AppSec Labs Ltd.
info@appsec-labs.com
https://appsec-labs.com

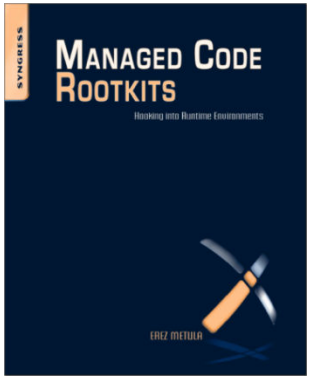
Dynamic Analysis of Android Apps

OWASP IL 2014

Erez Metula , Application Security Expert
AppSec Labs (Founder)
ErezMetula@AppSec-Labs.com

About me

- Founder of AppSec Labs
- Application security expert
- Book author
  - Managed Code Rootkits (Syngress)
- Speaker & Trainer
  - Presented at BlackHat, Defcon, RSA, OWASP USA, OWASP IL, etc..
  - Secure Coding / Hacking trainer
- Speaking for the 8th time in a row at OWASP IL ☺



AppSec Labs

The leading Application Security Company

- A bunch of Application Security Experts
  - Ninja Pentesters of Web & Mobile Apps
  - Elite Trainers for Hacking & Secure coding courses

AppSec Labs
Learning Management System

HP Software Security Training Center

Online courses and exercises for secure coding

Login

- Home
- KBase
- Software Security Q&A
- Courses
- Tour

Welcome to the **Software Security Training Center**!

The **Software Security Training Center** (SSTC) was created with the intent to enhance your awareness in application security and secure coding.

By understanding the threat landscape and implementing the appropriate countermeasures we are improving the integrity of our products.

Awareness is the name of the game!

SSTC is now available for you to enhance your knowledge of software security and improve our products' security.

Check out your personal training program and download course materials and lab environment to practice what you've learned under Courses in the menu.
Check out our easy-to-use knowledgebase for application security issues you are confronting in real time.

We wish you good luck in revealing the fascinating world of software security

Click here to take a tour and explore the system's features

HP SW IT Management Security & Trust Office

Contact

Questions, comments, or changes? Tell us: ITM.Securitytrust@hp.com

WTF ?!

A world without mobile technology ?

Agenda

- Why dynamic analysis?
- Memory dumps and analysis
- Smali debugging
- Setting breakpoints
- Native debugging with IDA (building signatures, types etc.)
- Runtime instrumentation and manipulation using ReFrameworker

Why dynamic analysis?

- Pentesing the app “from the inside”

Some examples – real world scenarios encountered in the wild

- Requests to the server side are encrypted , signed, or just cannot be MiTMed for some reason
  - Your proxy is useless.
- Dynamic values stored in memory - created while the app runs, received from network, etc.
  - Decompiling is useless. The value is not in the code
- Strings are obfuscated
  - Decompiling is hard
- The app is using some hard coded values such as URLS, encryption keys
  - Patching is time consuming

Example – requests with signed data

- Cannot manipulate with requests since they are signed

```
POST /GreatBank/TransferMoney.aspx HTTP/1.1
Host: GreatBank.com
Proxy-Connection: keep-alive
Content-Length: 274
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://GreatBank.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate,sdch
Accept-Language: he-IL,he;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: SessionId=rqpyxteu0d4a50ndzsfgfqrx

SourceAcc=111111111111&DestinationAcc=22222222222&Amount=10000&Signature=MjNiYjMONXBvM3BvajRpam8=
```

signature

Example – requests with encrypted data

- Cannot view/manipulate with requests since they contain encrypted data

```
POST /GreatBank/SensitiveOperation.aspx HTTP/1.1
Host: GreatBank.com
Proxy-Connection: keep-alive
Content-Length: 274
Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Origin: http://GreatBank.com
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip,deflate,sdch
Accept-Language: he-IL,he;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: SessionId=rqpyxteu0d4a00hdzsfgfqrx
```

encrypted!


```
SuperSensitiveData=KCkqNTY3ODk6QORFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R1dnd4eXrp4uTg5efq6+jG9P
by+/n/ltzz+vHRqrq/EKyTAiRhYlZVY1EsHAA8X19aVGllWVhSU2tqGAyTwKPDtcSmmKllZCAh9OiwGbegFCARCCB
gAxcDAREgAhEBAxEBoAwgDAMBIAIRAxEgPyAgPWccURWnVm26WHgTLEgpPbxWJlnAxBLzNkVh/AsPGWjkHMQZXgZd
agjcJ+Apk2E6H/wkC9YnJOixU7VTKARRVKXWIjY7xFPAGBlOSMAlahS7IWFcEkA/3w9hPKwy9Czc5HBbKGV2R+wbZ
mV3VkPmyTlToxouFPRVZDSIG2tb5LBt3/QcZmqTeZBK9txpBH5YiAtuCFlJbgEH7pPvkTcOuwWwcnsl9FJhTvxlc2
NS7cRTGVQgACsGajIYIITHRnBGaFcCPW8sPhSTklBW5CAGUaVTG3V9OiIM7GjJICQB+zgequ+RIVg8L9/xW3sdKzQ
+HVdj6CXcVdwMKFVR7wUOEWJJKH9gD/nGOPMeCGRlXQlibSOhxQ==
```

