Regulating Smart Devices in Restricted Spaces

Vinod Ganapathy
vinodg@cs.rutgers.edu
Associate Professor of Computer Science
Rutgers, The State University of New Jersey

Appears in Proc. ACM MobiSys’16
Devices are everywhere!
Number of devices is increasing

- Predicted 1.2 billion new smartphones by 2018
- Predicted 50% device use increase year over year in enterprise sector until 2018 [Gartner 2014]
Devices are increasingly capable

<table>
<thead>
<tr>
<th>Model</th>
<th>CPU (GHz)</th>
<th>Screen (1000x)</th>
<th>Rear camera</th>
<th>Front camera</th>
<th>Battery (mAh)</th>
<th>Sensors other than Camera/Microphone</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>0.4</td>
<td>153</td>
<td>2MP</td>
<td>-</td>
<td>1,400</td>
<td>3 (light, accelerometer, proximity)</td>
</tr>
<tr>
<td>iPhone3</td>
<td>0.6</td>
<td>153</td>
<td>3MP</td>
<td>-</td>
<td>1,150</td>
<td>4 (+= compass)</td>
</tr>
<tr>
<td>iPhone4</td>
<td>0.8</td>
<td>614</td>
<td>5MP</td>
<td>0.3MP</td>
<td>1,420</td>
<td>6 (+= gyroscope, infrared)</td>
</tr>
<tr>
<td>iPhone5</td>
<td>1.3</td>
<td>727</td>
<td>8MP</td>
<td>1.2MP</td>
<td>1,560</td>
<td>7 (+=fingerprint)</td>
</tr>
<tr>
<td>iPhone6</td>
<td>2.0</td>
<td>1000</td>
<td>12MP</td>
<td>5.0MP</td>
<td>1,715</td>
<td>8 (+= barometer)</td>
</tr>
</tbody>
</table>
With great power …

… comes great responsibility
How can devices be misused?

1. Malicious end-users can leverage sensors to exfiltrate or infiltrate unauthorized data

2. Malicious apps on devices can achieve similar goals even if end-user is benign
Government or corporate office

• **Problem**: Sensitive documents and meetings can be ex-filtrated using the camera, microphone and storage media

• **Current solution**: Physical security scans, device isolation
Challenge: Bring your own device

Growing BYOD Trends

2013:
- SMBs supporting BYOD will increase by 14%
  - 2012: 59%
  - 2013: 73%

2014:
- Number of connected devices: 3.3/employee
- Gartner predicts 90% of companies will allow BYOD

Employee tablet use will see a year-to-year increase of 50%.

1.2 billion smartphones will enter the market in the next 5 years.

2018
Classroom and exam setting
Classroom and exam setting

- **Problem**: Personal devices can be used to infiltrate unauthorized information

[**NY Times July 2012**]

*At Top School, Cheating Voids 70 Pupils’ Tests*

By AL. BAKER  JULY 9, 2012

Seventy students were involved in a pattern of smartphone-enabled cheating last month at Stuyvesant High School, New York City officials said Monday, describing an episode that has blemished one of the country’s most prestigious public schools.

[**Financial Crypto 2014**]

Outsmarting Proctors with Smartwatches: A Case Study on Wearable Computing Security

Alex Micicovsky, Zakir Durumeric, Jeff Ringerberg, and J. Alex Halderman

---

Scanners catch JEE cheats

Roving invigilators armed with signal scanners to detect mobile data and call traffic inside examination halls caught five JEE candidates using a smartphone or a smartwatch to cheat on the first day of the test.
Classroom and exam setting

• **Current solution**: Deterrence via rules and threats. Invigilation to ensure compliance.

---

**NO MOBILE PHONES, iPODS, MP3/4 PLAYERS.**

**NO PRODUCTS WITH AN ELECTRONIC COMMUNICATION/STORAGE DEVICE OR DIGITAL FACILITY.**

Possession of unauthorised items is an infringement of the regulations and could result in **DISQUALIFICATION** from the current examination and the overall qualification. Candidates are advised that mobile phones in particular must not be in their possession whether switched on or not.

This poster must be displayed in a prominent place both inside and outside each examination room.
Challenge: Assistive devices

• Students may wish to use devices for legitimate reasons:
  – Smart glass or contacts for vision correction
  – Bluetooth-enabled hearing aids
  – Smart watches to monitor time
Other social settings

• Restaurants, conferences, gym locker rooms, private homes, ...

