System Calls Monitoring in Android:

An Approach to Detect Debuggers, Anomalies and Privacy Issues

Master's Degree programme in Computer Science

Final Thesis

Academic Year 2021-2022

Graduand
Matteo De Giorgi
Matriculation Number: 872029

Supervisor
Prof. Paolo Falcarin

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

The problem

- Android applications have a vast potential user base and access to sensitive data
- Factor combination that attracts the attention of many malicious actors.

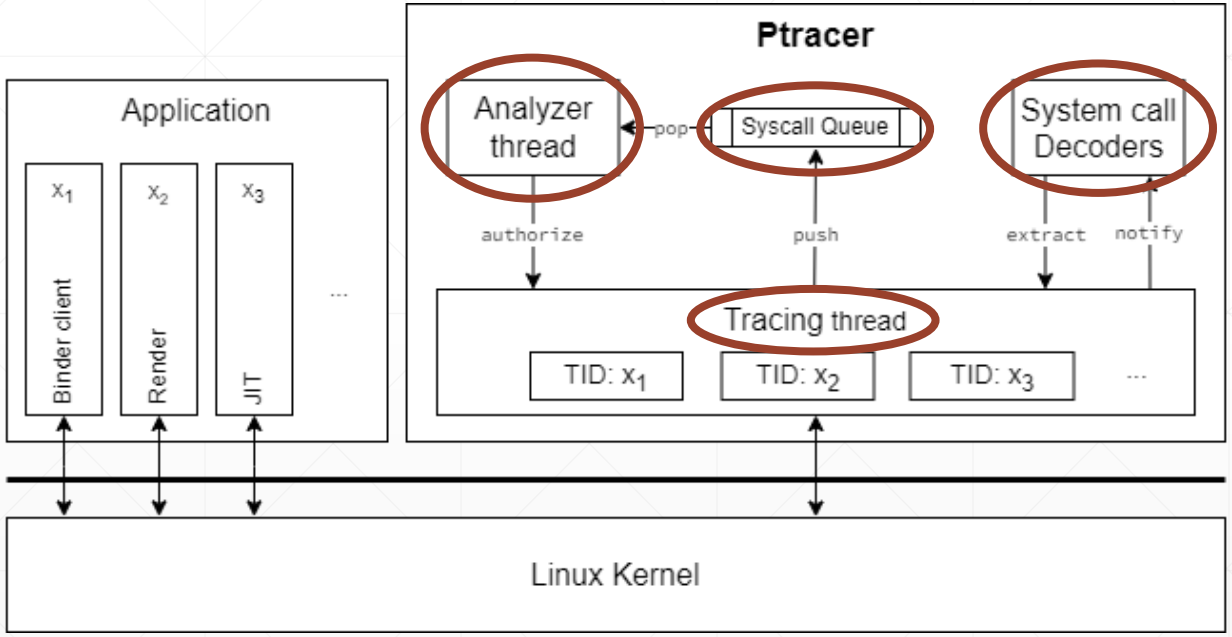
- There is a necessity to prevent or detect:
  - MATE Attacks using Debuggers, e.g., to modify the application's intended behaviour
  - Anomalies, e.g., the exploitation of a vulnerability
  - Privacy issues, e.g., unusual accesses to various sensors like the microphone

Related Work

- Debugger detection
  - Self-Debugging (B. Abrath et al., 2016)
  - OWASP Mobile Application Security Testing Guide (MASTG)
  - Remote detection via sensors data
- Anomaly detection based on System Calls
  - Dynamic analysis: VtPaths (H. Feng et al., 2003)
  - Static and Dynamic analysis: Execution Graphs (D. Gao et al., 2004)
- Privacy Issues
  - Virtualization based: CopperDroid (K. Tam et al., 2015), DroidScope (L. Kwong et al., 2012)
  - Static and Dynamic analysis: ProfileDroid (X. Wei et al., 2012)

Solution

- "*Ptracer*" was developed:
  - Capture every system call of a process
  - Extract stack traces for each system call
  - Learn and enforce a model of the observed behaviour
  - Deeply inspect the parameters passed to the system calls



