

HUNTING ANDROID MALWARE

---

**A RUNTIME TECHNIQUE FOR  
IDENTIFYING MALICIOUS APPLICATIONS**

# WHOAMI

- ▶ Chris Le Roy
- ▶ Security "Researcher/Engineer"
- ▶ @brompwnie

# OUTLINE

---

- ▶ Problem
- ▶ Question
- ▶ Idea
- ▶ PoC
- ▶ Results
- ▶ Conclusion

# MALWARE HAS AND IS A CONSTANT THREAT IN THE ANDROID ECOSYSTEM

ars TECHNICA UK **BLOG** TECH SCIENCE POLICY CARS GAMING & CULTURE

**2017**

**Malicious apps with >1 million downloads slip past Google defenses twice**

Malware scanners fail to detect 50 apps that charged for fake services.

DAN GOODIN (SS) - 10/9/2017, 10:50

**Found: New Android malware with never-before-seen spying capabilities**

Skygofree is among the most powerful spy platforms ever created for Android.

DAN GOODIN - 1/16/2018, 5:45 PM

**2018**

**Currency-mining Android malware is so aggressive it can physically harm phones**

This is your phone on mining software. Any questions?

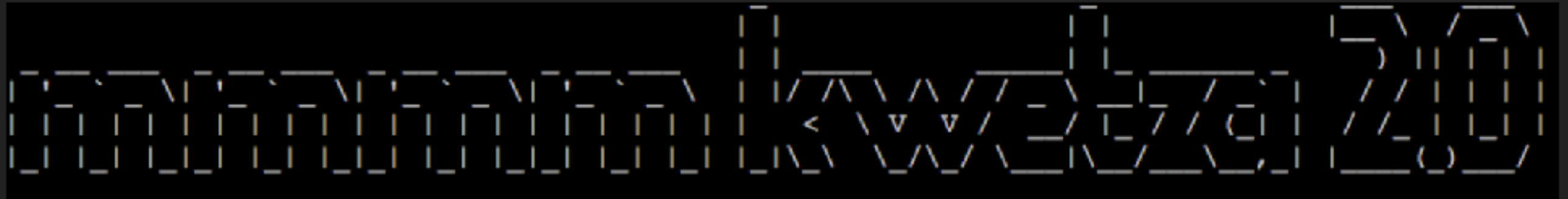
**2017**

DAN GOODIN - 12/19/2017, 8:40 PM

THE QUESTION

---

PREVIOUS RESEARCH LED ME TO THIS RESEARCH



# HOW DO I PROTECT MYSELF FROM THESE KIND OF ATTACKS?

- ▶ We have to look at the APK
  - ▶ Statically
  - ▶ In a sandbox of sorts

# WE LOOK AT MALWARE IN A FEW WAYS

- ▶ Hashes
- ▶ Code Signatures
- ▶ Permissions Reputation
- ▶ Behaviour

# HOW ARE WE PROTECTED BY MALWARE?

- ▶ Google Play Protect
- ▶ Google Playstore
- ▶ Third-Party Software
  - ▶ Anti-Virus
  - ▶ OS Support
  - ▶ MDM's, MAM's



# WHAT'S THE SHORTCOMINGS OF EXISTING TECHNIQUES

- ▶ Static Analysis is hard
- ▶ Can't run Cuckoo on my phone
- ▶ Scalability
- ▶ What if the app is not on an official Store?
- ▶ Bypassing AV is too easy
- ▶ Forensics is cool but how do you do it at realtime?
- ▶ Static analysis can only reveal a subset of the app's functionality

THE FRUSTRATION

---

**NO RELIABLE WAY TO DETECT MALWARE ON DEVICES**

# BUT THERE IS HEAPS OF DATA TO BE LOOKED AT!

- ▶ Android apps make use of objects
- ▶ Import statements are useful BUT
- ▶ You can import but not instantiate
- ▶ If it's instantiated, something is using the object
- ▶ Instantiated objects have data (some)

# ALL THIS DATA HAS TO BE SOMEWHERE TO BE LOOKED AT

- ▶ /PROC/{PID}/MAPS?
  - ▶ Analyse the heap regions?
- ▶ Analyse Heap Dumps? (HPROFs)
- ▶ Memory Forensics?
  - ▶ LiME ~ Linux Memory Extractor
  - ▶ Volatility
- ▶ (gdb) x /20xg 0x7fbd6208?
  - ▶ myObject.hashCode()
  - ▶ Not too bad with the DVM (dlmalloc)

## INSTRUMENTATION

- ▶ Objects exist on the heap so they are accessible
- ▶ Trace calls and monitor/engage with behaviour
- ▶ Its relatively easy
- ▶ Great way to gain insight into applications
- ▶ Object carving functionality is AWESOME

# FRIDA

# WOULDN'T IT BE COOL IF AT RUNTIME I COULD SEE

- ▶ Which objects an app is using
- ▶ Which objects are instantiated
- ▶ What are the values for these objects

