

ReFrameworker: The Android runtime manipulator



Erez Metula ,Application Security Expert
AppSec Labs (Founder)

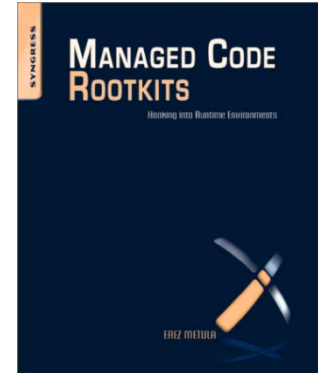
ErezMetula@AppSec-Labs.com

Agenda

- Introduction to AppUse - Android Application Pentest VM
- ReFrameworker - android runtime manipulator
 - How it works
 - Deep dive – placing hooks inside the dalvik runtime
 - The ReFrameworker dashboard
 - Pentesting with ease by app manipulation
- DEMOS

About me

- Founder of AppSec Labs
- Application security expert
- Book author
 - Managed Code Rootkits (Syngress)
- Speaker & Trainer
 - BlackHat, Defcon, RSA, OWASP, etc..



Android is everywhere

- It is not just about smart phones anymore..
 - Home appliances, Watches, Laptops , Smart TVs, Smart DECT home phones, Cars, Cameras, HAS (Home Automation Systems), Smart glasses, Game consoles



Samsung
RF4289HARS
refrigerator



NordicTrack
Elite 9500 Pro
treadmill



Even animals use Android..

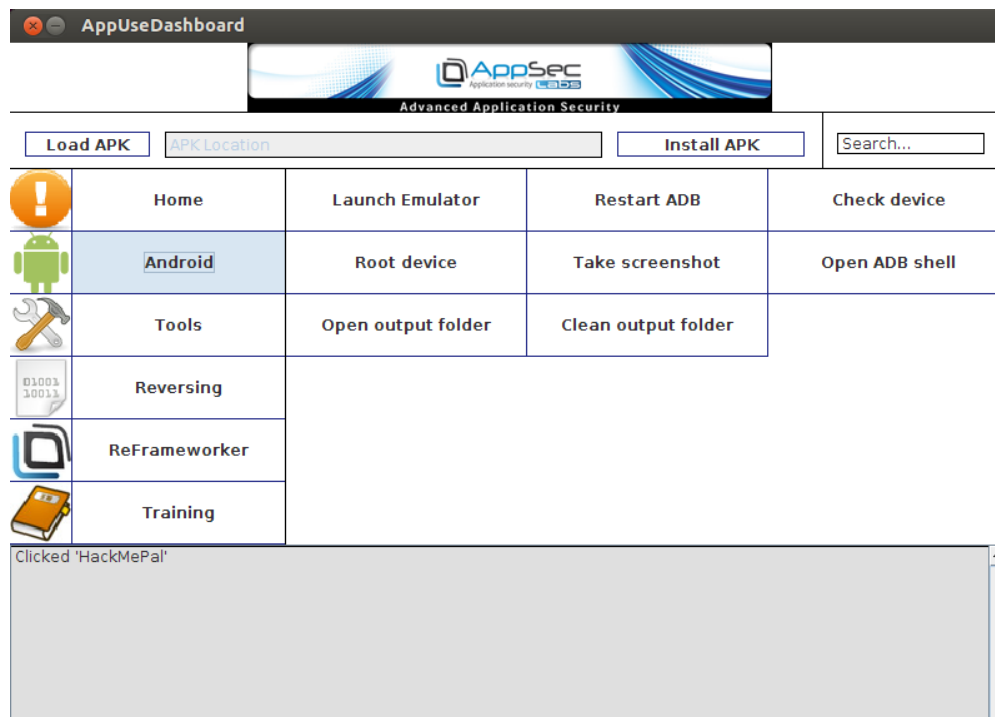
AppUse OS



- AppUse is a linux machine built by AppSec Labs containing Android SDK, emulator and custom ROM to fit penetration testers
 - custom ROM to ease with testing
 - preinstalled crypto keys to work with Burp
 - Runtime modifications to observe and change applications behavior
 - Lots of tools
 - Vulnerable target apps

