Extracting an ELF from an ESP32

By: Chris Lyne and Nick Miles

tenable

Who we are

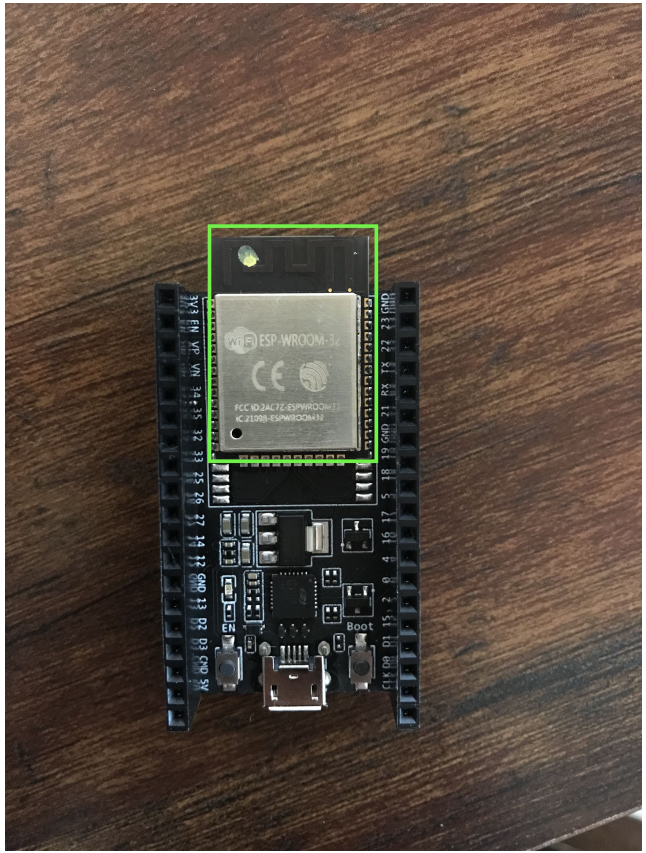
Chris Lyne
@lynerc

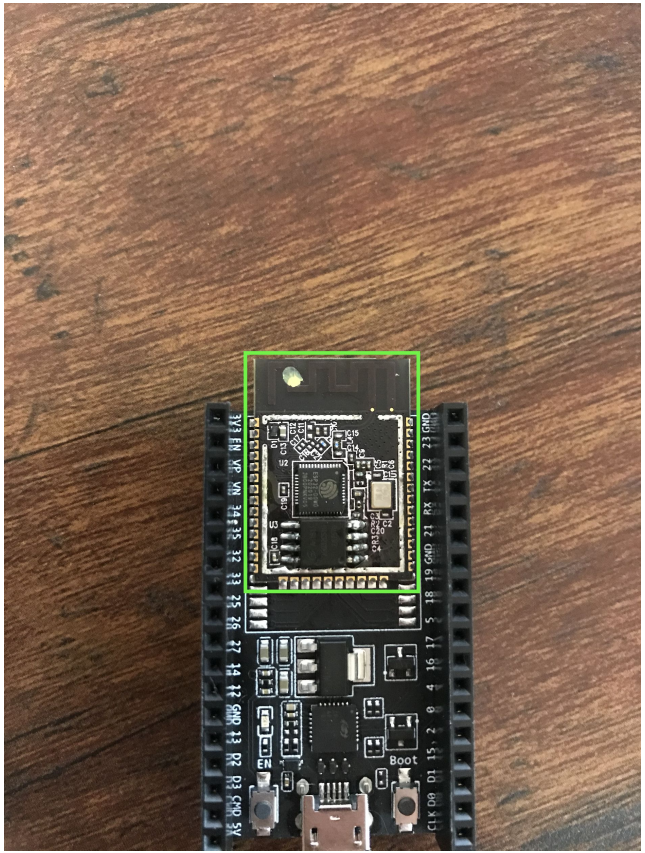
Nick Miles
@_NickMiles_

tenable

ESP32 Background

ESP-WROOM-32





tenable

What is an ESP32?

- System on a Chip (SoC)
  - Wi-Fi (2.4 GHz band)
  - Bluetooth 4.2
  - Dual high performance cores
  - Ultra Low Power co-processor
  - Several peripherals

Source: https://docs.espressif.com/projects/esp-idf/en/latest/get-started/index.html

tenable

ESP32 Design

- Dual-core with Xtensa LX6 CPUs
- All memory and peripherals are located on the data bus and/or the instruction bus of these CPUs.
- With some minor exceptions (see below), the address mapping of two CPUs is symmetric, meaning that they use the same addresses to access the same memory
- Multiple peripherals in the system can access embedded memory via DMA.
- The two CPUs are named “PRO_CPU” and “APP_CPU” (for “protocol” and “application”), however, for most purposes the two CPUs are interchangeable.
- Embedded Memory
  - 448 KB Internal ROM
  - 520 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory

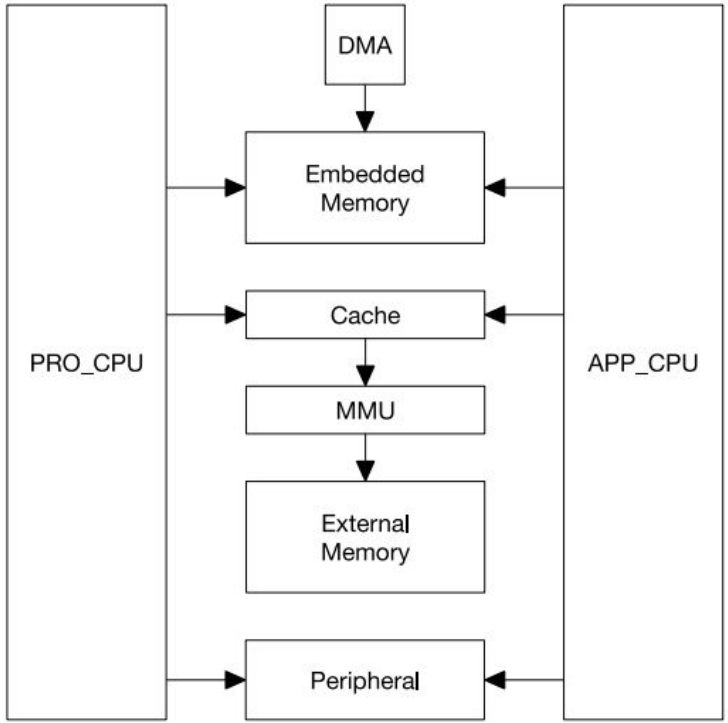


Figure 1: System Structure

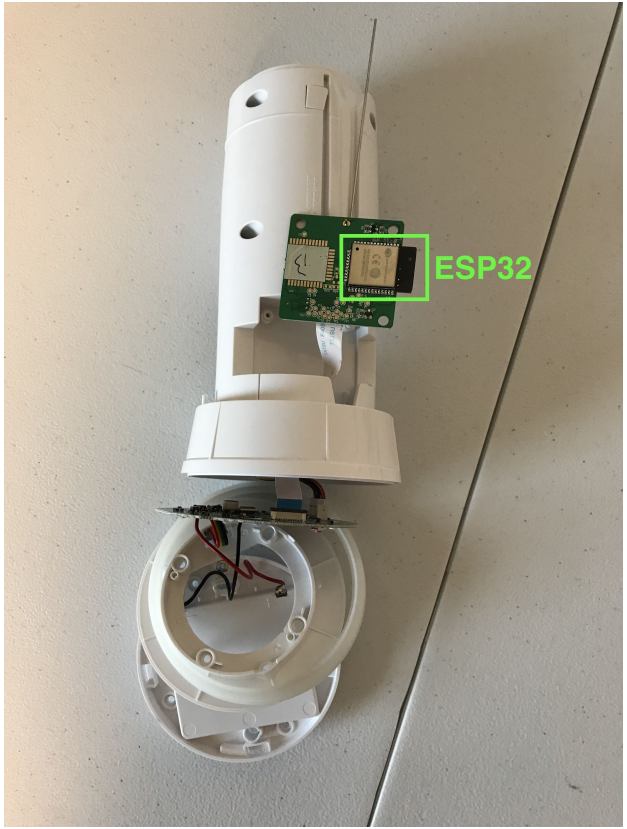
Source: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

SimpliSafe SS3 Alarm System

- Base station
  - Brains of the operation
  - Cloud connectivity
  - RF sensors
    - Doors/windows
    - Motion detector
    - Etc
  - RF/Wi-Fi module

tenable

SimpliSafe RF Module

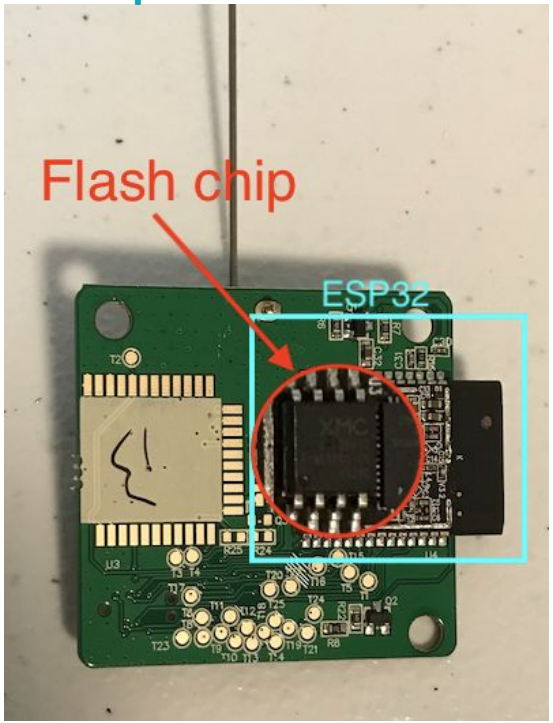


tenable

The problem we ran into

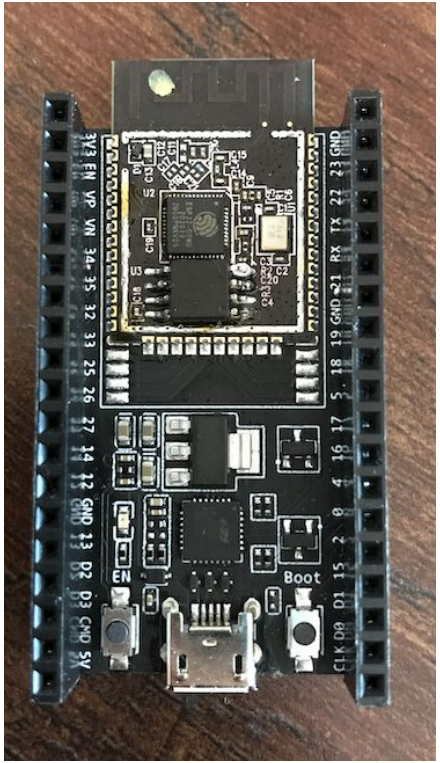
It’s “data”…

SimpliSafe RF Module



Remove flash with heat gun. Solder onto dev kit.

ESP32 Dev Kit




```
sh-3.2$ file esp32dump.bin
esp32dump.bin: data
```


tenable

binwalk

```
sh-3.2$ binwalk esp32dump.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
8272          0x2050          Unix path: /home/darrell/git/esp-idf/components/soc/esp32/rtc_clk.c
332736        0x513C0         Unix path: /mnt/ss3_fw_source/esp-idf/components/esp32/./crosscore_int.c
332968        0x514A8         Unix path: /mnt/ss3_fw_source/esp-idf/components/esp32/./heap_alloc_caps.c
333784        0x517D8         Unix path: /mnt/ss3_fw_source/esp-idf/components/esp32/./intr_alloc.c
335468        0x51E6C         Unix path: /mnt/ss3_fw_source/esp-idf/components/freertos/./heap_regions.c
336444        0x5223C         Unix path: /mnt/ss3_fw_source/esp-idf/components/freertos/./timers.c
336900        0x52404         Unix path: /mnt/ss3_fw_source/esp-idf/components/newlib/./locks.c
337320        0x525A8         Unix path: /mnt/ss3_fw_source/esp-idf/components/soc/esp32/rtc_clk.c
337808        0x52790         Unix path: /mnt/ss3_fw_source/esp-idf/components/spi_flash/./spi_flash_rom_patch.c
338692        0x52B04         Unix path: /mnt/ss3_fw_source/esp-idf/components/vfs/./vfs.c
351836        0x55E5C         Unix path: /mnt/ss3_fw_source/main/./device_nv.c
353121        0x56361         PEM certificate
354828        0x56A0C         Unix path: /mnt/ss3_fw_source/esp-idf/components/app_update/./esp_ota_ops.c
357656        0x57518         Unix path: /mnt/ss3_fw_source/components/at_core/./at_port.c
360740        0x58124         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/osi/future.c
361188        0x582E4         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/btc/core/btc_ble_storage.c
361480        0x58408         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/btc/core/btc_config.c
393096        0x5FF88         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/stack/btu/btu_task.c
450480        0x6DFB0         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/bta/sys/bta_sys_main.c
451348        0x6E314         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/btcore/bdaddr.c
452656        0x6E830         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/device/controller.c
454708        0x6F034         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/hci/hci_layer.c
455104        0x6F1C0         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/hci/hci_packet_factory.c
456124        0x6F5BC         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/hci/packet_fragmenter.c
457124        0x6F9A4         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/osi/alarm.c
457728        0x6FC00         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/osi/config.c
458580        0x6FF54         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/osi/fixed_queue.c
472420        0x73564         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/stack/btm/btm_ble_bgconn.c
503480        0x7AEB8         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/device/interop.c
504108        0x7B12C         Unix path: /mnt/ss3_fw_source/esp-idf/components/bt/bluedroid/hci/hci_hal_h4.c
506032        0x7B8B0         Unix path: /mnt/ss3_fw_source/esp-idf/components/driver/./rtc_module.c
511336        0x7CD68         Unix path: /mnt/ss3_fw_source/esp-idf/components/driver/./uart.c
518036        0x7E794         Unix path: /mnt/ss3_fw_source/esp-idf/components/driver/./gpio.c
521764        0x7F624         Unix path: /mnt/ss3_fw_source/esp-idf/components/esp32/./phy_init.c
540328        0x83EA8         Unix path: /home/xxt/work/code/esp32/ssc/components/smartconfig/./sc_sniffer.c
555736        0x87AD8         Unix path: /mnt/ss3_fw_source/esp-idf/components/freertos/./event_groups.c
556288        0x87D00         Unix path: /mnt/ss3_fw_source/components/libjansson/src/load.c
563384        0x898B8         PEM certificate
564808        0x89E48         SHA256 hash constants, little endian
578740        0x8D4B4         PEM RSA private key
578804        0x8D4F4         PEM EC private key
596892        0x91B9C         PEM RSA private key
598600        0x92248         PEM certificate
599812        0x92704         PEM RSA private key
601520        0x92DB0         PEM certificate
602736        0x93270         PEM RSA private key
```


tenable

Related tools/projects

- In the right direction, but not exactly what we wanted
  - https://boredpentester.com/reversing-esp8266-firmware-part-1/
    - Valuable insights, but ESP8266
  - https://github.com/themadinventor/ida-xtensa
    - Wouldn't load a flash dump. Need an ELF
  - https://github.com/jsandin/esp-bin2elf
    - Similar to our tool, but ESP8266
  - https://github.com/jrozner/esp-image-ida
    - Shoutout to jrozner. Found out about this too late, but similar goal to ours.

