Generating smartphone phishing applications for deception based defense

BY

KRUTI SHARMA

B.E., Rajiv Gandhi Proudyogiki Vishwavidyalaya, India, 2010

THESIS

Submitted as partial fulfillment of the requirements

for the degree of Master in Computer Science

in the Graduate College of the

University of Illinois at Chicago, 2017

Chicago, Illinois

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor Professor Mark Grechanik for giving me the opportunity to work under him as his Research Assistant. Research field was entirely new for me and with his guidance I was able to work in this thesis.

I would also like to thank my committee members - Professor Aravinda Sistla, Professor Chris Kanich and Dr. Mohak Shah for taking out their valuable time and served as committee members for my Master's thesis.

Contribution of Authors

The subsections: *Overview of GUI* and *Background on the GUI Structure* have been referenced from Dr. Mark Grechanik's previously published work [45, 66–69, 123] and Dr. Grechanik has authorized me to use the content as published by him in my master's thesis. The subsection Overview of GUI gives a general idea about the interface used across all the operating systems/platforms and helps my thesis work to give a general idea about the interface. The subsection Background on the GUI Structure helps my thesis work to elaborate on the general architecture of interface across different platform.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

MA^{2}P	Mobile Assistive APplications
UWD	Users With Disability
AS	Accessibility Services
GUI	Graphical User Interface
GAP	Graphical User Interface- based APplication
App	Application

SUMMARY

Graphical User Interface (GUI)-based APplications (GAPs) are ubiquitous, both in business and personal use and they are deployed on diverse software and hardware platforms. Unfortunately, close to 50Mil people have disabilities in the USA alone and over 600Mil worldwide, and it is difficult for *Users With Disabilities (UWDs)* to work with GAPs on their smartphones. Since there are hundreds of disabilities that impair people in vision, movement, thinking, remembering, learning, communicating, and hearing, UWDs need specialized enhancements to GUIs. *Mobile Assistive APplications (MA^{2}Ps)* provide these enhancement services using specialized accessibility technologies that are fundamentally insecure, thus exposing all smartphone users to a variety of attacks. The goal of this framework is therefore to investigate security problems with accessibility technologies and to explore a novel theoretical foundation to allow stakeholders to create, analyze, and predict the security and privacy behavior of complex MA^{2}Ps for UWDs.

A connecting thread in the research thrusts is a combination of GAP and MA^{2}P modeling and compositional intercomponent analysis using these models to create a prototype that can predict and mitigate security threats posed by accessiblity technologies for smartphone users. Also, the results of the proposed framework should inform the GUI security and assistive technologies communities about the possibilities and limits of program analyses and machine learning in dealing with security problems posed by accessibility technologies that make users unsafe.

Key Words: graphical user interface, security, program analysis, assistive technologies, smartphone apps

1. Introduction

All major operating systems on smartphones provide the accessibility service for creating *Mobile-Assistive APplications (MA^{2}Ps)* for *Users With Disabities (UWDs)*. Since there are hundreds of disabilities that impair people in vision, movement, thinking, remembering, learning, communicating, and hearing, UWDs need assistance in using *Graphical User Interface (GUI)-based APplications (GAPs)* by enhancing their GUIs [45, 66–69], and MA^{2}Ps provide these enhancement services. Google Play Store has thousands of applications that requests for `Accessibility Service` [80, 85] and two applications which have a rating of 4.3 out of 5 [104, 116] are popular and have been downloaded by more than ten million users. Making GAPs accessible for UWDs is extremely important nowadays [97], since GAPs are ubiquitous, both in business and personal use and they are deployed on diverse software and hardware platforms [34, 77] including TVs and cars [37]. Making GAPs accessible for users with various types of disabilities is legally required [63, 92], and a large MA^{2}P marketplace exists where there is no control over how applications use accessibility services.

On the 4th March, of the past year 2016, it was announced that an evolutionary malware that exploits the accessibility service using *clickjacking* impacted more than half a billion Android devices globally [29,30,42,61,87,90,108,109,128,129]. A report at the annual cyber-security RSA conference stated: "this very modern mobile malware had the capability to not be detected in scanner detection, which is usually based on signatures, static and dynamic analysis approaches." The malware can steal all sensitive information and take automated actions using other applications and even the operating system. On the 5th of May 2016, Symantec announced that Android banking malware tricks users into turning on the accessibility service, so that it can steal sensitive information from *all users*, not just from UWDs [126].

Moreover, security researchers from Symantec reported that the protection that Google released in May was not fool proof [127]. As of the 18th of May 2016 it is confirmed that the attack is not limited to older versions of Android and over 95% of Android devices are under this rapidly expanding threat [59]. Different attacks that use accessibility services were reported as early as in November last year [46] although security weaknesses of the accessiblity services were discussed as early as 2014 [88], and these attacks are expanding on a massive scale. If accessibility services are enabled, then applications with accessibility permissions are allowed to interact with GUIs on behalf of the user and do almost anything that the user can do and currently there is no protection against this attack.

1.1 Mobile app Overview

A **Mobile App** is a software application designed to run on mobile devices such as smartphones and tablet computers [121]. These apps allow users to access different applications easily on their smartphones like calendar, email etc. Different variety of platforms/operating systems like IOS, Android, Windows, Blackberry have an in-built support for many apps and extra set of apps may be installed from app stores. These apps are designed to provide users with services that they can access on their PCs [17]. Each app has specific and limited functions that it can perform and is designed to support the limited hardware and memory resources of a smartphone.

Figure 1: Mobile App Example [15]

Each smartphone can have any number of apps (highly depends on the memory and support given by each smartphones platform) ranging from providing weather forecast services to allowing users to access their mailbox. For example, as shown in Firgure: 1, there can be mulitple apps with some in-built provided by the platforms and other may be downloaded. The popularity of Google's Android has been increasing and is popularly accepted operating system for many smartphones [21]. There are around 2,600,000 apps [18] available for users to download from Google Play Store - the official market for publishing and downloading android apps. The variety of apps available for user's ranges from basic apps such as calendar, email to important apps such as banking/financial apps. The apps are designed specifically for smartphones, providing a compatible view and an easy one click for accessing them.

But how do the services and operations of a website varies on laptop and smartphone. There are various differences like:

(1) The memory, computing power, hardware and software resources of a smartphone is very less as compared to a laptop or PC. A smartphone generally has 512 MB - 6 GB of RAM, processors of very less computing power and small screen size ranging from $2.5''$ to $4.8''$. Thus loading a website on

smartphone and PC is not always same.

(2) Accessing a website on a PC mostly requires navigation either through keyboard or through mouse, whereas for a smartphone the main access is through touch events. So a button click on PC is basically a touch event which is equivalent to a click event on smartphone. But for events like hover - which may be used by many websites to show information just on hover (like when a mouse pointer is pointing to some text/button/link, extra information may be given about that element) does not works for smartphones as there is no hover event in smartphones.

(3) Many websites require extra plugins for supporting video or music streaming which may not be supported by smartphones browsers due to limited resources.

Many websites are specifically desgined with smartphone compatible view which require less memory and can work with limited resources. Let us explore an app, CitiBank Mobile App. The app is used by user's holding a CitiBank account and have netbanking enabled. The app gives a quick overview to the logged in user about their account, the balance for all associated accounts including checking, savings and credit cards. All other services which are provided by the CitiBank website (`www.myciti.com`) (like: adding account, transferring money etc.) are supported by the app too. A user can access their CitiBank account through the website also on their smartphone provided the smartphone has a browser (e.g.: chrome or any other browser supported by smartphones). But these apps are easy to use because the user can just click on the app to start interacting with CitiBank directly. Whereas when a user tries to access the website for CitiBank on their smartphone, they must first have a browser compatible with their smartphone and then they can open CitiBank website (i.e.: `www.myciti.com`. A major challenge with accessing these websites on smartphones is their compatibility. Sometimes the website may not have a smartphone compatible view i.e. the a website may not load properly on a small screen of the smartphone, may have overlapping content and many services and operations may not work in the desired way due to the limited resources in smartphone.

Thus apps play an important role in smartphones allowing users to access various services and they are designed as per the smartphones platform i.e. small screen, limited hardware, software resources and low processing power. Figure: 2 is a small snapshot of how CitiBank app looks like: (after the user has logged in)

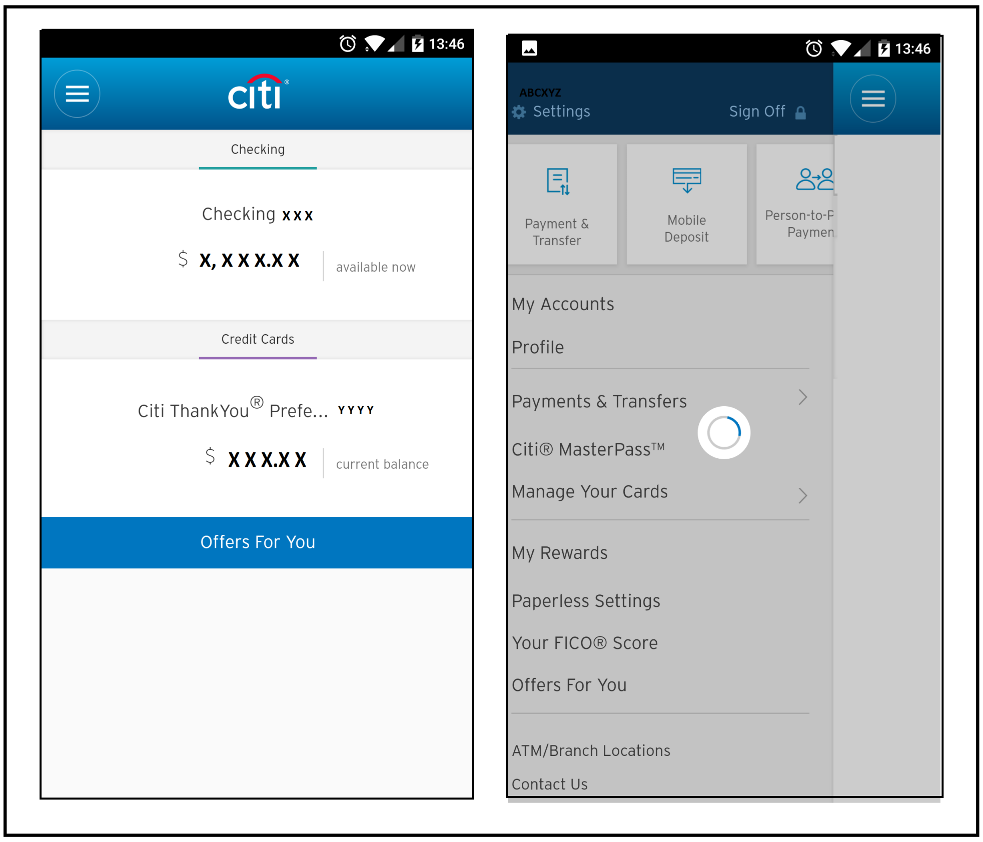


Figure 2: CitiBank Mobile App Snapshot

Each of the platforms (IOS, Android, Windows, Blackberry etc) have an official store for downloading their platform compatible apps. Google Play Store is the official store for downloading android apps, Apple Store for IOS apps , Windows Store for Windows apps etc. With the third party support and open nature of Android, there are many unofficial party stores that allows users to download and install android apps. The apps available on these stores can be downloaded for free or paid. The apps available on other third party source are generally not verified and any malwares or viruses or even apps (can perform unauthorized operations) can be easily distributed through these sources. The open source nature of android allowing installation of apps downloaded from third party/unverified sources makes it a target. Even though Google Play Store authenticates each app distributed from their store, still certain malicious apps can take advantage of side channels such as advertisements, using these as a medium and tricking the users to install some other packaged applications or perform some other operation from behind the scenes.

1.2 Overview of GUI

A GUI framework is a reusable and extensible set of GUI objects with well-defined interfaces that can be specialized to produce and to run custom GAPs [99]. A schematics of a model of GUI frameworks is shown in Figure 3. A purpose of this model is to show that there are four main components in GUI frameworks: GUI representation layer, GUI object library, GUI interfaces and the accessibility layer. GUI representation layer defines how GUI objects are represented programmatically as data structures in computer memory. For example, HTML pages are represented using a generic document object model in browsers while programming documentation for Windows defines proprietary data structures for GUIs and the events that they receive and send. The visualization/graphics engine interprets these data structures and visualizes them using some predefined settings for styles and layouts.

To allow users to interact with GUI objects, the underlying operating system or a virtual machine provides queues for receiving user inputs from peripheral devices (e.g., mouse, keyboard or a touch screen) and translates these inputs into event data structures that are passed to the corresponding GUI objects using its interfaces. GUI object libraries contain implementations of GUI objects and expose their interfaces; these libraries are extensible and many third-party vendors offer implementations of sophisticated GUI objects for different GUI frameworks. In general, GUI libraries developed by different vendors expose diverse interfaces.

Acessibility technologies provide assistance to Users With Disabilities in many ways [63] [103], for example: screen reader helps in assisting users with visual impairment, users with hearing loss are assisted via captions or visual indicators and many software compensate for users with motion impairments. In order to meet the requirements of *Electronic and Information Accessibility Standards* [63], accessibility technologies are provided by many platforms/operating systems. For example, applications running on Windows [50, 70] provide assistance to its users via *Microsoft Active Accessibility (MSAA)*, android has its own *Android Accessibility* framework to assist users in accessing mobile applications [2] on smartphones. With amendment of laws [4], [3], many latest platforms/operating system incorporate accessibility technologies which helps all the users to access applications on these platforms.

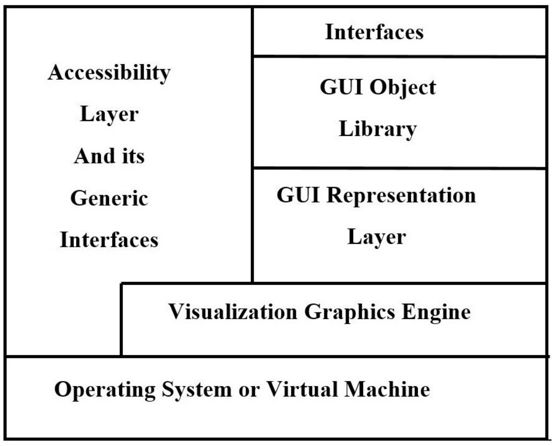

Accessibility Layer And its Generic Interfaces
Interfaces
GUI Object Library
GUI Representation Layer
Visualization Graphics Engine
Operating System or Virtual Machine

Figure 3: Framework Model

In order to retrieve the GUI objects, accessibility technologies provide services which can be used to set and retrieve the values of each GUI object [123]. A key element of accessibility technologies is that they provide a generic set of *Application Programming Interface (API)* calls that different GUI libraries should implement as a common programming standard for applications that use the accessibility layer. For example, for any Windows GUI objects to be accessible and controlled through MSAA API calls, it must implement the `IAccessible` interface [69, 123] . This helps the programmers to access and control the GUI objects of GAPs in their code like as if they are accessing the objects of a standard programs.

1.3 Background on the GUI Structure

Chapter 4.3 The Structure of GAPs (Previously published as K. M. Conroy and M. Grechanik and M. Hellige and E. S. Liongosari and Q. Xie (2007) Automatic Test Generation From GUI Applications For Testing Web Services, pages=345-354.)

In event-based windowing systems (e.g., Windows, Android), each GAP has the main window (which may be invisible), which is associated with the event processing loop [66, 123]. Closing this window causes the application to exit by sending a corresponding event to the loop. The main window contains other GUI objects of the GAP. A GAP can be represented as a tree, where nodes are GUI objects and edges specify that children objects are contained inside their parents. The root of the tree is the main window, the nodes are container objects, and the leaves of the tree are basic objects [45, 66–69].

While topologies of different GUI trees (i.e., the connectivity among its nodes) may be same, the actual GUIs can differ from one another because of the different types of GUI objects and their attributes. Each GUI object is assigned a category (class) that describes its functionality. For example, in Windows, the basic class of all GUI objects is the class `window`. Some GUI objects serve as containers for other objects, for example, dialog windows, while basic objects (e.g., buttons and edit boxes) cannot contain other objects and are designed to perform some basic functions. Thus, differentiating the GUI trees includes their topologies and labels of their nodes [45, 66–69].

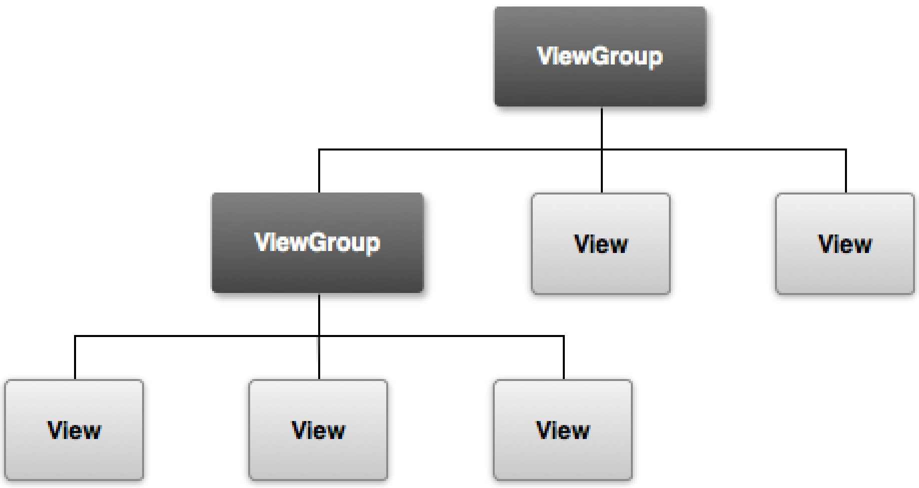


Figure 4: Android UI Hierarchy [12]

1.4 Android User Interface Overview

View and **ViewGroup** [12] forms the base for all the elements on the Android *User Interface (UI)*. The layout of an interface can be defined as: the ViewGroup holds the View and ViewGroup where *View* is an object with which the user can interact, for example: textbox, button etc. Android has a collection of subclasses of View and ViewGroup that allows user to choose specific controls like textbox, button etc. and different layout models like frame, linear, relative etc. Figure: 12 shows the basic hierarchy of GUI elements in and Android app.

The View and ViewGroup class are extended by several classes and these subclasses define the various control collection. The various subclasses of View are *ImageView, TextView, ProgressBar,* etc and for ViewGroup are *DrawerLayout, LinearLayout, RelativeLayout,* etc. In order to bind the elements on an interface for developing any mobile app, Android Studio can be used as the Integrated Development Environment (IDE) which has all the packages for different API levels and android versions. The IDE has several shortcuts for creating activities and generates codes for certain commonly used layouts such as Login Apps. Android apps have java as their base language. The compiled code, data and resources are packaged into a single file knows as **Android PacKage (APK)** file which can be installed on an android smartphone [9].

In order to understand how an android apps function, let first look into the basic building blocks for Android [9] :

① **Activity** - An Activity is the interface of an android app and the entry point for any user to start interacting with the app. An activity can be created either using XML or programmatically using java language. Generally the developers use XML to generate the activity but when the activity needs to load controls dynamically or bind some custom controls, then developer's opt to code for those controls programmatically. Listing: 1 shows a sample of Activity created using XML.

Listing 1: Sample Activity Creation Code in XML

```xml
"<?xml version="1.0" encoding="utf-8"?>"
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="example.com.seaphish.uicarise.writenodesxml.SimpleActivity"
    android:baselineAligned="false"
    android:orientation="vertical"
    android:id="@+id/mainLayout">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="20dp"
        android:text="Hello, Welcome!" />

    <<TextView>
        <android:layout_width="wrap_content">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:text="User Name"/>>

    <<EditText>
        <android:layout_width="match_parent">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:id="@+id/etUserName"/>>

    <<TextView>
        <android:layout_width="wrap_content">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:paddingTop="10dp">
        <android:text="Password"/>>

    <<EditText>
        <android:layout_width="match_parent">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:id="@+id/etPassword">
        <android:inputType="textPassword"/>
</LinearLayout>
```

(2) **Services** - The Services component of android allows an app to work in background even when the user has switched or started interacting with another app. For example, a music player continues to play songs in background even when the user has locked the phone screen or starts interacting with some other screen. This component plays an important role for apps working which may require some long running tasks and hence does not requires the user to wait or keep the app open till the tasks are finished.

(3) **Content Providers** - Android provides a shared set of app data which can be stored on a file system, on server or SQLite Database where the app can access data and if proper permissions are requested and granted, other apps can query/access/modify this data. For example, an app may request to access/modify user's Contacts List, so if the permission is granted by the user to other apps, these apps can access the data through Content Provider.

(4) **Broadcast receivers** - This component helps the app to receive notifications even if when they are not running. This helps the apps to respond to certain events or announcements. Every app needs to register for broadcast events to get notified about the changes. For example, some app may require to restart whenever the smartphone is restarted or upgraded. Thus when a notification is sent by the system about smartphone restart or upgrade, these apps can again restart on receiving this broadcast.

For our SEAPHISH framework, we focus mainly on **Activity** and **Services**. All the configurations of registering all the components must be done in AndroidManifest.xml. Listing: 2 is the sample configuration code which must be added for each activity/service in order to be recognized by Android SDK: (Note: this is just a sample code which shows the configuration of an activity and service, does not contain all the android tags/options which are provided for activity)

Listing 2: Activity/Service config in AndroidManifest.xml [8]

```
<application>
    <!-- decalring activity -->
    <activity
        android:name=".MainActivity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme.NoActionBar" >
        ......
    </activity>

    <!-- decalring service -->
    <service
                android:name=".Service_WriteXML"
                android:enabled="true"
                android:exported="true"
                .....
          </service>
</application
```

The Service component of android can be extended to create different types of services. There are three types of services:

(1) Scheduled - When certain jobs or tasks needs to be executed at some point of time or after some event (as per the requirement) are known as Scheduled Services.

