Unwanted features:

Finding and exploiting an in ROM buffer overflow on the LPC55S69

Intro

- Hi! I'm Laura!

Intro

- Hi! I'm Laura!
- I work for Oxide! We're making a server!

Intro

- Hi! I'm Laura!
- I work for Oxide! We're making a server!
- It has a Hardware Root of Trust

Extending Trust

```
+---------+-----------+    +-------------+   +-------------+
|+-------+|           |    |             |   |             |
||       ++-->        |    |             |   |             |
||  ROM   |  RoT Code +--->|     SP      +-->|    Host     |
||        |           |    |             |   |             |
|+--------+           |    |             |   |             |
+---------+-----------+    +-------------+   +-------------+
```

Root of Trust Requirements

- A strong assertion regarding the integrity and authenticity of RoT firmware and hardware configuration
- A tamper-resistant, impersonation-resistant unique ID
- A mechanism for extending trust to additional devices
- A mechanism for re-establishing trust after a compromise

NXP LPC55S69

- Dual-core Cortex-M33
- CPU0 has TZ-M and MPU
- AES, SHA, and GF(p) accelerators
- SRAM-based PUF w/ protected key path to AES accelerator
- Secure boot (RSA-2048 or RSA-4096)
- DICE for measuring running code

We're shipping software

- Hardware features are useless without software
- We need a way to deliver software updates