Example – obfuscated code/values

- Cannot read values from decompiled code since it is obfuscated

```
import com.whatsapp.App;

public class e
{
  public static boolean g;
  private static final String[] z;
  public String a = "";
  public String b = "";
  public List<String> c = new ArrayList();
  public byte[] d;
  public Set<String> e = new HashSet();
  public Set<String> f = new HashSet();

  static
  {
    String[] arrayOfString = new String[7];
    char[] arrayOfChar1 = "*k|/Lv\035m1Vc\024n$Wc8,.J|.6}".toCharArray();
    int i = arrayOfChar1.length;
    int j = 0;
    char[] arrayOfChar2;
```

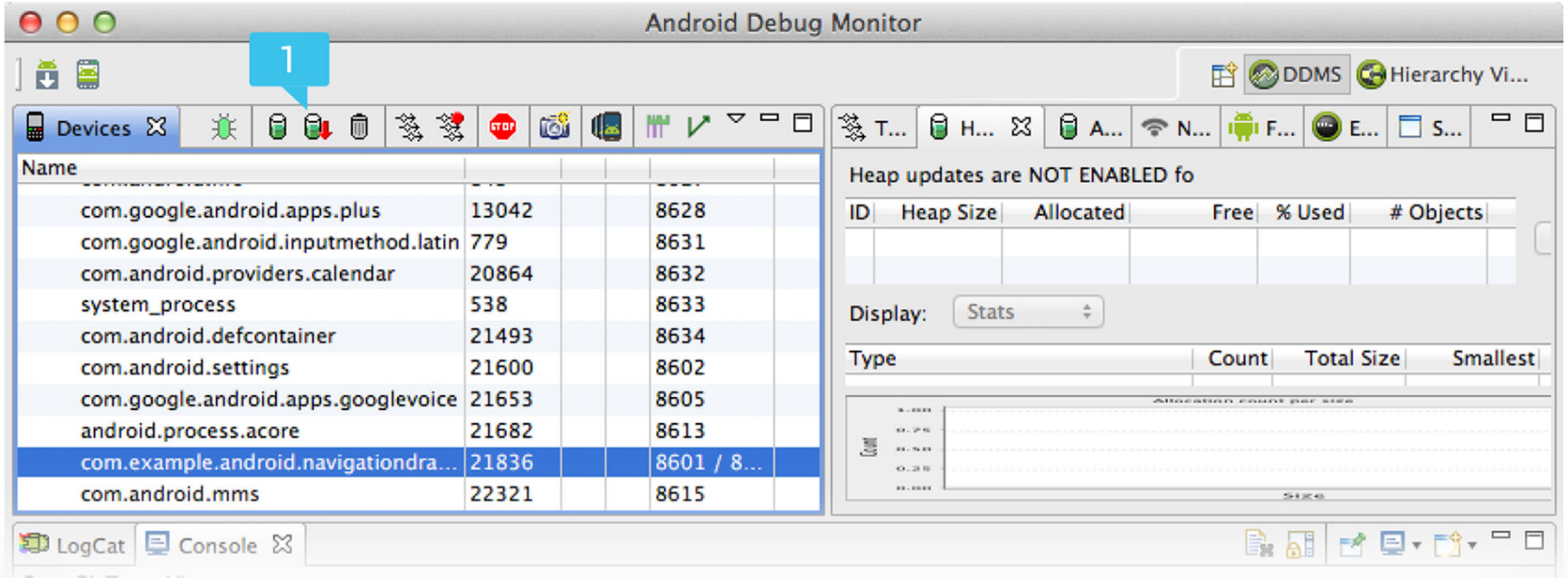
What to do?

- We must “work from the inside”
- Let’s start with direct memory analysis

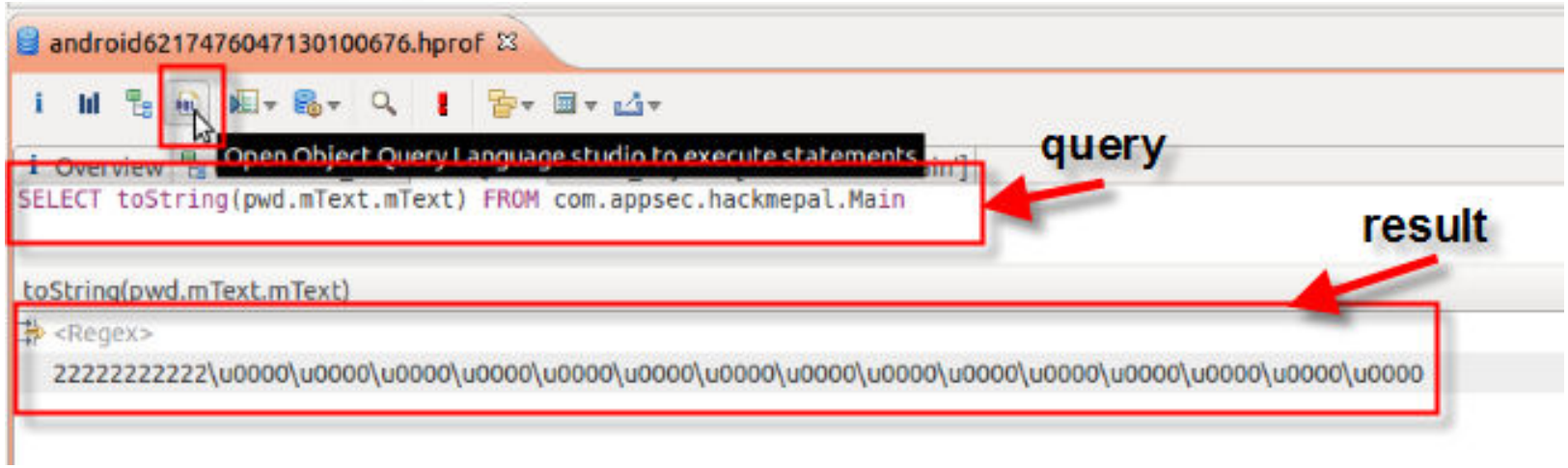
- Exposure of
  - Code sections
  - Sensitive data – application data, passwords, encryption keys, network traffic, calculations, etc.
  - Interactions with OS – files, processes, etc.