(2) Started - When an application component (like activity) calls the **startService()** function to initiate a service explicitly, the specified service is started and now it can run in background which can be used to run any long processes or any tasks as desired by the app. The service keeps on running even if the app is closed and generally stops itself on completing its tasks.

(3) Bound - A service that provides client-server interface allowing users to interact with the service component of an app , perform inter process communication (IPC) etc. are known as Bound

services. These services can be created when the application component calls **bindService()** and can run as long as the application component(s) is bound to it.

For SEAPHISH framework, we focus on Accessibility Service which are extended from Service. A more detailed explanation of these services is given in later section.

1.5 Architecture of Accessibility Framework in Android

The Accessibility Framework in the android operating system allows an android developer to interact with the GUI controls and simulate a human-user interaction which helps in assisting UWDs to interact with the GAPs. For example: **TalkBack** is Google's Accessibility Service that can be enabled on an android smartphone and used by vision-impaired in accessing all android apps and features. Vision-impaired users can use TalkBack to access android features via different feed backs like spoken word, vibration and other audible feed back which helps the user understand what is there on the screen, what are they touching and what can they do with it [71].

The developer's generally develop the apps to be accessible allowing it to be usable by any user including UWD's. The android list a set of simple design guidelines which can be incorporated in individual apps. For example, an app may be developed with wrappers i.e. the controls may not be exposed directly but at the same time these controls are fed with extra information which helps the accessibility layer to identify each controls function. On Analyzing and retrieving all the GUI Controls from CitiBank Mobile android app, we found that each control is wrapped and only exposed as android.view.View. This is the basic building block class for user interface controls and all the controls extends from this class. Thus when the accessibility layer wants to understand what the control is actually, they can fetch the content of **android:ContentDescription** tag along with values from other tag: **clickable**. So if there is a clickable text (a link), the androidContentDescription tag will contain the text for the link and it will have the clickable tag set as true.

1.6 Accessibility Service

Accessibility Service [7] runs in background and are mainly used to assist users with disability. **AccessibilityEvents** are received as a callback by Accessibility Services which are nothing but the user's interaction with the smartphone.

Accessibility Service can be extended and developers can create their own accessibility services which can be used either to serve the developer's app specifically or they may intend it to serve other apps installed on the smartphone. There are various events which the service receives as callbacks and as per the assistance required, these service can provide feed back to the users. Each service can register for specific or all events and can provide specific or all feed backs. Listing: 3 shows the configuration for the accessibility service which can be done in a separate xml.

Listing 3: Accessibility XML Config

```
<?xml version="1.0" encoding="utf-8"?>

<accessibility-service xmlns:android="http://schemas.android.com/apk/res/
        android"
    android:description="@string/accessibility_service_name"
    android:accessibilityFlags="flagRetrieveInteractiveWindows|
        flagRequestFilterKeyEvents|flagReportViewIds|
        flagRequestEnhancedWebAccessibility|flagRequestTouchExplorationMode"
    android:accessibilityFeedbackType="feedbackSpoken|feedbackHaptic|
        feedbackAudible|feedbackVisual|feedbackGeneric|feedbackAllMask"
    android:canRetrieveWindowContent="true"
    android:settingsActivity="com.example.android.accessibility.
        ServiceSettingsActivity"
/>
```

The Listing: 3 is a basic sample of XML configuration of Accessibility Service. The important tags present in Listing: 3 can be summarized as:

- **android:accessibilityFlags** : This tag specifies the flags for service indicating what operations or resources it can access from an app. For example: flagRetrieveInteractiveWindows specifies the service can retrieve the contents of interactive windows i.e. the window that is currently being used by user. Each flags are associated with certain set of Accessibility Events. These events are basically the callbacks which the service can receive. Any event like button click or screen scroll are specified in accessibility service and in developing a service, these events are registered in order to receive callbacks when user performs any action.

- **android:accessibilityFeedbackType** : This tag specifies the feed back or output that can be generated by service in response to an accessibility event i.e. any event received as a callback by the service on user's action. For example: feedbackSpoken is one of the feedback that provides a voice feedback to the user in response to an action performed (like clicking a button) by user on an apps window.

- **android:canRetrieveWindowContent** : This tag is set as true when the service wants to access all the GUI Controls on the interactive window. This flag when set true along with flagRetrieveInteractiveWindows gives the service ability to fetch all GUI elements.

Listing 4: Accessibility Service Configuration in AndroidManifest.xml

```
<application>
    <service
        android:name=".Service_WriteXML"
        android:enabled="true"
        android:exported="true"
        android:permission="android.permission.BIND_ACCESSIBILITY_SERVICE">
        <intent-filter>
            <action android:name="android.accessibilityservice.
                AccessibilityService" />
        </intent-filter>

        <meta-data
            android:name="android.accessibilityservice"
            android:resource="@xml/AS_XML"
        />
    </service>
</application>
```

The Listing: 4 shows the configruation for accessibility service is added in the AndroidManifest.xml like any other service but it in order to qualify the service as an Accessibility Service, it must bind the permission: `BIND_ACCESSIBILITY_SERVICE` and define the intent-filter specifying it as accessibility service. The meta-data tag specifies the xml configuration of service i.e. the xml as defined in Listing: 3, must be referenced in the meta-data tag as declared in Listing: 4 [16]

1.7 Accessibility Events

An accessibility service [6] can receive accessibility events as a callback which helps the service to identify the user's interaction with the smartphone. There are various accessibility events for each type of action that a user can perform on a smartphone. The events can be categorized as follows:

① For VIEW TYPES: events like TYPE_VIEW_CLICKED, TYPE_VIEW_LONG_CLICKED, TYPE_VIEW_SELECTED etc. These all events come under the click/touch event performed by user in accessing any element. For example: when a button is clicked or when a textview link is clicked.

② For TRANSITION TYPES: events like TYPE_WINDOW_STATE_CHANGED, TYPE_WINDOWS_CHANGED etc. These events occur when there is a change in the state of the screen i.e. of a user opens an app or there is some content loaded on the screen after some click event.

③ For NOTIFICATION TYPES: events like TYPE_NOTIFICATION_STATE_CHANGED. These events occur when there are any notification messages sent by either the platform (like phone restart) or by other apps (a calendar meeting notification)

④ For EXPLORATION TYPES: events like TYPE_VIEW_HOVER_ENTER, TYPE_TOUCH_INTERACTION_START etc. These events occur when a user starts interacting with the screen, like when a user clicks on edit text box, then first a hover enter event is fired showing the user is focusing on this element.

⑤ For MISCELLANEOUS TYPES: events like TYPE_ANNOUNCEMENT. These events can occur when there some general announcements by the platform like if the smartphones operating system is updated.

Each event has some properties associated that helps in fetching the information about the event being performed.Below are few events and their properties which we have used in developing our framework:

- TYPE_WINDOW_STATE_CHANGED: When there is a state change of window i.e. if a user scrolls the screen, opens a new app etc. shows a change in the state of window and this event is received.

- TYPE_WINDOW_CONTENT_CHANGED: This event occurs specifically when the content on the current window is changed i.e. when a user is interacting with a screen or opens an app, this event is also fired along with TYPE_WINDOW_STATE_CHANGED depending on the loading of the elements on interface. Some content like an image, may load after the entire screen is loaded.

• TYPE_VIEW_CLICKED: Whenever a user clicks on any button, textviews or any elements, this event is fired showing a click or touch event performed by user.

Properties: The properties associated with above (1-3) transition type events (mainly used in out framework) are:

• getPackageName(): Returns the package name of the current app being used by user. This helps to identify which app or screen the user is interacting.

• getSource(): Returns the current screen element. With the help of this, we can fetch all the elements of a screen by accessing and looping through this elements all children.

• getEventType(): Returns the type of action performed by user.

There are many other events and properties, which can be checked from [6]

1.8 Android Permissions Model

With the increasing popularity of android, users tend to download apps not only from Google Play Store but also from several third party app stores. While installing any app, the users generally ignore the list of permissions that the app is requesting. The Android's Permission System plays an important role in allowing the apps to work functionally correct on a smartphone and also protect the users sensitive data. Each app can request for a set of permissions from the user which allows them to access/modify the resources of smartphone. For example: Lyft is a popular app used by many people for booking rides. When the user installs this app for the first time, it asks for a set of permissions to be granted which ranges from accessing user's location, calendar, camera, sending/receiving messages etc. and generally the users tend to accept and grant all permissions to the app. Only the latest version of android ($>= 6.0$) allows the users to select the permissions which they wish to grant to the apps and can revoke some permissions as per their choice. But for the older versions of Android, if a user is intending to install an app, then it is mandatory for the user to grant all permissions requested by the app. With different types of apps requesting different sets of permissions, users tend to accept all the requested permissions blindly. Even though with latest version of Android, user's can revoke certain permissions, but this may lead to the app not functioning properly. With a large number of users still using Android $<= 5.0$ version [13], leaves a huge chunk of smartphones for open attacks.

The accessibility service largely bypass permission-based models. A general problem with permission-based security models is that the users do not follow the least privilege principle when granting permissions, and even in the absence of disabilities users cannot make informed decisions about which permissions are really needed. UWDs routinely approve the accessibility permission requested by the MA^{2}P, and it is given access to the accessibility *Application Programming Interface (API)*. In general, operating systems do not properly enforce permissions [23, 32, 33, 35, 44, 51, 54, 57, 64, 73, 83, 106, 111, 119]. As a result, GAPs obtain permissions that are unnecessary, and an investigation of of 940 applications showed that one-third are overprivileged [32, 53, 64, 78, 81, 98]. Moreover, studies showed that users often are not prepared to make informed privacy and security decisions to select from over 130 different permissions [55, 56, 78, 86, 94, 113, 115], especially considering high complexity of GUIs [110]. *A key component of the accessibility API calls is to simulate a user of the smartphone, so that programmers can mimic user interactions with GAP by accessing and controlling GUI objects programmatically from MA^{2}Ps. This fundamental strength of assistive technologies is also their fundamental security weakness, since it is very difficult to create general permissions that will not affect the usability of MA^{2}Ps when considering hundreds of different impairments of UWD*s [80].

This is what makes this security threat unique and impossible to address with existing security algorithms and techniques. The accessibility service creates a legitimate and a very powerful backdoor into the smartphone that cannot be easily disabled without affecting UWDs thereby violating the Americans with Disabilities Act of 1990. On the other hand, the current fast-spreading attack puts our financial and defense systems under direct threat. Understanding this attack and developing solutions is a task of the utmost urgency.

The specific permission of enabling AS for an app can be provided only by the users and in order to make the user understand about the possible threat, android alerts the user about the actions. But generally users neglect such warnings, thus leaving the smartphone susceptible to malware attacks. Once a user enables AS for an app, this app can utilize different accessibility events that have the capability to read and control the GUI elements of not only their own app, but can also interact with all the other installed apps on the phone.

1.9 The Attacker Model

Using the workflow that is shown in Figure 5 we show how MA^{2}Ps can attack UWDs by stealing sensitive information, which could cause serious harm to the individual, organization or company owning it if this information is compromised through alteration, corruption, loss, misuse, or unauthorized disclosure [118] [24, pages 137-156]. Removing the accessibility services from smartphones will prevent a wide array of security exploits [80]; however, a brute-force solution like this has a nasty downside. Unfortunately, close to 50Mil people have disabilities in the USA alone and over 600Mil worldwide, which makes it very difficult for them to use GAPs without accessibility services [60, 95, 96, 100]. It has been said by many distinguished people including Mahatma Gandhi, Hubert H. Humphrey, and Cardinal Roger Mahony, that the measure of a civilization is how it treats its weakest members. It is a formidable challenge to achieve a measure in which no UWD feels different from other users when working with GAPs to accomplish everyday tasks [39, 41, 47, 72, 91, 101, 102], and disabling the accessibility services is not an option. Even though the accessibility service is created for UWDs, recent massive security breaches showed that all smartphone users are the victims of the following attacks. Hackers do not just use accessibility services to target UWDs only, but rather the general population of smartphone users who use accessibility services to implement functionality that is otherwise not available.

(1) A MA^{2}P obtains the data from the GUI objects of some GAP, enhances this data for UWDs, and transmits it to the MA^{2}P server, thus releasing sensitive information in this data into the wild. Interestingly, the MA^{2}P may not be malicious, it may use the power of server offloading for additional processing of the data. In general, distinguishing clearly between malicious and unwanted behaviours is a very difficult problem [40,62,75,124]. Of course, the MA^{2}P that requests accessibility permissions may not even use the accessibility to provide assistive services, and it can masquerade as an assistive application to steals sensitive information.

(2) MA^{2}P developers use the accessibility API calls to inject threads and register callback functions (or hooks) which are basically the events or actions performed by users on GUI objects of some GAP. For example, MA^{2}P called InputObserver utilizes these hooks to measure UWD's usage statistics [52].

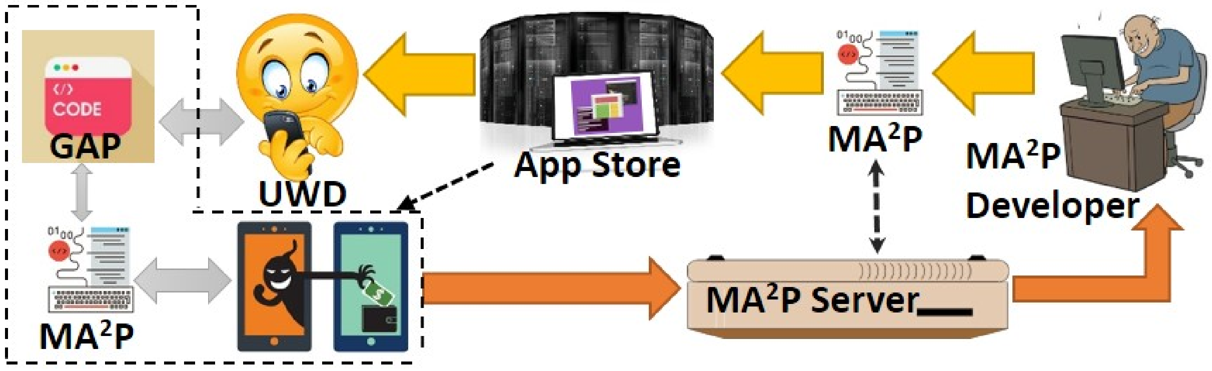


Figure 5: Using assistive technologies (MA^{2}Ps) in attacks on Users With Disabilities (UWD)

In general, such MA^{2}Ps may be viewed as a case of plagiarizing other smartphone apps [105] or a library modification threat [74]. Thus, an attacking MA^{2}P can avoid protection mechanisms that analyze it to determine if it tries to send sensitive data outside the smartphone [53]. Hacker tools like this have been used to attack desktop GAPs for a few years [48]. Moreover, recent sophisticated screen overlay malware lays screens over legitimate applications to capture users' inputs [128, 129].

③ MA^{2}Ps can compose others GAPs with access to smartphone on-board sensors to attack UWDs [49, 82]. It is an instance of the documented attack where sensory malware can convey raw sensor data (e.g., video and audio) to a remote server [112]. For example, consider a MA^{2}P that reads a financial statement to a UWD and it spawns a third party background application on the same smartphone that uses API calls to access the microphone and the phone hardware to capture the voice that is read by the MA^{2}P and to send it to an external server or even an answering machine. We easily realized this attack as it uses a display-centric text recorder known as Capture, which has an access to real-time foreground and background process of windows which integrates with the accessibility layer [89]. Of course, attacking sensors on smartphone is not new – PlaceRaider constructs rich, three dimensional models of indoor environments for remote burglars [117], but combining different GAPs under the guise of assistive technologies is a new type of attack. Currently, there is no unified approach to protect all users and not just UWDs from these attacks that the accessibility API calls while preserving usability, which is extremely important for UWDs [26].

1.10 Our Contribution

In our proposed of SEAPHISH framework for securing assistive technologies for smartphone users we will investigate whether it is possible to effectively protect users from security attacks. We advocate an untested, but a potentially transformative alternative research idea that will enable researchers to create solutions using a common abstraction from a unified perspective to protect UWDs while balancing compatibility with accessibility APIs, usability, and security. The proposed research program will consist of the following tasks.

Task 1: Define a novel abstraction for creating application-specific security models automatically. We abstract MA^{2}Ps and GAPs as relations that map their I/O endpoints using information flow that is computed and controlled via placing restrictions on dataflow paths automatically that connect the mapped I/O endpoints. Using these security models, we will investigate the applicability of different security analysis frameworks that will determine if MA^{2}Ps can reveal sensitive information of its UWDs. Moreover, whereas finding sensitive information leaks is important, using accessibility services enables attackers to navigate and control the GUI, which means that security-related settings can be modified, making it more than a dataflow problem.

Task 2: Create a prototype that uses obtained models to pinpoint security violations that can be caused by using accessibility API calls in MA^{2}Ps with a high degree of precision and to create and apply remedial actions to prevent release of UWDs' sensitive information. We will perform initial experiments and case studies to evaluate our prototype.

2. Background and Related Work

The UI Redressing attack [76] aims at presenting a sensitive UI element of target application out of context to the user. Thus when a user is interacting with the target application UI element like clicking or voice controlling, it is actually triggering some other action behind the scenes not intended by the user. The clickjacking attack by malwares tricks the users to click on a UI element for example: the Like button on Facebook and post some other feeds stating that the user ”Liked” this feed. The new InContext defense technique allows a user to ensure and verify their activity on sensitive UI element. This implementation helps in maintaining context integrity and defeat clickjacking attacks.

The security of smartphone GUI frameworks remains an important yet under-scrutinized topic. In this paper [43], we report that on the Android system (and likely other OSes), a weaker form of GUI confidentiality can be breached in the form of UI state (not the pixels) by a background app without requiring any permissions. Our finding leads to a class of attacks which we name UI state inference attack. The underlying problem is that popular GUI frameworks by design can potentially reveal every UI state change through a newly-discovered public side channel – shared memory. In our evaluation, we show that for 6 out of 7 popular Android apps, the UI state inference accuracies are 80-90% for the first candidate UI states, and over 93% for the top 3 candidates.

Graphical User Interface (GUI)-based APplications (GAPs) [65] are ubiquitous and provide a wealth of sophisticated services. Nontrivial GAPs evolve through many versions, and understanding how GUIs of different versions of GAPs differ is crucial for various tasks such as testing, cross-platform UI comparison and project effort estimation. We offer a novel approach for differencing GUIs that combines tree edit distance measure algorithms with accessibility technologies for obtaining GUI models in a non-intrusive, platform and language-independent way, and it does not require the source code of GAPs.

Leveraging openness of Android app markets and the lack of security testing, malware authors commonly employ a suite of widely available tools to facilitate the app development. Analysis of individual apps for malware detection often requires understanding of app functionality and complex, time-consuming analysis of its behavior. Since tools tend to leave traces in the program structure, we can potentially use visual exploration of these artifacts to enrich reverse engineering of malware analysis. In this paper [79], we focus on this approach and investigate internal structure of Android executable files and their characteristics under various tools and development conditions.

Android Debug Bridge [93] is a developer tool provided by Google which has the permissions to use critical resources. These may be used by developers to perform programmatic screenshots. On analyzing ADB permission which may be granted along with internet permissions can be easily exploited by malwares to stealthily steal sensitive and private data from user's smartphone. There is no third party API available for android developers to perform screenshot and thus enabling such permissions which are not well-guarded by android can be misused to peak into sensitive data like passwords through key strokes or bank cheques screenshots. Thus we present a mitigation mechanism which helps in exposure of ADB to only authorized apps.

The dynamic analysis of malicious applications on smarthphones are more comprehensive but running them in real time is expensive and a big overhead on the operating system. We devise a new static analysis [122] for detecting malicious applications. The AndroidManifest.xml file contains the information of all activities, permissions required by an app. Using this information along with the API calls, an application can be characterized for its behavior and with different algorithms modelling malware behavior, we can distinguish and detect Android malware.

Many Android vulnerabilities share a root cause of malicious unauthorized applications executing without user's consent. In this paper [28], we propose the use of a technique called process authentication for Android applications to overcome the shortcomings of current Android security practices. We demonstrate the process authentication model for Android by designing and implementing our runtime authentication and detection system referred to as DroidBarrier.

Malicious applications pose a threat to the security of the Android platform. The growing amount and diversity of these applications render conventional defenses largely ineffective and thus Android smartphones often remain unprotected from novel malware. In this paper [31], we propose DREBIN, a lightweight method for detection of Android malware that enables identifying malicious applications directly on the smartphone.

Some platforms, such as Android, are more open than others and are therefore easier to exploit than other platforms. In order to curb such attacks it is important to know how these attacks originate. We take a previously unexplored step in this direction and look for the answer at the interface between mobile apps and the Web. Numerous in-app advertisements work at this interface: when the user taps on an advertisement, she is led to a web page which may further redirect until the user reaches the final destination. In order to study such attacks we develop a systematic methodology [107] consisting of three components related to triggering web links and advertisements, detecting malware and scam campaigns, and determining the provenance of such campaigns reaching the user.

