Becoming a tyrant:

Implementing secure boot in embedded devices

Irving Tjiptowarsono

Hi, I’m Irving

- I want to talk about secure boot
  - Some resources on how, not much on *why* use secure boot
- This is a talk on embedded devices, not servers with UEFI boot
  - Some circumstances are different on embedded!
- Slides available on https://ssfivy.github.io

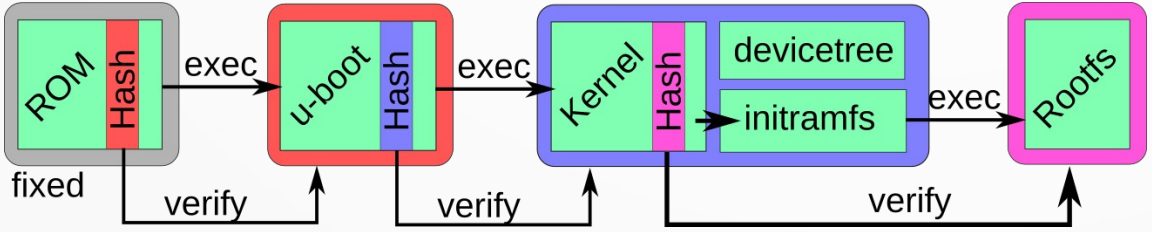
Legal bits

- All opinions in this presentation are my own and not from my employer or any other associates
- All references are to publicly available information
- Neither I nor my employer takes any legal or commercial responsibility that may arise from anything in this presentation

Secure Boot: How it works?

Chain of trust mechanism

- Verify integrity of next component before executing
- Can use hashes or public keys
- Can provide some protection against tampering (incl. physical)



Factors to Consider

Threat Model

The Tyrant

- Whoever controls the keys/hashes, controls everything
- Someone could be locking you out
- You could be the one locking others out
- Who should hold the keys?

Who is your adversary?

- Can be used in a variety of scenarios
- Important to determine who has control and who has none
- Useful to prevent access / modification on lower level
- Not the best tool against attacks from network

Hyphothetical scenario 1

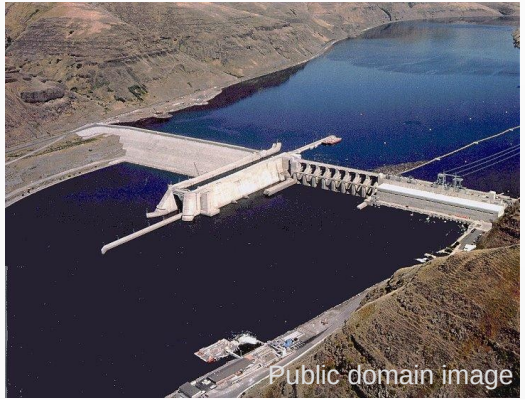
- Alice Computers Pty Ltd makes laptops and its software.
- They want to control the software that users can run.
- They lock their laptops with secure boot.
- Alice Computers is locking you out from your stuff.

Hyphothetical scenario 2

- Bob has a steel mill full of custom control devices
- Bob does not want malware in his control devices
- Bob is tech-savvy, so he uses secure boot to lock his devices
- Bob is locking everyone else out from his stuff

Hyphothetical scenario 3

- Dave owns a hydroelectric dam full of custom control devices
- Dave does not want malware in his control devices
- Dave is not tech-savvy, so he asks Carol for help.
- Carol adds secure boot into Dave’s devices and holds the key
- Carol is locking everyone else out of Dave’s stuff



Factors to Consider

Tamperproofing / Trustworthy Systems

Anything involving financial transactions

- ATMs
  - Skimming: Steal card info
  - Jackpotting: dump out contents

(c) Wired 2010

Anything involving financial transactions

- EFTPOS terminals
  - Skimming: Steal card info
  - Play video games on them

(c) MWR Labs

Anything involving financial transactions

- Pokies / Slot machines
  - Get more jackpot
- Phones
  - Access personal / financial data

Automotive ECU / Industrial controls

- Some devices control heavy and powerful things
- Cars, cranes, industrial equipments, steam turbines
- Tampering can cause injury, death, and legal liabilities

But I should be able to modify my devices!

