Firmware Analysis - 1

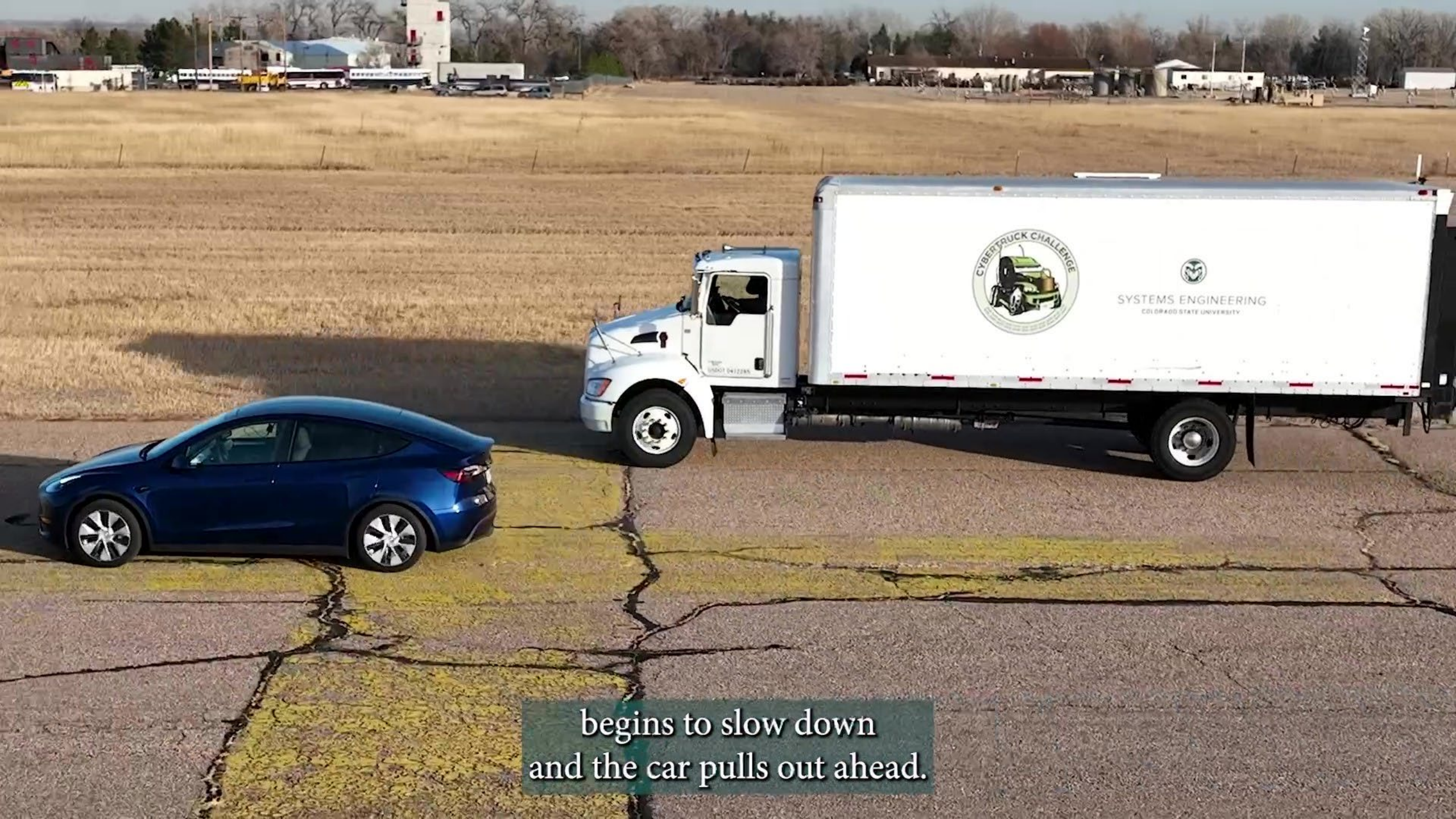
By Jake Jepson

About Me: Jake Jepson

- Security Researcher at Red Balloon Security
- Masters in Systems Engineering from CSU
- Bachelors in Computer Science with a concentration on Networks and Security from CSU


in as far as we know, the first ever
wireless drive by attack on a truck.


We were successful in this attack.
The car drove up along side it.



begins to slow down
and the car pulls out ahead.


About 30 secs later, the truck

Let's do that on a boat!

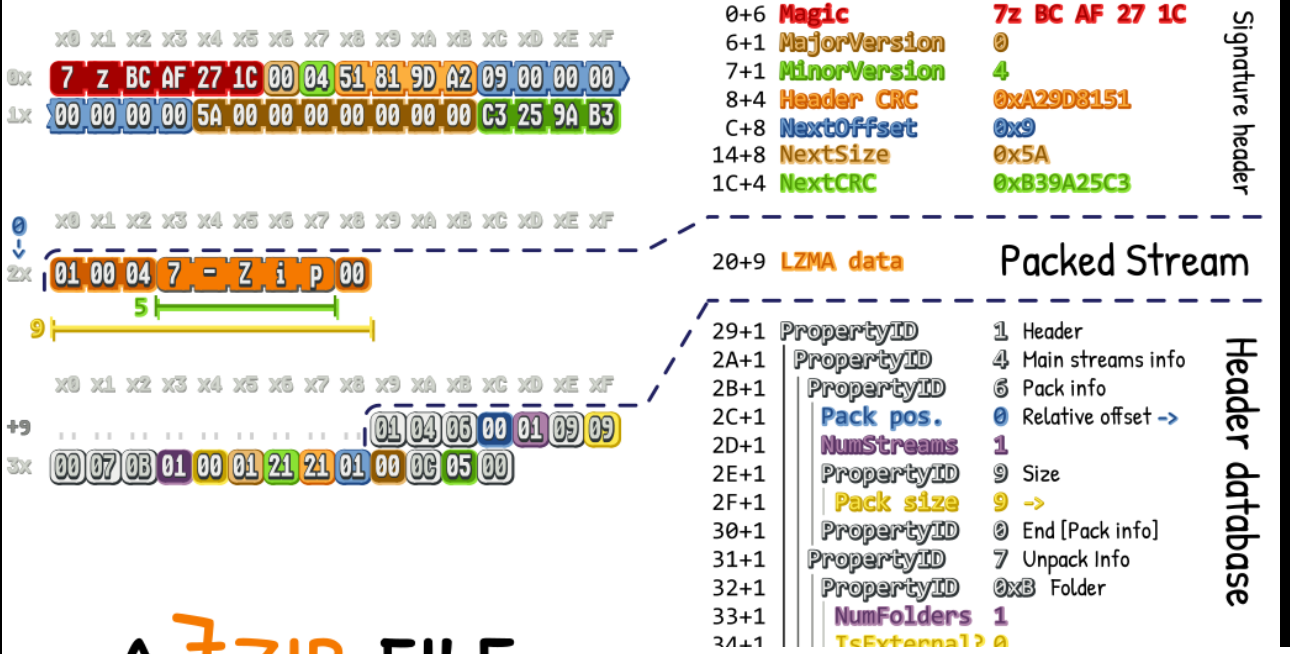
Class Overview

- What is Firmware?
- Tools and Techniques
- NAC-1 Autopilot Device Jump Start – I don't have an exploit developed, but I can help you get started.
  - Basic Recon
  - OTA firmware update exploration
  - Device firmware exploration

Take this time to boot into Linux if you have not done so already.

