It’s assembler, Jim, but not as we know it!

Morgan Gangwere
DEF CON 26
Dedication

Never forget the shoulders you stand on.

Thanks, Dad
Hoopy Frood
Been fiddling with Linux SOCs since I fiddled with an old TS-7200 EmbeddedARM board
I’ve used ARM for a lot of services: IRC, web hosting, etc.
I’ve built CyanogenMod/LineageOS, custom ARM images, etc.
A word of note

There are few concrete examples in this talk. I’m sorry.

This sort of work is
   One part science
   One part estimation
   Dash of bitter feelings towards others
   Hint of “What the fuck was that EE thinking?”

A lot comes from experience. I can point the way, but I cannot tell the future.

There’s a lot of seemingly random things. Trust me, It’ll make sense.
ARMed to the teeth

From the BBC to your home.
Short history of ARM

Originally the Acorn RISC machine
Built for the BBC Micro!
Acorn changed hands and became ARM Holdings
Acorn/ARM has never cut silicon!
Fun fact: Intel has produced ARM-based chips (StrongARM and XScale) and still sometimes does!
The ISA hasn’t changed all that much.
Embedded Linux 101
Anatomy of an Embedded Linux device.

Fundamentally 3 parts
- Storage
- SoC/Processor
- RAM

Everything else? Bonus.
- PHYs on everything from I2C, USB to SDIO
- Cameras and Screens are via MIPI-defined protocols, CSI and DSI respectively

At one point, they all *look* mostly the same.
What is an SoC?

Several major vendors:
   Allwinner, Rockchip (China)
   Atheros, TI, Apple (US)
   Samsung (Korea)

80-100% of the peripherals and possibly storage is right there on die

Becomes a "just add peripherals" design

Some vendors include SoCs as a part of other devices, such as TI's line of DSPs with an ARM SoC used for video production hardware and the like.

In some devices, there may be multiple SoCs: A whole line of Cisco-owned Linux-based teleconferencing hardware has big banks of SoCs from TI doing video processing on the fly alongside a DSP.
What is an SoC?

- LTE modem
- Ethernet PHY
- I2C/SPI
- RAM (sometimes)
- SD Card/NAND/etc
- Peripheral Controller
- Storage Controller
- GPU
- CPU Core(s)

Internal bus
3. BLOCK DIAGRAM

Figure 3-1 shows the block diagram of the R8.

Figure 3-1. R8 Block Diagram
Bay Trail SOC Block Diagram
Storage

Two/Three common flavors

MTD (Memory Technology Device): Abstraction of flash pages to partitions
Storage

**Two/Three common flavors**

MTD (Memory Technology Device): Abstraction of flash pages to partitions

eMMC: Embedded MultiMedia Card
Storage

Two/Three common flavors

MTD (Memory Technology Device): Abstraction of flash pages to partitions

eMMC: Embedded MultiMedia Card, SPI SD card

SD cards
Storage

Two/Three common flavors

MTD (Memory Technology Device): Abstraction of flash pages to partitions

eMMC: Embedded MultiMedia Card, SPI SD card

SD cards
Storage

Two/Three common flavors

- MTD (Memory Technology Device): Abstraction of flash pages to partitions
- eMMC: Embedded MultiMedia Card, SPI SD card
- SD cards

Then there’s UFS

- Introduced in 2011 for phones, cameras: High Bandwidth storage devices
- Uses a SCSI model, not eMMC’s linear model
Variations

Some devices have a small amount of onboard Flash for the bootloader

Commonly seen on phones, for the purposes of bootstrapping everything else

Every vendor has different tools for pushing bits to a device and they all suck.

Samsung has at least three for Android
Allwinner based devices can be placed into FEL boot mode
Fastboot on Android devices
RAM

The art of cramming a lot in a small place

Vendors are seriously tight-assed

Can you cram everything in 8MB? Some routers do.
- The WRT54G had 8M of RAM, later 4M
- Modern SoCs tend towards 1GB, phones 4-6G

In pure flash storage, ramfs might be used to expand on-demand files (http content)
Peripherals

Depends on what the hardware has: SPI, I2C, I2S, etc are common sights.

