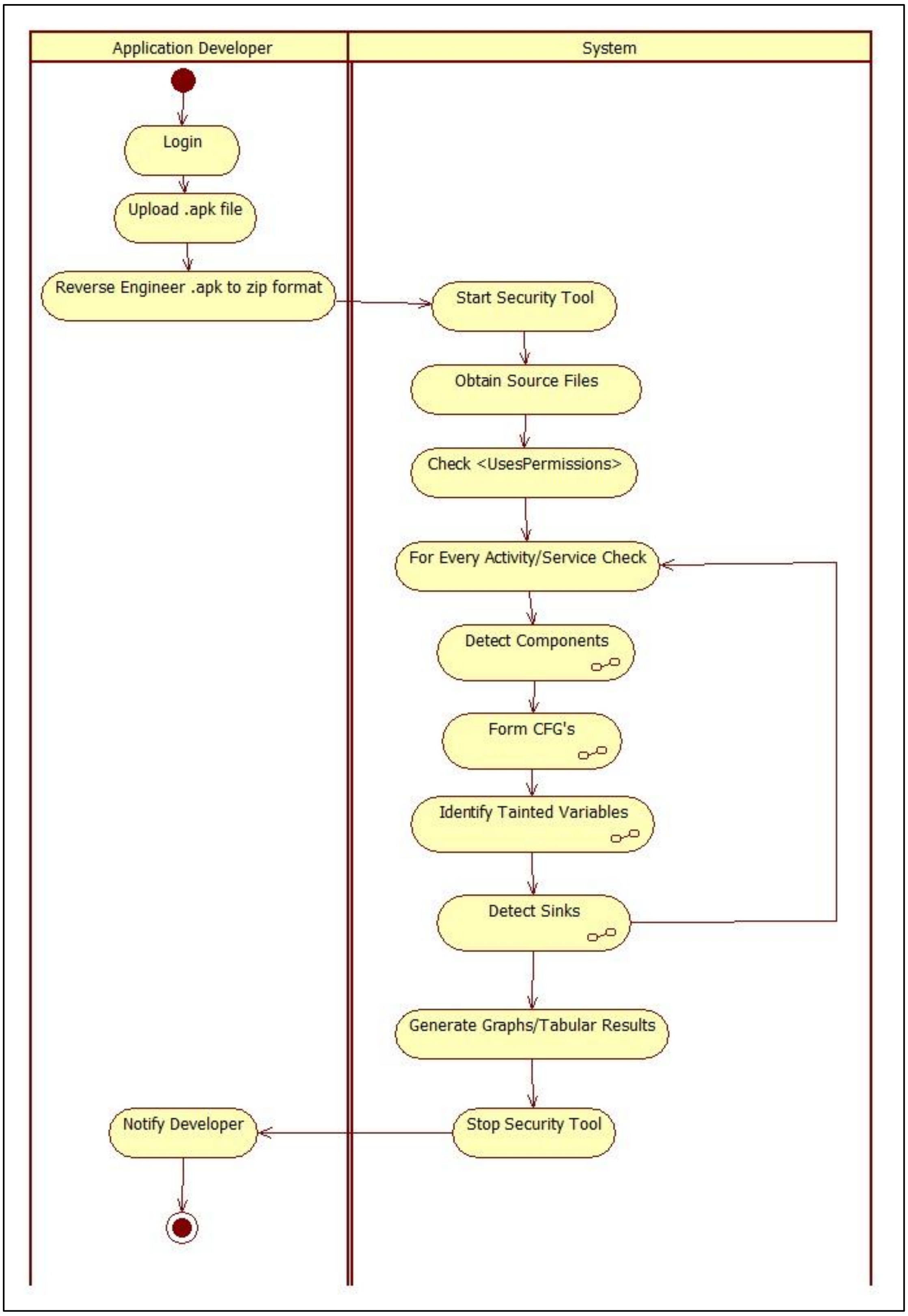
Flow Diagram 5:


User/Client
Service
Server
Kernel Module
boot up phone
intent BOOT_COMPLETED sent
Start Service
Log system call frequency
Background Monitoring
send application uid via netlink sockets
Android App launch
Detect New App Launch
Extract permissions
Detect phone calls and sms
send frequency string
Prepare Feature Vector and MD5 Checksum
send feature vector and MD5 over HTTPPost
Compare MD5 cheksum
Check database for pkgname
if found in database
if not found in database
Apply K-Nearest Neighbor algorithm
Display result
if not malware
Apply J48 Decision Tree Algorithm
if malware
insert pkgname in database

Flow Diagram 6:



Flow Diagram 7:



4.5.3 Non Functional Requirements

Performance requirements:

1. The Security tool developed must be quick to launch on boot up of the system.
2. The system, if deployed on cloud, must support multiple devices at the same time.
3. The security tool runs continuously in background and hence must not include large overhead in terms of space and time.

Safety requirements:

1. In case user launches many applications, then monitoring all the application's behavior may result in a very heavy service. Various algorithms for load balancing are also required.
2. Limits on the no. of applications to be served at a particular instance of time.
3. Internet Connection should be made available for querying and updating the database.
4. Decision of whether to make the system display more false positives rather then less false negatives has to be weighed.

Security requirements:

1. The system should be secured and easy to access.
2. The client should not modify the data stored in any database directly.
3. In case of access to central repository, all the clients should be authenticated.
4. All update queries must be authenticated and analyzed carefully by the administrator at the site of central database.

Software quality attributes:

1. Availability:

1.1 System may be installed on each of the Android Systems as a background service.

1.2 Deployment of entire system on cloud, so that users can directly submit queries to the distributed system.

1.3 Both of the above solutions are reliable. Local installation of system must be compact and lower in capacity. Central repository can be made large, efficient and scalable with increasing requirements.

2. Modifiability:

1.1 System in both cases is modifiable. Irrespective of stem location, databases are made distributed in nature and hence easily modifiable.

3. Reliability:

1.1 Local systems are difficult to test by users. Special tools must be deployed to handle testing locally.

1.2 Central system can be tested by developers at central site with lesser efforts.

4.5.4 Design Constraints

4.5.4.1 Categorization of the applications

4.5.5 Software Interface Description

The software interface(s)to the outside world is(are) described. The requirements for interfaces to other devices/systems/networks/human are stated. This system will provide require least user interaction and provide very friendly user interface. Most of the time the system will run as service in background. Notifications will be provided on completion of tasks. User has to perform simple directing role with the tool to ensure security. Developer’s tool needs the developer to upload the .apk package file. The tool performs analysis and gives graphical and tabular results easy to understand.

4.6.0 Behavioral Model and Description

A description of the behavior of the software is presented.

4.6.1 Description for software behavior

A detailed description of major events and states is presented in this section.

4.6.1.1 Events

1. BOOT_COMPLETED Intent
2. Application Installed Intent
3. Application Update Completed Intent
4. Application Launched Event
5. Sampling timer of 60 seconds
6. Success of a “do” operation
7. Failure of a “do” operation
8. Detection of malware.
9. Notification to the user.

4.6.1.2 States

A listing of states (modes of behaviour) that will result as a consequence of events is presented.

1. Boot_up
2. Start Security tool
3. Priviledge Escalation Attack Check
4. Allow / Disallow update
5. Permission Handler
6. Installation
7. Sampling
8. Execution
9. Static Analysis
10. Dynamic Analysis
12. Success in State
13. Failure in state
14. Start
15. End

4.6.2 State Transition Diagrams

Depicts the overall behavior of the system.

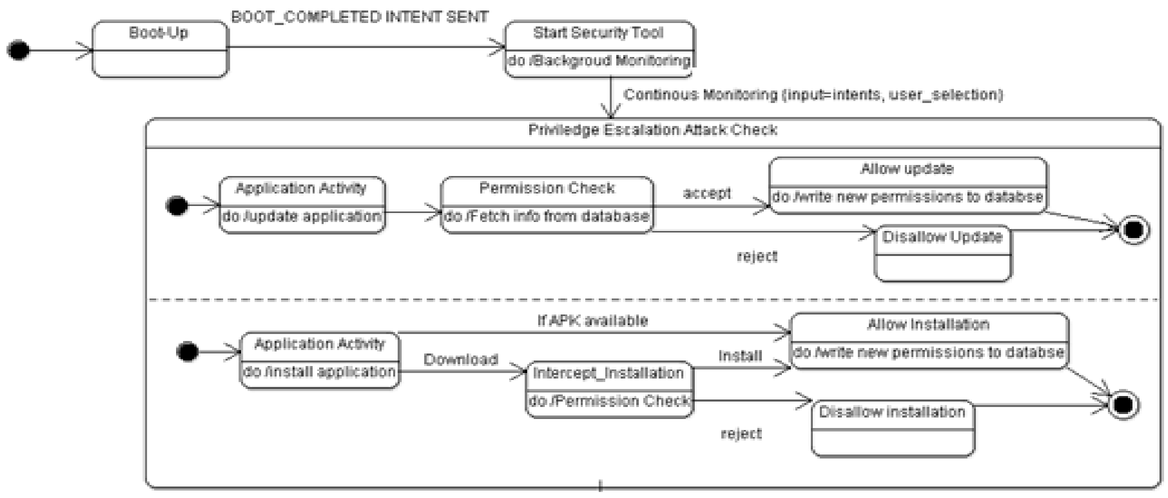
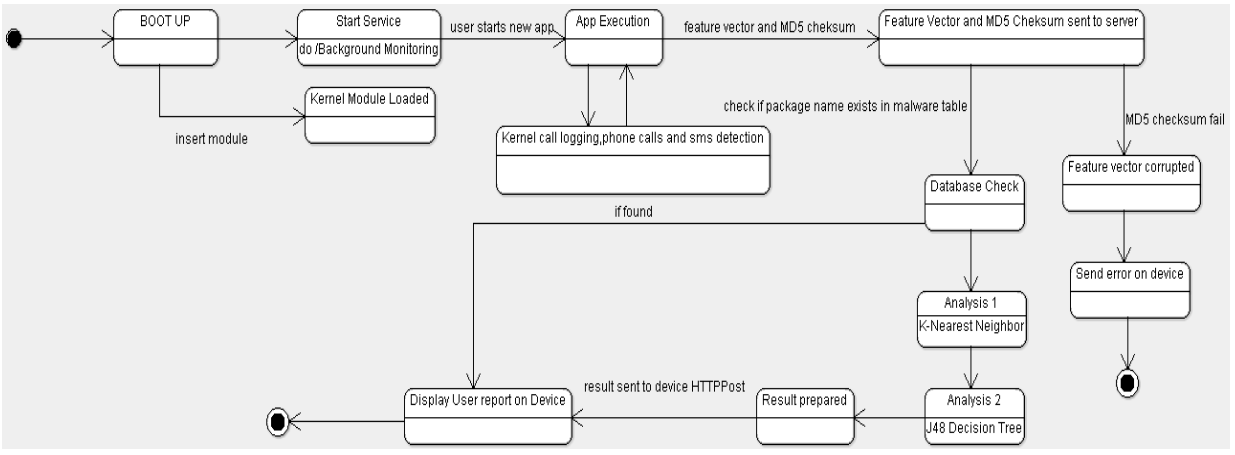
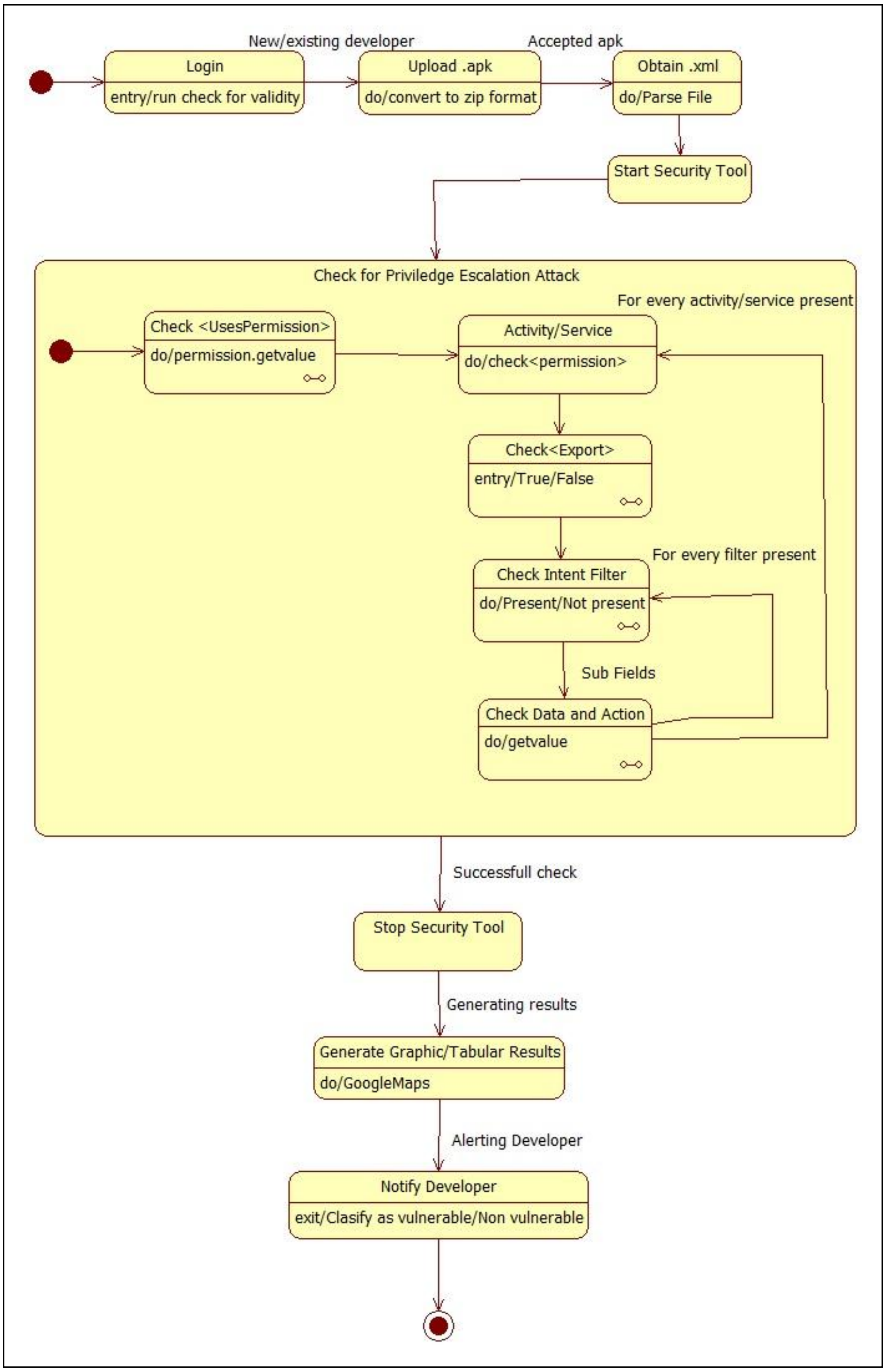


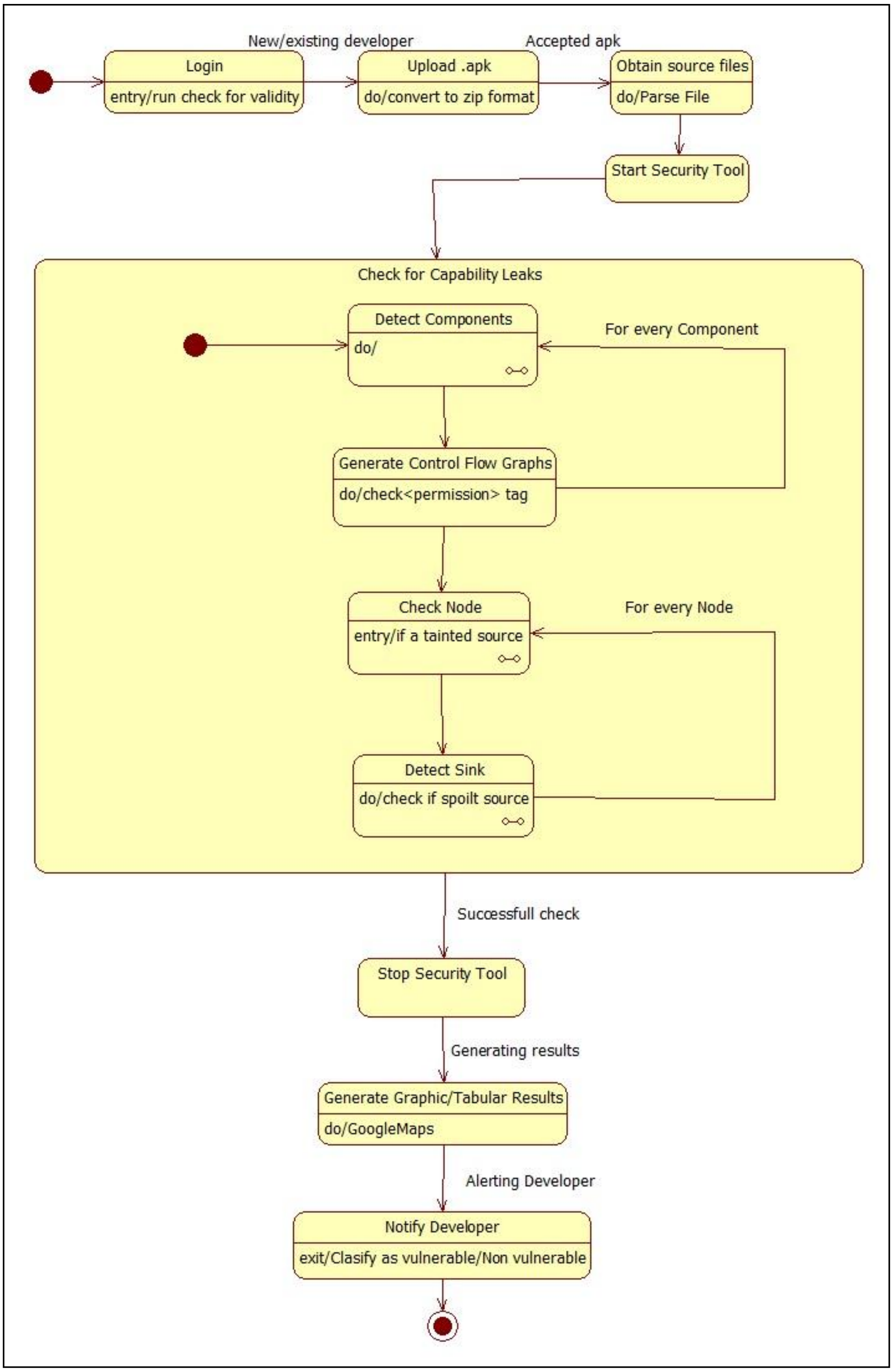
Fig. State Diagram 1: Static Analysis



BOOT UP
Start Service
do /Background Monitoring
user starts new app
App Execution
feature vector and MD5 cheksum
Feature Vector and MD5 Cheksum sent to server
Kernel Module Loaded
insert module
Kernel call logging,phone calls and sms detection
check if package name exists in malware table
MD5 checksum fail
Feature vector corrupted
Database Check
if found
Send error on device
Analysis 1
K-Nearest Neighbor
Analysis 2
J48 Decision Tree
Result prepared
result sent to device HTTPPost
Display User report on Device

Fig. State Diagram 2: Dynamic Analysis


New/existing developer
Accepted apk
Login
entry/run check for validity
Upload .apk
do/convert to zip format
Obtain .xml
do/Parse File
Start Security Tool
Check for Priviledge Escalation Attack
For every activity/service present
Check <UsesPermission>
do/permission.getvalue
Activity/Service
do/check<permission>
Check<Export>
entry/True/False
For every filter present
Check Intent Filter
do/Present/Not present
Sub Fields
Check Data and Action
do/getvalue
Successfull check
Stop Security Tool
Generating results
Generate Graphic/Tabular Results
do/GoogleMaps
Alerting Developer
Notify Developer
exit/Clasify as vulnerable/Non vulnerable


New/existing developer
Accepted apk
Login
entry/run check for validity
Upload .apk
do/convert to zip format
Obtain source files
do/Parse File
Start Security Tool
Check for Capability Leaks
Detect Components
do/
For every Component
Generate Control Flow Graphs
do/check<permission> tag
Check Node
entry/if a tainted source
For every Node
Detect Sink
do/check if spoilt source
Successfull check
Stop Security Tool
Generating results
Generate Graphic/Tabular Results
do/GoogleMaps
Alerting Developer
Notify Developer
exit/Clasify as vulnerable/Non vulnerable

4.7 Restrictions, Limitations, and Constraints

Special issues which impact the specification, design, or implementation of the software are:

1. Categorization of the application is user dependent
2. Corrective actions are user dependent.
3. Newer malware may go undetected if absent in database.
4. Malware Detection accuracy depends on the training set on server.

