Next generation mobile rootkits

Hack In Paris 2013
Hi!

- Thomas Roth
  - Embedded security (Payment terminals etc.)
  - Distributed computing (and breaking)
  - Web-stuff
  - Social: @stacksmashing
Prologue
Mobile rootkits & Trustzone
What are we protecting?

- Communication
- Data
- Credentials
- Payment
- Tracking
In the wild

- CarrierIQ (Usage statistics)
- FinFisher (Governmental surveillance)
- Cloaker (Research)
Where do they hide?

- Memory
- Baseband
- CPU
Where do they hide?

- Memory
- Baseband
- CPU
- TrustZone
Short ARM intro

• 32-bit RISC architecture

• All instructions 32 bit long
  
  • Except in “Thumb-mode”, 16-bit instructions

• Peripherals are mapped in memory (i.e. IOs at 0x40000000)

• CP15 = System Control Coprocessor
  
  • Controls features like MMU, power saving, caches ...
ARM TrustZone

• Allows the processor to switch into a “secure mode”
• Secure access to screen, keyboard and other peripherals
• Tries to protect against: “...malware, trojans and rootkits”
• So called TEEs (Trusted Execution Environments) run in it
• Introduced with ARMv6KZ

ARM TrustZone

- Allows the processor to switch into a “secure mode”
- Secure access to screen, keyboard and other peripherals
- Tries to protect against: “...malware, trojans and rootkits”
- So called TEEs (Trusted Execution Environments) run in it
- Introduced with ARMv6KZ

ARM TrustZone

• Splits the CPU into two worlds: Secure and normal
• Communication between both worlds via shared memory mappings
• State of the CPU is indicated to peripherals using a bit on AXB/AHB
  • Allowing secure-only on- & off-chip peripherals
Trusted Execution Environments

• Small operating system running in TrustZone and providing services to the ‘real’ operating system/apps

• You write apps for them and pay ??? to integrate them

• Use them via a driver in your operating system

• Drivers are often open-source
Example: Netflix

- Netflix requires a device-certification
- For SD, the device just needs to be fast enough to play video
- For HD, the labels require ‘end-to-end’ DRM, so that the video-stream can’t be grabbed at any time
- Video decoding running in TrustZone with direct access to screen, no way to record it from Android

Attacker Model

- Protect against:
  - Device owners (DRM and other copy protection methods)
  - Malware (Steal PayPass informations from the device)
  - Freedom
How does it work?

- A second register set to the first CPU core
- Mode switch from normal world:
  - SMC instruction (Secure Monitor Call)
  - Hardware exception (IRQ, FIQ, etc.)
- NS bits on the internal bus (AXI/AHB) for indicating state to peripherals
How does it work?

How does it actually work?

Standard CPU modes: User, FIQ, IRQ, SVC ...

Hardware interrupt/SMC call

SMC Mode

State handling & NS-bit
How does it *actually* work?

- SMC: (Always has the NS bit enabled)
  - Detect whether coming from normal or secure world
  - Store registers of current world
  - Load registers of new world
  - Toggle NS bit
  - Give execution to new world
Memory in TrustZone

Normal World

Secure World

TrustZone enabled MMU

Normal-world memory

Shared

Secure-only memory
Memory in TrustZone

- Normal-world only sees its own memory as well as the shared segment.
- Secure-world can access everything.
Memory in TrustZone

• Normal-world only sees its own memory as well as the shared segment

• Secure-world can access **everything**

I guess you see why this is could useful...
Boot process

1. stage bootloader (ROM)
   - Hardware abstraction
   - Integrity verification of 2. stage bootloader

2. stage bootloader (Flash)
   - Initialization of TrustZone + TrustZone Kernel
   - Lockdown of TrustZone
   - Load & configure OS

Operating System
   - Android
   - Windows
   - ...

Start
Boot process

1. stage bootloader (ROM)
   - Hardware abstraction
   - Integrity verification of 2. stage bootloader

2. stage bootloader (Flash)
   - Initialization of TrustZone + TrustZone Kernel
   - Lockdown of TrustZone
   - Load & configure OS

Operating System
   - Android
   - Windows
   - ...

leveldown security

Thomas Roth
By the way

• “Unlocking” a bootloader does not mean that you can do what you want.