Driven in part by federal law, accessibility (a11y) support for disabled users is becoming ubiquitous in commodity OSs. Assistive technologies can be defined as computing subsystems that either transform user input into interaction requests for other applications and the underlying OS, or transform application and OS output for display on alternative devices. Inadequate security checks on these new I/O paths make it possible to launch attacks from accessibility interfaces. In this paper [80], we present the first security evaluation of accessibility support for four of the most popular computing platforms: Microsoft Windows, Ubuntu Linux, iOS, and Android. We identify twelve attacks that can bypass state-of-the-art defense mechanisms deployed on these OSs, including UAC, the Yama security module, the iOS sandbox, and the Android sandbox.

3. Motivation, Challenges, and Proposed Framework

The accessibility framework was developed by the android developers with an intention to support UWDs in accessing the apps on android smartphones. These services once enabled allows reading the text from the smartphone, perform actions on behalf of the user etc. But such enhancements and modifications in input output controls may have inadequate security checks and certain malwares or malicious apps can attack using these interfaces. The Skycure demonstration of a simple game which was able to trick users to give accessibility permission via clickjacking and read all content from any apps used by user shows an attack which can leak personal data from a users smartphone . This was demonstrated for android with version <= 4.4 (between froyo - kitkat). The new smartphones although are not prone to these type of attacks, but accessibility framework can still be misused by malicious applications. The design of our framework is inspired by the below researches:

① MockDroid [36] is a modified version of the Android operating system which allows a user to ’mock’ an application’s access to a resource. This resource is subsequently reported as empty or unavailable whenever the application requests access. This approach allows users to revoke access to particular resources at run-time, encouraging users to consider the trade-off between functionality and the disclosure of personal information whilst they use an application. We demonstrate the practicality of our approach by successfully running a random sample of 23 popular applications from the Android Market.

The above implementation uses mocking of resources as a way to allow the users to revoke the access at run-time. They take advantage of the apps running with reduced functionality i.e. in the absence of network or resources. Our motivation is to design a framework that mocks not just the resources but the entire GUI of app dynamically i.e. mock the original app with a phishing app.

② The number of Android malware has been increasing dramatically in recent years. Android malware can violate users’ security, privacy and damage their economic situation. Study of new malware will allow us to better understand the threat and design effective anti-malware strategies. In this paper, we introduce a new type of malware exploiting Android’s accessibility framework and describe a condition which allows malicious payloads to usurp control of the screen, steal user credentials and compromise user privacy and security. We implement a proof of concept malware [88] to demonstrate such vulnerabilities and present experimental findings on the success rates of this attack. We show that

100% of application launches can be detected using this malware, and 100% of the time a malicious Activity can gain control of the screen.

③ Digital conflicts are no different as the use of deception has found its way to computing since at least the 1980s. However, many computer defenses that use deception were ad-hoc attempts to incorporate deceptive elements. In this paper [27], we present a model that can be used to plan and integrate deception in computer security defenses. We present an overview of fundamental reasons why deception works and the essential principles involved in using such techniques. We investigate the unique advantages deception-based mechanisms bring to traditional computer security defenses. A successful deception should present plausible alternative(s) to the truth and these should be designed to exploit specific adversaries' biases. We investigate these biases and discuss how can they be used by presenting a number of examples.

④ Mobile applications are part of the everyday lives of billions of people, who often trust them with sensitive information. These users identify the currently focused app solely by its visual appearance, since the GUIs of the most popular mobile OSes do not show any trusted indication of the app origin. In this paper [38], we analyze in detail the many ways in which Android users can be confused into misidentifying an app, thus, for instance, being deceived into giving sensitive information to a malicious app. Our analysis of the Android platform APIs, assisted by an automated state-exploration tool, led us to identify and categorize a variety of attack vectors (some previously known, others novel, such as a non-escapable full screen overlay) that allow a malicious app to surreptitiously replace or mimic the GUI of other apps and mount phishing and click-jacking attacks. Limitations in the system GUI make these attacks significantly harder to notice than on a desktop machine, leaving users completely defenseless against them. To mitigate GUI attacks, we have developed a two-layer defense. To detect malicious apps at the market level, we developed a tool that uses static analysis to identify code that could launch GUI confusion attacks.

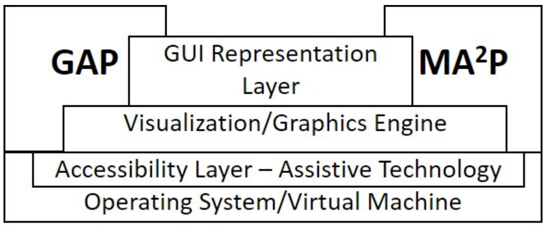
⑤ App-based deception attacks are increasingly a problem on mobile devices and they are used to steal passwords, credit card numbers, text messages, etc. Current versions of Android are susceptible to these attacks. Recently, Bianchi et al. proposed a novel solution [38] that included a host-based system to identify apps to users via a security indicator and help assure them that their input goes to the identified apps. Unfortunately, we found that the solution has a significant side channel vulnerability as well as susceptibility to clickjacking that allow non-privileged malware to completely compromise the defenses, and successfully steal passwords or other keyboard input. We discuss the vulnerabilities found, propose possible defenses [58], and then evaluate the defenses against different types of UI deception attacks.

The results and analysis from above papers reflect how AS can be exploited by malwares. The main challenge which lies in analyzing an app is: whether the app is actually harmful/malicious or not. Some apps may use the data (like user's current location) as per their requirement and some apps may maintain log. These information may never be used or apps may distribute it to third party advertisers, but still this does not quantifies the app to be malicious. Security leak can be different for different people. For example: TrueCaller is a very popular app used by many people to reveal a user's name if that number is not present in the user's contacts. The simple concept followed by TrueCaller is to access the contacts of each smartphone on which it is installed and this is how the entire database is built which helps them in showing the name of any number. The accessing and storing of each user's contacts list on their own server may be considered as a information leak and at the same time it may not be of a big concern. Thus simulating an actual malicious behavior and quantifying if an app is actually performing some serious unauthorized actions is challenging.

The research from the previous papers helps in uncovering the possible threats due to AS. We get many important insights about how AS can be exploited, but we aim to use the same AS to defend such malwares. Since AS allows to read the screen contents, it is possible to extract all GUI elements for each app which are accessed by a user. We can extract which app is being currently used, what are the controls on that screen and a sufficient information about the location of each control which can help us to build the phished app alone from this log. The app contains all the controls as exposed by AS. Each control has a set of properties which is required to bind that control on UI. Running the test of accessibility services on more than 30 apps, we found that most of the apps reveal all the controls bindings and even if the controls are wrapped, sufficient information is provided in tags such as **android:ContentDescription** which helps us to decode what this control is meant for. We propose the phishing app for deceiving a malware and analyze the actions performed by them on these phishing apps.

4. Solution

The research plan for this SEAPHISH proposal includes abstracting smartphone GAPs automatically for security analyses, and predicting security and privacy attacks using these abstractions. We concentrate on Android in this proposal, since it has a dominating share in the mobile market and Android malware accounts for over 90% of all mobile threats [114, 125]. Our proposal is at the intersection of the use of assistive technologies and smartphone GAP security, and little work exists at this intersection. Although a recent study with 14 visually impaired participants shows that their unique privacy and security needs remain largely unaddressed when using smartphone GAPs [25]. A GAP may use different GUI object libraries (e.g., Swing, TCL/Tk) that use services from the underlying GUI Representation Layer, also known as a windows manager that controls the placement and appearance of windows within a windowing system [120]. The lower-level Visualization/Graphics Engine paints GUIs on the screen. GAPs and MA^{2}Ps are generally run in separate protected memory address spaces from each other, and therefore the GUI objects cannot be accessed and manipulated as programming objects by MA^{2}Ps. These objects are a part of GAP and GAP has its own single process space which cannot be accessed directly but requires MA^{2}Ps to use GUI objects of GAPS through accessibility layers that is a part of all major operating systems.


GAP
GUI Representation Layer
MA²P
Visualization/Graphics Engine
Accessibility Layer – Assistive Technology
Operating System/Virtual Machine
Figure 6: A model of GUI accessibility.

4.1 Assumptions

Following are the assumption about Malicious application that we have assumed while developing this framework:

① Malicious applications are time-sensitive; once they ”believe” that they have access to sensitive data, they will attempt to use the data within a short period of time.

② Malicious applications do not perform any sophisticated control and data flow analyses on user’s smartphone as this will lead to significant performance problems that the users will notice and identify the malicious application as the offender.

③ Malicious application can use other applications (e.g., phone dialer or email clients) to transmit sensitive data from the smartphone to some other server, from which the attack may be hosted using the stolen sensitive data.

④ Once captured and transmitted, sensitive information can be used both from the victim smartphone and from some external computing device to mount an attack.

⑤ Malicious applications stealing sensitive information from GUI objects of victim applications do not use other form of attacks, e.g., rootkits or OS binary code manipulation as rootkit attacks generally require malicious app to modify the platform configurations and are generally distributed through unverified sources. We focus on apps which may exploit accessibility services or permission model to steal sensitive information.

⑥ The nature of sensitive data is that they can be used to perform operations like authorized transactions on the behalf of the victims to obtain material benefits. For example, stealing bank login information will lead to using the victim's banking application to login into the account to steal money or some malwares may deceive users by presenting a phished application instead of a real one where the user may believe its interaction with a real application but instead the application is fake and just steals the login information when the user interacts with this phished application. Stealing contact information will lead to calling these contacts or emailing them with spam. We also consider photo and video capturing GUI applications that can be manipulated into taking pictures of sensitive documents and transmitting these pictures to the attackers.

⑦ Malicious applications can analyze stolen data to determine if they are fake. For example, a malicious application may determine if the password is too long and thus is unlikely to be real or it can examine the state of the GUI after the user attempts to login to determine if the login action succeeded.

4.2 GUI Extraction

SEAPHISH generates a phishing application using the real application as its reference. The generated phishing app resembles the GUI of the real application, except its functionality is fake. The framework proposes to use android's accessibility service for extracting the GUI elements from any android app. As these services can run continuously in background once started by the user (till the user does not stops it manually again), we aim to extend this service for our framework. Once the user has installed SEAPHISH framework and enables the SEAPHISH's accessibility service, this service will be able to track the activities of smartphone. For the first time, when the SEAPHISH's accessibility service is started, following operations are performed in background:

(1) When the user enables SEAPHIHS's accessibility service, the **onserviceConnected** method (overridden in each class which extends accessibility service) is called. This method adds few banking apps like CitiBank, Chase Bank, Bank of America etc. into the tracking list of our service. For example: Listings 5, this sample code shows how we can add apps i.e. their package names to the collection list: packageName_Track or add these package names in info.packageName which allows us to monitor apps of our choice. Now our service will receive accessibility events from the smartphone for the apps added in the collection **packageName_Track** or in **info.packageNames**. We also propose to get the list of apps installed on the smartphone (using **PackageManager**) and add few commonly used and important apps automatically for tracking and monitoring. In general if we do not provide any filter in accessibility service for apps (i.e. package names for apps to be tracked), all activities performed by the user on smartphone will be tracked and monitored. In order to reduce the overhead of processing , we focus and monitor only few apps as per our choice. For each of these apps added into the tracking list, we will be able to fetch the window contents i.e. all the GUI elements.

Listing 5: onServiceConnected Sample Code

```java
@TargetApi(Build.VERSION_CODES.LOLLIPOP)
@Override
public void onServiceConnected()
{
    AccessibilityServiceInfo info = new AccessibilityServiceInfo();
    /* packageName_Track is a collection maintained to track the apps.
    It has the package names added of apps which will be monitored by this
        service */
     packageName_Track.add("com.citi.citimobile");
     packageName_Track.add("com.chase.sig.android");
    /* We could have also used the property of AccessibilityServiceInfo to
        track packages.
    For our intiial implementation, we are maintaining this in a separete
        collection.*/
    /* Tracking apps using the property of AccessibilityServiceInfo */
    info.packageNames = new String[]
        { "com.citi.citimobile","com.chase.sig.android"};
}
```

② The method : **onAccessibilityEvent** is overridden in each class extending from accessibility service to receive callbacks for accessibility events and then perform desired function. We override this method and focus on two specific events : window state changed and window content changed along with that we check if the user has opened an app which is being monitored by our service. For example: Listings 6, is a sample code showing how we override the method: **onAccessibilityEvent**. As seen in the sample code: if the current app opened by user is monitored by our service i.e. Line 14 which checks if the tracking package collection contains the package of the currently opened app, then only for the two events with eventType = 2048 or 32, we pass the source node (element) of current screen and iterate recursively on the source node to obtain all its children, then its children's children and till we fetch all GUI elements. Finally all these fetched GUI elements are written in a file with each app having its own file to maintain all elements.

Listing 6: onAccessibilityEvent Sample Code

```
1  public void onAccessibilityEvent(AccessibilityEvent event)
2  {
3      /* Get the source node when an accessibility event is received by the
           service*/
4      AccessibilityNodeInfo source = event.getSource();
5      /* Get the current event type*/
6      int eventType = event.getEventType();
7      /* get the package name of the current app being used by ther user */
8      if(event.getPackageName() != null && event.getPackageName() != "")
9          currentPackageName = event.getPackageName().toString();
10     /* if the current app used by user is being monitored by our service, then
           only we will track the elements
11         avoiding unnecessary processing on each event received */
12     if(packageName_Track.contains(currentPackageName)){
13         /* We track the elements of the app only when we have the below two
               accessibility events received
14             We maintain two seperate collections for maintaining all elements
                   from both events */
15         if (eventType == 2048) /*WINDOW_CONTENT_CHANGED */
16           {
17            if (source != null)
```

```
            {
                /* Collection to maintain all nodes for event = 2048*/
                allNodesInfo_2048 = new ArrayList<>();
                    allNodesInfo_2048.add(source.toString());
                /* function to recursively fetch all the GUI elements from the
                    parent element. */
                    getAllNodes(source, 2048);
                }
           }
        else if (eventType == 32) /*WINDOW_STATE_CHANGED*/
        {
            if (source != null)
            {
                /* Collection to maintain all nodes for event = 32*/
                allNodesInfo_32 = new ArrayList<>();
                    allNodesInfo_32.add(source.toString());
                /* function to recursively fetch all the GUI elements from the
                    parent element. */
                    getAllNodes(source, 32);
                }
           }
}}
```

4.3 Implementation details of Accessibility Service

Following are the details about each function implemented in SEAPHISH Accessibility Service:

① **Track Apps to be monitored** - SEAPHISH's accessibility service will be monitoring a set of apps which an individual user can specify. Any type of events like button click, text entered or extracting the GUI elements will be performed only for these specific apps. For now the interface is not developed for SEAPHISH and hence for evaluation purpose, we add the apps that needs to be monitored manually in the **onServiceConnected** method present in SEAPHISH's accessibility service. Adding the app for monitoring basically means adding the package name of that app to our monitoring list i.e. to the collection: packageName_Track.

② **GUI Controls Extraction** - The accessibility service receives callback events i.e. accessibility events like when a button is clicked or when an apps window loads or its window contents changes etc. For GUI extraction, we focus on mainly two events : `TYPE_WINDOW_CONTENT_CHANGED` and `TYPE_WINDOW_STATE_CHANGED`. Both the events occur when the GUI elements are being loaded on the screen or in simple words, when an app is opened by the user on their smartphone. Each of the events may trigger multiple times as per the apps functions to load the elements. We add all the GUI elements obtained from both the events in two different collections, one collection for window content changed and another collection for window state changed. Finally once the loading of GUI is completed, all the tracked GUI elements in both the collections (of both events) are written in a file. Adding and writing the GUI elements as obtained in the two accessibility event means: *write all the GUI elements properties as obtained from accessibility service*. Each apps GUI elements are maintained in a separate file (stored in the smartphone's memory which is few kBs) where the file name for each app is unique as it is derived from the package name of the app. Each screen of an individual app can be tracked provided there is a either a button click or text view click event.

Listing 7: Function to get all elements from parent element

```
public AccessibilityNodeInfo getAllNodes(AccessibilityNodeInfo source, int
    event)
{
    /* Once we get a node from accessibility , we check if its not null and if
        it has any more children*/
    if(source != null)
    {
        /* Get the child GUI element from the parent GUI object*/
        if(source.getChildCount() > 0)
        {
            /* iterate through all the nodes recursively i.e. from the parent -
            getChildNodes - for each of these child nodes again iterate to
            get their child nodes and so on till we  reach the leaf nodes
            i.e. nodes which have no children */
            for(int allNodes = 0 ; allNodes < source.getChildCount() ; allNodes
                ++) {
                if(source.getChild(allNodes) != null) {
                    if(event == 2048)
```

```
                        allNodesInfo_2048.add((String.valueOf(source.getChild(
                            allNodes))).replace("\n"," "));
                    else if(event == 32)
                        allNodesInfo_32.add((String.valueOf(source.getChild(allNodes
                            ))).replace("\n"," "));
                }
            /* iterate recursively to get all elements */
            getAllNodes(source.getChild(allNodes),event);
        }
....}}}
```

[5] Accessibility Service gives a detailed information about each GUI element and each of the fetched GUI elements have the type as **AccessibilityNodeInfo** which contains the following attributes about the GUI elements:

- **android.view.accessibility.AccessibilityNodeInfo** : This represents a unique id for each of the elements which can be tracked in the smartphone screen. Each element loaded can be tracked using this id and is important when we bind our phished app. When we write these elements to the file, there is a possibility of a same element being written twice. This is because we track all elements in two separate accessibility events and write all elements tracked in both events. An app can load in different manner, with some elements loaded after few seconds and thus all elements may not appear in one state change event.

- **boundsInScreen** : This property gives the bounds of a node with respect to screen coordinates. This helps us to determine the location of an element on the screen.

- **packageName** : This property gives the apps package name which is currently used by user. With the help of this package name, we can match if the current app being used by user needs to be monitored or not. For our implementation, we create a file with the package name and this file maintains all GUI elements as tracked by accessibility events for that package name or app.

- **className** : This property gives the class name of the element which denotes the element type i.e. whether the element is a button or text view etc.

- **text** : This property gives the text information associated with the element. If there is no text associated, the value is either blank or null. Generally elements like button, text views etc. have value for this property.

- **contentDescription** : This property gives the content description associated with the control. In order to make the elements accessible, developers add this extra information while binding the elements on GUI allowing any accessibility service to understand the element.

The full trace log for accessibility node info containing all properties about each GUI element is:

```
android.view.accessibility.AccessibilityNodeInfo@80007553;
boundsInParent: Rect(0, 0 - 1440, 2560);
boundsInScreen: Rect(0, 0 - 1440, 2560);
packageName: com.github.andlyticsproject;
className: android.widget.FrameLayout;
text: null; error: null;
maxTextLength: -1;
contentDescription: null;
viewIdResName: null;
checkable: false; checked: false;
focusable: false; focused: false;
selected: false;
clickable: false; longClickable: false;
contextClickable: false;
enabled: true;
password: false;
scrollable: false;
actions: [AccessibilityAction: ACTION_SELECT - null,
AccessibilityAction: ACTION_CLEAR_SELECTION - null,
AccessibilityAction: ACTION_ACCESSIBILITY_FOCUS - null,
AccessibilityAction: ACTION_UNKNOWN - null]
```

4.4 Phishing: GUI Creation

Once we have acquired the GUI elements for an app, we can now create our Phished GUI. Each GUI in android can be created either using XML layouts or programmatically adding each control on the GUI. A sample code for adding an element (like: button) on a screen using XML layout file: Listings 8

Listing 8: Add Button using XML Layout

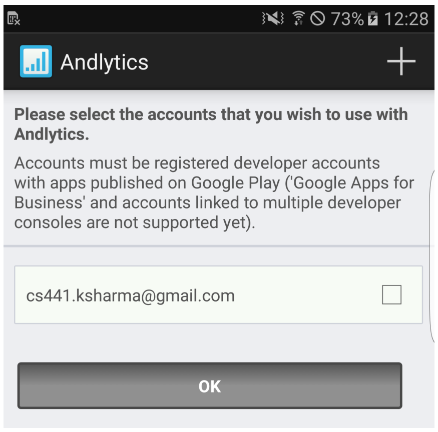
```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="example.com.seaphish.uicarise.writenodesxml.SimpleActivity"
    android:orientation="vertical"
    android:id="@+id/mainLayout">
   <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:paddingTop="10dp"
        android:text="Log On"
        android:id="@+id/login"/>
</LinearLayout>
```

Listing 9: Add Button programmtically using java code

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   /* add a new button to the screen */
    Button myButton = new Button(this);
    RelativeLayout myLayout = new RelativeLayout(this);
   /* add the button to the current screen layout */
    myLayout.addView(myButton);
    setContentView(myLayout);}
```

For our phished GUI generation, we will add each element programmatically in a java code file as logged and written by our accessibility service. Sample code for adding an element (like: button) on a screen programmatically i.e. using java code: Listings 9

A file created by SEAPHISH's accessibility service for an app(when the app is opened by user),


73% 12:28

Andlytics

Please select the accounts that you wish to use with Andlytics.

Accounts must be registered developer accounts with apps published on Google Play ('Google Apps for Business' and accounts linked to multiple developer consoles are not supported yet).

cs441.ksharma@gmail.com

Figure 7: Andlytics App: Original App Screen

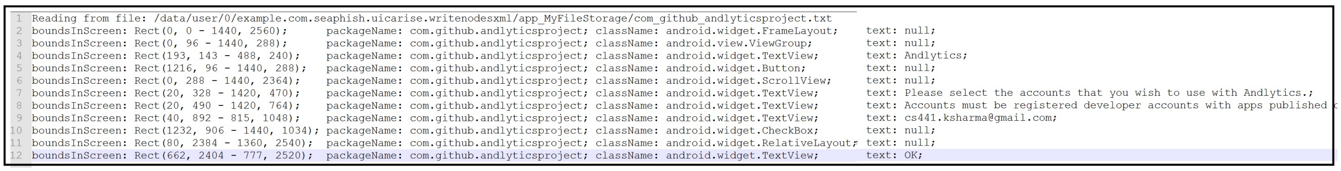

