Reverse Engineering Firmware Packing
Bratus RE W22

Prepared by River Loop Security
February 2022

River Loop Security

Intros for today

Ryan Speers

D'11 CS

Reverse engineering

Applied cryptography

Previous DoD Cybersecurity & Director at VC-backed security firm

Kareem ElFaramawi

RPI

Firmware reversing & unpacking

Cryptographic Reverse engineering

Speak up! We talk to each other enough...

We provide deep technical analysis across a range of embedded devices

Test

2009 – Present
Manual analysis of IoT for security vulnerabilities

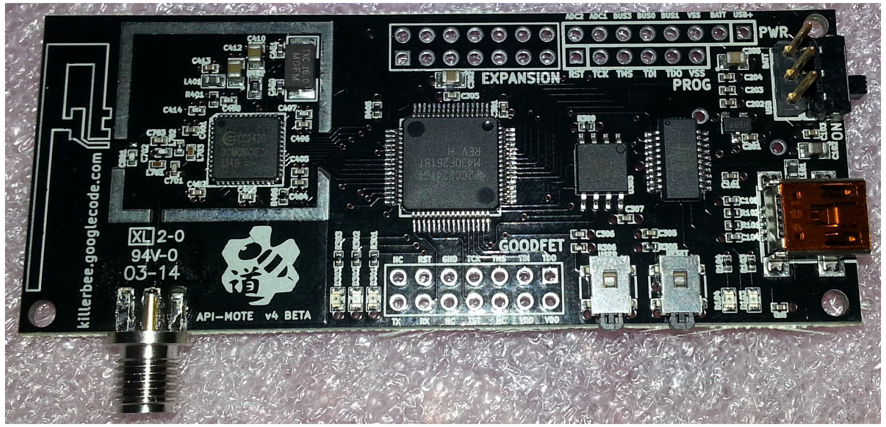
Design

2015 – Present
Design security into architecture for new products

Automate

2018 – Present
Automatically integrate security into the IoT product lifecycle

Open source contributions and research are a key parts of being active in the cybersecurity community



IEEE 802.15.4 Exploration Tools

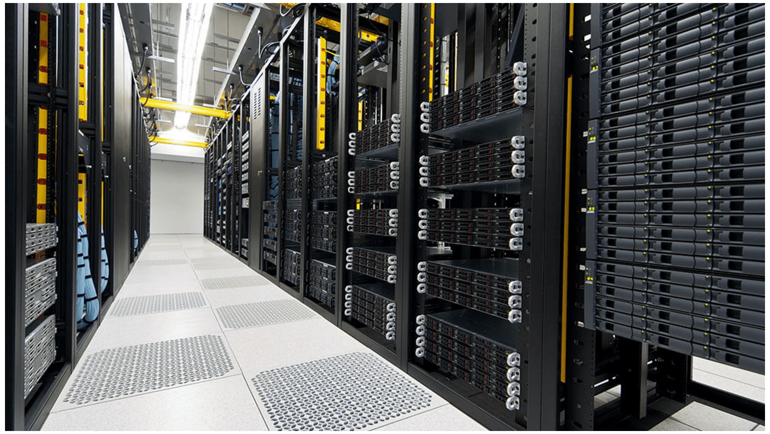
First researched at Dartmouth under
Prof. Smith & Bratus

Active Conference Presentations

DefCon, BlackHat, ShmooCon, Troopers,
EkoParty, NullCon, and others

Types of devices targeting

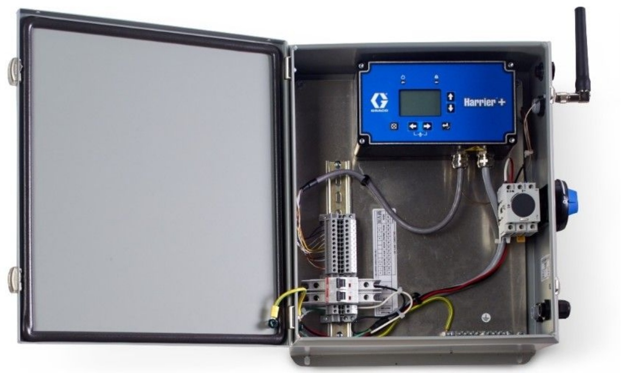
https://www.axis.com/en-rs/products/axis-p91-series



https://www.datacenters.com/news/companies-intend-to-purchase-more-on-premise-data-center-equipment