- You can hurt yourself with your modifications
- You can be hurt from someone else’s modifications
- How would you feel if several oncoming car in the opposite lane runs modified ECU code?

What about fixing bugs in ECUs?

- Valid concern, considering e.g. Toyota acceleration case
- Manufacturers may have buggy code, but they still have better test setups than end user
- No way for end user to check for regression

Vendor lock-in

- Tamperproofing can be used to lock out competitors
  - e.g. generic spare parts, consumables, self-repair
- On one side, third party products may be faulty / not up to spec
- On the other side, lack of competition is bad for customers
- At what point does tamperproofing becomes lock-in?

Factors to Consider

Protecting Secrets

What kind of secrets?

- User data
  - phones
- Authentication data
  - EFTPOS machines
  - Set-top boxes
- Content decryption keys
  - Digital cinema projectors
- Software itself
  - preventing clone products

What kind of protection?

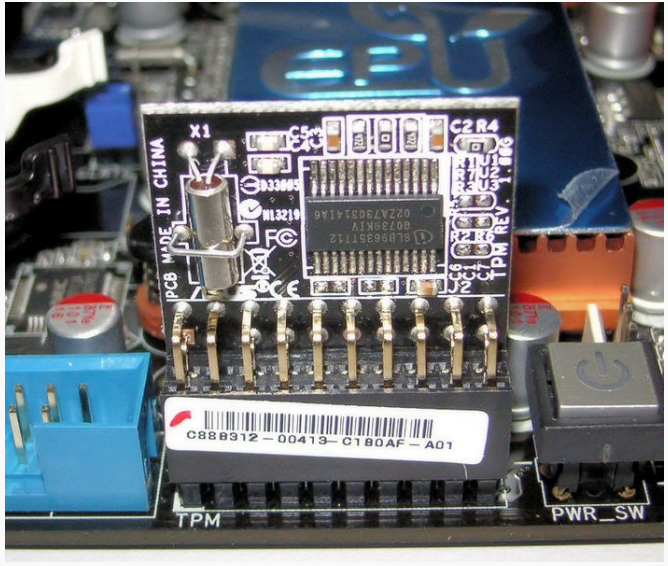
- Physical attacks
  - Prevent reading out secrets
  - Microcontroller has built-in readout protection
  - Embedded linux has exposed data bus between CPU and flash
- Network attacks are no different from servers

Why do we need secure boot for this?

- Blob / Filesystem / Full disk encryption is not enough
- Need a secure location to store decryption keys
- Need to restrict key access only to legitimate programs

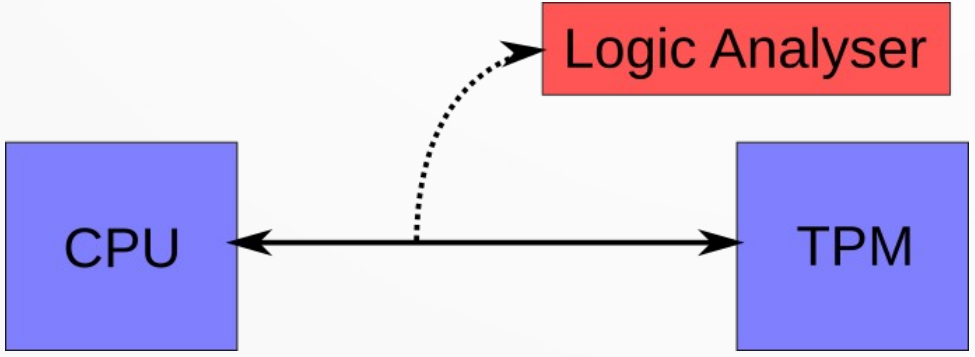
Trusted Platform Modules?

