Rage Against The Virtual Machine: Hindering Dynamic Analysis of Android Malware

Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Sotiris Ioannidis, Michalis Polychronakakis
Android Dominates Market Share

- Smartphones have overtaken client PCs
- Android accounted for 79% of global smartphone market in 2013

Q2 2013 Smartphone Market Share

- Android 79.0%
- iOS 14.2%
- Microsoft 3.3%
- Other 3.6%

Source: Gartner

Thanasis Petsas
Android Malware

• 98% of all mobile threats target Android devices

Source: KASPERSKY Lab

Distribution of mobile malware detected by platform – 2013
Android specific anti-malware tools

• Static analysis tools (AV apps)

• Dynamic analysis services
Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures

- Dynamic analysis services
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users

• Dynamic analysis services
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection

• Dynamic analysis services
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection
  – *How to evade static analysis?*

• Dynamic analysis services
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection
  – *How to evade static analysis?*

• Dynamic analysis services
Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - *How to evade static analysis?*

- Dynamic analysis services
  - Used by security companies
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection
  – How to evade static analysis?

• Dynamic analysis services
  – Used by security companies
  – Run applications on an Emulator
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection
  – *How to evade static analysis?*

• Dynamic analysis services
  – Used by security companies
  – Run applications on an Emulator
  – Detect *suspicious* behavior
Android specific anti-malware tools

- Static analysis tools (AV apps)
  - Identify malware through signatures
  - Usually installed by users
  - Real time protection
  - \textit{How to evade static analysis?}

- Dynamic analysis services
  - Used by security companies
  - Run applications on an Emulator
  - Detect \textit{suspicious} behavior
  - \textit{How to evade dynamic analysis?}
Android specific anti-malware tools

• Static analysis tools (AV apps)
  – Identify malware through signatures
  – Usually installed by users
  – Real time protection
  – How to evade static analysis?

• Dynamic analysis services
  – Used by security companies
  – Run applications on an Emulator
  – Detect suspicious behavior
  – How to evade dynamic analysis?

This work
Our Study

**Objective:** *Can we effectively detect Android emulated analysis environment?*

- A taxonomy of emulation evasion heuristics
- Evaluation of our heuristics on popular dynamic analysis services for Android
- Countermeasures
## VM Evasion Heuristics

<table>
<thead>
<tr>
<th>Category</th>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>Pre-installed static information</td>
<td>IMEI has a fixed value</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Dynamic information does not change</td>
<td>Sensors produce always the same value</td>
</tr>
<tr>
<td>Hypervisor</td>
<td>VM instruction emulation</td>
<td>Native code runs differently</td>
</tr>
</tbody>
</table>
Static Heuristics

• Device ID ($IdH$)
  – IMEI, IMSI

• Current build ($buildH$)
  – Fields: PRODUCT, MODEL, HARDWARE

• Routing table ($netH$)
  – virtual router
    address space: 10.0.2/24
  – Emulated network
    IP address: 10.0.2.15
Static Heuristics

- **Device ID (IdH)**
  - IMEI, IMSI

- **Current build (buildH)**
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table (netH)**
  - virtual router
    - address space: 10.0.2/24
  - Emulated network
    - IP address: 10.0.2.15
Static Heuristics

- **Device ID** (*IdH*)
  - IMEI, IMSI

- **Current build** (*buildH*)
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table** (*netH*)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15
Static Heuristics

- **Device ID** (*IdH*)
  - IMEI, IMSI

- **Current build** (*buildH*)
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table** (*netH*)
  - virtual router
    - address space: 10.0.2/24
  - Emulated network
    - IP address: 10.0.2.15

Android Pincer malware family
Static Heuristics

- **Device ID (IdH)**
  - IMEI, IMSI

- **Current build (buildH)**
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table (netH)**
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

---

Thanasis Petsas
Static Heuristics

- **Device ID (IdH)**
  - IMEI, IMSI

- **Current build (buildH)**
  - **IMEI**: 123456789012347
  - **MODEL**: Nexus 5
  - **null**
  - **google_sdk**