Our strategy

tenable

The goal

ESP32 Flash Dump

??

Our tooling

ELF

Analyze with IDA Pro

tenable

Step 1: Understand the Flash Dump

ESP32 Flash Dump

??

Our tooling

ELF

Analyze with IDA Pro

tenable

Bootloaders

- First stage
  - Stored in ROM
  - Initialization tasks
  - Loads second stage into RAM
- Second stage
  - Stored in flash at 0x1000
  - Mainly responsible for booting the application image

Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/general-notes.html

Boot Process

- Second stage bootloader
  i. Reads partition table from flash address 0x8000 (default location)
  ii. An app partition is chosen to boot up
  iii. Might be factory app or OTA

Flash Contents (high level)

...	Filled with 0xFF...
0x1000	Stage 2 Bootloader
0x8000	Partition Table
...	Other partitions as defined by the partition table

Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/general-notes.html

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0008,len:8
load:0x3fff0010,len:4400
load:0x40078000,len:11072
load:0x40080000,len:252
entry 0x40080034

(44) boot: ESP-IDF v3.0-dev-20-g9b955f4 2nd stage bootloader
(45) boot: compile time 14:06:42
(45) boot: Enabling RNG early entropy source...
(62) boot: SPI Speed      : 40MHz
(75) boot: SPI Mode       : DIO
(87) boot: SPI Flash Size : 4MB
(100) boot: Partition Table:
(111) boot: ## Label            Usage          Type ST Offset   Length
(134) boot:  0 phy_init         RF data          01 01 0000f000 00001000
(157) boot:  1 otadata          OTA data         01 00 00010000 00002000
(180) boot:  2 nvs              WiFi data        01 02 00012000 0000e000
(203) boot:  3 ble_data         unknown          40 00 00020000 00003000
(226) boot:  4 ota_0            OTA app          00 10 00040000 001e0000
(250) boot:  5 ota_1            OTA app          00 11 00220000 001e0000
(273) boot: End of partition table
(286) boot: Disabling RNG early entropy source...
(303) boot: Loading app partition at offset 00220000
```

Hardware Init

2nd stage bootloader loading...

Common partitions

- **nvs** - non-volatile storage is designed to store key-value pairs in flash
- **phy_init** - Initially was designed to be used for storing configuration parameters related to physical devices (e.g. Wi-Fi tx power, channel). It's no longer really used for that purpose anymore, these parameters are compiled into app and this partition can usually be deleted if necessary to save space.
- **factory** - factory application. Most applications remove this and use ota_0 as factory partition.
- **OTA Data** - over the air data (used to determine which application to boot. Factory or ota_x? Updated during ota updates)
- **OTA app** - over the air application (e.g. for firmware updates)

tenable

Step 1: Understand the Flash Dump

ESP32 Flash Dump

FF FF FF FF FF FF FF FF FF FF FF FF
Stage 2 Bootloader
Partition Table
NVS
otadata
phy_init
Factory
Ota_0
Ota_1

ELF

Analyze with IDA

tenable

Step 2: Understand Application Image Format

ESP32 Flash Dump

FF FF FF FF FF FF FF FF FF FF FF FF	
Stage 2 Bootloader	
Partition Table	
NVS	
otadata	
phy_init	
Factory	?? ?? ?? ?? ?? ?? ?? ??
Ota_0	?? ?? ?? ?? ?? ?? ?? ??
Ota_1	?? ?? ?? ?? ?? ?? ?? ??

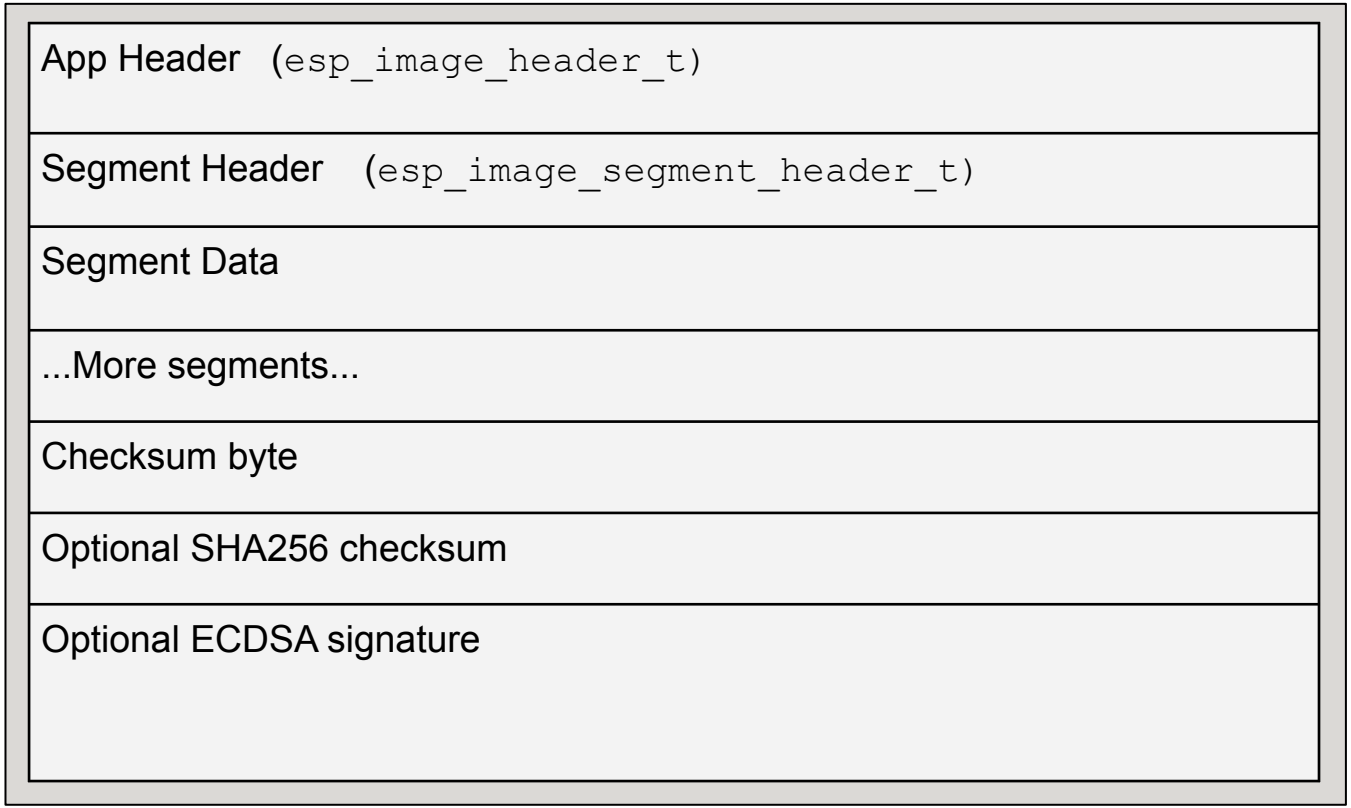
ELF

Analyze with IDA

tenable

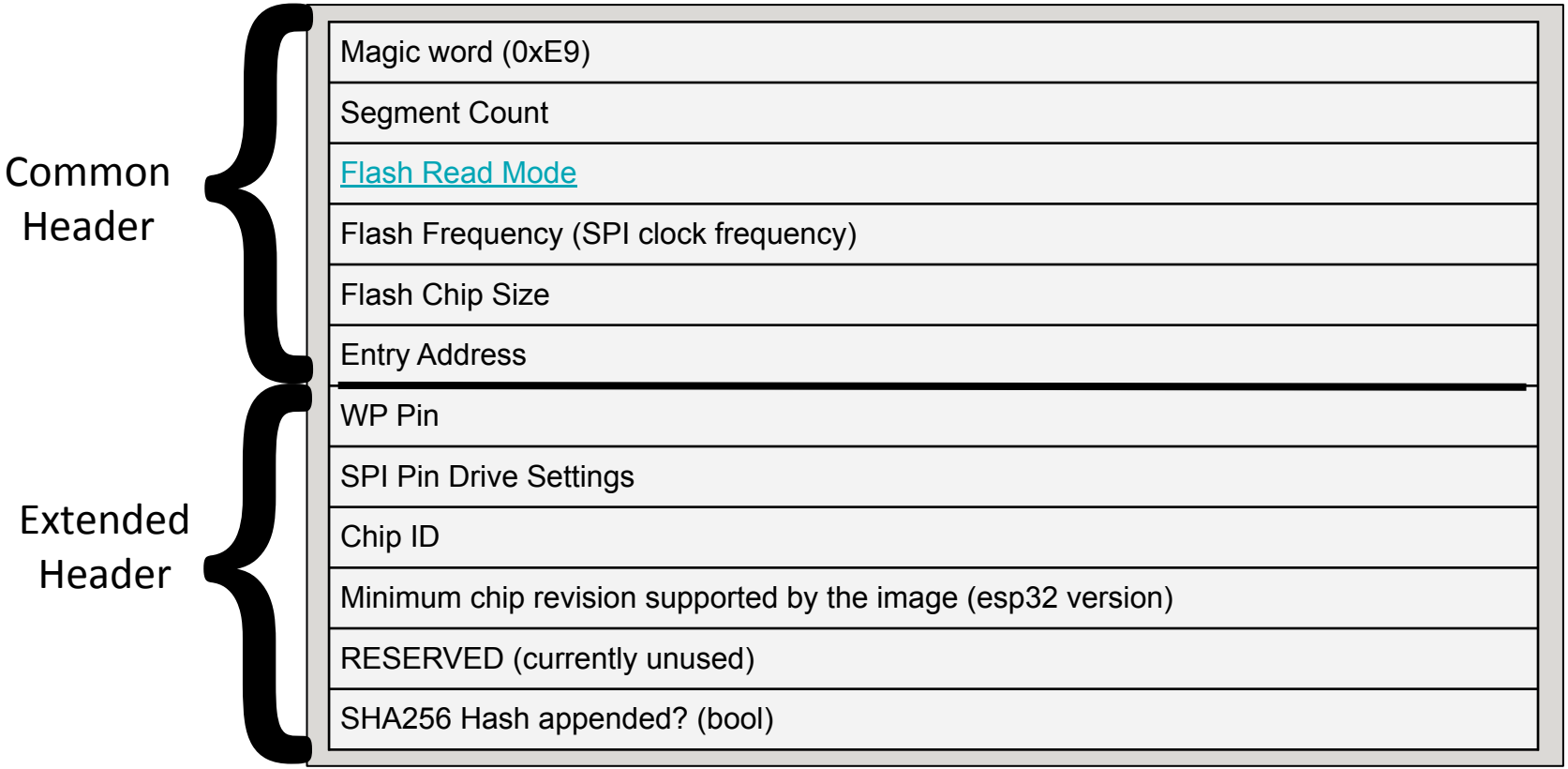
ESP32 Application Image Format

App Image Partitions



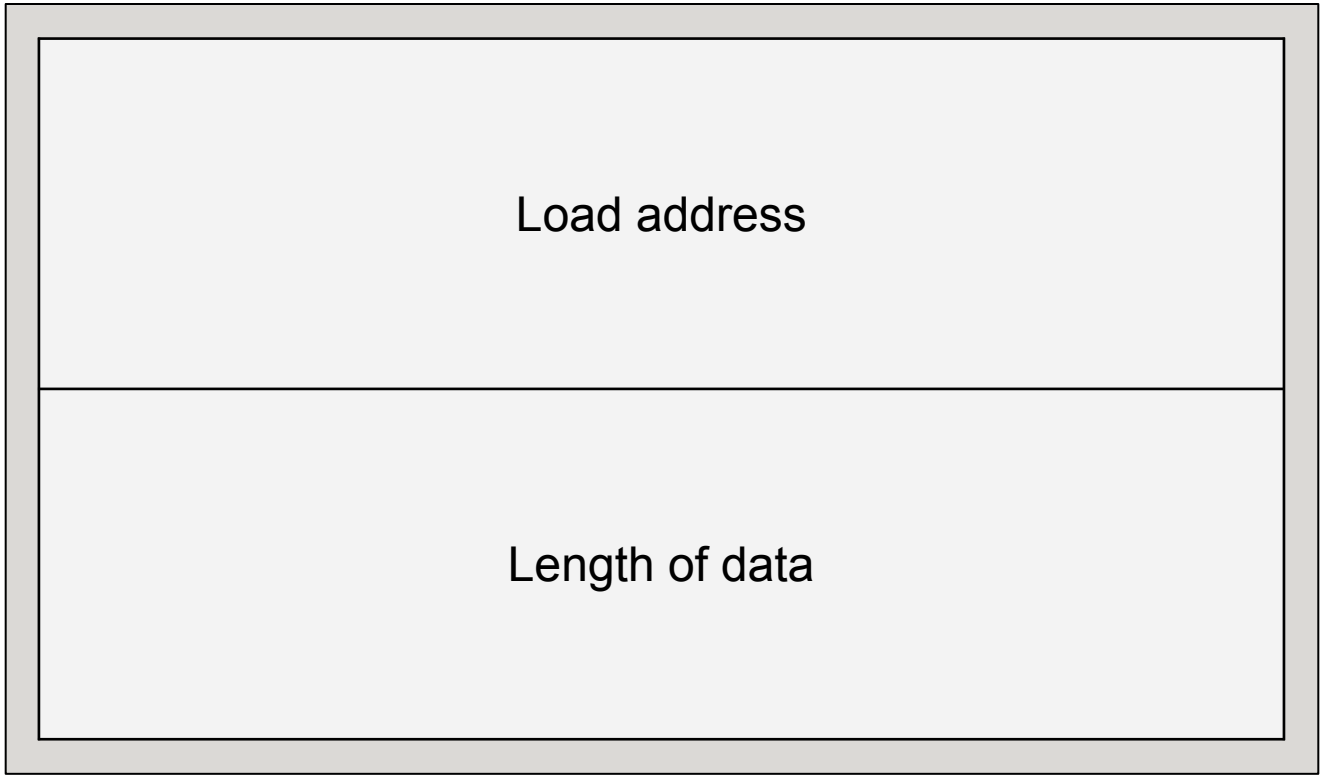
Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/system/app_image_format.html

App Image Header (`esp_image_header_t`)



Source: https://github.com/espressif/esp-idf/blob/826b9f638/components/bootloader_support/include/esp_app_format.h

Segment Header (`esp_image_segment_header_t`)



Source: https://github.com/espressif/esp-idf/blob/826b9f638/components/bootloader_support/include/esp_app_format.h

Segment Types - App

- App code can be executed from several memory regions.
  - **IRAM** - instruction RAM (in Internal SRAM0)
    - First 64 kb block is used for PRO and APP CPU caches
    - The rest is used to store parts of the application which need to run from RAM
    - A few components of ESP-IDF and parts of WiFi stack are placed into this region using the linker script.
  - **IROM** - code executed from Flash
    - By default, this is where code is placed
    - Flash MMU is utilized to execute this code
  - **RTC FAST**
    - Code that has to run after waking up from deep sleep is placed into RTC memory (e.g. wake up from low power mode)

Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/general-notes.html%23application-memory-layout

Segment Types - Data

- There are memory regions designated for data storage
  - **DRAM** - data RAM
    - Generally stores non-constant data and zero-initialized data
  - **DROM** - data stored in Flash
    - By default, constant data is stored here
    - Also has a specific purpose of describing the application, as seen in the next slide.
  - **RTC slow memory**
    - Global and static variables used by code which runs from RTC memory

Source: https://docs.espressif.com/projects/esp-idf/en/latest/api-guides/general-notes.html%23application-memory-layout

DROM Segment: Application Description

Magic Word (0xABCD5432)
Secure Version (for anti-rollback)
RESERVED
Application Version
Project Name
Compile Time
Compile Date
IDF Version
SHA256 of ELF File
RESERVED

Source: https://github.com/espressif/esp-idf/blob/826b9f638/components/bootloader_support/include/esp_app_format.h

Step 2: Understand Application Image Format

ESP32 Firmware (flash)

FF FF FF FF FF FF FF FF FF FF FF FF
Stage 2 Bootloader
Partition Table
NVS
otadata
phy_init

App Header	Segment Header	Segment data	More segments	Checksum	Optional sha256	Optional ECDSA sig

App Header	Segment Header	Segment data	More segments	Checksum	Optional sha256	Optional ECDSA sig

tenable

Step 2: Understand Application Image Format

ESP32 Firmware (flash)

FF FF FF FF FF FF FF FF FF FF FF FF	
Stage 2 Bootloader	
Partition Table	
NVS	
otadata	
phy_init	
Factory	App Header... Segments...
Ota_0	App Header... Segments...
Ota_1	App Header... Segments...

Zooming out...

ELF

Analyze with IDA

tenable

Step 3: Understand the Firmware Build Process

App Source Code

Hello world

Build Process

ESP32 Firmware

Stage 2 Bootloader

Partition Table

.....

Apps ...

Our tooling

ELF

Analyze with IDA Pro

tenable

How the firmware is built

ESP-IDF: Hello World