Ca' Foscari University of Venice

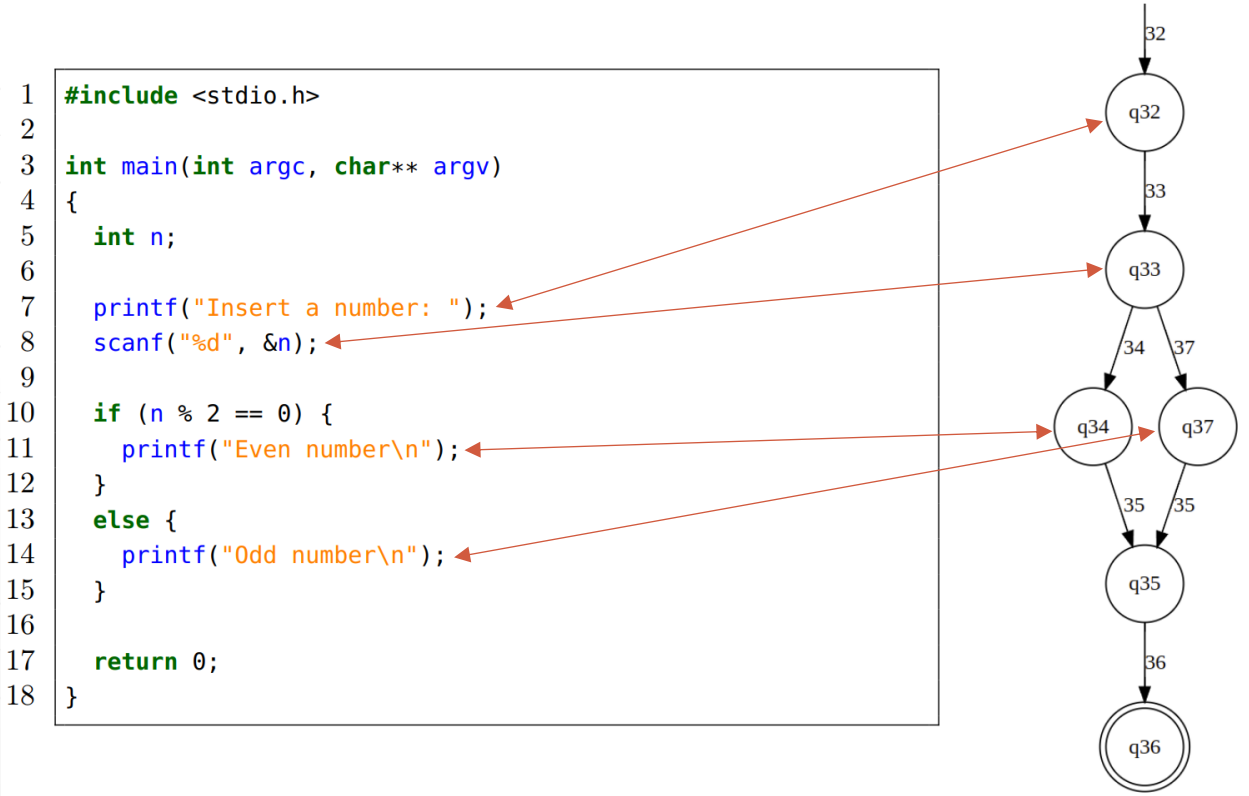
System Calls and Stack Traces

- For every system call:
  - Timing
  - System call number
  - Stack backtrace
  - The parameters passed to system call
  - The CPU registers

```
------------------ SYSCALL ENTRY START ------------------
Notification origin: it.matteodegiorgi.audiorecorder
PID: 14711
SPID: 14711
Timestamp: 1673561963897680
Syscall = ioctl (29)
Stack unwinding = {
  PC 0x000079d54b34d8 … - __ioctl @ 8
  PC 0x000079d546f2f4 … - ioctl @ 152
  PC 0x000079ca76e7ac … - android::IPCThreadState::talkWithDriver(bool) @ 288
  …
  PC 0x000079dc6fed3c … - it.matteodegiorgi.audiorecorder.MainActivity.startRecording @ 0
  …
  PC 0x000079d9c1bfb4 … - android::AndroidRuntime::start(…) @ 836
  PC 0x000057c299858c … - main @ 1336
  PC 0x000079d545e7dc … - __libc_init @ 96
}
Parameters = {
  0x000000000000003c
  0x00000000c0306201
  0x0000007fcfedfb68
  …
}
Registers = {
  PC: 0x00000079d54b34d8
  SP: 0x0000007fcfedfa50
  RET: 0x000000000000003c
}
------------------ SYSCALL ENTRY STOP ------------------
```