Memory Analysis

- Eclipse's MAT (Memory Analyzer Tool)
- Dump the application's current memory to disk
- Go to the "DDMS" Perspective, select the app and click "Dump HPROF file"



Query



DEMO – Memory Analysis

- Exposing obfuscated encryption key from memory

Debugging

- Debugging allows us to analyze the app in real time
  - Setting breakpoints
  - Bypassing restrictions
  - Jump into specific code sections
  - Expose secrets from memory

Debugging With Source

- Debugging with the source is easy
- Just load the project in eclipse
- Place your breakpoint
- And click debug

Debugging Without Source ("smali debugging")

- Most often you will not have the source
- Extracting the java code using dex2jar and creating an eclipse project is a bit tricky
  - Rebuilding the project dependencies
  - Decompiled code not always recompiles
- Alternatively, we can **remote debug smali code**

Major Steps

- Decode apk in debug (-d) mode:
  - apktool d -d app.apk out
- Make it debuggable at the AndroidManifest.xml <application> tag
  - <application .... **android:debuggable="true“** ...>
- Build new apk in debug (-d) mode:
  - apktool b -d out
- Sign, install and run new apk
  - signapk input.apk

Major Steps - Continued

- create Netbeans project
  - add new Java Project with Existing Sources, select "out" directory as project root and "smali" subdirectory as sources dir.
- Find application port using DDMS
  - it should be something like "86xx / 8700".
- Attached debugger in Netbeans
  - Debug -> Attach Debugger -> select JPDA and set Port to 8700 (or whatever you saw in previous step).
- Set breakpoint.

- NOTE – Officially, not all versions works, you need to use: netbeans 6.8 and apktool 1.4.1
- Currently, you can also use NetBeans 7.2 with Apktool v2.0.0-Beta9

DEMO

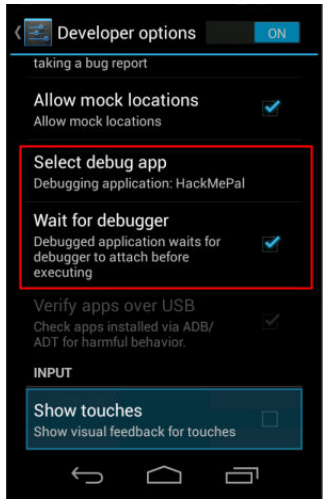
- Smali debugging

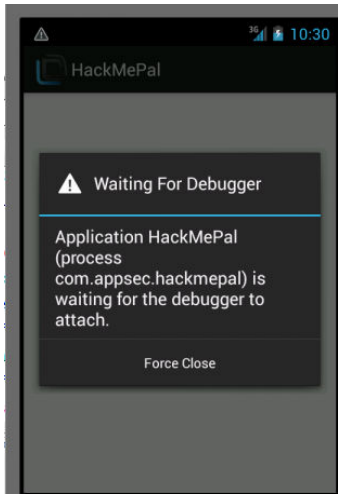
Tip - Wait for Debugger

- Programmatically – by calling
  android.os.Debug.waitForDebugger()

  or

- boolean debuggerAttached = false; while(!debuggerAttached) { ; }
- Another option – DEV tools



