SeCloak: ARM TrustZone-based Mobile Peripheral Control

Matthew Lentz, Rijurekha Sen, Peter Druschel, Bobby Bhattacharjee
Control Over Your Devices

Powerful sensing and communication capabilities
But can be misused by malicious software!

Consider important scenarios:
Journalists use airplane mode while meeting with source
Turn off microphone to prevent snooping
Users Have Limited Control

There are two fundamental issues:

Incomplete settings
e.g., Motion sensors on Android

No assurance that settings are enforced

Platform shown to be hard to secure as a whole
Problem Statement

What is minimally required to give users **secure** control over their devices?

Without

- affecting usability or stability
- changes to existing software
What is minimally required to give users **secure** control over their devices?

SeCloak - “Secure Cloak”

**Sensing**
- Camera
- Microphone
- Location
- Motion
- Orientation

**Communication**
- NFC
- Bluetooth
- WiFi
- Cellular

SeCloak provides secure “virtual” switches to users
What is minimally required to give users **secure** control over their devices?

SeCloak provides secure “virtual” switches to users.
SeCloak Design

- Trusted
- Untrusted

SeCloak Settings App → SeKernel
SeCloak Design

- **SeCloak Settings App**
  - Provides UI similar to traditional settings menus
  - Communicates policy settings to SeKernel

- **SeKernel**

- **Policy**

- **Trusted**
- **Untrusted**
SeCloak Design

- **SeCloak Settings App**
  - Provides UI similar to traditional settings menus
  - Communicates policy settings to SeKernel

- **SeKernel**
  - Secure (re)display and user confirmation of policy
  - Configure HW protections to disable untrusted access
  - Handle access faults to enforce user policy
ARM TrustZone supports two “worlds”
Isolates SeKernel from untrusted kernel and apps
Allows SeKernel to configure hardware protections
Hardware Protections

SeCloak Settings App

Kernel (e.g., Linux)

SeKernel

SMC

Set IRQ owner to **Kernel** or **SeKernel**
Hardware Protections

1. Set IRQ owner to **Kernel** or **SeKernel**
2. Configure to deny accesses made by **Kernel**
   - Reports access faults to **SeKernel**
Hardware Protections

How do we securely identify these protection domains for devices?

1. Set IRQ owner to **Kernel** or **SeKernel**
2. Configure to deny accesses made by **Kernel**
   Reports access faults to **SeKernel**
Device Tree (DT)

Device Tree specifies embedded hardware
Each node represents a device

Nodes contain configuration properties
Added **Security** and **Class** properties

Security corresponds to HW firewall configuration
Class associates a known setting name with a device

SeKernel verifies and parses a signed DT
Application Functionality

SeCloak Settings App

Select a mode:
- None
- Airplane
- Movie
- Stealth

Networking:
- Wi-Fi
- Bluetooth
- Cellular

Multimedia:
- Camera
- Speaker
- Microphone

Sensor:
- GPS
- Inertial
Example: Disabling Bluetooth

SecureCloak

Select a mode:
- None
- Airplane
- Movie
- Stealth

Networking:
- ON
- OFF
- CUSTOM
- Wi-Fi
- Bluetooth
- Cellular

Multimedia:
- ON
- OFF
- CUSTOM
- Camera
- Speaker
- Microphone

Sensor:
- ON
- OFF
- CUSTOM
- GPS
- Inertial

SeCloak Settings App

SeKernel

[Bluetooth Disabled]

CLOAK_SET([010...0])

SMC
Example: Disabling Bluetooth

Policy could be modified by malicious software!
SeKernel: Confirming Policy

1. Acquire the display and input devices
2. Turn on the LED to notify user that SeKernel is active
3. (Re)Display settings to user
4. Wait for user confirmation for whether to apply settings…

SeKernel

CLOAK_SET([010…0])
SeKernel: Applying Policy

CLOAK_SET([010…0])
SeKernel: Applying Policy

Bus
Peripheral
Interrupt Controller

Security = <25>
Class = “Touchscreen”

Security = <70>
Class = “Bluetooth”

CLOAK_SET([010...0])
SeKernel: Applying Policy

CLOAK_SET([010…0])
SeKernel: Applying Policy

Class = "Bluetooth"

Class = "Touchscreen"

CLOAK_SET([010...0])
SeKernel: Applying Policy

For all devices in the **subtree**:
- Secure and disable IRQs
- Configure firewall protections
- Setup fault handler for MMIO accesses
What happens if the **Kernel** accesses a protected device?
SeKernel: Fault Handling

1. Determine instruction and data address

   (LDR|STR) Reg, [Address]

   From disassembling the instruction

   From CPU fault information
SeKernel: Fault Handling

1. Determine instruction and data address
   - \((LDR|STR)\) Reg, [Address]
   - From disassembling the instruction
   - From CPU fault information

2. Lookup and enforce policy for address
   - **Deny**: Discard STR / Return 0 for LDR
   - **Allow**: Issue LDR/STR & Optionally modify value
     - Set of devices with common security group (or)
     - Device shared between NS/S worlds
Prototype for Nitrogen6X board
  i.MX6 SoC with ARM Cortex A9 (1GHz)