- Dedicated security chip, keeps secrets, does crypto operations
- Cheap, Standardised (Multiple vendors), Rich software APIs
- Secure - Not fast!



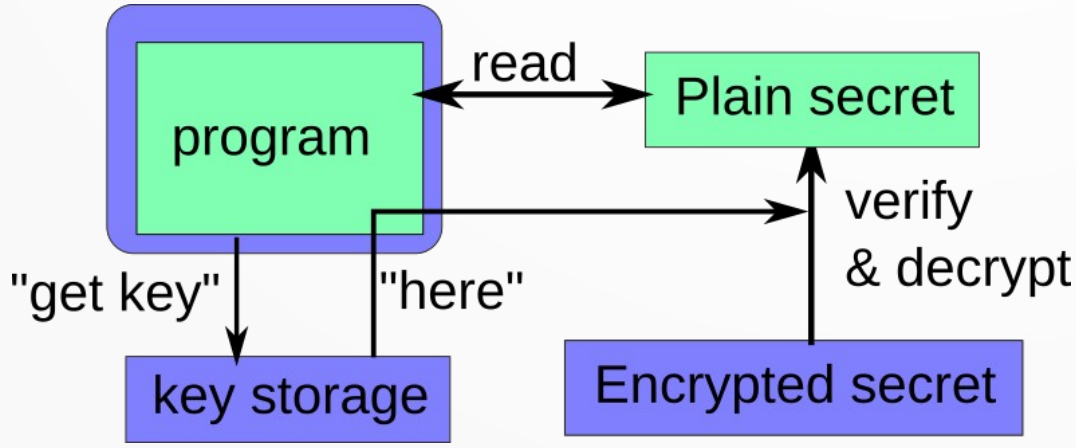
TPM pitfalls

- Enable parameter encryption
  - Encrypts communication between CPU and TPM
  - else sniffable by logic analyser!
- Need to be initialised by something trustworthy



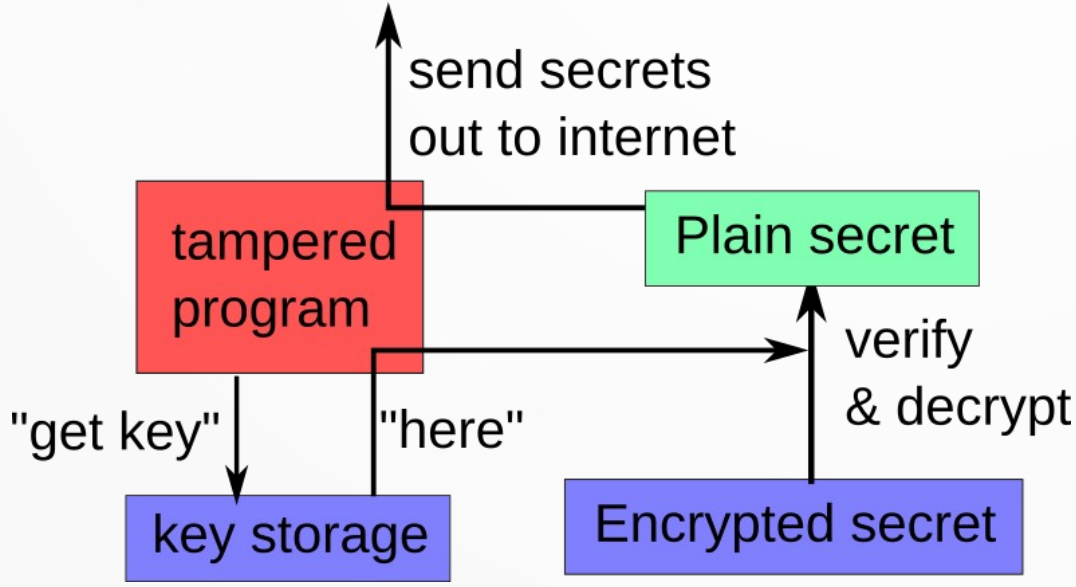
Encryption with secure boot

- At some point, some program decrypts disk/filesystem
- Attacker cannot modify anything in this sequence



Encryption without secure boot

- Same mechanism as before, but without verification
- Attacker modifies this program to decrypt *and* send data out



Factors to Consider

Disadvantages

Is it worth it?