THE IDEA

---

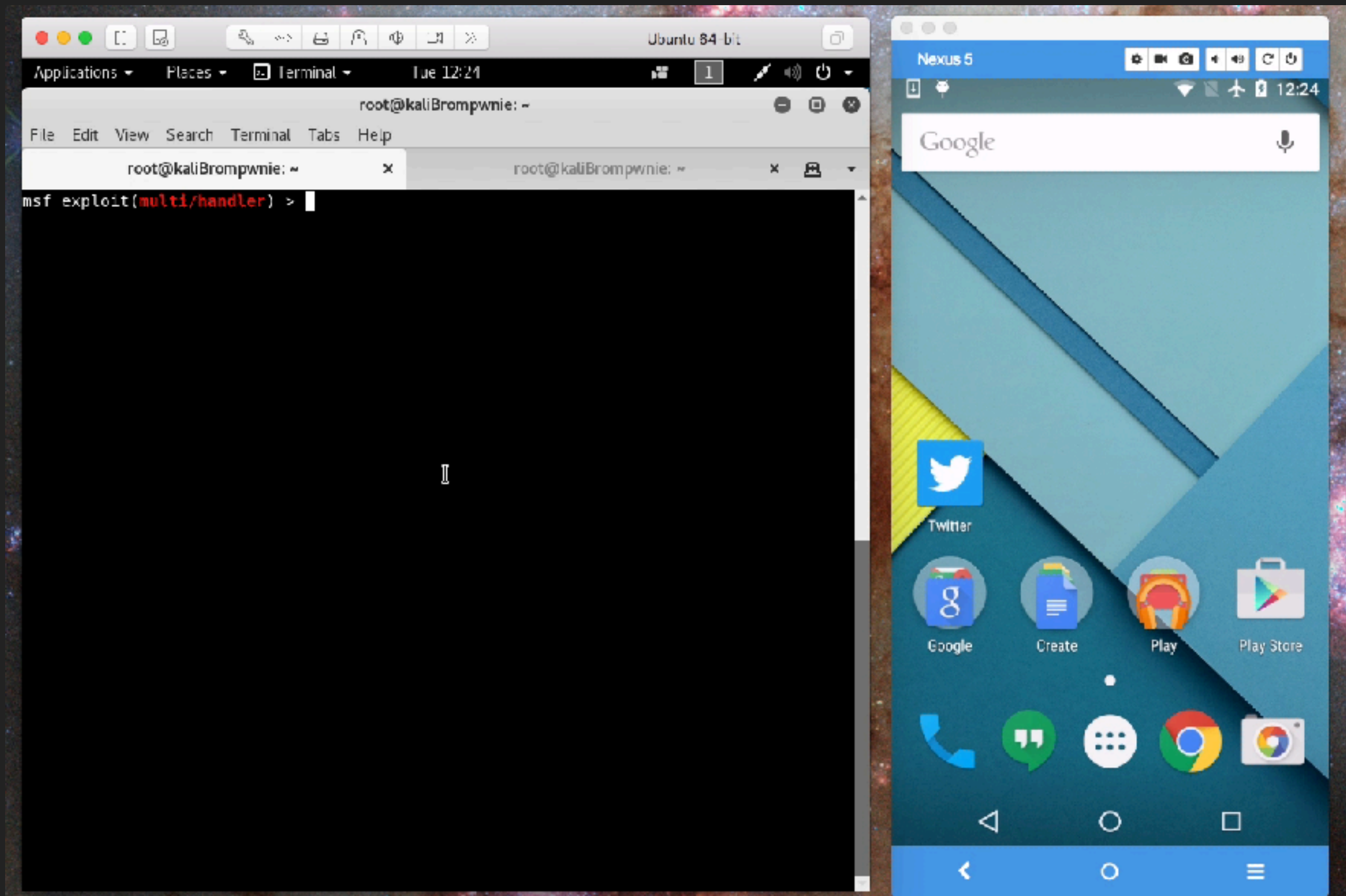
**THIS WOULD GIVE ME AN IDEA AS TO WHAT AN APP IS DOING AND HOW**

# FOR EXAMPLE, ANALYSING AN APP WITH A METERPRETER BACKDOOR:

- ▶ Experience tells me to look for:
  - ▶ DexClassLoader
    - ▶ And what this injected code does
  - ▶ TCP Connection
- ▶ Which tells me that this app is
  - ▶ Injecting code at runtime
    - ▶ For example, encryption/decryption routines
  - ▶ Communicating remotely



# DEMO: BASIC MALWARE INFECTION



IN ACTION

# DEMO: BASIC RUNTIME MALWARE ANALYSIS USING FRIDA

The image displays a development environment with two main components: a code editor on the left and a mobile emulator on the right.

**Code Editor (Left):** The editor shows a JavaScript file named `Demo1_MalwareObjects.js`. The script defines an array `objectsToLookFor` containing class names: `"java.net.Socket"`, `"dalvik.system.DexClassLoader"`, `"java.net.URLConnection"`, `"java.net.URL"`, and `"java.securi"`. A `for` loop iterates over these objects, and for each, a `Java.perform` function is called. Inside this function, a `Java.choose` function is used to find instances of the specified class. For each instance, an `"onMatch"` function is defined, which logs specific information to the console based on the class type. For example, for `java.net.URL`, it logs the protocol and communication details. For `dalvik.system.DexClassLoader`, it logs the instantiation and details. For `java.net.Socket`, it logs the instantiation and connection details. For `java.net.URLConnection`, it logs the instantiation and details. The script is currently at line 18, column 62.

**Terminal (Bottom Left):** The terminal shows the command `cleroy-ltmScripts cleroy$ frida -U -l Demo1_MalwareObjects.js com.twitter.android` being executed.