Gonna see some weird shit
- SDIO wireless cards
- “sound cards” over I2S
- GSM modems are really just pretending to be Hayes AT modems.
- Power management, LED management, cameras, etc.
- “We need an Ethernet PHY” becomes “We hooked an Ethernet PHY up over USB”

Linux doesn’t care if they’re on-die or not, it’s all the same bus.
Peripherals

- SPI, I2C, I2S, etc.
- SDIO wireless cards
- "sound cards" over I2S
- GSM modems are really just pretending to be Hayes AT modems.
- Power management, LED management, cameras, etc.
- "We need an Ethernet PHY" becomes "We hooked an Ethernet PHY up over USB"
- Linux doesn't care if they're on-die or not, it's all the same bus.
Peripherals

Depends on what the hardware has: SPI, I2C, I2S, etc are common sights.

Gonna see some weird shit
  SDIO wireless cards
  “sound cards” over I2S
  GSM modems are really just pretending to be Hayes AT modems.
  Power management, LED management, cameras, etc.
  “We need an Ethernet PHY” becomes “We hooked an Ethernet PHY up over USB”

Linux doesn’t care if they’re on-die or not, it’s all the same bus.
Peripherals

Depends on what the hardware has: SPI, I2C, I2S, etc. are common sights.

Gonna see some weird shit like SDIO wireless, "sound cards" over I2S, GSM modems that pretend to be Hayes AT modems. Power management, LED management, cameras, etc.

"We need an Ethernet PHY" becomes "We hooked an Ethernet PHY up over USB".

Linux doesn't care if they're on-die or not, it's all the same bus.
Bootloader

One Bootloader To Rule Them All: Das U-Boot
  Uses a simple scripting language
  Can pull from TFTP, HTTP, etc.
  Might be over Telnet, Serial, BT UART, etc.

Some don’t use U-Boot, use Fastboot or other loaders
  Android devices are a clusterfuck of options
Life and death of an SoC*

DFU Check
- First chance to load fresh code onto device, use for bootstrapping and recovery

IPL
- Does signature checking of early boot image
- Probably SoC vendor provided

Bootloader
- Early UART/network wakeup is likely here.

Load Kernel into RAM
- Some devices are dumb and load multiple kernels until one fails or they run out
- U-Boot is really running a script here.

Start kernel
- Kernel has to wake up (or re-wake) devices it wants. It can’t make any guarantees.

Userland
- Home of the party.
- Where the fun attacks are

* Some restrictions apply
Life and death of an SoC*

* Some restrictions apply

1. **DFU Check**
   - First chance to load fresh code onto device, use for bootstrapping and recovery

2. **Early Boot**
   - Does signature checking of early boot image
   - Probably SoC vendor provided

3. **U-Boot**
   - Early UART/network wakeup is likely here.
   - Some devices are dumb and load multiple kernels until one fails or they run out
   - U-Boot is really running a script here.

4. **Load Kernel into RAM**
   - Kernel has to wake up (or re-wake) devices it wants. It can’t make any guarantees.

5. **Start kernel**
   - Home of the party.
   - Where the fun attacks are

---

*Some restrictions apply*
A root filesystem contains the bare minimum to boot Linux: Any shared object libraries, binaries, or other content that is necessary for what that device is going to do.

Fluid content that needs to be changed or which is going to be fetched regularly is often stored on a Ramdisk; this might be loaded during init's early startup from a tarball.

- This is a super common thing to miss because it's a tmpfs outside of /tmp
- This is a super common way of keeping / “small”
- This often leads to rootfs extractions via tar that seem "too big"

There are sometimes multiple root filesystems overlaid upon each other.

- Android uses this to some extent: /system is where many things really are

Might be from NFS, might try NFS first, etc.
Attacking these devices
Step 0: Scope out your device

Get to know what makes the device tick

- Version of Linux
- Rough software stack
- Known vulnerabilities
- Debug shells, backdoors, admin shells, etc.

ARM executables are fairly generic

- Kobo updates are very, VERY generic and the Kobo userland is very aware of this.

Hardware vendors are lazy: many devices likely similar to kin-devices

- Possibly able to find update for similar device by same OEM
Don’t Reinvent The Wheel

Since so many embedded linux devices are similar, or run similarly outdated software, you may well have some of your work cut out for you.

OWASP has a whole task set devoted to IoT security:

https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project

Tools like Firmwalker (https://github.com/craigz28/firmwalker) and Routersploit (https://github.com/threat9/routersploit) are already built and ready. Sometimes, thinking like a skid can save time and energy for other things, like beer!

Firmware Security blog is a great place to look, including a roundup of stuff (https://firmwaresecurity.com/2018/06/03/list-of-iot-embedded-os-firmware-tools/)
Option 1: It’s a UNIX system, I know this

If you can get a shell, sometimes just beating against your target can be fun.