```
Reading from file: /data/user/0/example.com.seaphish.uicarise.writenodesxml/app_MyFileStorage/com_github_andlyticsproject.txt
boundsInScreen: Rect(0, 0 - 1440, 2560);      packageName: com.github.andlyticsproject; className: android.widget.FrameLayout;    text: null;
boundsInScreen: Rect(0, 96 - 1440, 288);      packageName: com.github.andlyticsproject; className: android.view.ViewGroup;        text: null;
boundsInScreen: Rect(193, 143 - 488, 240);    packageName: com.github.andlyticsproject; className: android.widget.TextView;       text: Andlytics;
boundsInScreen: Rect(1216, 96 - 1440, 288);   packageName: com.github.andlyticsproject; className: android.widget.Button;         text: null;
boundsInScreen: Rect(0, 288 - 1440, 2364);    packageName: com.github.andlyticsproject; className: android.widget.ScrollView;     text: null;
boundsInScreen: Rect(20, 328 - 1420, 470);    packageName: com.github.andlyticsproject; className: android.widget.TextView;       text: Please select the accounts that you wish to use with Andlytics.;
boundsInScreen: Rect(20, 490 - 1420, 764);    packageName: com.github.andlyticsproject; className: android.widget.TextView;       text: Accounts must be registered developer accounts with apps published
boundsInScreen: Rect(40, 892 - 815, 1048);    packageName: com.github.andlyticsproject; className: android.widget.TextView;       text: cs441.ksharma@gmail.com;
boundsInScreen: Rect(1232, 906 - 1440, 1034); packageName: com.github.andlyticsproject; className: android.widget.CheckBox;       text: null;
boundsInScreen: Rect(80, 2384 - 1360, 2540);  packageName: com.github.andlyticsproject; className: android.widget.RelativeLayout; text: null;
boundsInScreen: Rect(662, 2404 - 777, 2520);  packageName: com.github.andlyticsproject; className: android.widget.TextView;       text: OK;
```

Figure 8: Andlytics App: File created from AS sample Log

contains the information about each GUI element and its associated properties. In order to process and bind each apps GUI, we configure a TestActivity which will load each apps Phished GUI. Initially this TestActivity will not have any interface and the interface for this activity will be generated programmatically by adding elements to its layout. The GUI elements for this TestActivity will be added from the file which contains the elements properties as saved by SEAPHISH accessibility service. A sample app: Andylytics Figure 7 shows one of the apps downloaded from GitHub.

Sample of traced accessibility log for Andylytics app in Figure 8 for creating phished GUI: (we have shown only three properties which are mainly used to bind an element). If we analyze the GUI, as per our interpretation, we can see the following GUI elements:

(1) A Title on top.

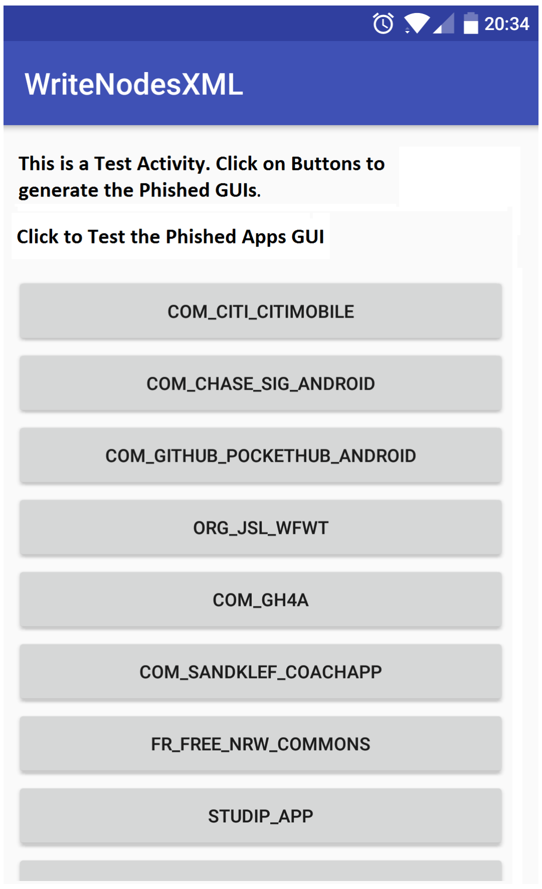
(2) An image of ”+” on the top right corner.

(3) Text areas count=3.

(4) Checkbox count=1.

(5) Button count=1 with OK text.

Now if we analyze the GUI elements information as traced by accessibility service, we can see there are total 12 lines traced, thus we can say, we have 12 GUI elements. Analyzing the classNames : we find there are four text views, which is similar to what we had observed. One textview equivalent


20:34

WriteNodesXML

This is a Test Activity. Click on Buttons to generate the Phished GUIs.

Click to Test the Phished Apps GUI

Figure 9: Test Activity for analyzing Phished GUIs

to title, 3 other textviews reflecting to the content present in body of the app. Two other GUI elements in log traced are: button and checkbox, which is similar to what we see on the app. As a part of any interface on android, each app has some layouts, the other elements in log reflect to those extra layouts.

For creating an interface from the log as shown in Figure: 8, we follow the below steps:

Step 1. Open the file (for example: com_github_andlyticsproject.txt which is the file created for Andlytics app from its package name: com.github.andlyticsproject) for which you want to create the phished GUI. For testing the creation of Phished GUI, we have created a TestActivity which loads the package names of all the apps which are been monitored by accessibility service. Each of these package names are loaded on a button, so on clicking the button it will load the GUI of that app. Figure: 9 shows the sample of TestActivity used for creating Phished GUI's various apps.

Step 2. Now as we click on any button (e.g: com_github_andlyticsproject.txt), a file is loaded from the smartphone's internal memory. This file is the same file as created by SEAPHISH accessibility service for apps. The file contains the properties for each GUI element and we maintain the properties of each element in a new object of our defined class: **GenerateMedata**. In order to understand how an element is bound to an interface and what are the basic minimum properties required, Listing: 10 is a sample of XML file which shows a simple login activity containing: labels, text box (for entering user name and password) and button.

Listing 10: Sample Activity Code in XML

```
"<?xml version="1.0" encoding="utf-8"?>"
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context="example.com.seaphish.uicarise.writenodesxml.SimpleActivity"
    android:baselineAligned="false"
    android:orientation="vertical"
    android:id="@+id/mainLayout">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:paddingBottom="20dp"
        android:text="Hello, Welcome!" />

    <<TextView>
        <android:layout_width="wrap_content">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:text="User Name"/>>

    <<EditText>
        <android:layout_width="match_parent">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:id="@+id/etUserName"/>>

    <<TextView>
        <android:layout_width="wrap_content">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:paddingTop="10dp">
        <android:text="Password"/>>

    <<EditText>
        <android:layout_width="match_parent">
        <android:layout_height="wrap_content">
        <android:paddingBottom="15dp">
        <android:id="@+id/etPassword">
        <android:inputType="textPassword"/>
</LinearLayout>
```

The Listing: 10 XML layout file shows the basic properties: **layout_width**, **layout_height** and **id** required by any element . **android:text** is associated for elements that need to display some text, **android:orientation** defines how the elements are positioned on the screen either horizontally or vertically. For building a phished app which looks visually similar to the original app, we mainly focus on the following properties:

(1) We read each line from the file (i.e. the file which contains the information traced from accessibility service of GUI elements for an app) and split the read line with separator as **semicolon (;)**. Now we have an array of strings which has information in the form of **property : value**. For example: Reading the first line from file: com_github_andlyticsproject.txt, we get:

```
Line : 1 ; android.view.accessibility.AccessibilityNodeInfo@80007553;
boundsInParent: Rect(0, 0 - 1440, 2560);
boundsInScreen: Rect(0, 0 - 1440, 2560);
packageName: com.github.andlyticsproject;
className: android.widget.FrameLayout;
text: null;
error: null;
```

```
maxTextLength: -1;
contentDescription: null;
viewIdResName: null;
checkable: false;
checked: false;
focusable: false;
focused: false;
selected: false;
clickable: false;
longClickable: false;
contextClickable: false;
enabled: true;
password: false;
scrollable: false;
;END
```

and when we split this line with split variable as **”semicolon (;)”** we get all the properties and their value in an array of string as: `packageName: com.github.andlyticsproject`, where property of an element is packageName and the value of packageName is com.github.andlyticsproject.

$\textcircled{2}$ We create an array of objects for GenerateMetadata (self defined class for tracking each GUI elements property) which has each data member representing a property. Each of these data members are initialized by the value as obtained from the file.

Listing: 11 shows the sample code for reading the properties of GUI elements as traced and saved from SEAPHISH’s accessibility service and using these properties to bind the GUI.

Listing 11: Read File and split each line to obtain and set each elements properties for GUI reconstruction

```
try{
   /* read the file for which the phished GUI needs to be generated */
   FileInputStream fis = new FileInputStream(myInternalFile);
    InputStreamReader in = new InputStreamReader(fis);
    BufferedReader br = new BufferedReader(in);
    String strLine;
   /* read each line from the traced accessibility log saved in file*/
   while ((strLine = br.readLine()) != null) {
      System.out.println("Data :" + strLine);
      if (!strLine.equals("")) {
         count++;
         /* maintain an array of objects for storing the properties of each
             element.
        We have defined a class GenerateMedata to track each elements
             properties */
         objects[count] = new GenerateMedata();
         /* fetch each line and split it with ";" to obtain each property of
             element */
         String[] tempArray = strLine.split(";");
         String[] tempSplit;
         if (tempArray != null && tempArray.length > 0){
            for (int i = 0; i < tempArray.length; i++){
               /* match each property and accordingly save it in the object's
                   matched property */
               if (tempArray[i].toLowerCase().contains("android.view.
                   accessibility.accessibilitynodeinfo")){
               tempSplit = tempArray[i].split("@");
               objects[count].setNodeId(tempSplit[1].toString().trim());
}....}}}.....}}
```

③ **className** is an important attribute which helps in classifying which element needs to be bound on interface. The value represents the class of the object and thus we create an instance of this object.

④ Once we know which class the elements belong to, we can set the properties like: the position of each of the elements, is that element enabled, clickable etc. The position that we obtain is relative to the screen and is fetched from **boundsInScreen** property. The **Rect(Left,Right,Top,Bottom)** shows the bounds of a node with respect to the screen. Each of the elements are assigned the position using **setX** and **setY** properties where setX is assigned the value of Left and setY is assigned the value of Top, as obtained from boundsInScreen Rect. The other two values Right and Bottom of boundsInScreen are used to calculate the height and width of an element.

Listing: reflst:bindPhisedGUI is a sample code to show how we position each element on the GUI as per the properties obtained from the SEAPHISH accessibility service.

Listing 12: Bind the elements on the interface as per their class names

```
FrameLayout ll = new FrameLayout(ctx);
/** after reading and assigning each elements properties to objects, iterating
    through all objects and binding the elements on Interface **/
/* In order to bind any element to the interface, we must know its class, so
   here we first check if the class name is "textview" */
if (objects[objCount] != null && objects[objCount].getClassName() != null &&
    (!objects[objCount].getClassName().equals("")) &&
    objects[objCount].getClassName().toLowerCase().contains("textview")){
    /* make an element for textview which will be added to the parent view */
    final TextView tt = new TextView(this);
    if (objects[objCount].getText() != null &&
     (!objects[objCount].getText().equals("")) && (!objects[objCount].getText()
        .equals("null")))
       /* setText is one of the properties of textview which displays the text
          */
       tt.setText(objects[objCount].getText(), TextView.BufferType.SPANNABLE);
    /* this tag is generally assinged value in order to make accessibility
       service understand about the element */
     if (objects[objCount].getContentDescription() != null
        && (!objects[objCount].getContentDescription().equals(""))
        && (!objects[objCount].getContentDescription().equals("null")))
     {
       tt.setContentDescription(objects[objCount].getContentDescription());
       /* for security reasons, sometimes the text is wrapped and may come as
          null, thus we make use of contentDescription to bind the text
        to the textview */
          if(tt.getText() == null ||  tt.getText().equals("") || tt.getText().
             equals("null"))
          {
           tt.setText(objects[objCount].getContentDescription(), TextView.
              BufferType.SPANNABLE);
        }
      }
    /* generate the id for the element. By default the value is -1, which may
        cause malware to think of this as some suspicious or fake screen  */
     String tempID = objects[objCount].getViewIdResName();
    tt.setId(getID(tempID));

    /* Get the cordinates of the element with respect to the screen and assign
         setX and setY property to position the element */
     String[] tempRect = objects[objCount].getBoundsInScreen().split("\\(");
     String[] tempRect2 = tempRect[1].split(",");
     String[] tempRect3 = tempRect2[1].split("-");

     tt.setX(Integer.parseInt(tempRect2[0].trim()));

    /* since we have removed the action tool bar from top and we try to create
        it on our own, thus we need to subtract some offset value,
        this helps us to predict which elements may be present in the header (
           aciton bar) of an app */
    if(Integer.parseInt(tempRect3[0].trim()) > 250){
        tt.setY((Integer.parseInt(tempRect3[0].trim()) - offset_y));
        ll.addView(tt);
     }
     else{
        /* now we bind these elements separtely as a part of custom tool bar */
        System.out.println("Part of custom action bar");
        customCount++;
          customObjects[customCount] = new GenerateMedata();
          customObjects[customCount] = objects[objCount];
          tt.setY((Integer.parseInt(tempRect3[0].trim())));    }}...
```

⑤ The elements for which we obtain only the base class i.e. if the **className** property has the value as `android.view.View`, we need extra information to bind such elements. Thus for understanding the actual class for these elements, we check the `contentDescription` and `isClickable` which gives us an idea about the element type. For example, an element having attributes value for contentDescription as checkbox and isClickable : true, gives the clarification about the class of the element as android.view.CheckBox and thus we can bind a checkbox to the interface.

⑥ Other properties like `isEnabled, isFocusable, isClickable` are set to all the elements. For example, isEnabled plays an important role for elements like buttons. We try to maintain all the properties and features of the elements as present in original app and then build the phished app.

⑦ For elements present in the topmost section of an app i.e. the elements that basically forms the header part of app are added separately. In the initial and base configuration of our phished app, we specify **NoActionBar**, thus the header of phished app is also painted programmatically. For all the elements which have the boundsInScreen(Left,Right,Top,Bottom) - Top value ranging between 0 to 250 are considered to be present as a part of top action bar. We add these elements as a part of custom elements and later bind them as a part of ToolBar.

4.5 GUI Analysis

In order to understand the binding of GUI on interface, we used 15 apps downloaded from Github and two android apps from Google Play Store: *CitiBank* and *ChaseBank* to see the variations of how can we re-create the GUI in a generic way allowing us to write single common functions which can reproduce GUI from any android app. We extracted the GitHub projects using OpenHub [1] a community and directory providing users with search tools and services. Ohloh, a free RESTful Application Programming Interface to OpenHub allows to search projects from the large repository of projects listing added on OpenHub. We used some filters in the query by focusing the search just on android apps and projects which were hosted on GitHub. Below is the search query used to browse the through android apps hosted on GitHub:

```
www.openhub.net//projects.xml?query=android+app&page=1&api_key=
c3943bda503b24b9ed76ba00add525ecd330720aa437f82fa3bc4cbeab330b7b
```

Table I: GITHUB DOWNLOADED APPLICATIONS OVERVIEW

App Name	Package Name	Apps Overview
App Name	Package Name	Apps Overview
PocketHub	com.github.pockethub.android	This app is a small version to connect to GitHub repository and sync repositories
WiFiTalkie	org.jsl.wfwt	App transmits sound recorded from microphone to some other devices running the same program on the same network segment
Coach Assistant	com.sandklef.coachapp	An app to assist instructors with coaching
Commons	fr.free.nrw.commons	This app allows users to upload pictures from their Android phone/tablet to Wikimedia Commons
StudIPApp	studip.app	A basic app that provides facilities and authorization to registered users to connect to their server and get information about courses
Jupiter Broadcasting	jupiter.broadcasting.live.tv	An app that allows users to stream and download videos from `www.jupiterbroadcasting.com`
Andlytics	com.github.andlyticsproject	The app collects statistics from the Google Play Developer Console
BeeExample	com.BeeFramework.example	This app allows users to create their own mobile apps using Objective-C and XML or CSS
Wikivoyage Offline	org.github.OxygenGuide	This is an offline app for android and data comes from OxygenGuide.

Table I Continued: GITHUB DOWNLOADED APPLICATIONS OVERVIEW

App Name	Package Name	Apps Overview
Meta Filter	mpt.metafilter	This app displays the latest metafilter RSS feeds.
SmartGadget	com.sensirion.smartgadget	app allows you to establish a BLE (Bluetooth Low Energy) connection to your Sensirion Smart Gadget
NFC on Android Example	com.fernandocejas.example.android.nfc	This app allows user to write data to NFC tags with custom mime-types
Try Haskell	nl.bneijt.tryhaskell	This app supports Haskell language few commands.
WTF (The MirOS Project)	de.naturalnet.mirwtfapp	Android front end to resolve acronyms on the command line as included in The MirOS Project
SMS Spoofer	net.thomascannon.smsspoofer	This app is a proof of concept which shows how SmsReceiverService permission of android is exploited to fake an incoming message.

The apps contained activities having GUI with elements like buttons, text boxes, edit text boxes, images etc. For elements like grid views or list views, the datasource is generally bound to these elements in code behind files and thus this information for such elements cannot be obtained from accessibility service. For elements like images, the source is generally present in the drawable folder (present under res) or sometimes the image may have an online web page source in which the image may not be the same every time. For all such elements - grid views, list views, images, etc. we just add a stub for these elements with the data source as blank. This is done in order to preserve the UI hierarchy and make the UI appear almost same as the original. The creation of the first phished GUI was done using the accessibility log obtained from Chase Bank app. This app has all the basic elements that are present in most of the apps for example: textviews, edittextviews, buttons, checkbox, image. While binding the elements for phished GUI, we make sure the elements have all the properties as present in original app. Let us analyze one line from the file of Chase Bank app which contains the elements as presented below:

```
Line : 15 ;
android.view.accessibility.AccessibilityNodeInfo@80009b5e;
boundsInParent: Rect(0, 0 - 1080, 164);
boundsInScreen: Rect(180, 1413 - 1260, 1577);
packageName: com.chase.sig.android;
className: android.widget.EditText;
text: null;
error: null;
maxTextLength: -1;
contentDescription: Password;
viewIdResName: com.chase.sig.android:id;
checkable: false;
checked: false;
focusable: true;
focused: false;
selected: false;
clickable: true;
longClickable: true;
contextClickable: false;
enabled: true;
password: true;
scrollable: false; ;END
```

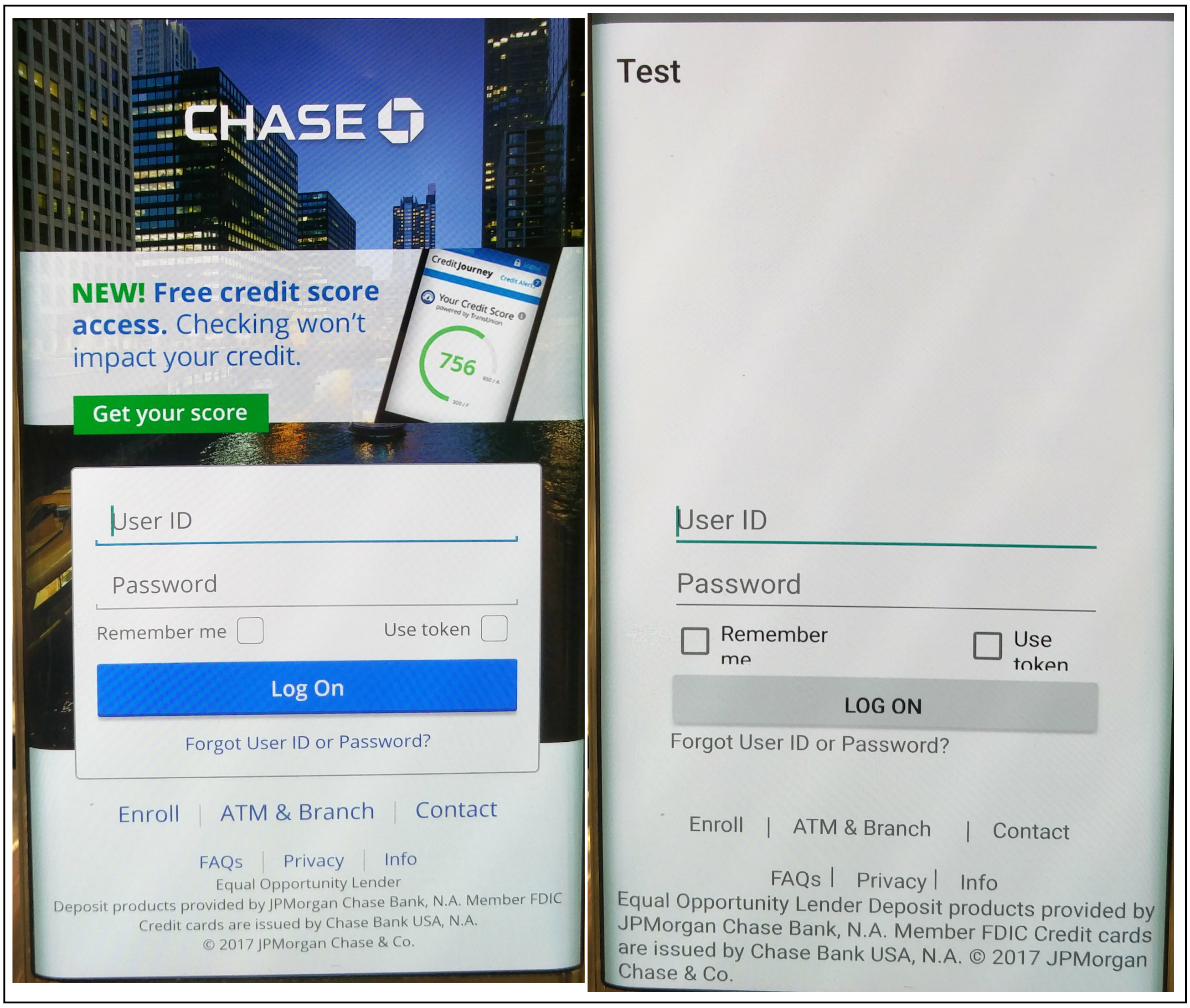
Considering only the important properties, we first check the *className* - in this case it *EditText* i.e. an input text box. The *contentDescription* - explains the element will contain password, additionally, we have another property: *password* which has the value as *true* which means that the element must take an input type as password. Listing: 13 is the sample of configuration for an edit text box which takes input as password type.

Listing 13: Edit Text Sample Configuration with property as Password

