Dynamic Analysis

Matthias Neugschwandtner
mneug@iseclab.org

Vienna University of Technology
Secure Systems Lab

Outline

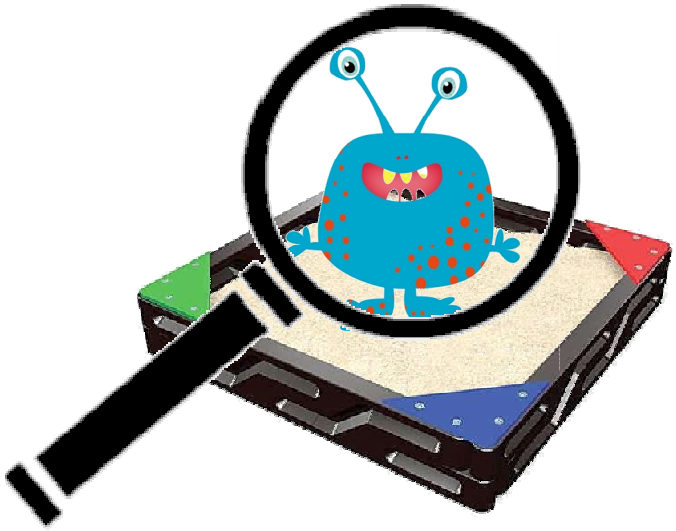
- What do we need dynamic analysis for?
- What is it to begin with?
- Why can’t we just use static analysis?
- How is it done?
- Hands-on: Andrototal, Andrubis

Motivation

- Assume you come across a suspicious APK
- First step?
  - Submit to Andrototal! (http://andrototal.org)
  - Virustotal for mobile apps andrototal (beta)
- Will tell you what AVs say about the sample
  - just a distinction between unknown/benign and malware/adware
  - opaque operation, won't tell you what it does!

Dynamic Analysis

- Actually execute a program
  - typically in a contained environment (sandbox)
- Observe behavior at runtime



Static vs. Dynamic: Information Detail

```
.line 177
    .restart local v3       #file:Ljava/io/File;
    .restart local v4       #fos:Ljava/io/FileOutputStream;
    .restart local v5       #i:I
    .restart local v6       #is:Ljava/io/InputStream;
    .restart local v7       #temp:[B
    :cond_1
    const/4 v8, 0x0

    invoke-virtual {v4, v7, v8, v5}, Ljava/io/FileOutputStream;->write([BII)V
```

Static Analysis Insight	Dynamic Analysis Insight
The app writes to a file.	The app writes to a file. Filename: malicious.zip Content: 0x7F454C46 0x…

Static vs. Dynamic: Coverage

```
invoke-virtual {v7}, Ljava/lang/String;->length()I
move-result v1
const/16 v3, 0x12
if-eq v1, v3, :cond_0
iget-object v1, p0, Lcom/example/xxshenqi/RegisterActivity
…
:cond_0
…
```

Static Analysis Coverage	Dynamic Analysis Coverage
Every instruction, regardless of the control flow	Only one execution trace through the program. Depends on environment and user input!

Static vs. Dynamic: Reflection

```
invoke-static {v0, v1, v2}, Lo/嚳$CON;->ą(III)Ljava/lang/String;
move-result-object v0
invoke-static {v0}, Ljava/lang/Class;
    ->forName(Ljava/lang/String;)Ljava/lang/Class;
move-result-object v0
const/16 v1, 0x98
const/16 v2, -0x12b
const/16 v3, -0x21
invoke-static {v1, v2, v3}, Lo/嚳$CON;->ą(III)Ljava/lang/String;
move-result-object v1
const/4 v2, 0x0
invoke-virtual {v0, v1, v2}, Ljava/lang/Class;->getMethod(Ljava/lang/String;
    [Ljava/lang/Class;)Ljava/lang/reflect/Method;
move-result-object v0
const/4 v1, 0x0
invoke-virtual {v0, v12, v1}, Ljava/lang/reflect/Method;
    ->invoke(Ljava/lang/Object;[Ljava/lang/Object;)Ljava/lang/Object;
```

hard to handle for static analysis!

Static vs. Dynamic: Encrypted Code

```
fill-array-data v0, :array_a
:array_a
.array-data 0x1,0x4t,0x4t,0x6t,0x8t,0xfat,0xe1t,0x2ct,…
const-class v0, Lo/岌$愿;
invoke-virtual {v0}, Ljava/lang/Class;
        ->getClassLoader()Ljava/lang/ClassLoader;
move-result-object v0
```

almost impossible to handle for static analysis!