Rhetorical Introduction Questions

- What is Firmware?
  - From Chatgpt: “Firmware is a specific class of computer software that provides low-level control for a device’s specific hardware. Examples include BIOS and Embedded Systems.”
- How do you identify file types?
  - File extensions (.txt, .bin, .exe, …)
  - What if it doesn’t have a file extension? What if the file extension is vague? Ex: .bin



A 7ZIP FILE

https://github.com/corkami/pics

File Signature AKA Magic Bytes

- Beginning bytes of a file that identifies the file type.
- What if it doesn't have a file signature? Ex: Extracted firmware from device.

EXECUTABLE AND LINKABLE FORMAT

ANGE ALBERTINI
http://www.corkami.com

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
    0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:                         34 00 20 00 01 00

40: 01 00 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00

60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

MINI

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1 ELFCLASS32, 1 EL
EI_VERSION	1 EV_CURRENT
e_type	2 ET_EXEC
e_machine	3 EM_386
e_version	1 EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001
p_type	1 PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5 PF_R|PF_X

ELF HEADER
IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

PROGRAM HEADER TABLE
EXECUTION INFORMATION

Firmware Image Format

This is technical documentation for the firmware image format used by the ROM bootloader. These are the images created by `esptool.py elf2image`.

The firmware file consists of a header, an extended header, a variable number of data segments and a footer. Multi-byte fields are little-endian.

File Header

The image header is 8 bytes long:

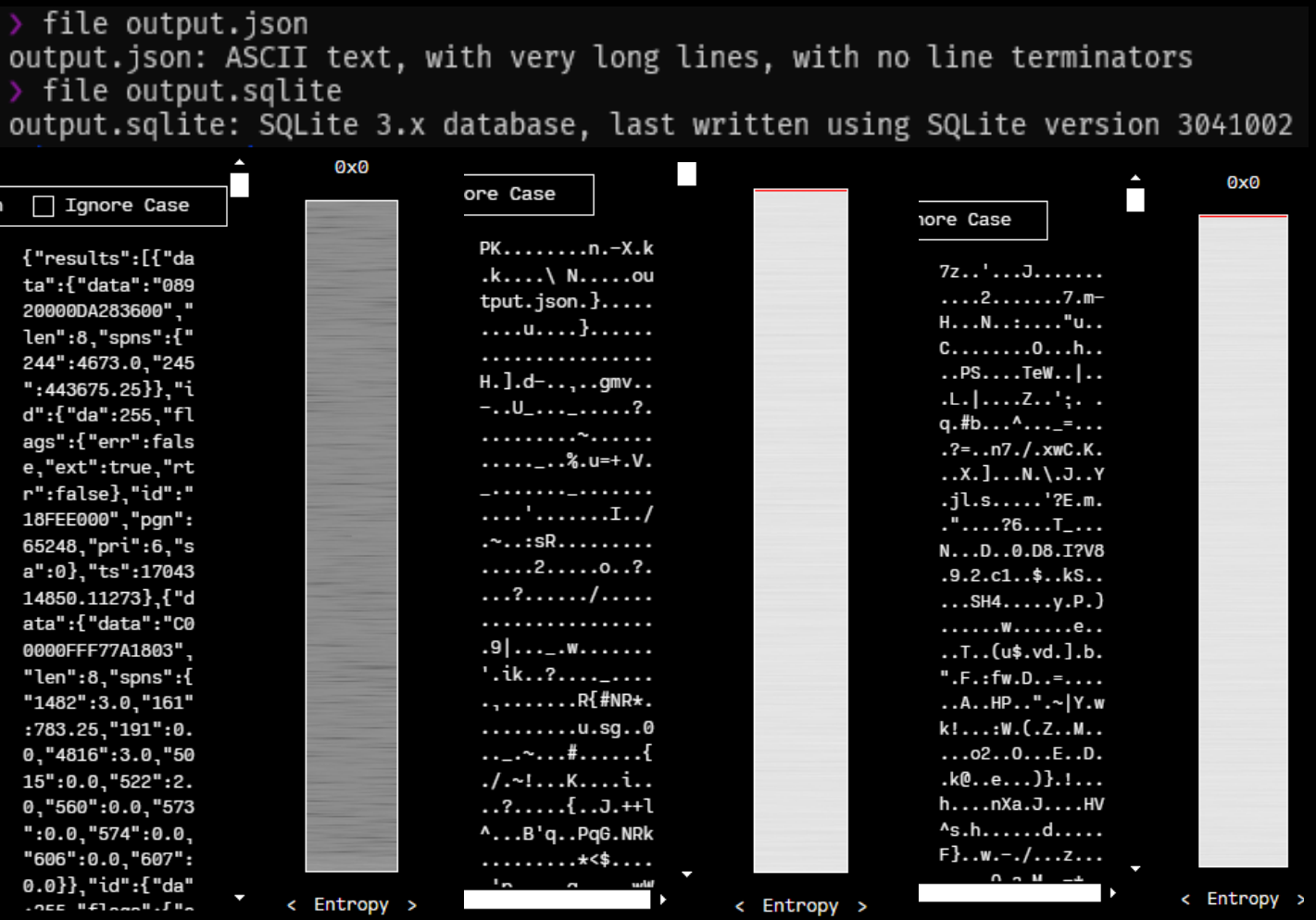
Byte	Description
0	Magic number (always `0xE9`)
1	Number of segments
2	SPI Flash Mode (`0` = QIO, `1` = QOUT, `2` = DIO, `3` = DOUT)
3	High four bits - Flash size (`0` = 1MB, `1` = 2MB, `2` = 4MB, `3` = 8MB, `4` = 16MB) Low four bits - Flash frequency (`0` = 40MHz, `1` = 26MHz, `2` = 20MHz, `0xf` = 80MHz)
4-7	Entry point address

`esptool.py` overrides the 2nd and 3rd (counted from 0) bytes according to the SPI flash info provided through the command line option. These bytes are only overridden if this is a bootloader image (an image written to a correct bootloader offset of 0x1000), in this case, the appended SHA256 digest is also updated to reflect the header changes. Generating images without SHA256 digest can be achieved by running `esptool.py elf2image` with the `--dont-append-digest` argument.

Context

- Ex: Extracted firmware from ESP device. (ESP/Espressif is a chip and board manufacturer) - The file type is ESP firmware.
- ESP32 Magic Byte is 0xE9. Is that enough to accurately identify the file as an ESP32 firmware image?