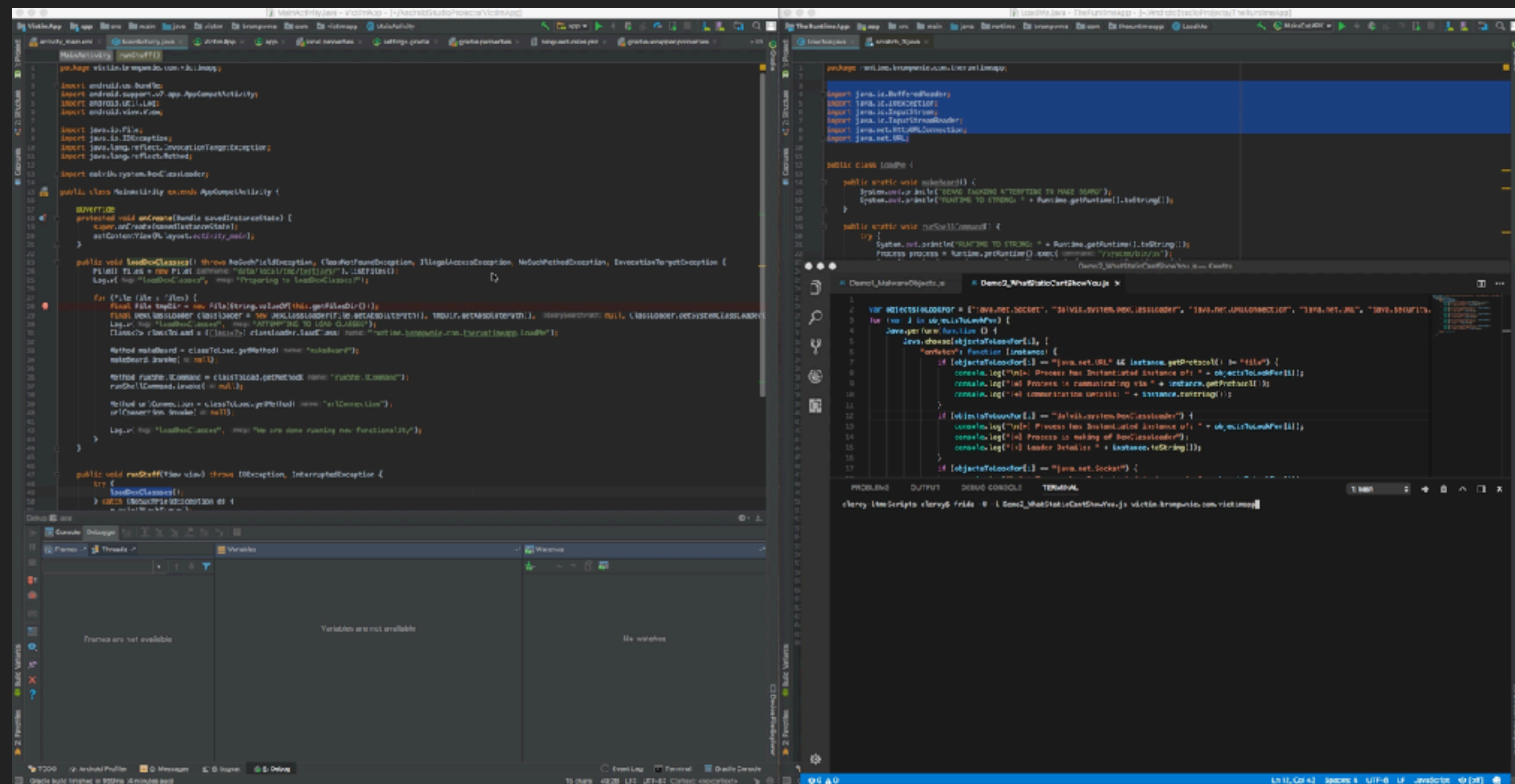
**Mobile Emulator (Right):** The emulator shows a Nexus 5 device running the Twitter app. The screen displays the "Log in to Twitter" screen, with fields for "Phone, email or username" (containing "Brompwnie") and "Password" (containing "apassword"). A "Log in" button is visible at the bottom right of the login form. The status bar at the top shows the time as 12:32.

# STATIC ANALYSIS WONT SHOW YOU EVERYTHING

- ▶ Runtime Injection
  - ▶ Class Loaders
    - ▶ Very powerful to inject your functionality at runtime, requires analyst to acquire the jar/apk/dex
  - ▶ What if you don't have the injected JAR/APK?
    - ▶ `/data/data/com.app.sandbox`
- ▶ Java.Lang
  - ▶ `Runtime.exec("/bin/sh")`
  - ▶ No Import Statements
  - ▶ Instantiated but kinda immutable