```java
EditText et = new EditText(this);

if (objects[objCount].getText() != null && (!objects[objCount].getText().
    equals("")) && (!objects[objCount].getText().equals("null")))
    et.setHint(objects[objCount].getText());
else if (objects[objCount].getContentDescription() != null && (!objects[
    objCount].getContentDescription().equals("")) && (!objects[objCount].
    getContentDescription().equals("null")))
{
    et.setHint(objects[objCount].getContentDescription());
    et.setContentDescription(objects[objCount].getContentDescription());
}
if(objects[objCount].getPassword())
    et.setInputType(InputType.TYPE_CLASS_TEXT | InputType.
        TYPE_TEXT_VARIATION_PASSWORD);
```

A sample of original app vs phished app for ChaseBank can be seen in Figure: 10 (NextPage).


CHASE
NEW! Free credit score access. Checking won't impact your credit.
Get your score
Credit Journey
Your Credit Score
756
User ID
Password
Remember me
Use token
Log On
Forgot User ID or Password?
Enroll | ATM & Branch | Contact
FAQs | Privacy | Info
Test
User ID
Password
Remember me
Use token
LOG ON
Forgot User ID or Password?
Enroll | ATM & Branch | Contact
FAQs | Privacy | Info

Equal Opportunity Lender
Deposit products provided by JPMorgan Chase Bank, N.A. Member FDIC
Credit cards are issued by Chase Bank USA, N.A.
© 2017 JPMorgan Chase & Co.

Equal Opportunity Lender Deposit products provided by JPMorgan Chase Bank, N.A. Member FDIC Credit cards are issued by Chase Bank USA, N.A. © 2017 JPMorgan Chase & Co.

Figure 10: Chase Bank: Comparing Original Vs Phished App

Once we obtain a good structure of the GUI for the Chase Bank app, we proceeded and tested the same code on Citi Bank app. Every app may be developed in different ways. For Chase Bank app, the accessibility log helped us easily to understand each and every element as the log specified the actual class name of the element. For CitBank app we observed that for each element, only the base class name (android.view.View) was logged through accessibility service. Thus this introduced another set of conditions to be implemented and handled when we do not have the clarity about the actual class name of element. Consider the lines in box: **??** (Line: 50), which shows only one line of log from Citi Bank app file,the information traced from accessibility service.

```
Line : 50 ;
android.view.accessibility.AccessibilityNodeInfo@90ec;
boundsInParent: Rect(10, 10 - 37, 37);
boundsInScreen: Rect(120, 1040 - 228, 1148);
packageName: com.citi.citimobile;
className: android.view.View;
text: null;
error: null;
maxTextLength: -1;
contentDescription: Remember user id checkbox;
viewIdResName: rememberUserIDcheckBox;
checkable: false;
checked: false;
focusable: false;
focused: false;
selected: false;
clickable: true;
longClickable: false;
contextClickable: false;
enabled: true;
password: false;
scrollable: false;
;END
```

Analyzing the log, we see the *className* value as *android.view.View* which does not gives any information which element do we need to bind. So now we analyze *contentDescription* and *clickable* properties. We get the clickable property as *true* and contentDescription as *Remember user id checkbox* which gives us the information about the element - this is a checkbox with text for the checkbox same as content description. Thus we bind a checkbox for this as presented in Listing: 14.

Listing 14: CheckBox element identified from android.view.View class

```
/* check if the class name contains android.view.View */
else if (objects[objCount] != null && objects[objCount].getClassName() != null
    &&
      (!objects[objCount].getClassName().equals("")) &&
      objects[objCount].getClassName().equals("android.view.View")){
   /* first check if the element is clickable */
   if (objects[objCount].getClickable()){
      /* now checking the content description to understand what type of
          element do we need to bind */
      if (objects[objCount].getContentDescription() != null && (!objects[
          objCount].getContentDescription().equals("")) && (!objects[objCount
          ].getContentDescription().equals("null"))){
         if (objects[objCount].getContentDescription().toLowerCase().contains(
             "checkbox"))
         {
         objCount++;
         /* create the new element as CheckBox */
              CheckBox chkBox = new CheckBox(this);
              if (objects[objCount].getText() != null && (!objects[objCount
                  ].getText().equals("")) && (!objects[objCount].getText().
                  equals("null")))
            chkBox.setText(objects[objCount].getText());
              else
              {
            chkBox.setContentDescription(objects[objCount].
                getContentDescription());
                chkBox.setText(objects[objCount].getContentDescription());
              }.....
         /* setting other properties as obtained from accessibility */
         chkBox.setEnabled(objects[objCount].getEnabled());
              chkBox.setFocusable(objects[objCount].getFocusable());
         chkBox.setClickable(objects[objCount].getClickable());
         chkBox.setChecked(objects[objCount].getChecked());....}}}}
```

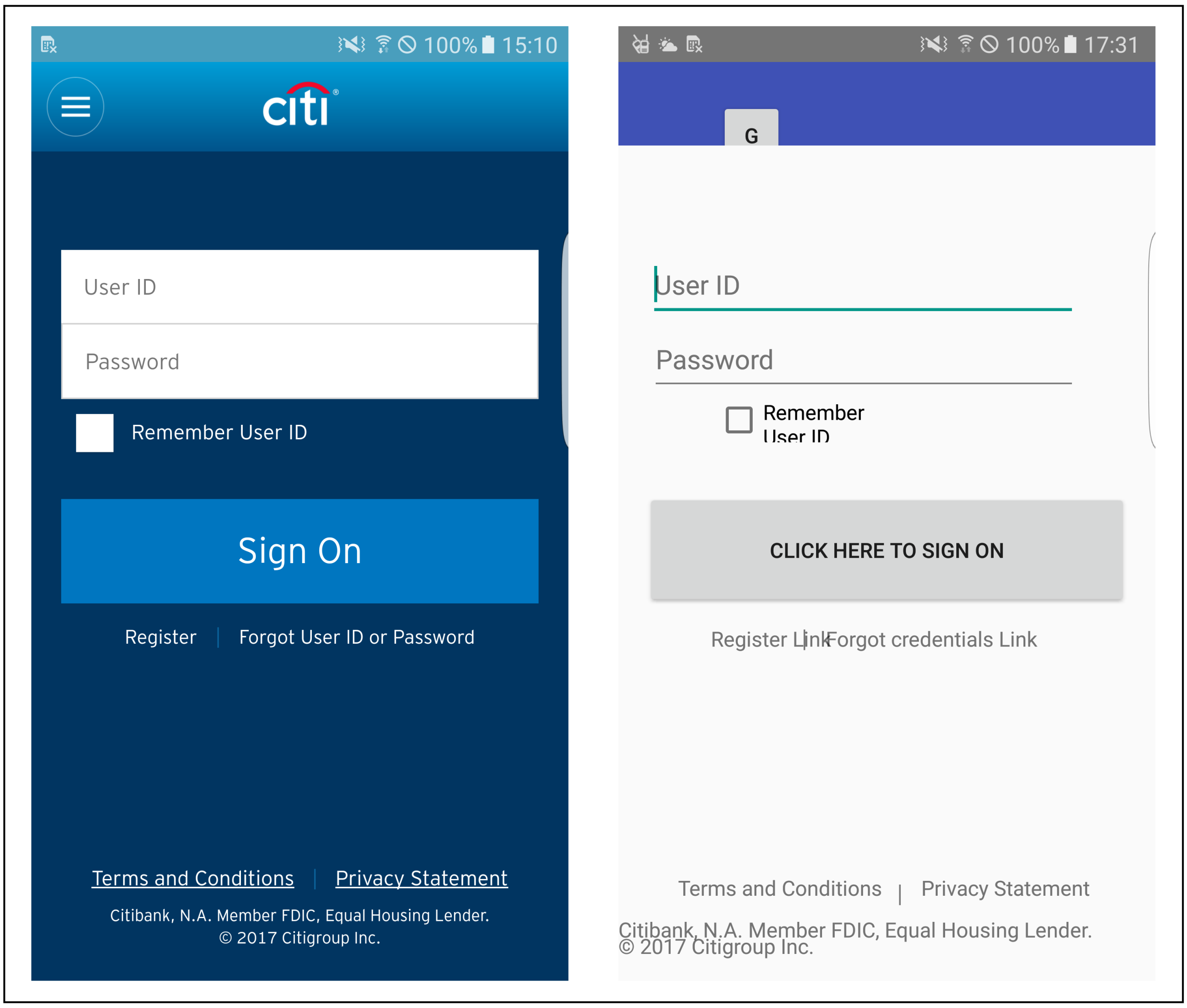


Figure 11: CitiBank: Comparing Original Vs Phished App

A sample of original app vs phished app for CitiBank can be seen as in Figure: 11. We can see the difference in the text and the difference is there because we build the phished GUI using the **contentDescription** as for the property of **text** is either null or blank for all the elements.

5. Experimental Results

SEAPHISH framework extends android's accessibility service to develop its own accessibility service which extracts the GUI elements and its properties for an android app and re-creates Phished GUI using the extracted GUI elements and their properties. Our framework aims to detect and defend malicious apps which may be stealing sensitive information from user's smartphone. These malicious apps may use various permissions granted by users or use other mechanisms to steal information, send it over network, or with apps having accessibility service enabled can perform different unauthorized operations. For making our framework function successfully, we need to analyze the phished app being generated. For our analysis, we downloaded a set of 15 apps from GitHub repository using BlackDuck OpenHub (as explained in Solutions) and obtained the phished GUI for these apps. Following were the first set of analysis performed:

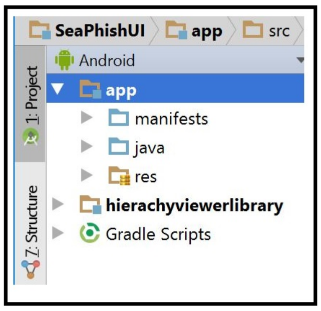


Figure 12: Android Project Structure Part - 1

(1) Analyze the resources of an app: Resources for an android app consists of images, styles definitions, text (strings) bound on GUI etc. All these information are maintained in different XML files. As we analyze the phished GUI, certain elements which need resources cannot be obtained from accessibility service. These resources are present in *res* folder of the project. Figure: 12 is the snapshot of how a folder structure for any android app looks like in Android SDK. This is the Android View (in Android SDK - IDE used for developing android apps), where we can see: **app** as the root (our main application) and on expanding - as in Figure: 13 we have sub-folders: manifest, java and res. This is the initial project structure for SEAPHISH framework. A brief overview about each folder:

- **manifests** - This folder contains *AndroidManifest.xml* which is the main file for any android application defining about each of the activities, services, permissions which will be used by the app.

- **java** - This folder contains our code files including the code behind file for all activity and services.

- **xml** - This folder contains the extra metadata information for any services bound to an app. In our design, we add the metadata information for SEAPHISH's accessibility service. Listing: 15 is sample file which contains the metadata information for SEAPHISH's accessibility service.

Listing 15: Mets-data for SEAPHISH accessibility service

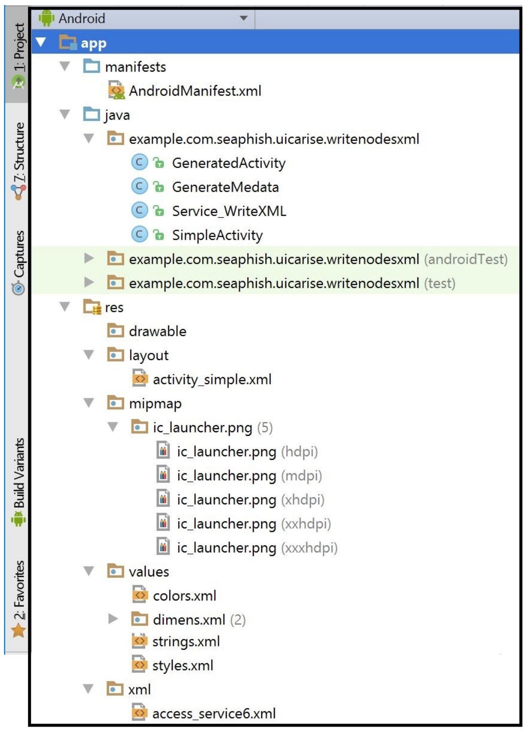
```
<?xml version="1.0" encoding="utf-8"?>

<accessibility-service xmlns:android="http://schemas.android.com/apk/res/android"
    android:description="@string/service_6"
   android:accessibilityFlags="flagRetrieveInteractiveWindows|flagRequestFilterKeyEvents|flagReportViewIds|
                              flagRequestEnhancedWebAccessibility|flagRequestTouchExplorationMode"
    android:accessibilityFeedbackType="feedbackSpoken|feedbackHaptic|feedbackAudible|feedbackVisual|feedbackGeneric|feedbackAllMask"
    android:canRetrieveWindowContent="true"
    android:settingsActivity="com.example.android.accessibility.ServiceSettingsActivity"
/>
```

- **res** - This folder serves as a repository for all resources that are used in an android app. It contains four sub-folders with each folder containing some resources which are used across the entire app in single or multiple activities. Each of the four sub-folders contain xml files maintaining resource in the form of key-value pair.

  ⋆ **drawable** : contains images which are used across all the activities in app. In order to access any image from this folder, user can write **@drawable/image_1** in their XML file for binding an image to an element. *image_1* represents a key in XML file and it will have a value corresponding to an image. Similarly to bind the image in code behind file, user can specify **R.drawable.image_1** where **R** represents the resources and each of the folders can be accessed using R.

  ⋆ **layout** : contains the xml layout files defining the GUI for each activity. A user can add elements to their GUI directly by adding elements in the XML file or they can add elements in the code behind file (present in java folder). XML layouts are easy to configure and also allows user to preview their GUI.



```
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ example.com.seaphish.uicarise.writenodesxml
        GeneratedActivity
        GenerateMedata
        Service_WriteXML
        SimpleActivity
    ▶ example.com.seaphish.uicarise.writenodesxml (androidTest)
    ▶ example.com.seaphish.uicarise.writenodesxml (test)
  ▼ res
      drawable
    ▼ layout
        activity_simple.xml
    ▼ mipmap
      ▼ ic_launcher.png (5)
          ic_launcher.png (hdpi)
          ic_launcher.png (mdpi)
          ic_launcher.png (xhdpi)
          ic_launcher.png (xxhdpi)
          ic_launcher.png (xxxhdpi)
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
        styles.xml
    ▼ xml
        access_service6.xml
```

Figure 13: Android Full Project Structure

- ⋆ **minmap** : contains the icons that may be used in general by apps. By default this folder has an android launcher icon in different sizes. A user can bind any icon present in this folder using code: **@mipmap/ic_launcher**, where *ic_launcher* is one of the icons present under this folder.

- ⋆ **values** : contains four xml files each file acting as a common resource for all the activities. The four files are: *values.xml* - contains key value pair for any strings used in activities. So instead of developers providing a text to a button, the same text is added in this xml and now instead of using the text, we use its key to bind the value, *colors.xml*, *dimens.xml* and *styles.xml* contains the key-value pair for defining styles, colors and dimensions (like padding, width, height etc.) which is common across the app.

While creating the phished app, we need certain resources, like images present on original app. For binding these images in our phished app, we require a prior knowledge about the images being used and which set of images are bound to which elements on the GUI. One way of gaining access to all resources is by using **APKTOOL** [22] which is a reverse engineering tool for 3rd party, closed, binary Android apps. Apktool [14] can extract all XML files including the layout files, resources files from the *apk* file as its input. **Apk** [10] is Android application package file format which contains the android code files, resources and data compiled and packaged in one single file. Apk is used in distribution of android apps. Unzipping apk file will give us all compiled java code with files such as *file1.dex* and *resources.arsc* which is not readable. We used Apktool to reverse engineer an android app and obtain all files in readable format.

Using Apktool: **apktool d testapp.apk** is the command for running Apktool on a testapp.apk (any android app apk file) where **d** represents **decode**. The output from Apktool is a folder that contains all the code files, resources and XML files. This reverse engineering can be used in order to either extend any project or for analysis purpose. But even after we receive the original resources, we have few other problems:

① Using Apktool for decoding just for minor resources is too much of overhead for framework and it looses its dynamic behavior.

② Even after we receive the resources (like :images, all XML resource files), it is still difficult to match the elements with their respective resources. For example: from the accessibility service we understand what type of GUI element we are binding, so if the element is an Image, we require the source of this GUI element i.e. we require an image present in one of the resource folder. Analyzing the properties of each GUI element obtained via accessibility service, we still do not get any information about the image resource bound to a GUI element. Thus we may have 3 GUI elements with 3 different images and from our reverse engineered apk, we have a resource folder which may contain any number of images. So we have insufficient information about the resource that needs to be matched with a particular GUI element. The insufficient information applies for all other resources that we may require for any GUI element.

We also tried using **SilkuliX** [19] which automates anything we see on our screen (Windows, Mac etc.). This is used when the user does not have an easy access to the source code or GUI internals of the application or web page that the user wants to analyze. It helps in locating images on a screen, can be used to search text in images etc. For using it on a android device, Sikuli can be used with an **emulator** or with **Android Debugging Bridge (ADB)**. ADB [11] is a command line tool that allows to install or debug apps either on emulator or a connected android device. It is a client-server program which allows various device actions (installing/debugging) and has three components:

① **client** - runs on the development machine and can be invoked using *adb* command from a command-line terminal.

2 **daemon (adbd)** - a background process on each device and runs command on the device.

3 **server** - for managing communication between the client and daemon and runs as a background process on the development machine.

Sikuli can be used in integration with ADB to obtain images for a running app (i.e. the app is currently used by user) either on emulator or on a connected android device. But this process needs to have the source code of the app and again requires pre-processing of the entire source code. This may help us to obtain the images for an app, but we still have to analyze the co-ordinates of the obtained image and then process these co-ordinates to match with the respective GUI element. Even after achieving the images, there are possibilities that the individual apps may update - either change the image or remove the GUI element with image. For Chase Bank app, we observed the background image used was different most of the time, the image may differ or has some extra texts. Even after we resolve the issue of image, we still do not obtain other resources including the styles, background color etc. These problems gives us an understanding that just for an image, this pre-processing of code is a big overhead. We want the framework to be very simple, it should run dynamically not depending on the source code at all. In simple, we want the framework to be very generic and not depending highly on apps source code.

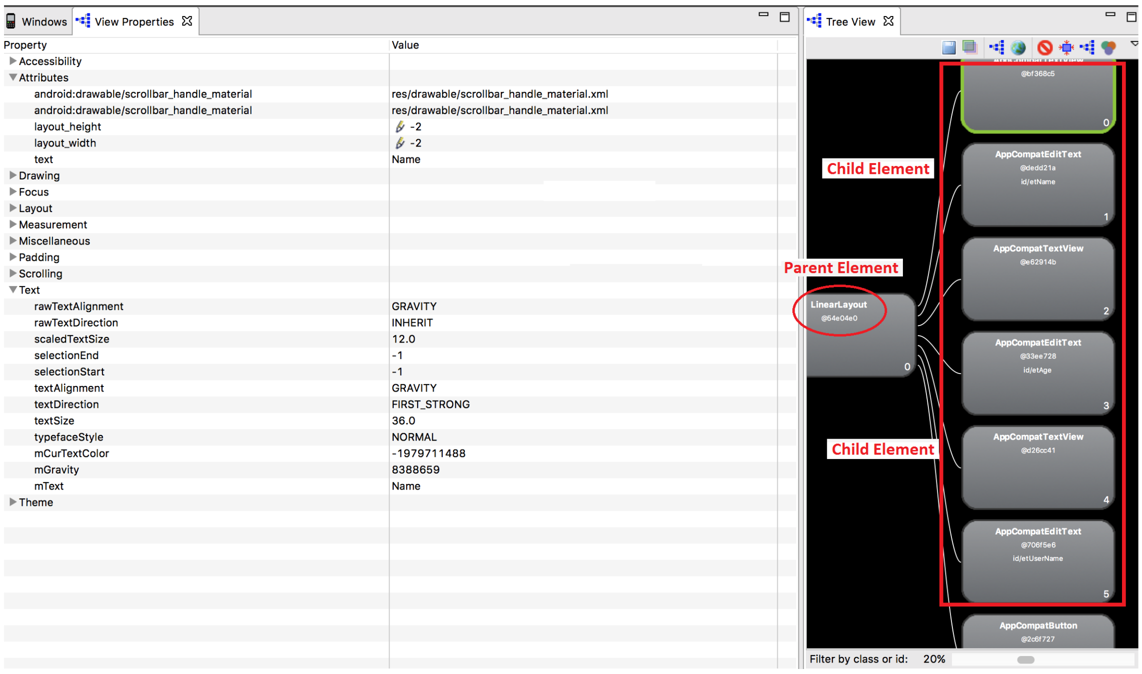
On creating the phished GUI, although we are not able to show the actual image, but we bind a stub for each image, which has its resource as blank. The image resource may be empty, but still it will be present in the UI hierarchy and on analyzing this GUI's elements hierarchy we find each element as present in the original app.

Further we tried using the in-built tool of Android Studio: **Android Device Monitor (ADM)**. This is generally used to enhance and improve the GUI structure as it gives a tree-structure type view to analyze the GUI of an app. The pre-requisites in order to analyze the UI hierarchy [20] for an app using ADM are:

1 For running the app on a hardware instead on an emulator, connect the device using USB to the computer and check if the computer is able to detect the device.

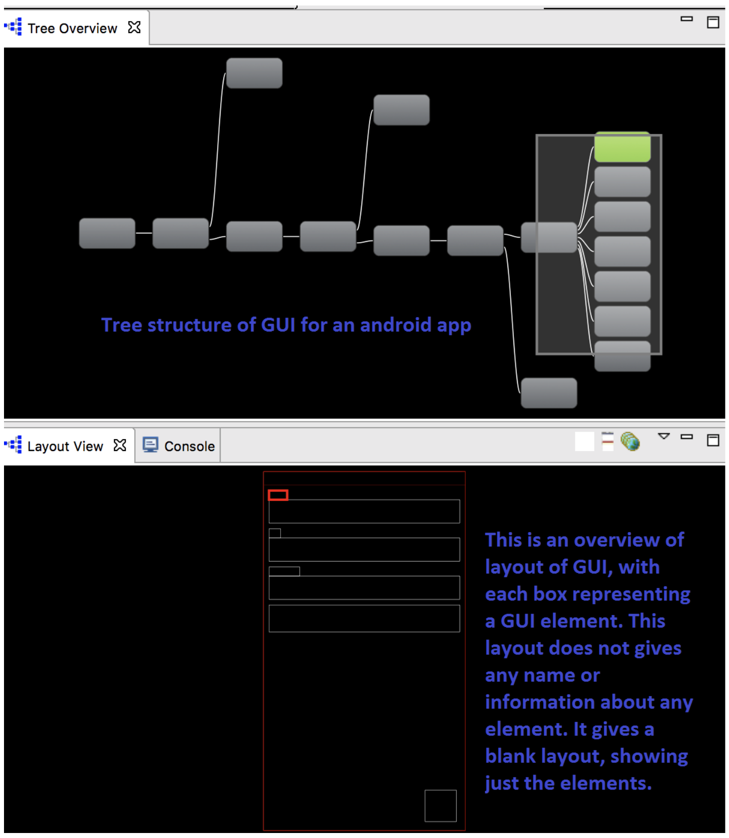
2 Enable Android Debugging Bridge (ADB). In **Android Studio** -> **Tools** -> **Android** -> if there is a check on ADB, means its enabled, if there is no check, click on **Enable ADB integration** to enable.

3 If there is any instance of Android Device Monitor, it must be shutdown.



Property	Value
▶ Accessibility	
▼ Attributes	
android:drawable/scrollbar_handle_material	res/drawable/scrollbar_handle_material.xml
android:drawable/scrollbar_handle_material	res/drawable/scrollbar_handle_material.xml
layout_height	-2
layout_width	-2
text	Name
▶ Drawing	
▶ Focus	
▶ Layout	
▶ Measurement	
▶ Miscellaneous	
▶ Padding	
▶ Scrolling	
▼ Text	
rawTextAlignment	GRAVITY
rawTextDirection	INHERIT
scaledTextSize	12.0
selectionEnd	-1
selectionStart	-1
textAlignment	GRAVITY
textDirection	FIRST_STRONG
textSize	36.0
typefaceStyle	NORMAL
mCurTextColor	-1979711488
mGravity	8388659
mText	Name
▶ Theme	

Figure 14: Hierarchy Viewer GUI analysis



This is an overview of layout of GUI, with each box representing a GUI element. This layout does not gives any name or information about any element. It gives a blank layout, showing just the elements.

Figure 15: Hierarchy Viewer Layout analysis

Figure: 14 is a snaphshot of the HierarchyViewer tree-structure obtained for a sample app. In Firgure: 14, the right side of the figure shows a snapshot of some GUI elements present in the app and the elements (like AppCompatTextView, AppCompatEditText etc.)- marked in Red Box are the children element for **LinearLayout** (Parent Element for these elements). With this structure obtained, it was still difficult to compare as when we tried to dump the entire hierarchy log and save it to a file, we still have to manually format the hierarchy of nodes. Listing: 16 is a small sample of log obtained from hierarchy viewer:

Listing 16: Mets-data for SEAPHISH accessibility service