Other Techniques



json ~ 0.56 zip ~ 0.96 Encrypted 7z ~ 0.97

- Difficult to determine file type without unique file extension, file signature or context.
- Narrow down file type with additional techniques:
  - Encoding – Representation of data.
    - ASCII – plain text
    - Unicode – multi-language, emojis, etc.
    - Binary
    - Hundreds more…
  - Entropy – Measurement of randomness or unpredictability.
    - Encrypted: Very High
    - Compressed: High-Very High
    - Text/Binary/Other: Often Low to Medium

Some Common Tools

- file - Command-line tool that uses file signature database to identify the type of a file.
  - `Usage: file <filename>`
- binwalk - “Binwalk is a fast, easy to use tool for analyzing, reverse engineering, and extracting firmware images.” Also heavily relies on file signatures.
  - `Usage: binwalk <filename>`
- ofrak - A wrapper for many file identification and file manipulation tools including file and binwalk.
  - `Usage: ofrak gui and then upload the file and select Identify`
- strings - Extracts null-terminated strings from files. Does not work well with Rust and Go binaries as they don’t use null-terminated strings.
  - `Usage: strings <filename>`
- xxd/hexdump - Dumps the data in both hexadecimal and ascii format.
  - `Usage: <command> <filename>`
- There are many other file manipulation and identification tools out there.

Lab 0: Tool Familiarization – 4m

1. Linux is not required but highly recommended for these next couple of labs.
2. Grab a copy of the file `firmware_analysis_1_class_files` from the share.
3. Use file, binwalk, and ofrak to identify the file type and compare their results.
    1. `file <filename>`
    2. `binwalk <filename>`
    3. `ofrak gui` → upload file to webpage → then select `Identify`
4. Decompress/Extract the file based on the file type. Not sure how? Here are some examples:
    1. zip (.zip) → `unzip <filename>`
    2. 7z (.7z) → `7z e <filename>`
    3. tar (.tar.*) → `tar -xvf <filename>`
    4. gzip (.gz) → `gunzip <filename>` or `gzip -d <filename>`
    5. xz (.xz) → `xz -d <filename>`
    6. bzip2 (.bz) → `bzip2 -d <filename>`

Lab 0: Answer

- It was a Zip archive.
- It can be decompressed using:

```
unzip <filename>
```

- Revealing another file named: `whatami`

Lab 1: Puff the Magic Dragon Ate the Magic Bytes – 5m

- I have zero'd out the file signature of the `whatami` file.

1. What do the popular file identification tools tell us:
    1. File?
    2. Binwalk?
2. What's the entropy of the file?
    1. `binwalk -E <filename>`
3. What does binwalk's LZMA scan tell us? (LZMA is a popular compression algorithm with a somewhat unique pattern.)
    1. `binwalk -Z <filename>`
    2. Note: Scan takes awhile. Not necessary to complete the full scan.
4. What are some potential next steps to determine the file type?

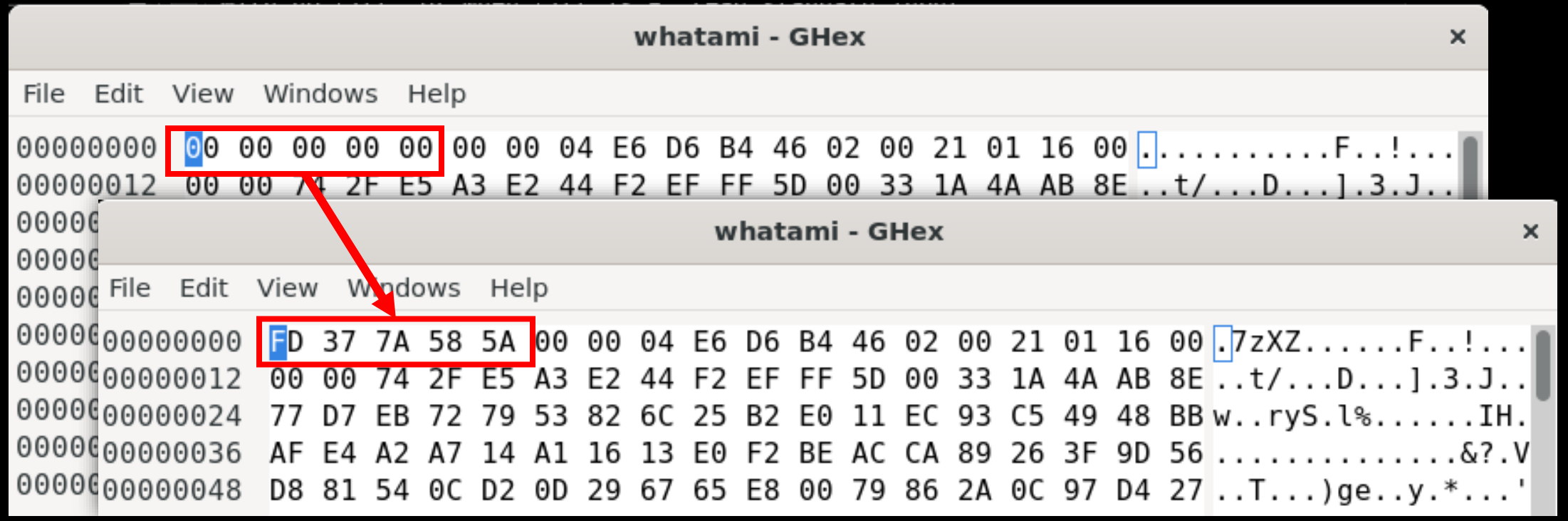
What else can we try?

- Multiple valid approaches to further narrowing down the file type including but not limited to:
  - Bruteforce – try different magic bytes, decompress, check results
  - Studying popular compression algorithm headers and comparing patterns.
- Let's try a ML/AI tool called Magika from Google
  - ML model trained to identify file types. 10M files. 120 Types.
  - Like most ML/AI tools its not perfect but sometimes works like *magic*.
- Go to: https://google.github.io/magika/ and upload the *whatami* file.
  - *Don't worry it runs locally in the browser…*
- What does Magika identify it as?

Lab 1: Answer

- The `whatami` file is an XZ compressed archive. Decompressing it would reveal a tarball.