• Problems:
  – Recording private conversations
  – Pictures of individuals taken and posted to social networks without their consent
  – Pictures and videos of otherwise private locations, e.g., private homes
Other social settings

• **Current solutions**: Informal enforcement

• **Challenge**: Social isolation 😊

“For the first time ever this place, Feast, in NYC just asked that I remove Google Glass because customers have complained of privacy concerns […] I left”

---

“NOTICE
No wearable cameras permitted in locker room.”
Malicious apps exploiting sensors

Sensory malware

Early example of sensory malware [CCS 2011]

- Use accelerometer and record keystroke press vibrations
- Up to 80% accuracy in word recovery
Malicious apps exploiting sensors

Sensory malware

• Attacks have now been demonstrated using every imaginable sensor
• Attack accuracy will *improve* with each generation of devices and sensors
Claim
Smart devices will become integrated with daily lives → Ad hoc solutions, e.g., banning device use, will no longer be acceptable

Vision
Need systematic methods to regulate devices and ensure responsible use

Discussion: Only considering overt device use. Covert use detection still requires traditional physical security measures.
What solutions exist today?

Mobile device management (MDM) solutions

KNOX Workspace

Supported MDMs

- Absolute Software: Supports 90 of 142 KNOX Workspace Features
- AirWatch: Supports 100 of 142 KNOX Workspace Features
- BlackBerry: Supports 103 of 142 KNOX Workspace Features
- CA Technologies: Supports 83 of 142 KNOX Workspace Features
- Centrify: Supports 122 of 142 KNOX Workspace Features
- Citrix: Supports 76 of 142 KNOX Workspace Features
- F5: Supports 102 of 142 KNOX Workspace Features
- Good: Supports 0 of 142 KNOX Workspace Features
Mobile device management

• Solution for enterprises that offer *Bring your own device* (BYOD) models
• Employees are given a mobile device outfitted with a secure software stack
• Enterprise policies “pushed” to device when employee changes device persona
Mobile device management

Main shortcoming of current MDM solutions

- Enterprise must trust software stack on guest device to enforce policies correctly
- But guest devices under control of possibly malicious end-users

- Solution for enterprises that offer *Bring your own device* (BYOD) models
- Employees are given a mobile device outfitted with a secure software stack
- Enterprise policies “pushed” to device when employee changes device persona
Contributions of our work

• **Restricted space**: Location owned by a host, where guest devices must follow the host’s usage policies

• Enable guest devices to prove policy compliance to restricted space hosts

• Use a simple, low-level API that reduces size of trusted computing base on guest devices
Key technical challenges

1. Guest devices are under the control of possibly malicious end-user
   - **Solution**: Use trusted hardware on guest device

2. What constitutes proof of compliance?
   - **Solution**: Send guest device configuration, showing policy compliance, to host

3. Doesn’t that compromise guest device privacy?
   - **Solution**: Allow guest to vet all communication to and from the host
Threat model

• **Trusted hardware on guest devices:**
  – Guest devices equipped with ARM TrustZone

• **Hosts and guests are mutually-distrusting:**
  – Hosts do not trust end-user of guest device or its end-user software stack
  – Guests do not trust host’s *reconfiguration requests* to ensure policy compliance

• **Guest devices are used overtly:**
  – Host must still use traditional physical methods to detect covert device use
Guest device check-in