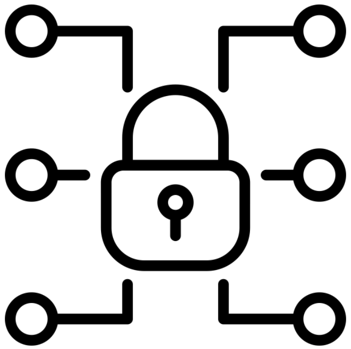
https://sepems.com/products/defibrillators/zoll-x-series-refurbished/

http://www.wired.co.uk/article/strangest-internet-of-things-devices



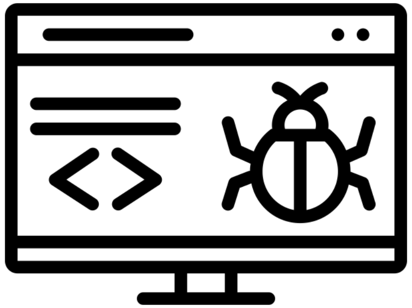
https://www.graco.com/us/en/in-plant-manufacturing/product/b52a00.html

Product security objective in a nutshell: **need to integrate security into the lifecycle**



Secure Design
Architect embedded systems securely from the start by designing for security

Supply Chain Security
Assure security in hardware supply chains to validate that specs make into final products

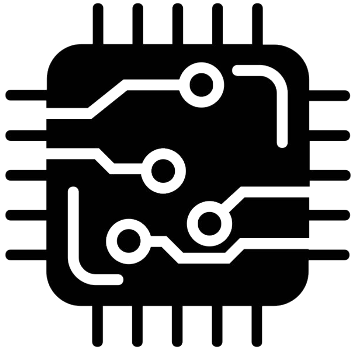


Penetration Testing
Test pre-prod and deployed products to find and remediate vulnerabilities

Vulnerability Response
Respond to disclosures in a timely, proactive, and responsible manner

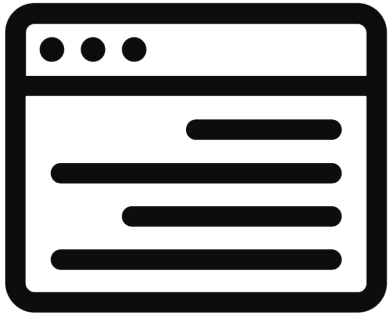
Automation & integration of security into culture and processes

Components at risk for IoT are broad – and sometimes hard to define



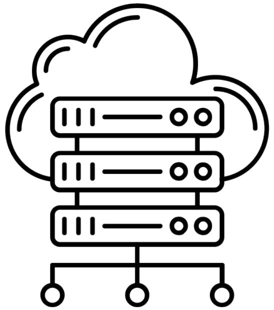
Hardware

PCB, chips, casing, all other "parts"