# STATIC VS RUNTIME ANALYSIS

## DEMO: WHAT STATIC ANALYSIS CAN'T SHOW YOU



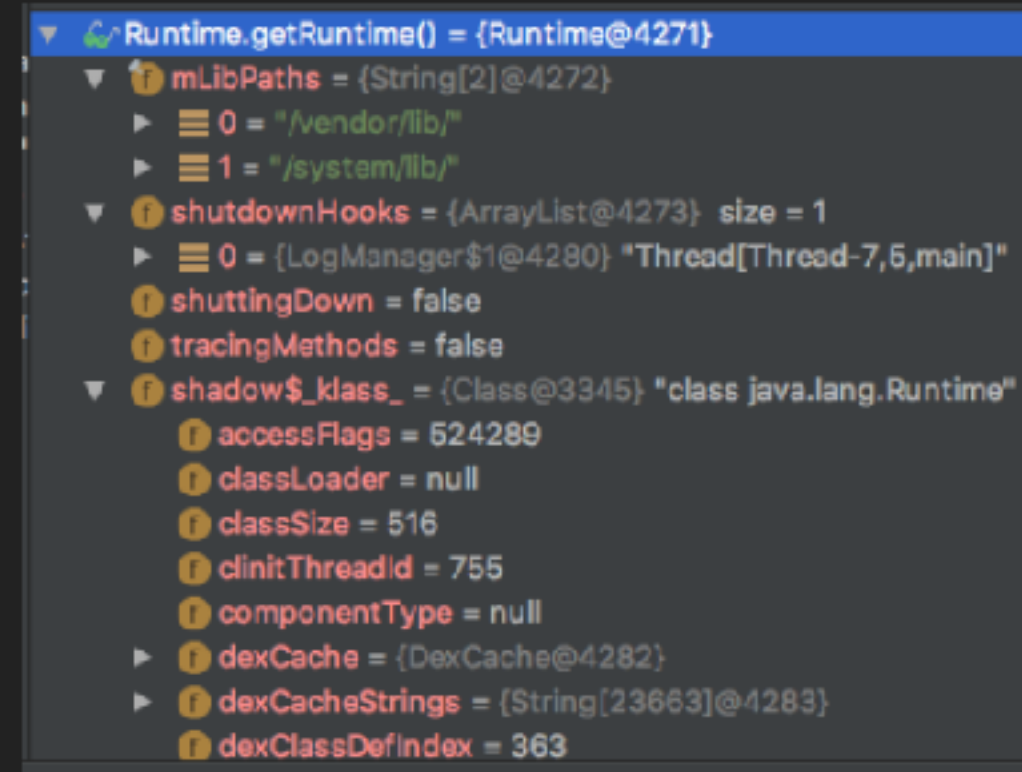


# HEAPS OF LOVE

- ▶ Don't have to trawl code
- ▶ Identify specific anomalies

# THERE IS ALSO HEAPS OF FRUSTRATION

- ▶ `java.lang.Runtime`
- ▶ Kind of immutable?
- ▶ `exec("/system/bin/ps")`
  - ▶ Does not have much of a footprint



## MORE FRUSTRATION

---

# WHATS THE PLAN?

`exec(String command)`

Executes the specified string command in a separate process.

`exec(String[] cmdarray)`

Executes the specified command and arguments in a separate process.

`exec(String[] cmdarray, String[] envp)`

Executes the specified command and arguments in a separate process with the specified environment.

`exec(String[] cmdarray, String[] envp, File dir)`

Executes the specified command and arguments in a separate process with the specified environment and working directory.

`exec(String command, String[] envp)`

Executes the specified string command in a separate process with the specified environment.