The software update problem

The software update problem

Vendor value add

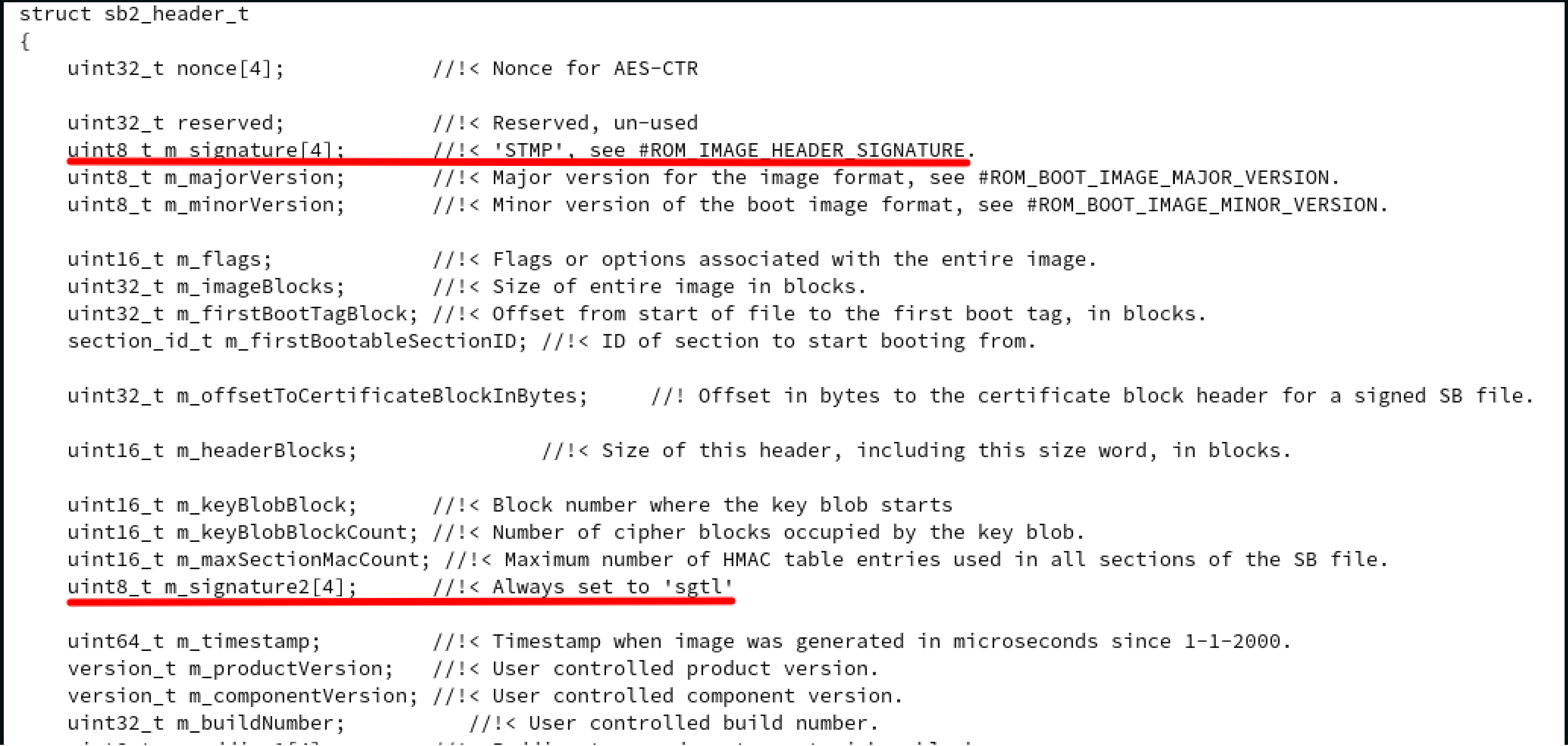
NXP's format for updates: SB2

- https://github.com/NXPmicro/spsdk
- Stands for Secure Boot
- SB2.0 = Encrypted, SB2.1 = Signed and Encrypted

SB2 format details

- Works in 16 byte blocks (also encryption block size)
- Unencrypted header (fixed size number of blocks)
- HMACs for commands/data
- Keyblob for encryption
- x.509 certificate if signed
- Commands and data (encrypted!)

SB2 format details



```
struct sb2_header_t
{
    uint32_t nonce[4];            //!< Nonce for AES-CTR

    uint32_t reserved;            //!< Reserved, un-used
    uint8_t m_signature[4];       //!< 'STMP', see #ROM_IMAGE_HEADER_SIGNATURE.
    uint8_t m_majorVersion;       //!< Major version for the image format, see #ROM_BOOT_IMAGE_MAJOR_VERSION.
    uint8_t m_minorVersion;       //!< Minor version of the boot image format, see #ROM_BOOT_IMAGE_MINOR_VERSION.

    uint16_t m_flags;             //!< Flags or options associated with the entire image.
    uint32_t m_imageBlocks;       //!< Size of entire image in blocks.
    uint32_t m_firstBootTagBlock; //!< Offset from start of file to the first boot tag, in blocks.
    section_id_t m_firstBootableSectionID; //!< ID of section to start booting from.

    uint32_t m_offsetToCertificateBlockInBytes;     //! Offset in bytes to the certificate block header for a signed SB file.

    uint16_t m_headerBlocks;               //!< Size of this header, including this size word, in blocks.

    uint16_t m_keyBlobBlock;      //!< Block number where the key blob starts
    uint16_t m_keyBlobBlockCount; //!< Number of cipher blocks occupied by the key blob.
    uint16_t m_maxSectionMacCount; //!< Maximum number of HMAC table entries used in all sections of the SB file.
    uint8_t m_signature2[4];      //!< Always set to 'sgtl'

    uint64_t m_timestamp;         //!< Timestamp when image was generated in microseconds since 1-1-2000.
    version_t m_productVersion;   //!< User controlled product version.
    version_t m_componentVersion; //!< User controlled component version.
    uint32_t m_buildNumber;          //!< User controlled build number.
```

A short history of silicon consolidation

- Came from Sigmatel, used in media chips
- Sigmatel was acquired by Freescale
- Freescale was merged into NXP

Keys on the LPC55S69

- UDS in key store is used with DICE
- SBKEK in the Key Store is used for decryption
- CMPA contains hash of public keys
- When sealed CMPA and Key Store cannot be modified

Threat modeling

- Parsing/Generating this code seems hard.
- This is the Root of Trust, if this is broken everything is broken
- Difficult things are where bugs can be found!