Obfuscators

- Proguard
  - basic name obfuscation, shrinking, optimization
- Dexguard
  - encryption, reflection, tamper detection
- DexProtector
  - encryption, tamper detection
- APKProtect
  - native code protection

Static vs. Dynamic Analysis

	Static	Dynamic
Detail	✖	✔
Coverage	✔	✖
Reflection	✖	✔
Dynamic code loading	✖	✔
Encryption	✖	✔
Time	✔	✖

Sandbox options

- What is this “contained environment”?
- Typical setup:
  - Android emulator (qemu)
  - running Android OS
  - install & run a malware sample

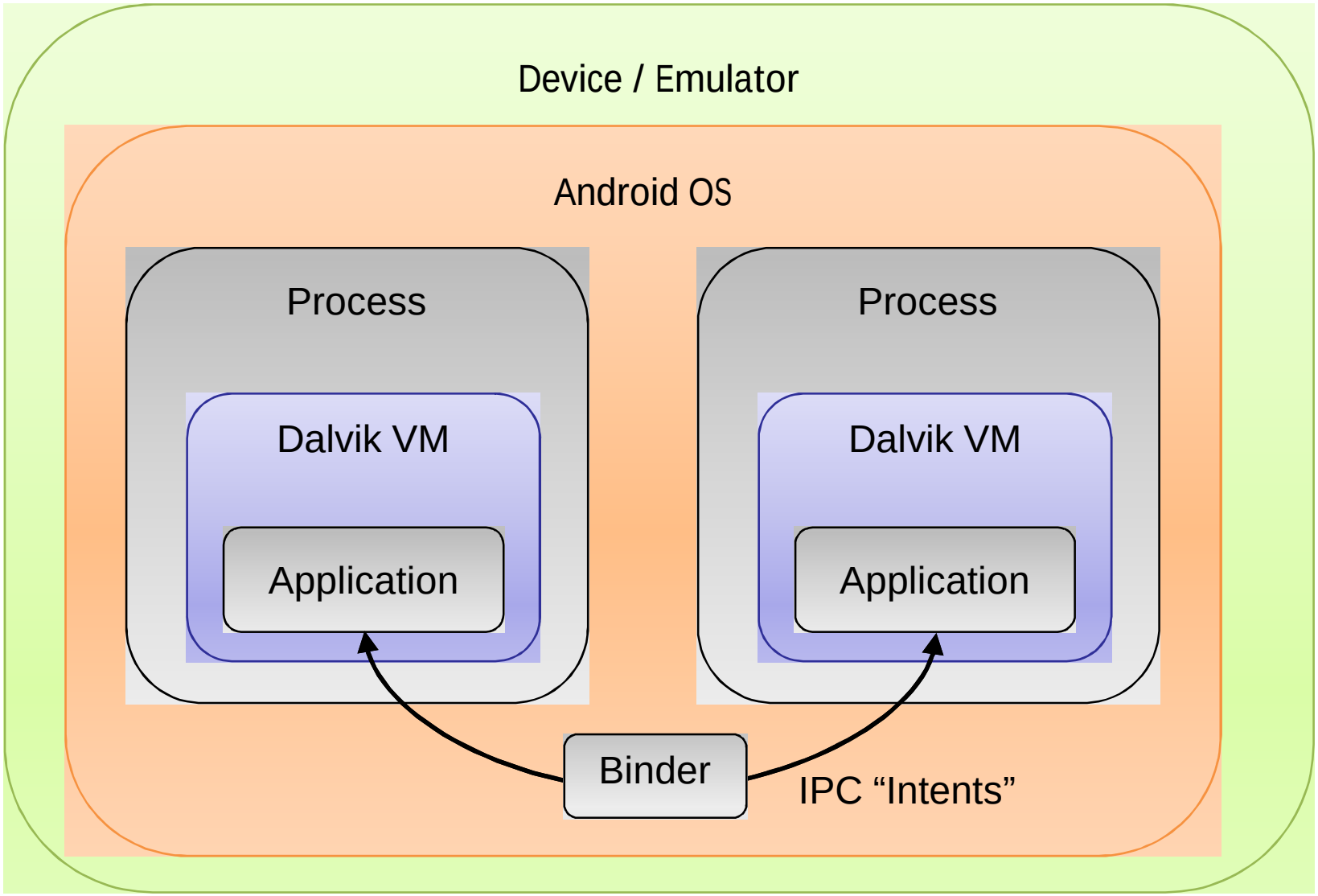
Capturing behavior

- Effect the malware has on a system
  - file operations
  - network operations
  - interaction with other apps/processes
- Specific to mobile environment
  - phone activity (calls, text messages)
  - usage of sensitive data (location, phone book)