`exec(String command, String[] envp, File dir)`

Executes the specified string command in a separate process with the specified environment and working directory.

```
Java.perform(function () {
    var targetClass = Java.use("java.lang.Runtime");

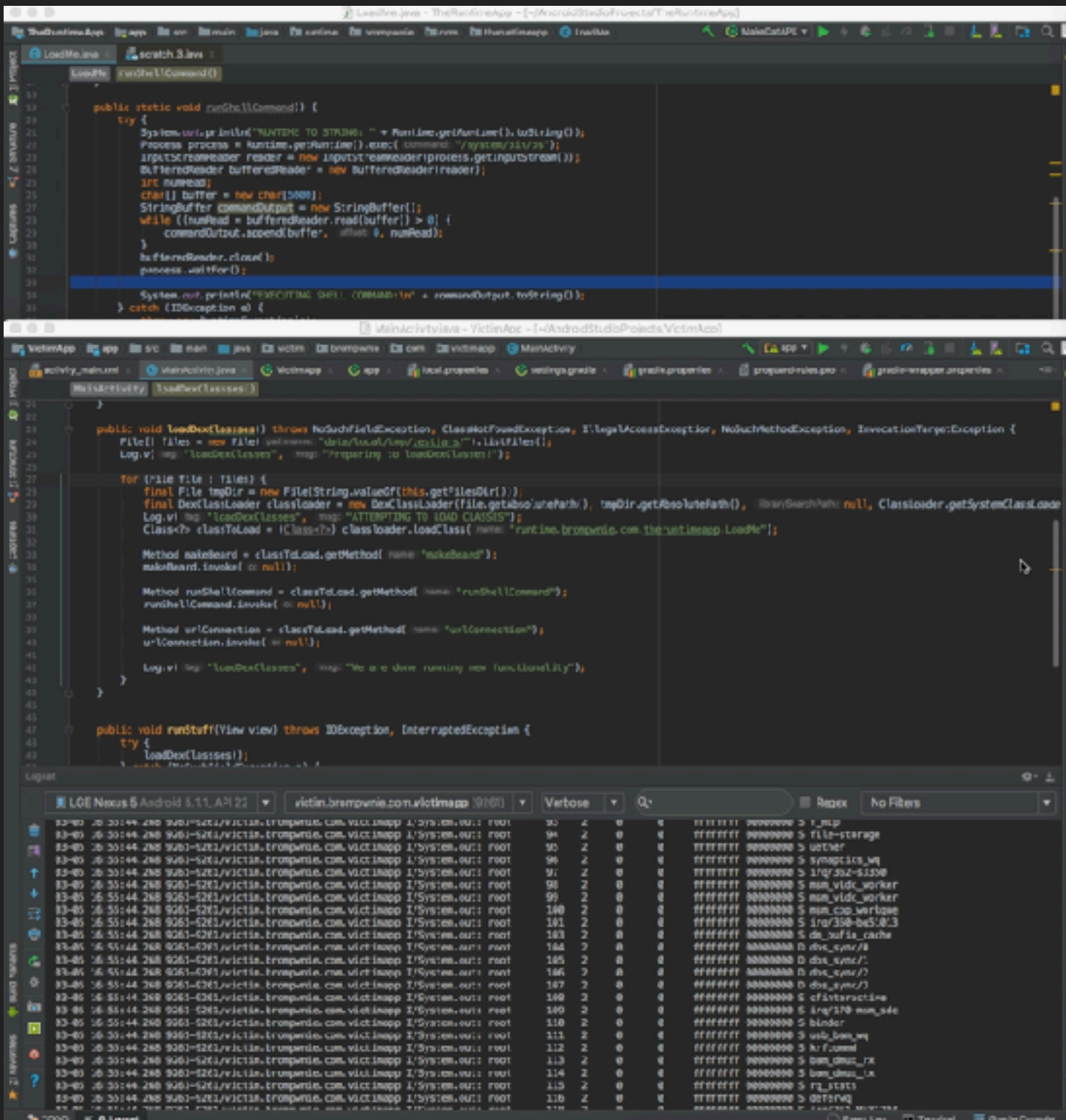
    targetClass.exec.overload('java.lang.String').implementation = function (x) {
        console.log("[*] exec() got called!: " + x);
        return this.exec(x);
    };

    targetClass.exec.overload('[Ljava.lang.String;').implementation = function (x) {
        console.log("[*] exec() got called!: " + x);
        return this.exec(x);
    };

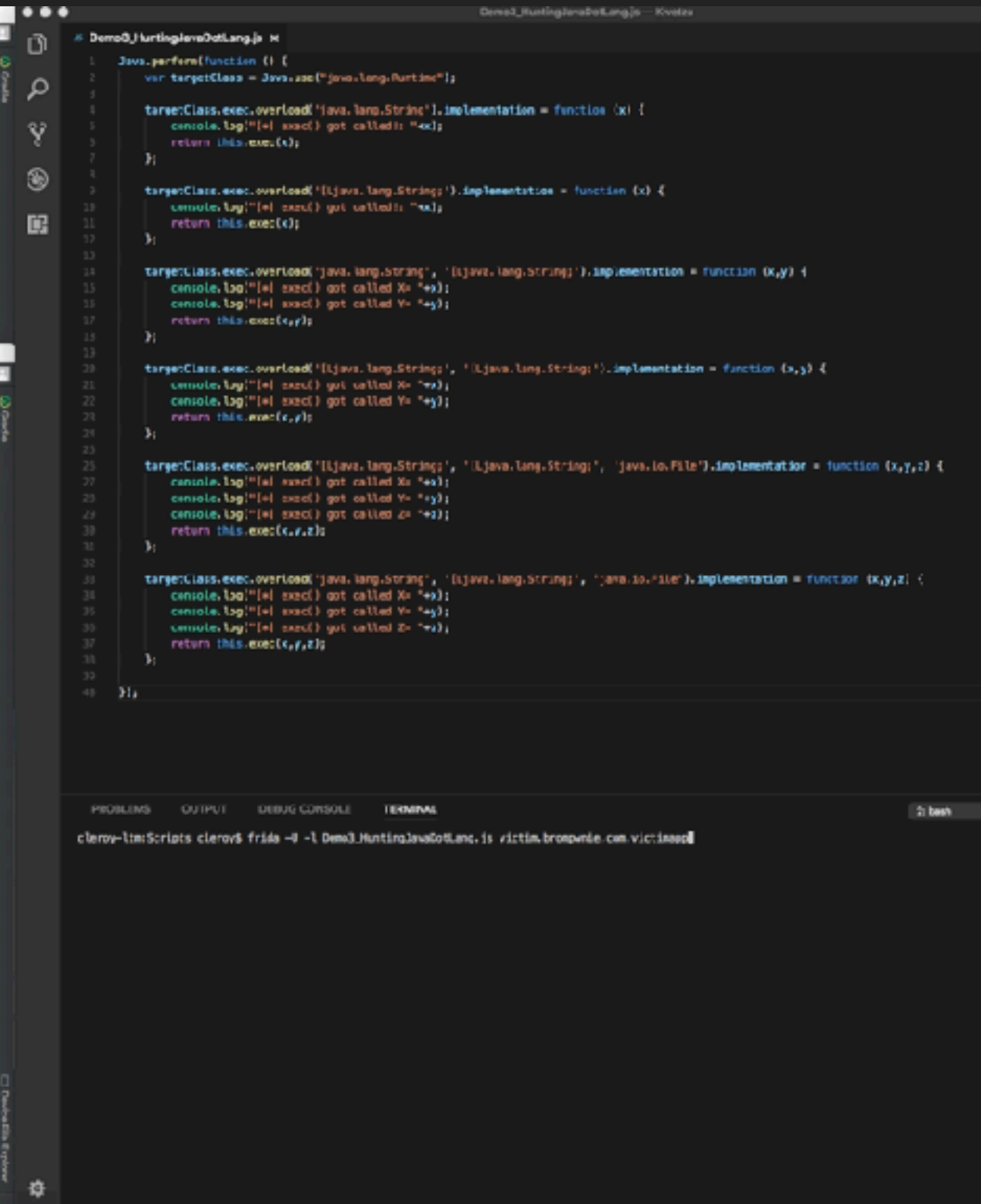
    targetClass.exec.overload('java.lang.String', '[Ljava.lang.String;').implementation = function (x,y) {
        console.log("[*] exec() got called X= " + x);
        console.log("[*] exec() got called Y= " + y);
        return this.exec(x,y);
    };
});
```

