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

Thanasis Petsas,[1] Giannis Voyatzis,[1] Elias Athanasopoulos,[1] Michalis Polychronakis,[2] and Sotiris Ioannidis [1]

[1] Institute of Computer Science, Foundation for Research and Technology—Hellas, Greece
[2] Columbia University, USA

{petsas,jvoyatz,elathan,sotiris}@ics.forth.gr, mikepo@cs.columbia.edu

Motivation

- Android anti-virus products that offer real-time protection to mobile users can be evaded through transformation techniques[2]
- There exist many tools and web services that dynamically analyze Android apps in order to detect *zero-day* malware and enhance anti-virus capabilities
- Can these *dynamic analysis* tools also be evaded?
- How can we protect these tools from evasion techniques?

Anti-analysis Techniques

Static Heuristics

Checking pre-initialized static information

- Device ID (**idH**)
- Current build (**buildH**)
- routing table (**netH**)

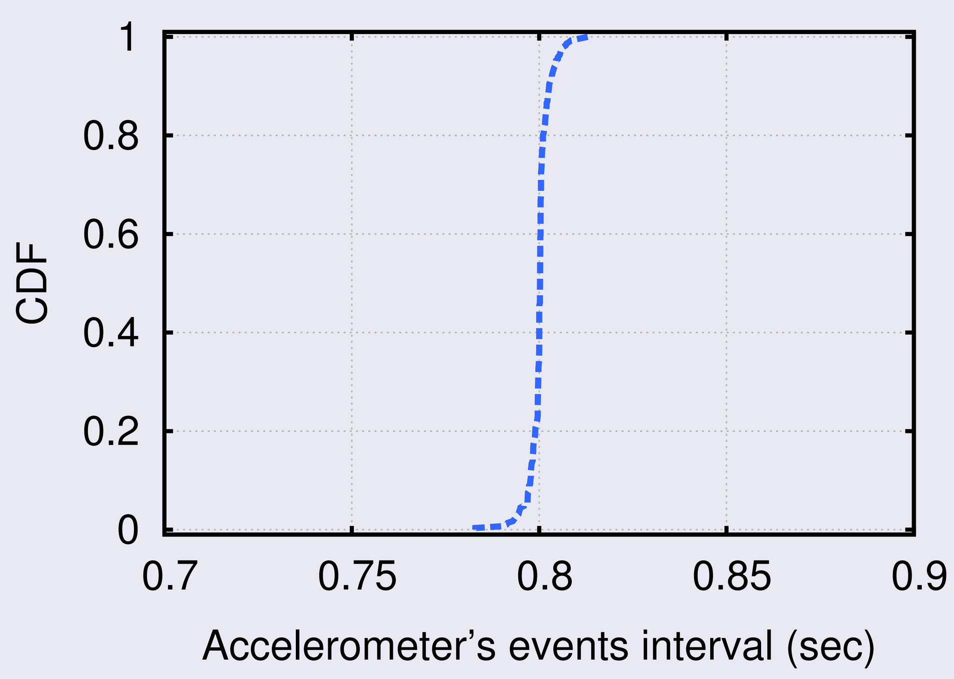
Examples

- IMEI, GSMI, etc
  - By default IMEI=`null` in Android Emulator
- Fixed `Build` attributes
  - `PRODUCT=google_sdk`
  - `HARDWARE=goldfish`
- Android Emulator behind a virtual router
  - addresss space: 10.0.2/24

Dynamic Heuristics

Sensors produce always the same values at equal intervals

- accelerometer (**accelH**)
- magnetic field (**magnFH**)
- rotation vector (**rotVecH**)
- proximity (**proximH**)
- gyroscope (**gyrosH**)



Hypervisor Heuristics

Cases where native code runs differently

- Identifying QEMU scheduling (**BTdetectH**)
- Identifying QEMU caching behavior (**xFlowH**)