Previous work

- Undocumented ROM patch hardware block can be used to break privilege boundaries
- "Breaking TrustZone-M: Privilege Escalation on LPC55S69" @ DEFCON 2021 with my colleague Rick Altherr
- Good reason to be suspcious!
- Handy ROM dump hanging around

NXP's ROM

- First code that gets run
- Entry points for runtime (writing to flash, power management, signed image verificaiton, update code)
- Global state held in SRAM `0x1400_0000-0x1400_8000`

ISP (In-System Programming)

- Protocol over UART/SPI/I2C/CAN
- Fixed set of commands
- No access via JTAG/SWD when in ISP mode!
- Commands are restricted when CMPA is sealed, must use SB2 format to make modification to flash

ISP mode

```
+-----------+            +------------+
|           |   UART     |            |
|   LPC55   |<--------->|  Host      |
|           |            |            |
+-----------+            +------------+
```

Parsing the update

PSEUDO CODE

```
struct parsing_state {
        ...
        uint32_t next_addr;
        ...
        uint8_t[16] copied_data;
        ...
}
```

- 752 byte global storing parsing state

Parsing the update

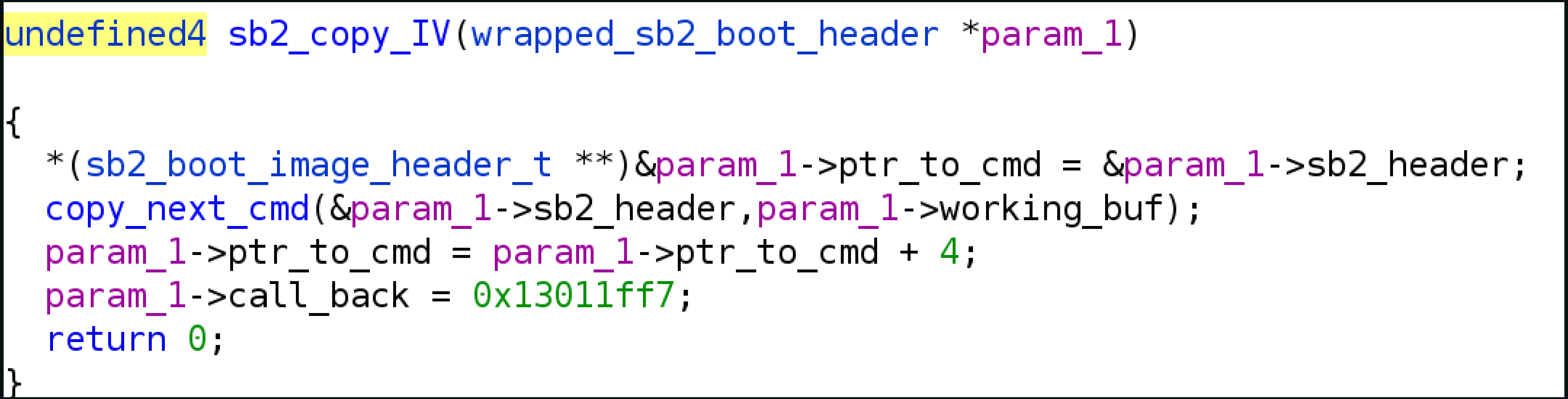
```c
struct parsing_state global_parsing_state;

void init_parsing_state() {
        memzero(global_parsing_state,
                        sizeof(global_parsing_state));
        global_parsing_state.next_addr = first_sb2_fn;
}
```

Parsing the update

PSEUDO CODE

```
struct parsing_state global_parsing_state;

uint32_t parse_update_bytes(uint8_t *data, uint32_t len) {
        uint32_t offset = 0;

        while offset < len {
                memcpy(&global_parsing_state.copied_data + offset,
                             data + offset, 16);
                global_parsing_state.next_addr();
                offset += 16;
        }
}
```