Firmware

Software, libraries, scripts, custom binaries



Back-end

Data store, API, logging, web app

TOC

Types of Firmware

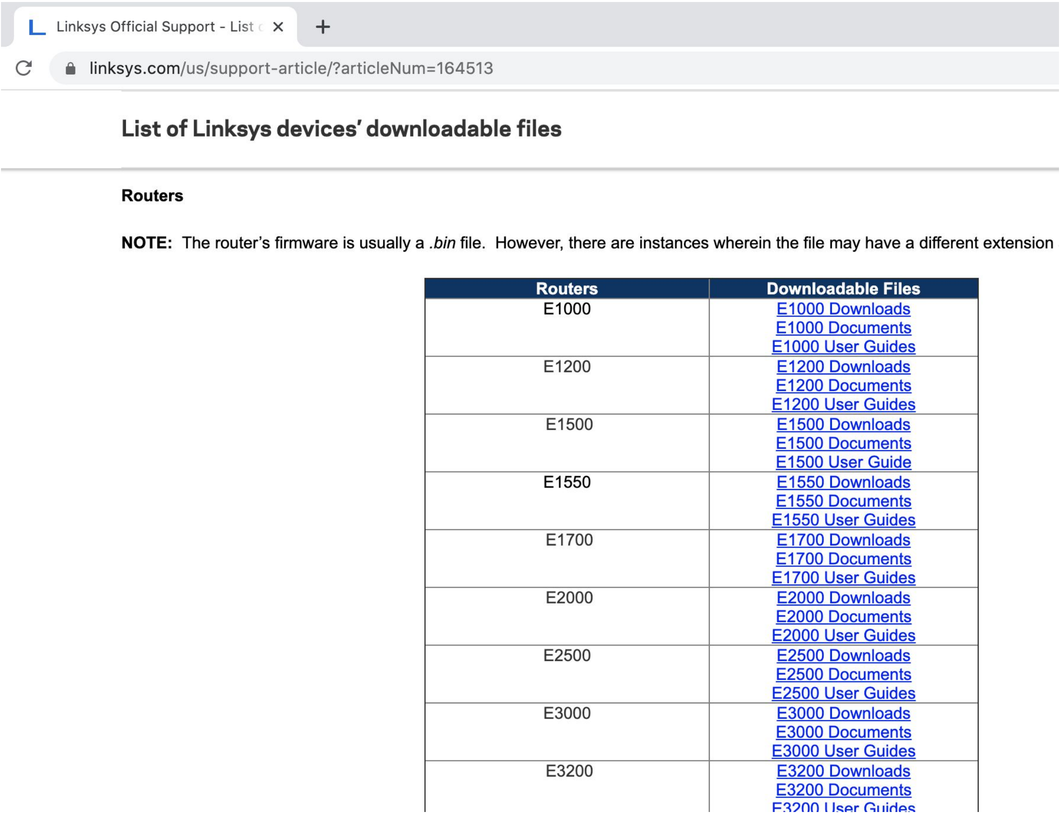
Types of Firmware

Processor Architectures
ARM
MIPS
x86
RISC-V, etc.

OS (or lack thereof)
Embedded Linux
RTOS (eCos, VxWorks, QNX, etc.)
Bare Metal
Embedded Windows (yuck)

Firmware – where we get it?

Vendor Downloads



Linksys Official Support - List

linksys.com/us/support-article/?articleNum=164513

List of Linksys devices' downloadable files

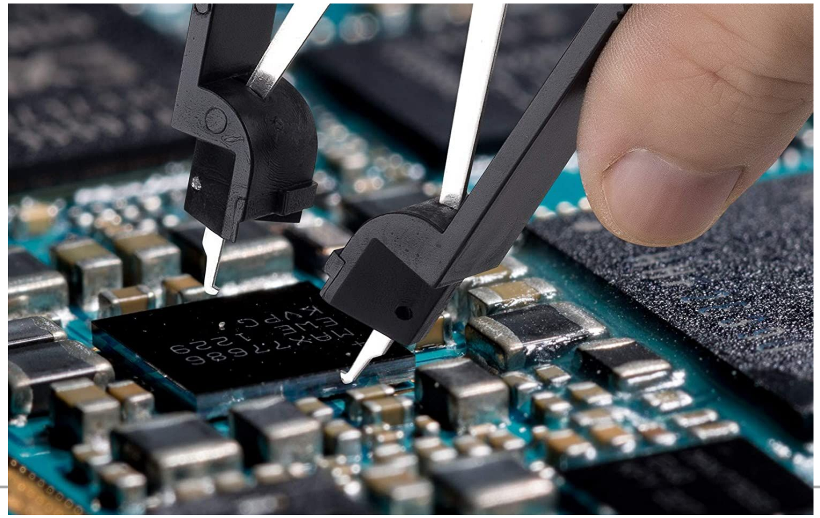
Routers

NOTE: The router's firmware is usually a *.bin* file. However, there are instances wherein the file may have a different extension