```c
void app_main(void)
{
    printf("Hello world!\n");

    /* Print chip information */
    esp_chip_info_t chip_info;
    esp_chip_info(&chip_info);
    printf("This is %s chip with %d CPU cores, WiFi%s%s, ",
            CHIP_NAME,
            chip_info.cores,
            (chip_info.features & CHIP_FEATURE_BT) ? "/BT" : "",
            (chip_info.features & CHIP_FEATURE_BLE) ? "/BLE" : "");

    printf("silicon revision %d, ", chip_info.revision);

    printf("%dMB %s flash\n", spi_flash_get_chip_size() / (1024 * 1024),
            (chip_info.features & CHIP_FEATURE_EMB_FLASH) ? "embedded" : "external");

    for (int i = 10; i >= 0; i--) {
        printf("Restarting in %d seconds...\n", i);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    printf("Restarting now.\n");
    fflush(stdout);
    esp_restart();
}
```


Source: https://github.com/espressif/esp-idf/blob/dd8db6621b930a9cbfeef14a0d4daba8c8506e74/examples/get-started/hello_world/main/hello_world_main.c

Hello World: idf.py build

- Build directory
  - Hello-world.elf (elf = executable and linkable format)
  - Hello-world.bin (app image)
  - Bootloader directory
  - Partition_table directory

```
osboxes@osboxes:~/esp/hello_world/build$ ls
bootloader          CMakeFiles              config.env              flasher_args.json          hello-world.elf          ldgen_libraries           project_elf_src.c
bootloader-prefix   cmake_install.cmake     esp-idf                 flash_partition_table_args hello-world.map          ldgen_libraries.in        rules.ninja
build.ninja         compile_commands.json   flash_app_args          flash_project_args         kconfigs.in              partition_table
CMakeCache.txt      config                  flash_bootloader_args   hello-world.bin            kconfigs_projbuild.in    project_description.json
```


tenable

Hello World: idf.py build

- Bootloader directory
  - Bootloader.elf
  - bootloader.bin

```
osboxes@osboxes:~/esp/hello_world/build$ ls bootloader
bootloader.bin  bootloader.map  CMakeCache.txt  cmake_install.cmake    config      esp-idf      kconfigs_projbuild.in     project_elf_src.c
bootloader.elf  build.ninja     CMakeFiles      compile_commands.json  config.env  kconfigs.in  project_description.json  rules.ninja
```

Hello World: idf.py build

- partition_table directory
  - partition-table.bin

```
osboxes@osboxes:~/esp/hello_world/build$ ls partition_table/
partition-table.bin
```


tenable

Hello World: idf.py build

Now flash it...

```
python esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset
--chip esp32  write_flash --flash_mode dio --flash_size detect --flash_freq 40m \
0x1000 build/bootloader/bootloader.bin \
0x8000 build/partition_table/partition-table.bin \
0x10000 build/hello-world.bin
```

Where did these .bin files come from...?

Hello World: idf.py build

ELF is converted to bin

```
python esptool.py --chip esp32 elf2image --flash_mode dio --flash_freq 40m
--flash_size 4MB --elf-sha256-offset 0xb0 -o
/home/osboxes/esp/hello_world/build/hello-world.bin hello-world.elf
```

Hello World: idf.py build

And the partition table?

```
python gen_esp32part.py -q --offset 0x8000 --flash-size 4MB
partitions_two_ota.csv build/partition_table/partition-table.bin
```

Factory app, two OTA definitions: partitions_two_ota.csv

```
# Name,   Type, SubType, Offset,   Size, Flags
nvs,      data, nvs,       ,   0x4000,
otadata,  data, ota,       ,   0x2000,
phy_init, data, phy,       ,   0x1000,
factory,  app,  factory,   ,   1M,
ota_0,    app,  ota_0,     ,   1M,
ota_1,    app,  ota_1,     ,   1M,
```

Hello World: idf.py build

Flash back...

```
python esptool.py -p (PORT) -b 460800 --before default_reset --after hard_reset
--chip esp32  write flash --flash mode dio --flash_size detect --flash_freq 40m \
0x1000 build/bootloader/bootloader.bin \
0x8000 build/partition_table/partition-table.bin \
0x10000 build/hello-world.bin
```

Flash Contents

...	Filled with 0xFF...
0x1000	Stage 2 Bootloader
0x8000	Partition Table
0x9000	nvs (storage)
0xd000	otadata (defines app to boot)
0xf000	Phy_init (rf conf)
0x10000	factory
0x110000	ota_0 (app)
0x210000	ota_1 (app)

Hello World: esptool.py write_flash

```
esptool.py v2.9-dev
Serial port /dev/ttyUSB0
Connecting.....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: c4:4f:33:3e:bb:ed
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 26064 bytes to 15380...
Wrote 26064 bytes (15380 compressed) at 0x00001000 in 0.3 seconds (effective 601.4 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 138...
Wrote 3072 bytes (138 compressed) at 0x00008000 in 0.0 seconds (effective 3188.8 kbit/s)...
Hash of data verified.
Compressed 147056 bytes to 77413...
Wrote 147056 bytes (77413 compressed) at 0x00010000 in 1.7 seconds (effective 674.8 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin…
```

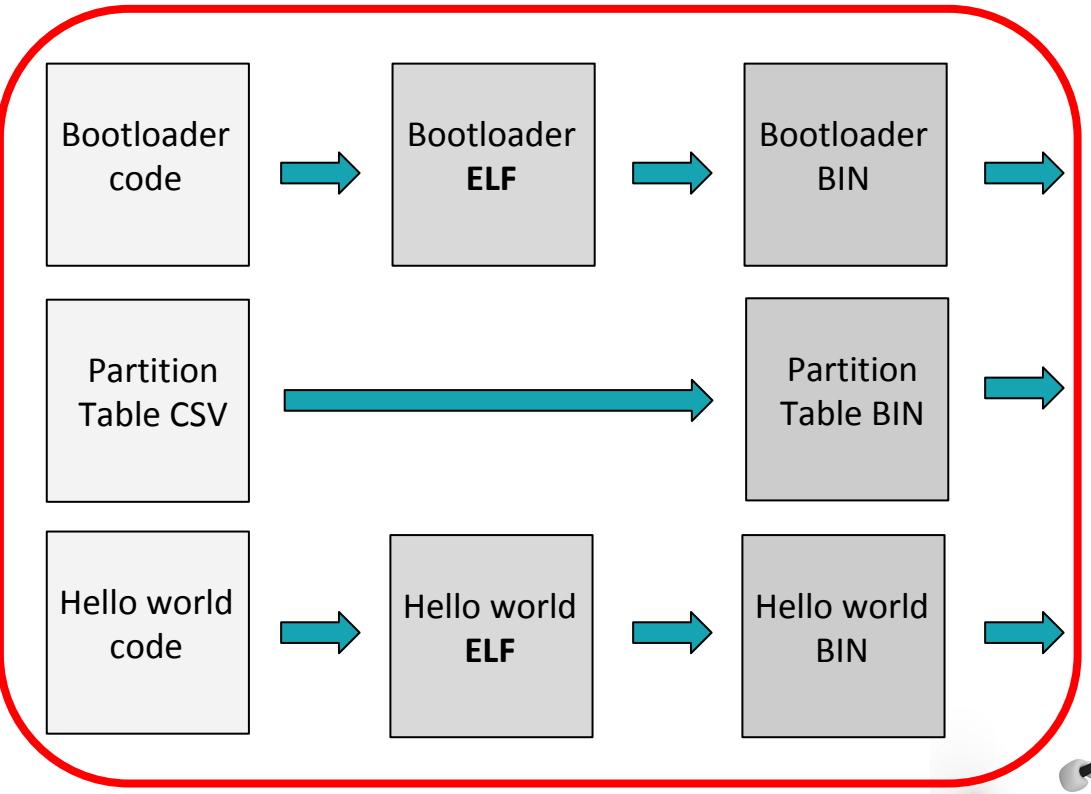

tenable

Hello World: idf.py monitor

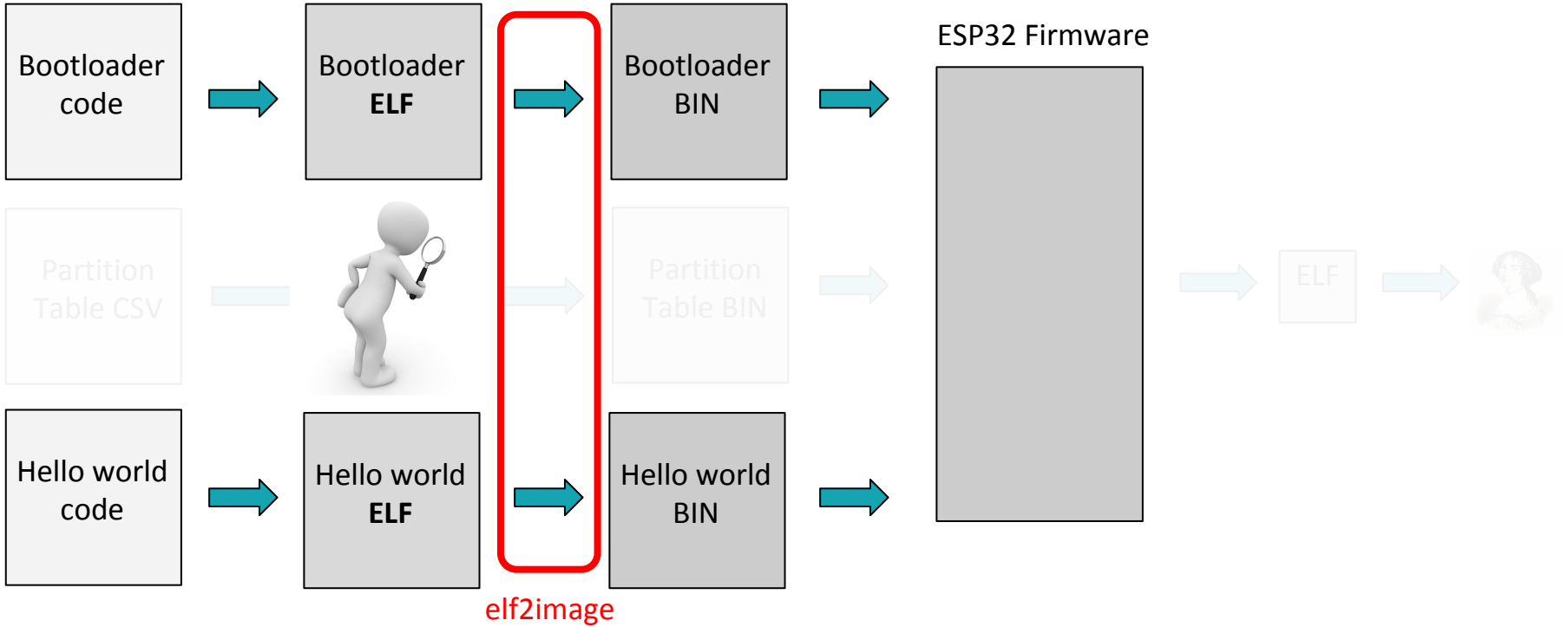
Hello world app output