NFA Model

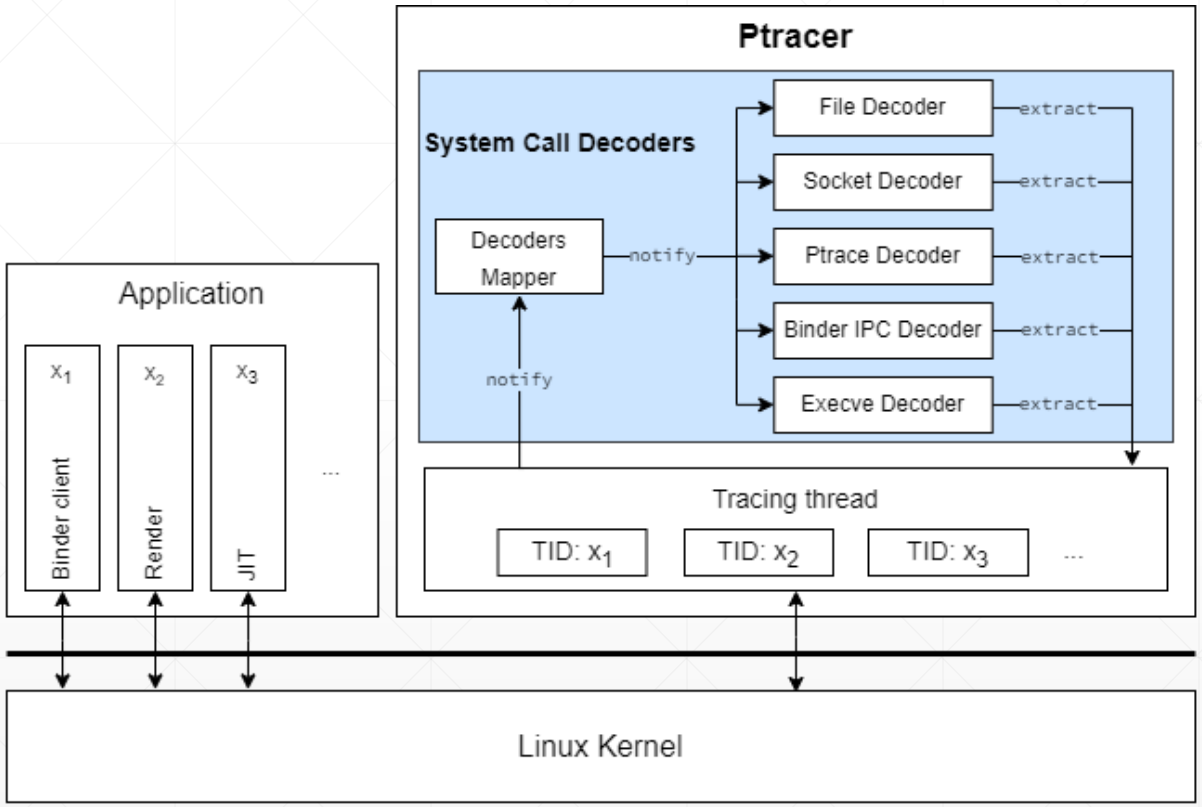
- Every state is an integer mapped to a 3-tuple:
  - Executable name
  - System call number
  - Stack trace
- A state is marked as final if the process terminated when on it



System Call Decoders

- Provide deep inspection of meaningful system calls

- Roles:
  - Intercept and capture communications to/from file descriptors
  - Read the destination addresses of network communications
  - Read communications with the Android Binder IPC
  - More…