AppUse Dashboard

- Command & Control center
- Cooperation of tools to meet common goals
- Multiple tasks in a click of a button



Traditional android PT involves

- Either your target is to attack the client or the server side app, many times you find yourself wasting time on tasks that get in your way to do the real work
 - Certificate checks that blows away your proxy
 - App logic is based on device identifier such as phone number, IMEI, etc.
 - App is restricted to wifi only (common with voip apps)
 - Figuring out where and what files the app is storing
 - Figuring out which tables the app is querying
 - etc

Traditional reversing & patching

- Reversing/patching is **a tool, not a target**
- Heavy artillery to solve problems that are not part of the main target
- There are easier ways to avoid such roadblocks
- **DEMO – HackMePal IMEI restriction**



We are wasting too much time
on unimportant tasks that put
us out of focus !

The challenge – what if...

Same call you need to patch appears 1000 times?

The code base is too big to even locate what you need to patch?

The app doesn't decompile?

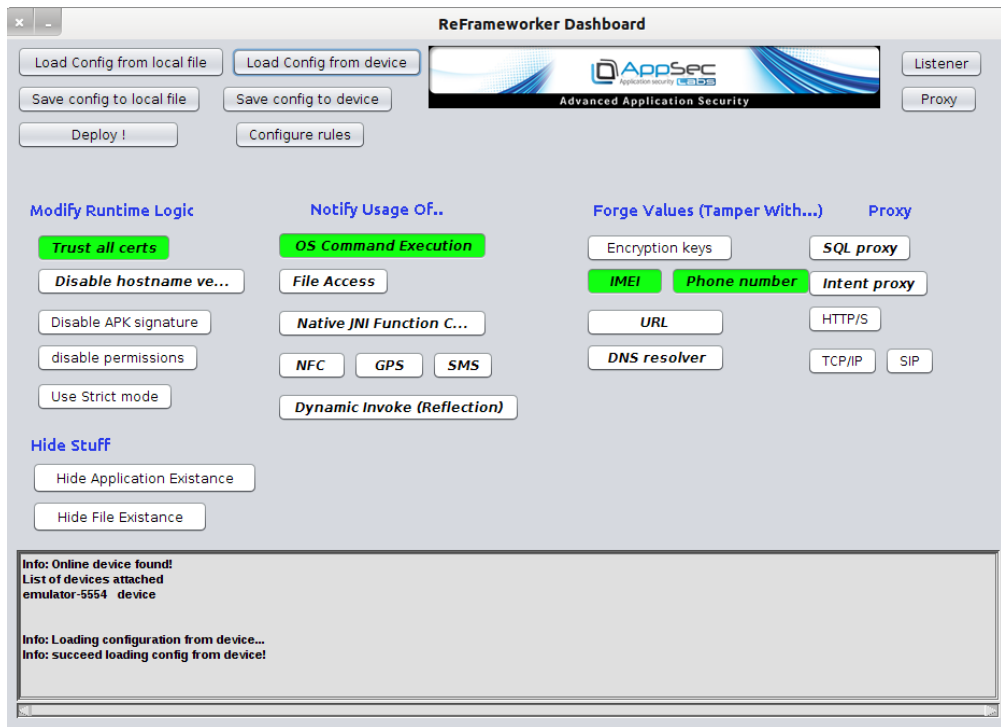
The code is obfuscated?

Taking a different approach

- Changing the app behavior by manipulating with its runtime
- BEST PART – no need to modify the application code !
- Inspired from the research I did on my book “managed code rootkits”

The ReFrameworker platform

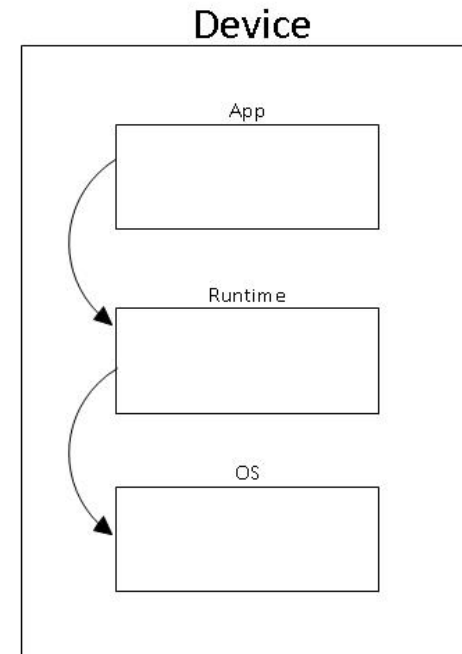
- Runtime manipulation framework by AppSec Labs
- Integrated as part of AppUse
- Released at BlackHat USA 2013