Routers	Downloadable Files
E1000	E1000 Downloads E1000 Documents E1000 User Guides
E1200	E1200 Downloads E1200 Documents E1200 User Guides
E1500	E1500 Downloads E1500 Documents E1500 User Guide
E1550	E1550 Downloads E1550 Documents E1550 User Guides
E1700	E1700 Downloads E1700 Documents E1700 User Guides
E2000	E2000 Downloads E2000 Documents E2000 User Guides
E2500	E2500 Downloads E2500 Documents E2500 User Guides
E3000	E3000 Downloads E3000 Documents E3000 User Guides
E3200	E3200 Downloads E3200 Documents E3200 User Guides

MITM of Updates/Downloads

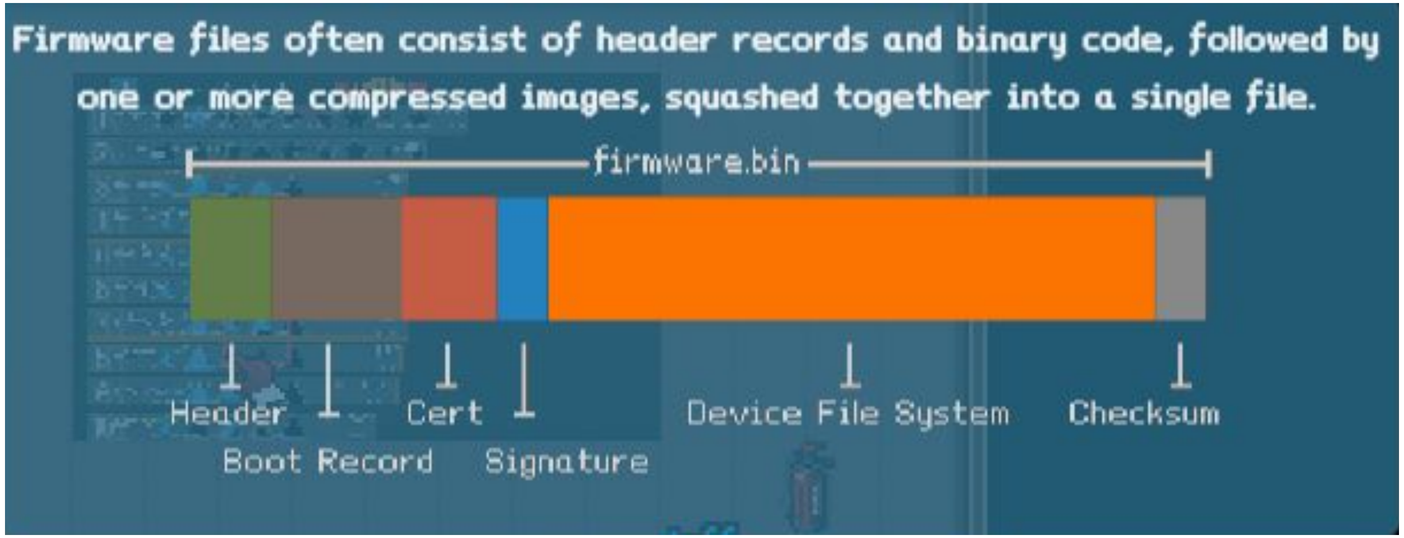
Hardware Chip Extraction



What’s in a firmware?