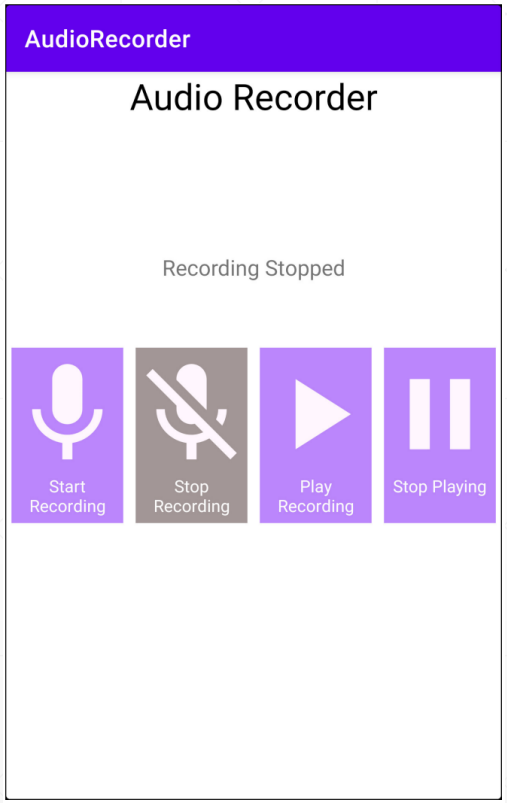


Ca' Foscari University of Venice

Validation

- Can Ptracer be used on Android applications for:
  - Anomaly detection
  - Debugger detection
  - Identifying privacy issues

- A sample application was created: AudioRecorder
  - Contains an Unsafe Deserialization vulnerability triggered by a malicious configuration file



Validation - Anomaly Detection

Learning

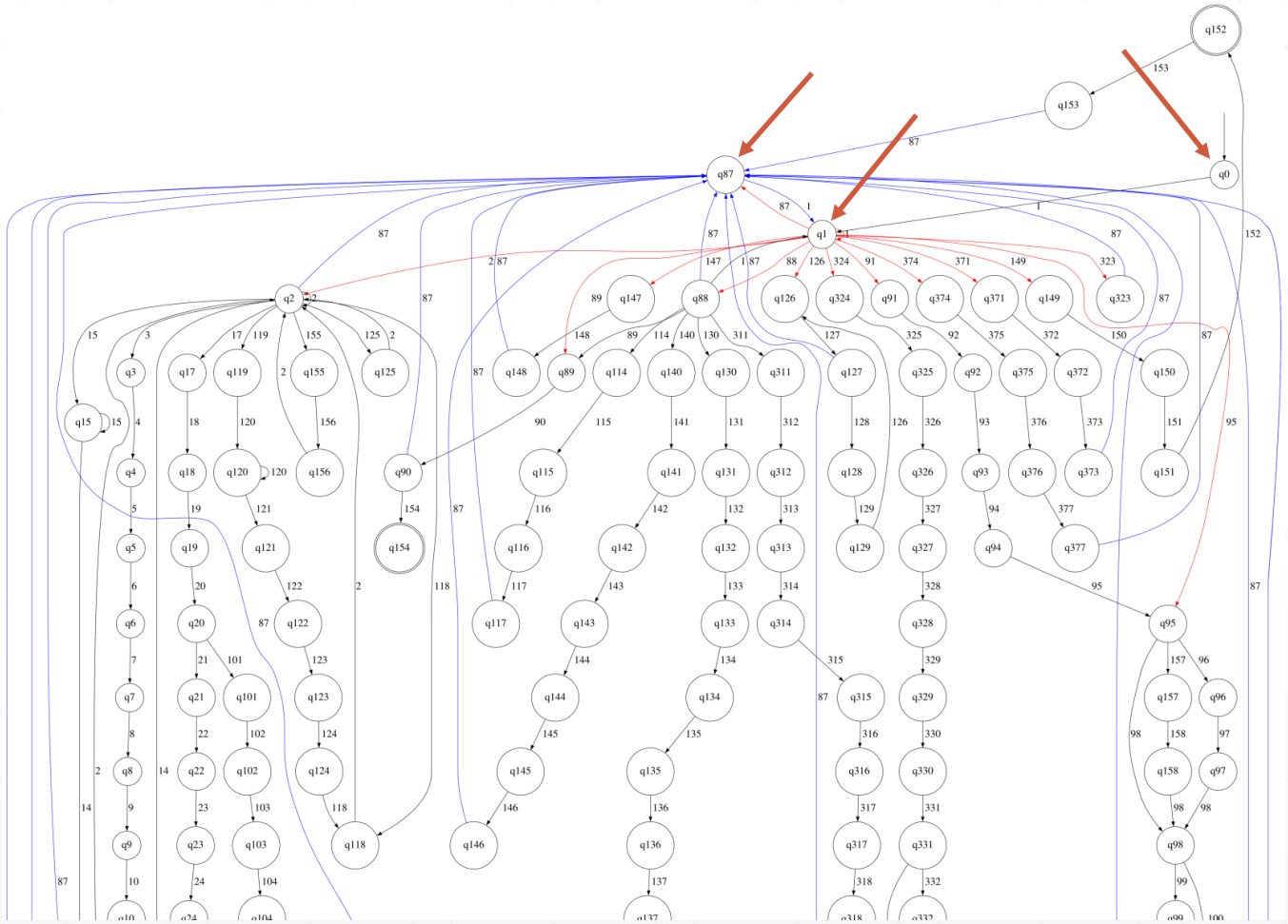
1. Learn the NFA model through multiple learning iterations
2. Once the model is stable, switch to enforce mode

