Mobile Application Testing using Automation Testing Mechanism

Sharanjeet Kaur1, Kulvinder Singh2, Sanjeev Dhawan3
1Student of M.Tech. (Software Engineering), 2,3Faculty of Computer Science & Engineering, Department of Computer Science & Engineering, University Institute of Engineering and Technology (U.I.E.T), Kurukshetra University, Kurukshetra (K.U.K)-136119, Haryana, INDIA.

Abstract: The research work proposes a new testing program to test and analyses the functionalities of mobile applications related to privacy information. Proposed framework and model have various valuable and novel elements, including support for Android's single-and different example parts, drifting names, declassification and underwriting capacities, and backing for legacy applications. We have built up a model of our framework on Android 4.0.4 and tried it on a Smart mobile, confirming that it can authorize basically helpful strategies that can be executed with negligible adjustment to off-the-rack applications. Automation of testing based upon these use cases will be further complicated by the increasing use of touch screens and physical input (such as gestures).

Keywords: Android APP, Privacy information, Tester, Mobile applications, app libraries.

I. INTRODUCTION

Android utilizes authorizations to secure segments and delicate APIs: a segment or API ensured by consent can be called just by applications that hold this authorization. Authorizations are strings (e.g., android. authorization. Web) characterized by the framework or proclaimed by applications. Applications secure authorizations just at introduce time, with the client's assent. Also, content suppliers use URI consents to progressively concede and repudiate access to their records, tables, and databases [1]. Different difficulties are connected with testing mobile applications. In the first place, it is hard to recreate the generation environment. Most testing must be performed utilizing test systems and emulators. Be that as it may, regardless of the fact that we can reenact a few parts of the application, the handset for instance, we can't make certain what happens when we attempt it over a genuine mobile system. These outcomes in numerous fields are failures [2]. Additionally, with different mobile applications accessible, and quickly advancing mobile innovation, analyzers end up at a misfortune to recognize and make a risk profile in view of an absence of involvement in this space. Along these lines, test cases may not be adequately intense, centered, or comprehensive. Test computerization frequently speeds up the procedure of test execution by diminishing the manual info required. It is a method that spares a considerable measure of time by empowering the execution of redundant tests with the assistance of PCs. On account of mobile applications, it is hard to robotize even the ordinary tests because of the inborn requirements of equipment like less memory and poor handling power on which these applications execute. These tests must be executed physically. This requests more manual testing asset and time. Experiment prioritization taking into account hazard turns out to be progressively essential, to minimize the quantity of tests, and segregate the all the more capable tests from the frail ones. A risk inventory helps in experiment prioritization, by permitting the product analyzer to concentrate on the disappointment classes of interest and guide the risks in the application under test from a pre-organized risk profile [3].

Contributions of the paper are as follows:

- The main objective was to plan frameworks that offer users with Visibility into which of their conduct can be connected by mobile administrations and control over this data presentation.
- To make a security insurance framework for Android that shields private information from being penetrated by substituting shadow information set up of the information that clients’ needs to keep private and by blocking system transmissions of delicate information to untrusted destinations.
- Malware that goes about as a botnet, uncovering a variety of mobility controlled gadget capacities.

II. BACKGROUND STUDY

Cell phones have turned into a focal gadget in individuals’ everyday lives, and they hold a great deal of individual data including photographs, contacts, area, et cetera. This data can be gotten to by numerous applications worked by outsider engineers to give clients usefulness. Since applications are not as a matter of course believed, it is vital
to see how applications utilize and uncover client data. Data stream following is an apparatus that gives an approach to comprehend what private data is spilled when it goes over the limit of a framework [4]. Individual data available by mobile gatherings is not restricted to clearly private information, for example, what we saw from cell phones, however incorporates how individuals use applications or scan pages. Administration suppliers attempt to associate however much action as could be expected, for instance, following client conduct on the web or applications is presently a typical practice. While there are insurance systems against following in the application layer, system level fingerprints (as in the IP location of a customer) still uncover a generous measure of data about the client’s action. Additionally, clients have no influence over system level fingerprints and what data can be surmised from them [5].

Cloud testing which offers a mutual test environment so client need not to set up and keep up different testing stages to guarantee site or mobile application convey ability and similarity and different issues like usefulness, interoperability, execution and so forth. Test scripts are recorded by the client from the neighborhood program or with any of the OS if there should raise an occurrence of a mobile application. The scripts are then submitted to the cloud for testing to be executed naturally with the most recent testing devices accessible there. In spite of the fact that cloud testing is giving an expansive number of static and element testing and different administrations however applications facilitated on mobile mists have lower controllability, vulnerabilities and discernibleness when contrasted with routine in house facilitated applications [6].

