INTERNATIONAL JOURNAL FOR TRENDS IN ENGINEERING & TECHNOLOGY
VOLUME 4 ISSUE 2 – APRIL 2015 - ISSN: 2349 - 9303

EETA: Enhancing and estimating the transformation of attacks in android application

P.Kalaiarasi
B.Tech(IT)-Final yr
SNS College of Engineering
Coimbatore
kalaiprem4321@gmail.com

F.Merlyene Rovina
B.Tech(IT)-Final yr
SNS College of Engineering
Coimbatore
merlyene2@gmail.com

R.Sowdeeswari
B.Tech(IT)-Final yr
SNS College of Engineering
Coimbatore
sowdusns@gmail.com

A.Roshini
Asst.Professor
SNS College of Engineering
Coimbatore
roshini_cse@yahoo.com

ABSTRACT:
Android app security plays a major role in the entire mobile development industry. We tend to evaluate the industrial mobile anti-malware merchandise for Android and take a look at however resistant they're against numerous common obfuscation techniques (even with identified malware). Such an evaluation ensure out there defense against mobile malware threats, however conjointly proposing effective, next-generation solutions. We tend to use DroidChameleon, a systematic framework with numerous transformation techniques is used for enhancing and evaluating the attacks. The majority of the existing tools will is trivially defeated by applying slight transformation over identified malware with very little effort for malware developers. Finally, in light-weight of our results, we tend to propose potential remedies for up this state of malware detection on mobile devices.

Index Terms— Android App security, Antimalware, Transformation attacks

1. INTRODUCTION:

Mobile computing devices like smartphones and tablets are getting progressively common. This quality attracts malware developers too. In reality, mobile malware has already become a heavy concern. It has been according that one in common on android, one in all the foremost common smartphone platforms [2], malware has perpetually been on the rise and therefore the platform is seen as "clearly today's target" [3], [4]. With the expansion of malware, the platform has conjointly seen Associate in evolution of anti-malware tools, with a variety of free and paid offerings currently on the market within the official android app market, in Google Play. In this paper, we have estimated and enhanced the deductions of transformation of anti-malware tools on android with in the face of assorted evasion techniques.

2. BACKGROUND STUDY:

Android is an operating system for mobile devices like smartphones and tablets. It supports the Linux kernel and provides a middleware implementing subsystems like telecommunication, window management, communication between applications to hardware and so on. Applications area unit programmed primarily in Java although the programmer's area unit allowed to try and do native programming via JNI(Java native interface). Rather than running Java bytecode, Android runs Dalvik byte code, which is created from Javabyte code. In Dalvik, rather than having multiple .class files as within the case of Java, all the categories area unit packed along in a very single .dex file.

Android applications area unit fabricated from four kinds of parts, namely activities, services, broadcast receivers, and content providers. These application parts area uniten forced as classes in application code and area unit declared within the Android-Manifest. The middleware interacts with the app through these parts. Android application packages area unit jar files containing the application bytecode as a categories.dex file, any native code libraries, application resources like pictures, .config files and so on, and a manifest, referred to as Android Manifest.

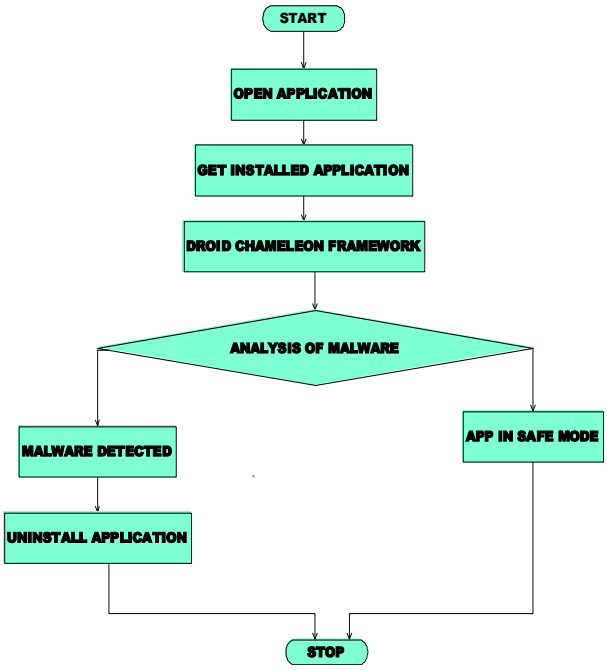
It's a binary XML file that declares the appliance package name, a string that's imagined to be distinctive to an application, and the different parts within the application. It additionally declares other things (such as application permissions) that aren't therefore relevant to the current work. The AndroidManifest is written in XML and is remodeled to binary XML during application build. Only digitally signed applications could also be put in on An Android device. Linguistic communication keys area unit sometimes in hand by individual developers and not by a central authority, and there's no chain of trust. All third party applications run unprivileged on Android.