Firmware Contents

- Headers
  - Version Info & Type Info
  - Signature Information
- Contents
  - Kernel or OS
  - Filesystem(s)
  - Configuration Region(s)
- ...Multiple Contents
  - A/B Banking
  - Fallbacks/Recovery Images



Credit to mcpa.github.io image from SANS Hack Holiday 2015

OpenWRT Example from Documentation:

The generic Flash layout is:

Layer0	raw flash					
Layer1			OpenWrt firmware partition			
Layer2	bootloader partition(s)	optional SoC specific partition(s)	Linux Kernel	**rootfs** mounted: "/", OverlayFS with /overlay		optional SoC specific partition(s)
Layer3				**/dev/root** mounted: "/rom", SquashFS size depends on selected packages	**rootfs_data** mounted: "/overlay", JFFS2 "free" space	

Common Concepts in Firmware Packing

(and why…)

Compressed Filesystems

- Typical file systems we think of are R/W
  - For embedded, low overhead is key - how do we save space
    - Bigger compression blocks
    - Read-only constructs

- May or may not handle bad blocks (see ECC)
  - Doesn't handle: SquashFS, JFFS2, ...
  - Handles: UBI (Unsorted Block Image) which underlies UBIFS, ...

ECC

Memory has errors…

to handle these, store redundant data encoded into an ECC

many types of algorithms for this - from simple to very complex

"Raw NOR flash" - generally small (4 MiB – 16 MiB) and error-free

i.e. there cannot be bad erase blocks ->

the installed file system(s) don't need to account for bad erase blocks

"Raw NAND flash" - generally larger (32 MiB+) and not error-free

i.e. it may contain bad erase blocks ->

a system like UBIFS would be needed

How many bits can be detected, how many corrected?

Correct 1 bit errors, detect 2 bit errors: Hamming

Corrects multiple bits: BCH or Reed–Solomon

Using Entropy Edges

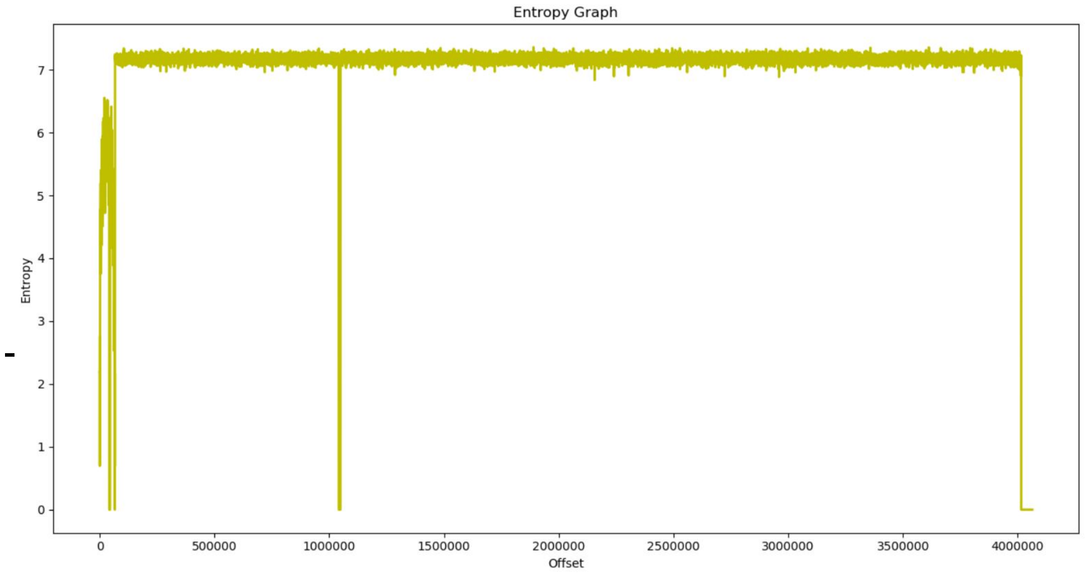
To get a feel for a firmware's blocks, if things are potentially compressed or encrypted, etc - use Shannon entropy.