Public space

Restricted space
Guest device check-in

Public space

->

Restricted space
Mutual authentication

Host’s policy server

Mutual authentication

Restricted space
Host requests guest analysis

Host’s policy server

Request device memory

Addr1, Addr2, Addr3, …

Restricted space
Guest vets host’s request

Forward host’s request
Addr1, Addr2, Addr3, …

Host’s policy server

Request device memory
Addr1, Addr2, Addr3, …

Restricted space
Guest vets host’s request

Guest’s vetting service

29

Host’s policy server

Restricted space

or

Guest’s vetting service

2

Guest’s vetting service

Restricted space

Host’s policy server
Host analyzes guest device

Guest’s vetting service

Host’s policy server

Send device memory

Addr1, Addr2, Addr3, …

Restricted space
Host pushes policy to guest

Host’s policy server

Send memory updates

Restricted space
Guest vets host’s updates

Forward host’s requested updates

Host’s policy server

Send memory updates

Restricted space

Guest’s vetting service

Forward host’s requested updates

Send memory updates

Restricted space
3

Guest applies host’s updates

Host’s policy server

Apply memory updates

Guest’s vetting service

Restricted space
Host requests proof

Guest’s vetting service

Host’s policy server

Request proof of policy compliance

Restricted space
Guest sends proof

Host’s policy server

Verification token

Restricted space

Guest’s vetting service
Guest device check-out

Revert changes

Public space

Restricted space
Operational details

1. How can host trust guest to apply policy?
   - **Answer**: Leverage ARM TrustZone

2. Why memory snapshots and updates?
   - **Answer**: Powerful low-level API. Reduces TCB

3. How does vetting service ensure safety?
   - **Answer**: Simple, conservative program analysis

4. Can’t guest device simply reboot to undo?
   - **Answer**: REM-suspend protocol
The ARM TrustZone

Guest device

Normal world
(Untrusted)

Normal-world memory

Secure world
(Protected by H/W)

Secure-world memory

ARM TrustZone
Secure boot protects secure world

Normal world
(Untrusted)

Normal-world memory

Secure world

Secure boot

Secure-world memory

ARM TrustZone
Secure world stores keys

Normal world
(Untrusted)

Normal-world memory

Secure world

Secure boot
PubKeyG, PrivKeyG

ARM TrustZone
Memory is partitioned

Normal world (Untrusted)

Normal-world memory

Secure world

Secure boot

PubKeyG, PrivKeyG

ARM TrustZone
Memory is partitioned

Normal world (Untrusted)

Secure world

Secure boot
PubKeyG, PrivKeyG

Normal-world memory

Secure-world memory

ARM TrustZone
We enhance the secure world

Normal world (Untrusted)

Secure world (booted securely)

1. Authentication
2. NW analysis
3. NW updates
4. Verif. tokens

ARM TrustZone

Normal-world memory

Secure-world memory
Mutual authentication

Goal
Establish shared session key $k_s$ between host and guest

Host’s policy server

Secure world

$\text{Secure-world memory}$

ARM TrustZone
Establishing session key

Simplified TLS/SSL handshake

- Host’s keypair: PubKeyH, PrivKeyH
- Guest’s keypair: PubKeyG, PrivKeyG

1. Guest ↔ Host: Exchange/verify public keys
2. Host → Guest: $\text{Enc}_{\text{PubKeyG}}(k_s) + \text{Signature}_{\text{PrivKeyH}}$
3. Guest (secure world): Verify host signature, decrypt message and obtain $k_s$
Guest device analysis

Request device memory
Addr1, Addr2, Addr3, ...

Send device memory
Addr1, Addr2, Addr3, ...

Host’s policy server

$K_s$

Restricted space

$K_s$
SW reads NW memory

Normal world
(Untrusted)

Secure world

Normal-world memory

Secure-world memory

ARM TrustZone

1. Normal-world memory
2. Secure-world memory
3. $k_s$
Analysis of NW memory snapshot

- Infer what peripherals are installed, and where in memory their drivers are installed
- Detect guest device for malware infection, including kernel-level rootkits

[Baliga, Ganapathy, Iftode, ACSAC’08, TDSC’11]
Why look for NW rootkits?

Normal world (Untrusted)

Rootkit

Normal world OS

Secure world

Secure world applies updates

Secure-world memory

ARM TrustZone
Why look for NW rootkits?

Secure world

\[ k_s \]

Normal world
(Untrusted)

ARM TrustZone

Rootkit

Rootkit undoes host’s changes

Normal world OS

Secure-world memory
Analysis of NW memory snapshot

Host’s policy server

Recursive traversal of memory data structures

Root symbols & kernel entry points

Code whitelist

Data invariants

Code pages

Data structs

Code pages

Data structs

Root symbols & kernel entry points

Code whitelist

Data invariants

Recursive traversal of memory data structures
Vetting host’s requests

- Vetting server ensures that host’s requests do not compromise guest privacy
- **Vetting policy**: Host only allowed to request guest device’s kernel memory
Guest device update

Host’s policy server

Memory updates

Restricted space
SW updates NW memory

Normal world (Untrusted)

Secure world

Secure-world memory

ARM TrustZone

k_S
Device drivers in normal world control execution of device peripherals
Introduce dummy driver to control peripheral (e.g., disable it). Update kernel driver hooks.
Are driver updates effective?