- **Routing table (netH)**
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15
Static Heuristics

- **Device ID** ($IdH$)
  - IMEI, IMSI

- **Current build** ($buildH$)
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table** ($netH$)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15
Static Heuristics

- **Device ID** ($IdH$)
  - IMEI, IMSI

- **Current build** ($buildH$)
  - Fields: PRODUCT, MODEL, HARDWARE

- **Routing table** ($netH$)
  - virtual router address space: 10.0.2/24
  - Emulated network IP address: 10.0.2.15

---

**Android Pincer malware family**

<table>
<thead>
<tr>
<th>IMEI</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>Nexus 5</td>
</tr>
<tr>
<td>/proc/</td>
<td>google_sdk</td>
</tr>
<tr>
<td>net/tcp</td>
<td>Ordinary</td>
</tr>
<tr>
<td></td>
<td>network</td>
</tr>
<tr>
<td></td>
<td>Emulated</td>
</tr>
<tr>
<td></td>
<td>network</td>
</tr>
</tbody>
</table>
Static Heuristics

• Device ID ($IdH$)
  – IMEI, IMSI

• Current build ($buildH$)
  – Fields: PRODUCT, MODEL, HARDWARE

• Routing table ($netH$)
  – virtual router address space: 10.0.2/24
  – Emulated network IP address: 10.0.2.15

Thanasis Petsas
Dynamic Heuristics (1/3)

Sensors:

- A key difference between mobile & conventional systems
- new opportunities for mobile devices identification
- *Can emulators realistically simulate device sensors?*

---

GPS
Accelerometer  Gyroscope
Gravity Sensor  Proximity Sensor
Rotation Vector  Magnetic Field
Dynamic Heuristics (1/3)

Sensors:

- A key difference between mobile & conventional systems
- new opportunities for mobile devices identification
- *Can emulators realistically simulate device sensors?*
  - Partially: same value, equal time intervals
Dynamic Heuristics (2/3)

CDF

Accelerometer’s events interval (sec)
Dynamic Heuristics (2/3)

Generation of the same value at equal time intervals

0.8 ± 0.003043
Dynamic Heuristics (3/3)

- Sensor-based heuristics

- Android Activity that monitors sensors’ output values

- We implemented this algorithm for a variety of sensors
  - Accelerometer (\texttt{accelH})
  - magnetic field (\texttt{magnFH})
  - rotation vector (\texttt{rotVecH}),
  - proximity (\texttt{proximH})
  - gyroscope (\texttt{gyrosH})
Hypervisor Heuristics

• Try to identify the hosted virtual machine

• Android Emulator is based on QEMU

• Our heuristics
  – Based on QEMU’s incomplete emulation of the actual hardware
  – Identify QEMU scheduling
  – Identify QEMU execution using self-modifying code
Identify QEMU Scheduling (1/2)

• Virtual PC in QEMU
  – is updated only after the execution of a basic block (branch)
  – OS scheduling does not occur during a basic block

• QEMU Binary Translation (BT) Detection
  – Monitor scheduling addresses of a thread
    • Real Device: Various scheduling points
    • Emulator: A unique scheduling point
  – BTdetectH
Identify QEMU Scheduling (2/2)

![Graph showing CDF vs Scheduling point (basic block offset)]
Identify QEMU Scheduling (2/2)

![Graph showing CDF vs. Scheduling point (basic block offset)].

**Emulator:** A specific scheduling point.
ARM Architecture

Device

Memory

I-Cache  D-Cache

Emulator

Memory

Cache
ARM Architecture

Device

- Memory
  - I-Cache
  - D-Cache

Emulator

- Memory
  - Cache

old code
ARM Architecture

Device

Emulator

Memory

I-Cache

D-Cache

Cache

old code

new code

Thanasis Petsas
ARM Architecture

Caches are not coherent!
ARM Architecture

Device

Emulator

Memory

I-Cache  D-Cache

old code

new code

Cache
ARM Architecture

Device

Emulator

Memory

I-Cache

D-Cache

old code

new code

Clean the D-Cache range
ARM Architecture

Device
- Memory
  - I-Cache
  - D-Cache

Emulator
- Memory
  - Cache

Invalidate the I-Cache

old code
new code
ARM Architecture

Device

Emulator

old code

new code
ARM Architecture

![Diagram showing ARM Architecture]