Enforcing

3. Test if there are false positives during multiple executions
4. Test if the Unsafe Deserialization exploit can be detected and prevented

Learning phase

- Final automaton in numbers:
  - 32 learning iterations
  - 1274 states
  - 1594 transitions
  - 14 final states



Ca' Foscari University of Venice

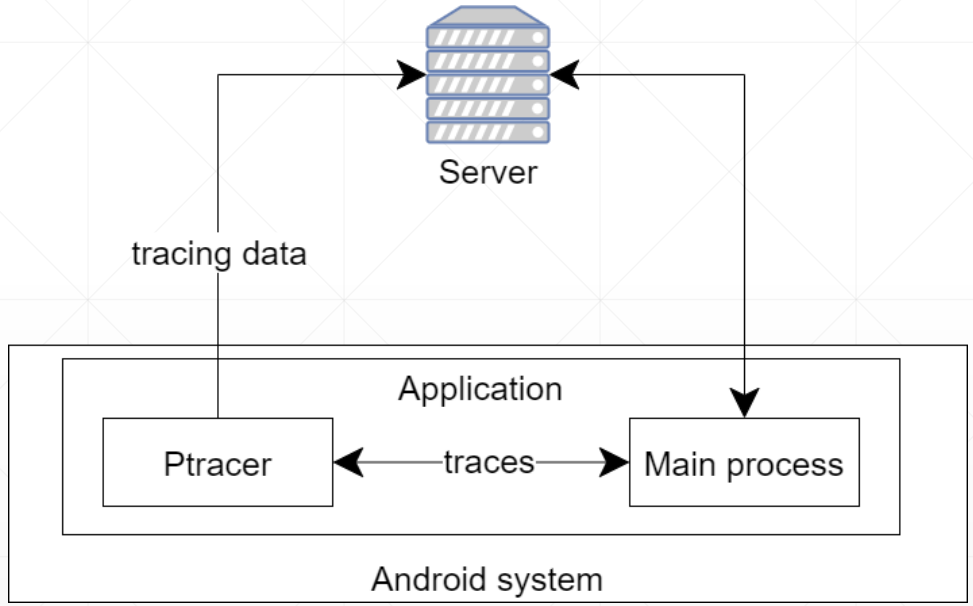
Enforcing phase

- The model proved well-trained for the normal application execution
- False positives emerged after many subsequent executions when re-opening the application
- The Unsafe Deserialization exploit was correctly prevented

```
------------------ SYSCALL ENTRY START ------------------
Notification origin: it.matteodegiorgi.audiorecorder
PID: 28738
SPID: 28738
Timestamp: 1676751181776975
Syscall = read (63)
Stack unwinding = {
PC 0x0000007c4d0f4238 … - read @ 8
PC 0x0000007986469b6c … - Linux_readBytes(_JNIEnv*, …, int, int) @ 172
  …
}
Parameters = { … }
Registers = { … }
------------------ SYSCALL ENTRY STOP ------------------
There are no possible transitions from q57 to q64
System call NOT authorized
Warning! Encountered a transition that has never been observed before!
Possible actions:
1 - Kill the target process
2 - Allow it by adding a new transition in the model
Choice:
```