With the development in wearable gadgets and PC vision algorithms, applications may utilize a live video stream as a contribution to the not so distant future. Our preparatory study [7] demonstrated that these applications would utilize a greater number of assets from the cell phone than beforehand suspected. I then investigated routes by which we can give proficient framework backing to constant mobile vision (in a joint effort with scientists from Microsoft Research). We handled this issue from two headings. In the first place, we investigated a camera design that utilizes low-control sensors to specifically empower video detecting [8]. At that point, we assembled a system for running profound neural systems, MCDNN. With machine learning procedures, MCDNN can produce numerous variations of a model having diverse asset use and precision, then timetable solicitations over a cell phone and the cloud to get high exactness while staying inside the vitality and cost budgets.

### III. LITERATURE REVIEW

V. Balasubramanian [9] has talked about different administration models of distributed computing concentrating on Taas model, its engineering different administration parts and the key elements of the Taas design. Chonho Lee [10] has talked about additional about Mobile distributed computing engineering and applications furthermore different issues identified with the cloud environment, security and the protection of the information and the execution. The definition, Scope, significance and the mobile testing process on cloud and diverse methodologies with correlation with customary mobile testing methodologies are illuminated by Jerry Gao [11]. The favorable circumstances talked about incorporates lesser test environment cost because of different virtual machines (VM) for test situations on one host server, disconnected VM so that on the off chance that one VM accidents would not influence the other VM's and put away VM picture documents that can be uninhibitedly duplicated or moved.

Riungu and Taipale [12] talked about that testing of programming's on cloud is affected by conditions, for example, level of space learning expected to test an application, adaptable and financially savvy, security and economy of scale and distributed computing as a blasting worldview and the product analyzers to clean their aptitudes. According to Vengattaraman [13] in cloud environment for programming testing there are quantities of mists having distinctive testing systems. So it is the obligation of administration director to oversee and co-ordinate between the center exercises of separate layered substances of general cloud environment and characterizes the arrangement of required mists relying upon the testing strategies required by any result of the customer. What is the significance of cloud based testing of mobile applications and what are the different sorts of CBT are talked about by Mohata [14].

Computerization has had a responsible effect in the testing field is examined by Fuyang Peng [15] i.e. implantation and engineering of cloud based programmed programming test environment (CASTE). To test programming on cloud it has two conditions i.e. the framework under test is available online and testing foundation is facilitated in the cloud. The robotized testing environment is accessible all the time round the clock as the real point of preference. Testing endlessly affects software's; equipment's and systems also.

A great deal of work is done here and significantly more to go, in the event that we accentuate on programming testing of mobile applications, Muccini [16] talked about that before conventional methodology was taken after. Presently a day's robotized trying is assuming a key part to test mobile applications to conquer the difficulties as it is unique in relation to desktop testing. Mobile application testing needs diverse and specific procedures. A portion of the particular idiosyncrasies and ramifications of testing for apps4mobile (conventional applications changed to keep running on cell phones) and mobile applications (that make utilization of logical data to create connection based yields) e.g. mobile availability needs test for unwavering quality execution and so on., self-rule
needs observing of vitality utilization, adjustment needs test for adjustment rightness, differing qualities of telephones and OS needs similarity and OS testing, new PL needs white box and discovery testing and most critical now a days for touch screen needs touch screen reaction and ease of use testing.

As indicated by Starov and vilkomir [17] there are TaaS arrangements in mobile advancement as they have proposed CTMOS (Cloud Testing of mobile frameworks) with center foundation that includes all center testing functionalities, nonfunctional testing, experiment era and test arrangement approaches together actualized in one stage over the cloud. It additionally gives plausibility of significantly all use cases.

IV. PROPOSED WORK

Proposed work has two major enhancements in regards with existing work, these are as follows:

- Proposed work includes the text based privacy based as well as non-text-based application testing. Text based includes application like message, email notifications, contact numbers, addresses, phone numbers, and call records etc. And non-text-based includes runtime context of application like phone state, battery state, camera and GPS information etc. Details of this enhancement are explained in next section experimental design. The APIEM encoding algorithm used in existing work APIEM-DES that focuses on text based privacy information. While in proposed work we are using a hybrid concept of APIEM-DES and APIEM-Cesar algorithm to enhance the results in terms of access time.