SeKernel implemented in <15k LoC
Based on pared-down OP-TEE OS
Includes drivers for CSU, Framebuffer, GPIO, and Keypad

Benchmarks demonstrate reasonable overhead:

<table>
<thead>
<tr>
<th>Execution</th>
<th>Instruction Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Load (LDR)</td>
</tr>
<tr>
<td>Baseline</td>
<td>0.11</td>
</tr>
<tr>
<td>Emulated</td>
<td>1.14</td>
</tr>
</tbody>
</table>

Repeated accesses to WiFi controller register
SeCloak enforces user-specified on/off control policies. Small enforcement kernel runs alongside any OS.

Source code is available at: www.cs.umd.edu/projects/secureio
Backup Slides
## SeKernel: LoC Breakdown

<table>
<thead>
<tr>
<th>Type</th>
<th>LOC Breakdown</th>
<th></th>
<th></th>
<th>Total</th>
<th>Stmt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C Src</td>
<td>C Hdr</td>
<td>ASM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td>3233</td>
<td>2357</td>
<td>1391</td>
<td>6981</td>
<td>3781</td>
</tr>
<tr>
<td>Drivers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CSU</td>
<td>45</td>
<td>9</td>
<td>0</td>
<td>54</td>
<td>29</td>
</tr>
<tr>
<td>Device Tree</td>
<td>401</td>
<td>57</td>
<td>0</td>
<td>458</td>
<td>261</td>
</tr>
<tr>
<td>Frame Buffer</td>
<td>146</td>
<td>29</td>
<td>0</td>
<td>175</td>
<td>113</td>
</tr>
<tr>
<td>GPIO</td>
<td>562</td>
<td>15</td>
<td>0</td>
<td>577</td>
<td>284</td>
</tr>
<tr>
<td>GPIO Keypad</td>
<td>169</td>
<td>14</td>
<td>0</td>
<td>183</td>
<td>89</td>
</tr>
<tr>
<td>&lt;Other&gt;</td>
<td>579</td>
<td>167</td>
<td>0</td>
<td>746</td>
<td>265</td>
</tr>
<tr>
<td>Drivers Total</td>
<td>1902</td>
<td>291</td>
<td>0</td>
<td>2193</td>
<td>1041</td>
</tr>
<tr>
<td>Libraries</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>libfdt</td>
<td>1220</td>
<td>350</td>
<td>0</td>
<td>1570</td>
<td>840</td>
</tr>
<tr>
<td>bget/malloc</td>
<td>1421</td>
<td>68</td>
<td>0</td>
<td>1489</td>
<td>797</td>
</tr>
<tr>
<td>&lt;Other&gt;</td>
<td>1479</td>
<td>1182</td>
<td>81</td>
<td>2742</td>
<td>1212</td>
</tr>
<tr>
<td>Libraries Total</td>
<td>4120</td>
<td>1600</td>
<td>81</td>
<td>5801</td>
<td>2849</td>
</tr>
<tr>
<td>Total</td>
<td>9255</td>
<td>4248</td>
<td>1472</td>
<td>14975</td>
<td>7671</td>
</tr>
</tbody>
</table>
## Micro: Emulated LDR/STRs

<table>
<thead>
<tr>
<th>Execution</th>
<th>Instruction Time (μs)</th>
<th>Load (ldr)</th>
<th>Store (str)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td></td>
<td>0.11</td>
<td>0.29</td>
</tr>
<tr>
<td>Linux+SOM</td>
<td></td>
<td>0.27</td>
<td>0.33</td>
</tr>
<tr>
<td>Emulated</td>
<td></td>
<td>1.14</td>
<td>1.19</td>
</tr>
</tbody>
</table>
Macro: Emulated Wi-Fi

- Download (Linux)
- Download (Linux+SOM)
- Download (Emulated)
- Upload (Linux)
- Upload (Linux+SOM)
- Upload (Emulated)

CDF vs. Transfer Time (Seconds)
SeKernel: Emulation Details

Non-Secure
- LDR R0, [R1]
- CMP R0, #0

Monitor
Data Abort:
- Save Regs to Ctxt
- I_PA = NS_V2P(LR)
- D_PA = NS_V2P(DFAR)
- Emu(Ctx, I_PA, I_DA)
- Load Regs from Ctxt
- RFE

Emu(Ctx, I_PA, D_PA):
- I_VA = S_P2V(I_PA)
- D_VA = S_P2V(D_PA)
- Instr = Disasm(*I_VA)
- Reg = &Ctx[Instr.Rt]
- switch(Instr.Type)

Bus Access
- Case STR:
  - if(AllowWrite(D_PA, Reg, ...))
  - IOWrite(*Reg, D VA)
- Case LDR:
  - if(AllowRead(D_PA, ...))
  - *Reg = IORead(D VA)
  - UpdateRead(D_PA, Reg, ...)

CPU Core

Bus Error

Firewall

Dev 1

Dev 2

Dev 3