Validation - Debugger Detection

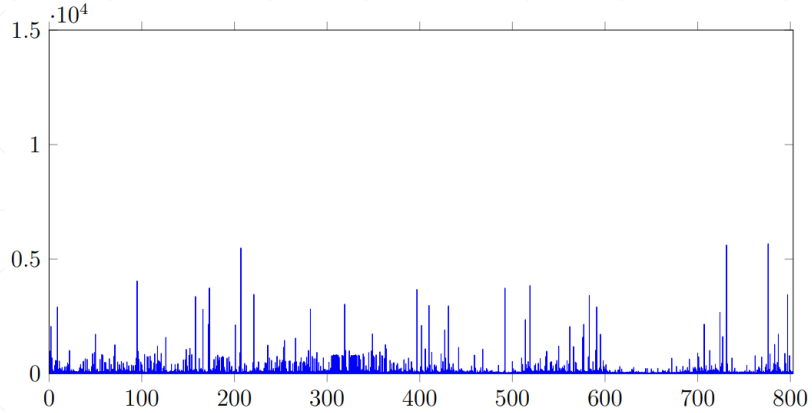
- *Ptracer* is embedded into the application
- The server requires a stream of legitimate tracing data to consider the device genuine



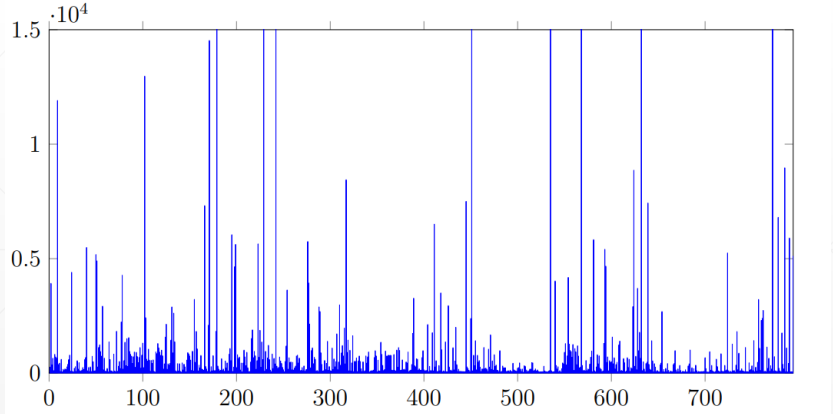
- *Can a Java debugger be detected as an anomaly?*
- *Does a debugger cause significant delays between system calls?*

Debugger Detection results

- The Java debugger causes alterations in the stack traces → it is immediately detected as an anomaly
- The time spent between system calls is more than doubled on average when a Java debugger is present
- *Ptracer* is a good fit for detecting debuggers in the proposed architecture



(a) Plot showing time elapsed between system calls with no Java debugger attached



(b) Plot showing time elapsed between system calls when a Java debugger is attached