Monitoring Options

- Code execution
  - from internal function invocations down to single instructions
  - very detailed
- Library usage
  - invocation of typical library functions
  - sufficient for capturing behavior

Android System Overview



Instrumenting the DVM

- Monitoring
  - dalvik instructions
  - function/library invocations
- High level of semantic detail
  - intrinsic notion of processes
  - aware of object types (java classes, strings, …)

Instrumenting the Emulator

- Monitoring
  - native code
    - JNI, ART, root exploits (RATC)
  - system calls
- Huge semantic gap
  - low level of semantic detail
  - requires reconstructing basic OS concepts
    - processes, kernel/user mode
    - no notion of Java objects
- Easy access to network traffic

Taint Tracking

- Basic concept
  - mark (“taint”) sensitive data
  - track flow of data from source to sink
- Raise alert if sensitive data is leaked
  - e.g. IMEI is sent over the network
- Typically implemented in the DVM
  - Taintdroid (http://appanalysis.org)
  - unable to track taint across process boundaries

Stimulation

- There is no “main” method! Apps have multiple entry points
  - activities (GUI screens, listed in manifest)
  - services (background processes, not necessarily started)
  - broadcast receivers (intent handlers)
- Apps react to “common events”
  - incoming texts, calls, GPS lock
- Apps sometimes require user input
  - e.g. TAN for a banking trojan

Analysis services

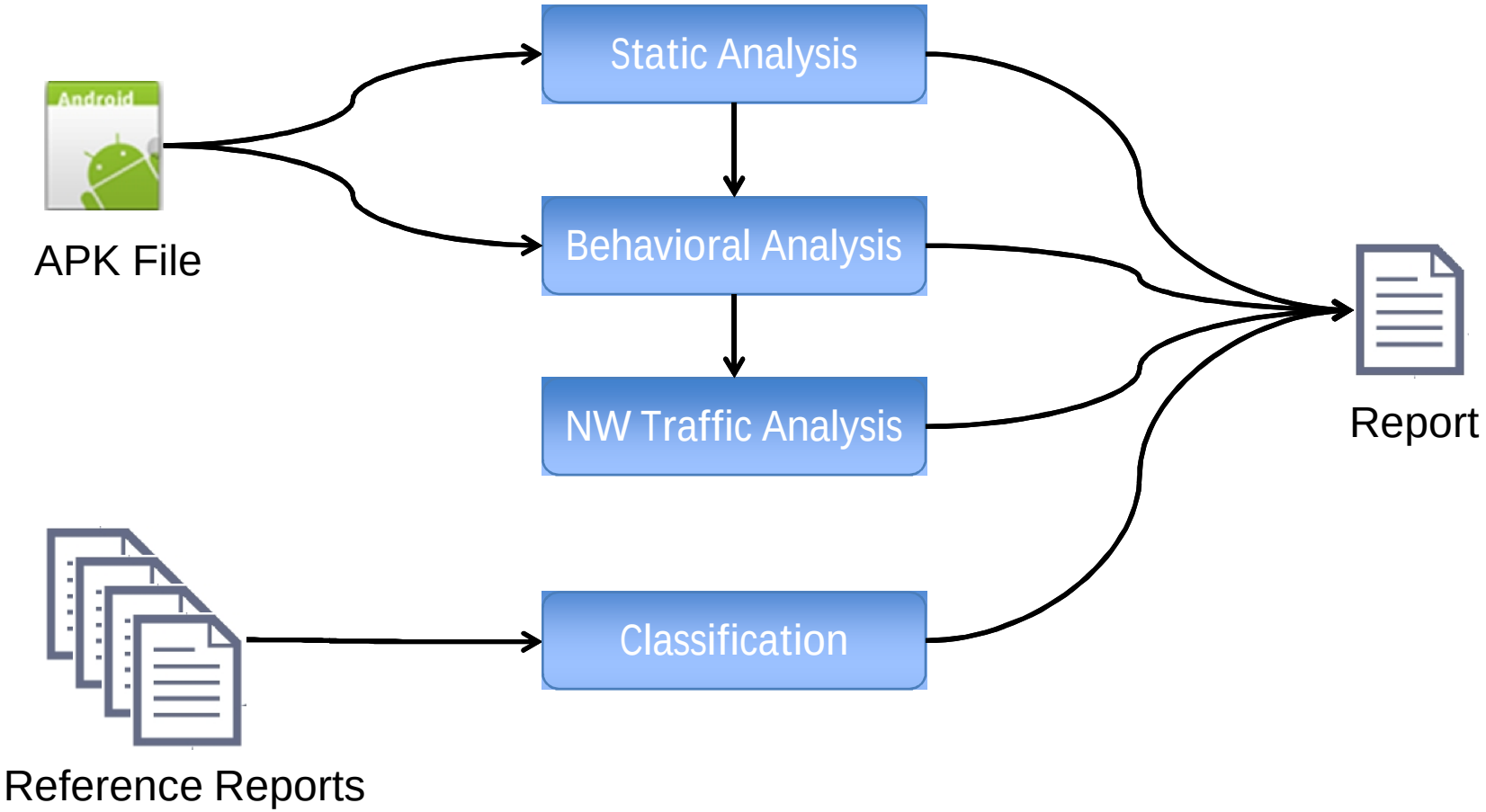
- Andrubis (http://anubis.iseclab.org)
  - bit-of-everything
  - basic static analysis
  - API usage
  - NW analysis