```
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external
flash
Restarting in 10 seconds...
Restarting in 9 seconds...
Restarting in 8 seconds...
Restarting in 7 seconds...
Restarting in 6 seconds...
Restarting in 5 seconds...
Restarting in 4 seconds...
Restarting in 3 seconds...
Restarting in 2 seconds...
Restarting in 1 seconds...
Restarting in 0 seconds...
Restarting now.
```

Step 3: Understand the Firmware Build Process



Step 4: Understand the ELF to BIN Process



How an ELF is converted to a BIN

esptool.py elf2image

```
def elf2image(args):
    e = ELFFile(args.input)
    if args.chip == 'auto':  # Default to ESP8266 for backwards compatibility
        print("Creating image for ESP8266...")
        args.chip = 'esp8266'

    if args.chip == 'esp32':
        image = ESP32FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = int(args.min_rev)
    elif args.chip == 'esp32s2beta':
        image = ESP32S2FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = 0
    elif args.version == '1':  # ESP8266
        image = ESP8266ROMFirmwareImage()
    else:
        image = ESP8266V2FirmwareImage()
    image.entrypoint = e.entrypoint
    image.segments = e.sections  # ELFSection is a subclass of ImageSegment
    image.flash_mode = {'qio':0, 'qout':1, 'dio':2, 'dout': 3}[args.flash_mode]
    image.flash_size_freq = image.ROM_LOADER.FLASH_SIZES[args.flash_size]
    image.flash_size_freq += {'40m':0, '26m':1, '20m':2, '80m': 0xf}[args.flash_freq]

    if args.elf_sha256_offset:
        image.elf_sha256 = e.sha256()
        image.elf_sha256_offset = args.elf_sha256_offset

    image.verify()

    if args.output is None:
        args.output = image.default_output_name(args.input)
    image.save(args.output)
```

hello-world.elf

dio
4mb
40m

0xb0

hello-world.bin

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: ELFFile()

```
class ELFFile(object):
    SEC_TYPE_PROGBITS = 0x01
    SEC_TYPE_STRTAB = 0x03

    LEN_SEC_HEADER = 0x28

    def __init__(self, name):
        # Load sections from the ELF file
        self.name = name
        with open(self.name, 'rb') as f:
            self._read_elf_file(f)
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: ELFFile()

```
def _read_elf_file(self, f):
    # read the ELF file header
    LEN_FILE_HEADER = 0x34
    try:
        (ident,_type,machine,_version,
         self.entrypoint,_phoff,shoff,_flags,
         _ehsize, _phentsize,_phnum, shentsize,
         shnum,shstrndx) = struct.unpack("<16sHHLLLLLHHHHHH", f.read(LEN_FILE_HEADER))
    except struct.error as e:
        raise FatalError("Failed to read a valid ELF header from %s: %s" % (self.name, e))

    if byte(ident, 0) != 0x7f or ident[1:4] != b'ELF':
        raise FatalError("%s has invalid ELF magic header" % self.name)
    if machine != 0x5e:
        raise FatalError("%s does not appear to be an Xtensa ELF file. e_machine=%04x" % (self.name, machine))
    if shentsize != self.LEN_SEC_HEADER:
        raise FatalError("%s has unexpected section header entry size 0x%x (not 0x28)" % (self.name, shentsize, self.LEN_SEC_HEADER))
    if shnum == 0:
        raise FatalError("%s has 0 section headers" % (self.name))
    self._read_sections(f, shoff, shnum, shstrndx)
```

Validate…

- ELF header structure
- CPU is Xtensa
- Section header entries are proper size
- Has section headers

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: ELFFile()

```
prog_sections = [s for s in all_sections if s[1] == ELFFile.SEC_TYPE_PROGBITS]
```

```
prog_sections = [ELFSection(lookup_string(n_offs), lma, read_data(offs, size)) for (n_offs, _type, lma, size, offs) in prog_sections
                 if lma != 0 and size > 0]
self.sections = prog_sections
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: ELFFile()

- Sections of type SHT_PROGBITS
  - .comment
  - .ctors
  - .data
  - .data1
  - .debug
  - .dtors
  - .fini
  - .got
  - .init
  - .interp
  - .line
  - .note.GNU-stack
  - .plt
  - .rodata
  - .rodata1
  - .text
- What about…?
  - .bss
  - .shstrtab
  - .strtab
  - **.symtab**

Source: http://man7.org/linux/man-pages/man5/elf.5.html

esptool.py elf2image: ELFFile()

- ELF header structure
- CPU is Xtensa
- Section header entry size of 0x28
- Has section headers

Meeting this criteria:

- Type is PROGBITS
- Address != 0
- Size > 0

Source: http://man7.org/linux/man-pages/man5/elf.5.html

esptool.py elf2image

```
def elf2image(args):
    e = ELFFile(args.input)
    if args.chip == 'auto':  # Default to ESP8266 for backwards compatibility
        print("Creating image for ESP8266...")
        args.chip = 'esp8266'

    if args.chip == 'esp32':
        image = ESP32FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = int(args.min_rev)
    elif args.chip == 'esp32s2beta':
        image = ESP32S2FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = 0
    elif args.version == '1':  # ESP8266
        image = ESP8266ROMFirmwareImage()
    else:
        image = ESP8266V2FirmwareImage()
    image.entrypoint = e.entrypoint
    image.segments = e.sections  # ELFSection is a subclass of ImageSegment
    image.flash_mode = {'qio':0, 'qout':1, 'dio':2, 'dout': 3}[args.flash_mode]
    image.flash_size_freq = image.ROM_LOADER.FLASH_SIZES[args.flash_size]
    image.flash_size_freq += {'40m':0, '26m':1, '20m':2, '80m': 0xf}[args.flash_freq]

    if args.elf_sha256_offset:
        image.elf_sha256 = e.sha256()
        image.elf_sha256_offset = args.elf_sha256_offset

    image.verify()

    if args.output is None:
        args.output = image.default_output_name(args.input)
    image.save(args.output)
```

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: ESP32FirmwareImage()

```
def __init__(self, load_file=None):
    super(ESP32FirmwareImage, self).__init__()
    self.secure_pad = False
    self.flash_mode = 0
    self.flash_size_freq = 0
    self.version = 1
    self.wp_pin = self.WP_PIN_DISABLED
    # SPI pin drive levels
    self.clk_drv = 0
    self.q_drv = 0
    self.d_drv = 0
    self.cs_drv = 0
    self.hd_drv = 0
    self.wp_drv = 0
    self.min_rev = 0

    self.append_digest = True
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

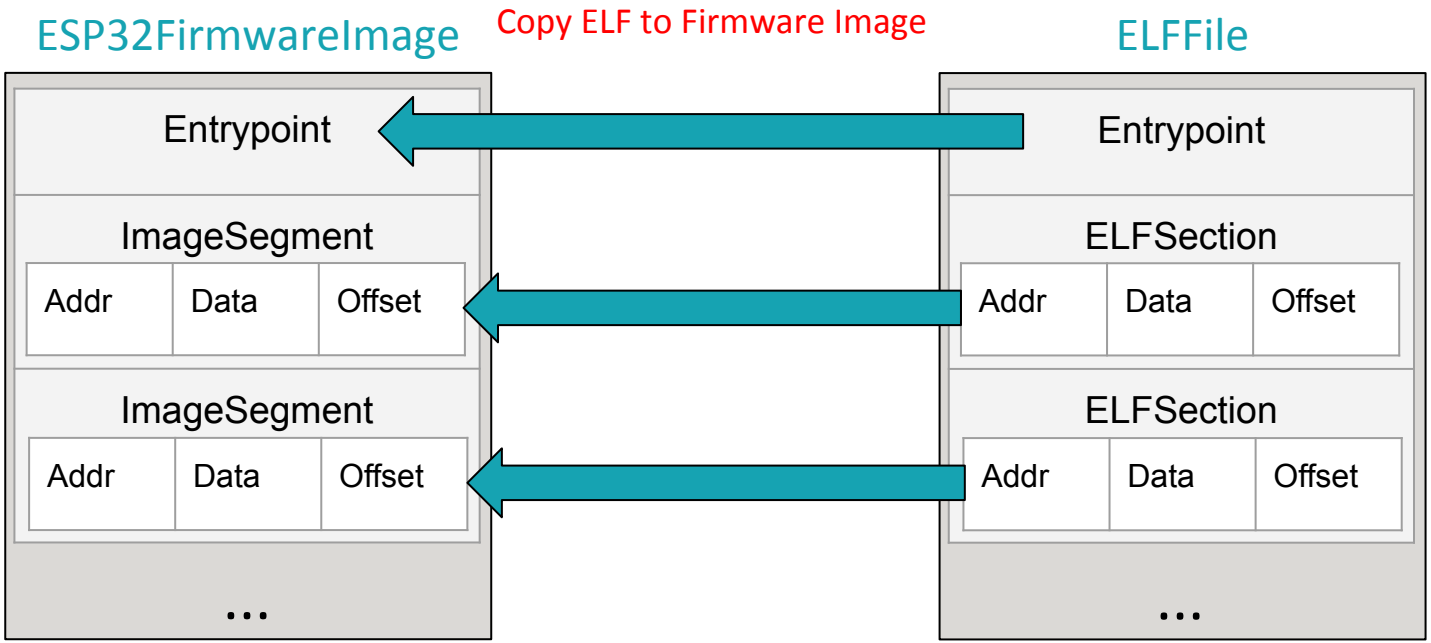
esptool.py elf2image

```
image.entrypoint = e.entrypoint
image.segments = e.sections  # ELFSection is a subclass of ImageSegment
image.flash_mode = {'qio':0, 'qout':1, 'dio':2, 'dout': 3}[args.flash_mode]
image.flash_size_freq = image.ROM_LOADER.FLASH_SIZES[args.flash_size]
image.flash_size_freq += {'40m':0, '26m':1, '20m':2, '80m': 0xf}[args.flash_freq]
```

```
class ImageSegment(object):
    """ Wrapper class for a segment in an ESP image
    (very similar to a section in an ELFImage also) """
    def __init__(self, addr, data, file_offs=None):
        self.addr = addr
        self.data = data
        self.file_offs = file_offs
        self.include_in_checksum = True
        if self.addr != 0:
            self.pad_to_alignment(4)  # pad all "real" I
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image



Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image

```
def elf2image(args):
    e = ELFFile(args.input)
    if args.chip == 'auto':  # Default to ESP8266 for backwards compatibility
        print("Creating image for ESP8266...")
        args.chip = 'esp8266'

    if args.chip == 'esp32':
        image = ESP32FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = int(args.min_rev)
    elif args.chip == 'esp32s2beta':
        image = ESP32S2FirmwareImage()
        image.secure_pad = args.secure_pad
        image.min_rev = 0
    elif args.version == '1':  # ESP8266
        image = ESP8266ROMFirmwareImage()
    else:
        image = ESP8266V2FirmwareImage()
    image.entrypoint = e.entrypoint
    image.segments = e.sections  # ELFSection is a subclass of ImageSegment
    image.flash_mode = {'qio':0, 'qout':1, 'dio':2, 'dout': 3}[args.flash_mode]
    image.flash_size_freq = image.ROM_LOADER.FLASH_SIZES[args.flash_size]
    image.flash_size_freq += {'40m':0, '26m':1, '20m':2, '80m': 0xf}[args.flash_freq]

    if args.elf_sha256_offset:
        image.elf_sha256 = e.sha256()
        image.elf_sha256_offset = args.elf_sha256_offset

    image.verify()

    if args.output is None:
        args.output = image.default_output_name(args.input)
    image.save(args.output)
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

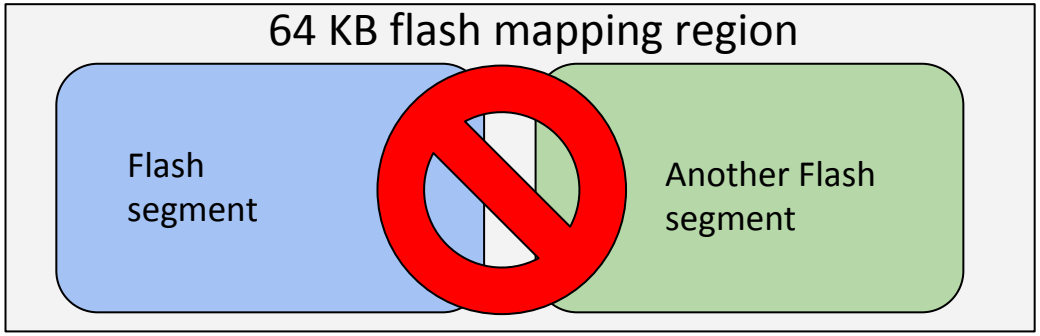
esptool.py elf2image: image.save()

```
def save(self, filename):
    total_segments = 0
    with io.BytesIO() as f:  # write file to memory first
        self.write_common_header(f, self.segments)

        # first 4 bytes of header are read by ROM bootloader for SPI
        # config, but currently unused
        self.save_extended_header(f)

        checksum = ESPLoader.ESP_CHECKSUM_MAGIC

        # split segments into flash-mapped vs ram-loaded, and take copies so we can mutate them
        flash_segments = [copy.deepcopy(s) for s in sorted(self.segments, key=lambda s:s.addr) if self.is_flash_addr(s.addr)]
        ram_segments = [copy.deepcopy(s) for s in sorted(self.segments, key=lambda s:s.addr) if not self.is_flash_addr(s.addr)]
```

flash address range = 0x400D0000 — 0x40400000

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()




```
# check for multiple ELF sections that are mapped in the same flash mapping region.
# this is usually a sign of a broken linker script, but if you have a legitimate
# use case then let us know (we can merge segments here, but as a rule you probably
# want to merge them in your linker script.)
if len(flash_segments) > 0:
    last_addr = flash_segments[0].addr
    for segment in flash_segments[1:]:
        if segment.addr // self.IROM_ALIGN == last_addr // self.IROM_ALIGN:
            raise FatalError(("Segment loaded at 0x%08x lands in same 64KB flash mapping as segment loaded at 0x%08x. " +
                              "Can't generate binary. Suggest changing linker script or ELF to merge sections.") %
                             (segment.addr, last_addr))
        last_addr = segment.addr
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()