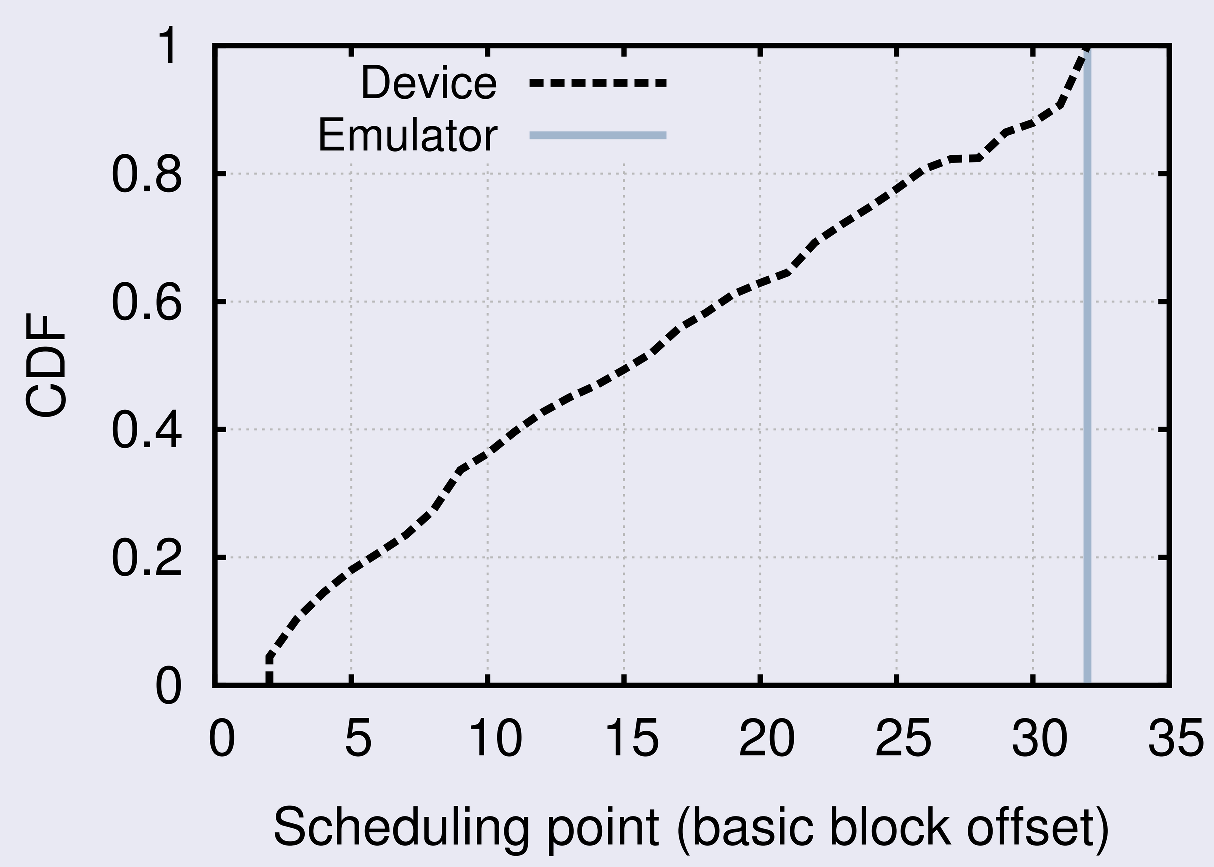
BTdetectH [1]

- QEMU optimization: Virtual PC is updated only after `branch`
- Device: Various scheduling points
- Emulator: A unique scheduling point

xFlowH

- QEMU does not emulate the ARM split cache

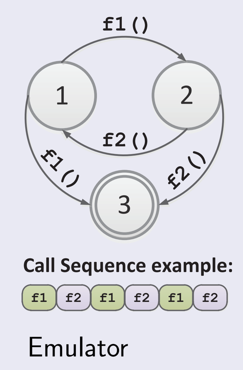
BTdetectH Heuristic Effectiveness

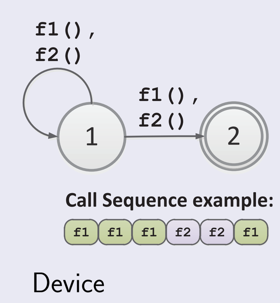


Due to optimizations many of the scheduling events that can take place are not exhibited on an emulated environment.

xFlowH

- Self-modifying code
- **Device:** random call sequence
  - D-Cache and I-Cache: Not synchronized ⇒ I-Cache may contain stale instructions
- **Emulator:** consistent call sequence
  - QEMU does not emulate the ARM cache
  - code in cache always matches the code in memory