- Copperdroid (http://copperdroid.isg.rhul.ac.uk/)
  - focuses on native code analysis

CopperDroid

- Tracedroid (http://tracedroid.few.vu.nl/)
  - method-level execution tracing

Andrubis


Static Analysis
Behavioral Analysis
NW Traffic Analysis
Classification
APK File
Report
Reference Reports

Andrubis – Geinimi

- Let's take a look at one of those reports!
- Permissions used
  - GPS, SMS, WRITE_STORAGE
  - NW for ad library and main program
- Network operations
  - opens a socket
- Crypto operations
  - network endpoints
  - android.provider.Telephony.SMS_RECEIVED

Yet another malware sample

- Has no designated AV label
- Network C&C traffic
- Wireshark
  - _THE_ tool to inspect network traffic
  - can-opener for pcap files
  - filters protocols, network streams
  - https://www.wireshark.org/

Andrubis

- Hands-on!
- Andrototal
  - http://andrototal.org/
- Andrubis
  - http://anubis.iseclab.org
  - username: summerschool
  - password: syssec_2014
- Sample reports
  - http://bit.ly/1uH6Tna

FakeBank

- MD5 1f68addf38f63fe821b237bc7baabb3d
- File operations
  - nothing interesting
- Network operations
  - regular heartbeat to C&C
  - cleartext data leakage in POST-request

FakeAV

- MD5 7d25d4cdbf3cfc8b6e9466729b84d348
- File operations
  - writes to SD card
- Network operations
  - resolves appsota.net
  - HTTP POST requests to this host
  - taint analysis reports leaks
- Crypto
  - data to be exfiltrated is encrypted
  - IMEI + OS/device fingerprint
  - intercepted SMS

GGSmart

- MD5 f5f2c897249947a6948176edd9148397
- File operations
  - nothing interesting
- Network operations
  - android.clients.google.com – Google Play Statistics Spyware ;)
  - encrypted endpoint go.docrui.com
  - exfiltrates data in cleartext via XML file

HgSpy

- MD5 3709f87d2b6ff0bd7937112974dc1143
- File operations
  - writes ELF files!
- Dynamic code loading
  - classes.dex
  - multiple .so files NOT from the system directory
- Network operations
  - obfuscated IMEI leakage

Pletor

- MD5 236b9bd036ab89ebac2d318c605a2979
- adult.free.xxx.video.player ?
  - ransomware! ;)
- No interesting file operations
- C&C
  - POST to server + data leakage (installed apps), IMEI
  - active reply, but command field not set
- Screenshot
  - would like to have admin rights

Krysanec

- MD5 9315f9ff9e88a0c3ac6c3186661bec2e
- File operations
  - writes an ELF file
  - executes commands!

```
$ (echo -17 > /proc/528/oom_adj) &> /dev/null
$ stat /sbin/su echo F*D^W@#FGF 0 $?
stat: permission denied F*D^W@#FGF 0 127

$ ls /sbin/su echo F*D^W@#FGF 8 $?
/sbin/su
```

SmsFw

- MD5 71e81a47080f809452ca524a85ec428f
- No interesting file/NW operations
- Text messages
  - uses SMS as C&C channel
  - registration
  - forwards incoming text + number

SmsReg

- MD5 086c7d9292d9adaaf70f08ad33541542
- File operations
  - writes ELF file
  - loads it as a native library later on
- Network operations
  - C&C traffic
  - fire up Wireshark ;)