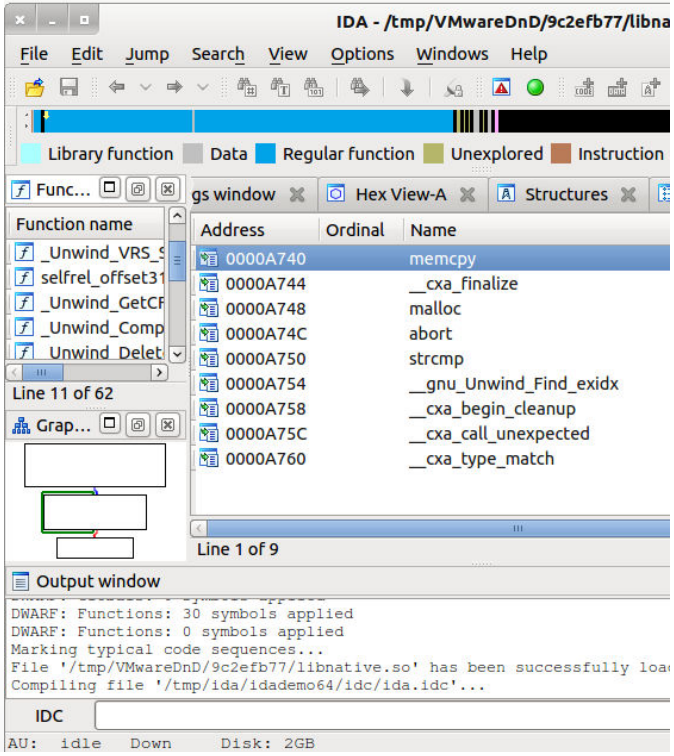


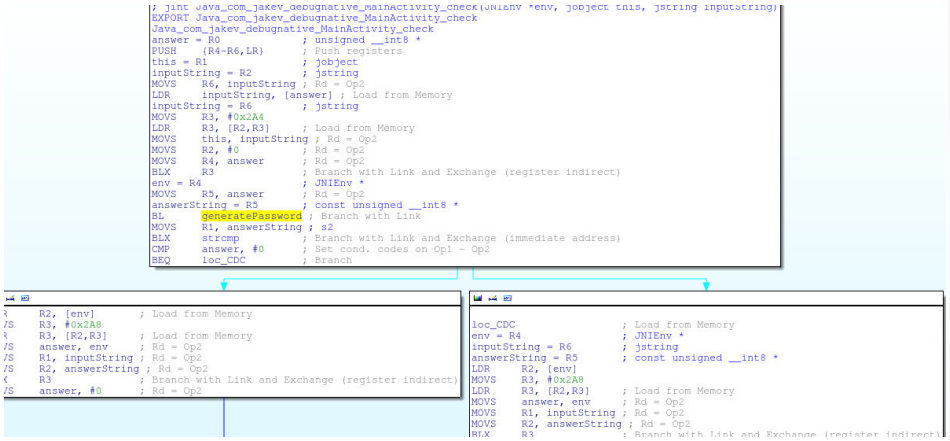
JNI Debugging

- Suppose our target code is inside native .so files.
- We can use IDA to analyze it, and GDB to remotely debug it

Using IDA

- You can use existing static binary analysis (such IDA) to better understand the code
- It will give you the idea where to start, where to set breakpoints, etc.





JNI Debugging - Main Steps

- Find the process id, attach to it and create a listener port inside the device. Then remotely debug the app

ps

```
130|root@android:/ # ps
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     296    208    c0098770 0000e840 S /init
root      2     0     0      0      c005048c 00000000 S kthreadd
root      3     2     0      0      c0042268 00000000 S ksoftirqd/0
root      4     2     0      0      c004ce30 00000000 S events/0
root      5     2     0      0      c004ce30 00000000 S khelper
root      6     2     0      0      c004ce30 00000000 S suspend
```

gdbserver :5050 --attach 1234 **//pid=1234, port=5050**

```
root@android:/ # gdbserver :5050 --attach 1768
Attached; pid = 1768
Listening on port 5050
```

adb forward tcp:5050 tcp:5050

ndk-gdb

target remote :5050

```
(gdb) target remote :5050
Remote debugging using :5050
warning: .dynamic section for "libc.so" is not at the expected
brary or version mismatch?)
warning: Could not load shared library symbols for 73 libraries
n/linker.
Use the "info sharedlibrary" command to see the complete listir
Do you need "set solib-search-path" or "set sysroot"?
warning: Unable to find dynamic linker breakpoint function.
GDB will retry eventurally.  Meanwhile, it is likely
that GDB is unable to debug shared library initializers
or resolve pending breakpoints after dlopen().
0x40037ebc in vsyslog_r () from libc.so
(gdb)
```

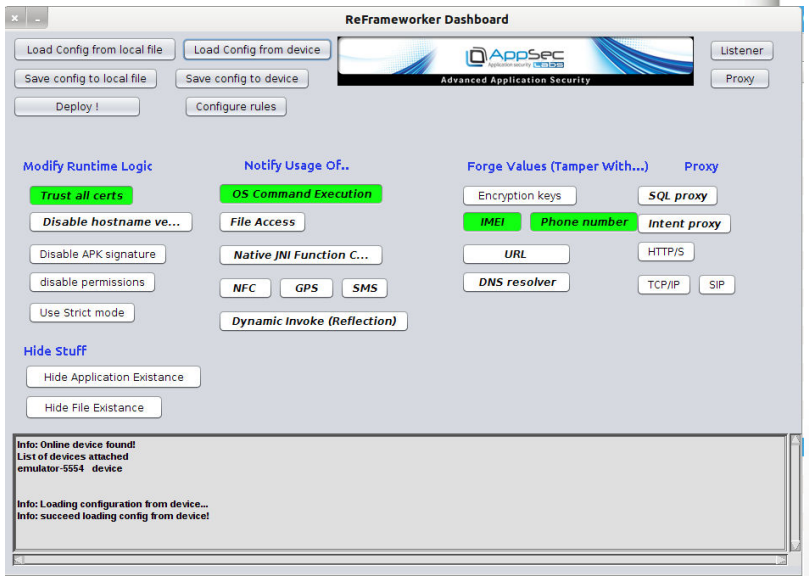
- Then use regular GDB commands such as break, continue, finish, etc.