First function

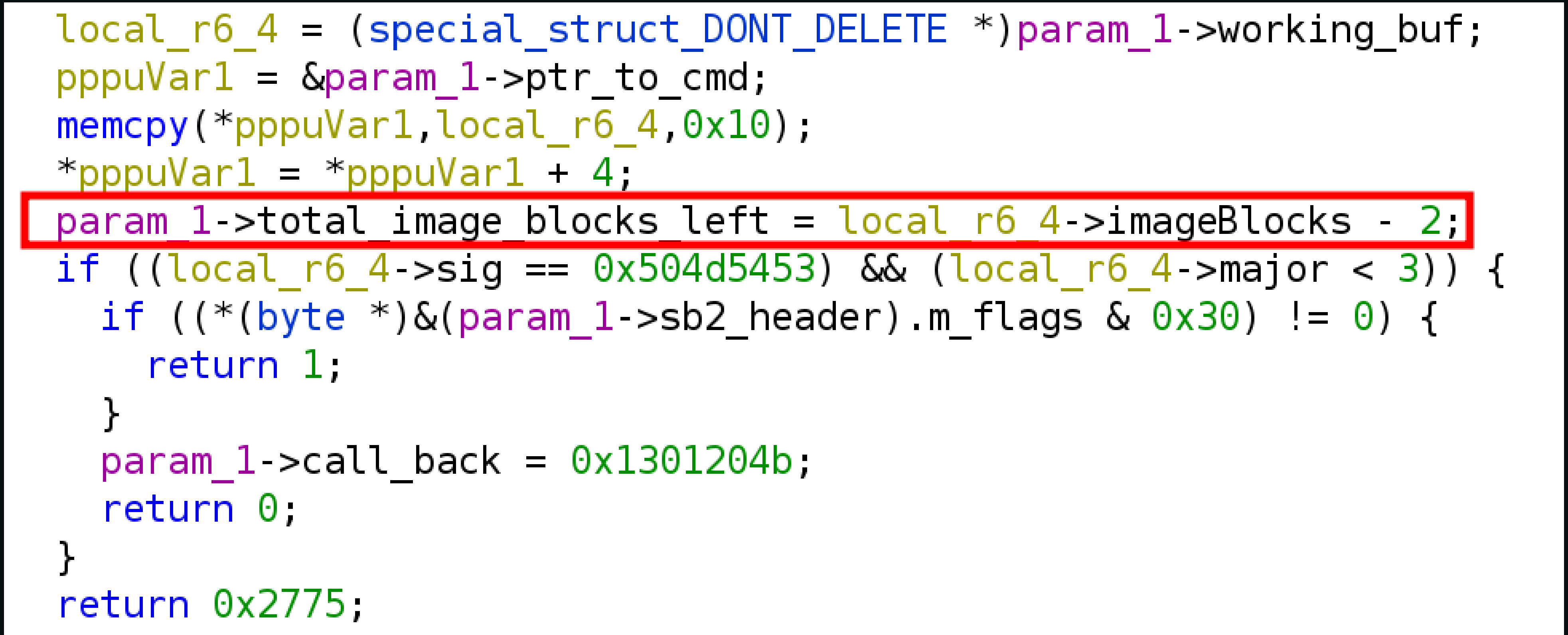


```
undefined4 sb2_copy_IV(wrapped_sb2_boot_header *param_1)

{
  *(sb2_boot_image_header_t **)&param_1->ptr_to_cmd = &param_1->sb2_header;
  copy_next_cmd(&param_1->sb2_header,param_1->working_buf);
  param_1->ptr_to_cmd = param_1->ptr_to_cmd + 4;
  param_1->call_back = 0x13011ff7;
  return 0;
}
```

Updated global state

```
struct parsing_state {
    ...
    uint32_t next_addr;
    ...
    uint8_t[16] copied_data;
    ...
    sb2_header header;
    ...

}
```

