ECE 471 – Embedded Systems
Lecture 22

Vince Weaver

https://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

25 October 2023

Announcements

- Don't forget HW#7

More Firmware

Quick Review of System Startup

- On bare-metal machine, embedded device jumps to entry point and immediately runs your code
- When running an OS this is a bit more complicated

Booting Linux

- Bootloader jumps into OS entry point
- Set Up Virtual Memory
- Setup Interrupts
- Detect Hardware / Install Device Drivers
- Mount filesystems
- Pass control to userspace / call init (systemd?)

- Run init scripts
- rc boot scripts, /etc/rc.local
  Start servers, or “daemons” as they’re called under Linux.
- fork()/exec(), run login, run shell

How a Program is Loaded on Linux

- Kernel Boots
- `init` started
- `init` calls `fork()`
- child calls `exec()`
- Kernel checks if valid ELF. Passes to loader (ld.so)
- Loader loads it. Clears out BSS. Sets up stack. Jumps to entry address (specified by executable)
- Program runs until complete.
- Parent process returned to if waiting. (`wait()`)

Otherwise, init.

Aside about Shared vs Staic Libraries

- Shared libraries, only need one copy of code on disk and in memory
  - Good for embedded system (less room needed)
  - Good for security updates (only need to update lib, not every program using it
- Static libraries, all libraries included
  - No dependencies
- These days maybe containers, docker, kubertenes
- Also Flatpack, Snap. Why? Stability, Know package

will work on all distributions, Not have to install dependencies

- Can use `ldd` to view library usage

More on Firmware

- What is firmware?
- Code that runs on an embedded system
  - Is it only bare metal?
  - Could a full Operating System be included?
- Traditionally is a binary “blob”
- Often in ROM or Flash and can be hard to update

Device Firmware

- Devices are their own embedded systems these days. Lots of small things have microcontroller inside
- Need to run code. Firmware.
- In ROM? Or upgradable? Why might you want to upgrade? (bug fixes, economy, etc.)

We previously talked about boot Firmware

Provides booting, configuration/setup, sometimes provides rudimentary hardware access routines.
Kernel developers like to complain about firmware authors.
Often mysterious bugs, only tested under Windows, etc.

- BIOS – legacy 16-bit interface on x86 machines
- UEFI – Unified Extensible Firmware Interface
  ia64, x86, ARM. From Intel. Replaces BIOS
- OpenFirmware – old macs, SPARC
- LinuxBIOS

Binary Blobs

- What is in the firmware?
  - Can you modify it yourself?
- Can it contain a full operating system?
  - ThreadX on Pi, Minix on Intel servers?
- Where does it live?
  - Hardcoded in ROM?
  - Upgradable by flash?
  - RAM that's uploaded at boot?

Non-persistent Firmware

- RAM that gets uploaded after boot (loses state on power off)
  What happens if your hard-drive or some other boot critical hardware uses this?
- Can be annoying if writing low-level code, you bought hardware but it might not work at all unless you upload a bunch of code to it each time you power up

Open Source Firmware Issues

- Can you run a “completely” free computer with completely open software?
- Depends on how you classify firmware
- Linux can be a pain to deal with firmware
  - licensing issues
  - can boot media (CD, USB) ship with firmware
  - What if distro (like Debian) has rules against non-open binaries
  - How can you netboot if the network card requires

proprietary firmware to work (common problem with wifi cards)

◦ Weird workarounds, webcam that had MacOS driver so had to extract firmware from that and copy to Linux partition

Firmware Licensing Issues

- Is a computer system truly free software if binary blobs running (like on a Pi)
  - Aside, FSF has weird definition where they only get upset about updatable firmware
- Debian tried to have a firmware-free install by default, but had to give up as not practical (especially at install)
- If someone ships firmware that has GPL code in it, what are their responsibilities?

Offloading Work to Firmware

- Companies don't like sharing their code, which makes things like Linux drivers hard
- Can they put as much as possible into firmware with only a small stub in OS?
  - Benefit: OS does less work, is simpler
  - Downside: you have no idea what this secret chunk of code is doing
- On Linux, NVIDIA drivers famously contentious. Possibly they are now planning to push more and more

code onto firmware on the card itself

Trusted Firmware

- Firmware can be dangerous: runs below/outside of the Operating System, doesn't matter how secure your OS is if firmware compromised
- Can you trust your firmware to be not-evil?
- Evil Maid problem – what if someone breaks into your hotel room and replaces your firmware – could you tell?

Firmware Dangers

- What if firmware reprogrammed on USB key to be evil?
- What if firmware on USB keyboard is evil? How would you know?
- What if hard-drive firmware reprogrammed
- Do you audit the firmware on your system?
- What could evil firmware do?
  - USB keyboard, log keypresses, send key commands at 2am to open web browser and cut-and-paste data to evil website

- Disk could refuse to write certain values, or maybe just notice writing a spreadsheet and randomize numbers, etc
- USB key could look like 4GB storage but then later change its ID to a network card and send data

What can we do?