# OVERCOME SOME FRUSTRATION

## DEMO: OVERLOAD METHOD CALLS TO OBSERVE OTHERWISE HARD TO OBSERVE ANOMALIES



```
1 public static void runShellCommand() {
2     try {
3         System.out.println("RUNTIME TO STRING: " + Runtime.getRuntime().toString());
4         Process process = Runtime.getRuntime().exec("system/xbin/sh");
5         InputStreamReader reader = new InputStreamReader(process.getInputStream());
6         BufferedReader bufferedReader = new BufferedReader(reader);
7         if (null == bufferedReader) {
8             return;
9         }
10        char[] buffer = new char[5000];
11        StringBuffer commandOutput = new StringBuffer();
12        while ((null != bufferedReader.read(buffer)) > 0) {
13            commandOutput.append(buffer, 0, bufferedReader.read());
14        }
15        bufferedReader.close();
16        process.waitFor();
17        System.out.println("EXECUTING SHELL COMMAND: " + commandOutput.toString());
18    } catch (IOException e) {
19        e.printStackTrace();
20    }
21 }
22
23 public void loadDexClasses() throws NoSuchFieldException, ClassNotPatchedException, IllegalAccessException, NoSuchMethodException, InvocationTargetException {
24     File[] files = new File[] { new File(getAssets().getDataPath() + "/classes.dex") };
25     Log.v("loadDexClasses", "Preparing to loadDexClasses");
26
27     for (File file : files) {
28         final File tmpDir = new File(String.valueOf(this.getAssetsDir()));
29         final DexClassLoader classLoader = new DexClassLoader(file.getAbsolutePath(), tmpDir.getAbsolutePath(), null, ClassLoader.getSystemClassLoader());
30         Log.v("loadDexClasses", "Attempting to load classes");
31         ClassLoader classLoader = classLoader.loadClass("victim.browserside.com.victimapp.MainActivity");
32
33         Method makeBeard = classLoader.getMethod("makeBeard");
34         makeBeard.invoke(null);
35
36         Method runShellCommand = classLoader.getMethod("runShellCommand");
37         runShellCommand.invoke(null);
38
39         Method urlConnection = classLoader.getMethod("urlConnection");
40         urlConnection.invoke(null);
41
42         Log.v("loadDexClasses", "We are done running new functionality");
43     }
44 }
45
46 public void runStuff(View view) throws IOException, InterruptedException {
47     try {
48         loadDexClasses();
49     } catch (Exception e) {
50         e.printStackTrace();
51     }
52 }
```



```
1 Java.perform(function () {
2     var targetClass = Java.use("java.lang.Runtime");
3
4     targetClass.exec.overload("java.lang.String").implementation = function (x) {
5         console.log("[*] exec() got called: " + x);
6         return this.exec(x);
7     };
8
9     targetClass.exec.overload("java.lang.String").implementation = function (x) {
10        console.log("[*] exec() got called: " + x);
11        return this.exec(x);
12    };
13
14    targetClass.exec.overload("java.lang.String", "java.lang.String").implementation = function (x, y) {
15        console.log("[*] exec() got called X= " + x);
16        console.log("[*] exec() got called Y= " + y);
17        return this.exec(x, y);
18    };
19
20    targetClass.exec.overload("java.lang.String", "java.lang.String").implementation = function (x, y) {
21        console.log("[*] exec() got called X= " + x);
22        console.log("[*] exec() got called Y= " + y);
23        return this.exec(x, y);
24    };
25
26    targetClass.exec.overload("java.lang.String", "java.lang.String", "java.io.File").implementation = function (x, y, z) {
27        console.log("[*] exec() got called X= " + x);
28        console.log("[*] exec() got called Y= " + y);
29        console.log("[*] exec() got called Z= " + z);
30        return this.exec(x, y, z);
31    };
32
33    targetClass.exec.overload("java.lang.String", "java.lang.String", "java.io.File").implementation = function (x, y, z) {
34        console.log("[*] exec() got called X= " + x);
35        console.log("[*] exec() got called Y= " + y);
36        console.log("[*] exec() got called Z= " + z);
37        return this.exec(x, y, z);
38    };
39 });
```



# WE HAVE THE ABILITY TO:

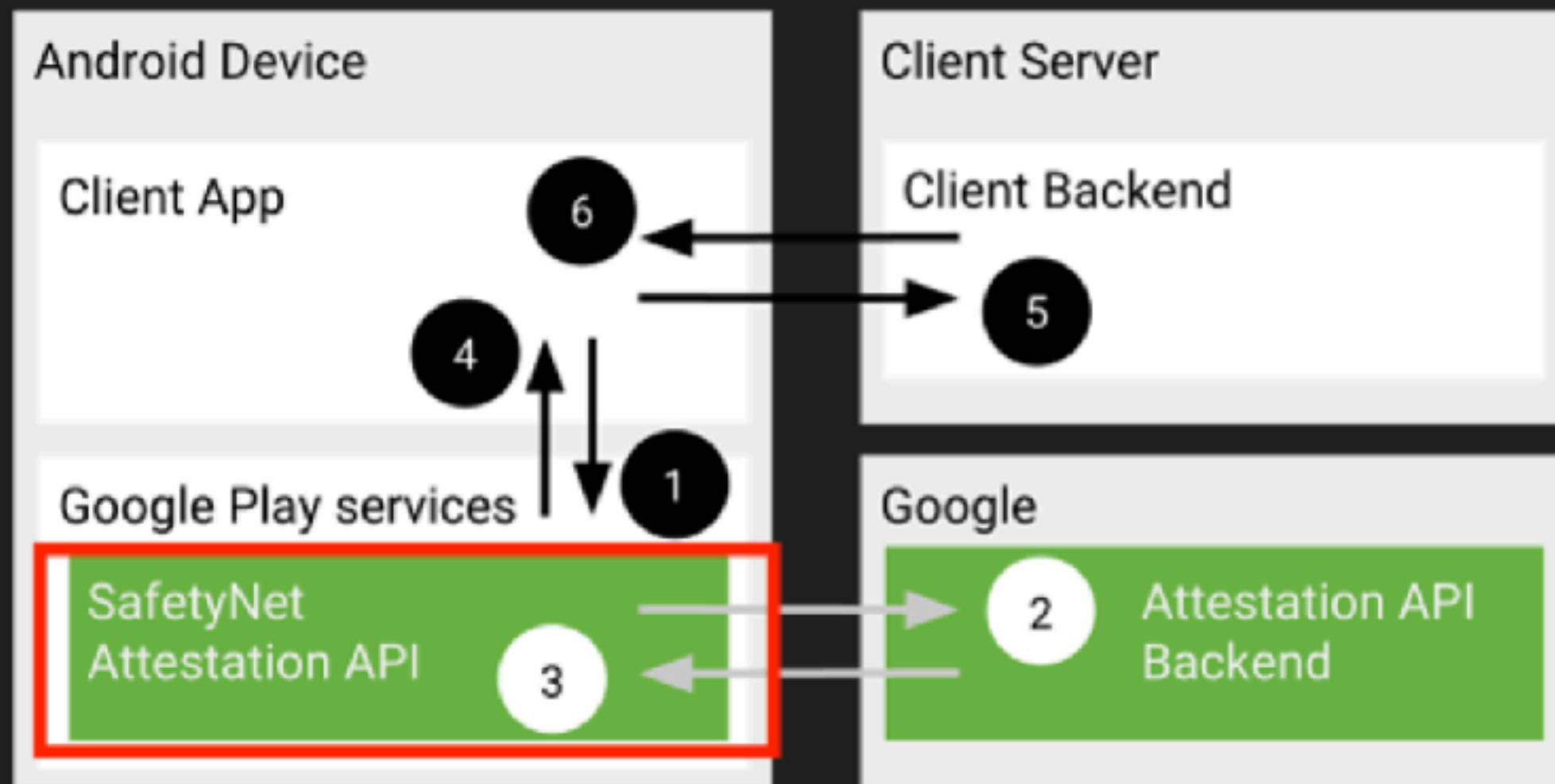
- ▶ Analyse objects on the heap
- ▶ Hook methods for certain objects
- ▶ Do all this at runtime on a device
- ▶ See more than static analysis
- ▶ Perform the above from a workstation