```
# try to fit each flash segment on a 64kB aligned boundary
# by padding with parts of the non-flash segments...
while len(flash_segments) > 0:
    segment = flash_segments[0]
    pad_len = get_alignment_data_needed(segment)
    if pad_len > 0:  # need to pad
        if len(ram_segments) > 0 and pad_len > self.SEG_HEADER_LEN:
            pad_segment = ram_segments[0].split_image(pad_len)
            if len(ram_segments[0].data) == 0:
                ram_segments.pop(0)
        else:
            pad_segment = ImageSegment(0, b'\x00' * pad_len, f.tell())
        checksum = self.save_segment(f, pad_segment, checksum)
        total_segments += 1
    else:
        # write the flash segment
        assert (f.tell() + 8) % self.IROM_ALIGN == segment.addr % self.IROM_ALIGN
        checksum = self.save_flash_segment(f, segment, checksum)
        flash_segments.pop(0)
        total_segments += 1
```

Pad if necessary

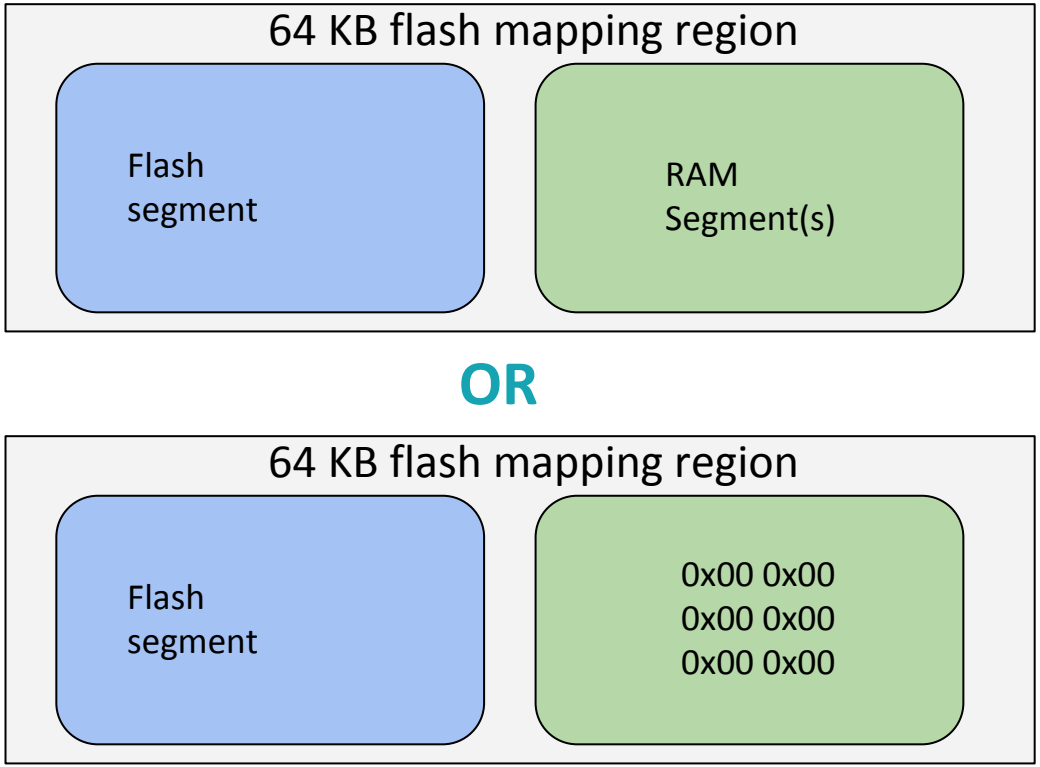
Either with RAM segments

or null bytes

If no padding necessary, just write the flash segment

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()



Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()

```
# flash segments all written, so write any remaining RAM segments
for segment in ram_segments:
    checksum = self.save_segment(f, segment, checksum)
    total_segments += 1
```


Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()

```
if self.secure_pad:
    # pad the image so that after signing it will end on a a 64KB boundary.
    # This ensures all mapped flash content will be verified.
    if not self.append_digest:
        raise FatalError("secure_pad only applies if a SHA-256 digest is also appended to the image")
    align_past = (f.tell() + self.SEG_HEADER_LEN) % self.IROM_ALIGN
    # 16 byte aligned checksum (force the alignment to simplify calculations)
    checksum_space = 16
    # after checksum: SHA-256 digest + (to be added by signing process) version, signature + 12 trailing bytes due to alignment
    space_after_checksum = 32 + 4 + 64 + 12
    pad_len = (self.IROM_ALIGN - align_past - checksum_space - space_after_checksum) % self.IROM_ALIGN
    pad_segment = ImageSegment(0, b'\x00' * pad_len, f.tell())

    checksum = self.save_segment(f, pad_segment, checksum)
    total_segments += 1

# done writing segments
self.append_checksum(f, checksum)
```

Pad with null bytes

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()

```
# kinda hacky: go back to the initial header and write the new segment count
# that includes padding segments. This header is not checksummed
f.seek(1)
try:
    f.write(chr(total_segments))
except TypeError:  # Python 3
    f.write(bytes([total_segments]))

if self.append_digest:
    # calculate the SHA256 of the whole file and append it
    f.seek(0)
    digest = hashlib.sha256()
    digest.update(f.read(image_length))
    f.write(digest.digest())

with open(filename, 'wb') as real_file:
    real_file.write(f.getvalue())
```

Update segment count to include padding segments

Source: https://github.com/espressif/esptool/blob/master/esptool.py

esptool.py elf2image: image.save()

App Image Header

- Common header
- Extended header

Memory Segments

1. Align flash (ROM) segments on 64 KB boundaries
   a. If misaligned, pad flash segments with RAM segments (or 0x00)
   b. Note: RAM segments may be split
2. Write remaining RAM segments
3. If secure_pad, align with 0x00

Misc

- Checksum
- Update segment count
- If SHA256, add that

Source: https://github.com/espressif/esptool/blob/master/esptool.py

Readelf

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .rtc.text         PROGBITS        400c0000 026bcf 000000 00   W  0   0  1
  [ 2] .rtc.dummy        PROGBITS        3ff80000 026bcf 000000 00   W  0   0  1
  [ 3] .rtc.force_fast   PROGBITS        3ff80000 026bcf 000000 00   W  0   0  1
  [ 4] .rtc_noinit       PROGBITS        50000000 026bcf 000000 00   W  0   0  1
  [ 5] .rtc.force_slow   PROGBITS        50000000 026bcf 000000 00   W  0   0  1
  [ 6] .iram0.vectors    PROGBITS        40080000 00a000 000403 00  AX  0   0  4
  [ 7] .iram0.text       PROGBITS        40080404 00a404 009730 00  AX  0   0  4
  [ 8] .dram0.data       PROGBITS        3ffb0000 007000 002160 00  WA  0   0 16
  [ 9] .noinit           PROGBITS        3ffb2160 026bcf 000000 00   W  0   0  1
  [10] .dram0.bss        NOBITS          3ffb2160 009160 000800 00  WA  0   0  8
  [11] .flash.rodata     PROGBITS        3f400020 001020 0055bc 00  WA  0   0 16
  [12] .flash.text       PROGBITS        400d0020 014020 012baf 00  AX  0   0  4
  [13] .iram0.text_end   PROGBITS        40089b34 026bcf 000000 00   W  0   0  1
  [14] .dram0.heap_start PROGBITS        3ffb2960 026bcf 000000 00   W  0   0  1
  [15] .xtensa.info      NOTE            00000000 026bcf 000038 00      0   0  1
  [16] .comment          PROGBITS        00000000 026c07 000065 01  MS  0   0  1
  [17] .debug_frame      PROGBITS        00000000 026c6c 0070d0 00      0   0  4
  [18] .debug_info       PROGBITS        00000000 02dd3c 1302d7 00      0   0  1
  [19] .debug_abbrev     PROGBITS        00000000 15e013 017ba1 00      0   0  1
  [20] .debug_loc        PROGBITS        00000000 175bb4 039ba1 00      0   0  1
  [21] .debug_aranges    PROGBITS        00000000 1af758 002e00 00      0   0  8
  [22] .debug_ranges     PROGBITS        00000000 1b2558 003300 00      0   0  8
  [23] .debug_line       PROGBITS        00000000 1b5858 085fa7 00      0   0  1
  [24] .debug_str        PROGBITS        00000000 23b7ff 0183f2 01  MS  0   0  1
  [25] .symtab           SYMTAB          00000000 253bf4 007830 10     26 1020  4
  [26] .strtab           STRTAB          00000000 25b424 006089 00      0   0  1
  [27] .shstrtab         STRTAB          00000000 2614ad 00014e 00      0   0  1
```

- Readelf output
  - Highlighted will be written to firmware image
    - PROGBITS
    - Addr != 0x0
    - Size > 0x0

tenable

readelf vs esptool.py elf2image

```
[ 6] .iram0.vectors    PROGBITS        40080000 00a000 000403 00  AX  0   0  4
[ 7] .iram0.text       PROGBITS        40080404 00a404 009730 00  AX  0   0  4
[ 8] .dram0.data       PROGBITS        3ffb0000 007000 002160 00  WA  0   0 16
[ 9] .noinit           PROGBITS        3ffb2160 026bcf 000000 00   W  0   0  1
[10] .dram0.bss        NOBITS          3ffb2160 009160 000800 00  WA  0   0  8
[11] .flash.rodata     PROGBITS        3f400020 001020 0055bc 00  WA  0   0 16
[12] .flash.text       PROGBITS        400d0020 014020 012baf 00  AX  0   0  4
```

```
python3 esptool.py --chip esp32 elf2image --flash_mode dio --flash_freq 40m --flash_size 4MB
--elf-sha256-offset 0xb0 -o hello-world3.bin ~/esp/hello_world/build/hello-world.elf
esptool.py v2.9-dev

Writing flash segment '.flash.rodata' to 0x3f400020
Writing padded segment '.dram0.data' to 0x3ffb0000
Writing padded segment '.iram0.vectors' to 0x40080000
Writing padded segment '.iram0.text' to 0x40080404
Writing flash segment '.flash.text' to 0x400d0020
Writing segment '.iram0.text' to 0x400888c4
```


tenable

elf2image vs image_info

```
python3 esptool.py --chip esp32 elf2image --flash_mode dio --flash_freq 40m --flash_size 4MB
--elf-sha256-offset 0xb0 -o hello-world3.bin ~/esp/hello_world/build/hello-world.elf
esptool.py v2.9-dev

Writing flash segment '.flash.rodata' to 0x3f400020
Writing padded segment '.dram0.data' to 0x3ffb0000
Writing padded segment '.iram0.vectors' to 0x40080000
Writing padded segment '.iram0.text' to 0x40080404
Writing flash segment '.flash.text' to 0x400d0020
Writing segment '.iram0.text' to 0x400888c4
```

```
osboxes@osboxes:~/esptool$ python3 esptool.py --chip esp32 image_info hello-world3.bin
esptool.py v2.9-dev
Image version: 1
Entry point: 40081050
6 segments

Segment 1: len 0x055bc load 0x3f400020 file_offs 0x00000018 [DROM]
Segment 2: len 0x02160 load 0x3ffb0000 file_offs 0x000055dc [BYTE_ACCESSIBLE, DRAM, DMA]
Segment 3: len 0x00404 load 0x40080000 file_offs 0x00007744 [IRAM]
Segment 4: len 0x084c0 load 0x40080404 file_offs 0x00007b50 [IRAM]
Segment 5: len 0x12bb0 load 0x400d0020 file_offs 0x00010018 [IROM]
Segment 6: len 0x01270 load 0x400888c4 file_offs 0x00022bd0 [IRAM]
Checksum: 86 (valid)
Validation Hash: 1e9b8b0087d0f2d3481815e875e2a89a933c34d02612e199c68945bf21d18587 (valid)
```


tenable

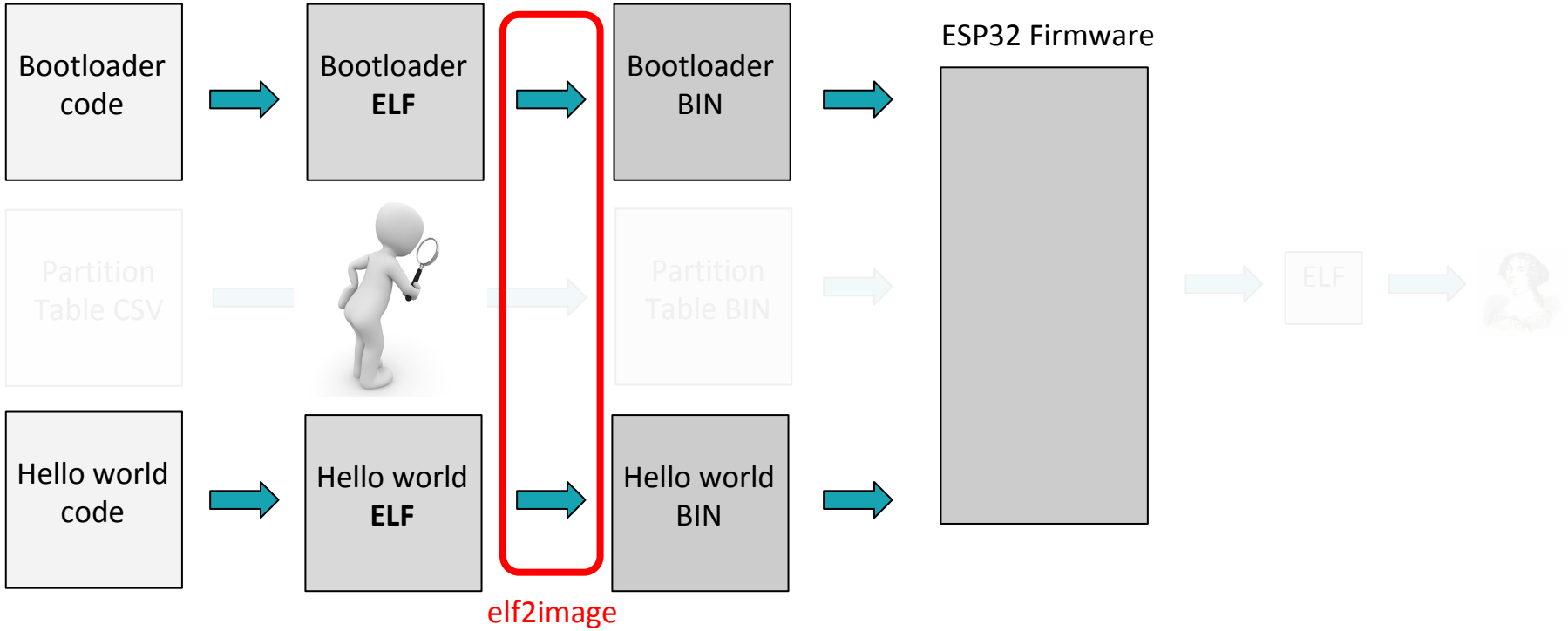
ELF Sections vs App Image Segments

Observed mapping

ELF Section
.flash.rodata
.dram0.data
.iram0.vectors
.iram0.text *
.flash.text
.iram0.text *

App Image Segment
DROM
BYTE_ACCESSIBLE, DRAM, DMA
IRAM
IRAM
IROM
IRAM

Step 4: Understand the ELF to BIN Process



Step 5: Extract that ELF!

