HUNTING ANDROID MALWARE

A RUNTIME TECHNIQUE FOR IDENTIFYING MALICIOUS APPLICATIONS
WHOAMI

- Chris Le Roy
- Security “Researcher/Engineer"
- @brompwnie
THE PROBLEM

MALWARE HAS AND IS A CONSTANT THREAT IN THE ANDROID ECOSYSTEM

---

**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

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

This is your phone on mining software. Any questions?

*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
CURRENT MALWARE ANALYSIS TECHNIQUES

WE LOOK AT MALWARE IN A FEW WAYS

- Hashes
- Code Signatures
- Permissions Reputation
- Behaviour
CURRENT DEFENCES

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)
THE IDEA

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)
THE IDEA

INSTRUMENTATION

- Objects exist on the heap so they are accessible
- Trace calls and monitor/engage with behaviour
- It's relatively easy
- Great way to gain insight into applications
- Object carving functionality is AWESOME
WON'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
THIS WOULD GIVE ME AN IDEA AS TO WHAT AN APP IS DOING AND HOW
THE IDEA

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
IN ACTION

DEMO: BASIC MALWARE INFECTION
DEMO: BASIC RUNTIME MALWARE ANALYSIS USING FRIDA
STATIC VS RUNTIME ANALYSIS

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
MORE FRUSTRATION

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

WHAT'S THE PLAN?

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

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

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

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

DEMO: OVERLOAD METHOD CALLS TO OBSERVE OTHERWISE HARD TO OBSERVE ANOMALIES
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
A SOLUTION

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**

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

HEX ANDROID, GIVE ME RUNNING PROCESSES

```java
@override
protected String doInBackground(String... params) {
    UitkykUtils uitkykUtils = new UitkykUtils();
    return uitkykUtils.fridaPS(fridaHost, fridaPort);
}
```
Hey Frida, tell me if this application looks malicious.

```javascript
var objectsToLookFor = [
  "java.net.Socket",
  "dalvik.system.DexClassLoader",
  "java.net.URLConnection",
  "java.net.URL",
  "java.security.cert.X509Certificate"
];
for (var i in objectsToLookFor) {
  Java.perform(function () {
    Java.choose(objectsToLookFor[i], function (instance) {
      if (objectsToLookFor[i] == "java.net.URL" && instance.getProtocol() == "file") {
        console.log("\n\n+ Process has instantiated instance of: " + objectsToLookFor[i]);
        console.log("+ Process is communicating via " + instance.getProtocol());
        console.log("\n\n- Communication Details: " + instance.toString());
      }
      if (objectsToLookFor[i] == "dalvik.system.DexClassLoader") {
        console.log("\n\n+ Process has instantiated instance of: " + objectsToLookFor[i]);
        console.log("+ Process is making of DexClassLoader:\n" + instance.toString());
      }
      console.log("\n\n- Loader Details: " + instance.toString());
    });
  });
}
```

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

Frida 10.6.51 - A world-class dynamic instrumentation toolkit

Commands:
- help → Displays the help system
- object? → Display information about 'object'
- exit/quit → Exit

More info at http://www.frida.re/docs/home/

[+] 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=50695]

[+] 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]
A SOLUTION

HEY ANDROID TELL ME IF THIS APPLICATION LOOKS MALICIOUS

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

**WHY UITKYK API?**

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

 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
import socket
TCP_IP = '10.42.0.15'
TCP_PORT = 1337
BUFFER_SIZE = 100

# AUTH
MESSAGE = 'v\x24v43v\x55v54lv4fv8r\x80'

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

# AUTH ANONYMOUS 4744427573292e31
MESSAGE2 = 'v\x24v43v\x55v54lv4fv8r\x80'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Sending -- AUTH ANONYMOUS 4744427573292e31"
s.connect((TCP_IP, TCP_PORT))
data2 = s.recv(BUFFER_SIZE)
print "received data", data2
s.close()

# BEGIN
MESSAGE3 = 'v\x24v43v\x55v47v46v4dv80v80'

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print "Sending -- BEGIN"
s.connect((TCP_IP, TCP_PORT))
data3 = s.recv(BUFFER_SIZE)
print "received data", data3
s.close()
WHERE TO GET IT ALL

- Library
  - github.com/brompwnie/uitkyk

- Frida Scripts
  - 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?