$$H_2(X) = -\sum_{i=1}^{n} \frac{count_i}{N} \log_2\left(\frac{count_i}{N}\right)$$

```
DECIMAL       HEXADECIMAL  ENTROPY
--------------------------------------------------------------------------------
0             0x0          Rising entropy edge (0.995068)
10553344      0xA10800     Falling entropy edge (0.000000)

vs

DECIMAL       HEXADECIMAL  ENTROPY
--------------------------------------------------------------
0             0x0          Falling entropy edge (0.584989)
15360         0x3C00       Falling entropy edge (0.729575)
```



Common Implementations

(embedded linux focus)

Image/zImage/uImage

Linux Kernel Image Variants

zImage - self-extracting

uImage - wrapped in U-Boot info that includes type/loader info

squashfs

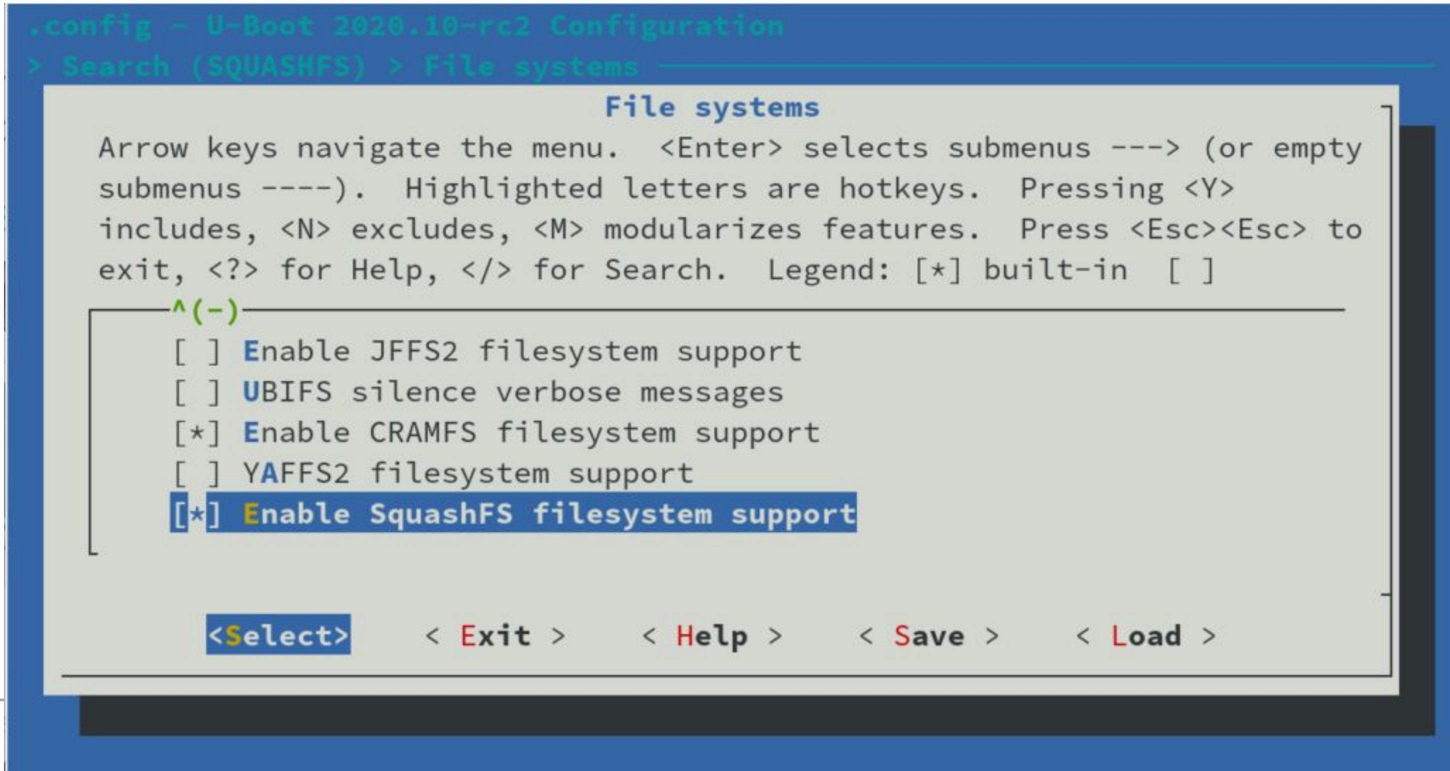
Compressed read-only filesystem for constrained memory environments.

can use many compressions incld. gzip, LZMA, LZO, ...