![Fig. 1: Proposed design of tester](image1)

V. TESTING PROCESS

To analyses the UI components of the application that you want to test, perform the following steps after installing the application given in the example. [18]

- Connect your Android device to your development machine
- Open a terminal window and navigate to `<android-sdk>/tools/`
- Run the tool with this command

```
uiautomatorviewer
```

Commands would be followed as shown below

![Fig. 2: Command window to start UI Automator Viewer](image2)
You will see the following window appear. It is the default window of the UI Automator Viewer. Click on the devices symbol at the upper right corner. It will begin taking the UI XML preview of the screen as of now opened in the device. It would be something like this.

After that, you will see the snapshot of your device screen in the UI automator viewer window. On the right half of this window, you will see two allotments. The upper allotment clarifies the Nodes structure, the way the UI parts are orchestrated and contained. Tapping on every hub gives point of interest in the lower segment. [19] As a sample, consider the underneath figure. When you tap on the catch, you can find in the upper parcel that Button is chosen, and in the lower segment, its points of interest are appeared. Since this catch is snap capable, which is the reason its property of snap capable is set to true.

UI Automator Viewer additionally helps you to look at your UI in various introductions. For instance, simply change your device introduction to scene, and again catch the screen shot. Now you can create your own test cases and run it with uiautomatorviewer to examine them. In order to create your own test case, you need to perform the following steps – [20]

- From the Project Explorer, right-click on the new project that you created, then select Properties > Java Build Path, and do the following –
  - Click Add Library > JUnit then select JUnit3 to add JUnit support.
  - Click Add External JARs... and navigate to the SDK directory. Under the platforms directory, select the latest SDK version and add both the uiautomator.jar and android.jar files.
  - Extend your class with UiAutomatorTestCase
  - Right the necessary test cases.
  - Once you have coded your test, follow these steps to build and deploy your test JAR to your target Android test device.
  - Create the required build configuration files to build the output JAR. To generate the build configuration files, open a terminal and run the following command:

```bash
<android-sdk>/tools/android create uitest-project -n <name> -t 1 -p <path>
```

This is the name of the project that contains your UI automator test source files, and this is the path to the corresponding project directory.

- From the command line, set the ANDROID_HOME variable.

```bash
set ANDROID_HOME=<path_to_your_sdk>
```

- Go to the project directory where your build.xml file is located and build your test JAR.

```bash
ant build
```

- Deploy your produced test JAR document to the test device by utilizing the adb push order.

```bash
adb push <path_to_output_jar> /data/local/tmp/
```

- Run your test by following command –

```bash
adb shell uiautomator runtest LaunchSettings.jar -c com.uia.example.my.LaunchSettings
```

### VI. EXPERIMENTAL RESULTS

We note there are no proportional industrious fields when seeing advertisements through a program, and in this way these security vulnerabilities are exceptional to Android in-application promoting. Present study is completely executed and has been tried on a Smartphone. We extended Android's show document linguistic structure to bolster our names. Performance evaluation is measured in terms of total access time of overhead evaluation that includes the execution time taken to access the text based privacy information like conversation list, SMS or message details, contact details and contact list as well as execution time taken to access the non-text based privacy information like battery state, GPS information and passwords etc.

\[
\text{Total access time} = \text{text based privacy information} + \text{non-text based privacy information}
\]

<table>
<thead>
<tr>
<th>Method</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>375</td>
</tr>
<tr>
<td>APIEM-Cesar</td>
<td>383</td>
</tr>
<tr>
<td>APIEM-DES</td>
<td>386</td>
</tr>
<tr>
<td>Proposed</td>
<td>391</td>
</tr>
</tbody>
</table>

Table-1: Comparison table for Access time

Fig. 3 and table 1 presents the comparison of different APIEM encoding algorithm with proposed algorithm in terms of their access time.
A summary of these test outcomes are additionally appeared in Table 1. Not all capacities were really tried, sufficiently only of them to demonstrate that the test technique was working for this application. In light of these outcomes I guarantee that this technique is appropriate for computerized testing of capacities that are generally extremely tedious to test. In reality, the log document investigation arrangement will by and large discover the mistakes that are specifically associated with the components of call setup and hinderers in the framework.

VII. CONCLUSION

With all the manual work going into testing of mobile applications on all these distinctive gadgets and variants, the issue is to figure out how to locate a mechanized approach to perform this testing. Mobile application testing is different from traditional desktop testing. Paper describes the respective peculiarities and implications for Apps4mobile and mobile Apps. The need of testing automation towards all layers i.e. application frame work, OS and hardware layers. Outsourcing the testing on cloud or by crow-based solution is cost effective and feasible. Information security and testing at all layers. Automation of testing of mobile application is also discussed. Further testing for Nokia; nonetheless, there are comparative applications for Android, Windows Phone, and Apple's iOS that could profit by comparable investigative studies and testing.

VIII. REFERENCES