<table>
<thead>
<tr>
<th>Peripheral considered</th>
<th>Update size (bytes)</th>
<th>Guest device</th>
<th>Peripheral disabled?</th>
</tr>
</thead>
<tbody>
<tr>
<td>USB webcam</td>
<td>302</td>
<td>i.MX53</td>
<td>✓</td>
</tr>
<tr>
<td>Camera</td>
<td>212</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
<tr>
<td>WiFi</td>
<td>338</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
<tr>
<td>3G (Data)</td>
<td>252</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
<tr>
<td>3G (Voice)</td>
<td>224</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
<tr>
<td>Microphone</td>
<td>184</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>132</td>
<td>Nexus phone</td>
<td>✓</td>
</tr>
</tbody>
</table>
Vetting host’s updates

• An untrusted host can introduce new code into guest devices

• **Vetting policy**: Ensure that dummy drivers are a *subset* of the original drivers
  
  – Via ARM-binary analysis on

![Diagram showing vetting process and policy](image-url)
4 Proof of compliance

Host’s policy server

Request proof of policy compliance

Verification token

Restricted space
Verification tokens

- Host requests proof of compliance
- Secure world computes a fresh snapshot of all NW memory locations updated by host
- Verification token:

\[ \text{HMAC}(\text{memory locations}, k_s) \]

- Verification token matches if and only if normal world memory still in compliance with the host’s usage policy
Memory updates are ephemeral

• Guest device can violate host’s usage policies by simply rebooting to undo host’s memory updates!

• Once device checked in, secure world must:
  – Mediate all low-battery and power-off interrupts
  – Checkpoint device memory to disk
  – Upon power up, must restore device memory from checkpoint
Device checkpoint

• **Problem**: Checkpoint stored on disk
  – Readable by untrusted end-user
  – But session key $k_s$ must not be stored in clear
  – Otherwise, malicious end-user can use it to impersonate guest’s trusted secure world!

• **Solution**: REM-suspend protocol
REMsuspend

• ARM TrustZone equips each device with a device-specific key $K_{\text{DEV}}$.

• The key $K_{\text{DEV}}$ is only accessible from the secure world.

• We use $K_{\text{DEV}}$ to encrypt $k_s$ in device checkpoint.

• When device is powered again, secure world uses $K_{\text{DEV}}$ to decrypt and restore $k_s$. 
REM-suspend

Guest device storage

ARM TrustZone

Secure world

$k_s$  $K_{DEV}$

$k_s$
REM-suspend

Guest device storage

Secure world

Secure storage

ARM TrustZone

ks

KDEV

ks

ks

ks

KDEV
Are memory updates the right API?

- Powerful, low-level API for device control
- Simplifies design of secure world (TCB) and keeps it device-independent

<table>
<thead>
<tr>
<th>TCB component</th>
<th>SLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory manager</td>
<td>1381</td>
</tr>
<tr>
<td>Authentication</td>
<td>1285</td>
</tr>
<tr>
<td>Memory ops., verification tokens</td>
<td>305</td>
</tr>
<tr>
<td>REM-suspend</td>
<td>609</td>
</tr>
<tr>
<td>SHA1 + HMAC</td>
<td>861</td>
</tr>
<tr>
<td>X509</td>
<td>877</td>
</tr>
<tr>
<td>RSA</td>
<td>2307</td>
</tr>
</tbody>
</table>
Do memory updates affect app stability?

**Passive updates:** Update memory and start the app

<table>
<thead>
<tr>
<th>USB</th>
<th>MobileWebCam</th>
<th>ZOOM FX</th>
<th>Retrica</th>
<th>Candy Cam</th>
<th>HD Cam Ultra</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Error</td>
<td>Android Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>Android Error</td>
</tr>
<tr>
<td>Camera</td>
<td>Android Cam</td>
<td>Camera MX</td>
<td>ZOOM FX</td>
<td>Droid HD Cam</td>
<td>HD Cam Ultra</td>
</tr>
<tr>
<td>Android Error</td>
<td>App Error</td>
<td>App Error</td>
<td>Android Error</td>
<td>Android Error</td>
<td></td>
</tr>
<tr>
<td>WiFi</td>
<td>Spotify</td>
<td>Play Store</td>
<td>YouTube</td>
<td>Chrome</td>
<td>Facebook</td>
</tr>
<tr>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td></td>
</tr>
<tr>
<td>3G (Data)</td>
<td>Spotify</td>
<td>Play Store</td>
<td>YouTube</td>
<td>Chrome</td>
<td>Facebook</td>
</tr>
<tr>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td></td>
</tr>
<tr>
<td>3G (Voice)</td>
<td>Audio rec</td>
<td>Easy voice rec</td>
<td>Smart voice rec</td>
<td>Snd/voice rec</td>
<td>Smart voice rec</td>
</tr>
<tr>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td></td>
</tr>
<tr>
<td>Microphone</td>
<td>Audio rec</td>
<td>Easy voice rec</td>
<td>Smart voice rec</td>
<td>Snd/voice rec</td>
<td>Smart voice rec</td>
</tr>
<tr>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td></td>
</tr>
</tbody>
</table>
Do memory updates affect app stability?