• Most hardware vendors still lockdown a significant part of the boot chain, locking you out of TrustZone.
Hardware support

- All modern smartphone CPUs:
  - Qualcomm Snapdragon
  - TI OMAP
  - ...

So basically...

- The vendor installs a small operating system in a part of the CPU
- This OS can do -anything- and third party apps are installed in it
So basically...

- The vendor installs a small operating system in a part of the CPU
- This OS can do *anything*- and third party apps are installed in it

What could possibly go wrong?
Chapter 1
Building a rootkit in TrustZone
What?

• Super small rootkit that runs in TrustZone (and now even in SMC!)

• First TrustZone rootkit

• Implemented entirely in assembler

  • (Compilers are for losers)
Why?

• It’s fun!

• People haven’t talked about the problems that come with TrustZone

• The ‘trusted’ in Trusted Computing is not only about the user trusting the hardware, but also about the vendor distrusting the user
Why?
Where to test?

• Developing on actual hardware:
  • Need an OMAP HS (not GP) Dev-board
  • Beagleboards & co (OMAP) switch to normal world in ROM
  • (Hardware is locked down with strong keys)
Where to test?

• Software emulation:
  • QEMU supports ARM

• Paper “A flexible software development and emulation framework for ARM TrustZone”
  • “qemu-trustzone”
  • Johannes Winter, Paul Wiegele, Martin Pirker, and Ronald Toegl

• Open-source (GPL) TEE by sierraware (Open Virtualization)
Where to test?

- Using hardware hack for $vendor to execute code in TrustZone
Where to test?

• Let’s just hack into a TEE!
Getting a binary-image of a TEE

- Firmware updates are signed, but not encrypted.
- Firmware updates are often downloaded via HTTP.
- Vendors have ‘hidden’ FTP-servers.
Workflow

• Disassemble bootloader

• IDA Pro & co can’t just find code parts, need to analyze by hand & with scripts

• Analyze coprocessor instructions to find memory layout

• (almost automated)

leveldown security
Looking for mitigations

NOT SURE IF OBFSUCATED

OR REALLY NOT EXISTENT
Looking for mitigations

• No ASLR
• No DEP (NX)
• Executable heap, stack, data, everything
Exploiting like it’s 1999

Yeah Baby!
Let's talk about strncpy

- strncpy(char *destination, char *source, size_t destination_size);
- strncpy sucks, it only NUL-terminates if:
  - strlen(destination) < destination_size
- (Please, use strlcpy. Also, did I mention I do code reviews?)
- Still often pretty hard to exploit.
Where to test?

• Official boards seem to be hard to obtain
• QEMU
• On smartphone via hardware hack
• On smartphone via software exploit
Scheduling the rootkit

• Switch to monitor mode in (ir)regular intervals without help of the operating system

• The latency induced by the switch to the rootkit must be kept as low as possible
Latency

- Normal world interrupt/SMC
- Monitor code
- Rootkit scheduler
- Monitor code
- Back to normal world
Latency

- Normal world interrupt/SMC
- Monitor code
- Rootkit scheduler
- Monitor code
- Back to normal world
Scheduling the rootkit

- ARM actually gives tips for running TrustZone invisible to the normal world:

  Does the Secure world care about direct or indirect Normal world visibility of its execution?

  If yes, then consider obfuscating interrupt timing, disabling Non-secure access to the Performance Counters, and performing selective cache maintenance on critical address ranges on world switch.
Scheduling the rootkit

• How is control transferred to the monitor mode?
  • IRQ? Used a lot by the OS & can be masked by normal world
  • External abort? Too unreliable & environment dependent
  • FIQ? Can be locked down (NMI)
  • Overwrite the interrupt vector (like Cloaker does)
  • TZIC - TrustZone Interrupt Controller
IRQ ‘interception’

1. IRQ Triggered
2. Monitor saves state
3. "Real" IRQ handler is called
4. Monitor restores state
5. Rootkit executes
Boot process

1. Setup secure-world
2. Setup monitor
3. Lockdown TrustZone (Switch to normal world)
4. Start operating system/bootloader
Boot process