- Need lots of development resources
  - Engineering time
  - multiple hardware kits
  - NDAs, export controls

Is it worth it?

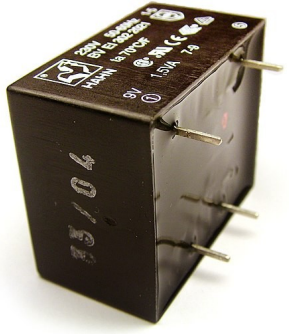
- Risk of bricking devices
  - What to do when verification fails?
  - Stop working but keep secrets safe?
  - Continue “working”, even if possibly compromised?

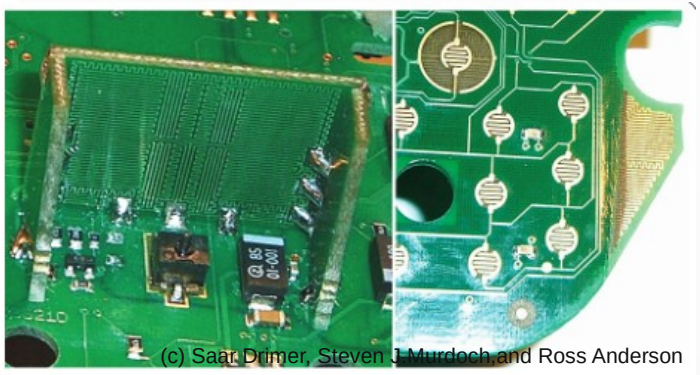
Is it worth it?

- Harder to use GPLv3 software
  - End-user must be able to modify the GPLv3 code in the device
  - Seems completely contradictory to secure boot?
  - Harder, not impossible!

Is it worth it?

- Not foolproof vs physical attack
  - Does nothing against logic analysers
  - Power analysis may still work
  - Chips can be desoldered, may open new vectors
  - Consider potting or tamper detection circuits





Is it worth it?

- Things have security bugs
  - Can be in software or hardware/ROM
  - Can be impossible to patch, all your effort is moot

How to become a tyrant

First stage (hardware-specific)

- Always vendor-specific, so start with vendor instructions
- Get multiple hardware kits
  - You will need to burn e-fuse and test different signed builds
  - prepare to brick some devices too
- Beware JTAG access, lock it up
- Look for hardware-unique support for encrypting secrets
  - May consider adding TPM

Second stage (bootloaders)

- Usually u-boot, may involve TF-A and/or OP-TEE
- Recommend using fitImage verification
  - Convenient: Verifies kernel, dtb, initramfs together
  - Not hardware/vendor-specific
- Disable environment import and console!
  - Can be used to override boot scripts

Third stage+ (initramfs, rootfs)

- Dm-crypt, dm-verity, dm-integrity
- Mix-and-match depending on your usecase, partition layout, etc
- Dm-verity implies read-only partition
- Once init starts – back in familiar territory

Builds

- Lots of signing done at buildtime - keep keys secure!
- Use separate dev / prod keys, secure boot always on
  - Less risk vs disabling secure boot in debug mode
- Don’t forget binaries for realtime coprocessor cores!
  - These cores can have main memory access too

Firmware updates

- You should use signed images
  - Don’t let anyone insert random code!
  - Ensure public key is secured by your secure boot process
- If you use encrypted images:
  - Need a secure location in the device to store decryption key