1. Using a hex editor such as ghex. Open the file and fix the magic bytes like so:
    1. `ghex whatami`

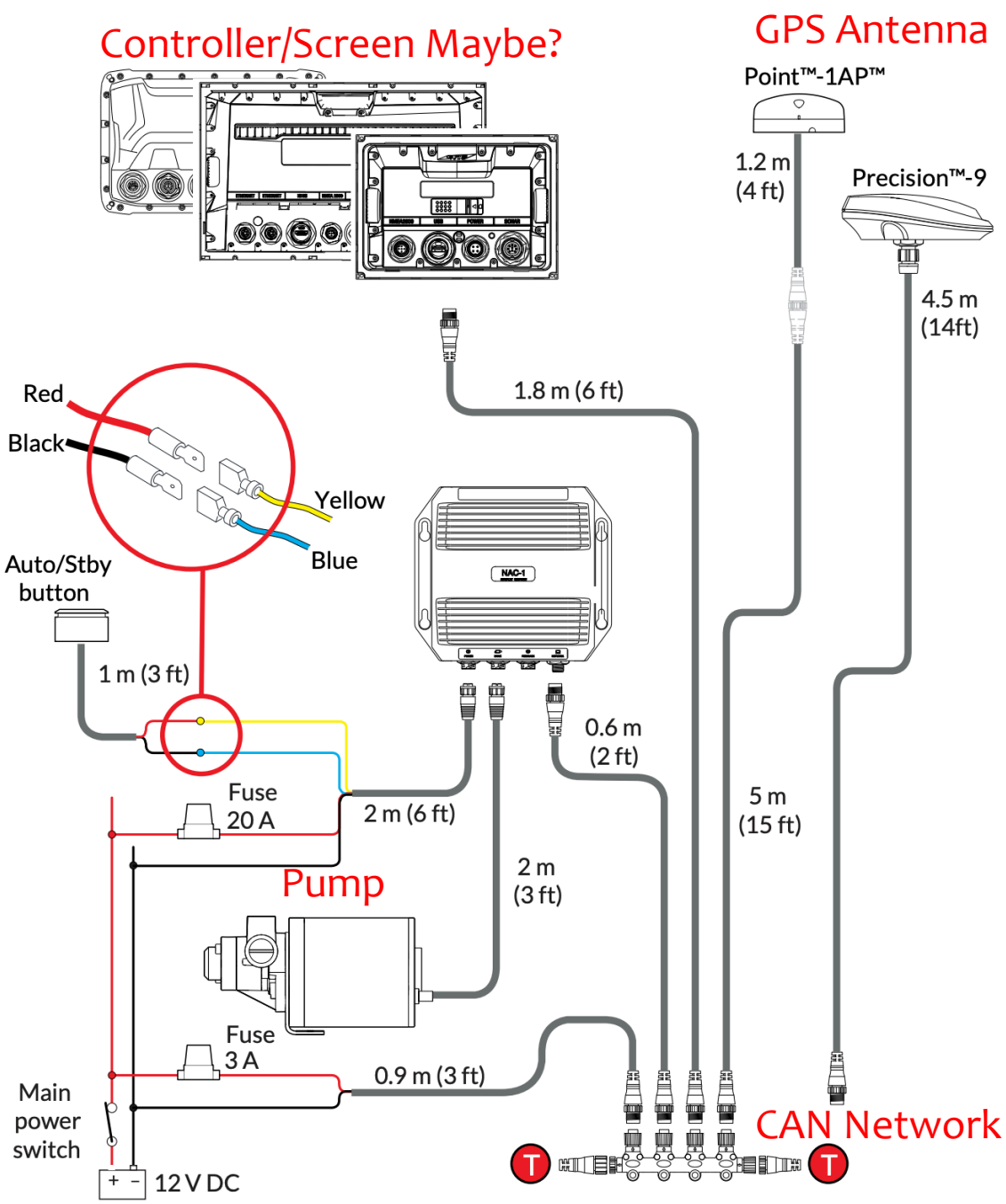


2. For simplicity decompress the archive and extract the files from the tarball using the command:
    1. `tar -xvf whatami`

Enough practice lets look at a real device!

NAC-1 Autopilot from Simrad enters the chat…

WIRING From NAC-1_Hydraulic_Pilot_Pack[…].pdf



NAC-1 Autopilot: Basic Recon

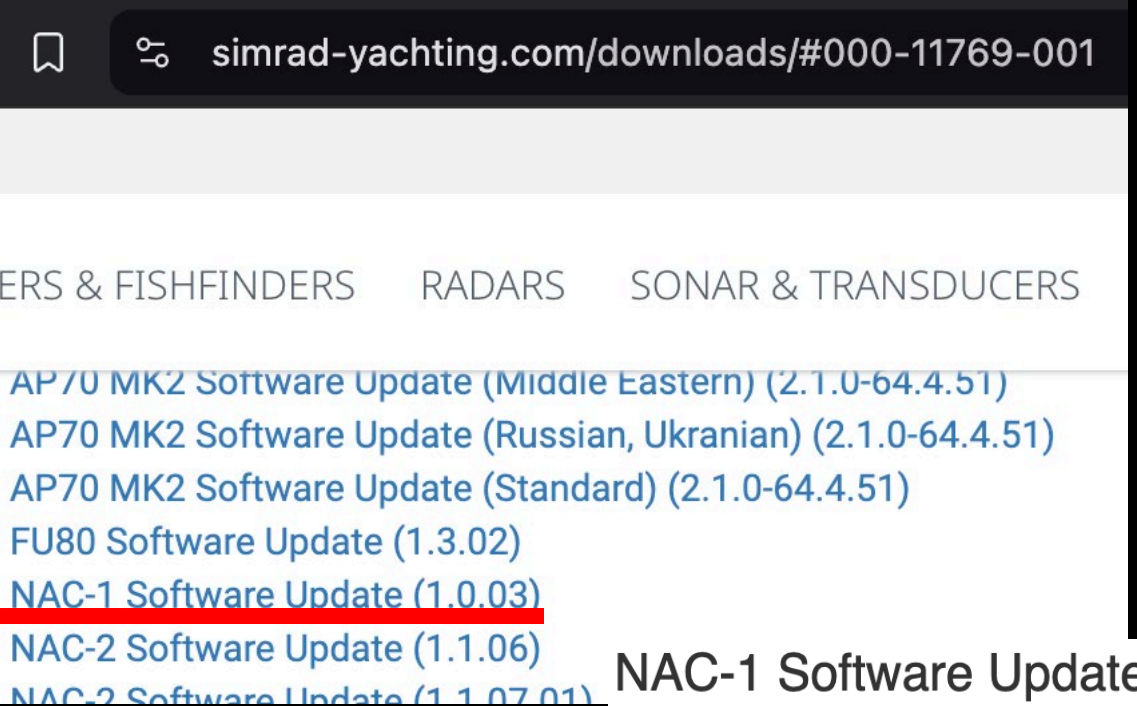
- What is it?
- What does it do in a boat?