ESP32 Flash Dump

Bootloader
Partition Table
Apps

ELF

Our tooling

Analyze with IDA Pro

tenable

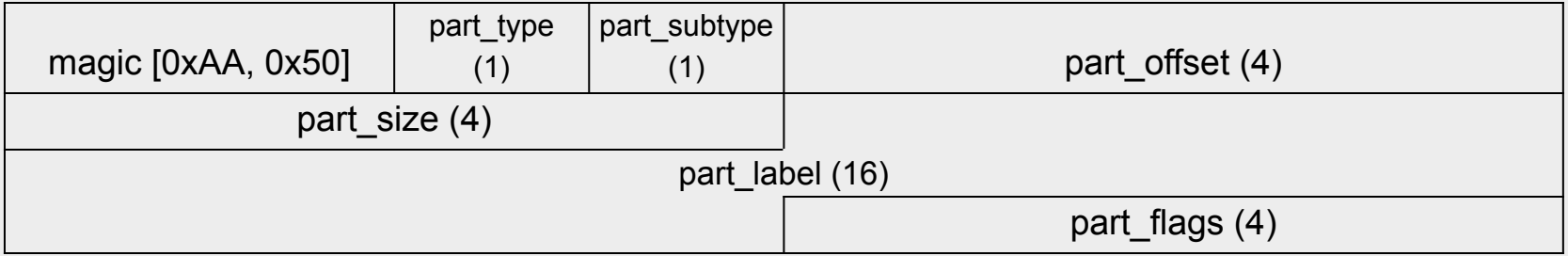
How we converted back to ELF

Our toolkit

- Can:
  - Show all partitions found in the flash dump (show_partitions)
  - Dump a specified partition to disk (dump_partition)
  - Convert a specified app partition back into an ELF (**create_elf**)
  - Dump NVS partition contents

Partition Table Parsing

Table Entry (32 bytes)



- magic
  - 0x50, 0xAA if valid partition table entry exists
- part_type
  - 0x00 if type “APP”
  - 0x01 if type “DATA”
- part_subtype
  - If part_type is “APP”:
    - 0x00 is “FACTORY”
    - 0x20 is “TEST”
  - If part_type is “DATA”
    - 0x00 is “OTA”
    - 0x01 is “RF”
    - 0x02 is “WIFI”
    - 0x04 is “NVS”
- part_offset
  - 32-bit unsigned integer for partition offset
- part_size
  - 32-bit unsigned integer for partition size
- part_label
  - Null terminated string containing label for partition
- part_flags
  - For now, just contains one flag to show if partition is encrypted

tenable

Converting back to ELF: Initialization

```
def image2elf(filename, output_file, verbose=False):
    image = LoadFirmwareImage('esp32', filename)

    # parse image name
    # e.g. 'image.bin' turns to 'image'
    image_name = image_base_name(filename)

    elf = ELF(e_machine=EM.EM_XTENSA, e_data=ELFDATA.ELFDATA2LSB)
    elf.Elf.Ehdr.e_entry = image.entrypoint
```

Load app image segments. Thanks esptool!

initialize ELF little-endian, Xtensa CPU

entrypoint

Used https://github.com/v3l0c1r4pt0r/makeelf

Converting back to ELF: Create Mapping

Observed mapping

ELF Section
.flash.rodata
.dram0.data
.iram0.vectors *
.iram0.text *
.flash.text
.iram0.text *

App Image Segment
DROM
BYTE_ACCESSIBLE, DRAM, DMA
IRAM
IRAM
IROM
IRAM

```
# maps segment names to ELF sections
section_map = {
    'DROM'                          : '.flash.rodata',
    'BYTE_ACCESSIBLE, DRAM, DMA': '.dram0.data',
    'IROM'                          : '.flash.text'
}
# TODO RTC
```


tenable

Converting back to ELF: Define Section Header Attrs

```
[ 6] .iram0.vectors    PROGBITS        40080000 00a000 000403 00  AX  0   0  4
[ 7] .iram0.text       PROGBITS        40080404 00a404 009730 00  AX  0   0  4
[ 8] .dram0.data       PROGBITS        3ffb0000 007000 002160 00  WA  0   0 16
[ 9] .noinit           PROGBITS        3ffb2160 026bcf 000000 00   W  0   0  1
[10] .dram0.bss        NOBITS          3ffb2160 009160 000800 00  WA  0   0  8
[11] .flash.rodata     PROGBITS        3f400020 001020 0055bc 00  WA  0   0 16
[12] .flash.text       PROGBITS        400d0020 014020 012baf 00  AX  0   0  4
```

```
# map to hold pre-defined ELF section header attributes
# http://man7.org/linux/man-pages/man5/elf.5.html
# ES    : sh_entsize
# Flg   : sh_flags
# Lk    : sh_link
# Inf   : sh_info
# Al    : sh_addralign
sect_attr_map = {
        '.flash.rodata' : {'ES':0x00, 'Flg':'WA',  'Lk':0, 'Inf':0, 'Al':16},
        '.dram0.data'   : {'ES':0x00, 'Flg':'WA',  'Lk':0, 'Inf':0, 'Al':16},
        '.iram0.vectors': {'ES':0x00, 'Flg':'AX',  'Lk':0, 'Inf':0, 'Al':4},
        '.iram0.text'   : {'ES':0x00, 'Flg':'AX',  'Lk':0, 'Inf':0, 'Al':4},
        '.flash.text'   : {'ES':0x00, 'Flg':'AX',  'Lk':0, 'Inf':0, 'Al':4}
}
# TODO rtc not accounted for
```


tenable

Converting back to ELF: Construct Section Data

```
section_data = {}

##### build out the section data #####
#####################################
iram_seen = False
for seg in sorted(image.segments, key=lambda s:s.addr):

    # name from image
    segment_name = ", ".join([seg_range[2] for seg_range in image.ROM_LOADER.MEMORY_MAP if seg_range[0] <= seg.addr < seg_range[1]])

    # TODO when processing an image, there was an empty segment name?
    if segment_name == '':
        continue

    section_name = ''
    # handle special case
    # iram is split into .vectors and .text
    # .iram0.vectors seems to be the first one.
    if segment_name == 'IRAM':
        if iram_seen == False:
            section_name = '.iram0.vectors'
        else:
            section_name = '.iram0.text'
        iram_seen = True
    else:
        section_name = section_map[segment_name]

    # if we have a mapped segment <-> section
    # add the elf section
    if section_name != '':
        # might need to append to section (e.g. IRAM is split up due to alignment)
        if section_name in section_data:
                section_data[section_name]['data'] += seg.data
        else:
            section_data[section_name] = {'addr':seg.addr, 'data':seg.data}
```

iram0.vectors and iram0.text separated into multiple IRAM segments

Padding segments might be split

Converting back to ELF: Append ELF Sections

```
##### append the sections #####
################################
for name in section_data.keys():
    data = section_data[name]['data']
    addr = section_data[name]['addr']
    # build the section out as much as possible
    # if we already know the attribute values
    if name in sect_attr_map:
        sect = sect_attr_map[name]
        flg = calcShFlg(sect['Flg'])
        elf._append_section(name, data, addr,SHT.SHT_PROGBITS, flg, sect['Lk'], sect['Inf'], sect['Al'], sect['ES'])
    else:
        elf.append_section(name, data, addr)
```

PROGBITS

Section header attributes

tenable

Converting back to ELF: String and Symbol Tables

```
elf.append_special_section('.strtab')
elf.append_special_section('.symtab')
add_elf_symbols(elf)
```

Converting back to ELF: Program Headers

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x001020 0x3f400020 0x3f400020 0x055bc 0x055bc RW  0x1000
  LOAD           0x007000 0x3ffb0000 0x3ffb0000 0x02160 0x02960 RW  0x1000
  LOAD           0x00a000 0x40080000 0x40080000 0x09b34 0x09b34 R E 0x1000
  LOAD           0x014020 0x400d0020 0x400d0020 0x12baf 0x12baf R E 0x1000

 Section to Segment mapping:
  Segment Sections...
   00     .flash.rodata
   01     .dram0.data .dram0.bss
   02     .iram0.vectors .iram0.text
   03     .flash.text
```

```
# segment flags
# TODO rtc
segments = {
    '.flash.rodata' : 'rw',
    '.dram0.data'   : 'rw',
    '.iram0.vectors': 'rx',
    '.flash.text'   : 'rx'
}
```

Converting back to ELF: Program Headers

```
# TODO this logic might change as we add support for rtc
size_of_phdrs = len(Elf32_Phdr()) * len(segments) # to pre-calculate program header offsets

##### add the segments ####
############################
print_verbose(verbose, "\nAdding program headers")
for (name, flags) in segments.items():

    if (name == '.iram0.vectors'):
        # combine these
        size = len(section_data['.iram0.vectors']['data']) + len(section_data['.iram0.text']['data'])
    else:
        size = len(section_data[name]['data'])

    p_flags = calcPhFlg(flags)
    addr = section_data[name]['addr']
    align = 0x1000
    p_type = PT.PT_LOAD

    shstrtab_hdr, shstrtab = elf.get_section_by_name(name)
    offset = shstrtab_hdr.sh_offset + size_of_phdrs # account for new offset

    # build program header
    Phdr = Elf32_Phdr(PT.PT_LOAD, p_offset=offset, p_vaddr=addr,
            p_paddr=addr, p_filesz=size, p_memsz=size,
            p_flags=p_flags, p_align=align, little=elf.little)


    print_verbose(verbose, name + ": " + str(Phdr))
    elf.Elf.Phdr_table.append(Phdr)
```


tenable

Converting back to ELF: Write to disk

```
# write out elf file
if output_file is not None:
    out_file = output_file
else:
    out_file = image_name + '.elf'
print("\nWriting ELF to " + out_file + "...")
fd = os.open(out_file, os.O_WRONLY | os.O_CREAT | os.O_TRUNC)
os.write(fd, bytes(elf))
os.close(fd)
```

Step 5: Extract that ELF!

ESP32 Flash Dump

Bootloader
Partition Table
Apps

Our tooling

ELF

Analyze with IDA Pro

tenable

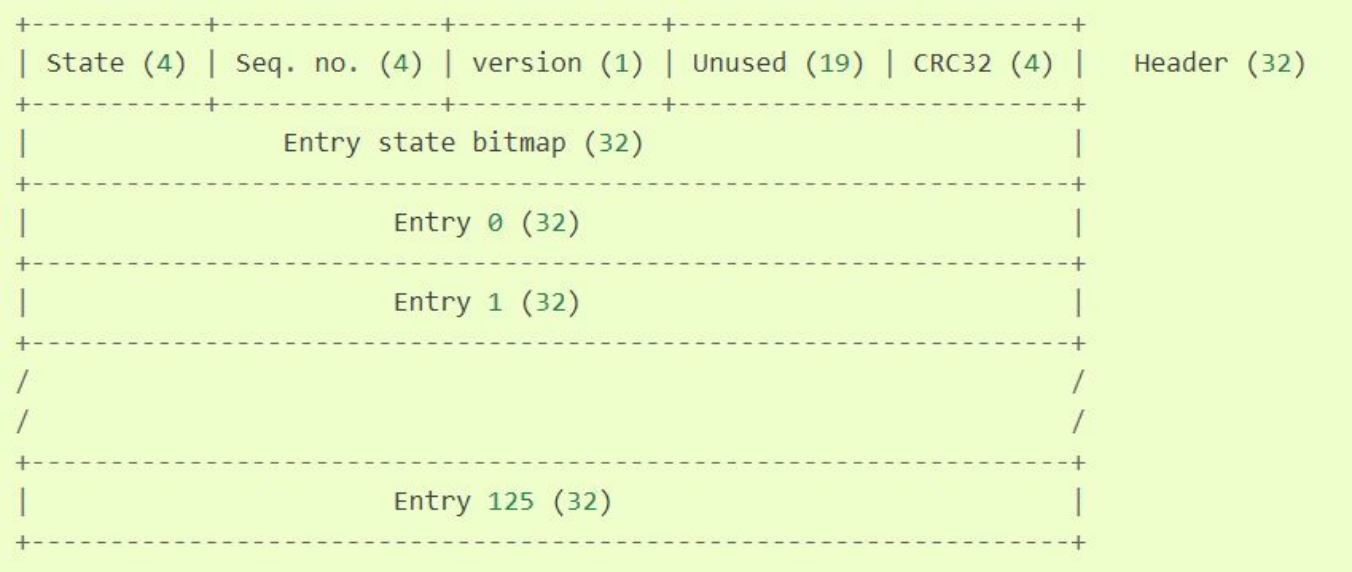
NVS (non-volatile storage)

- Key/value data pairs
- Encryption supported
- We can dump it
- Organized into pages (4096 bytes each) which contain entries (32 bytes each)

If NVS encryption is not used, it is possible for anyone with physical access to the flash chip to alter, erase, or add key-value pairs. With NVS encryption enabled, it is not possible to alter or add a key-value pair and get recognized as a valid pair without knowing corresponding NVS encryption keys. However, there is no tamper-resistance against the erase operation.

https://docs.espressif.com/projects/esp-idf/en/latest/api-reference/storage/nvs_flash.html

NVS Storage - Page




```
page state : FULL
seq no. : 0
version : 1
crc 32 : 3705476802
entry state bitmape (decoded) :
22220020200000000022222000020000000000000000022002002222222022
22200220200330033333333333333333333333333333333333333333333333
```

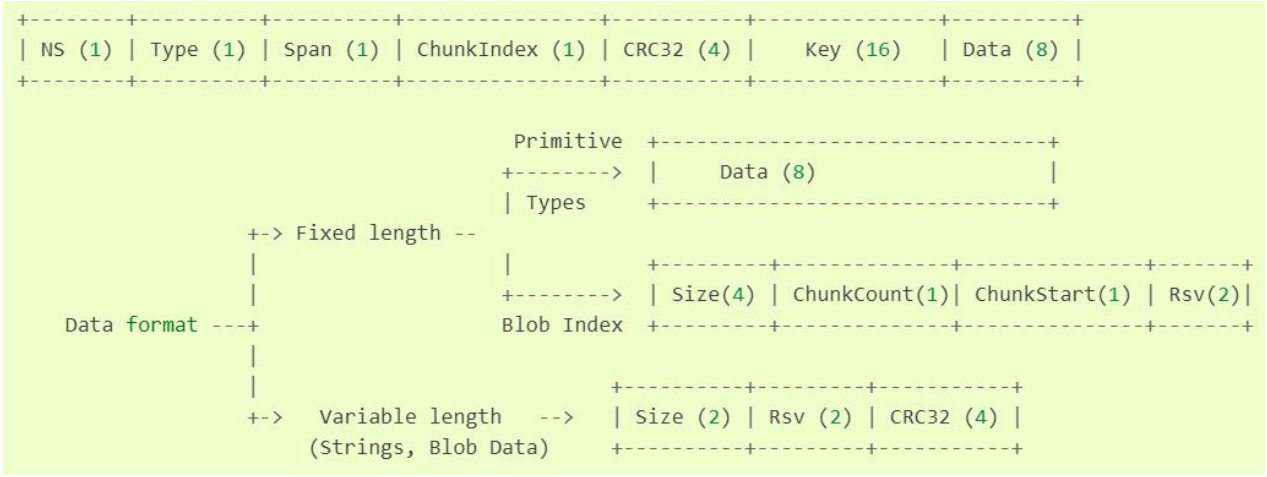
- 4096 bytes total
- Contains 126 pages
- Bitmap contains state of each subsequent entry
- Seq. no. useful for reconstructing blobs spanning multiple pages
- Erased entries aren't wiped, so you can pull forensic data

Bitmap:

- 3 - Empty
- 2 - Written
- 0 - Erased

tenable

NVS Storage - Entry




```
Entry 6
Bitmap State : Written
  Written Entry 6
    NS Index : 2
        NS : nvs.net80211
    Type : BLOB
    Span : 3
    ChunkIndex : 255
    Key : sta.ssid
    Data (Blob) :
      Size : 36
      Data :
00000000: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000010: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF  ................
00000020: FF FF FF FF                                       ....