DEMO (if time permits ☺)

- Analyzing .so files using IDA
- Remotely debugging native code using GDB

The ReFrameworker Platform Changing App Behavior Without Patching Any Code

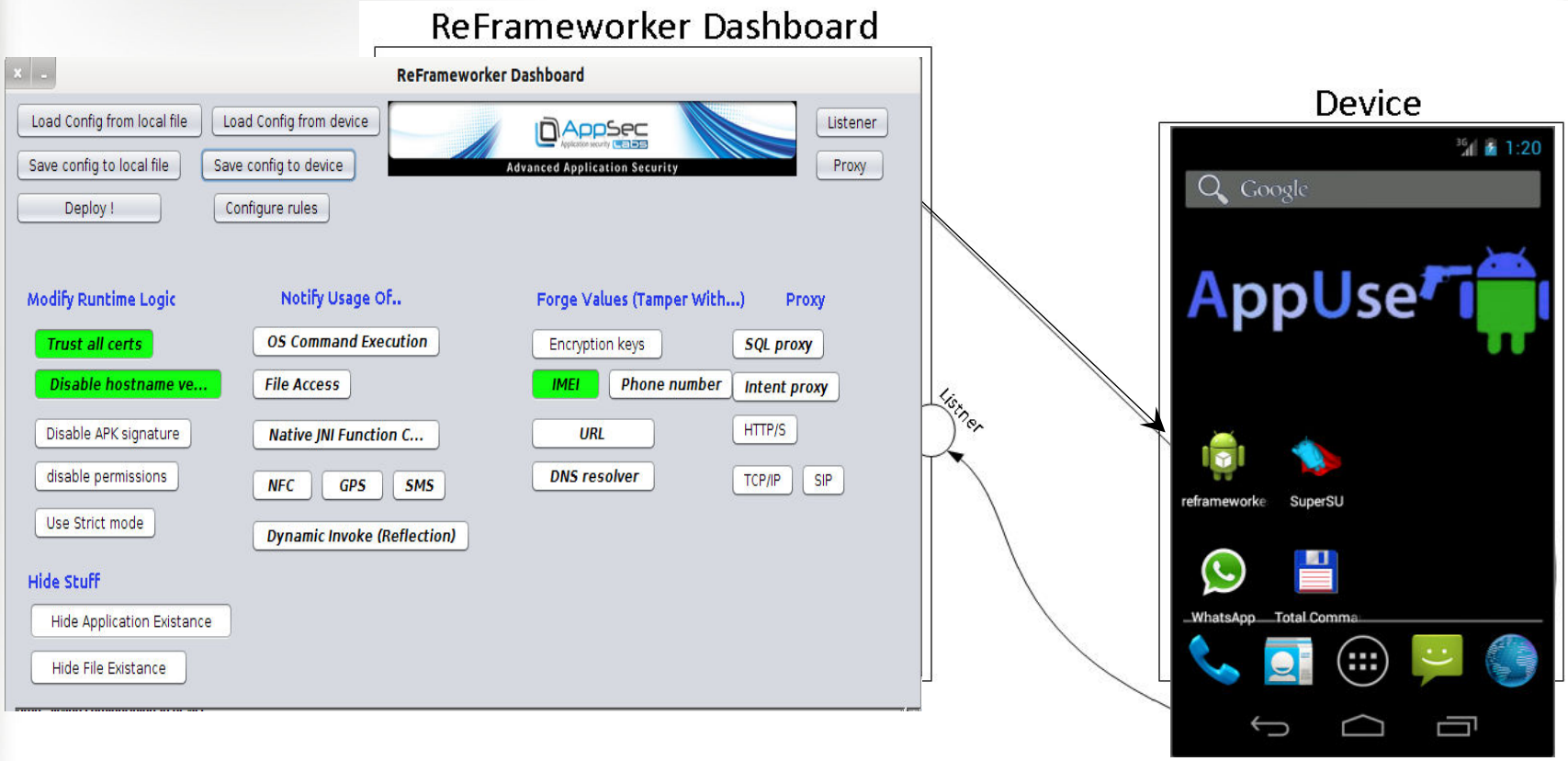
- Runtime manipulation framework by AppSec Labs
- Integrated as part of AppUse
- Released at BlackHat USA 2013
- Presented at OWASP IL 2013 – **look for the slides from last year for more info!!**