4.8 Validation Criteria

4.8.1 Classes of tests

White Box Testing

This ensures that the modules used in the construction are functional and robust to some degree.

Black Box Testing

In this stage there will be testing for its functionality, performance and usability, Robustness and security.

Alpha Testing

Developer does this testing during development phase and it will be completed before delivering the software to the beta testers. First we will test the software using white box techniques. Additional inspection is then performed using black box.

Beta Testing

If a major bug is reported only that bug will be rectified before the release of the software.

Test Case 1

Test Cases	Expected Output	Actual Output
Extract Permissions	All the requested permissions by an application must be extracted from its androidManifest File.	All the requested Permissions by an application will be extracted from its androidManifest File except in cases of usage of wrong tags.
Categorize	The application must be assigned category from the set of pre-defined categories.	The application is assigned category, the first time it is downloaded / installed to the device by the user.
Intercept Downloading and Installation	The automatic installation of an application by the Package Installer must be hooked.	The probability of managing to hook between installing and downloading might not be allowed.

Test Description	To decompile apk and extract permissions
Steps	Decompile the application apk using dex2jar. Extract the permissions from the AndroidManifest.xml
Expected Results	Permissions given to the Android application are extracted successfully.

Test Case 2:

Test Case Name	Permission Checker
Test Description	To check if mapping of category and permissions is compatible
Steps	1. Let user select the category of the application from a set of values. 2. Use the extracted permissions from the androidManifest.xml 3. Map the category – permissions requested to the database. 4. Check for compatibility.
Expected Results	1. The developer should be able to determine the validity and authenticity of the repository. 2. Granting Permissions should be safe.

Test Case 3:

Test Case Name	Static Analysis
Test Description	Checking for malicious activities before application is installed on the Android device.
Steps	1. Android application is decompressed. 2. Main activity that can be launched is extracted from the manifest file. 3. Classes.dex is decompiled into a Java typical folder hierarchy containing files with easily parsed to pseudo code. 4. Finally disassembled code is scanned for suspicious patterns.
Expected Results	Some of the commonly known malicious patterns are detected before installation of the application and the users are warned.

Test Case 4:

Test Case Name	Dynamic Analysis
Test Description	Checking for malicious activities after the Android application is installed on the device
Steps	1. New application is executed for minimum of 60 seconds. 2. Android LKM catches system calls and logs their frequency. 3. Detection of phone calls made and SMS sent. 4. Generation of feature vector for analysis. 5. Feature vector is sent to server where malware db is first checked and then K-Nearest Neighbor and J48 decision tree algorithms are used to classify application. 6. User is notified.
Expected Results	Any malicious activity is to be detected during runtime is flagged to the user.

Test Case 6

Test Description	Checking for possible privilege escalation attack in the developed application
Steps	1. Application androidmanifest.xml is parsed 2. Manifest is checked for presence of permissions tag if any. 3. Every activity or service is then checked for intent filters properties, exported tags and the data and action in the filters. 4. Results are then analysed and graphical outputs are generated 5. Any attempt of privilege escalation attack if any is detected during this analysis phase
Expected Results	Graphical charts and tabular data showcasing the vulnerable sections of code.

CHAPTER 5

DETAIL DESIGN DOCUMENT

Components for the software.

5.0 Introduction

The purpose of the project "Android Security Based Tools" is to create a full-fledged security ensuring system for the Android platform. The document provides the details about the design of the tool such as data design, architectural and component design, program structure, user interface design and restrictions and constraints of the tool.

5.1 Architectural design

A description of the program architecture is presented along with System and Subsystem Diagram.

Overall System:

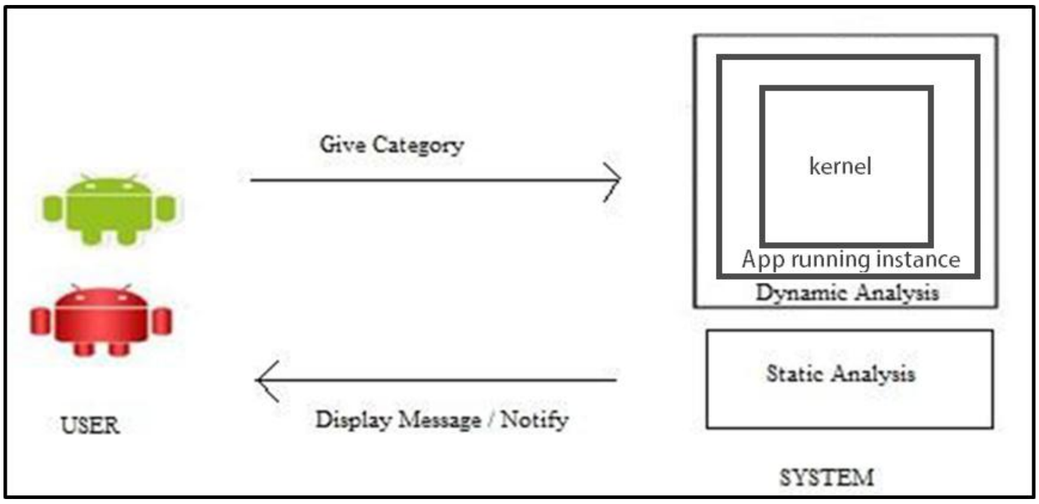
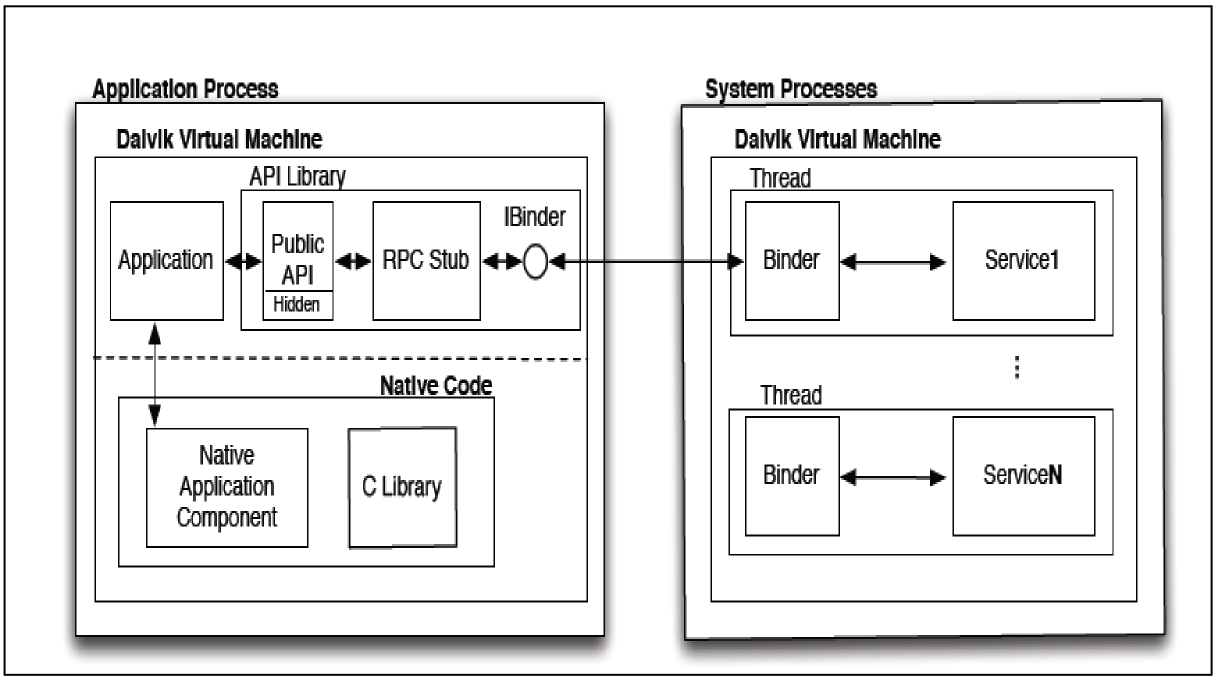


FIGURE 5.2.1

Sub-System 1:

The architecture of the Android platform. Permission checks occur in the system process.



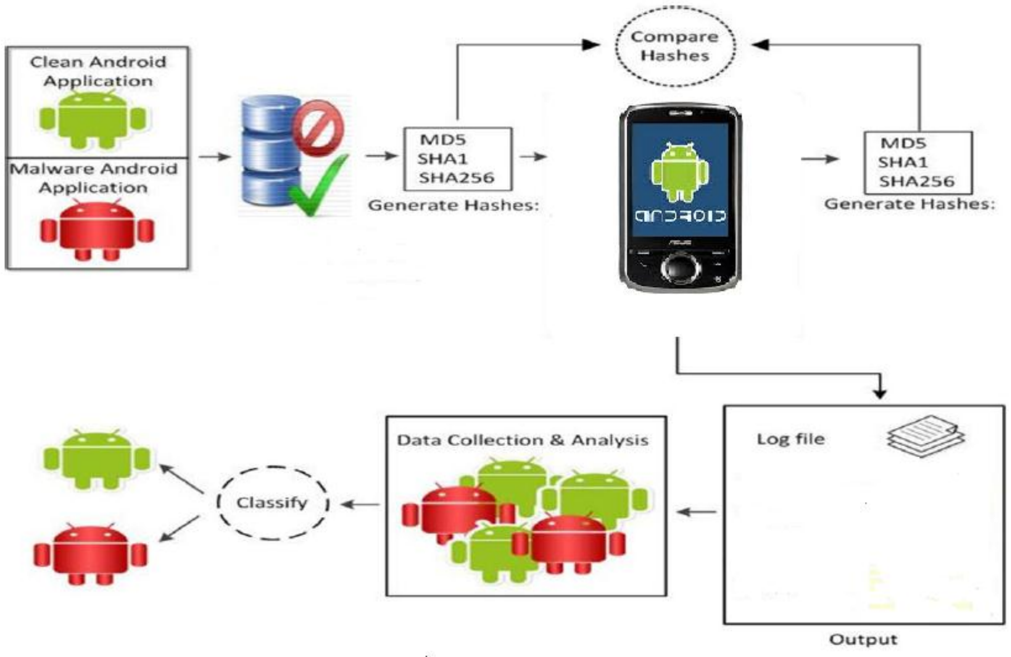


FIGURE: 5.2.3

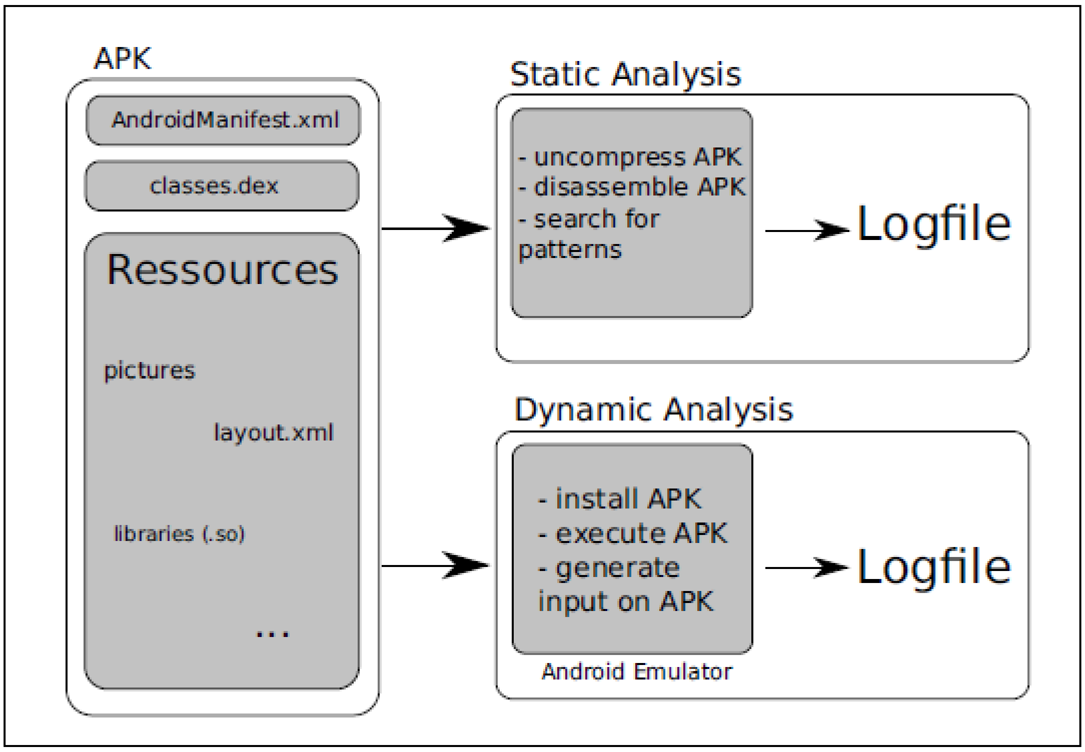
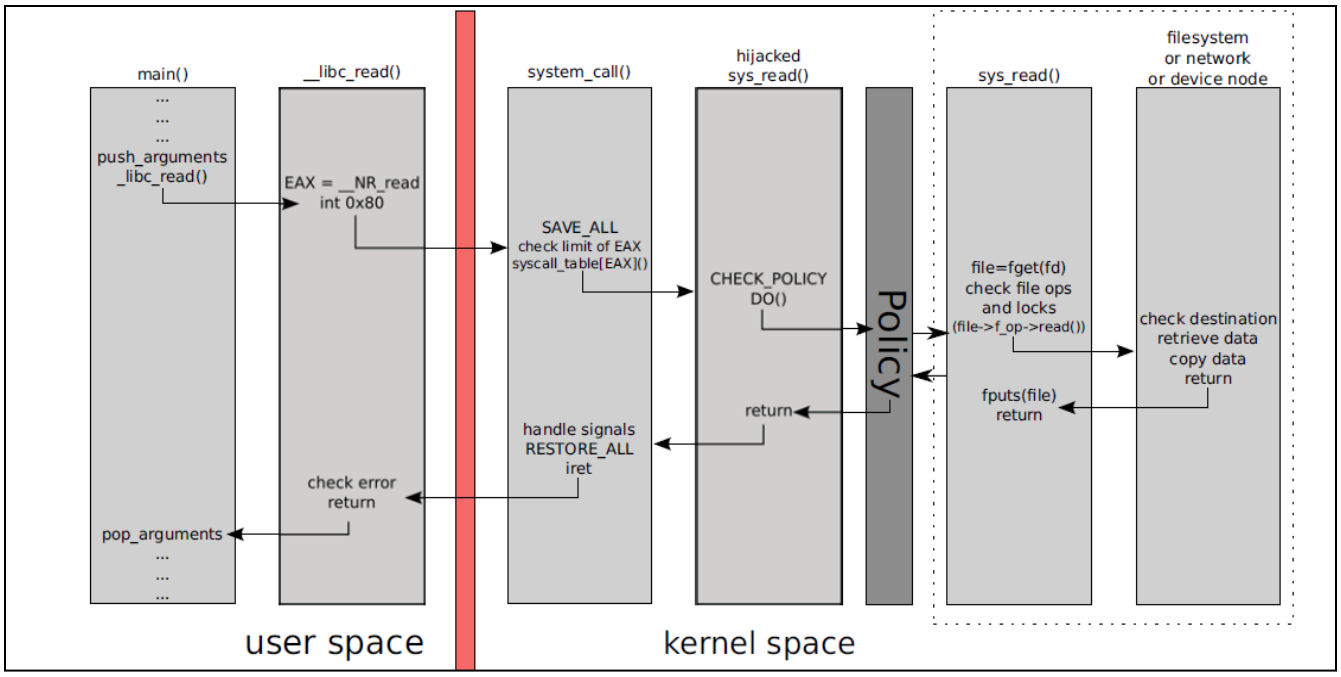


FIGURE 5.2.4: Steps involved in performing a hijacked read () system call.



5.3.0 Data design [using Appendices A and B]

A description of all data structures including internal, global, and temporary data structures, database design (tables), file formats.

5.3.1 Internal software data structure

Data structures that are passed among components the software are described.

1. Class: Category

Description: Category class models the category of the application downloaded or installed. Category may be derived by the features / behavior of the application or may be specified by the user at install time. Category class is further used for mapping with the standard Category—Permissions database. Also the category and permissions are also stored in the private database of the system for future references during updating privilege escalation

Attributes: Category <string> , Category_ID <number> , Family <String>

2. Class: Permissions

Description: The Permission class models the each permission required by the application at install time. It is referenced by the Category—Permission database.

Attributes: Permission<string>, Permission_ID<number>, Threat_Type<string>, Threat_level<number>

Threat type: {Normal permissions,Dangerous Permissions,System,Signature}

3. Feature Vector

Description: It is passed from the service to the server. Based on this vector the classification is done.

Attributes: featurestr<String>

5.3.2 Global data structure

Data structured that are available to major portions of the architecture are described.

1 Clean Applications Collection:

Description: A collection of clean applications used for the purpose of testing our security tool.

2 Malicious Applications Collection:

Description: A collection of malicious applications used for the purpose of testing our security tool.

5.3.4 Database description

Database(s) / Files created/used as part of the application is (are) described.

5.3.4.1 Malware Signatures database

Description: It contains the mapping of a malware to its signature in hex or binary or any other format. It is used for malware detection in static and dynamic analysis.This is a huge database that needs to be continuously updated to include more and more malwares as and when detected. This database can be deployed on the cloud for distributed use by all users. Also the users can also submit functionalities that appear to be malicious but not notified by the tool. This would help in updation of this database.