```
Start display list (0x7f7522c400, ActionBarOverlayLayout)
  Save 3
  ClipRect 0, 0, 1440, 2560
  Draw RenderNode 0x7f7522c800 FrameLayout
  Start display list (0x7f7522c800, FrameLayout)
    Save 3
    Translate (left, top) 0, 320
    ClipRect 0, 0, 1440, 2240
    Draw RenderNode 0x7f750b5800 LinearLayout
    Start display list (0x7f750b5800, LinearLayout)
      Save 3
      ClipRect 0, 0, 1440, 2240
      Draw RenderNode 0x7f75208000 TextView
      Start display list (0x7f75208000, TextView)
        Save 3
        ClipRect 0, 0, 1440, 76
        Save flags 3
        ClipRect  0.00  0.00 1440.00 76.00
        Draw Text of count 52, bytes 104
        Restore to count 1
      Done (0x7f75208000, TextView)
     ......
.....
```

Since we obtain the log in a raw format, forming this to obtain a proper structure to compare the hierarchy for two GUIs is an extra overhead. It would be much easier if we can obtain a log of the GUI elements written in a proper hierarchy. For analysis and evaluation of phished GUI, we followed the analysis as:

① In order to compare the original GUI and phished GUI, we downloaded 15 working apps from GitHub.

② With these 15 android apps, we generated the phished GUI.

③ We built a common library : **hierarchyviewerlibrary** that will be used to fetch the GUI elements and its entire hierarchy. The library contains the code which obtains the root node of the GUI after the app is loaded on the screen and with this root node, it will iterate to fetch each of its children. Listing: 17 is the main function that is used to build the hierarchy:

Listing 17: Hierarchy Viewer class function to obtain GUI Elements

```java
private static void debugChildViewIds1(View view, String logtag, int spaces) {
  /* check if the element as a part of viewGroup — the main base class for any element in android */
  if (view instanceof ViewGroup) {
    /* get the group of the element */
    ViewGroup group = (ViewGroup)view;
    countElements+=group.getChildCount();
    /* iterate through the child of the current element — if the count > 0 */
    for (int i = 0; i < group.getChildCount(); i++) {
    /* get the child element which is part of the view */
    View child = group.getChildAt(i);
    /* Log the element with proper tags — GUIObject is a tag given to each element */
    Log.v(logtag, padString("<GUIObject><Name>" + child.getClass().getSimpleName() +"</Name><Id>"+ child.getId() +"</Id><Location x=\""+child.getX()+"\" y=\""+child.getY()+"\" bottom=\""+child.getBottom()+"\" right=\""+child.getRight()+"\" />", spaces));
    /* if the current element has any more child, first fetch those.*/
    debugChildViewIds1(child, logtag, spaces + 1);
    /* once the element has no more children, we can close the GUIObject tag for the element */
    Log.v(logtag,padString("</GUIObject>",spaces)); }}}
```

The code as shown in Listing: 17 generates a log of GUI elements, with each parent element wrapping its children element within its tag. Each element is wrapped in GUIObject tag and the name for these objects denote the className of each element. This className is basically the classifier about each element present on the screen. We tried evaluating other elements, like position, width, height etc. but due to the varying size of screen and with difference in the pixels density of each smartphone, evne if we obtain the positions, width, height etc. this might still change when rendered on different smartphones. Thus we focus mainly on the className that is assigned to each GUIObject. Listing: 18 is the sample of log generated from above function:

Listing 18: Output of log generated from Hierarchy Viewer class function

```
<State Alias=”al” ScreenName=”Andlytics”>
 <GUIObject><Name>LinearLayout</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
  <GUIObject><Name>ViewStub</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
  </GUIObject>
  <GUIObject><Name>FrameLayout</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
   <GUIObject><Name>ActionBarOverlayLayout</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
    <GUIObject><Name>ContentFrameLayout</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
     <GUIObject><Name>FrameLayout</Name><Location x=”0.0” y=”0.0” bottom=”0” right=”0” />
      <GUIObject><Name>TextView</Name><Location x=”20.0” y=”78.0” bottom=”0” right=”0” />
      </GUIObject>
      <GUIObject><Name>TextView</Name><Location x=”20.0” y=”240.0” bottom=”0” right=”0” />
      </GUIObject>
      <GUIObject><Name>TextView</Name><Location x=”40.0” y=”642.0” bottom=”0” right=”0” />
      </GUIObject>
      <GUIObject><Name>CheckBox</Name><Location x=”1232.0” y=”656.0” bottom=”0” right=”0” />
      </GUIObject>
      <GUIObject><Name>TextView</Name><Location x="662.0" y="2154.0" bottom="0
        " right="0" />
      </GUIObject>
     </GUIObject>
    </GUIObject>
    <GUIObject><Name>ActionBarContainer</Name><Location x="0.0" y="0.0" bottom
       ="0" right="0" />
     <GUIObject><Name>Toolbar</Name><Location x="0.0" y="0.0" bottom="0" right
        ="0" />
      <GUIObject><Name>AppCompatTextView</Name><Location x="0.0" y="0.0"
         bottom="0" right="0" />
      </GUIObject>
     </GUIObject>
     <GUIObject><Name>ActionBarContextView</Name><Location x="0.0" y="0.0"
        bottom="0" right="0" />
     </GUIObject>.....
</State>
```

In Listing: 18, we get all the elements represented as a GUIObject and we try to build it in a tree structure with each GUI element having its start tag as <GUIObject> with properties : Name and Location. Although location property did not give us proper information about the element as it changes with respect to API level being used and gives different information with different screen sizes. The **Name** property is the main classifier. Each GUIObject has its starting and ending tags. If there are any GUIObjects present between the start and end tag of another GUIObject, it means the GUIObjects (or GUI elements) are actually the child of the enclosing GUIObject and thus each child in our log is shifted by one space to right which gives us a clear indication about the parent and child elements and overall gives us a clear picture about the UI hierarchy of the app.

At first we tried importing the library directly into the project and use the functions defined in this library. But since the downloaded projects had varying versions of Android, thus importing this library, changing its configuration in each project for making it compatible was time consuming. So we used the functions written in this library directly in the main activity of each app. The main activity of each app is defined in AndoridManifest.xml with **intent-filter** tag having action as **MAIN** and category as **LAUNCHER**. Listing: 19 is the sample code showing the configuration of Main Activity present in

AndoridManifest.xml under <application>

Listing 19: Main Activity Definition in AndroidManifest.xml

```
<activity android:name=".SimpleActivity">
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

After getting the knowledge about the main activity of each app, we search for **onCreate** method in the main activity. This function is overridden in each activity and once the elements are bound to the interface, we can take the root node or element of the GUI and pass this element as the parameter to the hierarchy viewer function. Obtaining the current screen's root element can be done using: **getWindow().getDecorView()**. Listing: 20 is a sample code of onCreate method with the function call to hierarchy viewer:

Listing 20: onCreate method calling hierarchy viewer function

```
public void onCreate( Bundle savedInstanceState )
{
  super.onCreate( savedInstanceState );
    Log.d( LOG_TAG, "onCreate" );
    getWindow().addFlags( WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON );
  /* setContentView shows the binding of ui elements is done */
    setContentView( R.layout.main );
  /* we fetch the current window and then get the root node to pass it to
      another function
     which iterates through all the parent and child nodes. */
  debugViewIds(this.getWindow().getDecorView(), "andylytics");
    System.out.println("Total Count: "+countElements);
}
```

After obtaining the logs for each of original and phished app, we used the Tree-Edit algorithm with original app xml and phished app xml containing nodes in a tree structure with main focus on comparing the the name property associated with each GUIObject. The tree-edit algorithm can take input either two trees or two xmls.

5.1 Tree Edit Distance Algorithm Overview

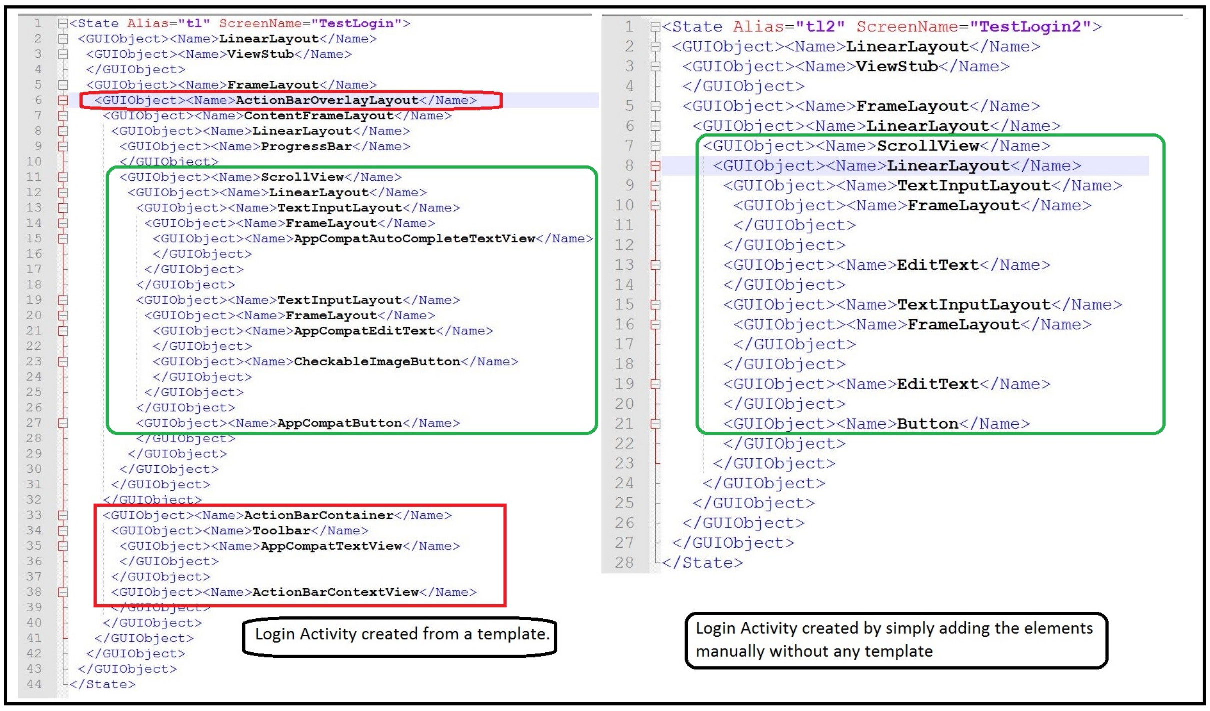
Edit distance [65] is a popular approach for quantifying how dissimilar two strings are to one another by counting the minimum number of operations required to transform one string into the other [84]. Algorithms based on edit distances are used in different areas, e.g., for automatic spelling correction in natural language processing and in bioinformatics to quantify the similarity of DNA molecules. Edit distance algorithms are based on relabeling, inserting and deletion of nodes, the same key operations that stakeholders use to modify GUIs. We use tree edit distance algorithms to difference GUIs by computing the minimum cost solution where the solution comprises of operations like deleting and relabeling existing GUI objects, inserting new GUI objects which transforms one tree into the another.

When we try to compare the two GUI XMLs for our original and phished app, we get the results indicating the two GUIs do no match at all i.e. the operations consisting of deleting, relabeling involves almost all elements. The reason behind the negative results were:

① When we try to build an android app, there are certain elements which may be embedded by Android SDK. For example: for an app developed with pre-configured templates can have many extra elements which are not visible and may not be scanned by accessibility service. Figure: 16 shows a sample of XML for a login application in which one was manually developed (without using any pre-configured templates of login as present in Android SDK) and the other was developed using the login activity provided by Android SDK.

As highlighted in Figure: 16 (next page), which is comparing the GUI elements log, we can see that the activity created using the template has an additional element just after the 3rd element - FrameLayout (highlighted on left). So if we try to compare the two XML's, the results shows no match with maximum elements as a part of mismatch and having no similarity.

Although when we clearly observe the two XML's we can see that the middle portion (highlighted in green) of the two XML's match to some extent with the Name property not entirely same. This is because now android has introduced compatible elements in the latest version and thus we obtain the elements name as defined with compatibility. Although the elements are same (like AppCompatEditText and EditText are same elements representing an Edit Text Box), but there Name property differs which gives us negative results on using Tree-Edit Algorithm. Scanning till the end we observe an extra section of code embedded (highlighted with red box on left), which is totally missing in the activity created manually. If we try to compare just the look and feel of the two activities, then they



```
<State Alias="tl" ScreenName="TestLogin">
 <GUIObject><Name>LinearLayout</Name>
  <GUIObject><Name>ViewStub</Name>
  </GUIObject>
  <GUIObject><Name>FrameLayout</Name>
   <GUIObject><Name>ActionBarOverlayLayout</Name>
    <GUIObject><Name>ContentFrameLayout</Name>
     <GUIObject><Name>LinearLayout</Name>
      <GUIObject><Name>ProgressBar</Name>
      </GUIObject>
      <GUIObject><Name>ScrollView</Name>
       <GUIObject><Name>LinearLayout</Name>
        <GUIObject><Name>TextInputLayout</Name>
         <GUIObject><Name>FrameLayout</Name>
          <GUIObject><Name>AppCompatAutoCompleteTextView</Name>
          </GUIObject>
         </GUIObject>
        </GUIObject>
        <GUIObject><Name>TextInputLayout</Name>
         <GUIObject><Name>FrameLayout</Name>
          <GUIObject><Name>AppCompatEditText</Name>
          </GUIObject>
          <GUIObject><Name>CheckableImageButton</Name>
          </GUIObject>
         </GUIObject>
        </GUIObject>
        <GUIObject><Name>AppCompatButton</Name>
        </GUIObject>
       </GUIObject>
      </GUIObject>
     </GUIObject>
    </GUIObject>
    <GUIObject><Name>ActionBarContainer</Name>
     <GUIObject><Name>Toolbar</Name>
      <GUIObject><Name>AppCompatTextView</Name>
      </GUIObject>
     </GUIObject>
     <GUIObject><Name>ActionBarContextView</Name>
     </GUIObject>
    </GUIObject>
   </GUIObject>
  </GUIObject>
 </GUIObject>
</State>
```

Login Activity created from a template.

```
<State Alias="tl2" ScreenName="TestLogin2">
 <GUIObject><Name>LinearLayout</Name>
  <GUIObject><Name>ViewStub</Name>
  </GUIObject>
  <GUIObject><Name>FrameLayout</Name>
   <GUIObject><Name>LinearLayout</Name>
    <GUIObject><Name>ScrollView</Name>
     <GUIObject><Name>LinearLayout</Name>
      <GUIObject><Name>TextInputLayout</Name>
       <GUIObject><Name>FrameLayout</Name>
       </GUIObject>
      </GUIObject>
      <GUIObject><Name>EditText</Name>
      </GUIObject>
      <GUIObject><Name>TextInputLayout</Name>
       <GUIObject><Name>FrameLayout</Name>
       </GUIObject>
      </GUIObject>
      <GUIObject><Name>EditText</Name>
      </GUIObject>
      <GUIObject><Name>Button</Name>
      </GUIObject>
     </GUIObject>
    </GUIObject>
   </GUIObject>
  </GUIObject>
 </GUIObject>
</State>
```

Login Activity created by simply adding the elements manually without any template

Figure 16: Comparing XML Log for Login Activity

almost look similar. Figure: 17 (next page) shows a comparison of two GUI's.

(2) The apps can be developed with any type of template, with some dynamic controls loading in code behind and thus when we try to compare the XML Logs of two GUIs directly, it often fails. There may be difference in which the name for each element appears (due to difference in android version), thus we cannot simply apply Tree Edit Algorithm instead, we require different approach where we can calculate the difference in the branches of two trees.