How it Works

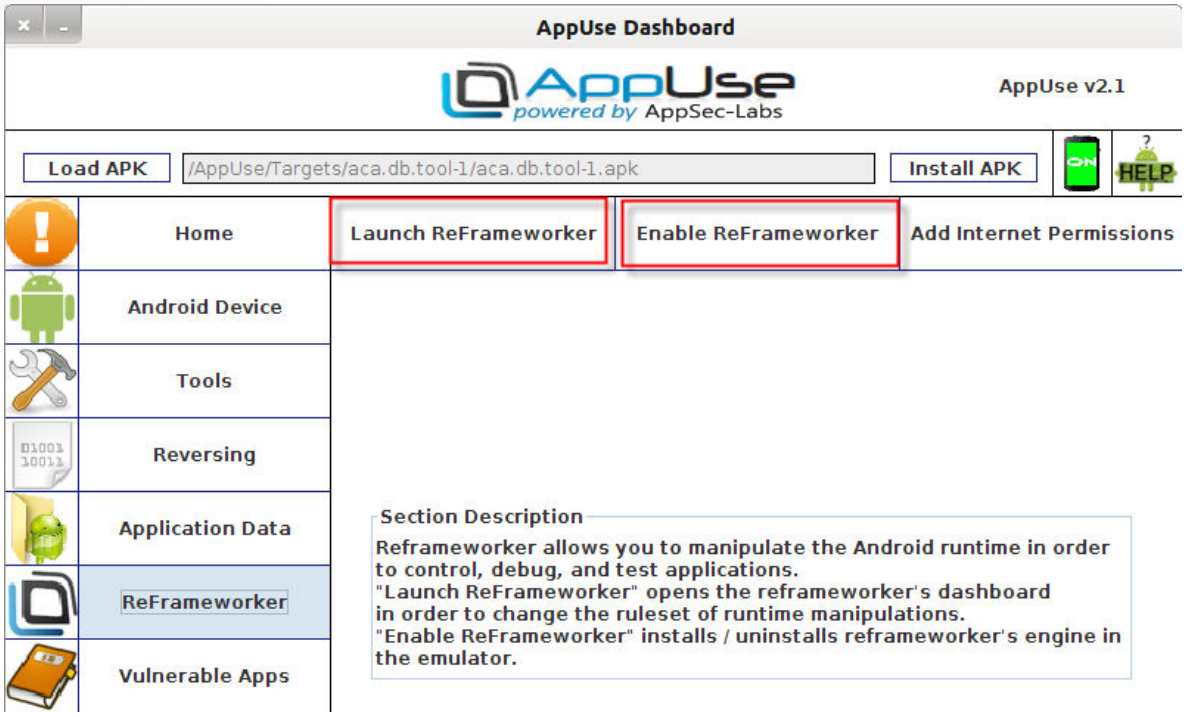
- The Android runtime was compiled with many hooks placed into key placed inside its code.
- The hooks look for a file called "Reframeworker.xml", located inside /data/system.
- So each time an application is executed, whenever a hooked runtime method is called, it loads the ReFrameworker configuration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

