## Creating malware application - MathSucks.apk

Research data is data that is collected, observed, or created, for purposes of analysis to produce original research results. The word “data” is used throughout this site to refer to research data. Research data can be generated for different purposes and through different processes, and can be divided into different categories. Each category may require a different type of data management plan.

* Observational: data captured in real-time, usually irreplaceable. For example, sensor data, survey data, sample data, neurological images.
* Experimental: data from lab equipment, often reproducible, but can be expensive. For example, gene sequences, chromatograms, toroid magnetic field data.
* Simulation: data generated from test models where model and metadata are more important than output data. For example, climate models, economic models.
* Derived or compiled: data is reproducible but expensive. For example, text and data mining, compiled database, 3D models.
* Reference or canonical: a (static or organic) conglomeration or collection of smaller (peer-reviewed) datasets, most probably published and curated. For example, gene sequence databanks, chemical structures, or spatial data portals.

I need to test my application before upgrade as well as after the upgrade. The plan is to creating test Malware applications which will be simulating the real world malware applications. I have created following Android malware files to test data.

| **#** | **Malware app name** | **Description** |
| --- | --- | --- |
| 1 | MathStuck | Unsigned application |
| 2 | MathStuck2 | Signed Application |
| 3 | MathStuck3 | Obfuscated Application |

**Testing Unsigned Application:** It is the simplest test to do because normally applications are not in the public without signing. There is no user identification for the application. Malware application may be exists in the world without signing up. So there is no identification about the application. I will use my MatchStuck application without signing initially. So DroidBox should analysis the application and provide the results.

**Testing Signed Application:** Android requires that all apps be digitally signed with a certificate before they can be installed. Android uses this certificate to identify the author of an app, and the certificate does not need to be signed by a certificate authority. Android apps often use self-signed certificates. The app developer holds the certificate's private key.

You can sign an app in debug or release mode. You sign your app in debug mode during development and in release mode when you are ready to distribute your app. The Android SDK generates a certificate to sign apps in debug mode. To sign apps in release mode, you need to generate your own certificate.

#### **Signing Considerations**

You should sign all of your apps with the same certificate throughout the expected lifespan of your applications. There are several reasons why you should do so:

* App upgrade: When the system is installing an update to an app, it compares the certificate(s) in the new version with those in the existing version. The system allows the update if the certificates match. If you sign the new version with a different certificate, you must assign a different package name to the application—in this case, the user installs the new version as a completely new application.
* App modularity: Android allows apps signed by the same certificate to run in the same process, if the applications so requests, so that the system treats them as a single application. In this way you can deploy your app in modules, and users can update each of the modules independently.
* Code/data sharing through permissions: Android provides signature-based permissions enforcement, so that an app can expose functionality to another app that is signed with a specified certificate. By signing multiple apps with the same certificate and using signature-based permissions checks, your apps can share code and data in a secure manner.

If you plan to support upgrades for an app, ensure that your key has a validity period that exceeds the expected lifespan of that app. A validity period of 25 years or more is recommended. When your key's validity period expires, users will no longer be able to seamlessly upgrade to new versions of your application.

In my testing I should use the signed application to test the DroidBox.

**Obfuscated Application**: Code shrinking is available with **ProGuard**, which detects and removes unused classes, fields, methods, and attributes from your packaged app, including those from included code libraries (making it a valuable tool for working around the 64k reference limit). ProGuard also optimizes the bytecode, removes unused code instructions, and obfuscates the remaining classes, fields, and methods with short names. The obfuscated code makes your APK difficult to reverse engineer, which is especially valuable when your app uses security-sensitive features, such as licensing verification.

Resource shrinking is available with the Android Plugin for Gradle, which removes unused resources from your packaged app, including unused resources in code libraries. It works in conjunction with code shrinking such that once unused code has been removed, any resources no longer referenced can be safely removed as well.

Malware application is a simple calculator which runs on Android OS. It should be run on both Android 4.0 emulator as well as the latest Android versions. It will work as a calculator but in the meantime it will access users contact details and push to an external endpoint without the user consent.

### Layout Design

I have used Android SDK tool to design the MathSucks layout. It is an simple layout which has two input fields, button and one output field. In the layout >> content\_main.xml file I have added (drag and dropped) these fields and buttons. Giving specific id for each and every field it is necessary to access these objects in the MainActivity.java classs.

### MainActivity Class

MainActivit.java class has the implementation. Where I have used “TextView” data type to define the each text field.

TextView **totalTextView**;

TextView **percentageTextView**;

TextView numberTextView;

onCreate() has being overridden to implement the calculator functionality and introduced an button also.

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);

setSupportActionBar(toolbar);

totalTextView = (TextView) findViewById(R.id.totalTxtView);

percentageTextView = (TextView) findViewById(R.id.percentageTxtView);

numberTextView = (TextView) findViewById(R.id.numberTxtView);

Button calckBtn = (Button) findViewById(R.id.calcBtnView);

calckBtn.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

float percentageValue = Float.parseFloat(percentageTextView.getText().toString());

float dec = percentageValue / 100;

float numberValue = Float.parseFloat(numberTextView.getText().toString());

float totalValue = numberValue \* dec;

totalTextView.setText(Float.toString(totalValue));

}

});

}