- Memory
  - I-Cache
  - D-Cache

Device

- Old code
- New code

Emulator

Run the code
ARM Architecture

Device

Emulator

Memory

I-Cache

D-Cache

Cache

Run the code

old code

new code

Thanasis Petsas
ARM Architecture

**Device**

- Memory
- I-Cache
- D-Cache

**Emulator**

- Memory
- Cache

Android `cacheflush`:
1. Clean the D-Cache range
2. Invalidate the I-Cache

---

Thanasis Petsas
ARM Architecture

Device
- Memory
  - I-Cache
  - D-Cache

Emulator
- Memory
  - Cache

- old code
- new code

Thanasis Petsas
Identify QEMU execution – xFlowH

typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}
Identify QEMU execution – xFlowH

typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANON,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);
    code_func();
}
Identify QEMU execution – xFlowH

typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}

Thanasis Petsas
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(  
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);
    cacheflush();
    code_func();
    patch_code(&swap, &patch, &f2);
    cacheflush();
    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);
    cacheflush();
    code_func();
    patch_code(&swap, &patch, &f2);
    cacheflush();
    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);
    cacheflush();
    code_func();
    patch_code(&swap, &patch, &f2);
    cacheflush();
    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch, &f2);

for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);
    cacheflush();
    code_func();
    patch_code(&swap, &patch, &f2);
    cacheflush();
    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);
    cacheflush();
    code_func();
    patch_code(&swap, &patch, &f2);
    cacheflush();
    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(  
    NULL,  
    16 * 4,  
    PROT_READ | PROT_WRITE | PROT_EXEC,  
    MAP_PRIVATE | MAP_ANONYMOUS,  
    -1,  
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}

with cacheflush:  

without cacheflush:
Identify QEMU execution – xFlowH

```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}
```
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t * patch;
uint32_t * swap;

uint32_t * code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2);

    code_func();
}

with cacheflush:

same behavior.

without cacheflush:
typedef void (*code_func_t) (void);

code_func_t code_func;
uint32_t *patch;
uint32_t *swap;

uint32_t *code = mmap(
    NULL,
    16 * 4,
    PROT_READ | PROT_WRITE | PROT_EXEC,
    MAP_PRIVATE | MAP_ANONYMOUS,
    -1,
    0);

code_func = (code_func_t) code;
write_code(&swap, &code, &patch,
for (i=0; i<N; i++) {
    patch_code(&swap, &patch, &f1);

    code_func();
    patch_code(&swap, &patch, &f2)
    code_func();
}
Implementation

• Use of Android SDK for static & dynamic heuristics

• Use of Android NDK for hypervisor heuristics

• Implementation of an Android app
  – runs the heuristics
  – send the results to an HTTP server

• Repackaging of well known Android malware samples
  – Smali/Baksmali
  – Apktool
  – Patching the Smali Dalvik Bytecode
# Evaluation: Malware Set

<table>
<thead>
<tr>
<th>Family</th>
<th>Package name</th>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BadNews</td>
<td>ru.blogspot. playsib.savageknife</td>
<td>magnFH</td>
<td>Data extrusion</td>
</tr>
<tr>
<td>BaseBridge</td>
<td>com.keji.unclear</td>
<td>accelH</td>
<td>Root exploit</td>
</tr>
<tr>
<td>Bgserv</td>
<td>com.android. vending.sectool.v1</td>
<td>netH</td>
<td>Bot activity</td>
</tr>
<tr>
<td>DroidDream</td>
<td>com.droiddream. bowlingtime</td>
<td>gyrosH</td>
<td>Root exploit</td>
</tr>
<tr>
<td>DroidKungFu</td>
<td>com.atools.cuttherope</td>
<td>rotVecH</td>
<td>Root exploit</td>
</tr>
<tr>
<td>FakeSMS Installer</td>
<td>net.mwkekdsf</td>
<td>proximH</td>
<td>SMS trojan</td>
</tr>
<tr>
<td>Geinimi</td>
<td>com.sgg.sp</td>
<td>buildH</td>
<td>Bot activity</td>
</tr>
<tr>
<td>Zsone</td>
<td>com.mj.iCalendar</td>
<td>idH</td>
<td>SMS trojan</td>
</tr>
<tr>
<td>JiFake</td>
<td>android.packageinstaller</td>
<td>BTdetectH</td>
<td>SMS trojan</td>
</tr>
<tr>
<td>Fakemart</td>
<td>com.android.blackmarket</td>
<td>xFlowH</td>
<td>SMS trojan</td>
</tr>
</tbody>
</table>