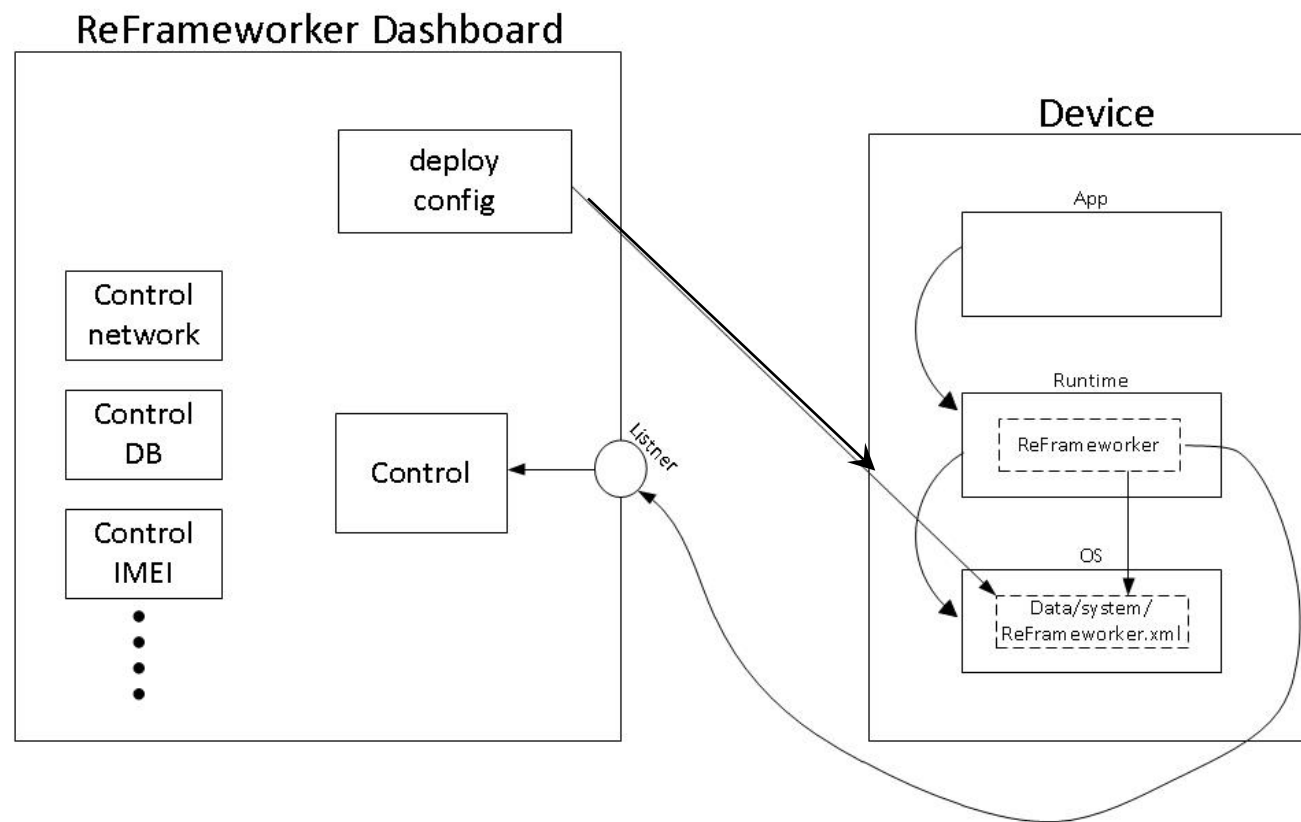
Overview - How it works

- The Android runtime was compiled with many hooks placed into key places inside its code.
- The hooks look for a file called "Reframeworker.xml", located inside /data/system.
- So each time an application is executed, whenever a hooked runtime method is called, it loads the ReFrameworker configuration along with the contained rules ("items") and acts accordingly.