Entry 9
```

```
Entry 0
Bitmap State : Written
  Written Entry 0
    NS Index : 0
    Type : U8
    Span : 1
    ChunkIndex : 255
    Key : misc
    Data (U8) : 1

Entry 1
Bitmap State : Written
  Written Entry 1
    NS Index : 1
        NS : misc
```

- Data Types
  - uint/int 8, 16, and 32
  - String
  - Blob
  - Blob index (for blobs spanning multiple pages)
- If span > 1, subsequent entries are raw data
- If ns = 0, entry is namespace definition
- For blob index, data field is metadata to allow reconstruction of data spanned across multiple pages.
- For blob and string, size and crc32 are in data field.

tenable

NVS Dump Excerpt

Simplisafe Wi-Fi Creds

```
Entry 5
Bitmap State : Written
  Written Entry 5
    NS Index : 2
        NS : nvs.net80211
    Type : BLOB
    Span : 4
    ChunkIndex : 255
    Key : sta.pswd
    Data (Blob) :
      Size : 65
      Data :
00000000: 6E 65 77 77 69 66 69 70  61 73 73 77 6F 72 64 21  newwifipassword!
00000010: 21 31 32 33 00 00 00 00  00 00 00 00 00 00 00 00  !123............
00000020: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000040: 00                                               .
```

NVS Dump Excerpt (Erased)

Bluetooth Config

```
Entry 0
Bitmap State : Erased
  Written Entry 0
    NS Index : 5
        NS : bt_config.conf
    Type : BLOB
    Span : 41
    ChunkIndex : 255
    Key : bt_cfg_key
    Data (Blob) :
      Size : 1268
      Data :
00000000: 5B 30 30 3A 31 61 3A 37  64 3A 64 61 3A 37 31 3A  [00:1a:7d:da:71:
00000010: 31 33 5D 0A 41 64 64 72  54 79 70 65 20 3D 20 30  13].AddrType = 0
00000020: 0A 44 65 76 54 79 70 65  20 3D 20 32 0A 4C 45 5F  .DevType = 2.LE_
00000030: 4B 45 59 5F 50 45 4E 43  20 3D 20 30 61 30 64 33  KEY_PENC = 0a0d3
00000040: 30 35 61 36 35 61 32 66  63 62 39 64 66 63 62 32  05a65a2fcb9dfcb2
```


tenable

Demo

Demo Time

```
osboxes@osboxes:~/esp32_image_parser$  sudo esptool.py --chip esp32 --port /dev/ttyUSB0 -b 460800 read_flash 0 0x400000 flashdump/esp32_flashdump.bin
[sudo] password for osboxes:
esptool.py v2.9-dev
Serial port /dev/ttyUSB0
Connecting.....
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: c4:4f:33:3e:bb:ed
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
4194304 (100 %)
4194304 (100 %)
Read 4194304 bytes at 0x0 in 102.6 seconds (327.0 kbit/s)...
Hard resetting via RTS pin...
```

Demo Time

```
osboxes@osboxes:~/esp32_image_parser$ python3 esp32_image_parser.py show_partitions flashdump/esp32_flashdump.bin
reading partition table...
entry 0:
  label      : nvs
  offset     : 0x9000
  length     : 20480
  type       : 1 [DATA]
  sub type   : 2 [WIFI]

entry 1:
  label      : otadata
  offset     : 0xe000
  length     : 8192
  type       : 1 [DATA]
  sub type   : 0 [OTA]

entry 2:
  label      : app0
  offset     : 0x10000
  length     : 1310720
  type       : 0 [APP]
  sub type   : 16 [ota_0]

entry 3:
  label      : app1
  offset     : 0x150000
  length     : 1310720
  type       : 0 [APP]
  sub type   : 17 [ota_1]

entry 4:
  label      : spiffs
  offset     : 0x290000
  length     : 1507328
  type       : 1 [DATA]
  sub type   : 130 [unknown]
```

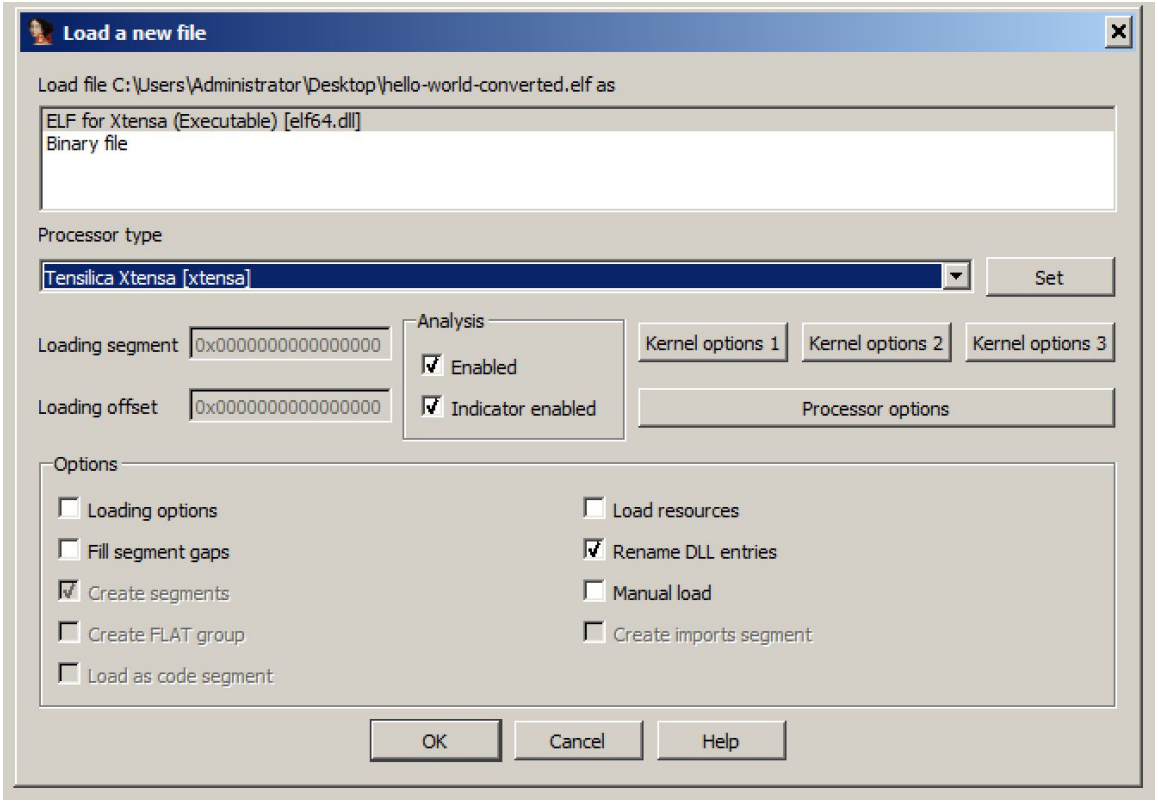

tenable

Demo Time

```
osboxes@osboxes:~/esp32_image_parser$ python3 esp32_image_parser.py create_elf flashdump/esp32_flashdump.bin -partition app0 -output flashdump/app0.elf