Source: [http://contagiominidump.blogspot.com/](http://contagiominidump.blogspot.com/)
Evaluation: Dynamic Analysis Services

• Stand alone tools
  – DroidBox, DroidScope, TaintDroid

• Online services
  – Andrubis, SandDroid, ApkScan, Visual Threat, TraceDroid, CopperDroid, APK Analyzer, ForeSafe, Mobile SandBox
Methodology (1/2)
Methodology (2/2)

Original Sample

1. Heuristics Incorporation

Samples Submission

2. Reports Comparison

Dynamic Analysis Service

3. Repackaged Sample

Thanasis Petsas
Resilience of dynamic analysis tools

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td>idH</td>
<td>buildH</td>
<td>nameH</td>
</tr>
<tr>
<td>DroidBox</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DroidScope</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>TaintDroid</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Andrubis</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>SandDroid</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>ApkScan</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>VisualThreat</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Tracedroid</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CopperDroid</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Apk Analyzer</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ForeSafe</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mobile Sandbox</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Thanasis Petsas
Resilience of dynamic analysis tools

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>idH</td>
<td>buildH</td>
<td>netH</td>
</tr>
<tr>
<td>DroidBox</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DroidScope</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>TaintDroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Andrubis</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SandDroid</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ApkScan</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>VisualThreat</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Tracedroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CopperDroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Apk Analyzer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ForeSafe</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mobile Sandbox</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

All studied services are vulnerable to 5 or more heuristics.
### Resilience of dynamic analysis tools

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>idH</td>
<td>buildH</td>
<td>netH</td>
</tr>
<tr>
<td>DroidBox</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>DroidScope</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>TaintDroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Andrubis</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SandDroid</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ApkScan</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>VisualThreat</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Tracedroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>CopperDroid</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Apk Analyzer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ForeSafe</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Mobile Sandbox</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

These tools failed to infer malicious behavior of the repackaged malware samples.
# Resilience of dynamic analysis tools

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
<th>Hypervisor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>idH</td>
<td>buildH</td>
<td>netH</td>
</tr>
<tr>
<td>DroidBox</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DroidScope</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>TaintDroid</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Andrubis</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SandDroid</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ApkScan</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>VisualThreat</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Tracedroid</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>CopperDroid</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Apk Analyzer</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ForeSafe</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Mobile Sandbox</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Only 1 service provides information about VM evasion attempts.
Countermeasures

• Static heuristics
  – Emulator modifications

• Dynamic heuristics
  – Realistic sensor event simulation

• Hypervisor heuristics
  – Accurate binary translation
  – Hardware-assisted virtualization
  – Hybrid application execution
Summary

• Evaluation of VM evasion to 12 Android dynamic analysis tools

• Only half of the services detected our most trivial heuristics

• No service was resilient to our dynamic and hypervisor heuristics

• Majority of the services failed to detect repackaged malware

• Only 1 service
  – generated VM evasion attempts
  – was resilient to all our static heuristics
Thank you!

Thanasis Petsas, Giannis Voyatzis, Elias Athanasopoulos, Sotiris Ioannidis,
{petsas, jvoyatz, elathan, sotiris}@ics.forth.gr
Michalis Polychronakis,
mikepo@cs.columbia.edu