Overview - without ReFrameworker

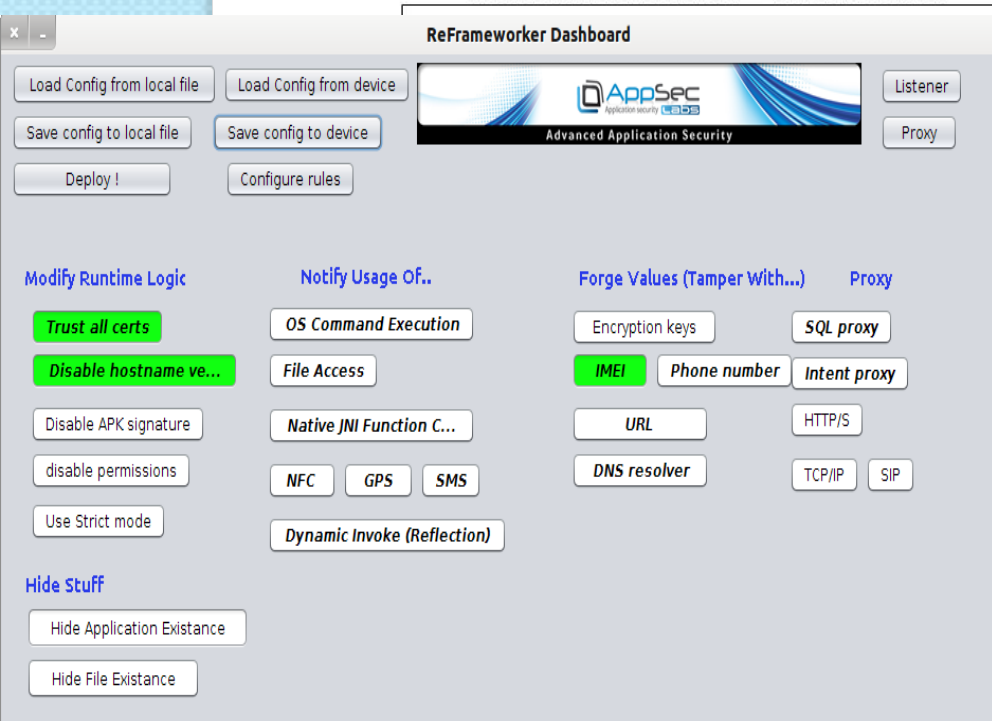


Overview - with ReFrameworker



Overview - with ReFrameworker

ReFrameworker Dashboard



Device



ReFrameworker – deep dive

- New namespace called AppSecLabs/ReFrameworker injected into ANDROID_SRC/libcore/luni/src/main/java
- Major classes:
 - ConfigManager – hook manager
 - Controller – handles calls by hooks
 - GeneralItem – general config
 - Item – defines a specific hook

The hooks

- The AppUse environment was compiled with lots of hooks at some key places.
- As part of the research, after finding out interesting places we want to control such as handling of files, communication, encryption, etc. we placed calls at those location to the ReFrameworker controller.
- The controller's responsibility is the check whether a rule is currently defined for this particular location, and if so it acts by its configuration.

Using the controller class

- Key design decisions
 - Placing a hook should be as clean as possible
 - Injecting hook in smali should be easy (i.e. few dependencies, just a “copy me” block)
 - The hook can wrap a specific value and control it
 - The runtime can work transparently whether the hook is enabled or not
 - The runtime should be affected on the fly
- example

```
private int executeSql(String sql, Object[] bindArgs) throws SQLException {  
    //added  
    sql = controller.operateString(sql);  
}
```

Example – hooking into SQLiteDatabase.execSQL

- all queries are passed through at.
- Hooking into this class will enable us to intercept all the local SQL queries sent from the application to its local DB.
- Our hook (which was placed inside the Android execSql method inside the SQLiteDatabase class) will intercept this value and do whatever was instructed at the configuration.