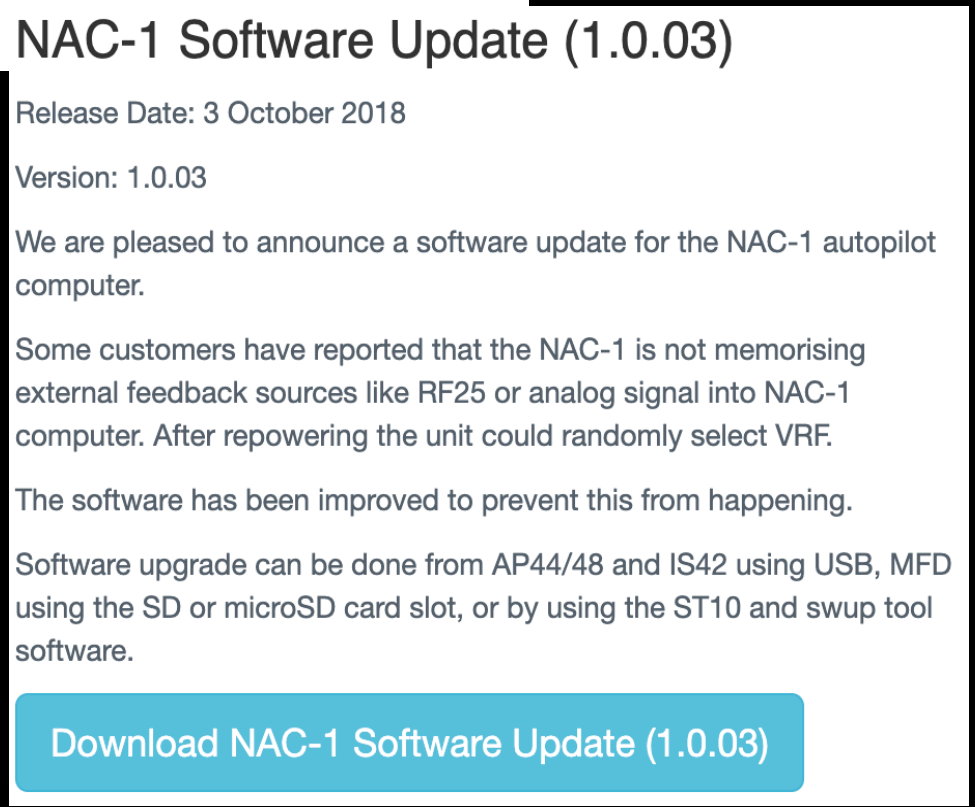
From https://www.simrad-yachting.com/

NAC-1 Autopilot Computer

SKU: 000-11769-001

The NAC-1 is the centre of your autopilot system, containing the electronics needed to operate and interface with other key components. It has been designed for hydraulic or cable-steered boats up to nine metres (30 feet) in length, and works with Simrad displays to provide a fully integrated autopilot solution.





NAC-1 Autopilot: Basic Recon

- Available software download… don't mind if I do…
- Software updates via multiple routes
  - AP44/48?
  - IS42?
  - ST10 and swup tool?

:tors/navico-can-to-usb-converter-st10/

SIMRAD

‹ Back to Cables & Connectors

ST10 Simnet Programming

SKU: 000-00020-001

ST10 Simnet programing Toolkit.

$169

Find a dealer

AP44 Autopilot Controller

SKU: 000-13289-001

The AP44 Autopilot Controller combines a full colour display with intuitive controls and modern glass helm styling. Free your hands from the wheel, cruise in comfort, and discover new ways to search for fish. Connect your chartplotter to enable waypoint navigation, and enjoy full autopilot control from compatible Simrad displays.

NAC-1 Autopilot: Basic Recon

- AP44/48 = Autopilot Controller
- IS42 = Display
- ST10 Simnet Programming Toolkit… Or is it?
  - Note URL: “navico-can-to-usb-converter-st10”
- Starting to piece together the picture:
  - Multiple devices can update the autopilot firmware over CAN.
- Further questions remain:
  - How do they secure it?
  - Is it secured or is it open?
  - Is the firmware encrypted or signed?

Lab 2: Firmware First Look 👀 - 4m

1. Navigate to `whatami_contents` to find the file:

   `NAC1_1_0_03.zip`

2. Decompress the archive using the command:

   `unzip NAC1_1_0_03.zip`

3. This should reveal a file named:

   `NAC1 1_0_03_00.swup`

4. Using any of the previously discussed tools determine the file type.

Lab 2: Firmware First Look 👀 – 4m

1. Navigate to `whatami_contents` to find the file:

   `NAC1_1_0_03.zip`

2. Decompress the archive using the command:

   `unzip NAC1_1_0_03.zip`

3. This should reveal a file named:

   `NAC1 1_0_03_00.swup`

4. Using any of the previously discussed tools determine the file type.

5. Answer: **XML**

6. Open `NAC1_1_0_03_00.swup` in a text editor and begin exploring its contents.

Lab 3: What now? 🤔 – 5m

- We know:
  - Its an XML file.
  - Contains a large amount of text data.
  - Somewhere in this file is a software update.
- From a quick skim of the file, we can observe:
  - Upload start address.
  - Checksum is used and its location.
  - Data length location.
- But where is the binary we can open in Binary Ninja, Ghidra, Angr, etc?