Used in: Live CD of Ubuntu/etc, Chromecast, OpenWrt routers, Linux AppImages, etc.

Age: 2002, added to Linux kernel 2009

Command lines: unsquashfs, mksquashfs



jffs2

Nodes can be:

- valid
- obsolete - newer version of the node elsewhere

Blocks (flash erase sized) are filled with nodes, these blocks are either:

- clean - only valid nodes
- dirty - at least one obsolete node
- free - no nodes

Garbage collection in background moves nodes to turn dirty to free blocks...

Notably requires all nodes to be scanned during mount – to figure out status/etc.

This means this solution doesn't scale well to bigger flash devices.

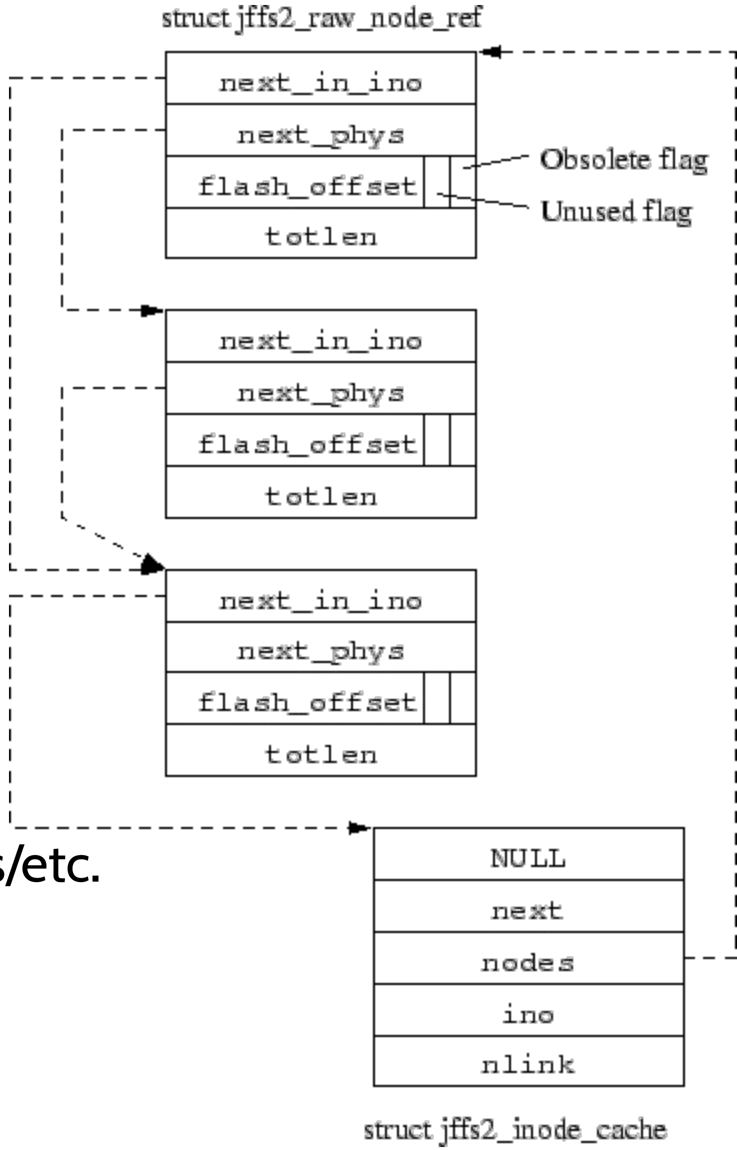


Image credit https://sourceware.org/jffs2/jffs2-html/node3.html

Intro to the Basic Toolkit

Manual analysis of firmware is critical, but not always scalable

Open-source tools (e.g., `binwalk`) are great to try to recover a filesystem and then investigate it for:

- Hardcoded keys, passwords
- Outdated libraries & binaries
- Configuration files
- Binaries to then reverse engineer manually

Fair warning:

- Complexity can quickly grow if open source tools don't work; need deep knowledge of binary formats to debug
- Many important items are outside the filesystems that are easy to look at (e.g., bootloader)
- Outdated libraries often compiled into binaries, …

Expect to need to try different variations

- Try manually carving a segment and looking at it for fingerprints yourself
  - did the magic detection have a false positive?
  - sanity check by computing checksums appropriate for the format, see if they match
- Is there a bit-order variation? BE/LE or even other swaps?
- If you dumped the chip, did you end up with OOB data in it that needs to be removed?
- Don't assume binwalk/etc will invoke the right scripts underneath it to unpack, try debugging and running these by hand to understand what they are seeing, and how to reason out the right steps to attempt.

Hands On: Unpack a Basic Embedded Linux Firmware

Hands On Section

SSH into your Linux workstation

CD to where the pre-provided file is (tenda_ac18...)

Execute binwalk on it to see the sections

Read through what it found via magic bytes

Execute binwalk -eMd3 on it to
- extract (e)
- recursively unpack (M) to depth 3 (d3)

CD into directory unpacked and explore

Walk-through: An Example of an Advanced Technique

REPAIRING A BROKEN HUAWEI NAND DUMP AND SINGLE-BIT ERRORS

BY KAREEM ELFARAMAWI | JULY 26, 2021

Introduction

One device that recently came across our desks was a Huawei EchoLife optical network terminal. As part of our standard analysis, we dumped the flash chip on the device in order to analyze the firmware. If you haven't already seen it, check out a previous Hardware Hacking 101 blog entry which goes over the basic process of identifying and dumping flash from a device.

In most cases, once we have a flash dump, an open-source tool like `binwalk` can handle the rest of the extraction. However, this was one of the rarer cases where considerably more work was needed before we could effectively extract the firmware to return the kernel and filesystems. In this blog post, we'll go over the process of finding out what was wrong with the flash dump and how we repaired it.

Repairing a NAND Dump

Normal Attempt at Unpacking

The first thing we typically do with a NAND dump is run it through a carving tool like `binwalk` to get an overall idea of its contents. This image was fairly small, and only came back with a few results.

```
$ ls -lh Micron_MT29F1G08ABAEA_00-07FFFFFF.bin
-rwxr--r-- 1 user user 128M Apr  1 14:28 Micron_MT29F1G08ABAEA_00-07FFFFFF.bin

$ binwalk -eM Micron_MT29F1G08ABAEA_00-07FFFFFF.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
88664         0x15A58         CRC32 polynomial table, little endian
90516         0x16194         CRC32 polynomial table, little endian
91682         0x16622         CRC32 polynomial table, little endian
1048576       0x100000        UBI erase count header, version: 1, EC: 0x0, VID header offset: 0x800, data
offset: 0x1000

$ tree -a _Micron_MT29F1G08ABAEA_00-07FFFFFF.bin.extracted
_Micron_MT29F1G08ABAEA_00-07FFFFFF.bin.extracted
├── 100000.ubi
└── ubifs-root
```

SEARCH

Search

CATEGORIES

blog (39)

conference (3)

conferences (7)

conferencess (1)

paper (5)

projects (5)

TAGS

010-EDITOR 802.15.4 AMAZON APIMOTE ARM BEEKEEPERWIDS BINARY-ANALYSIS CONFERENCES CORTEX-M CVE DARPA ECC

Walk-through:
An Example of
Non-Embedded-Linux
Unpacking

Hands On:
Unpack a Custom Embedded Linux Firmware

Questions