TRADITIONAL ATTACK METHODS:

We tend to judge the effectiveness of anti-malware tools on mechanical man within the face of varied evasion techniques. For example, polymorphism is employed to evade detection tools by transforming a malware in several forms ("morphs") however with the same code. Metamorphism is another common technique that can

change code so it now not remains identical but still has identical behavior. For easy presentation, we use the term polymorphism, in this paper to represent both obfuscation techniques. Additionally, we tend to use the term 'transformation' loosely, to confer with numerous polymorphic or metamorphic changes. Polymorphic attacks have long been an outbreak for ancient desktop and server systems. Whereas there exist earlier studies on the effectiveness of anti-malware tools on PCs [5], our domain of study is totally different in this we tend to completely specialize in mobile devices like smartphones, which need alternative ways for anti-malware style. Also, malware on mobile devices have recently escalated their evolution however the capabilities of existing anti-malware tools square measure for the most part not nevertheless understood. In the in the meantime, straight forward varieties of polymorphic attacks have already been seen within the wild [5].

SYSTEM ARCHITECTURE:



SYSTEM FRAMEWORK:

This system framework is designed by using two types of transformation attacks.

1. Trival Transformation
2. Transformation Attacks Detectible by static analysis

TRIVAL TRANSFORMATION

Trivial transformations do not require code-level changes.

We have the following transformations in this category.

1) Repacking
2) Disassembling and Reassembling
3) Changing Package Name

1) **REPACKING:** The packages used in android signed jar files. This is also unzipped with the regular utilities and then repacked once more with tools offered within the android SDK. Once applications are repacked, it is signed with custom keys. Detection signatures that match the developer keys or a check ofthe entire application package rendered ineffective by this transformation.

2) **DISASSEMBLING AND REASSEMBLING:** The compiled Dalvik bytecode in categories.dex of the applying package could be disassembled and so reassembled back once more. The varied items (classes, methods, strings, then on) during a dex file could be organized or described in additional than a way and therefore a compiled program is also described in several forms. Signatures that match the complete categories. Dex are crushed by this transformation.

3) **CHANGING PACKAGE NAME:** Each application is known by a package name distinctive to the applying. This name is defined within the package's Android Manifest. We modify the package name during a given malicious application to a different name.

In Android Manifest.xml
package="com.example.antimalware"

TRANSFORMATION ATTACKS DETECTIBLE BY STATIC ANALYSIS:

The application of DSA transformations doesn't break all types of static analysis. Specifically, styles of analysis that describe the linguistics, like data flows still attainable. Only easier checks like string matching or matching API calls is also blocked. We have the following transformations in this category:

1. Identifier Renaming:
2. Data Encoding
3. Call Indirections
4. Code Reordering:
5. Encrypting Payloads
6. Junk Code Insertions

1. IDENTIFIER RENAMING:

Most class, method, and field identifiers in bytecode are often renamed. We tend to note that many free obfuscation tools like Pro Guard give symbol renaming.

Listing two presents associate example transformation forcode in listing one.