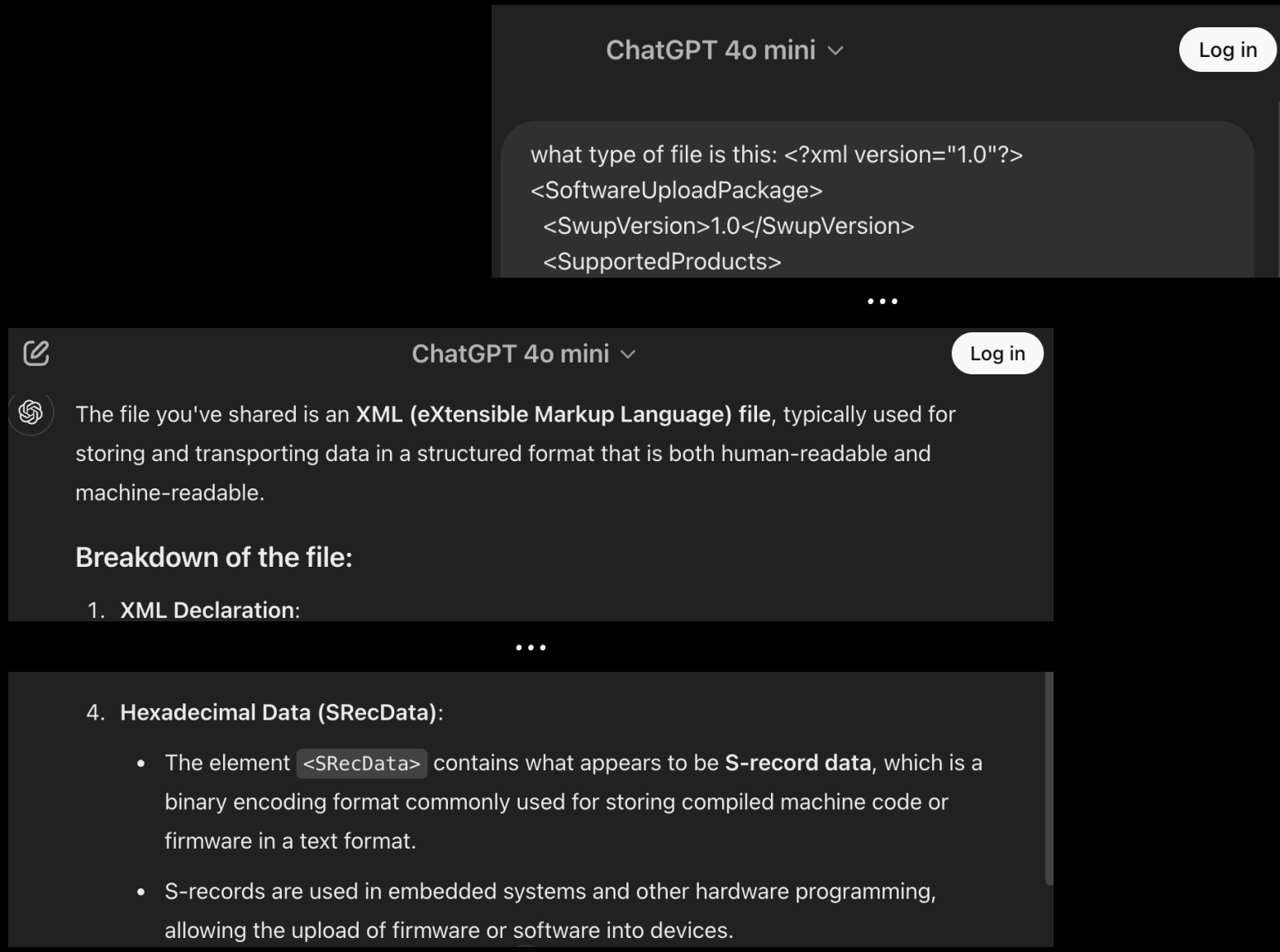
NAC1 1_0_03_00.swup

Users > jake > Desktop > NAC1 1_0_03_00.swup

```
1  <?xml version="1.0"?>
2  <SoftwareUploadPackage>
3    <SwupVersion>1.0</SwupVersion>
4    <SupportedProducts>
5      <Product Name="NAC1" Id="60003" SerialBitLength="14" DeviceClas
6    </SupportedProducts>
7    <SWVersionMajor>1</SWVersionMajor>
8    <SWVersionMinor>0</SWVersionMinor>
9    <SWVersionInternal>03</SWVersionInternal>
10   <SWVersionBuild>00</SWVersionBuild>
11   <UploadStartAddress>0×4000</UploadStartAddress>
12   <UserProgramStartAddress>0×4100</UserProgramStartAddress>
13   <ChecksumAddress>0×4004</ChecksumAddress>
14   <DataLengthAddress>0×4008</DataLengthAddress>
15   <SRecData>S02B0000633A2F6A656E6B696E732F776F726B73706163652F4D656
16 S21400410018F09FE518F09FE518F09FE518F09FE57A
17 S21400411018F09FE5586E20B920F11FE510F09FE5D6
18 S2140041203841000048420000 4C4200005042000067
19 S20C00413054420000 5842000052
20 S21400413820019FE5881090E5020411E30500000AB7
21 S2140041480110A0E3801080E5AA10A0E38C1080E59B
22 S2140041585510A0E38C1080E50010A0E3801080E5E1
23 S214004168AA10A0E38C1080E55510A0E38C1080E51B
24 S214004178A01190E5201081E3A01180E5A01190E53C
25 S214004188400011E3FCFFFF0A0110A0E30C1180E5D4
```

Lab 3: Answer

1. There are many valid ways to get this answer. Here's ChatGPT's answer.
2. Its a **S-Record** which is a basic ascii format for binary images.



Lab 4: SWUP_2_Binary.human – 5m

1. Extract the S-record from the SWUP using the included python program:

```
python3 swup_2_srec.py
```

2. This should generate a file called:

```
nac1_update_firmware.srec
```

3. Next, convert the S-record to binary by running:

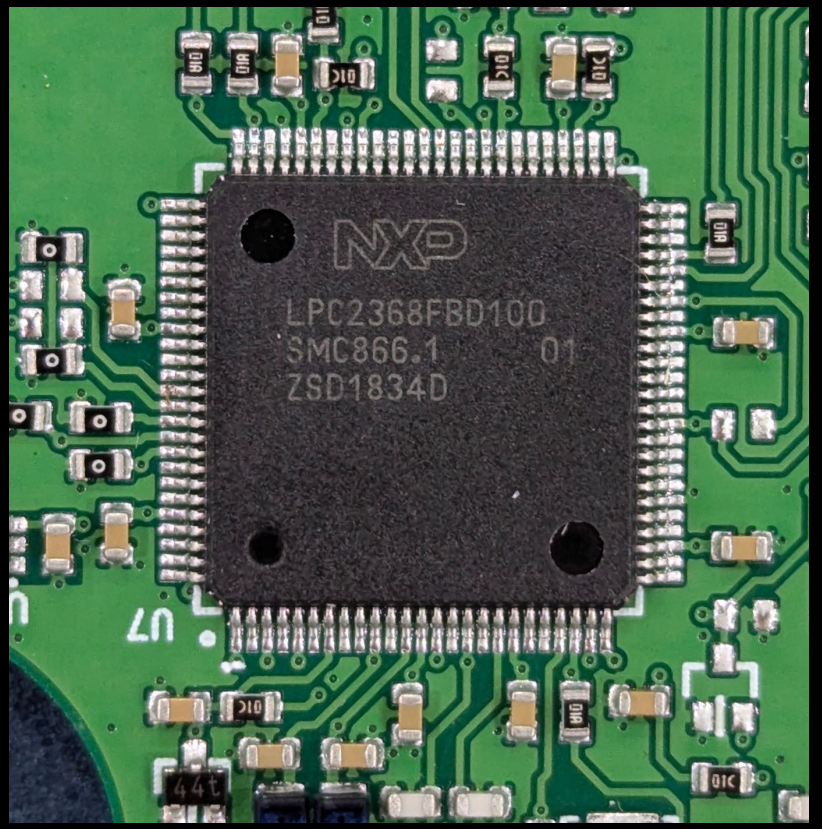
```
srec_cat nac1_update_firmware.srec -o nac1_update_firmware.bin -binary
```

4. **Note:** if you get an error about "srec_cat command not found" you can install it using:

```
sudo apt install srecord
```

Hold up. Let's Pivot.

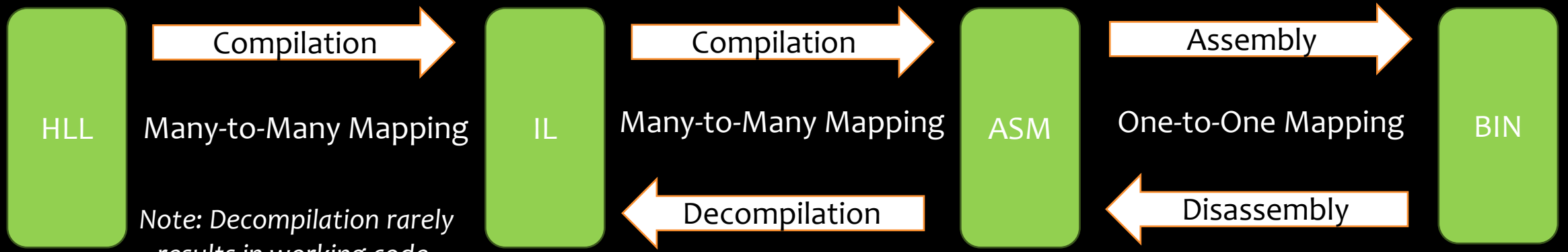
- Update files are often structured for distribution, not as full system representations.
- Decompilers rely on a memory map to link references accurately, but deriving this from update files is challenging without format details. *More on this later…*
- For device firmware, the chip's datasheet and reference manual provide the necessary memory map.
  - For this device the processor is a `NXP LPC2368` chip.
  - Datasheet: `LPC2364_65_66_67_68.pdf`
  - Reference Manual: `lpc23xx_um.pdf`
- Therefore, lets pivot to looking at the device's extracted firmware.