Implementation (decompiled android runtime source)

```
private int executeSql(String sql, Object[] bindArgs) throws SQLException {  
    //added  
    sql = controller.operateString(sql);
```

```
    acquireReference();  
    try {  
        if (DatabaseUtils.getSqlStatementType(sql) == DatabaseUtils.STATEMENT_ATTACH) {  
            boolean disableWal = false;  
            synchronized (mLock) {  
                if (!mHasAttachedDbsLocked) {  
                    mHasAttachedDbsLocked = true;  
                    disableWal = true;  
                }  
            }  
            if (disableWal) {  
                disableWriteAheadLogging();  
            }  
        }  
  
        SQLiteStatement statement = new SQLiteStatement(this, sql, bindArgs);  
        try {  
            return statement.executeUpdateDelete();  
        } finally {  
            statement.close();  
        }  
    } finally {  
        releaseReference();  
    }  
}
```

Injecting the hooks

- Hooks are usually placed around an important value, such that if a rule is define for this particular hook, then the controller's responsibility will be to do something with it.
- The controller can either
 - **do nothing** and leave that value as is (in case no rule is defined or the rule is disabled)
 - it can **send that data** to a remote location
 - it can allow the user to **break and modify that value at real time** (i.e in a similar manner as a proxy)
 - it can do an **automatic replace** for another value.

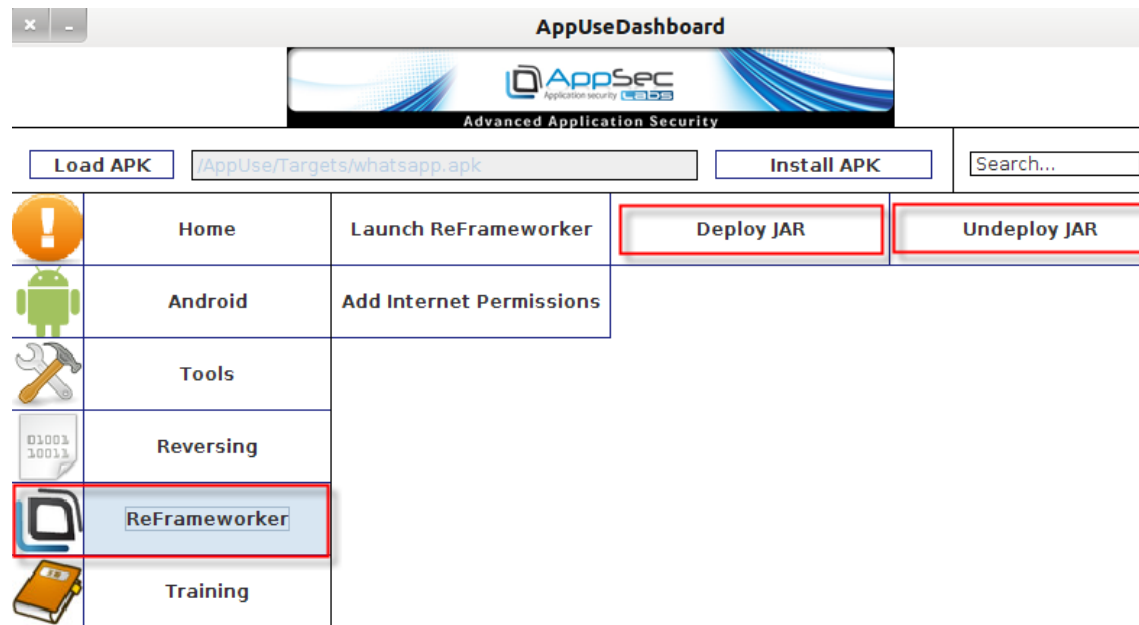
Replacing the runtime jar files

- After creating your own jars, it's time to deploy them
- Snippet from /AppUse/Android/ReFrameworker/runtime/deploy.sh:

```
cd /AppUse/Android/ReFrameworker/runtime
adb shell "adb remount"
adb shell "stop"
adb push 4.2.2/modified/core.jar /system/framework/
adb push 4.2.2/modified/framework.jar /system/framework/
adb shell "rm /data/dalvik-cache/*"
adb shell "start"
```