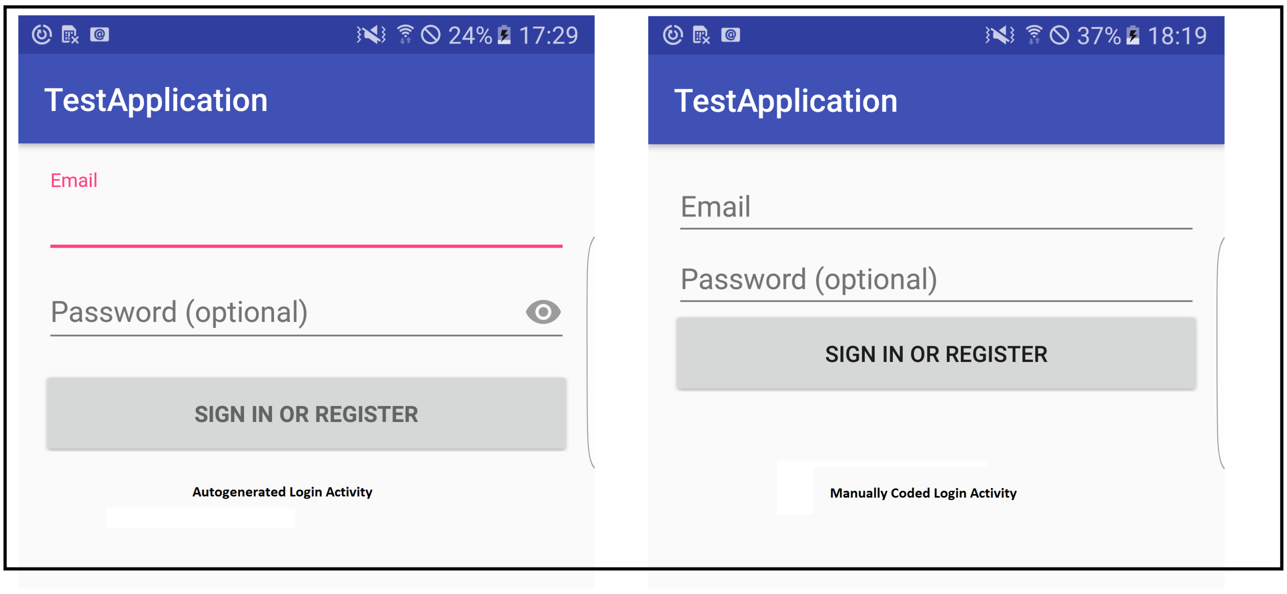
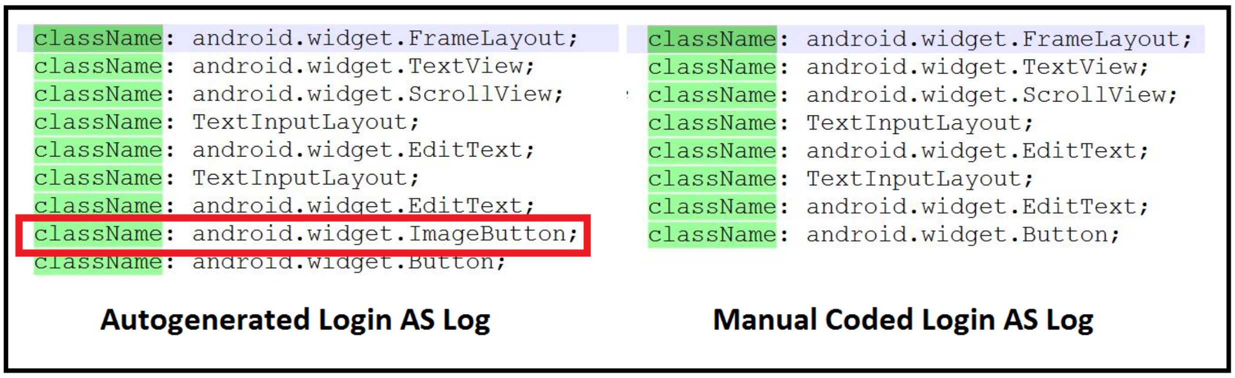


Figure 17: Comparing Login Activity - Snapshots

Comparing the two GUI's as shown in Figure: 17, we find two major differences: first in the layout, there are differences with respect to the positions of all the elements and second there is some difference in the elements used in the two GUIs. Although both GUIs make use of **TextInputLayout**, the GUI on left (from template) makes use of **AutoCompleteTextView** instead of a normal **EditText**. On further analysis, we obtained the accessibility log for both the apps fetching all the GUI elements. From the log obtained, as shown in Figure: 18, we can see only one difference. The app created without template does not have a **ImageButton**, rest analyzing the log we see the elements are all same. Figure: 18 is a small snapshot of the accessibility log for the two Login GUIs with just the classNames highlighted.



```
className: android.widget.FrameLayout;
className: android.widget.TextView;
className: android.widget.ScrollView;
className: TextInputLayout;
className: android.widget.EditText;
className: TextInputLayout;
className: android.widget.EditText;
className: android.widget.ImageButton;
className: android.widget.Button;
```

Autogenerated Login AS Log

```
className: android.widget.FrameLayout;
className: android.widget.TextView;
className: android.widget.ScrollView;
className: TextInputLayout;
className: android.widget.EditText;
className: TextInputLayout;
className: android.widget.EditText;
className: android.widget.Button;
```

Manual Coded Login AS Log

Figure 18: Accessibility Log for Login Activities.

5.2 Results from other apps

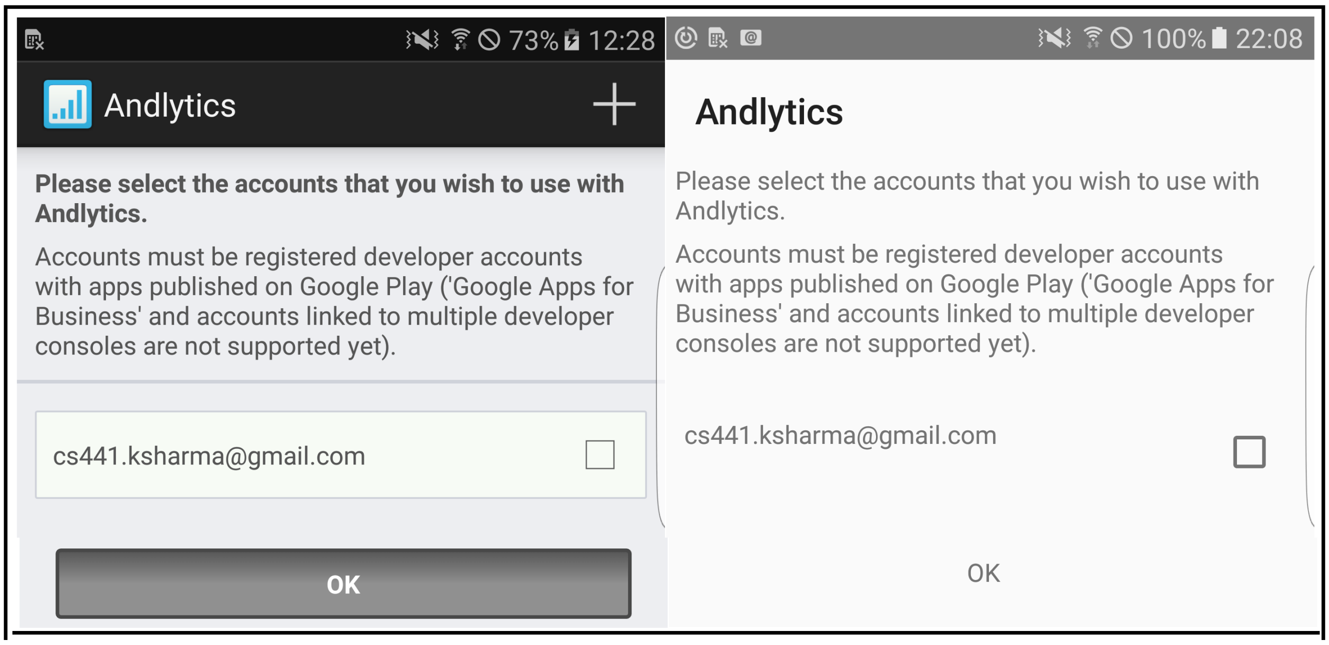
In order to analyze the phished GUIs, the hierarchy of GUI elements, their GUI and accessibility service traced log for 15 apps was tested manually. Following were some of the important results obtained:

Table II: GRAPHICAL USER INTERFACE, HIERARCHY AND ACCESSIBILITY LOG ANALYSIS FOR GITHUB DOWNLOADED APPS

App Name	GUI Layout	GUI Hierarchy and Accessibility Service (AS) Log Analysis
PocketHub	GUI Loads with button missing	Button which is a part of top header is missing from the phished GUI. The other elements are not logged in accessibility service and hence are missing in phished GUI's hierarchy log. The AS log matches
WiFiTalkie	GUI Loads with missing text and button	Button which is a part of top header is missing from the phished GUI. All other elements in phished GUI hierarchy matches with original GUI hierarchy. The AS log matches.
Coach Assistant	GUI matches perfectly	The hierarchy log differences in the names of the elements. The AS log matches.
Commons	GUI matches with image missing	The GUI elements hierarchy matches to some extent with GridView missing. The AS Log matches with only 2 repeated entries for TextView.
StudIPApp	GUI loads with missing image and text on the bottom	The original GUI hierarchy does not loads elements as the binding of elements is dynamic, thus the two GUI hierarchy logs do not match. The AS log matches.

Table II Continued: GRAPHICAL USER INTERFACE, HIERARCHY AND ACCESSIBILITY LOG ANALYSIS FOR GITHUB DOWNLOADED APPS

App Name	GUI Layout	GUI Hierarchy and Accessibility Service (AS) Log Analysis
Jupiter Broadcasting	GUI loads with missing image	The GUI element hierarchy log matches to some extent with only TableLayout missing. The AS log matches.
Andlytics	GUI loads with missing button	Button which is a part of top header is missing from the phished GUI. The GUI elements log differs in various aspects like missing progress bar as this does not appears in AS log. The AS log matches except the phished log has a missing button.
BeeExample	GUI loads with missing bottom texts	The elements GUI hierarchy log differs as many elements are not logged by AS and the tabs view is replaced by radio buttons, as we get radio buttons in AS log. The AS log has tabhost element missing and the rest log matches.
Wikivoyage Offline	GUI loads and matches	The elements GUI hierarch log matches with some portion of elements and phished log has a missing progress bar. The AS log matches.
Meta Filter	GUI loads and matches to some extent	The elements GUI hierarchy log mismatches since the loading of text on the screen is dynamic. Similarly, the AS log also mismatches.



Please select the accounts that you wish to use with Andlytics.

Accounts must be registered developer accounts with apps published on Google Play ('Google Apps for Business' and accounts linked to multiple developer consoles are not supported yet).

cs441.ksharma@gmail.com

OK

Please select the accounts that you wish to use with Andlytics.

Accounts must be registered developer accounts with apps published on Google Play ('Google Apps for Business' and accounts linked to multiple developer consoles are not supported yet).

cs441.ksharma@gmail.com

OK

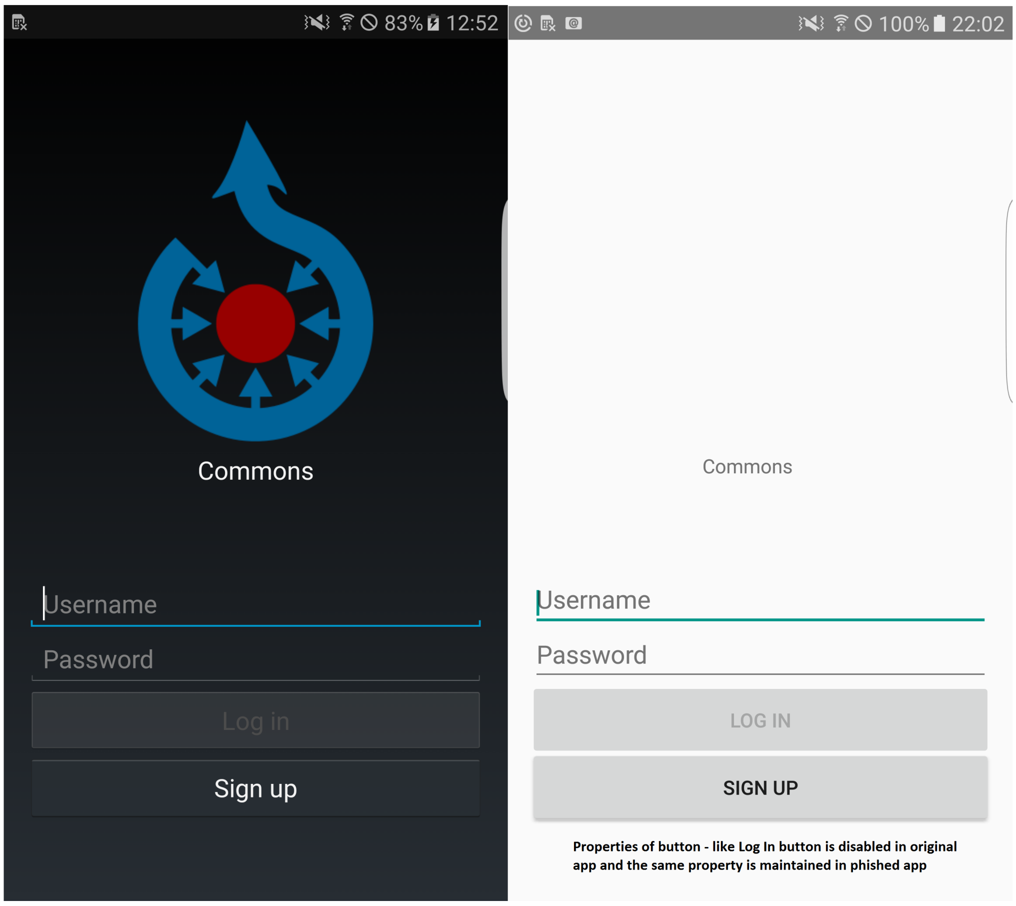
Figure 19: Original vs Phished GUI for **Andlytics App**

5.3 Snapshots comparing Original vs Phished GUIs

Figure: 19 to Figure: 21 are the phished GUI's obtained from SEAPHISH's framework. The left side of the images shows the original app and the right side shows the phished app. Each phished GUI contains almost all elements as seen in the original app. We have limitations in binding the header part of the app. Thus as seen in Figure: 19, the top right symbol - **+** is missing in our phished GUI.

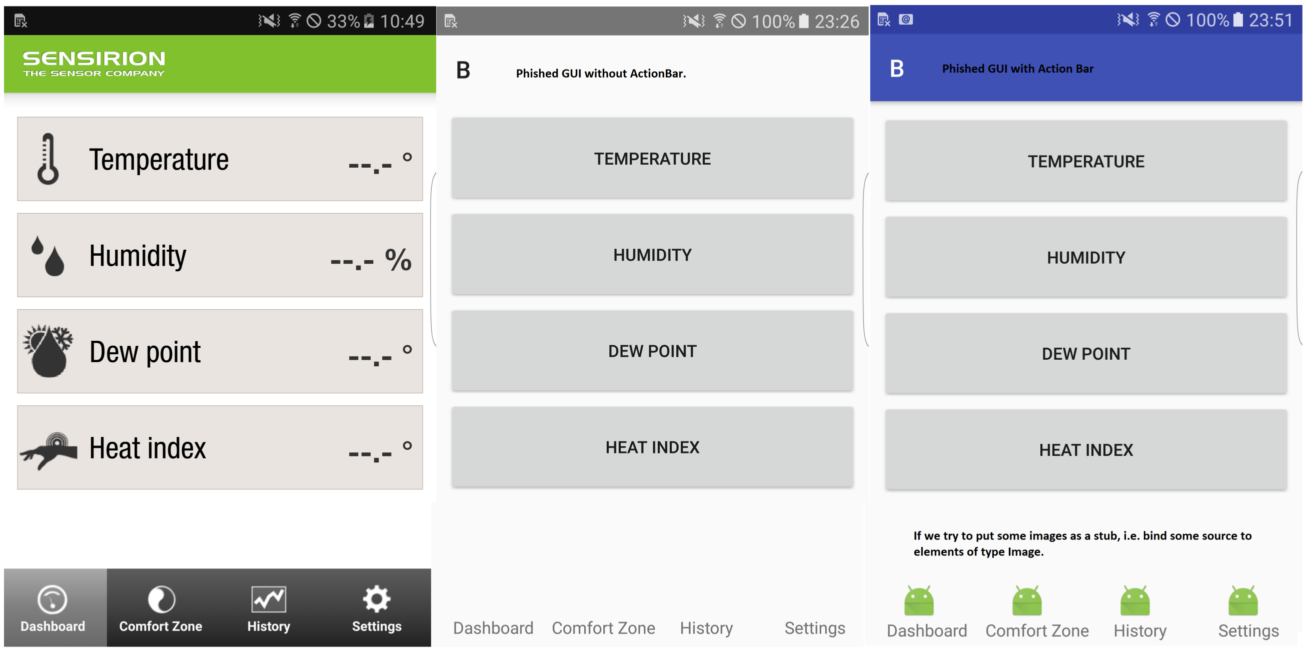
Apart from just re-constructing elements of the GUI, we also maintain all the properties of the elements. For example, in Figute: 20 we can see that the **Log In** button is disabled in original GUI app and the same is maintained in phished GUI app.

We also tried to re-construct the GUI by putting a stub image if we have any image elements in the original app. But the placing of the image to correct position and its width and height as calculated from the boundsInScreen property does not gives proper results in all apps. We were able to place the stub images accurately for one of the downloaded apps: **SmartGadget**. In Figure: 21, we have shown the two variations of phished GUI. The middle GUI shows the phished GUI which was created by removing the Action Bar and replacing it with Tool Bar. In the 3rd (the left most) GUI, we used a normal Action Bar and bound the interface with stub images. But these stub images did not work in other apps, the image size and location as calculated in our code does not provides accurate information and hence placing these stub images distorts the appearance.



Properties of button - like Log In button is disabled in original app and the same property is maintained in phished app

Figure 20: Original vs Phished GUI for **Commons App**



If we try to put some images as a stub, i.e. bind some source to elements of type Image.

Figure 21: Original vs Phished GUI for **SmartGadget App**

6. Conclusion

The malwares/viruses or malicious apps exploiting android's accessibility service do not just attempt to steal sensitive information, but can also perform unauthorized operations. Accessibility services plays an important role in all operating systems, but they also pose a serious security and privacy problem since these services can effectively simulate the human-user interaction and such channels can be exploited by malwares/viruses. There are several static and dynamic analysis methods [106], [51] etc. which helps in detecting malwares and malicious apps but these methods mainly analyze the permissions requested by an app, the API calls or tracks the flow of data. With SEAPHISH framework, we aim to dynamically detect the malicious apps or malwares that may/may not use accessibility service and attempt to steal sensitive information without providing access for that application and without requiring users to provide fine-grained permission control to applications, without asking users to identify or track sensitive information and without installing any virtual sandboxes or honeypot virtual environments.

Our goal is to automatically generate phishing apps that simulate the real smartphone apps from which the malicious apps may attempt to steal sensitive information or perform any unauthorized operation. In order to detect such malicious apps, we generate fake data which lies in the range of the real data as used by users. We try to defend the smartphone against malicious app(s) which may try to steal sensitive data from our generated phished app or even attempt to steal data from the servers which is connected to our phishing app. Thus, we use **phishing**, a deception to deceive the malicious app and prevent leaking of sensitive information and deny any unauthorized operations performed by malicious app.

References

1. About blackduck openhub. `http://blog.openhub.net/about`.
2. Accessibility. `http://developer.android.com/guide/topics/ui/accessibility/index.html`.
3. Ada standards for accessible design. `https://www.ada.gov/2010ADAstandards_index.htm`.
4. The americans with disabilities act of 1990 and revised ada regulations implementing title ii and title iii. `https://www.ada.gov/2010_regs.htm`.
5. Android accessibility: Accessibilitynodeinfo. `https://developer.android.com/reference/android/view/accessibility/AccessibilityNodeInfo.html`.
6. Android accessibility events overview. `https://developer.android.com/reference/android/viewaccessibility/AccessibilityEvent.html`.
7. Android accessibility service overview. `https://developer.android.com/reference/android/accessibilityservice/AccessibilityService.html`.
8. Android activity xml template. `https://developer.android.com/guide/topics/manifest/activity-element.html`.
9. Android application fundamentals. `https://developer.android.com/guide/components/fundamentals.html`.
10. Android application package. `https://en.wikipedia.org/wiki/Android_application_package`.
11. Android debug bridge. `https://developer.android.com/studio/command-line/adb.html`.
12. Android ui overview. `https://developer.android.com/guide/topics/ui/overview.html`.
13. Android version market share distribution among smartphone owners as of september 2016. `https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/`.
14. Apktool introduction. `https://ibotpeaches.github.io/Apktool/documentation/`.
15. Application development. `http://motivesense.com/application-development/`.
16. Developing an accessibility service. `https://developer.android.com/training/accessibility/service.html`.

17. Mobile application. https://www.techopedia.com/definition/2953/mobile-application-mobile-app.

18. Number of available applications in the google play store from december 2009 to december 2016. https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/.

19. Sikulix by raiman. http://sikulix.com/.

20. Sikulix by raiman. https://developer.android.com/studio/profile/am-basics.html.

21. Smartphone os market share, 2016 q3. http://www.idc.com/promo/smartphone-market-share/os.

22. A tool for reverse engineering android apk files. https://ibotpeaches.github.io/Apktool/.

23. S. Adappa, V. Agarwal, S. Goyal, P. Kumaraguru, and S. Mittal. User controllable security and privacy for mobile mashups. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 35–40, New York, NY, USA, 2011. ACM.

24. C. C. Aggarwal and P. S. Yu. *Privacy-Preserving Data Mining: Models and Algorithms*. Springer, 2008.

25. T. Ahmed, R. Hoyle, K. Connelly, D. Crandall, and A. Kapadia. Privacy concerns and behaviors of people with visual impairments. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 3523–3532, New York, NY, USA, 2015. ACM.

26. F. Akhter, M. C. Buzzi, M. Buzzi, and B. Leporini. Conceptual framework: How to engineer online trust for disabled users. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*, WI-IAT '09, pages 614–617, Washington, DC, USA, 2009. IEEE Computer Society.

27. M. H. Almeshekah and E. H. Spafford. Planning and integrating deception into computer security defenses. In *Proceedings of the 2014 New Security Paradigms Workshop*, NSPW '14, pages 127–138, New York, NY, USA, 2014. ACM.