R.java File

```
publicstaticfinalclass string {
publicstaticfinalintaction_settings=0x7f050001;
publicstaticfinalintapp_name=0x7f050000;
publicstaticfinalintevaluation=0x7f050003;
publicstaticfinalinthello_world=0x7f050002;
publicstaticfinalintinstallation=0x7f050004;
publicstaticfinalinttitle_activity_install=0x7f050005;
publicstaticfinalinttitle_activity_list_controls=0x7f050006;
publicstaticfinalinttitle_activity_permissions=0x7f050007;
   }
```

2.DATA ENCODING

Here the byte code conversion process done. The main reason for changing in to byte code conversion it helps to process in Dalvik Virtual machine.

```
In String.xml
<string name="evaluation">Evaluation of Anti-Malware</string>
Layout.xml
android:text="@string/evaluation"
```

3. CALL INDIRECTIONS

Here r.java file will be convert r.class file which helps in the execution process.

```
In XML File
android:id="@+id/Installationfiles"
In Java File
Button  myBtnInstall=(Button)
findViewById(R.id.Installationfiles);
```

4. CODE REORDERING:

This code tests with the package name. If any of package names is not given properly or rechanged this will identify the error and the program will not be executed.

```
List<ApplicationInfo> packages =
myPackageManager.getInstalledApplications(Package
Manager.GET_META_DATA);
for (ApplicationInfoapplicationInfo : packages) {
try {
PackageInfopackageInfo =
myPackageManager.getPackageInfo(applicationInfo.pac
kageName, PackageManager.GET_PERMISSIONS);
            if(myPackageManager.getLaunchIntentForPac
kage(packageInfo.packageName)!=null)
                                    {
            myPackagname.add(packageInfo.packageNam
e);
myAppname.add((String)
packageInfo.applicationInfo.loadLabel(getPackageMana
ger()));                            }
                        }
            } catch (NameNotFoundException e) {
Toast.makeText(getApplicationContext(), "Exception
Found", Toast.LENGTH_LONG).show();
                                    }}
```

5. ENCRYPTING PAYLOADS

Each and every component ID will be available in r.java file. In the program component ID will be encrypted using FINDVIEWBYID function.

```
In Xml File
<Button
android:id="@+id/Installationfiles"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_centerHorizontal="true"
android:layout_below="@+id/TexttImage"
android:text="@string/installation"android:textAppeara
```

```
nce="?android:attr/textAppearanceMedium"
android:textStyle="bold"
android:textColor="@android:color/white"
    />
```

6. JUNK CODE INSERTIONS:

This process is done using

1. layout.xml
2. strings.xml
3. Style.xml
4. menu.xml all these components has unique codes in r.java file to be inserted.

```
publicvoidpermissions_risk()
        {
                int result;
                int pars=131/6;
                ArrayList<HashMap<String,
String>> data=myData.SelectData();

        for(inti=0;i<myPackagePermissions.size();i++)
{for(int j=0;j<data.size();j++)
                {
        if(myPackagePermissions.get(i).equals(data.ge
t(j).get("per")))
                {
        pars +=Integer.parseInt(data.get(j).get("ris"));
                        }}}
```

CONCLUSION:

We evaluated ten anti-malware products on Android for their resilience against malware transformations. To facilitate this, we developed DroidChameleon, a systematic framework with various transformation techniques. Our findings using transformations of six malware samples show that all the anti-malware products evaluated are susceptible to common evasion techniques and may succumb to even trivial transformations not involving code-level changes. Finally, we explored possible ways in which the current situation may be improved and next-generation solutions may be developed. As future work, we plan to perform a more comprehensive evaluation using a much larger number of malware samples and anti-malware tools.

REFERENCES:

1. V. Rastogi, Y. Chen, and X. Jiang, "DroidChameleon: Evaluating Android anti-malware against transformation attacks," in Proc. ACM
2. R. Komondoor and S. Horwitz, "Semantics-preserving procedure extraction, in Proc. 27th ACM SIGPLAN-SIGACT Symp. POPL, 2000,pp. 155–169.
3. C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in Proc. 18th Conf. USENIX Security Symp., 2009, pp. 351–366.
4. I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behaviorbased malware detection system for android," in Proc. 1st ACM Workshop Security Privacy Smartphones Mobile Devices, 2011.
5. V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: Automatic security analysis of smartphone applications," in Proc. ACM CODASPY, Feb. 2013, pp. 209–220.