How to deploy ReFrameworker using the AppUse dashboard

- Replacing the original device jars with our modified version
- AppUse dashboard provides an easy to use deployer/undepolyer



The ReFrameworker dashboard

Managing the configuration file hooking rules



The ReFrameworker dashboard

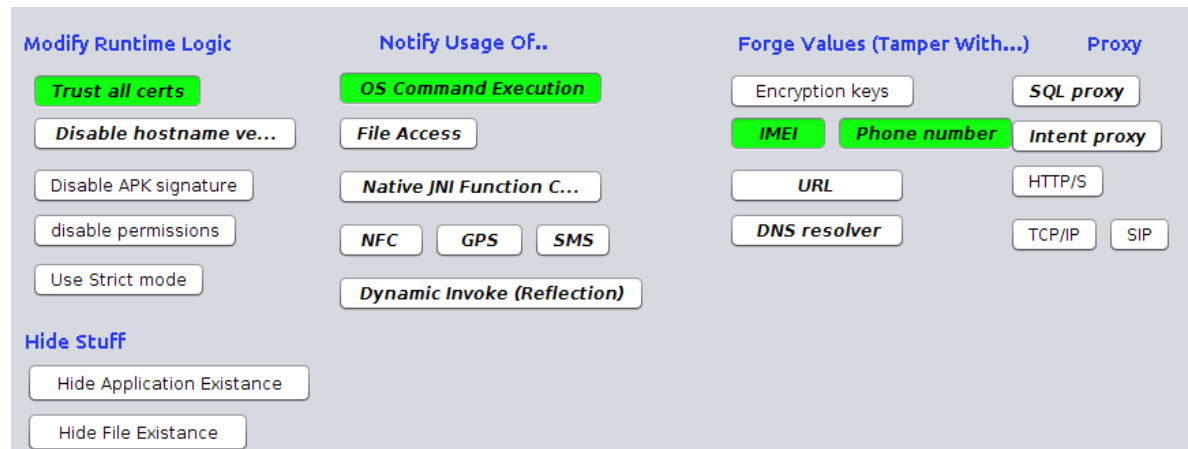
- Using the dashboard, you can define a set of rules that the Android runtime will obey.
- The dashboard will then generate a config file which the runtime will later parse and act accordingly.

Demo – changing the behavior of any app on-the-fly

Loading the config file

- it starts with loading a config file (Reframeworker.xml)
- the dashboard will immediately mark all the loaded rules and allow the user to enable and configure them.

After the file is loaded, the dashboard marks all the defined rules with **bold**, and highlights all rules which are also enabled as **green**.



The dashboard listener

- Since the device might communicate with the dashboard (sending some data, waiting for instructions, etc.) , the dashboard contains a listener for incoming communication established from the device. Therefore, the dashboard contains a button for the listener