Hand waves entire HWRE process due to time constraints

SWRE Basics 1/2

- **Definition:** Software reverse engineering is the process of understanding how software works without having the original source code.
- **Levels of Abstraction:**
  - **Binary:** Processors follow instructions made up of 1's and 0's.
  - **Assembly:** A low-level language that closely represents machine instructions (binary).
    - **Architecture-Specific:** Different CPU architectures (x86, ARM) have their own assembly languages.
  - **Intermediate Language:** The various language representations between the high-level language and assembly.
  - **High Level Language:** Human readable, architecture independent language.



Note: Decompilation rarely results in working code. Hence the missing arrow.

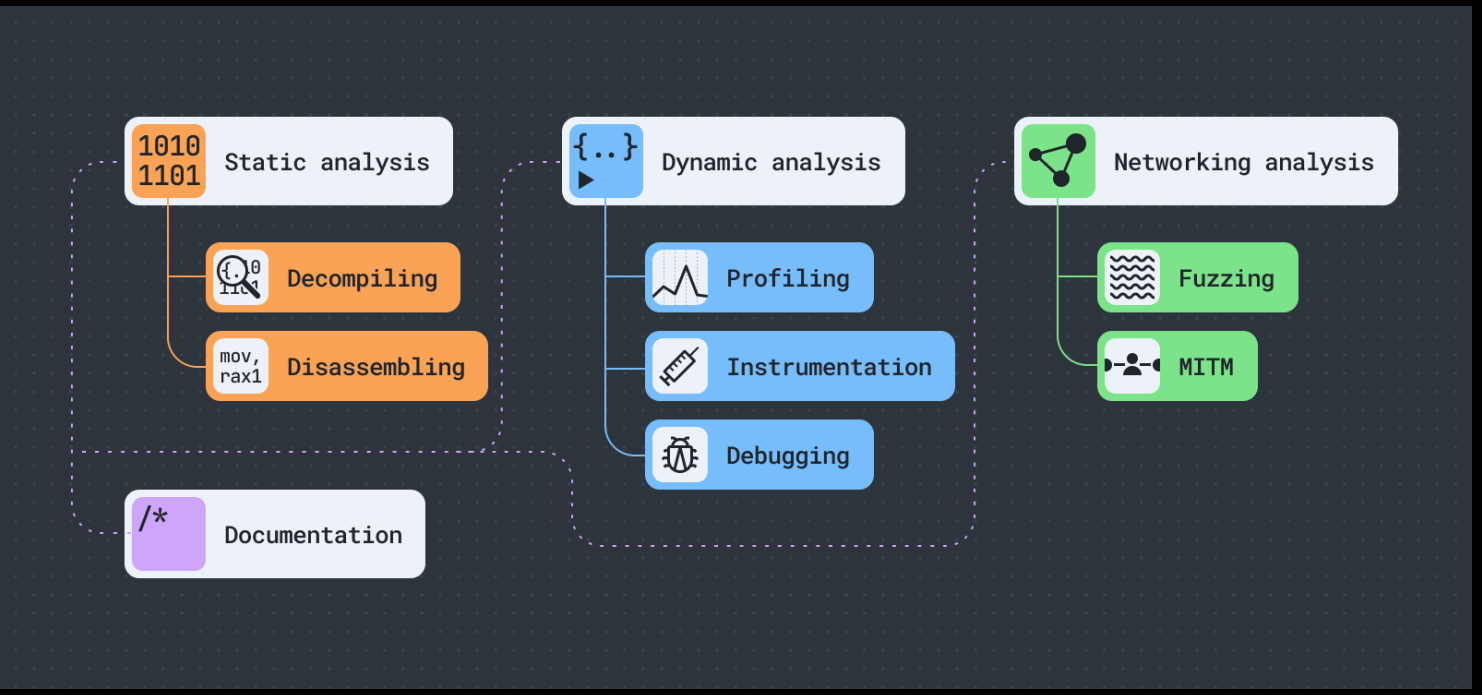


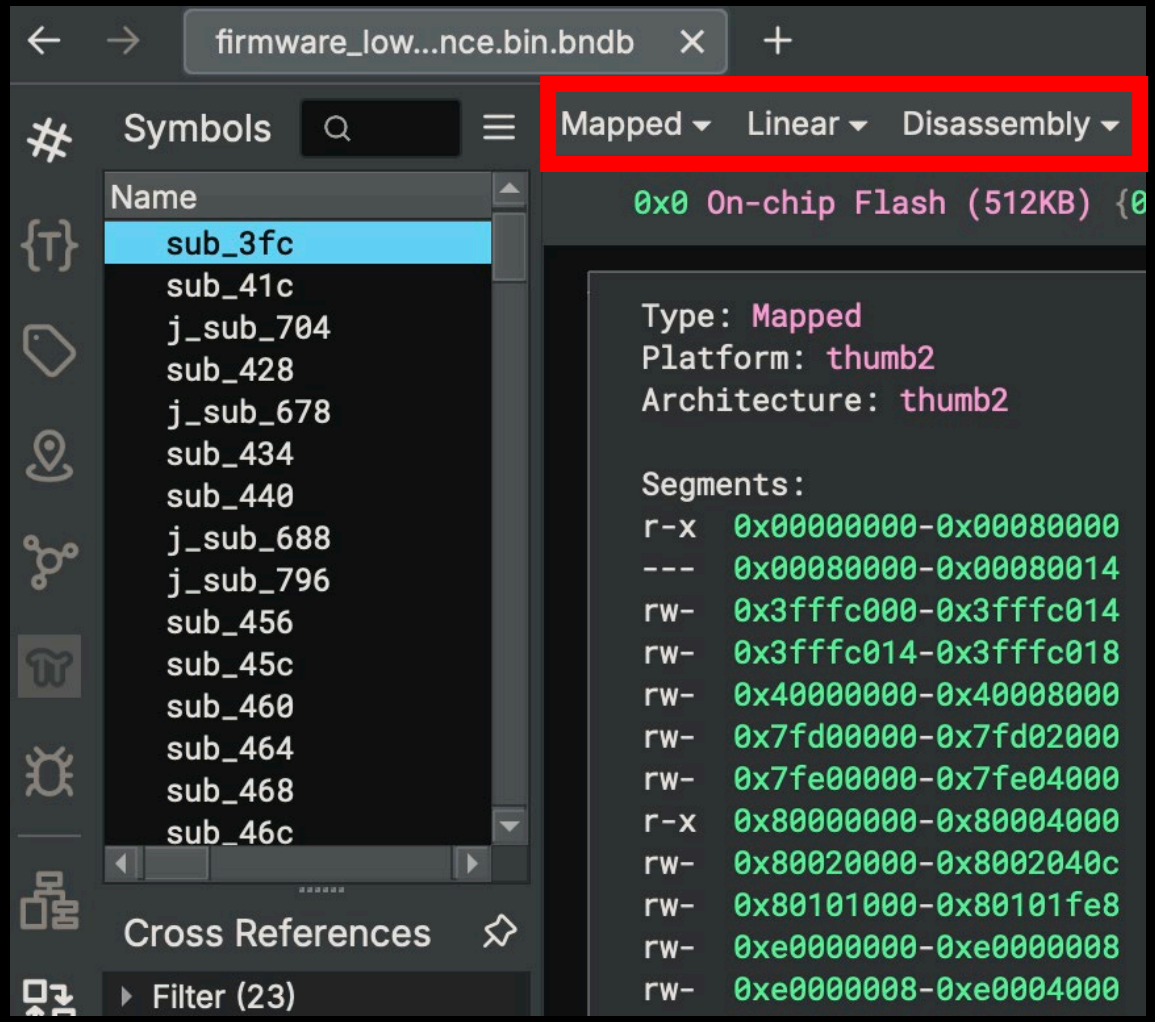
Image from: https://secretly.dev/images/reverse-engineering.png