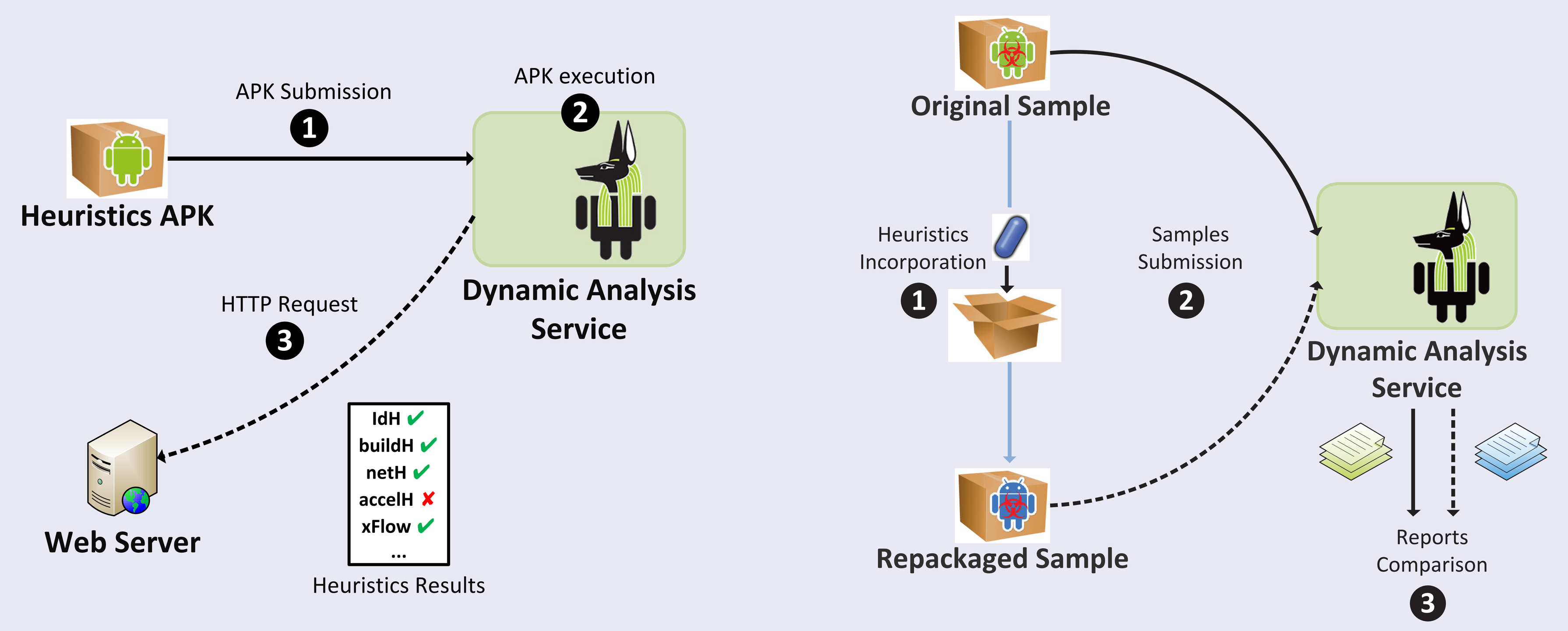



```
typedef void (*code_func_t) (void);

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();
}
```

Implementation

- Heuristics implementation: Use of Android SDK and NDK
- Android app that reports the effectiveness of the heuristics
- Incorporation of the heuristics in known Android malware samples
  - Patch the Dalvik bytecode with the bytecode of the heuristics
  - Use of Smali/Baksmali and Apktool for disassembling and reassembling

Evaluation Methodology



Evasion Results

	idH	buildH	netH	accelH	magnFH	rotVecH	proximH	gyrosH	BTdetectH	xFlowH
DroidBox	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
DroidScope	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
TaintDroid	✗	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
Andrubis	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
SandDroid	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗
ApkScan	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
VisualThreat	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
Tracedroid	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
CopperDroid	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
ApkAnalyzer	✓	✓	✓	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*
ForeSafe	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
M. Sandbox	✓	✗	✗	✗	✗	✗	✗	✗	*JNI NS*	*JNI NS*

Countermeasures

- Emulator Modifications
- Realistic Sensor Event Simulation
- Accurate Binary Translation
- Hardware-Assisted Virtualization
- Hybrid Application Execution

References

[1] Felix Matenaar and Patrick Schulz.
Detecting Android Sandboxes.
http://www.dexlabs.org/blog/btdetect.

[2] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang.
Droidchameleon: evaluating android anti-malware against transformation attacks.
In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS, 2013.

This work was supported in part by the FP7 projects NECOMA, OPTET, SysSec and by the FP7-PEOPLE-2010-IOF project XHUNTER, funded by the European Commission under Grant Agreements No. 608533, No. 317631, No. 254116 and No. 273765.

FORTH Institute of Computer Science

SEVENTH FRAMEWORK PROGRAMME