Dumping partition 'app0' to app0_out.bin

Writing ELF to flashdump/app0.elf...
```


tenable

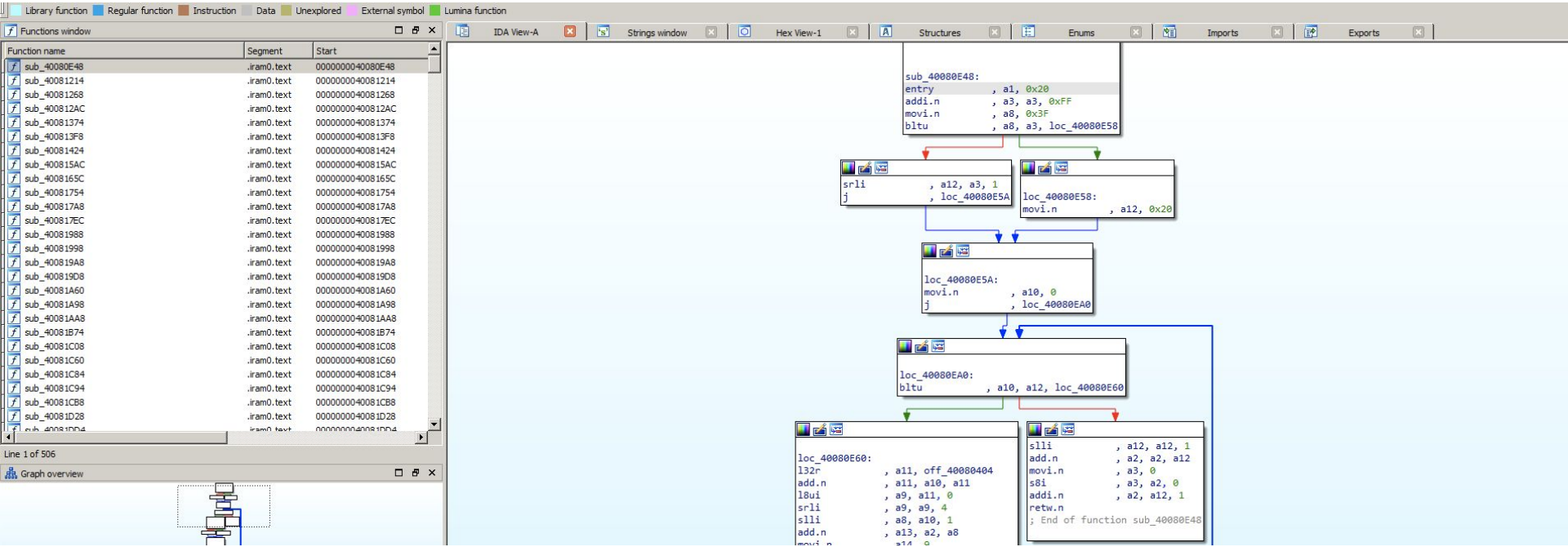
Demo Time



Thanks to https://github.com/themadinventor/ida-xtensa

tenable

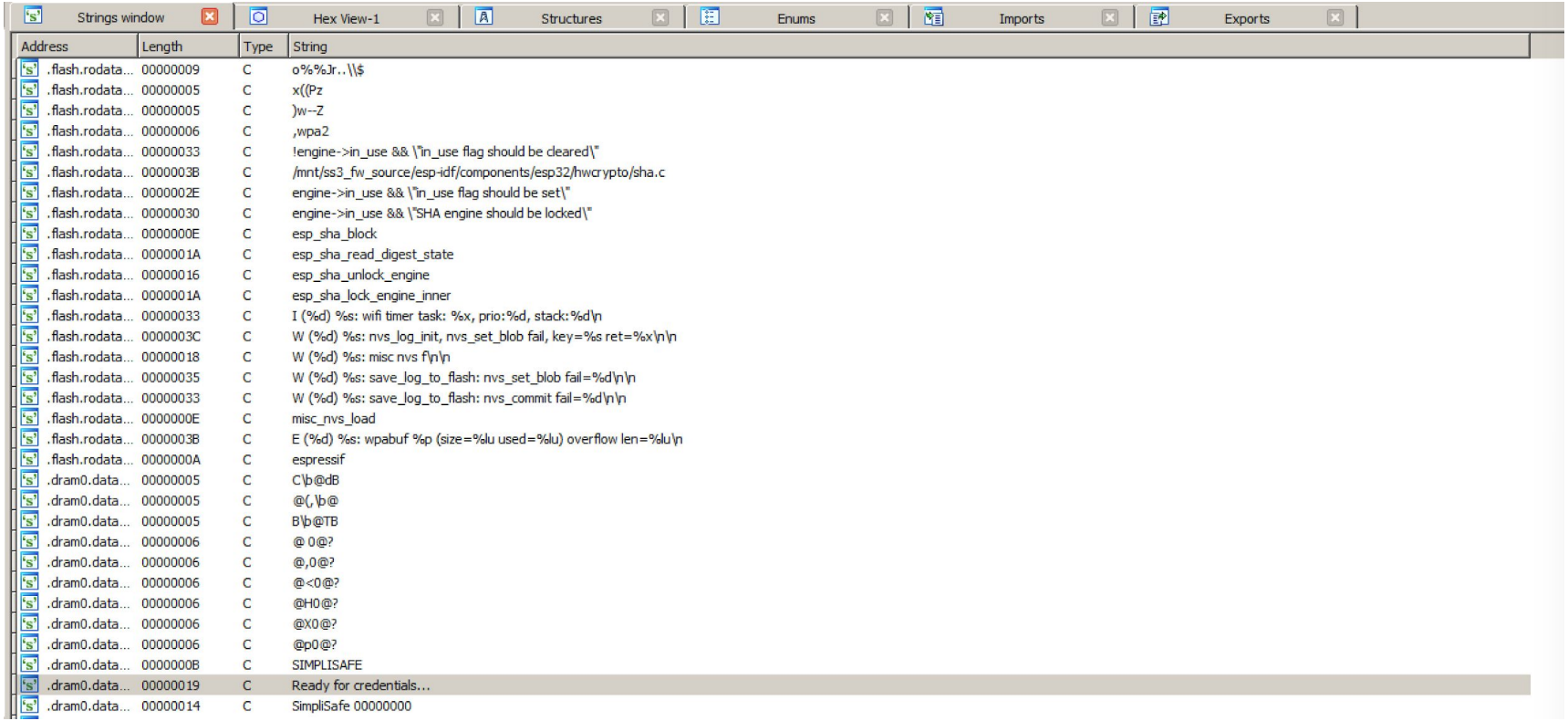
Demo Time



Thanks to https://github.com/themadinventor/ida-xtensa

tenable

Demo Time

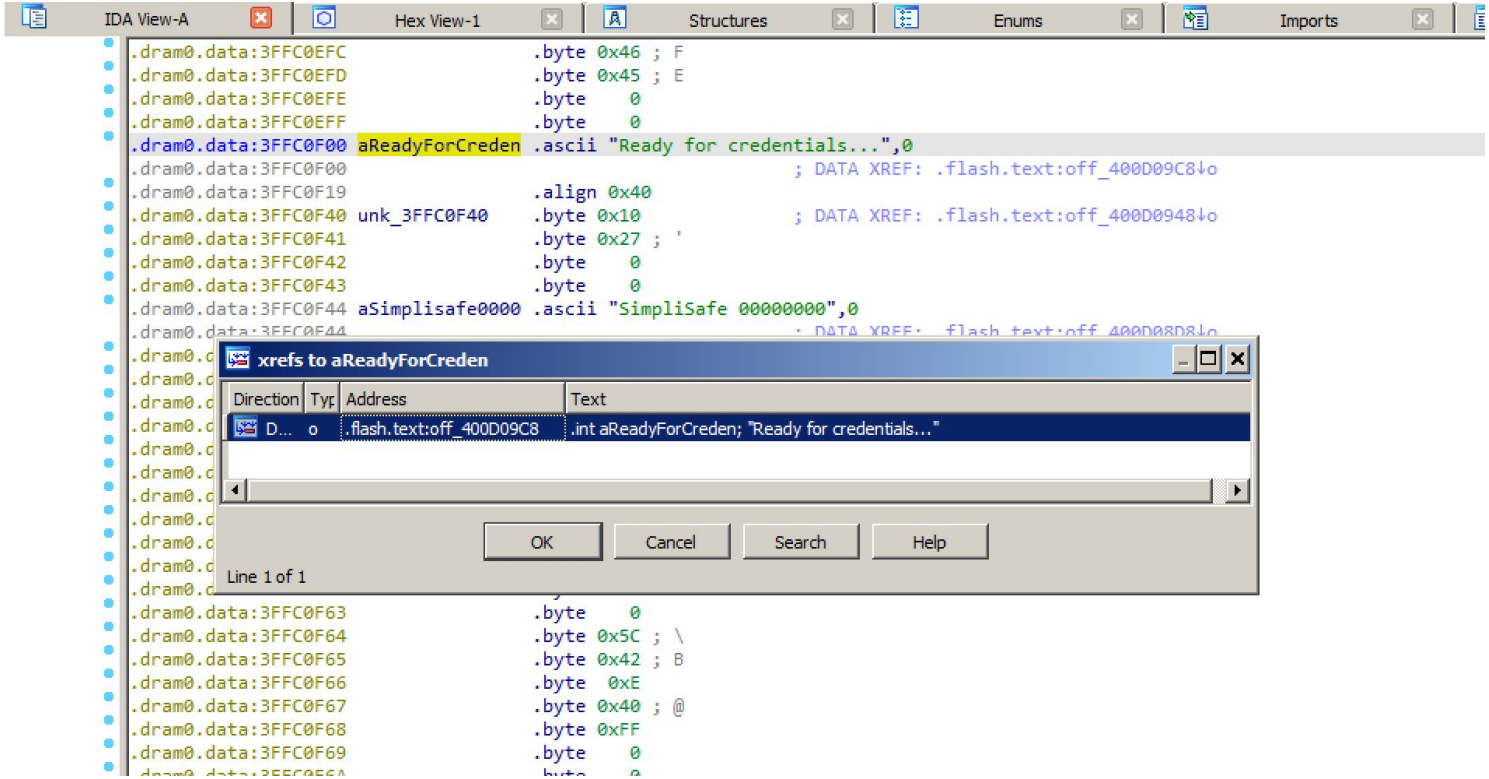


Address	Length	Type	String
.flash.rodata...	00000009	C	o%%Jr..\\$
.flash.rodata...	00000005	C	x((Pz
.flash.rodata...	00000005	C	)w--Z
.flash.rodata...	00000006	C	,wpa2
.flash.rodata...	00000033	C	!engine->in_use && \"in_use flag should be cleared\"
.flash.rodata...	0000003B	C	/mnt/ss3_fw_source/esp-idf/components/esp32/hwcrypto/sha.c
.flash.rodata...	0000002E	C	engine->in_use && \"in_use flag should be set\"
.flash.rodata...	00000030	C	engine->in_use && \"SHA engine should be locked\"
.flash.rodata...	0000000E	C	esp_sha_block
.flash.rodata...	0000001A	C	esp_sha_read_digest_state
.flash.rodata...	00000016	C	esp_sha_unlock_engine
.flash.rodata...	0000001A	C	esp_sha_lock_engine_inner
.flash.rodata...	00000033	C	I (%d) %s: wifi timer task: %x, prio:%d, stack:%d\n
.flash.rodata...	0000003C	C	W (%d) %s: nvs_log_init, nvs_set_blob fail, key=%s ret=%x\n\n
.flash.rodata...	00000018	C	W (%d) %s: misc nvs f\n\n
.flash.rodata...	00000035	C	W (%d) %s: save_log_to_flash: nvs_set_blob fail=%d\n\n
.flash.rodata...	00000033	C	W (%d) %s: save_log_to_flash: nvs_commit fail=%d\n\n
.flash.rodata...	0000000E	C	misc_nvs_load
.flash.rodata...	0000003B	C	E (%d) %s: wpabuf %p (size=%lu used=%lu) overflow len=%lu\n
.flash.rodata...	0000000A	C	espressif
.dram0.data...	00000005	C	C\b@dB
.dram0.data...	00000005	C	@(,\b@
.dram0.data...	00000005	C	B\b@TB
.dram0.data...	00000006	C	@ 0@?
.dram0.data...	00000006	C	@,0@?
.dram0.data...	00000006	C	@<0@?
.dram0.data...	00000006	C	@H0@?
.dram0.data...	00000006	C	@X0@?
.dram0.data...	00000006	C	@p0@?
.dram0.data...	0000000B	C	SIMPLISAFE
.dram0.data...	00000019	C	Ready for credentials...
.dram0.data...	00000014	C	SimpliSafe 00000000

Thanks to https://github.com/themadinventor/ida-xtensa

tenable

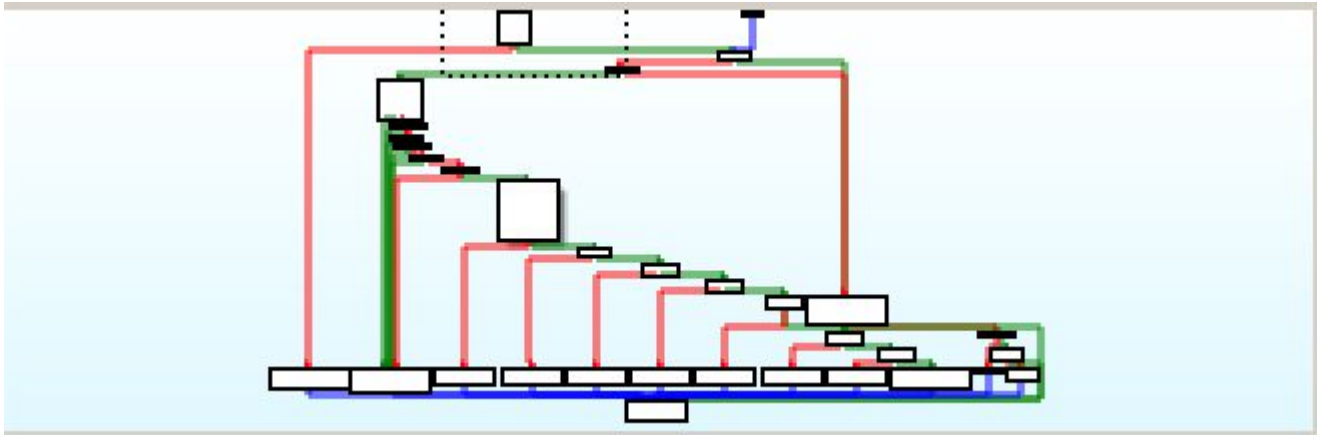
Demo Time



Thanks to https://github.com/themadinventor/ida-xtensa

tenable

Demo Time

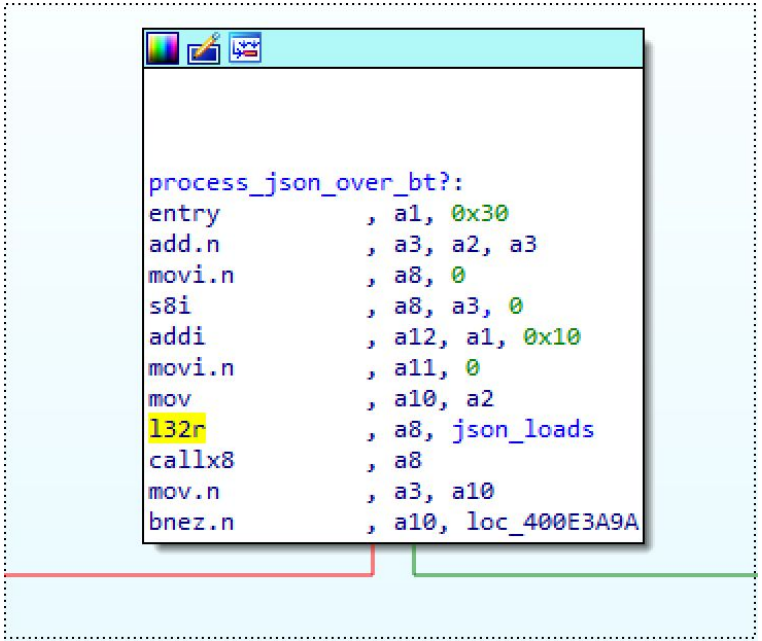


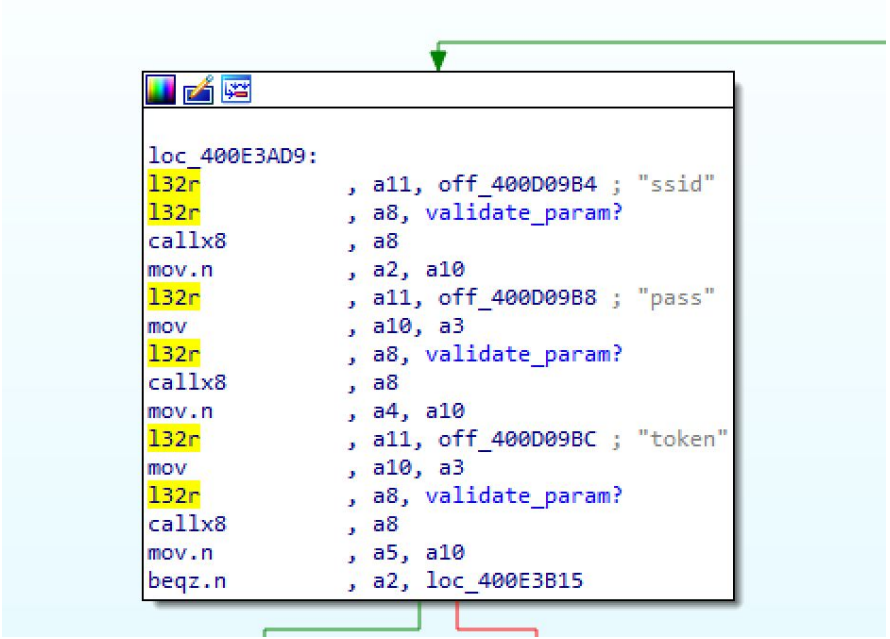
Thanks to https://github.com/themadinventor/ida-xtensa

tenable

Demo Time

CVE-2019-3998: SimpliSafe SS3 Wi-Fi Config Modification





Thanks to https://github.com/themadinventor/ida-xtensa

tenable

Demo Time

CVE-2019-3998: SimpliSafe SS3 Wi-Fi Config Modification

```
.[0;32mI (99406) bt_provisioning.c: GATT_WRITE_EVT, conn_id 0, trans_id 1, handle 42
.[0m
.[0;32mI (99406) bt_provisioning.c: GATT_WRITE_EVT, value len 18, value
{"ssid":"christes
.[0m
.[0;31mE (99412) BT: attribute value too long, to be truncated to 18.[0m
.[0;32mI (99506) bt_provisioning.c: GATT_WRITE_EVT, conn_id 0, trans_id 2, handle 42
.[0m
.[0;32mI (99506) bt_provisioning.c: GATT_WRITE_EVT, value len 18, value t", "pass":"mypass
.[0m
.[0;31mE (99512) BT: attribute value too long, to be truncated to 18.[0m
.[0;32mI (99605) bt_provisioning.c: GATT_WRITE_EVT, conn_id 0, trans_id 3, handle 42
.[0m
.[0;32mI (99606) bt_provisioning.c: GATT_WRITE_EVT, value len 18, value word", "token":"te
.[0m
.[0;31mE (99611) BT: attribute value too long, to be truncated to 18.[0m
id:0,Len:33,dp:0x3ffcba28

.[0;32mI (99706) bt_provisioning.c: GATT_WRITE_EVT, conn_id 0, trans_id 4, handle 42
.[0m
.[0;32mI (99707) bt_provisioning.c: GATT_WRITE_EVT, value len 5, value st"}

.[0m
.[0;31mE (99712) bt_provisioning.c: SSID: christest, PASSWORD: mypassword.[0m
I (99718) wifi: state: run -> init (0)
I (99722) wifi: n:6 0, o:6 0, ap:255 255, sta:6 0, prof:1
I (99727) wifi: pm stop, total sleep time: 66684074/34844666
```

Thanks to https://github.com/themadinventor/ida-xtensa

tenable

GitHub Project

https://github.com/tenable/esp32_image_parser

Questions?

Tenable Careers

tenable