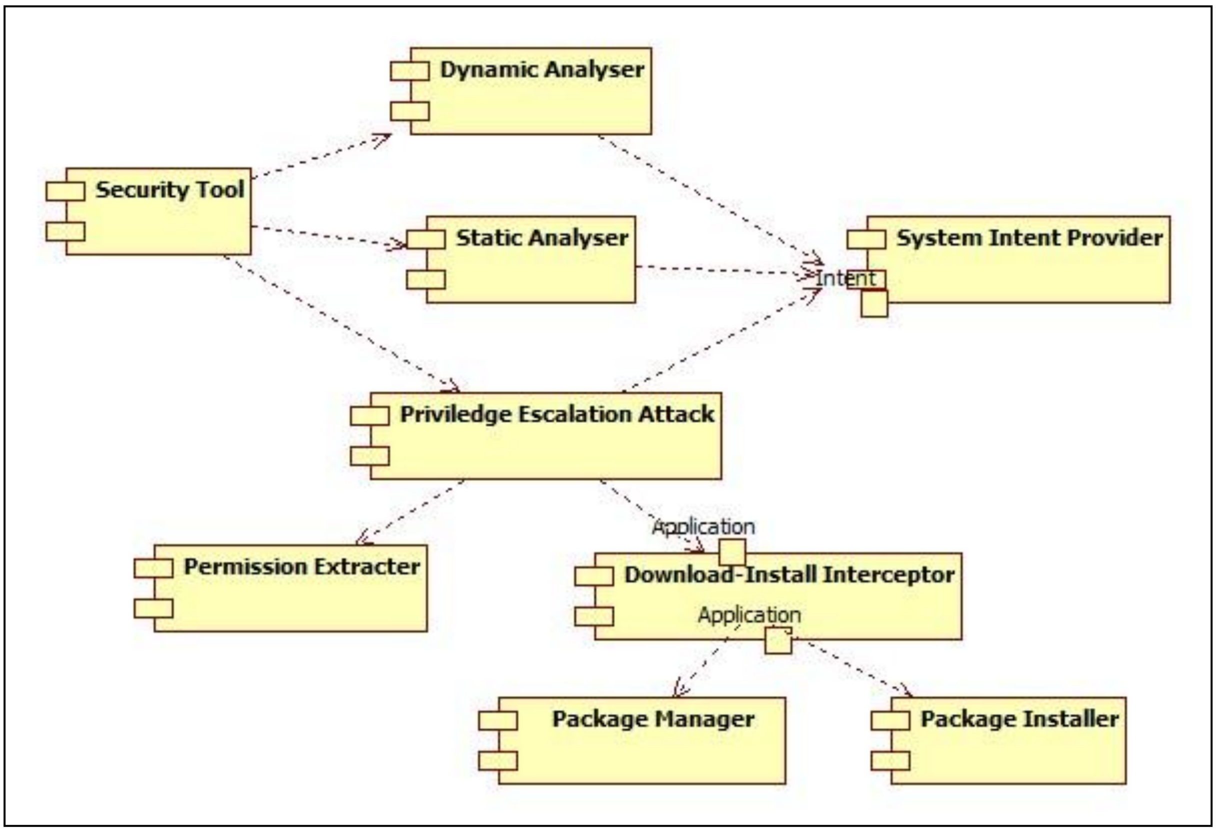
An example feature vector for malware application retrieved from its Android application package file

```
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0
,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,0,1,0,0,0,0,0,1,0,0,1,1,0,0,0,
0,0,0,1,0,1,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0
,0,0,0,1,0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,

4.156783, malware
```

5.4.0 Component design

Component diagram for the system:



Class diagram for the System:

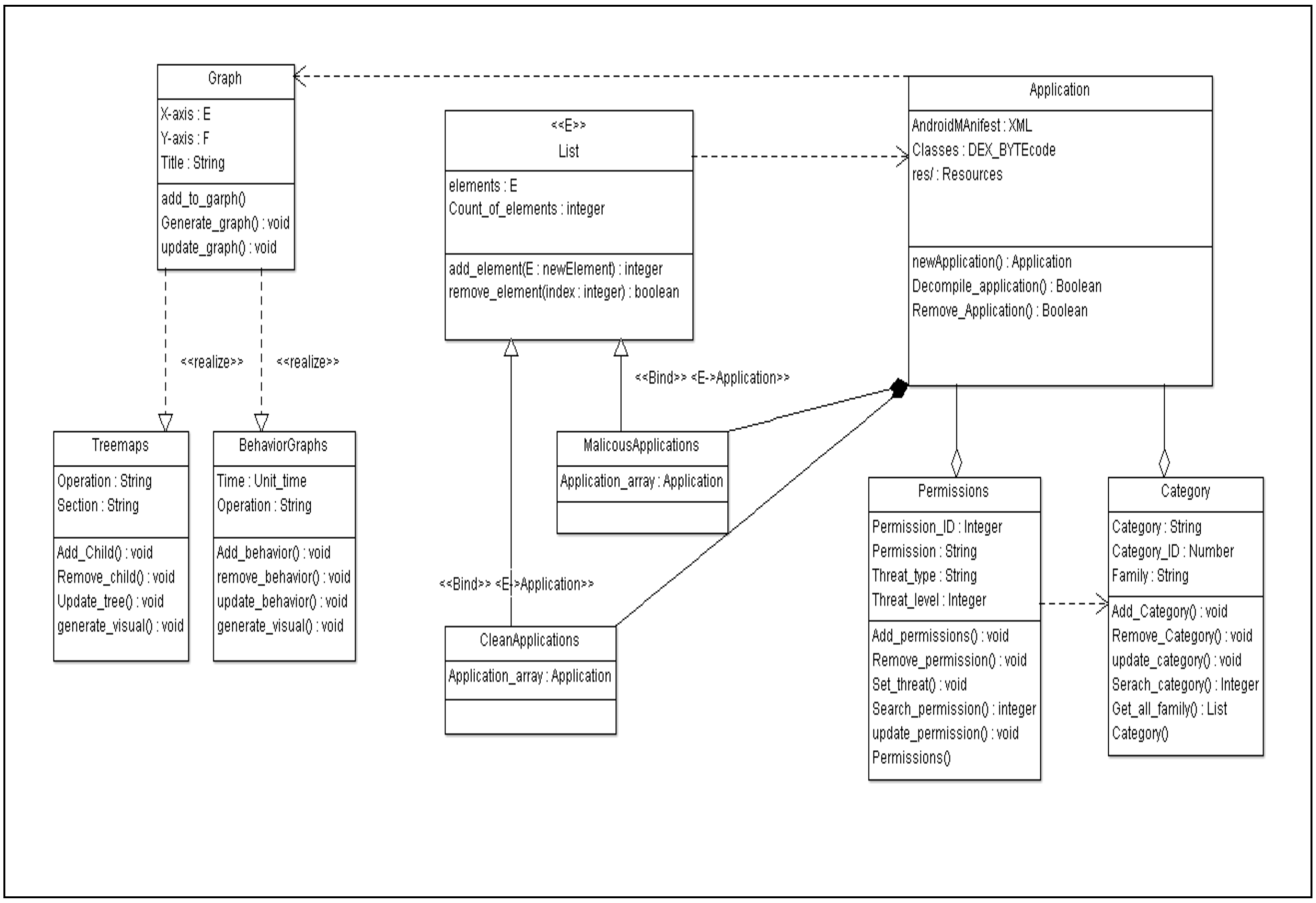


FIGURE: 5.4.1

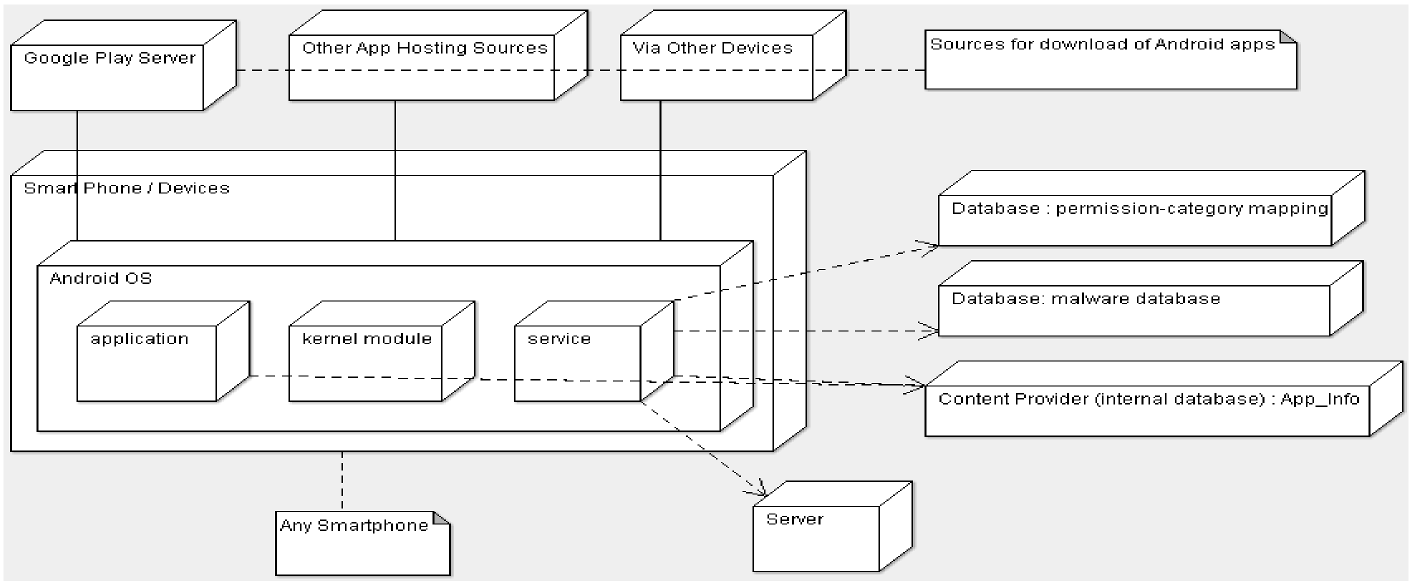

Google Play Server
Other App Hosting Sources
Via Other Devices
Sources for download of Android apps
Smart Phone / Devices
Android OS
application
kernel module
service
Database : permission-category mapping
Database: malware database
Content Provider (internal database) : App_Info
Any Smartphone
Server

FIGURE 5.4.2: DEPLOYMENT DIAGRAM

5.4.1 Program Structure

A description of the program structure/Design Model chosen for the application is presented.

The main components of the system are:

1. Security Tool
2. Dynamic Analyzer
3. Static Analyzer
4. Permission Extractor
5. Download-Install Interceptor
6. System Intent Provider
7. Android Kernel Module
8. Privilege Escalation attack tool detection for developer
9. Capability leak detection tool for developer

5.4.2 Description for Component

A description of each software component contained within the architecture is presented.

5.4.2.1 Security Tool :

It depicts the entire system that ensures a full-fledged security for the android platform ranging from android permissions and privilege escalation to malware detection.

5.4.2.2 Dynamic Analyzer:

It is a sub-component of the Security tool system. It is basically an Android service. It detects new application launches and analyses it. It detects whether the app made any phone calls or sent any SMS and generates a feature vector which is then sent to the server.

5.4.2.3 Static Analyzer:

It is a sub-component of the Security tool system. It looks for malware patterns from the android package statically using clustering, feature extraction and pattern recognition.

5.4.2.4 Permission Extractor & Checker:

It is a sub-component of the Security tool system. It extracts permissions from the application during its installation, stores the permissions internally for future use. Later during IPC , these permissions are referred to by the component for privilege escalation attacks.

5.4.2.4 Download-Install Interceptor:

It is a component that will prevent the Package Manager and Package Installer from directly installing the application to the phone after it is downloaded. It will intercept at the beginning of the installation process to allow security checks to execute.

5.4.2.5 System Intent Provider

It is a component internal to the Android System .It sends Intent (similar to events) along with Intent information. Important intents to be captured by the system will be: BOOT_COMPLETED, INSTALL_COMPLETED, UPDATE_COMPLETED and other intents sent by applications internally and externally for IPC.

5.4.2.6 Android Loadable Kernel Module

It is a component internal to the Android System. It catches the system calls made by the application during runtime. It logs the frequency and sends the data back to the service.

5.4.2.7 Privilege Escalation Attack detector

It is a tool which parses the androidmanifest.xml and checks for certain fields like uses permissions, intent filters, exported tag, data and action for every activity and service to check if an attack can take place.

5.4.2.8 Capability leak detection tool

It is a tool which parses the source files to check for probable components. CFG's are created, every node in the graph is check to see if it acts as a sink and if the source is tainted.

5.4.3 Software Interface Description

The software's interface(s) to the outside world are described.

5.4.3.1 External machine interfaces

Interfaces to other machines (computers or devices) are described.

1. Category Server: A server which stores the database for the mapping of category-permission. It is queried for testing if the permissions requested are actually in accordance to the nature of the application.
2. Malware Signature Server: A server which stores the database for mapping each of the malware to its signature. It has supervised machine learning algorithms running on it which are then applied on the feature vector to determine if the test app is malware. Database is updated if app is malware.

5.4.3.2 External system interfaces

Interfaces to other systems, products, or networks are described.

1. The system can be a part of a large organizational setup connected via intranet or connected through internet when deployed on cloud for distributed use.

5.4.3.3 Human interface

An overview of any human interfaces to be designed for the software is presented.

1. User Interface to accept the category of the application when selected from a list of categories by the user.
2. User Interface to accept action to be taken on notification for a possible security attack .That is, to submit report to central database, ignore or remove.
3. Graphical and Tabular results for the users.

5.5 User interface design

A description of the user interface design of the software is presented.

5.5.1 Interface design rules

Conventions and standards used for designing/implementing the user interface are stated. The interface are to GNOME standards and accessible to Development Environment.

The interface design rules required for the project are:-

1) The interface should be clean and user-friendly.

2) The interface should allow user to modify any feature of the service.

3) There should be provision of adding or removing a component.

4) The interface should help user understand what all options they have for selecting, adding or removing.

5) Enable frequent users to use shortcuts.

6) Display descriptive messages and text.

7) Offer simple and efficient error handling.

8) Permit easy reversal of actions.

9) Reduce short-term memory load.

10) Strive for consistency.

11) Allow users to customize the interface

12) Provide defaults

5.5.2 Components available

GUI components available for implementation are: Activities ,Toast (for notifications), other android. Graphics library in android.

5.6.1 Requirements traceability matrix

A matrix that traces stated components to software requirements is developed.

Requirements	Package Manager & Package Installer	Permission Extractor	Static Analyzer	Dynamic Analyzer
Modifiability	No	Yes	Yes	Yes
Reliability	Yes	Yes	Dependent on DB strength	Dependent on DB and training set strength
Privilege escalation	Yes	Yes		
Malware Detection			Yes	Yes

CHAPTER 6

TEST SPECIFICATION

6.1.0 Introduction

6.1.1 Goals and objectives

The test process is carried out to ensure full functionality and efficiency of our modules without affecting the performance of the device.

6.1.2 Statement of scope

Test the kernel module on the Android SDK emulator.

Test our service on the emulator.

Test the server.

6.1.3 Major constraints

As the kernel module can only be inserted in rooted phones or emulator, we can only carry out testing on certain devices.

The training set required for classification needs to be appropriately large for testing purposes.

6.2.0 Test Plan

6.2.1 Software (SCIís) to be tested

hijack_kernel (Android loadable kernel module)

ARMORNEW (Server – Play application)

myservice (Android service)

CHAPTER 7

FUTURE ENHANCEMENS

As a future enhancement to the system we plan to extend the system to the remote server. The system can be modified in following ways:

1. Automatic categorization.
2. Increasing dataset
3. Increasing the system calls to be hijacked.
4. Deploying the server system on the cloud using Amazon EC2 server instance.

CHAPTER 8

SUMMARY & CONCLUSION

Summary

In this project we have proposed the concept of automated malware detection for Android phone users to prevent any kind of data leaks, may be caused by a malicious application. The main advantage of this model is that the process will be running in background monitoring each call made by the application & authenticating it for its category. The developer’s tool also parses the entire package file hooking on to any modules which may fall prey to privilege escalation attack or a capability leak. It gives the developers tabular and graphical results which illustrates those modules in the codes he needs to strengthen. This will also secure the sensitive information on the phone, hence motivating users to use the cell phones for monetary transactions and other personal usage.

Conclusion

“ARMOR” detects malware and privilege escalation attacks through constant monitoring of inter process communication and notify it to the user with severity levels.

CHAPTER 9

REFERENCES

Android Application Sandbox System for Suspicious Software Detection. In 2010 5th International Conference on Malicious and Unwanted Software

[2] Yajin Zhou, Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In 2012 IEEE Symposium on Security and Privacy

[3] MoutazAlazab, VeelashaMoonsamy, Patrik LantzLynn. Analysis of Malicious and Benign Android Applications. In 2012 32nd International Conference on Distributed Computing Systems Workshops

[4] M. Becher, F. Freiling, and B. Leider. On the effort to create smartphone worms in windows mobile. In Information Assurance and Security Workshop, 2007. IAW '07. IEEESMC, pages 199–206, 20-22 June 2007.

[5] J. Bergeron, M. Debbabi, J. Desharnais, M. M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. In Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS'01), 2001.

[6] M. A. Bishop. The Art and Science of Computer Security. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[7] Iker Burguera, Urko Zurutuza, Simin Nadjm-Tehrani. Crowdroid: Behavior-Based Malware Detection System for Android.

[8] S. Forrest, S. Hofmeyr, and A. Somayaji. The evolution of system-call monitoring. In ACSAC '08: Proceedings of the 2008 Annual Computer Security Applications Conference, pages 418–430. IEEE Computer Society, 2008.

[9] R. Love. *Linux System Programming*. O'Reilly, 2007.

[10] C. Willems, T. Holz, and F. Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, 5(2):32–39, 2007.

[11] http//developer.android.com (Last cited on:25th September 2013)

[12] http://forum.xda-developers.com (Last cited on: 22th September 2013)

[13] https://play.google.com/store/apps (Last cited on: 17th August 2013)

[14] Wei-Meng Lee, *Beginning Android 4 Application Development*. Wrox, 2012

CHAPTER 10

ANNEXURES

ANNEXURE – I

Project Title: Android Security Based Tools

Sponsorship : Symantec Software India Pvt. Ltd

External Guide: Name: Ketan A. Karnick
Title: Sr. Software Engineer
Email: Ketan_karnick@symantec.com

Internal Guide: Prof. M.S. Wakode

1.1 Title of the project: Android Security Based Tools

1.2 Project Area: Android, Security, Access control, Pattern matching
1.3 Technical Keywords (Refer ACM Keywords):
D.4.6 Security and Protection
Authentication
Invasive Software
Security Kernels
Access Control
Verification
H.2.8 Database Applications
Data Mining
H.3.3 Information Search and Retrieval
Clustering
Relevance Feedback
E.1 Data Structures
Graphs
Trees

1.4 Names of at least two conferences where papers can be pu1. 2011 International Conference on Computer Science and Network2. 2012 1st International Conference on Recent Advances in I(RAIT).
3. 2011 International Conference on Advanced Computer Science a(ICACSIS).

1.5 Relevant mathematics associated with the Project:
Refer Annexure A

1.6 Plan of project execution:
Refer Annexure C

ANNEXURE – A

Mathematical Model

The structure of the system using set theory is :

Let ‘S’ be a System.

S={I,O,Suc,F,C}

Where

I be the set of Input to the system S

O be the set of Output of the system S

Suc be the set of Successful features of the System S

F be the set of Failures of the System S

C be the set of Components of an application

P be the set of Data Structures used

I: Input to the System S

I={i1,i2,i3,i4}

Where

i1 = Android Application (Installed or uninstalled)

i2 = Category Of the Application

i3 = Permissions Required during installations

i4 = All intents, Content Providers

O : Output of the system S

O={o1,o2,o3,o4}

o1 = Installed / Downloaded Apk does not request for permissions that are beyond its requirements.(based on the category of the app)

o2 = Detect over privilege in compiled Android applications.

o3 = Detect other malicious behaviour of the application.Flag it / remove the source of conflict.

o4 = Alert the user to the possible threats posed by the application.

Suc=Set of Successful features of the System S

Suc={s1,s2,s3}

s1 = Handles applications of varied categories. Correctly maps the app to the category set.

s2 = Alerts User to the malicious behaviour

s3 = Extracts all the permissions irrespective of the flag.