SWRE 2/2

- **Static Analysis:**
  - **What:** Examining code or binaries without running them.
  - **Tools:** Disassemblers (e.g., IDA Pro, Ghidra), Decompilers, Hex Editors.
- **Dynamic Analysis:**
  - **What:** Observing how the program behaves when it runs.
  - **Tools:** Debuggers (e.g., x64dbg, OllyDbg), Virtual Machines, Sandboxes.
- Both have their limitations. We will be focusing on static analysis today.

Lab 5: Looking at the Firmware – 3m

1. Open the `binaryninja` folder on the desktop.
2. Double click the `binaryninja` application to start it.
3. Select `File -> Open` and open the firmware analysis database named: `firmware_lowrance.bin.bndb`
4. At the center top of the screen there are three drop downs that control different views.
5. Spend a couple of minutes exploring the different “views”.

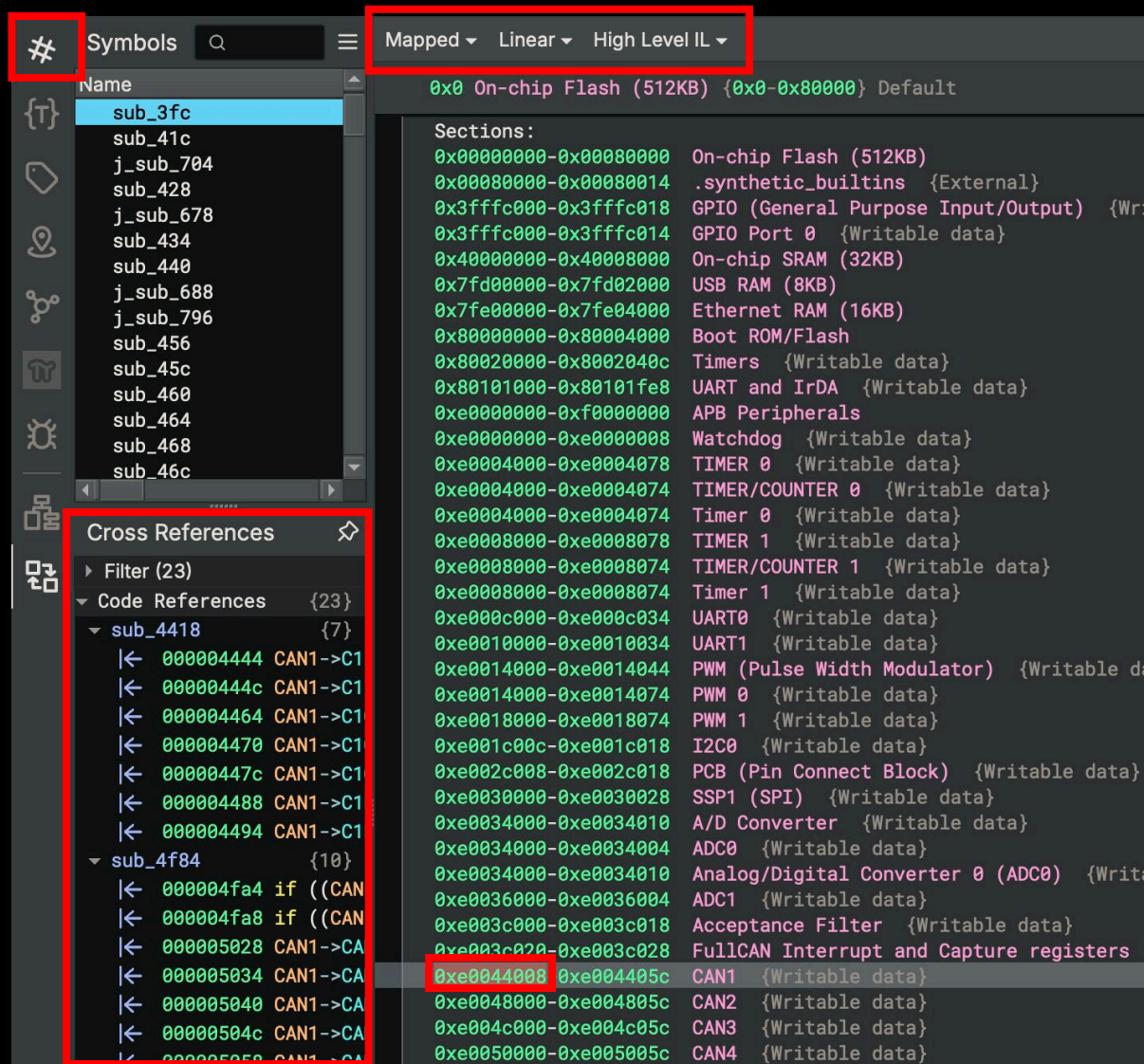


Lab 6: Scenario Setup

1. Say we want to send specific, potentially malicious CAN messages via modified autopilot firmware (similar to the earlier truck attack).
2. The normal high level CAN Message process:
    1. Code running on processor places values in specific memory/register locations (aka processor peripherals)
    2. Stuff occurs.
    3. CAN messages appear on bus.
3. By modifying the code responsible for sending CAN messages, we can alter the messages it transmits.
4. Using accurate peripheral memory maps, we can identify the code that references these peripherals. (I've setup the peripheral memory maps already)

Lab 6: Tracing References - 5m

1. With `firmware_lowrance.bin.bndb` open in Binary Ninja set the view selectors to:
    1. `Mapped` | `Linear` | `High Level IL`
2. Next in the main panel scroll to the `Sections` section (near the top of the page).
3. Click on the first address of the CAN1 section. Notice the `Code References` that appear on the left side of screen in the `Cross References` panel.
4. Explore these references and try to determine what these functions may do.



Lab 6: Answer

1. Without further testing we can't say for sure, but we can observe that:
    1. `sub_4418` interacts with CAN interrupt, status, timing, and error registers.
    2. `sub_4f84` is the only function that interacts with the CAN transmit frame register.
    3. `sub_56cc` is the only function that interacts with the CAN receive frame register.
2. Therefore, we can predict:
    1. `sub_4418` -> `can_setup`
    2. `sub_4f84` -> `can_write`
    3. `sub_56cc` -> `can_read`