Mass manufacturing

- Locking software/interfaces can limit manufacturing flexibility
  - e.g. if JTAG needed, must use it before locking
  - e.g. slightly safer to generate private keys after locking
- Easier if factory environment is trusted – reduces threat vector
- SOMs may provide extra flexibility
  - Allows separate programming HW and real HW

GPLv3

- Ban GPLv3, use old GPLv2 versions
  - Missing out: bash, coreutils, tar, gdbserver, tracing/lttng
  - Alternatives: busybox, zsh
- Allow updating GPLv3 for those who want it
  - Option for disabling secure boot
  - device may go into "devkit mode" and self-erases secrets

Further reading

Secure boot on NXP i.MX6/7

- *Secure Boot from A to Z* by Quentin Schulz & Mylène Josserand, Bootlin
  - https://www.youtube.com/watch?v=jtLQ8SzfrDU
- *NXP AN4581 : Secure Boot on i.MX50, i.MX53, i.MX 6 and i.MX 7 Series using HABv4*
- *NXP AN12714 : i.MX Encrypted Storage Using CAAM Secure Keys*

U-boot verified boot

- *Secure and flexible boot with U-Boot bootloader* by Marek Vasut
  - https://elinux.org/images/8/8a/Vasut--secure_and_flexible_boot_with_u-boot_bootloader.pdf
- *U-Boot – Multi image booting scenarios* by JagannadhaSutradharudu Teki and Suneel Garapati
  - https://www.denx.de/wiki/pub/U-Boot/OpenSourceIndia2013/Multi_image_booting_scenarios.pdf

Including GPLv3 in embedded

- *Safely Copylefted Cars: Reexamining GPLv3 Installation Information Requirements* by Bradley Kuhn & Behan Webster
  - https://events19.linuxfoundation.org/wp-content/uploads/2017/11/Safely-Copylefted-Cars-Reexamining-GPLv3-Installation-Information-Requirements-ALS-Bradley-Kuhn-Behan-Webster-1.pdf

ATM / EFTPOS attacks

- *Jackpotting Automated Teller Machines* by Barnaby Jack
  - https://www.youtube.com/watch?v=4StcW9OPpPc
- *POSWorld. Should You be Afraid of Hands-On Payment Devices?* by Aleksei Stennikov and Timur Yunusov
  - https://www.youtube.com/watch?v=1cuTn6qGAj4
- *PinPadPwn* by: Nils & Rafael Dominguez Vega
  - https://www.youtube.com/watch?v=18IAjDGOdKo

Trusted Platform Module

- TPM-JS
  - https://google.github.io/tpm-js/
- *A Practical Guide to TPM 2.0* by Will Arthur, David Challener, and Kenneth Goldman
  - https://link.springer.com/book/10.1007/978-1-4302-6584-9
- tpm2-software community
  - https://tpm2-software.github.io/

Real-world examples

- *Trusted platform module security defeated in 30 minutes, no soldering required* , Ars Technica
  - https://arstechnica.com/gadgets/2021/08/how-to-go-from-stolen-pc-to-network-intrusion-in-30-minutes/
- *Countdown to Zero Day* by Kim Zetter (Stuxnet)
- Toyota Acceleration case (CJ-2008-7969 , Oklahoma)
  - Case study by Phil Koopman: https://users.ece.cmu.edu/~koopman/lectures/ece642/10_koopman_public_toyota_talk.pdf
  - BOOKOUT V. TOYOTA 2005 Camry L4 Software Analysis by Michael Barr : https://www.safetyresearch.net/Library/BarrSlides_FINAL_SCRUBBED.pdf

Real-world examples

- Some relevant CVEs
  - CVE-2017-7932 affecting boot ROM in NXP i.MX chips
  - CVE-2017-15361 ROCA vuln affecting Infineon chips
  - CVE-2020-10648 affecting u-boot

Thank you for watching!