F=Set of Failure of the System S

F={f1,f2,f3}

f1 = System may fail to extract requested permissions if the android manifest file of the app uses wrong permission tags.

f2 = If the application permission levels are set to ‘normal’ , user is not informed of the permissions required and hence may disable the system.

f3 = Some advanced and new malicious behavior may be ignored by the system.

C=Set of All components of the application

C={c1,c2,c3,c4}

c1=Intent set

c2=Uses-Permissions

c3=Permission Function

c4=Permissions

An intent is a 4-tuple (a, b ,c ,d)

where

a = is an action string describing the action to be performed,

b = is a string representing the data,

c = the string representing the category and

d = name -> val is a function that maps names of extra information to their values.

The set of intents is denoted as I.

I = (a,b,c,d)

```
<intent-filter>
<action android:name=”net.learn2develop.ACTIVITY2” />
<category android:name=”android.intent.category.DEFAULT” />
</intent-filter>
```

Problem statement feasibility assessment using satisfiability analysis and NP Hard, NP Complete or P type using modern algebra:

An algorithm is Non-Deterministic when:

1. It uses external state such as user input, random value or stored disk data.
2. It operates in a way that is time sensitive.
3. If hardware error causes its state to change in an unexpected way.

The problem definition deals with user driven categorization of applications & malware database processing which requires external disk data. Hence it satisfies the first criteria of non-determinism.

- Non-Deterministic machines that cannot solve problems in polynomial time are NP.
  The solution to our problem depends upon network connectivity and speed of the network connection. However, the connectivity of the network cannot be guaranteed in polynomial time.
  But it can be verified in polynomial time that the solution is correct or not (using the satisfiability equation). Hence, the problem is NP.
- Satisfiability(SAT): Problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to TRUE.
- Also, if it can be reduced to the Boolean Satisfiability problem, then it is NP-complete (acc. to Cook's Theorem).

In this problem,

a = probability of assigning right category. Thus, P (a) =1

b= probability of metadata of malwares signatures for maximum coverage

c= probability of feature extraction from the application package is correctly done.

The Boolean expression (a AND b AND c) represents a format of the problem definition similar to the Boolean Satisfiability problem. If each element is satisfied, then the problem can be solved. The SAT problem being an NP-complete problem, the given problem definition is also NP-complete.

NP-completeness of the problem

As the no. of applications running on the device increases, scanning of all system calls goes on increasing & so the load on the system increases and the probability of successful results tend to zero.

ANNEXURE - B

Laboratory Assignments on Project Quality and Reliability Testing Of Project Design

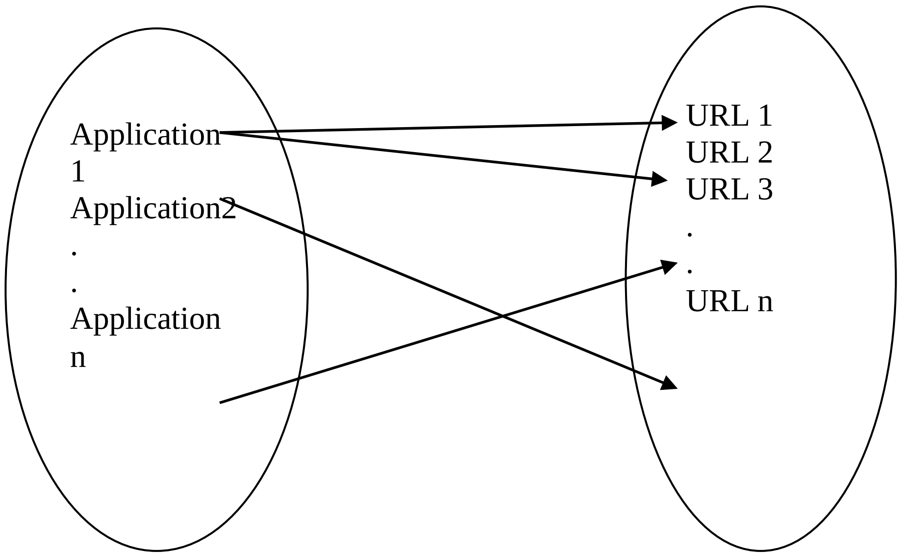
Assignment 2
Use of Assignment 1 to identify objects, morphism, overloading, functions and functional relations and any other dependencies

Sr. No.	Description	UML Design Observation
1.	Problem Description:	
	S= { USER, APPLICATION,CAT_S , MALWARE_S ,PERMISSION_EXTRACTOR,PRIVILEDGE_ESCALATION OUTPUT | Φ} Φ=Φ1+Φ2+Φ3+Φ4. Φ1=rules such that the DBS server contains the malware base. Φ2=rules to ensure reliable inter-process communication Φ3=rules to ensure reliable feature extraction Φ4=rules to ensure correct categorization	
2.	F_1=> (application_category) → {(application_permission),F_add} F_add=>(app_id,app_cat)→ DBS{(app_id,app_cat,{app_permissions})} $F_2 \Rightarrow \sum_{K=1}^{N1}$ {application_k} (F1 (application_i)) → $\sum_{K=1}^{N1}$ {IPC_check} $F_3 \Rightarrow \sum_{K=1}^{N1}$ (application_k)→$\sum_{K=1}^{N1}$ { Sampling} $F_4 \Rightarrow \sum_{K=1}^{N1} \sum_{I=1}^{N2} F_3$ (application $_{K,I}$)→$\sum_{J=1}^{N1}${Dynamic_Analysis Report} F_5 => F_4 (Android_System)→ {Reports,{Status=(Safe/Unsafe) } = NotificationF_6 =>Notification → OUTPUT	

3.	It includes morphism as the same DBS is replicated for various applications. Cat_DBS$\rightarrow$\{app_1, app_2, …..,app_n\} Content_provider $\rightarrow$ \{app1,app2,…..app_n\} Malware_DBS---$\rightarrow$\{app1,app2,…appn\}	Morphism and Overloading

Morphs:

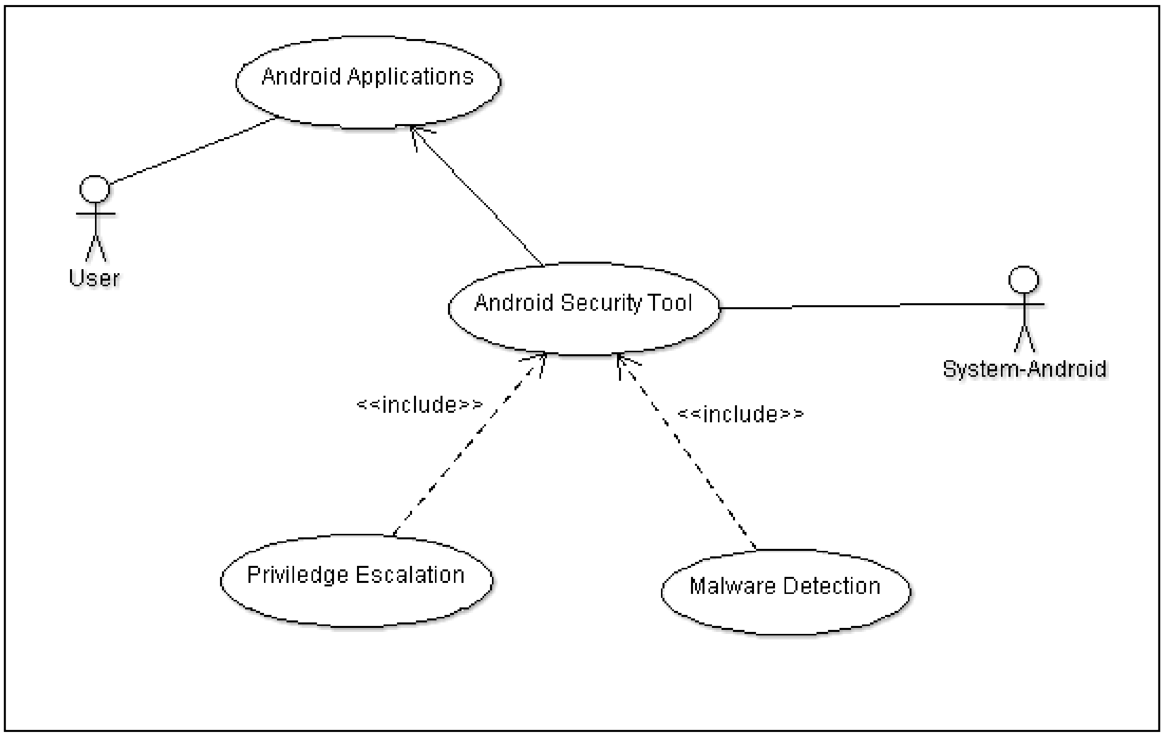
Application and Content-Providers (Accessed By URL)



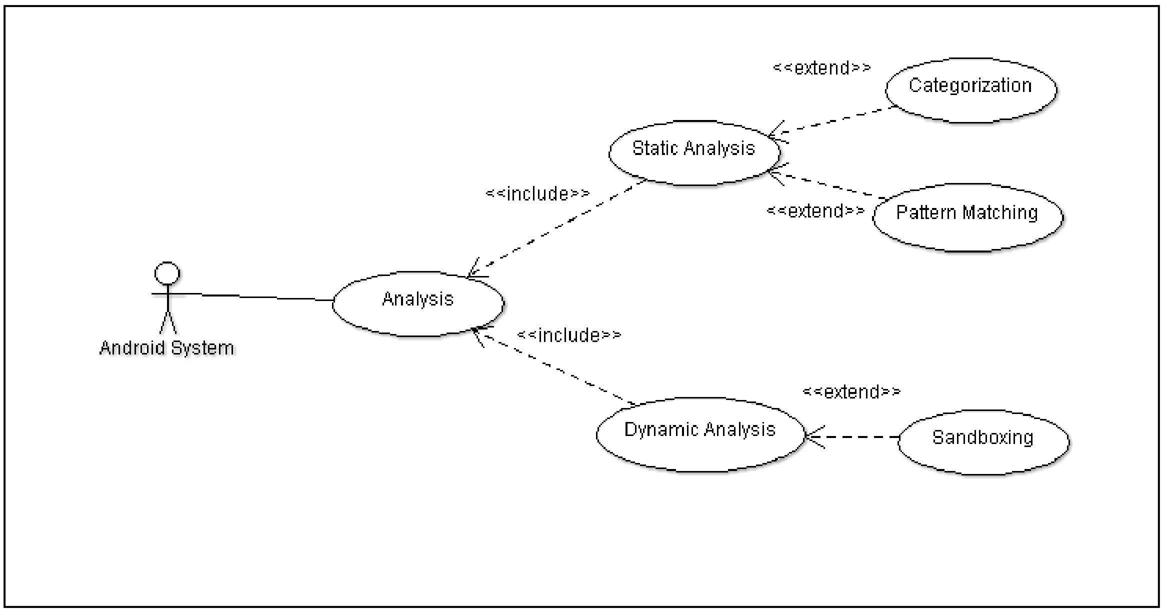
UML Diagrams

1. USE CASE DIAGRAMS

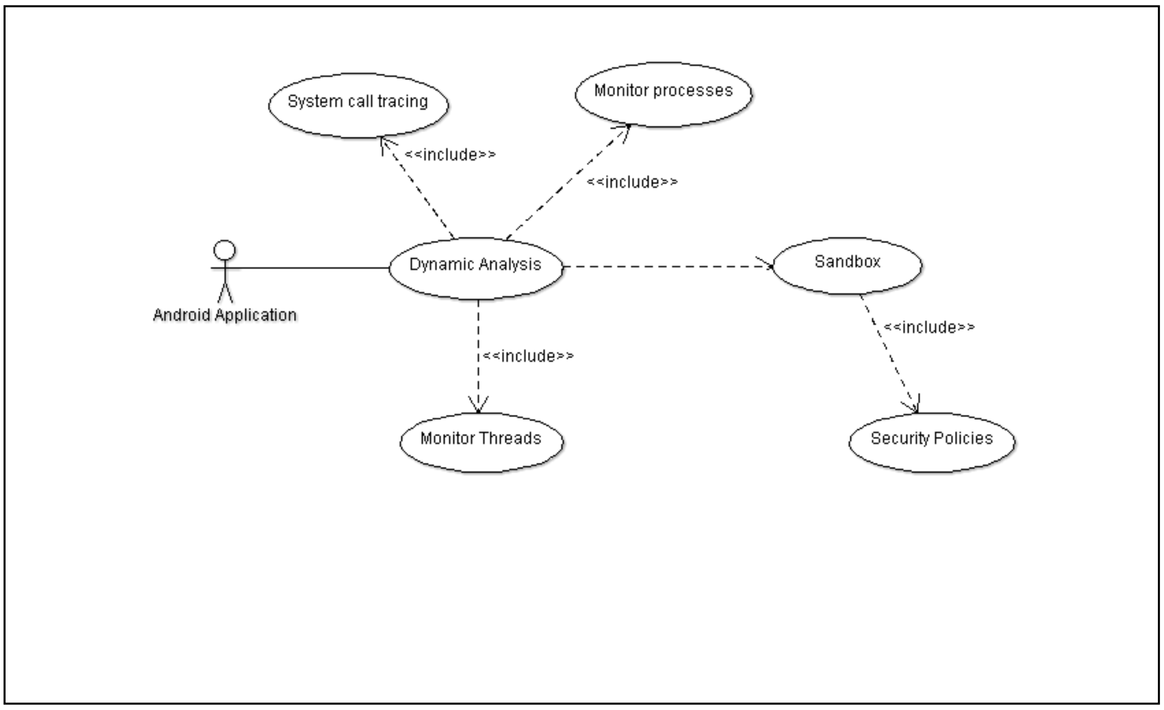
USE – CASE 1:



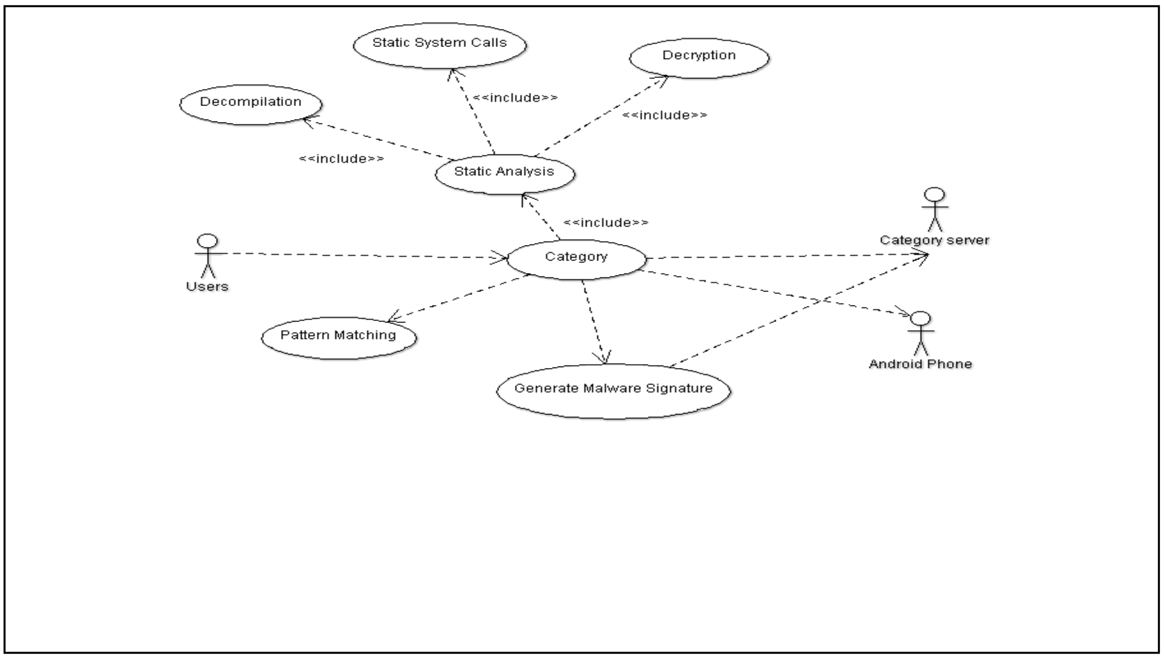
USE – CASE 2:



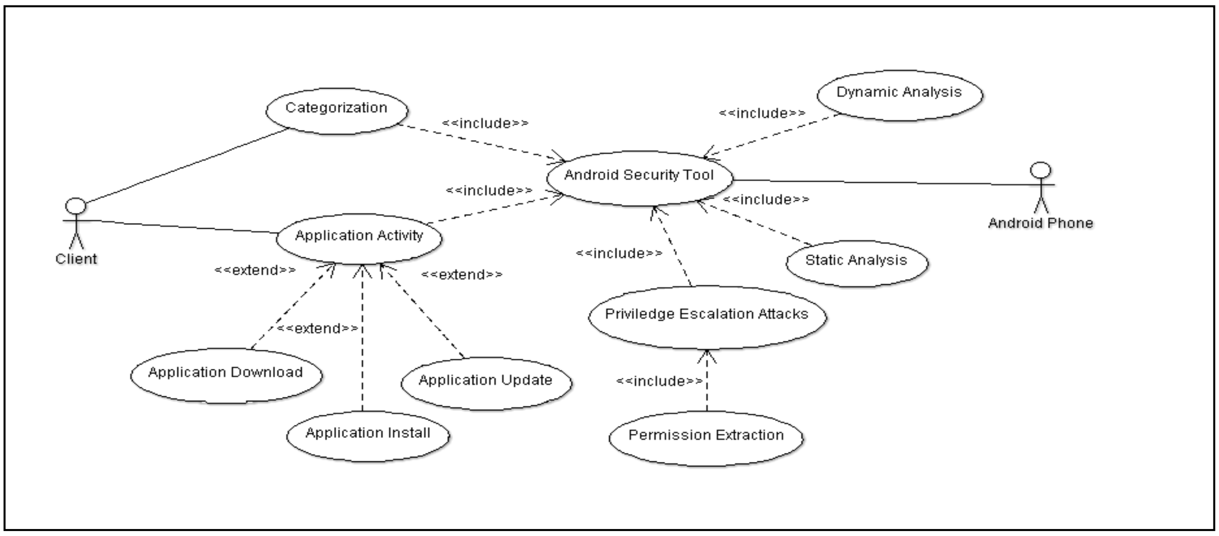
USE – CASE 3:



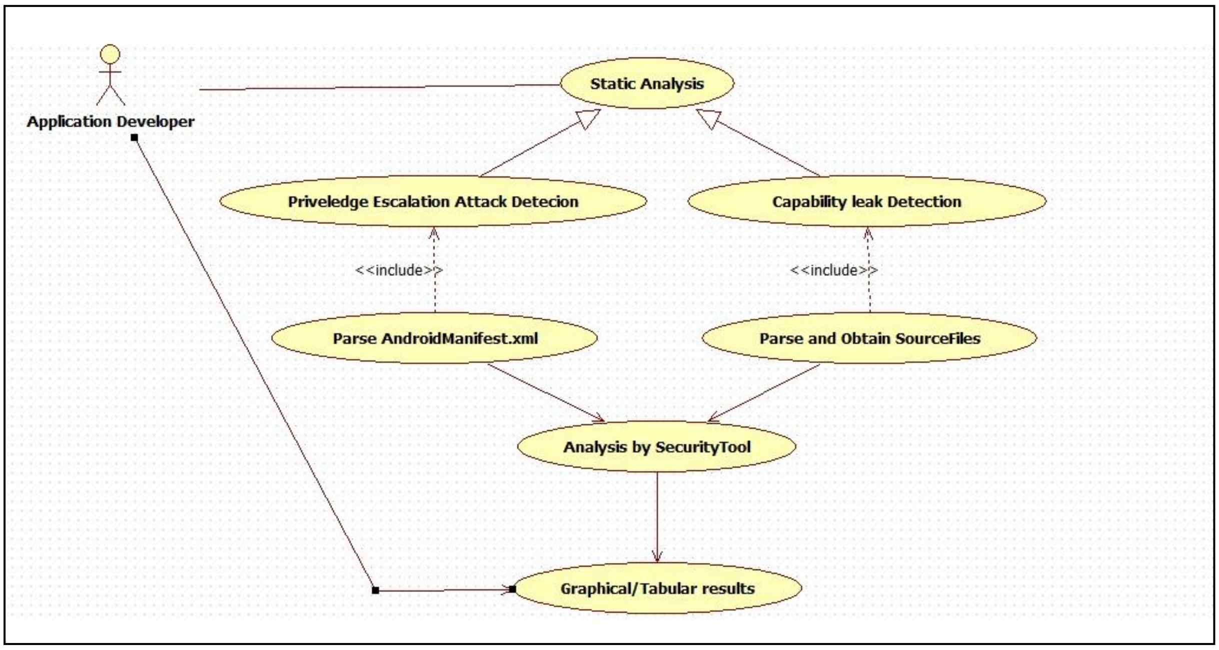
USE CASE 4:



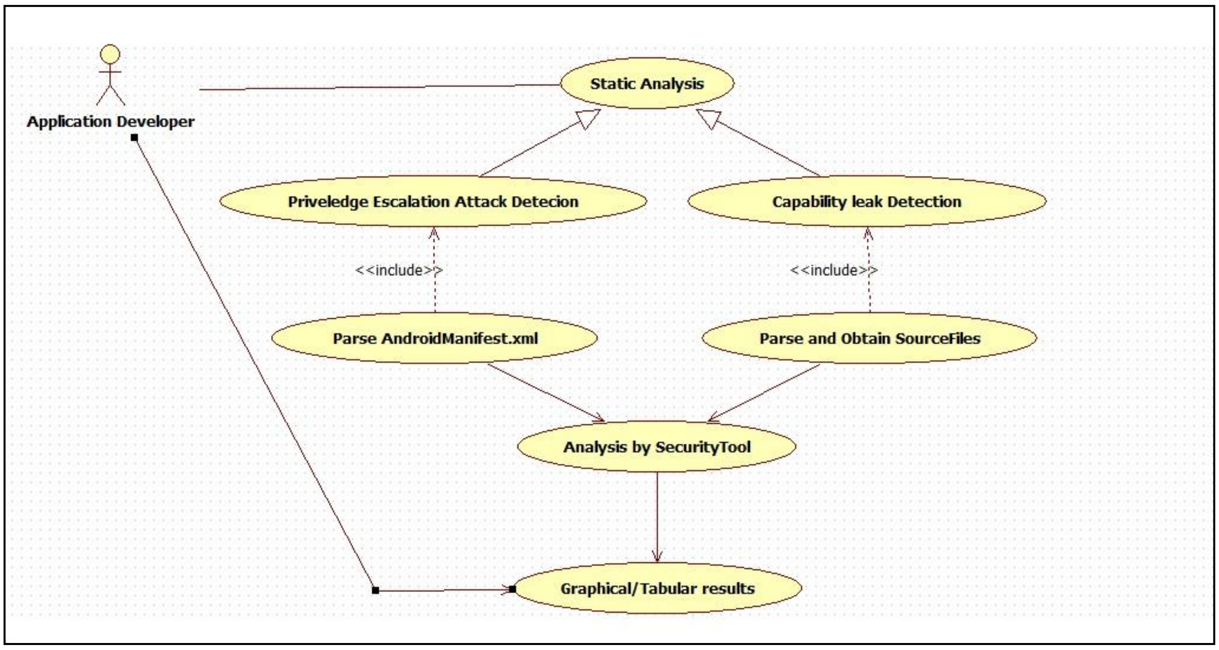
USE CASE 5:



Use Case 6:

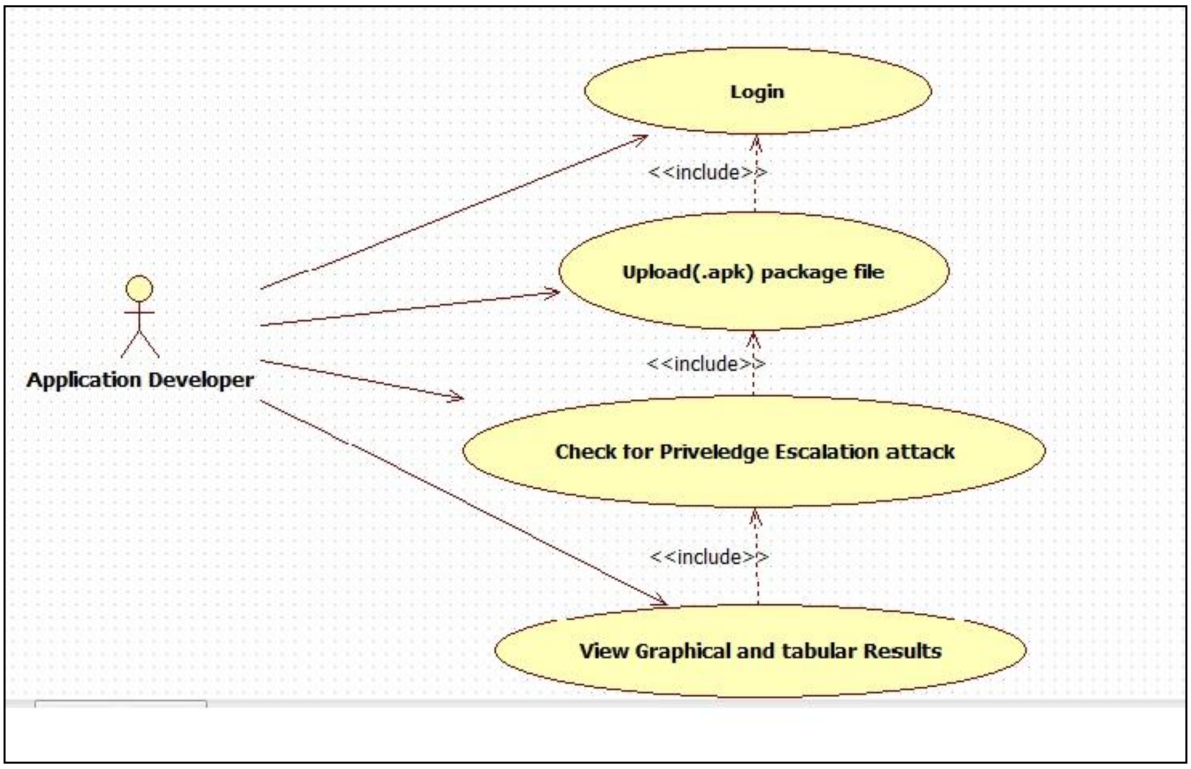

Application Developer
Static Analysis
Priveledge Escalation Attack Detecion
Capability leak Detection
<<include>>
<<include>>
Parse AndroidManifest.xml
Parse and Obtain SourceFiles
Analysis by SecurityTool
Graphical/Tabular results

Use Case 6:



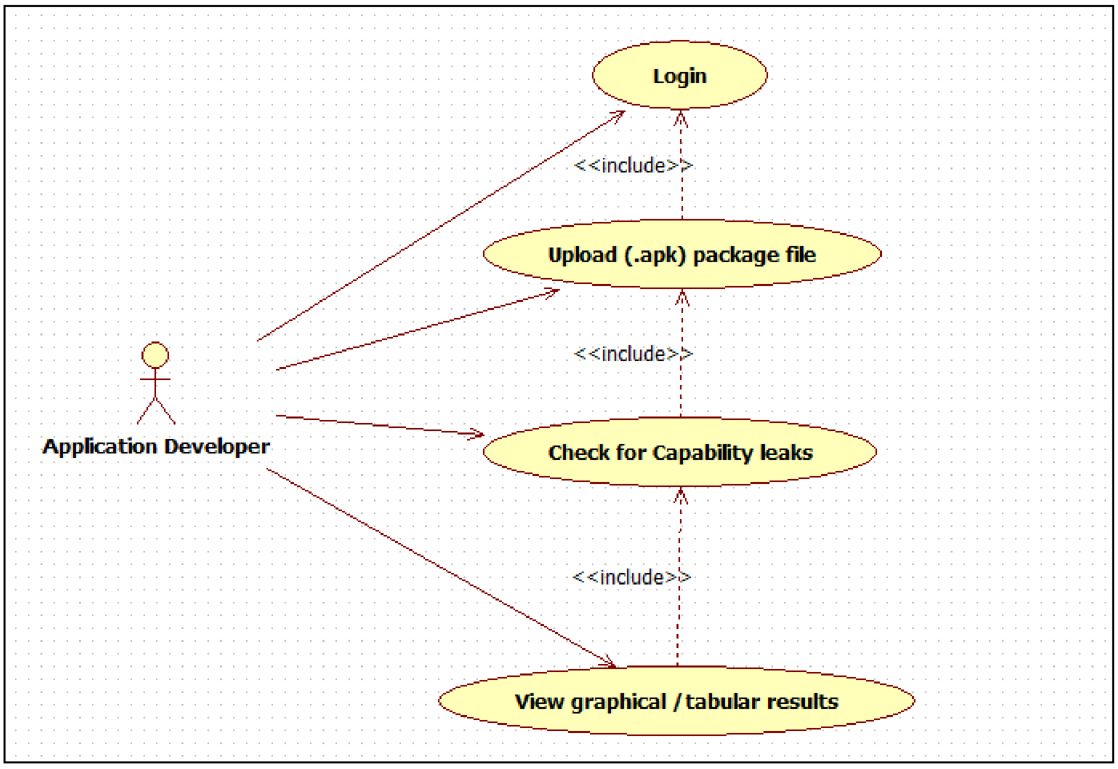
Application Developer
Static Analysis
Priveledge Escalation Attack Detecion
Capability leak Detection
<<include>>
<<include>>
Parse AndroidManifest.xml
Parse and Obtain SourceFiles
Analysis by SecurityTool
Graphical/Tabular results

Use Case 7:



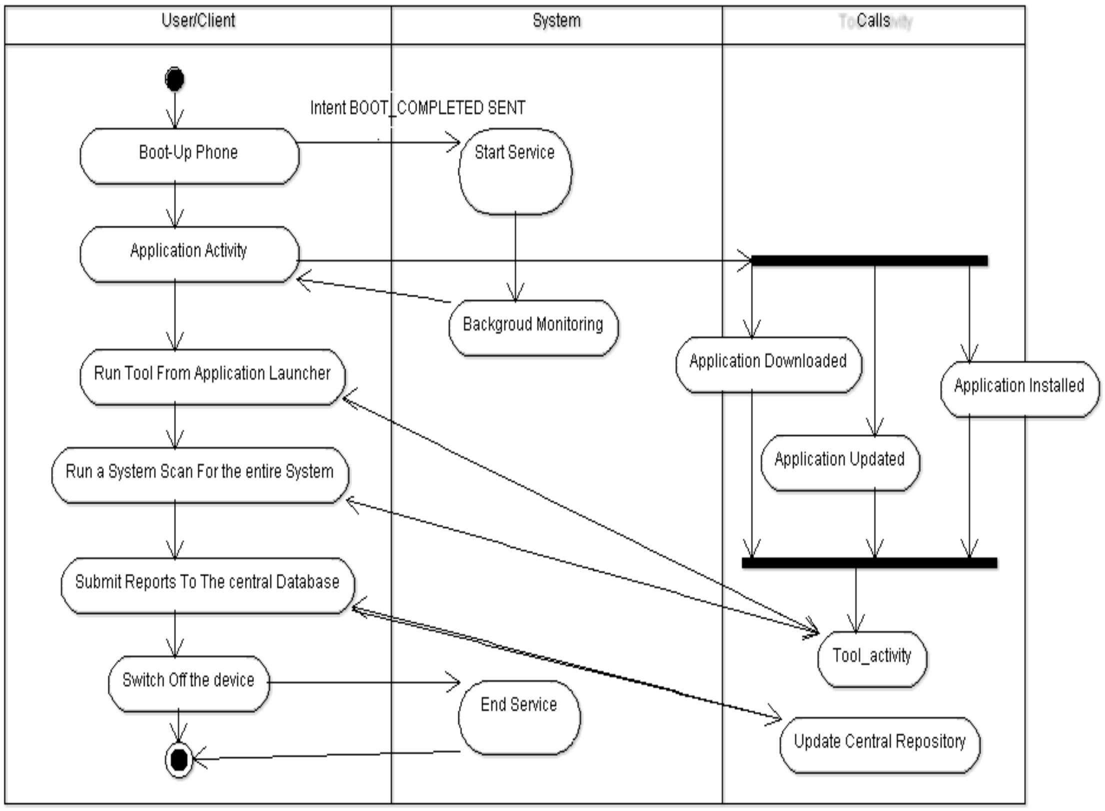
Login
<<include>>
Upload(.apk) package file
Application Developer
<<include>>
Check for Priveledge Escalation attack
<<include>>
View Graphical and tabular Results

Use Case 8:

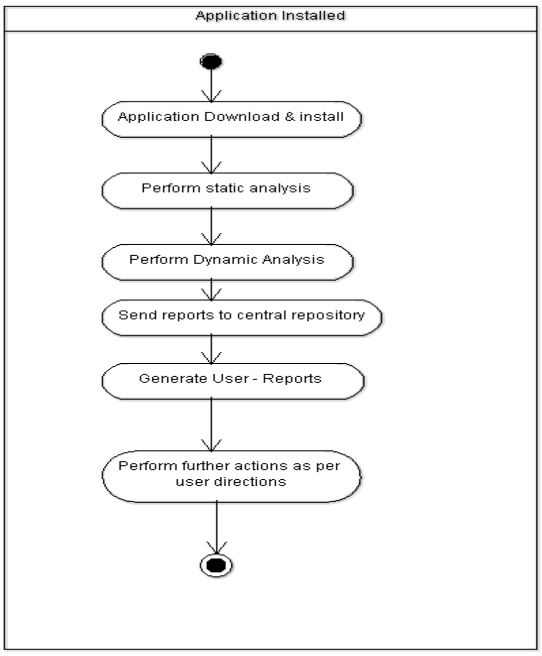

Login
<<include>>
Upload (.apk) package file
<<include>>
Application Developer
Check for Capability leaks
<<include>>
View graphical / tabular results

2. ACTIVITY / FLOW DIAGRAMS

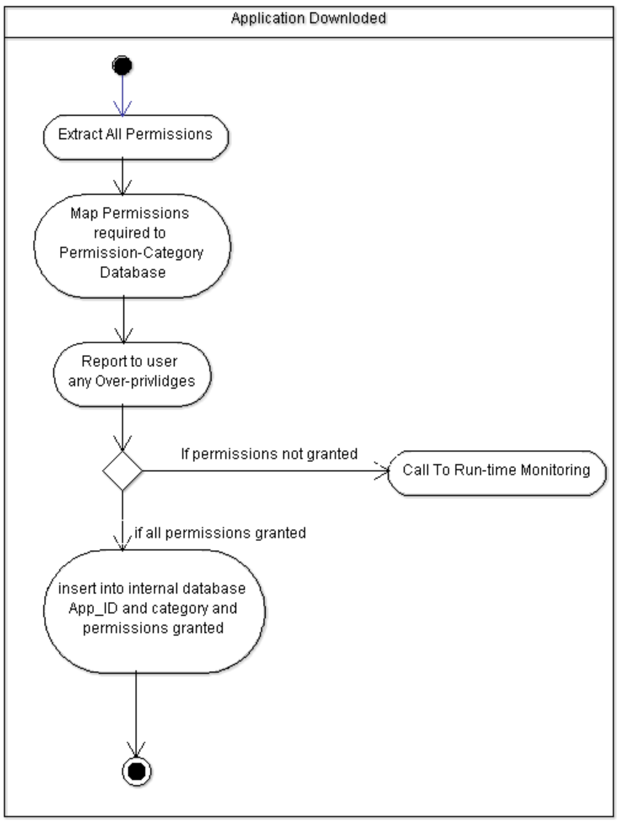
ACTIVITY DIAGRAM 1:



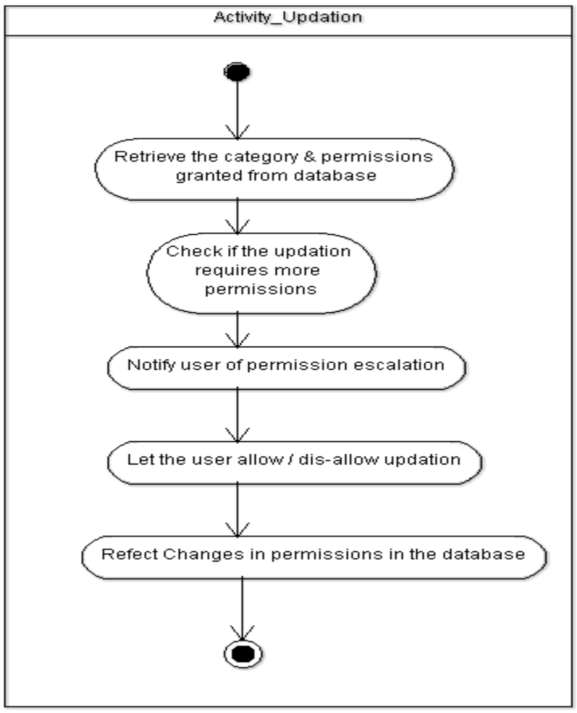
ACTIVITY DIAGRAM 2:


Application Installed
Application Download & install
Perform static analysis
Perform Dynamic Analysis
Send reports to central repository
Generate User - Reports
Perform further actions as per user directions

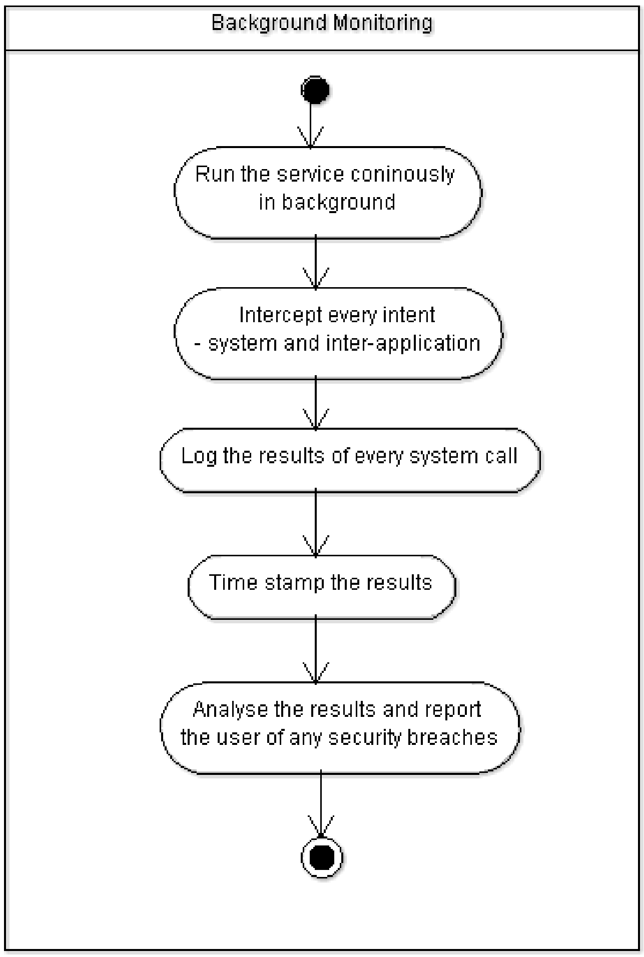
ACTIVITY DIAGRAM 3:


Application Downloded
Extract All Permissions
Map Permissions required to Permission-Category Database
Report to user any Over-privlidges
If permissions not granted
Call To Run-time Monitoring
if all permissions granted
insert into internal database App_ID and category and permissions granted

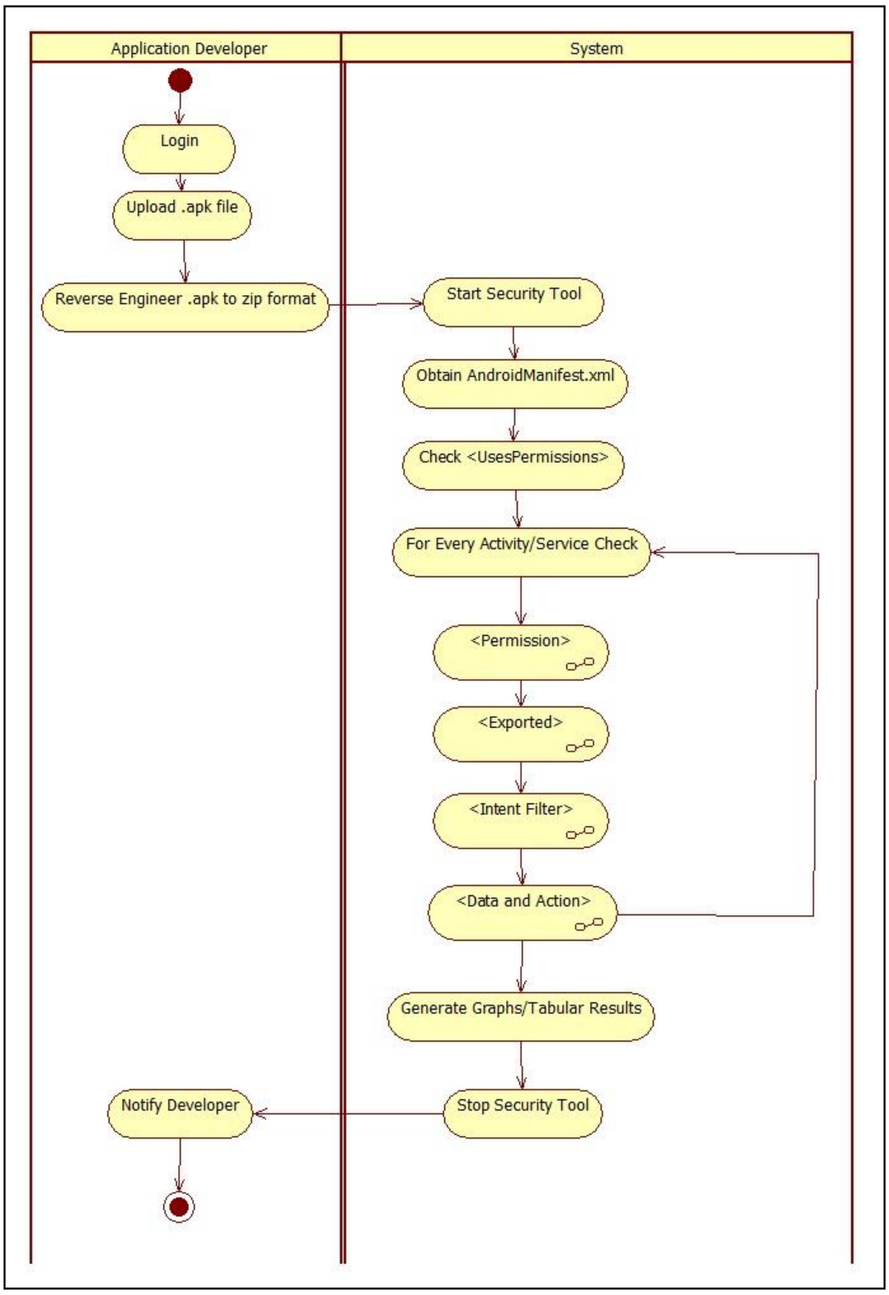
ACTIVITY DIAGRAM 4:


Activity_Updation
Retrieve the category & permissions granted from database
Check if the updation requires more permissions
Notify user of permission escalation
Let the user allow / dis-allow updation
Refect Changes in permissions in the database

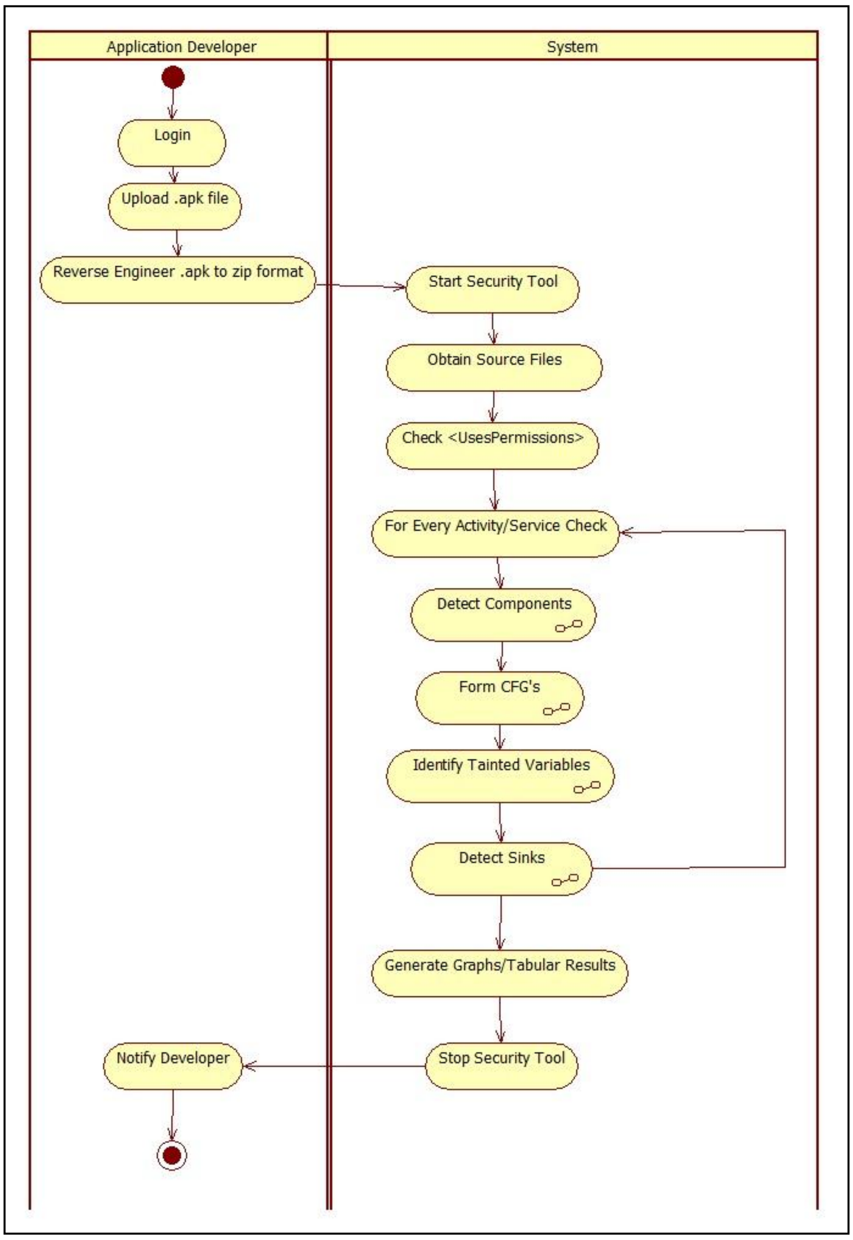
ACTIVITY DIAGRAM 5:


Background Monitoring
Run the service coninously in background
Intercept every intent - system and inter-application
Log the results of every system call
Time stamp the results
Analyse the results and report the user of any security breaches

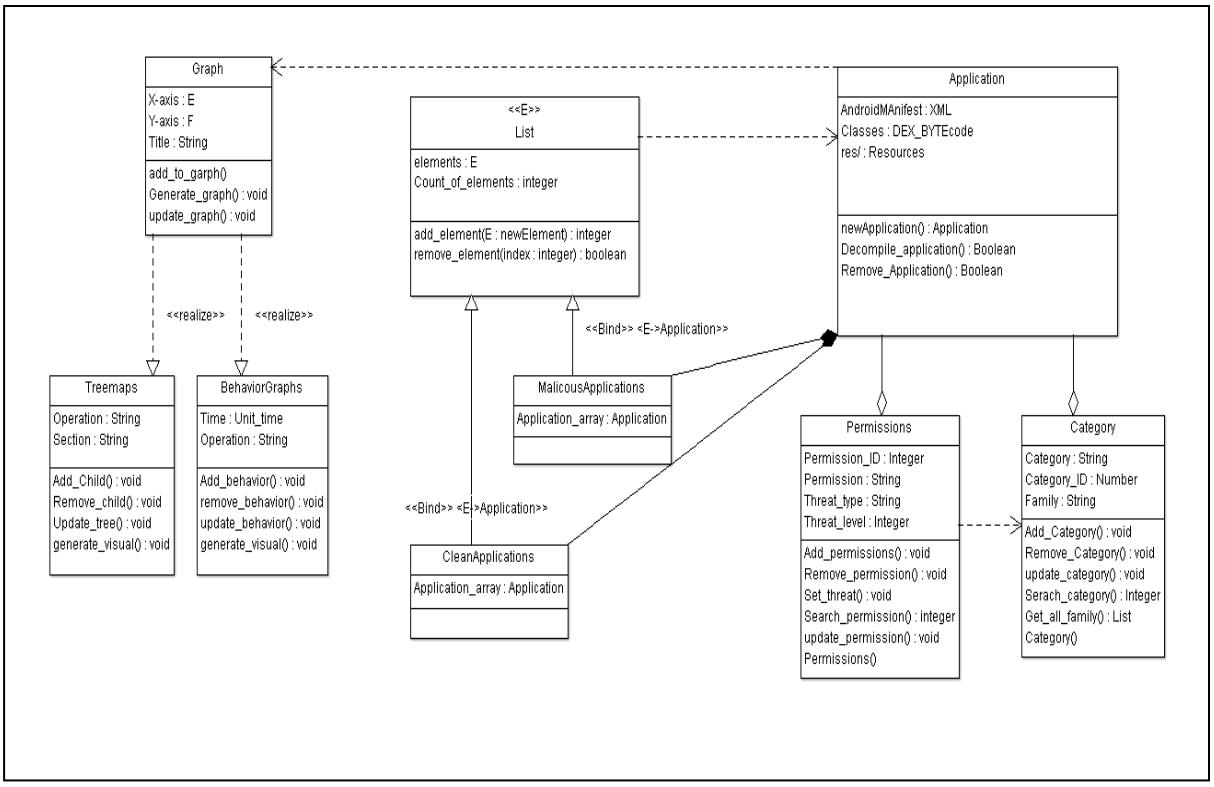
ACTIVITY DIAGRAM 6:



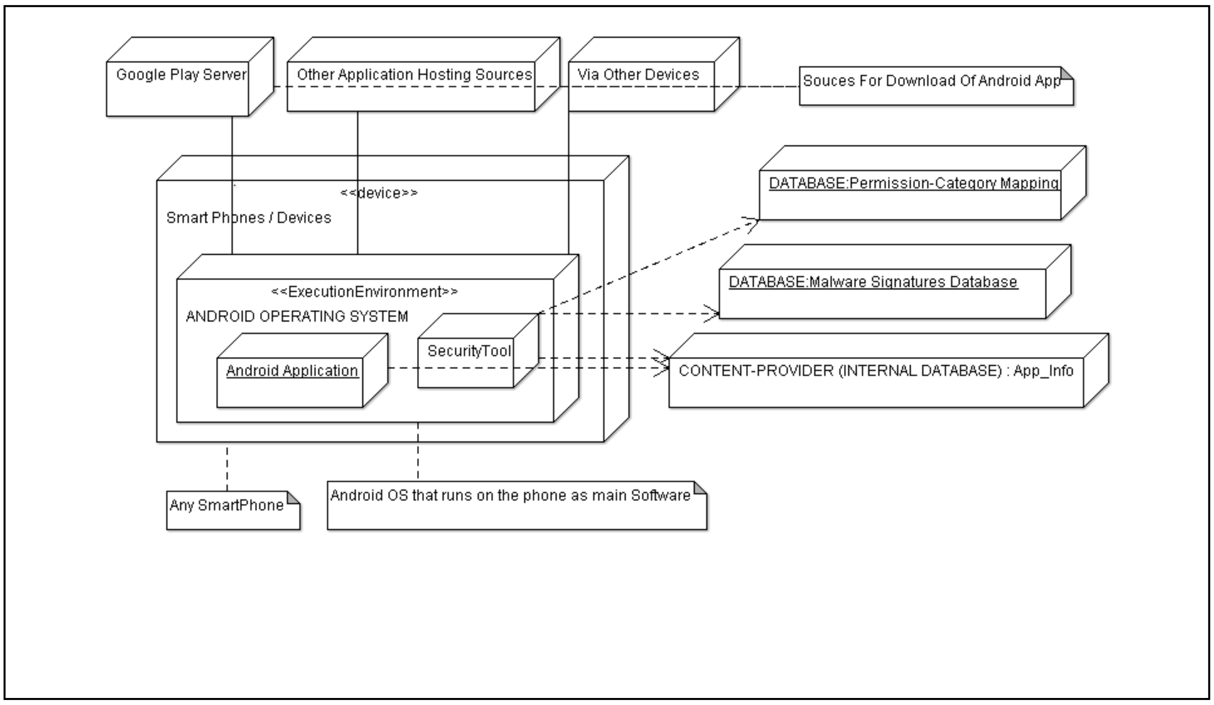
ACTIVITY DIAGRAM 7:



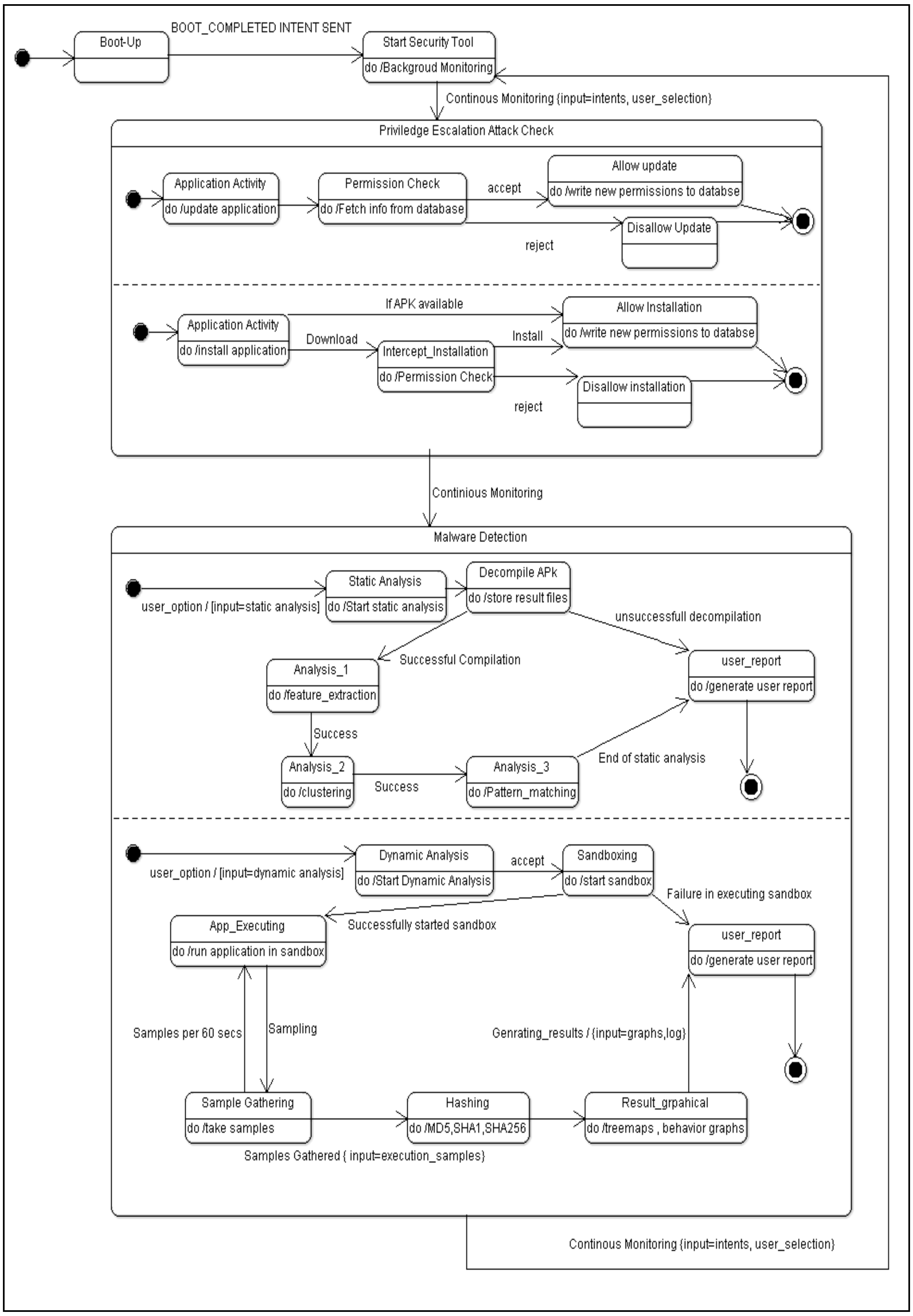
CLASS DIAGRAM:



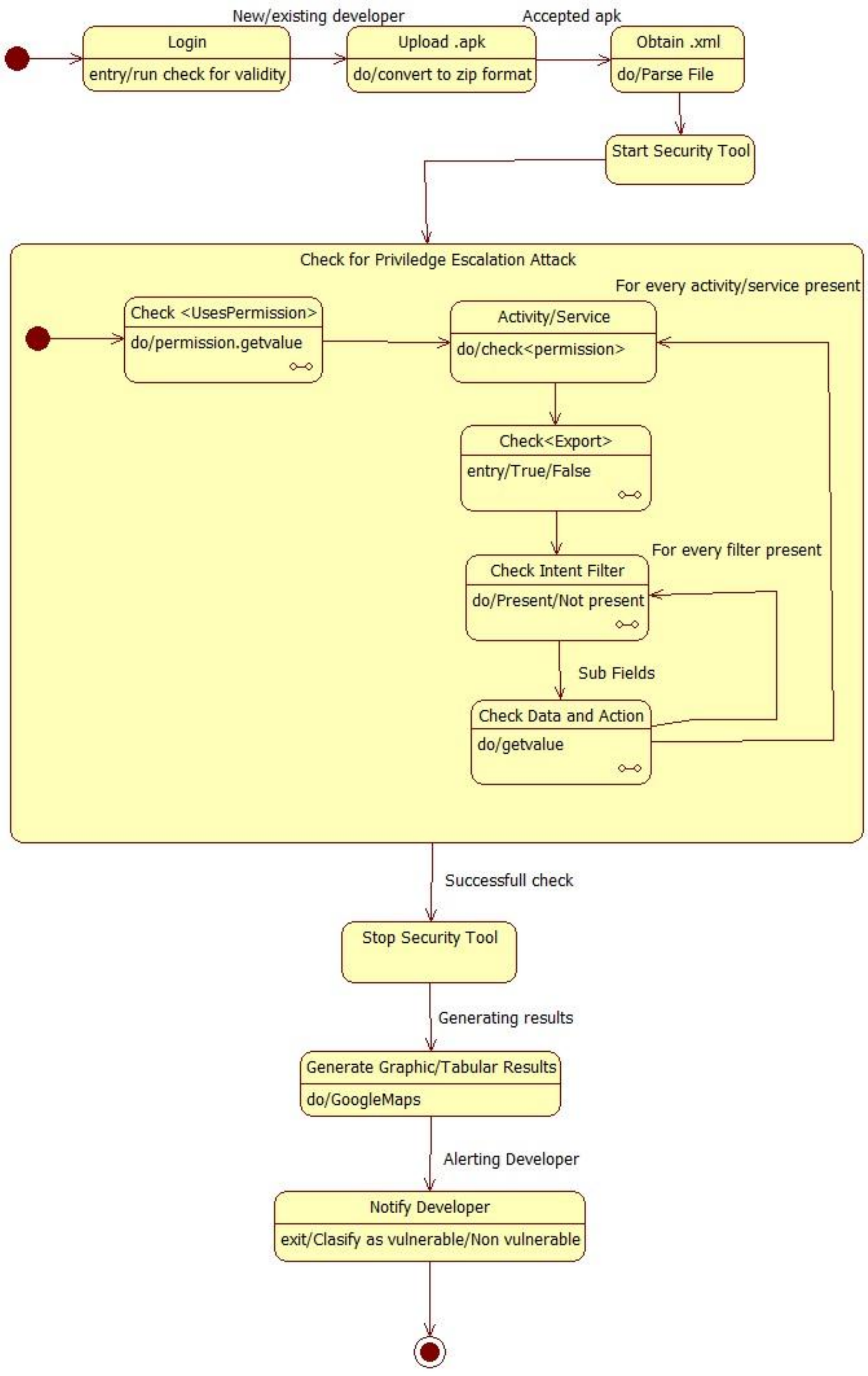
DEPLOYMENT DIAGRAM:


Google Play Server
Other Application Hosting Sources
Via Other Devices
Souces For Download Of Android App
<<device>>
Smart Phones / Devices
DATABASE:Permission-Category Mapping
DATABASE:Malware Signatures Database
<<ExecutionEnvironment>>
ANDROID OPERATING SYSTEM
SecurityTool
Android Application
CONTENT-PROVIDER (INTERNAL DATABASE) : App_Info
Any SmartPhone
Android OS that runs on the phone as main Software

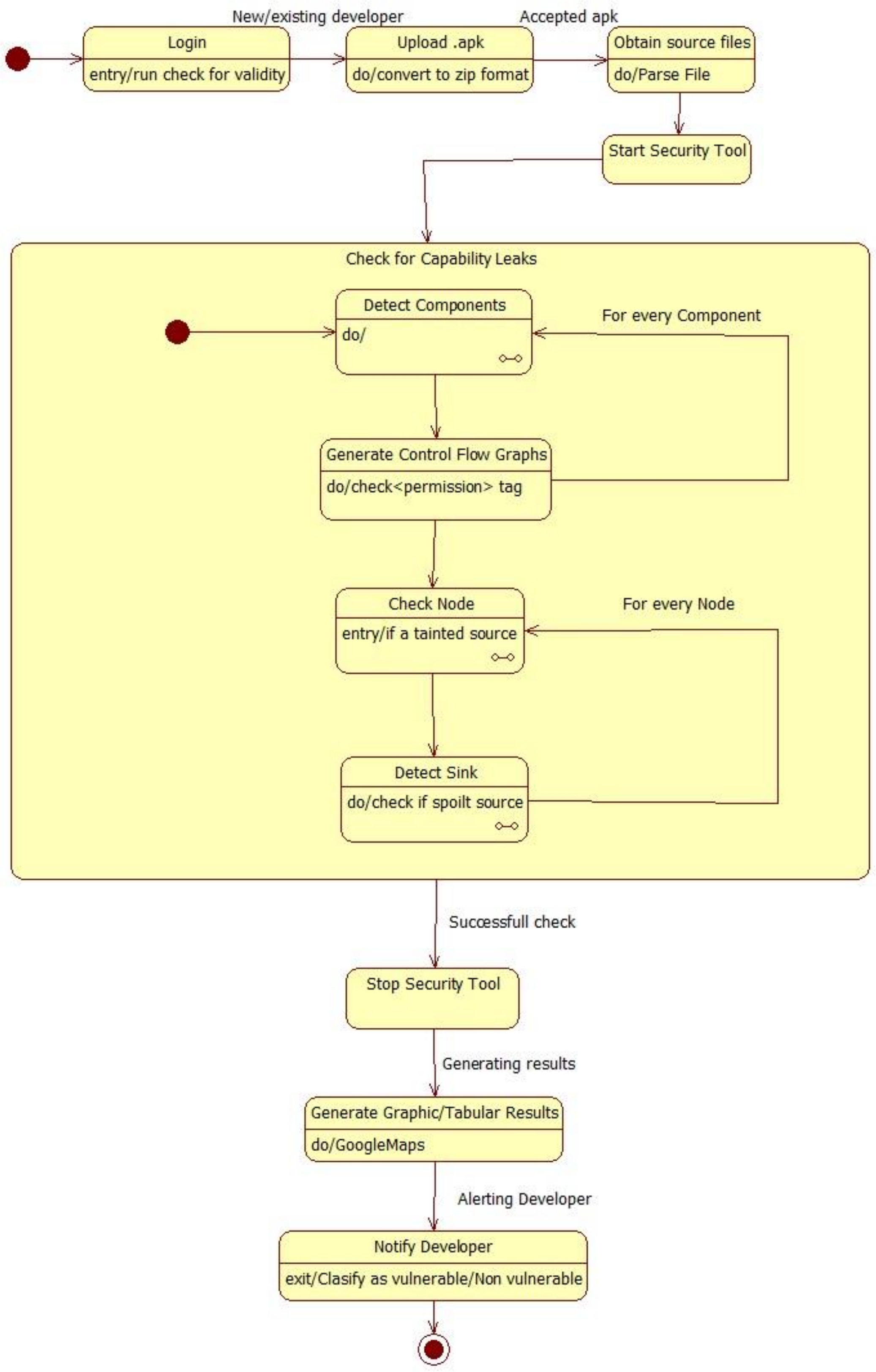
STATE DIAGRAM:



STATE DIAGRAM 3:



STATE DIAGRAM 4:



LABORATORY ASSIGNMENT 4

Unit testing

Each component is tested separately. A bottom up approach is used for testing.

Components for unit testing

1. Permission Extractor
2. Permission Checker
3. Dynamic & Static Analyser

Criteria for selection:

Components for unit testing are selected so that this individual unit of code can be tested separately for their working without depending on other components of the system.

Integration testing

Integration testing is a systematic technique for constructing the software architecture while at the same time conducting tests to uncover tests associated with interfacing. Integration testing is performed after unit testing. It focuses on the problems related to integration of units that function correctly when used individually. An integration testing may follow either a top-down approach or a bottom-up one. Here we have used bottom-up testing i.e. beginning with construction and testing of atomic modules. After integration testing is complete and the errors detected are fixed, regression testing is required to ensure that the changes made to the software have not introduced new errors.

Validation testing

The test environment is designed to be identical, or as close as possible, to the anticipated user's environment, including extremes of such that application continuously used by many organizations simultaneously which may cause multiple database accesses at the same time. To test the module for validation includes checking the performance of the system in multiple hits at the same time. These test cases accompanied by test case input data or a formal description of the operational activities (or both) to be performed—intended to thoroughly exercise the specific case—and a formal description of the expected results. Here the user first interacts with the system through GUI of the system.

High-order testing

High order testing checks that, the software meets customer requirements and that the software along with other system elements, meets the functional, behavioral and performance requirements. It uses black-box techniques. The users themselves perform high-order testing. High-order testing includes validation testing, system testing (focuses on aspects such as reliability, security, stress, usability, and performance), and acceptance testing (includes alpha and beta testing). The testing strategy specifies the type of high-order testing that the project requires. This depends on the aspects that are important in a particular system from the user perspective.

Integration testing

Bottom up integration testing begins construction and testing with atomic modules i.e. components at the lowest levels in the program structure Regression testing is re-execution of some subset of tests that have been already been conducted to ensure that changes have not propagated unintended side effects. Individual components after unit testing are integrated and tested. Each time a new component is added to the system an integration test is performed. Integration is done in hierarchy of how components are built.

ANNEXURE – C

Project Planner

Project Plan:

Project management tool used: Gantt Project 2.0.10

Task	Date of Commencement
Project Topic Search	01/07/2013
Project Topic Finalization	28/07/2013
Research in Area of Application	30/07/2013
Finalization of Problem Definition	20/08/2013
Narrow down the Problem Definition	22/09/2013
Detailed Design and Testing of Design	21/10/2013
Setup of Work-Environment and Start of Development	25/11/2013
Development	05/12/2013
Preliminary Proof of Concept	20/02/2014
Necessary Modifications based on Review of Preliminary POC	03/03/2014
Testing	04/04/2014
Finish Project	30/04/2014

Table 10.1 Project Plan

SDLC paradigm used: Spiral

Project task set

Different activities and phases are mentioned.

T1: Deciding the boundaries and scope of the Project.

T2: Installing Android SDK+eclipse , Bluestacks and other dependencies

T3: Deciding Inputs and Outputs

T4: learning the tools which are required for project

T5: Design Use cases

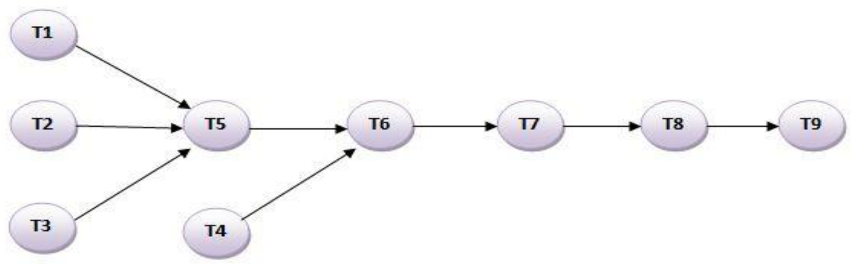
T6: Preparation of 1st prototype

T7: Revision of prototype

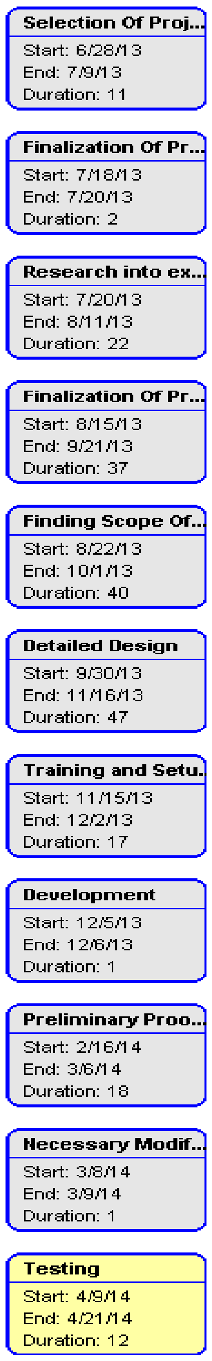
T8: Testing

T9: Documentation

Task network:



PERT CHART:



Selection Of Proj...
Start: 6/28/13
End: 7/9/13
Duration: 11

Finalization Of Pr...
Start: 7/18/13
End: 7/20/13
Duration: 2

Research into ex...
Start: 7/20/13
End: 8/11/13
Duration: 22

Finalization Of Pr...
Start: 8/15/13
End: 9/21/13
Duration: 37

Finding Scope Of...
Start: 8/22/13
End: 10/1/13
Duration: 40

Detailed Design
Start: 9/30/13
End: 11/16/13
Duration: 47

Training and Setu..
Start: 11/15/13
End: 12/2/13
Duration: 17

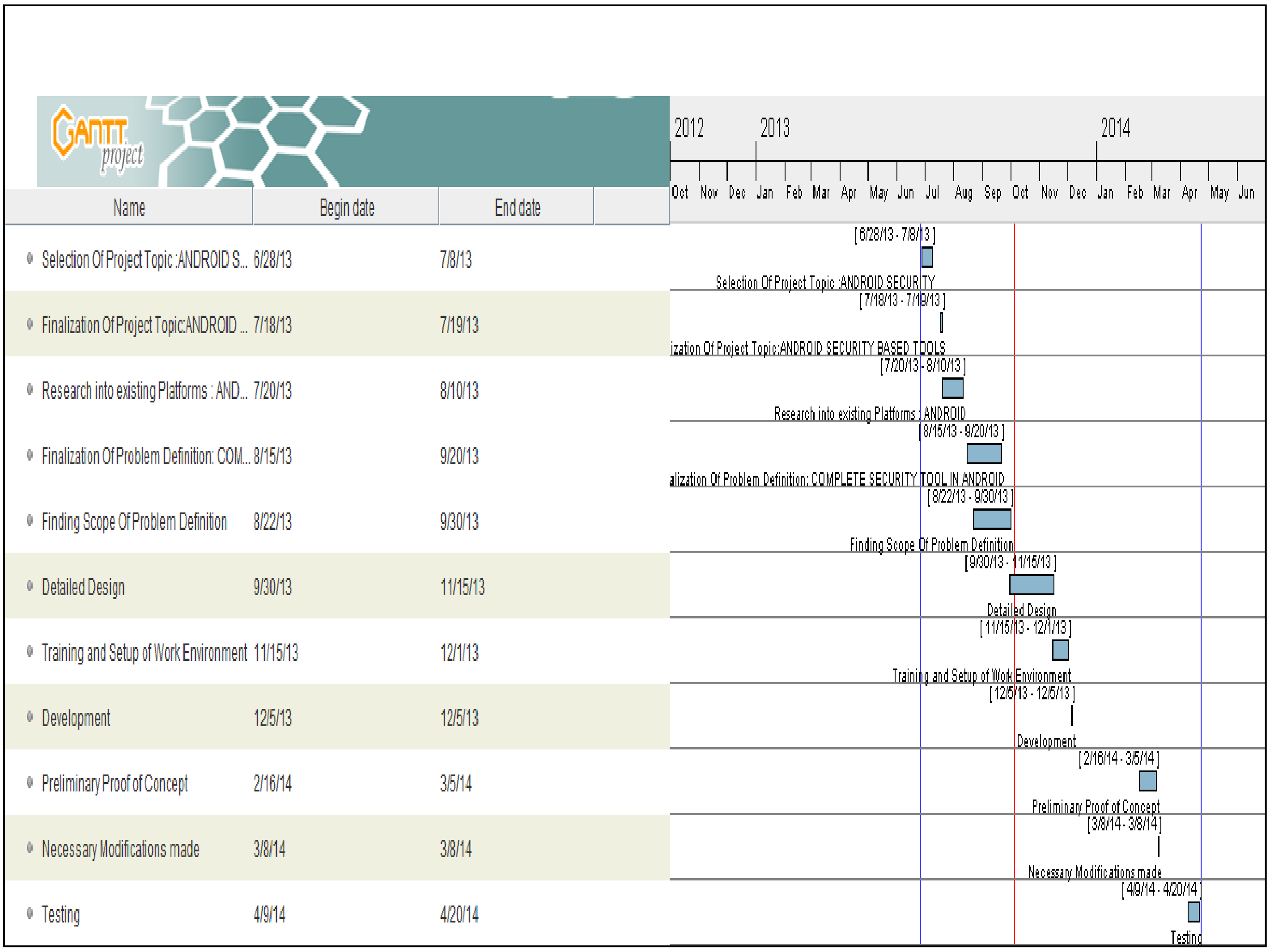
Development
Start: 12/5/13
End: 12/6/13
Duration: 1

Preliminary Proo...
Start: 2/16/14
End: 3/6/14
Duration: 18

Necessary Modif...
Start: 3/8/14
End: 3/9/14
Duration: 1

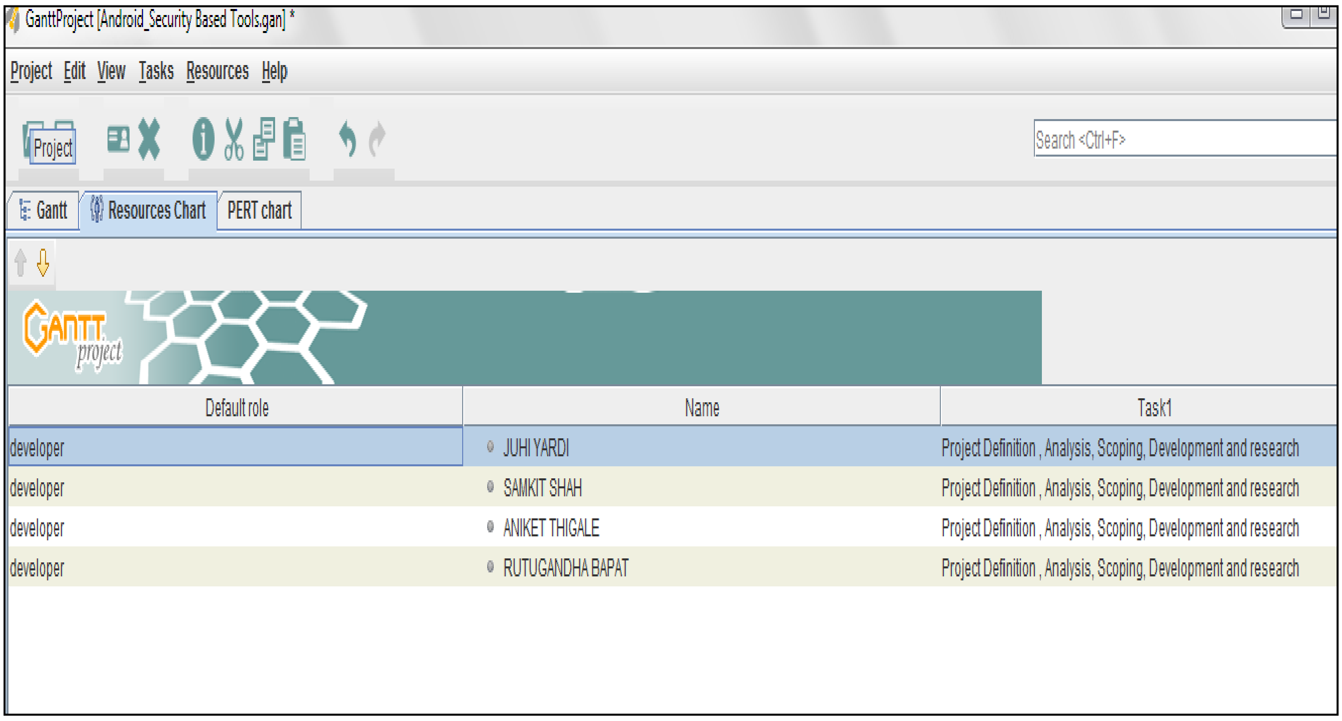
Testing
Start: 4/9/14
End: 4/21/14
Duration: 12

TIME CHART:



Name	Begin date	End date
Selection Of Project Topic :ANDROID S...	6/28/13	7/8/13
Finalization Of Project Topic:ANDROID ...	7/18/13	7/19/13
Research into existing Platforms : AND...	7/20/13	8/10/13
Finalization Of Problem Definition: COM...	8/15/13	9/20/13
Finding Scope Of Problem Definition	8/22/13	9/30/13
Detailed Design	9/30/13	11/15/13
Training and Setup of Work Environment	11/15/13	12/1/13
Development	12/5/13	12/5/13
Preliminary Proof of Concept	2/16/14	3/5/14
Necessary Modifications made	3/8/14	3/8/14
Testing	4/9/14	4/20/14

Resource Chart:


GanttProject [Android_Security Based Tools.gan] *
Project Edit View Tasks Resources Help
Project
Search <Ctrl+F>
Gantt
Resources Chart
PERT chart
GANTT project

Default role	Name	Task1
developer	JUHI YARDI	Project Definition , Analysis, Scoping, Development and research
developer	SAMKIT SHAH	Project Definition , Analysis, Scoping, Development and research
developer	ANIKET THIGALE	Project Definition , Analysis, Scoping, Development and research
developer	RUTUGANDHA BAPAT	Project Definition , Analysis, Scoping, Development and research

Project Schedule:

Sr. No.	Deliverables	Submission Date	Review Date
1	Group Formation	2nd Week of July	
2	Synopsis	3rd Week of August	4th Week of August (17th Aug – 3st Sep.)
3	Guide Allocation	1st Week of Sept.	3rd Week of Sept.
4	Detail problem Definition, Literature survey , Platforms, SRS(Software Requirement Specifications)	2nd Week of Sept.	2nd Week of Sep.
5	Project Plan	3rd Week of Sep	4th Week of Sep
6	High Level Design Document	1st Week of Oct.	1st Week of Oct.
7	Submission for I term Report consisting of(2,4,5,6)	3rd Week of Oct.	3rd Week of Oct.