**Active updates:** Update memory with “live” app

<table>
<thead>
<tr>
<th>USB</th>
<th>MobileWebCam</th>
<th>ZOOM FX</th>
<th>Retrica</th>
<th>Candy Cam</th>
<th>HD Cam Ultra</th>
</tr>
</thead>
<tbody>
<tr>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
<td>App Error</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Camera</th>
<th>Android Cam</th>
<th>Camera MX</th>
<th>ZOOM FX</th>
<th>Droid HD Cam</th>
<th>HD Cam Ultra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank Screen</td>
<td>App Error</td>
<td>Android Error</td>
<td>Blank Screen</td>
<td>Blank Screen</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WiFi</th>
<th>Spotify</th>
<th>Play Store</th>
<th>YouTube</th>
<th>Chrome</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3G (Data)</th>
<th>Spotify</th>
<th>Play Store</th>
<th>YouTube</th>
<th>Chrome</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td>No Connection</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3G (Voice)</th>
<th>Default call application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unable to place call</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Microphone</th>
<th>Audio rec</th>
<th>Easy voice rec</th>
<th>Smart voice rec</th>
<th>Snd/voice rec</th>
<th>Smart voice rec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty File</td>
<td>Empty File</td>
<td>Empty File</td>
<td>Empty File</td>
<td>Empty File</td>
<td>Empty File</td>
</tr>
</tbody>
</table>
Related approaches

• Device virtualization:
  – Heavyweight; probably not for all devices
  – Still requires host to trust hypervisor on guest

• Mobile device management solutions:
  – No proofs to host
  – Device-dependent TCB on guest

• Context-based access control:
  – Same shortcomings as MDM solutions above
Conclusion

A systematic method to regulate devices and ensure responsible use

• Low-level API allows hosts to analyze and control guests
  – Simplifies design and size of TCB

• Hosts can obtain proofs of guest compliance
  – Relies on ARM TrustZone hardware

• Vetting service balances guest privacy with host’s usage policies
Other research projects…

Generic theme: Computer Systems Security

• Improving cloud platform security
  [ACSAC’08a, RAID’10, CCS’12a, SOCC’14]

• Operating system reliability and security
  [ASPLOS’08, ACSAC’08b, ACSAC’09a, MobiSys’11, TDSC’11, TIFS’13]

• Hardware support for software and system security
  [CCS’08, ECOOP’12a, TIFS’13, MobiSys’16, RU-DCS-TR724]

• Web application and Web browser security
  [ACSAC’09b, ECOOP’12a, ECOOP’12b, ECOOP’14, FSE’14]

• Tools for cross-platform mobile app development
  [ICSE’13, ASE’15]

• Retrofitting legacy software for security
  [CCS’05, Oakland’06, ASPLOS’06, ICSE’07, CCS’08, CCS’12b]

• Validating security retrofitting transformations in optimizing compilers
  [Submitted]
A big thank you to my students

**Graduated PhDs**
- Dr. Mohan Dhawan (IBM Research India)
- Dr. Saman Zarandoon (Amazon.com)
- Dr. Shakeel Butt (Nvidia → now at Google)
- Dr. Liu Yang (HP Labs → now at Baidu)
- Dr. Rezwana Karim (Samsung Research America)
- Dr. Amruta Gokhale (Teradata)

**Former Postdocs**
- Dr. Arati Baliga (AT&T Security Labs)

**Graduated MS students**
- Jeffrey Bickford (AT&T Research)
- Yogesh Padmanaban (Microsoft)

**Current PhD students**
- Jay P. Lim, Hai Nguyen, Daeyoung Kim.
Email: vinodg@cs.rutgers.edu
URL: http://www.cs.rutgers.edu/~vinodg