Overview - With ReFrameworker

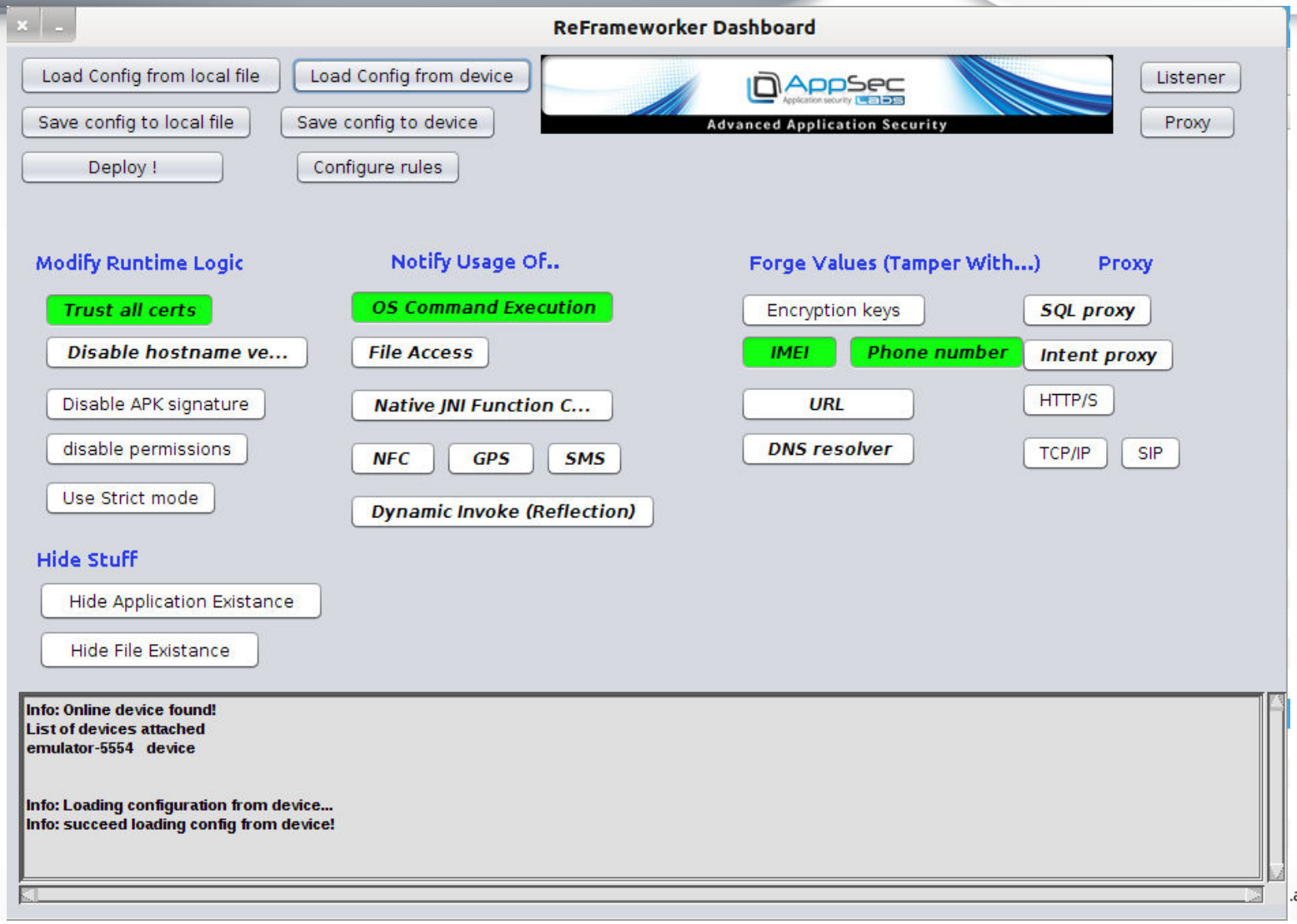


Enabling / Disabling ReFrameworker

- Replacing the original device jars with our modified version



The ReFrameworker Dashboard

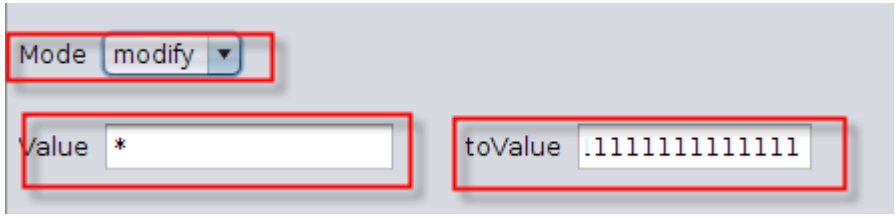


Defining Behavior

- 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.

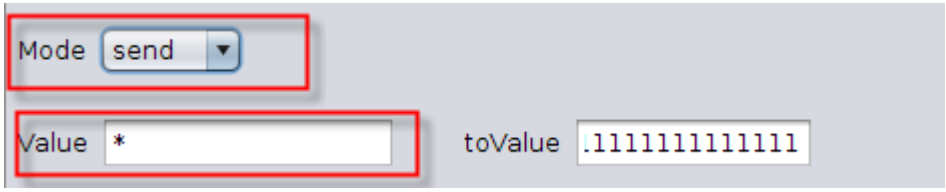
Modify Mode

- 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 you * as ANY (i.e. the hooked value will be sent 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 you * as ANY (i.e. the hooked value will be sent always)
- The toValue is ignored (not in use)

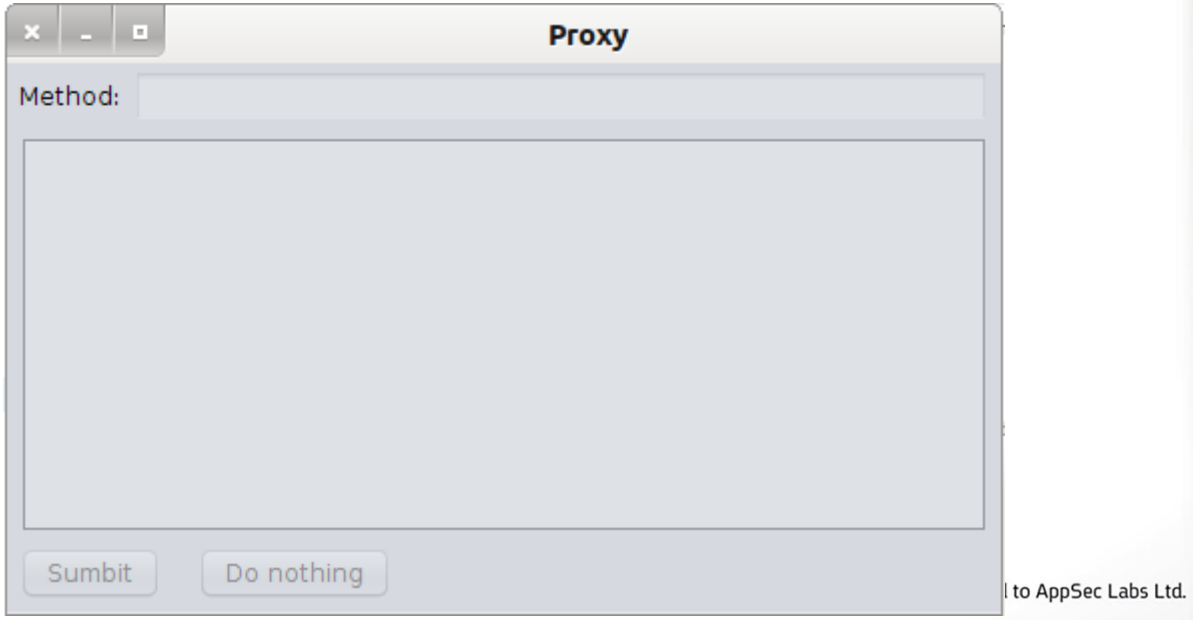


Proxy Mode

- 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.