8	Detail Design Document	2nd Week of Jan.	
9	Test Plan Review of Project with Demo	2nd Week of March.	
10	Report (Soft Copy)	1st Week of April	
11	Report (Hard Copy)	2nd Week of April	

Table : Project Schedule

ANNEXURE – D

Review of Design, Black box testing and corrective action in design

Black Box Testing
In this stage there will be testing for its functionality, performance and usability, Robustness and security.

Alpha Testing
Developer does this testing during development phase and it will be completed before delivering the software to the beta testers. First we will test the software using white box techniques. Additional inspection is then performed using black box.

Beta Testing
If a major bug is reported only that bug will be rectified before the release of the software.

Test Case 1

Test Description	To decompile apk and extract permissions

Test Cases	Expected Output	Actual Output
Extract Permissions	All the requested permissions by an application must be extracted from its androidManifest File.	All the requested Permissions by an application will be extracted from its androidManifest File except in cases of usage of wrong tags.
Categorize	The application must be assigned category from the set of pre-defined categories.	The application is assigned category, the first time it is downloaded / installed to the device by the user.
Intercept Downloading and Installation	The automatic installation of an application by the Package Installer must be hooked.	The probability of managing to hook between installing and downloading might not be allowed.

Steps	Decompile the application apk using dex2jar. Extract the permissions from the AndroidManifest.xml
Expected Results	Permissions given to the Android application are extracted successfully.

Test Case 2:

Test Case Name	Permission Checker
Test Description	To check if mapping of category and permissions is compatible
Steps	5. Let user select the category of the application from a set of values. 6. Use the extracted permissions from the androidManifest.xml 7. Map the category – permissions requested to the database. 8. Check for compatibility.
Expected Results	3. The developer should be able to determine the validity and authenticity of the repository. 4. Granting Permissions should be safe.

Test Case 3:

Test Case Name	Static Analysis
Test Description	Checking for malicious activities before application is installed on the Android device.
Steps	5. Android application is decompressed. 6. Main activity that can be launched is extracted from the manifest file. 7. Classes.dex is decompiled into a Java typical folder hierarchy containing files with easily parsed to pseudo code. 8. Finally disassembled code is scanned for suspicious patterns.
Expected Results	Some of the commonly known malicious patterns are detected before installation of the application and the users are warned.

Test Case 4:

Test Case Name	Dynamic Analysis
Test Description	Checking for malicious activities after the Android application is installed on the device
Steps	7. New application is executed for minimum of 60 seconds. 8. Android LKM catches system calls and logs their frequency. 9. Detection of phone calls made and SMS sent. 10. Generation of feature vector for analysis. 11. Feature vector is sent to server where malware db is first checked and then K-Nearest Neighbor and J48 decision tree algorithms are used to classify application. 12. User is notified.
Expected Results	Any malicious activity is to be detected during runtime is flagged to the user.

Test Case 6

Test Description	Checking for possible privilege escalation attack in the developed application
Steps	6. Application androidmanifest.xml is parsed 7. Manifest is checked for presence of permissions tag if any. 8. Every activity or service is then checked for intent filters properties, exported tags and the data and action in the filters. 9. Results are then analysed and graphical outputs are generated 10. Any attempt of privilege escalation attack if any is detected during this analysis phase
Expected Results	Graphical charts and tabular data showcasing the vulnerable sections of code.

ANNEXURE-E

Project Workstation selection and installation report

Module 1: (Android Application for Permission Checker)

1. Created in Eclipse using Android SDK Build: v21.0.1-543035.
2. OS: Windows 7.
3. Android Version: 4.2
4. Database: MySQL: 5.2.42 CE (server side) , SQLLite(on Android phone)
5. Dump file in \PHP_SQL folder.
6. Application requires internet connection.
7. Code Changes required: Change Server IP address in following address
 6.1. QueryServer.java: Line No. 43
 6.2 Service_intent.java: Line No. 44
8. Extra Libraries Imported: None.
9. Application Tested on: HTC 1.x, Ice-cream Sandwich 4.0.

Module 2: (Android Application to Alert User of Permission Escalation)

1. Android Application for alerting user of privilege escalation
2. Created in Eclipse using Android SDK Build: v21.0.1-543035.
3. OS: Windows 7.
4. Android Version: 4.2
5. Extra Libraries Imported: None.
6. Application Tested on: HTC 1.x, Ice-cream Sandwich 4.0.

Module 3: (Privilege Escalation & Capability Leak Detection Tool)

1. Java Web Application.
2. Apache Tomcat Server: 7.0.39
3. Uses MySQL : 5.2.42 CE database.
4. Dump file in \PHP_SQL folder.
5. Created in NetBeans I.D.E 7.2
6. Tested on Google Chrome/31.0.1650.63
7. Tested OS: Windows 7
8. External Libraries Imported: MySQL JDBC Library: 5.1.18.jar

Module 4: (Dynamic Analysis)

1. Download and install Oracle JDK.
2. Download and install the Android SDK+Eclipse bundle from the Google Android site.
3. Install 32-bit dependencies by :
sudo apt-get install ia32-libs
4. Download and extract Android NDK r9c cross compilation toolchain.
5. Download goldfish 2.6.29 kernel source code and android platform 4.0.3_r1 platform source code

6. Download the CodeSourcery Lite G++ ARM Toolchain from
   http://www.codesourcery.com/sgpp/lite/arm/
   After downloading the package open a terminal and go to the directory where the file is saved and type:
   $ chmod a+x arm-2010.......bin
   $ sudo ./arm-2010.........bin
   64-bit Linux users must make a few changes to their system. You can read them here on the Codesourcery website!
   If you get an error message in the terminal, type:
   $ sudo dpkg-reconfigure -plow dash
   And in the window that appears, choose No, and rerun the ./arm-2010.....bin command.
   During the installation select the following options:
   Typical installation

7. For compilation ofgoldfish kernel :

```
make ARCH=arm goldfish_armv7_defconfig
sudo apt-get install lib32ncurses5-dev
make ARCH=arm
make SUBARCH=arm
export CROSS_COMPILE=/usr/local/Codesourcery/bin/arm-none-linux-gnueabi-
make goldfish_armv7_defconfig
make menuconfig
make -j4
```

8. An example MakeFile for compiling kernel module:

```
VERSION = 2
PATCHLEVEL = 6
SUBLEVEL = 29
EXTRAVERSION =
obj-m += hijack.o

KDIR = /home/aniket/beproject/htconev/android/goldfish
PWD := $(shell pwd)
all:
        make -C $(KDIR) ARCH=arm CROSS_COMPILE=/usr/local/Codesourcery/bin/armnone-linux-
        gnueabi-SUBDIRS=$(PWD) modules



clean:
        make -C $(KDIR) ARCH=arm CROSS_COMPILE=/usr/local/Codesourcery/bin/armnone-linux-
        gnueabi-SUBDIRS=$(PWD) clean
```

9. Download the play framework 2.2.0 and Scala SDK+Eclipse Bundle.

10. Create an AVD using AVD manager for platform 4.0.3 and SDCARD = 2048 MB and name armor_v1.

```
eg) avd create -n armor_v1 -t 2
```

11. Start AVD using compiled kernel :

```
 emulator -kernel
/home/aniket/beproject/htconev/android/goldfish/arch/arm/boot/zImage -show-kernel verbose
@armor_v1
```

12. Start the play server.
13. Use following steps to load kernel module.Change directory to the folder containing the kernel module C code:

```
 aniket@ubuntu:~$ make
 aniket@ubuntu:~$ adb push hijack.ko /sdcard/hijack.ko
 aniket@ubuntu:~$ adb shell
 #insmod /sdcard/hijack.ko
```

To check if module has been loaded successfully :

```
 #lsmod
```

14. Then start our service.

ANNEXURE-F

Module development plan, Module descriptions

Module 1: Android Application for Permission Checker

Permissions are at the heart of the security mechanism in Android. Every application at its installation stage explicitly asks the user to allow it to use a certain set of permissions during its complete lifetime in the android environment. These permissions can either be granted in a bunch or not granted at all. If granted, these permissions reside with the application and can be used by it anytime without a notification to the user. Our study showed us that most of the applications demand for something extra which it does not need to fulfill its core functionality. Let us take an example: A simple game – my game in order to complete its core functionality should not ask for

1. RECEIVE_SMS
2. SEND_MMS etc.

Hence we developed a module which could keep a track of such extra permissions the application asks for during its install time and notify the user about the same. Our module consists of a service which runs on the end users phone. Also we have a server which has our initial permissions-map, category database. The module is in form of a client server pattern. Our module on the end user's phone does the following-

1. It is a service which will keep running in the background and will start once the phone is booted up.
2. It will keep checking for a install_app intent and fire up once it is found. This will tell the service when a new app is being installed.
3. The service will prompt the user to input the category of the application. It will then parse the AndroidManifest.xml of the app being installed and extract the permissions it currently asks for.
4. This set of permissions is converted into a series of bits to match the 144 permissions currently defined in android (Gingerbread). (Bit string has been taken to be 160 as of now in order to allow changes if new permissions are added with newer android versions).
5. 1 in the bit string corresponds to a permission defined and 0 for the one not defined.
6. The service then extracts the reference bit string from the server for the particular category of the application.
7. It compares the two result sets and checks for anomalies.
8. Any difference in the two bit sets tells us exactly which extra permissions the app demands for.
9. Result is then prompted to the user along with the extra permission name and description.
10. The user can now take corrective action. App can be uninstalled.

FLOWCHART

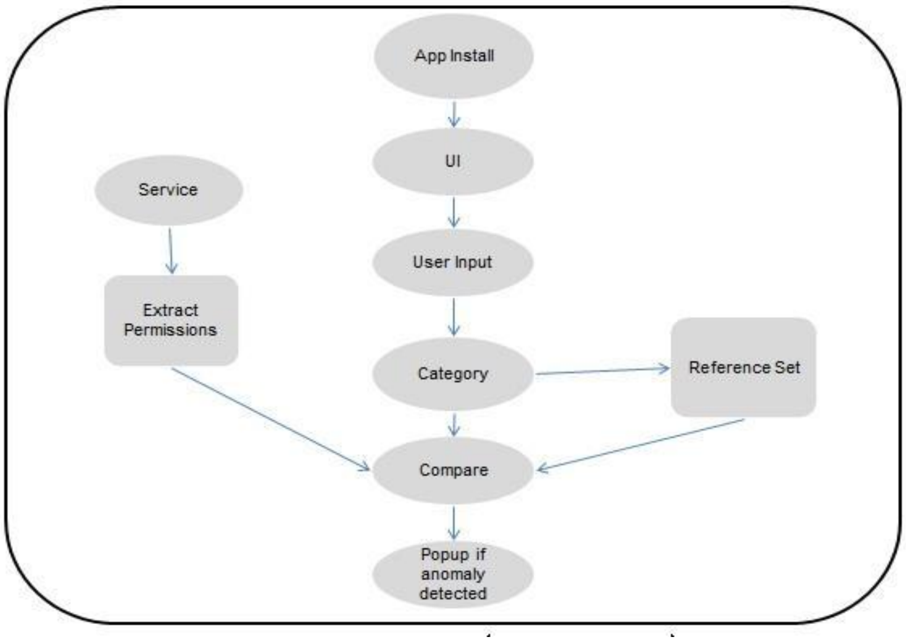


Fig 1: Static Analysis –Permission Mapping

The applications in android necessarily fall into at least one of the 27 broadly classified categories. Categorization is based on the domain to which the apps belong and their functionality such as Comics, Communications, Finance, Health, Music & Audio etc. The reference set (database) has been formed by carefully studying around 35-40 applications taken off the Google play in every category and extracting their permissions from their manifest.

MODULE 2 : Android Application to Alert User of Permission Escalation

This application consists of service that can be combined with the above module 1 to enhance the completeness of permission requesting by Package Installer at install time. We have found out that an application while on its own may not explicitly request for standard permissions, it may escalate its capabilities by communicating with another capable application. Currently the Package Installer alerts this capability escalation at installation time by specifying "Default Permissions". The ambiguity is handled by this service that recursively lists the permissions of applications that the current (to be installed) application communicates. Hence at install time the user is alerted of possible permission escalation. Hence Module 1 and Module 2 are jointly proposed changes to the Package Installer provided by Android to strengthen the Permission based Security system in android.

MODULE 3: CAPABILITY LEAK DETECTION & PRIVILEGE ESCALATION ATTACK DETECTION

Although applications are limited to perform actions that they are allowed to perform by the users at install time, they could acquire extra capabilities, which are the abilities to perform various actions like making phone calls, by exploiting interfaces exposed by more capable applications. The exposure of such interfaces may not be intentional though. It is hard for naive users to find out whether or not an application is exposing its capabilities. One way of tackling this problem is to look at the source code of the applications and and find if they are free from vulnerabilities. The tool is a static program analysis tool. It performs static taint checking to track the data flow in the application to find out data paths that represent capability leaks.

An application with less permission (a non-privileged caller) is not restricted to access components of a more privileged application (a privileged callee). Such attack is called privilege escalation attack or confused deputy attack. It essentially allows a potentially malicious application to indirectly obtain extra capabilities leaked from a benign application which did not do a good job to protect the permissions granted to it.

Checking for privilege escalation attack:

1. The tool parses the AndroidManifest.xml file which clearly defines two things.
a. the permissions an application needs.
b. the permissions other applications need to have so as to interact with the components of that application.
It is mandatory that every Android application includes the AndroidManifest.xml in its Android package file. Such a rule is enforced by the Android system.
2. It identifies components that are potential sources of capability leaks.
3. For each of such components, it looks into its source code and uses inter-procedural control flow searching to follow the taint propagation.
4. It then obtains data paths that lead to capability leaks.
5. It alerts and gives graphical and tabular results whenever such paths are found.

Capability leaks detection:

1. The APK file of the Android application to be analyzed is first converted into a JAR file.
2. The manifest file (AndroidManifest.xml) is extracted from the JAR file for further inspection. The manifest file defines the permissions an application uses and the permissions other applications need in order to access the components of that application.
3. A list of components that have the potential of leaking capabilities is then obtained.
4. These are the components that have extra capabilities, are publicly accessible, and are not guarded by any permission.

5. Finally using inter-procedural control flow components that will lead to capability leaks are found by the following checking policy
a. The application uses at least one permission and
b. There exists an activity or service component that is not protected by any permission and is publicly accessible.
If the above two conditions are satisfied, those components which satisfy the second rule has a potential of a capability leak.

MODULE 4: DYNAMIC ANALYSIS

We propose a system which handles threats at two levels which are application level & kernel level. We have also made use of a client server architecture. At the kernel level we will perform
dynamic behavioral analysis by hijacking the system calls using an Android Loadable Kernel Module made by the application running instances on the device and using machine learning algorithms to classify the applications as benign or malware on the server side.

We have used Android 4.0.3_r1 platform source code downloaded from the Android Development site and the goldfish kernel source for building the kernel for the emulator.

There are two components in our module:
a. Android Loadable Kernel Module
b. Android Service

The Android Loadable Kernel Module catches various system calls like read, write, open, close, execve etc. made by an application during its first launch and logs their frequency for a 60 seconds time duration them and returns this data to the service from kernel space to user space via Netlink Sockets. At the present only 32 system calls are caught by the LKM.

```
/* Back up pointer towards orginal sys_read() method */
asmlinkage long (*original_call_read) ();


/* Hooked sys_read() method definition */
asmlinkage long our_sys_read(unsigned int fd, char  *buf, size_t count)
{
 uid_t gtuid ;
 gtuid = getuid_call();
 if(gtuid==c_uid)
  { printk("our_sys_read ---> uid = %d  \n ", gtuid); counter[21]++; }
 return original_call_read(fd,buf,count);
}
```

```
//In Init Module
//Set Address of system call table
sys_call_table = (void*)0xc0027f44;
//read
original_call_read = sys_call_table[__NR_read];
sys_call_table[__NR_read]  = our_sys_read;

//In Exit Module
sys_call_table[__NR_read]  = original_call_read;
```

Eg) LKM catching system call read

Our service starts at boot time and keeps running in the background till device shutdown. It catches a new application launch and prepares a feature vector out of the package name, permissions bitstream and the data returned from the kernel. Also the service checks if the application made any phone calls or sent any SMS and appends two bits regarding the same to the vector. We calculate the MD5 checksum of this vector and send them both to the server for classification of the application via HTTP Post request. At the server side MD5 Checksum is again calculated to check that the feature vector data is intact. Then K- Nearest Neighbor and J48 Decision Tree algorithms are run on the vector to classify the app, whereon, the response is sent back to the device and the user is notified.

A. K – Nearest Neighbor Algorithm
1. Go through each item in dataset and calculate the distance from that data item to my specific sample.
2. Classify the sample as the majority class between K samples in the dataset having minimum distance of the sample.

B. J48 Decision Tree Algorithm

1. Check for base cases
2. For each attribute a
Find the normalized information gain ratio from splitting on a.
3. Let a_best be the attribute with the highest normalized information gain.
4. Create a decision node that splits on a_best
5. Recurse on the sublists obtained by splitting on a_best and add those nodes as children of node.

C. K – Means Clustering Algorithm

1. Select k random instances from the training data subset as the centroids of the clusters C1; C2; ...Ck.
2. For each training instance X:
a. Compute the Euclidean distance D(Ci,X), i=1...k:Find cluster Cq that is closest to X.

b. Assign X to Cq. Update the centroid of Cq.

3. Repeat Step 2 until the centroids of clusters C1;C2;...Ck stabilize in terms of mean-squared- error criterion.

4. For each test instance Z:
a. Compute the Euclidean distance D(Ci,Z), i=1...k. Find cluster Cr that is closest to Z.
b. Classify Z as an anomaly or a normal instance using the Threshold rule

5. Assign Z-->1 if P(w|Z) > Threshold. Otherwise Z = 0 And Z = 1 where 0 and 1 represent normal and malware Classes.