# A SOLUTION: SAFETY NET ATTESTATION API

*The SafetyNet Attestation API helps you assess the security and compatibility of the Android environments in which your apps run. You can use this API to analyze devices that have installed your app.*

HOW TO USE THIS?

## A SOLUTION: SAFETY NET ATTESTATION API



# UITKYK

- ▶ You can use this API to analyze applications that are installed on a Android device
- ▶ Custom Android Frida Library
- ▶ DBUS over TCP
- ▶ Frida Server Integration
- ▶ Can run all the previously demo'd tests
- ▶ And more!

## HEY FRIDA, GIVE ME RUNNING PROCESSES

```
$ frida-ps -U
```

PID	Name
207	adbd
27599	android.process.acore
1566	android.process.media
16850	app_process
16894	app_process
17005	app_process
17145	app_process
17163	app_process
17190	app_process
17327	app_process
17384	app_process
17430	app_process
26767	app_process32
194	bridgemgrd
27556	com.android.defcontainer
27799	com.android.keychain
27622	com.android.musicfx
1799	com.android.nfc
1855	com.android.phone
1544	com.android.systemui
27643	com.android.vending
27882	com.google.android.apps.cloudprint

# HEY ANDROID, GIVE ME RUNNING PROCESSES

```
@Override  
protected String doInBackground(String... params) {  
    UitkykUtils uitkykUtils= new UitkykUtils();  
    return uitkykUtils.fridaPS(fridaHost,fridaPort);  
}
```

```
$ frida -U -l Demo1_MalwareObjects.js com.twitter.android
```

```
[+] Process has Instantiated instance of: java.net.Socket
[*] Process is making of a Socket Connection
[+] Socket Details: Socket[address=/192.168.0.16,port=4444,localPort=60695]

[+] Process has Instantiated instance of: java.net.Socket
[*] Process is making of a Socket Connection
[+] Socket Details: Socket[address=settings.crashlytics.com/50.19.106.12,port=443,localPort=42306]
```

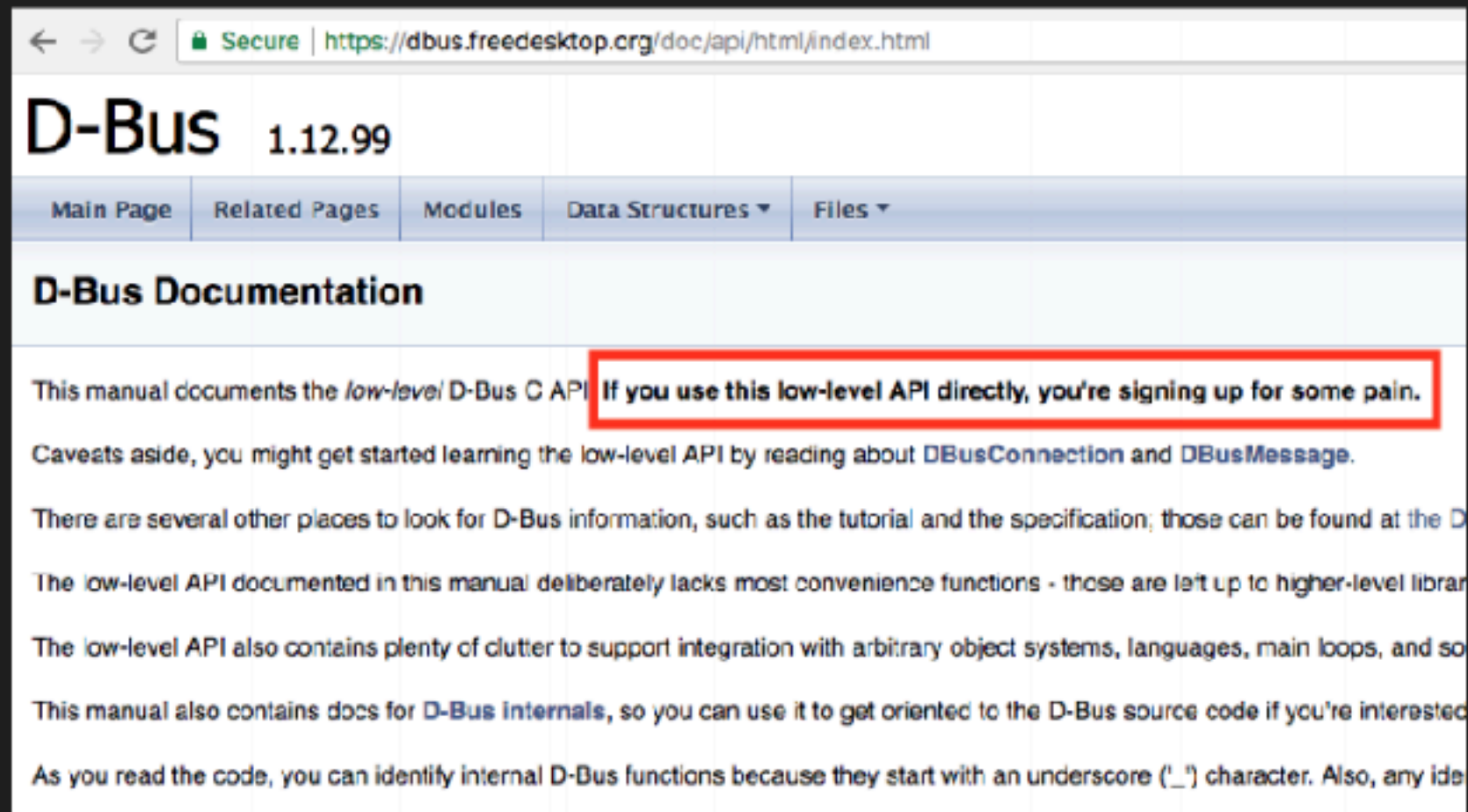
# HEY ANDROID TELL ME IF THIS APPLICATION LOOKS MALICIOUS

```
protected String doInBackground(Object... objects) {  
    UitkykUtils uitkykUtils = new UitkykUtils(fridaHost, fridaPort);  
    return uitkykUtils.analyzeProcess(this.pid);  
}
```