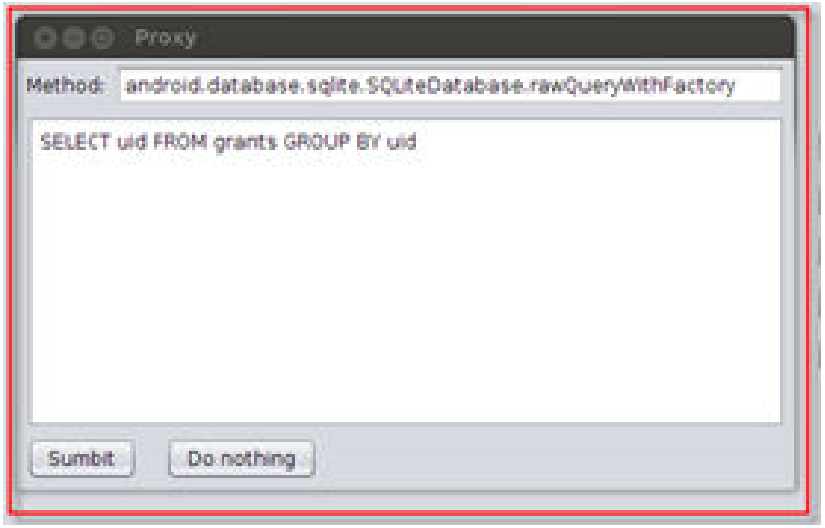
Start the proxy

The proxy window

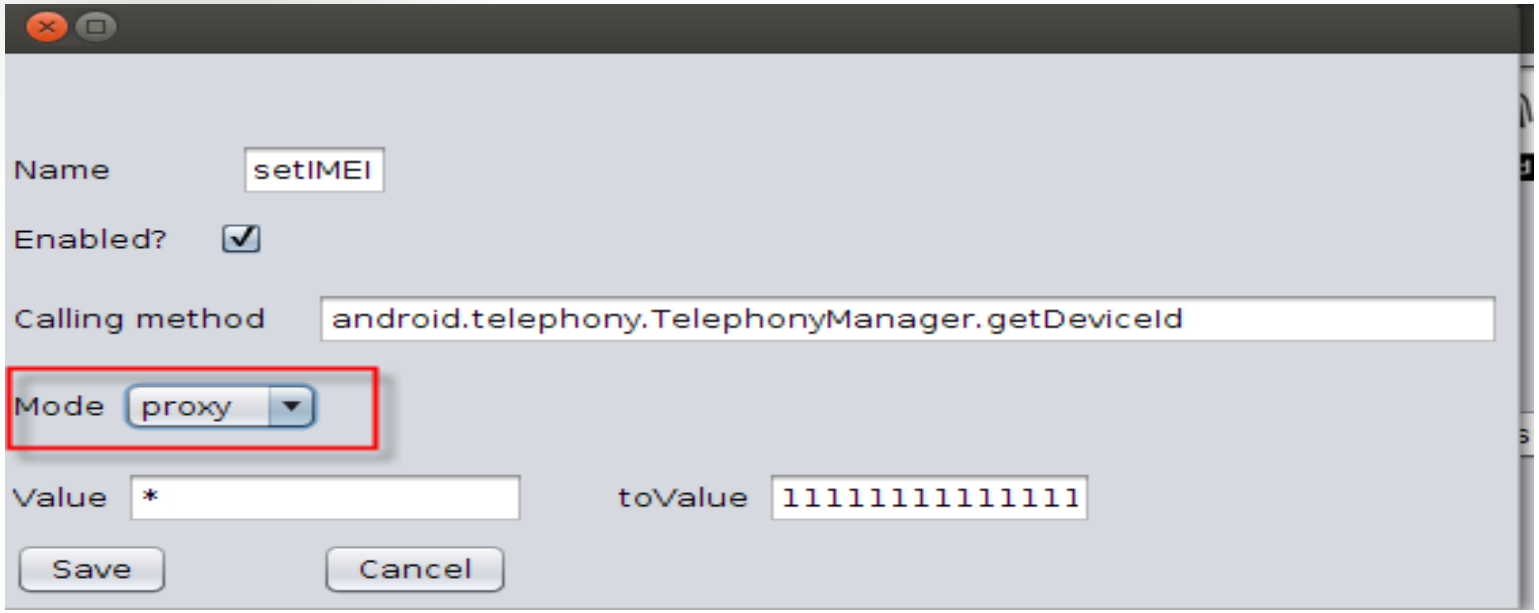


Intercepting Data with the Proxy

- 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



Item Example – Live Editing of the IMEI (Proxy Mode)



Explanation – mode is set to "proxy" since we want to modify this data at realtime. Other values stayed the same (compared to previous example).

Summary

- Runtime analysis provide us with the means to observe the behavior of an app during its execution
- It allows us to inspect issues such as communication, memory, file access, etc.
- We can detect problems that are hard to see using just static methods
- ReFreameworker is a great platform for that

AppSec Application security Labs

QUESTIONS ?

THANK YOU !

Erez Metula , Application Security Expert
AppSec Labs (Founder)
ErezMetula@AppSec-Labs.com

…and last thing: we're hiring !!!