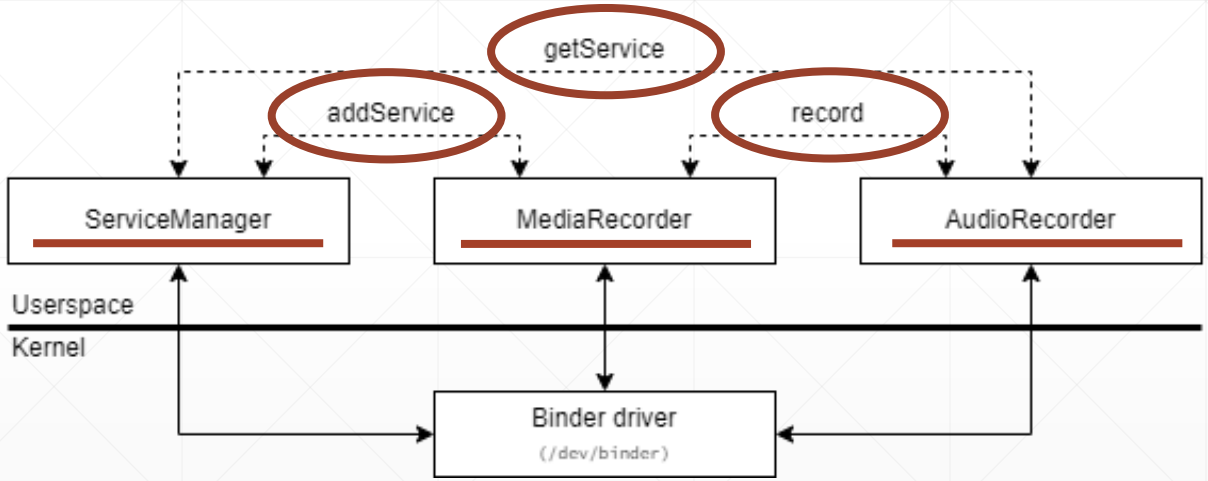
Validation - Privacy issues

- Goal: Validate that *Ptracer* can detect privacy-sensitive actions

- Validation objectives:
  1. Validate that it is possible to see when the microphone is accessed
  2. Detect that a new file is written into the file system
  3. There are no internet connections nor suspicious activities

Android landscape

- Android applications are based on events (e.g., a touch on the screen) and communicate through the Binder IPC
- The Binder can be considered the "heart" of the system
- *AudioRecorder* will interact with the MediaRecorder service to be able to access the microphone



Ca' Foscari University of Venice

Privacy issues on AudioRecorder

- *Ptracer* was able to intercept the RPC calls to the ServiceManager and MediaRecorder
- It was possible to determine all the files accessed by *AudioRecorder*
- No network communications were identified

```
------------------ FILE DECODER START ------------------
…
File Descriptor: 96 <--->
/storage/emulated/0/Android/data/it.matteodegiorgi.audiorecorder/files/audio/AudioRecording.3gp
…
------------------ FILE DECODER STOP ------------------
```

```
------------------ BINDER CALL START -------------------
Sent:
  Protocol: 0x40406300 (BC_TRANSACTION)
  Target: 0x32 (50)
  Code: 16
  Flags: 16 (TF_ACCEPT_FDS)
  Buffer pointer: 0x73af054ac0 (496857598656), Data size: 100
  Buffer content:
    …
  Interface: android.media.IMediaRecorder
  Offsets pointer: 0x735f07ba00 (495515580928), Offsets size: 8
  Offsets content: 0x735f07ba00: 4c 00 00 00 00 00 00 00           L.......
  Offset 0:
    Type: 0x66642a85 (BINDER_TYPE_FD)
    File Descriptor: 96
…
------------------ BINDER CALL STOP -------------------
```

Ca' Foscari University of Venice

Privacy issues on Instagram

- It was possible to trace *Instagram* with a noticeable slowdown
- More than 90.000 system calls and 1300 Binder interactions in 3 minutes

```
package android.location;

…

interface ILocationManager {
  @nullable Location getLastLocation(String provider,
                                     in LastLocationRequest request,
                                     String packageName,
                                     @nullable String attributionTag);
…
}
```

- The application acquired the device location
- It communicated an "Analytics ID" to the *Facebook* application

```
------------------ BINDER CALL START ------------------
Sent:
  Protocol: 0x40406300 (BC_TRANSACTION)
  Target: 0x43 (67)
  Code: 1
  Flags: 16 (TF_ACCEPT_FDS)
  Buffer pointer: 0x73ef136890 (497932265616), Data size: 164
  Buffer content:
    …
  Interface: android.location.ILocationManager
…
------------------ BINDER CALL STOP -------------------
```

Future Developments

- Using eBPF probes instead of the Linux process tracing interface
- Improve system call decoders and implement new ones
- Develop a server for analysing *Ptracer*'s data for debugger detection
- Model improvements:
  - Develop annotations to trace only the meaningful program sections
  - Implement an alternative model type based on learned rules (e.g., always write in this folder)
  - Include also system call parameters in the model → prevent logic attacks
  - Add a statistical component in the model → detect DoS attacks

Conclusions

- *Ptracer* proved itself valuable for monitoring and analysing Android applications:
  - It was able to prevent an unsafe deserialization exploit
  - Can detect Java debuggers
  - Can identify privacy-sensitive actions

- Future developments will:
  - Extend the data captured in the model
  - Implement a server for the debugger detection architecture
  - Improve the speed and insight gained by the system call decoders