Limited to only what is on the target (or what you can get to the target)

Can feel a bit like going into the wild with a bowie knife and a jar full of your own piss

Debugger? Fuzzer? Compiler? What are those?
Option 2: Black-Box it

Lots of fun once you’re used to it or live service attacks.

Safe: Never directly exposes you to “secrets” (IP)

You don’t have the bowie knife, just two jars of piss.
These options suck.
Option 3: Reverse it

Pull out IDA/Radare
Grab a beer
Learn you a new ISA
The way of reversing IoT things that don’t run Linux!

... but how the fuck do you get the binaries?
Yeah but I’m fucking lazy, asshole.

I don’t want to learn IDA. I want to fuzz.
Option 4: Emulate It

You have every tool at your disposal

   Hot damn is that a debugger?
   Oh shit waddup it’s fuzzy boiii

Once again, how the fuck do you get your binary of choice?
Getting root(fs)
Easy Mode: Update Packages

Updates for devices are the easiest way to extract a root filesystem

- Sometimes little more than a filesystem/partition layout that gets dd’d right to disk
- Android updates are ZIPs containing some executables, a script, and some filesystems
- Newer android updates (small ones) are very regularly "delta" updates: These touch a known filesystem directly, and are very small but don't contain a full filesystem.

Sometimes, rarely, they're an *actual executable* that gets run on the device

- Probably isn't signed
- Probably fetched over HTTP

Downside: They’re occasionally very hard to find or are incremental, incomplete patches. Sometimes they’re encrypted.
Medium: In-Vivo extraction

You need a shell
  Can you hijack an administrative interface?
  Some ping functions can be hijacked into shells
  Sometimes it’s literally “telnet to the thing”
  Refer to step 0 for more

You need some kind of packer (cpio, tar, etc)
  Find is a builtin for most busybox implementations.

You need some way to put it somewhere (netcat, curl, etc)

You might have an HTTPD to fall back on

Need to do reconnaissance on your device

Might need some creativity
  Wireshark, Ettercap, Fiddler, etc
Demo: Router firmware extraction (Actiontec Router)
What did we get?
Surprise Mode: Direct Extraction

Could be as simple as “remove SD card, image it”
Surprise Mode: Direct Extraction

Could be "remove SD card, image it"
Surprise Mode: Direct Extraction

Could be as simple as “remove SD card, image it”

Sever: The Anti-Villain Box (Canceled)

Get ready to no longer exist online.
Surprise Mode: Direct Extraction

Could be as simple as “remove SD card, image it”

eMMC is harder though, since you need to get to the data lines, but it can be done!

You will need to understand how the disk is laid out

   Binwalk can help later, as can “standard” DOS partition tables.

Having some in-vivo information is helpful
Surprise Mode: Direct Extraction

Could be as simple as "remove SD card, image it". eMMC is harder though, since you need to get to the data lines, but it can be done!

You will need to understand how the disk is laid out. Binwalk can help later, as can "standard" DOS partition tables. Having some in-vivo information is helpful.
Surprise Mode: Direct Extraction

Could be as simple as "remove SD card, image it". eMMC is harder, since you need to get to the data lines, but it can be done!

You will need to understand how the disk is laid out. Binwalk can help later, as can "standard" DOS partition tables.

Having some in-vivo information is helpful.
Surprise Mode: Direct Extraction

Could be as simple as "remove SD card, image it". eMMC is harder though, since you need to get to the data lines, but it can be done! You will need to understand how the disk is laid out. Binwalk can help later, as can "standard" DOS partition tables. Having some in-vivo information is helpful.

Surprise Mode: Direct Extraction

Could be as simple as “remove SD card, image it.”

EMMC is harder though, since you need to get to the data lines, but it can be done!

You will need to understand how the disk is laid out. Binwalk can help later, as can “standard” DOS partition tables.

Having some in-vivo information is helpful.
**Surprise Mode: Direct Extraction**

Could be as simple as “remove SD card, image it”

eMMC is harder though, since you need to get to the data lines, but it can be done!

You will need to understand how the disk is laid out
   Binwalk can help later, as can “standard” DOS partition tables.

Having some in-vivo information is helpful

**All Else Fails: Solder to the rescue**
   Might need to desolder some storage, or otherwise physically attack the hardware
   MTD devices are weird. Prepare to get your hands dirty

Interested in more? HHV and friends are the place to start looking.