## WHY UITKYK API?

- ▶ No Android Frida Library
- ▶ Wanted to use Frida
- ▶ Wanted a Client Server Model
- ▶ Didn't want pain



# HOW DOES UITKYK UITKYK?

- ▶ TCP Socket to Daemon
- ▶ Push and Pull Bytes
- ▶ Sniffed Frida sessions
- ▶ Outlined TCP Flags
- ▶ Identified key bytes (trial and error)
- ▶ Stared at my monitor
- ▶ Wash,rinse,repeat

```
byte[] AUTH_Message2 = {  
    (byte) 0x41, (byte) 0x55, (byte) 0x54, (byte) 0x48,  
    (byte) 0x20, (byte) 0x41, (byte) 0x4e, (byte) 0x4f,  
    (byte) 0x4e, (byte) 0x59, (byte) 0x4d, (byte) 0x4f,  
    (byte) 0x55, (byte) 0x53, (byte) 0x20, (byte) 0x34,  
    (byte) 0x37, (byte) 0x34, (byte) 0x34, (byte) 0x34,  
    (byte) 0x32, (byte) 0x37, (byte) 0x35, (byte) 0x37,  
    (byte) 0x33, (byte) 0x32, (byte) 0x30, (byte) 0x33,  
    (byte) 0x30, (byte) 0x32, (byte) 0x65, (byte) 0x33,  
    (byte) 0x31, (byte) 0x0d, (byte) 0x0a};
```

# SOMETHING ABOUT A POC

```

import socket

TCP_IP = '10.42.0.15'
TCP_PORT = 1337
BUFFER_SIZE = 100
# .AUTH
MESSAGE = '\x2e\x41\x55\x54\x48\x8d\x0n'
# MESSAGE = '\x41\x55\x54\x48\x8d\x0n'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Sending-> .AUTH"
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
print "received data:", data
# s.close()

# AUTH ANONYMOUS 474442757320302e31
MESSAGE2="\x41\x55\x54\x48\x20\x41\x4e\x4f\x4e\x59\x4d\x4f\x55\x53\x20\x34\x37\x34\x34\x34\x32\x37\x35\x37\x33\x32\x30\x33\x30\x32"
BUFFER_SIZE2 = 100
# s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Sending-> AUTH ANONYMOUS 474442757320302e31"
# s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE2)
data2 = s.recv(BUFFER_SIZE2)
print "received data:", data2
# s.close()

# #
# #BEGIN
MESSAGE3="\x42\x45\x47\x49\x4e\x0d\x0a"
BUFFER_SIZE3 = 100
# s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Sending-> BEGIN"
# s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE3)
# data3 = s.recv(BUFFER_SIZE3)
# print "received data:", data3
# # s.close()

# 6c 01 00 01 1b 00 00 00 01 02 00 02 60 02 03 02 08 01 57 00 01 73 00 20 01 01 6f 20 15 00 00 00 2f 72 65 2f 66 72 69 64 61 2f 48 6
# HostSession.....s.....GetAll.....s.....org.freedesktop.DBus.Properties.....re.frida.HostSession10.
MESSAGE4="\x6c\x01\x00\x01\x1b\x00\x20\x00\x01\x20\x00\x00\x60\x00\x20\x00\x08\x01\x67\x00\x01\x73\x00\x00\x01\x01\x6f\x00\x15\x00"
BUFFER_SIZE4 = 100
print "Sending-> HostSession"
s.send(MESSAGE4)
data4 = s.recv(BUFFER_SIZE4)
print "received data:", data4

```

## WHERE TO GET IT ALL

- ▶ Library

- ▶ [github.com/brompwnie/uitkyk](https://github.com/brompwnie/uitkyk)

- ▶ Frida Scripts

- ▶ [github.com/brompwnie/uitkyk](https://github.com/brompwnie/uitkyk)

- ▶ Videos

- ▶ <https://goo.gl/k6BNBq>

## SHORTCOMINGS

- ▶ Increased Attack Surface
- ▶ Abuse, it is process running as root
- ▶ We are still struggling to get basic security right

## CONCLUSION & QUESTIONS

- ▶ It's a journey
- ▶ Uitkyk is a step in the right direction
- ▶ No Silver Bullet
- ▶ Defence In Depth
- ▶ Android OS is key to protecting itself
- ▶ Questions?