28. H. M. Almohri, D. D. Yao, and D. Kafura. Droidbarrier: Know what is executing on your android. In *Proceedings of the 4th ACM Conference on Data and Application Security and Privacy*, CODASPY '14, pages 257–264, New York, NY, USA, 2014. ACM.

29. Y. Amit. Accessibility clickjacking – the next evolution in android malware that impacts more than 500 million devices [update – 1.34 billion devices!]. *Skycure, https://www.skycure.com/blog/accessibility-clickjacking/*, Mar. 2016.

30. Y. Amit. Inside the android accessibility clickjacking malware. *Techbeacon, http://techbeacon.com/inside-android-accessibility-clickjacking-malware*, Mar. 2016.

31. D. Arp, M. Spreitzenbarth, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of android malware in your pocket, 2014.

32. K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie. Pscout: Analyzing the android permission specification. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 217–228, New York, NY, USA, 2012. ACM.

33. M. Backes, S. Bugiel, and S. Gerling. Scippa: System-centric ipc provenance on android. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 36–45, New York, NY, USA, 2014. ACM.

34. V. L. Baker and J. Wong. Traditional development practices will fail for mobile apps. *Gartner, `http://www.gartner.com/newsroom/id/2823619`*, G00261561, Aug. 2014.

35. D. Barrera, J. Clark, D. McCarney, and P. C. van Oorschot. Understanding and improving app installation security mechanisms through empirical analysis of android. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 81–92, New York, NY, USA, 2012. ACM.

36. A. R. Beresford, A. Rice, N. Skehin, and R. Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 49–54, New York, NY, USA, 2011. ACM.

37. C. Bertolini and A. Mota. A framework for gui testing based on use case design. In *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '10, pages 252–259, Washington, DC, USA, 2010. IEEE Computer Society.

38. A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, and G. Vigna. What the app is that? deception and countermeasures in the android user interface. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 931–948, Washington, DC, USA, 2015. IEEE Computer Society.

39. M. F. Cabrera-Umpiérrez, A. R. Castro, J. Azpiroz, J. B. M. Colomer, M. T. Arredondo, and J. Cano-Moreno. Developing accessible mobile phone applications: The case of a contact manager and real time text applications. In *Proceedings of the 6th International Conference on Universal Access in Human-computer Interaction: Context Diversity - Volume Part III*, UAHCI'11, pages 12–18, Berlin, Heidelberg, 2011. Springer-Verlag.

40. E. Carter, A. Chiu, J. Esler, G. Serrao, and B. Stultz. When does software start becoming malware? `http://blogs.cisco.com/security/talos/infinity-toolkit?f_l=sd`, September 2015.

41. E. Chalkia and E. Bekiaris. A harmonised methodology for the components of software applications accessibility and its evaluation. In *Proceedings of the 6th International Conference on Universal Access in Human-computer Interaction: Design for All and eInclusion - Volume Part I*, UAHCI'11, pages 197–205, Berlin, Heidelberg, 2011. Springer-Verlag.

42. L. Chang. New malware, accessibility clickjacking, affects 65% of android devices. *Digital trends, `http://www.digitaltrends.com/mobile/android-accessibility-clickjacking/#:X858zf5ipHCOFA`*, Mar. 2016.

43. Q. A. Chen, Z. Qian, and Z. M. Mao. Peeking into your app without actually seeing it: Ui state inference and novel android attacks. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 1037–1052, Berkeley, CA, USA, 2014. USENIX Association.

44. D. Chu, A. Kansal, J. Liu, and F. Zhao. Mobile apps: It's time to move up to condos. In *Proceedings of the 13th USENIX Conference on Hot Topics in Operating Systems*, HotOS'13, pages 16–16, Berkeley, CA, USA, 2011. USENIX Association.

45. K. M. Conroy, M. Grechanik, M. Hellige, E. S. Liongosari, and Q. Xie. Automatic test generation from gui applications for testing web services. In *2007 IEEE International Conference on Software Maintenance*, pages 345–354, Oct 2007.

46. csinfotech. Android adware abuses accessibility service to install apps. *CyberSecurity,* `http://www.csinfotech.org/cyber-news?417/-Android-Adware-Abuses-Accessibility-Service-to-Install-Apps`, Nov. 2015.

47. K. Cunningham. *Accessibility Handbook: Making 508 Compliant Websites*. O'Reilly Media, 2012.

48. Darker. Darker's enabler. `http://progress-tools.x10.mx/denabler.html`, September 2015.

49. L. Deshotels. Inaudible sound as a covert channel in mobile devices. In *Proceedings of the 8th USENIX Conference on Offensive Technologies*, WOOT'14, pages 16–16, Berkeley, CA, USA, 2014. USENIX Association.

50. M. Dorigo, B. Harriehausen-Mühlbauer, I. Stengel, and P. S. Dowland. Survey: Improving document accessibility from the blind and visually impaired user's point of view. In *Proceedings of the 6th International Conference on Universal Access in Human-computer Interaction: Applications and Services - Volume Part IV*, UAHCI'11, pages 129–135, Berlin, Heidelberg, 2011. Springer-Verlag.

51. W. Enck. Defending users against smartphone apps: Techniques and future directions. In *Proceedings of the 7th International Conference on Information Systems Security*, ICISS'11, pages 49–70, Berlin, Heidelberg, 2011. Springer-Verlag.

52. A. Evans and J. O. Wobbrock. Input observer: Measuring text entry and pointing performance from naturalistic everyday computer use. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1879–1884, New York, NY, USA, 2011. ACM.

53. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638, New York, NY, USA, 2011. ACM.

54. A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 3–14, New York, NY, USA, 2011. ACM.

55. A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2Nd USENIX Conference on Web Application Development*, WebApps'11, pages 7–7, Berkeley, CA, USA, 2011. USENIX Association.

56. A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14, New York, NY, USA, 2012. ACM.

57. A. P. Felt, H. J. Wang, A. Moshchuk, S. Hanna, and E. Chin. Permission re-delegation: Attacks and defenses. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

58. E. Fernandes, Q. A. Chen, J. Paupore, G. Essl, J. A. Halderman, Z. M. Mao, and A. Prakash. Android UI Deception Revisited: Attacks and Defenses. In *Proceedings of the 20th International Conference on Financial Cryptography and Data Security (FC'16)*, Barbados, February 2016.

59. D. Fisher. Clickjacking bug affects 95 percent of android devices. *Pindrop Security,* `https://www.onthewire.io/accessibility-services-bug-affects-95-percent-of-android-devices/`, May 2016.

60. J. Foliot. User statistics – people with disabilities. `http://john.foliot.ca/user-statistics-people-with-disabilities/`, 2015.

61. D. Ganguly. Accessibility clickjacking attack threatens half a billion android devices. *Android Joint,* `http://www.androidjoint.com/2016/03/05/accessibility-clickjacking-attack-threatens-half-billion-android-devi` Mar. 2016.

62. A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035, New York, NY, USA, 2014. ACM.

63. U. S. Government. Section 508 of the Rehabilitation Act. *http://www.access-board.gov/508.htm*, 1998.

64. M. C. Grace, Y. Zhou, Z. Wang, and X. Jiang. Systematic detection of capability leaks in stock android smartphones. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

65. M. Grechanik, C. W. M. B.M. Mainul Hossain, A. Baisal, and D. S. Rosenblum. Differencing graphical user interfaces, 2012.

66. M. Grechanik and K. M. Conroy. Composing integrated systems using gui-based applications and web services. In *IEEE International Conference on Services Computing (SCC 2007)*, pages 68–75, July 2007.

67. M. Grechanik, K. M. Conroy, and K. S. Swaminathan. Creating web services from gui-based applications. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07)*, pages 72–79, June 2007.

68. M. Grechanik, Q. Xie, and C. Fu. Creating gui testing tools using accessibility technologies. In *Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation Workshops*, ICSTW '09, pages 243–250, Washington, DC, USA, 2009. IEEE Computer Society.

69. M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving gui-directed test scripts. In *2009 IEEE 31st International Conference on Software Engineering*, pages 408–418, May 2009.

70. K. Harris. Challenges and solutions for screen reader/i.t. interoperability. *SIGACCESS Access. Comput.*, (85):10–20, June 2006.

71. J. Hildenbrand. What is google talkback? `http://www.androidcentral.com/what-google-talk-back`, Sept. 2016.

72. D. Hoffman and L. Battle. Emerging issues, solutions & challenges from the top 20 issues affecting web application accessibility. In *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '05, pages 208–209, New York, NY, USA, 2005. ACM.

73. P. Hornyack, S. Han, J. Jung, S. Schechter, and D. Wetherall. These aren't the droids you're looking for: Retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 639–652, New York, NY, USA, 2011. ACM.

74. W. Hu, D. Octeau, P. D. McDaniel, and P. Liu. Duet: Library integrity verification for android applications. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless &#38; Mobile Networks*, WiSec '14, pages 141–152, New York, NY, USA, 2014. ACM.

75. J. Huang, X. Zhang, L. Tan, P. Wang, and B. Liang. Asdroid: Detecting stealthy behaviors in android applications by user interface and program behavior contradiction. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1036–1046, New York, NY, USA, 2014. ACM.

76. L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson. Clickjacking: Attacks and defenses, 2012.

77. S.-M. Hwang and H.-C. Chae. Design & implementation of mobile gui testing tool. In *Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology*, ICHIT '08, pages 704–707, Washington, DC, USA, 2008. IEEE Computer Society.

78. Q. Ismail, T. Ahmed, A. Kapadia, and M. K. Reiter. Crowdsourced exploration of security configurations. In *Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '15)*, pages 467–476, Apr. 2015.

79. A. Jain, H. Gonzalez, and N. Stakhanova. Enriching reverse engineering through visual exploration of android binaries. In *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, PPREW-5, pages 9:1–9:9, New York, NY, USA, 2015. ACM.

80. Y. Jang, C. Song, S. P. Chung, T. Wang, and W. Lee. A11y attacks: Exploiting accessibility in operating systems. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 103–115, New York, NY, USA, 2014. ACM.

81. J. Jeon, K. K. Micinski, J. A. Vaughan, A. Fogel, N. Reddy, J. S. Foster, and T. Millstein. Dr. android and mr. hide: Fine-grained permissions in android applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 3–14, New York, NY, USA, 2012. ACM.

82. X. Jiang and Y. Zhou. *Android Malware*. Springer Briefs in Computer Science. Springer, 2013.

83. J. Jung, S. Han, and D. Wetherall. Short paper: Enhancing mobile application permissions with runtime feedback and constraints. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, pages 45–50, New York, NY, USA, 2012. ACM.

84. D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.

85. S. K. Kane, C. Jayant, J. O. Wobbrock, and R. E. Ladner. Freedom to roam: A study of mobile device adoption and accessibility for people with visual and motor disabilities. In *Proceedings of the 11th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '09, pages 115–122, New York, NY, USA, 2009. ACM.

86. P. G. Kelley, S. Consolvo, L. F. Cranor, J. Jung, N. Sadeh, and D. Wetherall. A conundrum of permissions: Installing applications on an android smartphone. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security*, FC'12, pages 68–79, Berlin, Heidelberg, 2012. Springer-Verlag.

87. E. Kovacs. Most android devices prone to accessibility clickjacking attacks. *Security Week,* `http://www.securityweek.com/most-android-devices-prone-accessibility-clickjacking-attacks`, May 2016.

88. J. Kraunelis, Y. Chen, Z. Ling, X. Fu, and W. Zhao. *On Malware Leveraging the Android Accessibility Framework*, pages 512–523. Springer International Publishing, Cham, 2014.

89. O. Laadan, A. Shu, and J. Nieh. Capture: A desktop display-centric text recorder. In *Proceedings of the 14th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '12, pages 9–16, New York, NY, USA, 2012. ACM.

90. V. Lanaria. 500 million android devices at risk of accessibility clickjacking malware: What you should know. *Tech Times,* `http://www.techtimes.com/articles/138842/20160306/500-million-android-devices-at-risk-of-accessibility-clickjacking-mal` `htm`, Mar. 2016.

91. J. Lazar, D. Goldstein, and A. Taylor. *Ensuring Digital Accessibility through Process and Policy*. Morgan Kaufmann, 2015.

92. J. Lazar and H. Hochheiser. Legal aspects of interface accessibility in the u.s. *Commun. ACM*, 56(12):74–80, Dec. 2013.

93. C.-C. Lin, H. Li, X. yong Zhou, and X. Wang. Screenmilker: How to milk your android screen for secrets. In *NDSS*, 2014.

94. B. Liu, J. Lin, and N. Sadeh. Reconciling mobile app privacy and usability on smartphones: Could user privacy profiles help? In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 201–212, New York, NY, USA, 2014. ACM.

95. R. Lopes, K. Van Isacker, and L. Carriço. Redefining assumptions: Accessibility and its stakeholders. In *Proceedings of the 12th International Conference on Computers Helping People with Special Needs: Part I*, ICCHP'10, pages 561–568, Berlin, Heidelberg, 2010. Springer-Verlag.

96. J. Mankoff, G. R. Hayes, and D. Kasnitz. Disability studies as a source of critical inquiry for the field of assistive technology. In *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '10, pages 3–10, New York, NY, USA, 2010. ACM.

97. L. McLawhorn. Recent development: Leveling the accessibility playing field: Section 508 of the rehabilitation act. *NORTH CAROLINA JOURNAL OF LAW & TECHNOLOGY*, 3(1):63–100, 2001.

98. W. Meng, W. H. Lee, S. Murali, and S. Krishnan. Charging me and i know your secrets!: Towards juice filming attacks on smartphones. In *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*, CPSS '15, pages 89–98, New York, NY, USA, 2015. ACM.

99. A. Michail. Browsing and searching source code of applications written using a gui framework. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pages 327–337, New York, NY, USA, 2002. ACM.

100. Microsoft and Forrester. Wthe wide range of abilities and its impact on computer technology: A research study commissioned by microsoft corporation and conducted by forrester research, inc.,. *Forrester Research*, 2003.

101. L. Moreno and P. Martínez. A review of accessibility requirements in elderly users' interactions with web applications. In *Proceedings of the 13th International Conference on InteracciÓN Persona-Ordenador*, INTERACCION '12, pages 47:1–47:2, New York, NY, USA, 2012. ACM.

102. L. Moreno, P. Martinez, B. Ruiz, and A. Iglesias. Toward an equal opportunity web: Applications, standards, and tools that increase accessibility. *Computer*, 44(5):18–26, May 2011.

103. J. Mueller. *Accessibility for Everybody: Understanding the Section 508 Accessibility Requirements*. APress L. P., 2003.

104. PoPs. Pops ringtons & notifications. `https://play.google.com/store/apps/details?id=com.pops.app`, September 2015.

105. R. Potharaju, A. Newell, C. Nita-Rotaru, and X. Zhang. Plagiarizing smartphone applications: Attack strategies and defense techniques. In *Proceedings of the 4th International Conference on Engineering Secure Software and Systems*, ESSoS'12, pages 106–120, Berlin, Heidelberg, 2012. Springer-Verlag.

106. M. Rangwala, P. Zhang, X. Zou, and F. Li. A taxonomy of privilege escalation attacks in android applications. *Int. J. Secur. Netw.*, 9(1):40–55, Feb. 2014.

107. V. Rastogi, Y. C. Rui Shao, S. Z. Xiang Pan, and R. Riley. Are these ads safe: Detecting hidden attacks through the mobile app-web interfaces. NDSS '16, 2016.

108. A. Raza. Accessibility malware puts 65% percent of old version android devices at risk. *Hackread,* `https://www.hackread.com/accessibility-malware-puts-old-android-devices-risk/`, Mar. 2016.

109. M. Rotenberg. What you don't know about android application security.... *Whitesource,* `http://www.whitesourcesoftware.com/whitesource-blog/android-application-security/`, Mar. 2016.

110. A. Sahami Shirazi, N. Henze, A. Schmidt, R. Goldberg, B. Schmidt, and H. Schmauder. Insights into layout patterns of mobile user interfaces by an automatic analysis of android apps. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '13, pages 275–284, New York, NY, USA, 2013. ACM.

111. N. Santos, H. Raj, S. Saroiu, and A. Wolman. Trusted language runtime (tlr): Enabling trusted applications on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, pages 21–26, New York, NY, USA, 2011. ACM.

112. R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, and X. Wang. Soundcomber: A Stealthy and Context-Aware Sound Trojan for Smartphones. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium (NDSS)*, pages 17–33, Feb. 2011.

113. Sufatrio, D. J. J. Tan, T.-W. Chua, and V. L. L. Thing. Securing android: A survey, taxonomy, and challenges. *ACM Comput. Surv.*, 47(4):58:1–58:45, May 2015.

114. M. Sun, M. Zheng, J. C. S. Lui, and X. Jiang. Design and implementation of an android host-based intrusion prevention system. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 226–235, New York, NY, USA, 2014. ACM.

115. J. Tan, K. Nguyen, M. Theodorides, H. Negrón-Arroyo, C. Thompson, S. Egelman, and D. Wagner. The effect of developer-specified explanations for permission requests on smartphone user behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 91–100, New York, NY, USA, 2014. ACM.

116. G. L. D. Team. Go launcher ex notification. `https://play.google.com/store/apps/details?id=com.gau.golauncherex.notification`, September 2015.

117. R. Templeman, Z. Rahman, D. J. Crandall, and A. Kapadia. Placeraider: Virtual theft in physical spaces with smartphones. *CoRR*, abs/1209.5982, 2012.

118. V. S. Verykios, E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin, and Y. Theodoridis. State-of-the-art in privacy preserving data mining. *SIGMOD Rec.*, 33(1):50–57, Mar. 2004.

119. T. Vidas, D. Votipka, and N. Christin. All your droid are belong to us: A survey of current android attacks. In *Proceedings of the 5th USENIX Conference on Offensive Technologies*, WOOT'11, pages 10–10, Berkeley, CA, USA, 2011. USENIX Association.

120. Wikipedia. Window manager. `http://en.wikipedia.org/wiki/Window_manager#cite_note-1`, June 2015.

121. F. Wikipedia. Mobile app. `https://en.wikipedia.org/wiki/Mobile_app`.

122. D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Proceedings of the 2012 Seventh Asia Joint Conference on Information Security*, ASIAJCIS '12, pages 62–69, Washington, DC, USA, 2012. IEEE Computer Society.

123. Q. Xie, M. Grechanik, C. Fu, and C. Cumby. Guide: A gui differentiator. In *2009 IEEE International Conference on Software Maintenance*, pages 395–396, Sept 2009.

124. W. Yang, X. Xiao, R. Pandita, W. Enck, and T. Xie. Improving mobile application security via bridging user expectations and application behaviors. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, HotSoS '14, pages 32:1–32:2, New York, NY, USA, 2014. ACM.

125. W. Zhou, Y. Zhou, M. Grace, X. Jiang, and S. Zou. Fast, scalable detection of "piggybacked" mobile applications. In *Proceedings of the Third ACM Conference on Data and Application Security and Privacy*, CODASPY '13, pages 185–196, New York, NY, USA, 2013. ACM.

126. Z. Zorz. Almost all android users vulnerable to accessibility clickjacking attacks. *HelpNetSecurity,* `https://www.helpnetsecurity.com/2016/05/17/android-accessibility-clickjacking/`, May 2016.

127. Z. Zorz. Android banking malware may start using adware tricks. *HelpNetSecurity,* `https://www.helpnetsecurity.com/2016/05/05/android-banking-malware-adware/`, May 2016.

128. Z. Zorz. Screen overlay android malware is on the rise. *HelpNetSecurity,* `https://www.helpnetsecurity.com/2016/04/29/screen-overlay-malware-rise/`, Apr. 2016.

129. Z. Zorz. Source code of game changer android banking malware leaked online. *HelpNetSecurity,* `https://www.helpnetsecurity.com/2016/02/23/source-code-of-game-changer-android-banking-malware-leaked-online/`, Feb. 2016.

APPENDIX

Dr. Mark Grechanik and I have been working together on this research. Dr. Grechanik has authorized me (Ms. Kruti Sharma) to use his proposal plan which will be submitted to ACM and IEEE in future.

The subsections: *1.2 Overview of GUI* and *1.3 Background on the GUI Structure* have been referenced from Dr. Mark Grechanik's previously published work [45, 66–69, 123] and Dr. Grechanik has authorized me to use the content in my master's thesis. The section: *2. Background and Related Work* and *3. Motivation, Challenges and Proposed Framework* contains the text from the various papers that have been used as a reference in this research.

The text in all subsections (1.2,1.3) and section (2,3) are included as per the below copyright policies:

- **ACM Copyright Policy** :
  `https://www.acm.org/publications/policies/copyright-policy`
- **IEEE Copyright Policy** :
  `http://www.ieee.org/documents/author_version_faq.pdf`
- **Springer Copyright Policy** :
  `http://www.springer.com/us/open-access/authors-rights`

VITA

NAME:	Kruti Sharma
EDUCATION:	B.E., Computer Science, Rajiv Gandhi Proudyogiki Vishwavidyalaya, India, 2010
	M.S., Computer Science, University of Illinois at Chicago, Chicago, IL, 2017
EXPERIENCE:	University of Illinois at Chicago, Chicago, IL, Graduate Research Assistant, Aug 2016- May 2017
	University of Illinois at Chicago, Chicago, IL, Graduate Assistant, Aug 2015-Aug 2016
	Fulcrum WorldWide Software Ltd, India, Senior Software Engineer, Aug 2013 - July 2015
	Infosys Limited, India, Senior Software Engineer, Oct 2010 - July 2013