JTAG, etc. might be the hard way out.
Logic Analyzers
Salae makes a good one
Cheap
Basic
Runs over USB
Hardware interfaces

http://dangerousprototypes.com/docs/Bus_Pirate

https://www.crowdsupply.com/excamera/spidriver
Now that we have that, what do?

Try mounting it/extracting it/etc. `file` might give you a good idea of what it thinks it might be, as will `strings` and the like.

eMMCs sometimes have real partition tables

SD cards often do

Look at the reconnaissance you did
- Boot logs: lots of good information about partitions
- Fstab, /proc/mounts
Let automation do your work

Binwalk!
- Takes a rough guess at what might be in a place
- Makes educated guesses about filesystems
- High false positive rate

Photorec might be helpful

Get creative
- Losetup and friends are capable of more than you give them credit for.
- There are a lot of filesystems that are read-only or create-and-read, like cramfs and such. These are often spotted by binwalk but are even sometimes seen as lzma or other compressed or high-entropy data

If you’re only looking to play in IDA/Radare/etc, the bulk extraction from binwalk might help.
QEMU: the Quick EMUlator

QEMU is a fast processor emulator for a variety of targets.

Targets you’ve never heard of?
- Mainframes
- ARM, MIPS
- PowerPC
- OpenRISC
- DEC Alpha

Lots of different ports and targets have been ported.
Did someone say “OSX on Amiga hardware”? 

Mac OS X 10.4 Tiger PowerPC works on QEMU 2.1.50 with KVM-PR enabled.

Host system: AmigaONE X1000 Nemo
"n" grab
get some help.

Or Haiku on BeOS?
Two ways to run QEMU

**AS A FULL FAT VM**

You preserve full control over the whole process

You’ve got access to things like gdb at a kernel level

Requires zero trust in the safety of the binary

But you probably want a special kernel and board setup, though there are generic setups to get you started

Any tools are going to need to be compiled for the target environment
  - I hate cross-compilers.

**AS A TRANSLATING LOADER**

You have access to all your existing x86-64 tools (or whatever your native tools are)

They’re not only native, but they’re running full-speed.
  - You can run AFL like it’s meant to

Runs nicely in containers

You don’t even need a container!
Full-Fat VM: 9 tracks of DOS
Binfmt: Linux’ way of loading executables

Long ago, Linux added loadable loaders

Originally for the purposes of running JARs from the command line like God and Sun Microsystems intended.

Turns out this is a great place to put emulators.

Debian ships with support for this in its `binfmt-misc` package.

QEMU can be shipped as a “static userspace” environment (think WINE)

Uses “magic numbers” – signatures from a database – to determine which ones are supposed to load what.

Fairly simple to add new kinds of binaries. You could actually execute JPEGs if you really wanted to?
QEMU as loader

**WITHOUT A CONTAINER**

Dumb simple to set up:
qemu-whatever-static <binary>
With binfmt, just call your binary.

Must trust that the executable
is not malicious

Might depend on your local
environment looking like its
target environment

This works best for static,
monolithic executables

**WITH A CONTAINER**

Bring that whole root filesystem
along!

Run it in the confines of a jail,
Docker instance, even
something like Systemd
containers

Might need root depending on
your container (systemd)

Great for when your binary
loads its own special versions of
libraries that have weird things
added to them
QEMU user Demo
AFL setup

Oh boy. Let’s talk about AFL.

AFL needs to compiled with QEMU support
   Magic sauce: CPU_TARGET=whatever
   ./build_qemu_support.sh

AFL needs to bring along the host’s libraries
   Easiest bound with systemd-nspawn

Don’t do this in a VM
   It hurts
AFL Demo
Wrapping up

What did we learn today?

- Hardware vendors are lazy
- Attacking hardware means getting creative
- QEMU is pretty neato
- AFL runs really slow when you’re emulating an X86
- Emulating an IBM mainframe.

Going forward:

- I hope I’ve given you some idea of the landscape of tools
- Always remember rule 0
More Resources

Non-Linux targets:
RECON 2010 with Igor Skochinsky: Reverse Engineering for PC Reversers:
JTAG Explained: https://blog.senr.io/blog/jtag-explained
https://beckus.github.io/qemu_stm32/ (among others)

Linux targets:
eLinux.org – Fucking Gigantic wiki about embedded Linux.
linux-mips.org – Linux on MIPS
Thank you
Keep on hacking.