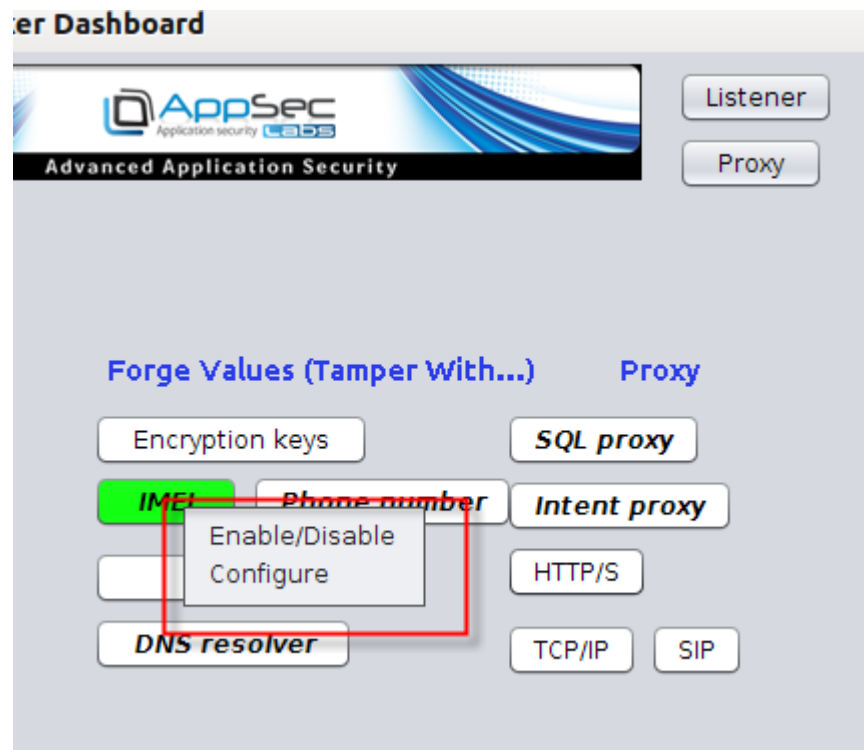


Configuring the rules (“items”)

- User defines required behavior
 - can turn on sniffing of important information
 - bypass of certain logic
 - doing some string replacement
 - sending some data to the ReFrameworker dashboard
 - Etc.

Item configuration

- Configuring the behavior of each rule can be achieved by clicking on the rule's item, and selecting "configure"

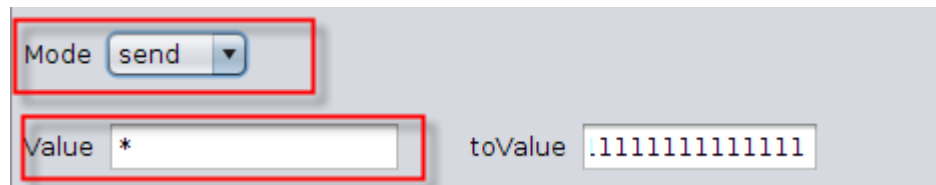


Item rule properties

- **Name** – the name of the rule
- **Enabled** – is it enabled?
- **Affected application** - condition for the hooked application. An asterisk (*) means always.
- **Calling method** – the name of the runtime method upon which this rule should apply
- **Mode** – can have 3 possible values – Send, Proxy, or Modify.
 - Send – send the hooked content to the ReFrameworker dashboard
 - Proxy - let the user control the value of the hooked content by using a proxy-like UI
 - Modify – replace a particular content with another content
- **Value** – specify the condition for the hooked content. An asterisk (*) means always.
- **toValue** - specify the action for the hooked content. An asterisk (*) means always

Send mode

- send the hooked content to the ReFrameworker dashboard
- Requires the listener to be up
- The inspected value should match the value of the defined item
- You can use * as ANY (i.e. the hooked value will be sent always)
- The toValue is ignored (not in use)

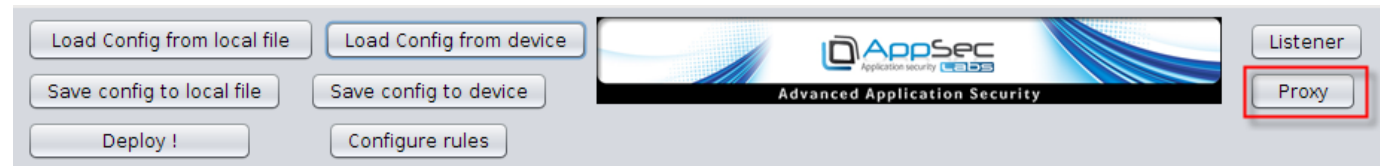


The screenshot shows a configuration window with two red rectangular highlights. The first highlight is around the 'Mode' dropdown menu, which is set to 'send'. The second highlight is around the 'Value' text input field, which contains an asterisk (*). To the right of the 'Value' field is a 'toValue' text input field containing a string of 12 ones (111111111111).