- Epoxy shut USB ports on computers?
- Some secure sites actually do that
- It's a real threat, viruses have spread over USB
  Sneaky people can drop USB keys in parking lots in hopes they'll be plugged in to see who owns it

Signed Firmware

- Best you can do is trust it to be the same firmware released by your vendor (you still have to trust them)
- Use cryptographic signing. Hardware will only run code “signed” by a trusted entity.
- A signed firmware can run a signed bootloader which can run a signed operating system which can run signed apps

Signed Firmware Tradeoffs

- Downside: no longer general purpose, average person cannot run code they wrote unless they can get it signed
- Code still has to be well written. "jailbreaks" on phones and video game consoles are due to trusted code having bugs and then jumping into unsigned code.
- Will you still be able to run Linux? Trust Microsoft to keep signing bootloader for us?
- Walled gardens, restricted App stores (see Apple / EPIC lawsuit)

- Maybe makes sense if the firmware in your microwave locked down, but also might mean no one can ever fix it

Signed Firmware on “General Purpose” Computers

- Hardware manufacturer has keys that only allow booting trusted signed boot firmware (UEFI)
  - Signed UEFI only allows booting trusted signed bootloader
  - Signed bootloader only allows booting trusted signed operating system
  - Signed operating system only allows signed device drivers

  - Signed operating system only allows running signed apps
- Already here in a lot of ways
  - Get warnings if MacOS apps compiled/released by someone without a signature
  - Windows 12 requiring hardware with TPM hardware
  - Some smartphones and game systems have been here a while
  - Linux on trusted hardware, Microsoft owns the keys and for now will sign a bootloader that will run Linux. Do we trust them?

Signed Firmware Tradeoffs

- Right to Repair Laws
- People concerned that car repair and cell phone repair becoming impossible as the companies lock things down with micocontrollers so you can only use officially approved parts
- Actually tractors and farm equipment big deal too
- Maine Ballot initiative on this November 2023

Firmware on Desktop/Laptop Systems

- What is the Pi GPU doing?
- What about the T2 processor on macs?
- New for ARMv8: ARM Trusted Firmware (ATF). Two standards, vendors have possibly made a mess of it already.
- Other platforms have it too. DRM to keep you from copying movies or video games.
- Windows 11 requiring TPM2 module

Firmware Hacking

- How can you figure out what the firmware is doing?
- Reverse engineering
- Looking for Security Issues
- Bypassing security measures

Firmware Security

- Encryption, but key has to live somewhere
  - If it's in ROM or RAM, can be dumped unless careful
  - Also might travel bus in open (original Xbox)
  - Decapping, people dissolve tops off chips and look at with microscope to see contents
- Extra logic gates (work by profs here at UMaine)
- Glitching
  - If give improper power (too little/too much), temperature (too cold/too hot), noise/sparks, etc, can

cause code to jump to places it shouldn't

- Poor user code
  - Save game bugs popular cause of jailbreaks on consoles
  - Console code can be super tight but if you sign a game from a customer and it has a bug, all might be lost

Locking down Hardware

- Many embedded boards, like STM32, you can "blow the fuses" after programming so that you can't read out or re-flash ROM
- Recent news: locked down STM32 boards, but when debugger hooked up still shows address of interrupts. What if move interrupt table into protected code and trigger interrupt? Code appears as interrupt address?

Why Lock Down Hardware

- Companies say it's for safety, security. Are there other reasons?
  - Profit. Can they lock you into one phone vendor? Lock you into one app store?
  - Secrecy. Don't want secrets of how their code works getting out.
  - "Piracy". Companies super worried you'll save video / music / games and then copy to your friends without paying for them. Digital Rights Management (DRM)

Question

- Instead of epoxying shut USB, just have OS ignore any USB ports?
  - In theory you could do that. Which is cheaper, unskilled labor epoxying USB or else writing your own kernel (or convincing Microsoft) to disable USB?
  - According to spec if OS is ignoring USB, are you safe?
  - (Ignoring the possibility of a USB device with super-capacitors on board to explode your system)
  - Past DMA attacks with Firewire which could bypass

OS and read/write memory

- What if hardware vendor for whatever reason talks to the USB device even though OS hasn't asked it to? It probably shouldn't, but how can you audit that? What about the firmware on the USB controller itself?
- The USB controller hooks up to a main CPU bus somehow, maybe over PCIe, maybe directly to CPU. It can potentially do lots of sneaky things.
- Why would there even be a controller for USB? Well when you plug in has to negotiate power, speed, and other stuff too. Probably a microcontroller

- It all depends how paranoid you want to be
- Even things like sandboxes and if you have all the code, can you *prove* the code is correct? That the compiler is trustworthy? This is actually a field of study, trying to prove code is correct. It's not trivial.