1. Setup secure-world
2. Setup monitor
3. Lockdown TrustZone (Switch to normal world)
4. Start operating system/bootloader
Secure World Memory Setup

• Create page table for secure world
• Turn on MMU
• Configure stack

• Recommendation if no TEE is already available:
  • Use physical address space next to a hardware mapping
Secure World Initialization

- Run initialization-routine of rootkit
- Configure ‘system-call’ facility of rootkit (IRQ)
- Load and create contexts for modules
Secure World

• Uses a timer-based slice ‘scheduler’ by default to keep time ~constant

• (Timing by TZIC - TrustZone Interrupt Controller)

• Store files in secure storage if available
Boot process

1. Setup secure-world
2. Setup monitor
3. Lockdown TrustZone (Switch to normal world)
4. Start operating system/bootloader
Monitor

• Store normal world register banks and switch on NS bit
• Execute rootkit scheduler
• Store secure world register banks and switch off NS bit
Monitor

• Check if coming from normal or secure world:

  • mrc p15, 0, r0, c1, c1, 0
  • tst r0
  • beq to_normal
  • bne to_secure
Monitor

• Storing the state of all register banks:
  
  STMIA r0!, {r0 - r13}
  CPS 0x19 ; SVC mode
  STMIA r0!, {r1, r13, lr}
  ... for all processor modes

• ...And vice versa for restoring the register banks
Monitor

• Be aware of cache + pipeline stuff!
  • Use pipeline-flushing instructions
  • Be sure to have your cache configuration right
Monitor setup

• Configure TCM for latency reduction

• Setup the interrupt interception mechanism

  • (Set IRQ to secure in SCR (Secure Configuration Register))
Boot process

1. Setup secure-world

2. Setup monitor

3. Lockdown TrustZone (Switch to normal world)

4. Start operating system/bootloader
Lockdown: SCR

- Security Configuration Register (C1 C1 on CP15)

- Disable modification of A & F bit in CPSR from Normal

- World for External Abort, FIQ, IRQ

- Actual NS bit
Lockdown: SCR

• Sample configuration:
  • `mov r0, #0x3E`
  • `mcr p15, 0, r0, c1, c1, 0`

• Sets world for IRQ + FIQ to secure

• I decided to let the normal world mask the interrupts
  • (Avoid suspicion)
Lockdown

• Depending on the hardware, some other configurations might be needed, too

• i.e. External storage may requires configuration to know which part is only accessible as secure

• TZPC (TrustZone Protection Controller) configuration
Boot process

1. Setup secure-world
2. Setup monitor
3. Lockdown TrustZone (Switch to normal world)
4. Start operating system/bootloader
Start operating system

- TrustZone is configured and ready to go
- Operating system + loader starts unmodified
- Nothing to see here, move along!
Real world problems

- $vendor does power management setup in TrustZone
- Had to integrate vendor blob into custom TZ image
Chapter 2

It boots! And now?
We...

• Have an execution environment that is entirely separated from the normal OS

• Configured the CPU and its peripherals to hide any traces from our existence

• Get control of the CPU regularly
We can...

• access all user data

• manipulate memory of **everything**

• communicate (kind of)
Yay!
Communication

• Difficult

• Talk to baseband
  • Access to UMTS, which at least very difficult to sniff

• Create IP packets by directly talking to the network hardware
  • See Cloaker paper

• Suspicious and easily detected
What else to do?

- There’s quite some secret stuff in TrustZone implementations
Installation in the wild

• Via exploit in app

• Vendor delivered (either during manufacturing or by FW update)

• Baseband attack
Interoperability

• Ports to new hardware can be done in under a week:
  • TrustZone is almost the same on all platforms
  • Mostly peripherals and memory stuff differs

• TEE may has to be integrated, too

• Easy to do, i.e. modify OpenVirtualization TEE or binary patch
Detection methods

• Latency

• Be paranoid and only use phones that don’t have TrustZone

• Good luck with that one.
Thank you!

• Any questions?

• Thomas Roth - thomas.roth@leveldown.de - @stacksmashing

• Slides: http://leveldown.de/hip_2013.pdf