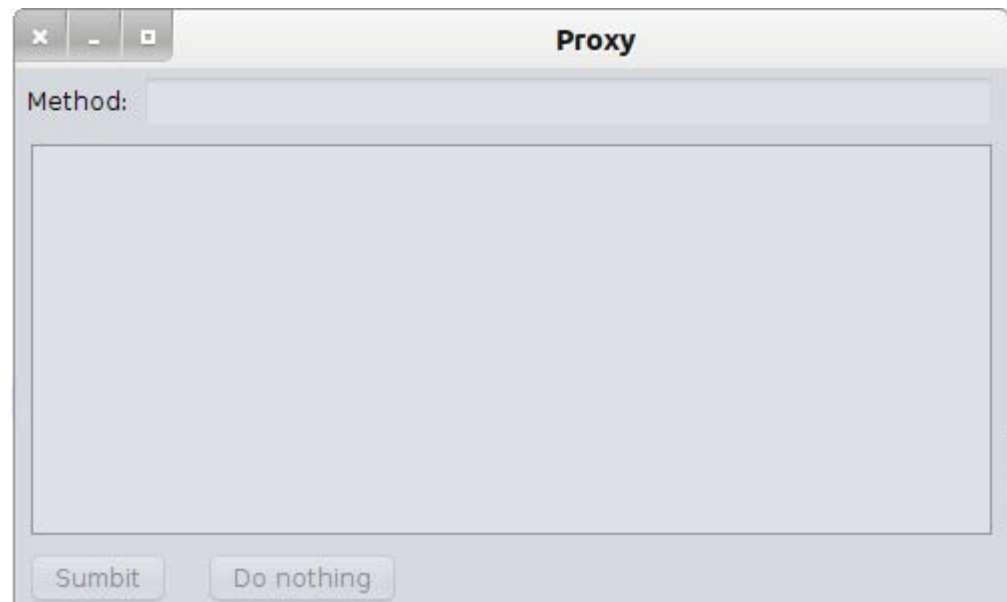
Proxy mode

- Now each time the hooked method is called, the device will send this data to the proxy, and will replace the original value with modified received value.

Start the proxy

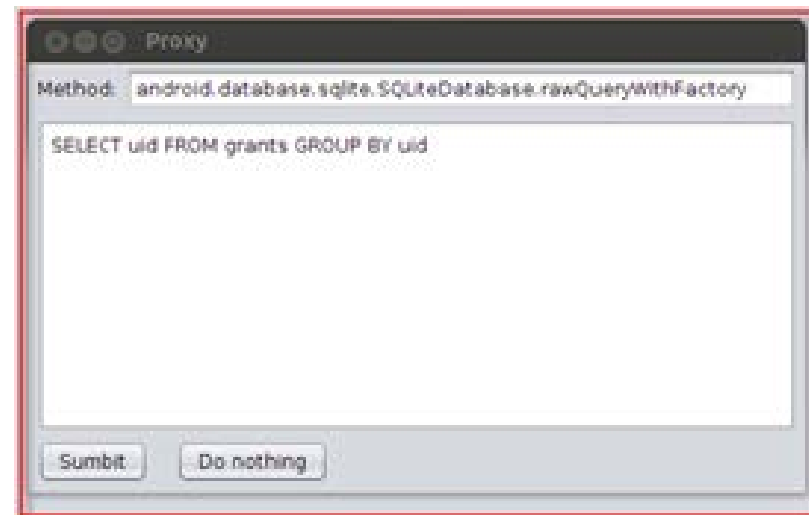


The proxy window




Intercepting data with the proxy

- When a message will be received, the proxy will wake up and give the user the opportunity to observe the message **AND** modify it – while the android app is waiting for the response



Modify mode

- replace a particular content with another content
- The inspected value should match the value of the defined item
- The toValue contains the new value to be set
- You can use * as ANY (i.e. the hooked value will be sent always)



Mode modify ▾

Value

toValue

DEMO – retake on HackMePal IMEI restriction

- This time, without touching the application code !!!

Summary

- AppUse is the Pentester's best friend when looking for Android app vulnerabilities
 - Contains a customized emulator
 - Has an easy to use dashboard
 - Loaded with tools, vulnerable apps, etc.
- ReFramrworker is your hooking platform – pentest the app from the inside out

Questions?

Thank you !

- Get the Latest AppUse from here:

<https://www.appsec-labs.com/AppUse>



WE ARE HIRING !