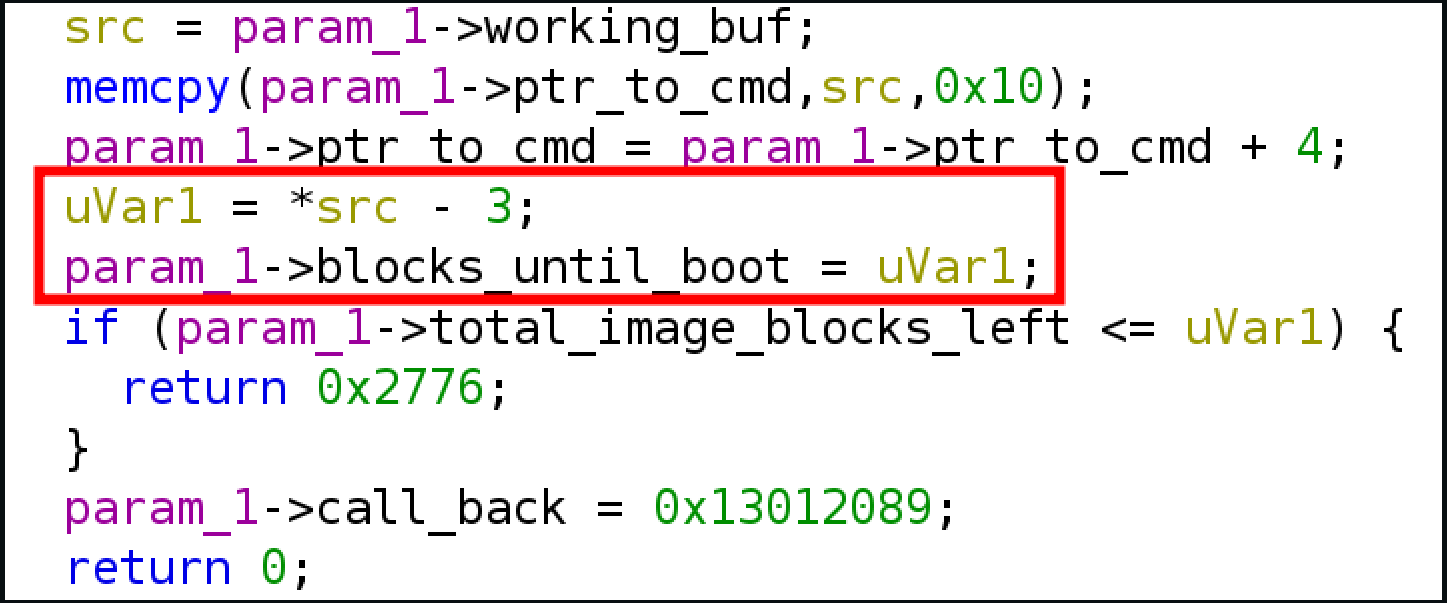
Second function



```
local_r6_4 = (special_struct_DONT_DELETE *)param_1->working_buf;
pppuVar1 = &param_1->ptr_to_cmd;
memcpy(*pppuVar1,local_r6_4,0x10);
*pppuVar1 = *pppuVar1 + 4;
param_1->total_image_blocks_left = local_r6_4->imageBlocks - 2;
if ((local_r6_4->sig == 0x504d5453) && (local_r6_4->major < 3)) {
  if ((*(byte *)&(param_1->sb2_header).m_flags & 0x30) != 0) {
    return 1;
  }
  param_1->call_back = 0x1301204b;
  return 0;
}
return 0x2775;
```

- This is untrusted input!

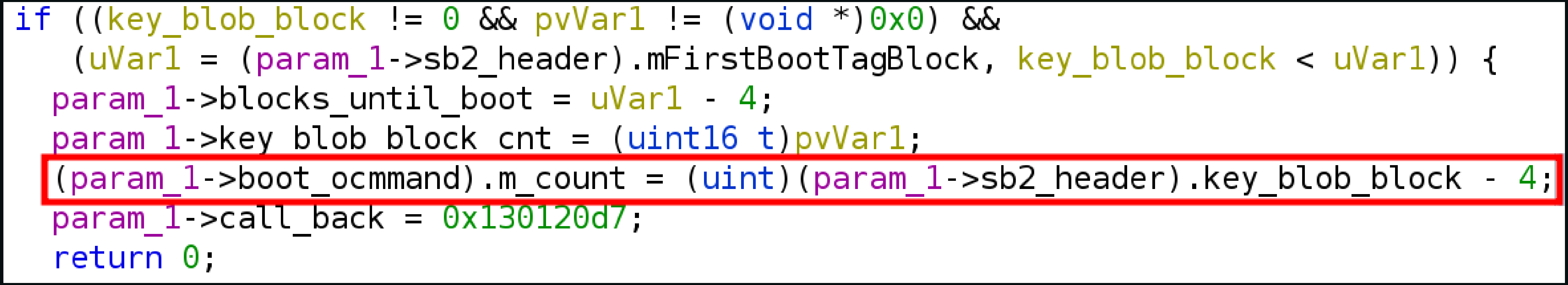
Third function



```
src = param_1->working_buf;
memcpy(param_1->ptr_to_cmd,src,0x10);
param_1->ptr_to_cmd = param_1->ptr_to_cmd + 4;
uVar1 = *src - 3;
param_1->blocks_until_boot = uVar1;
if (param_1->total_image_blocks_left <= uVar1) {
  return 0x2776;
}
param_1->call_back = 0x13012089;
return 0;
```

- Oh no another underflow

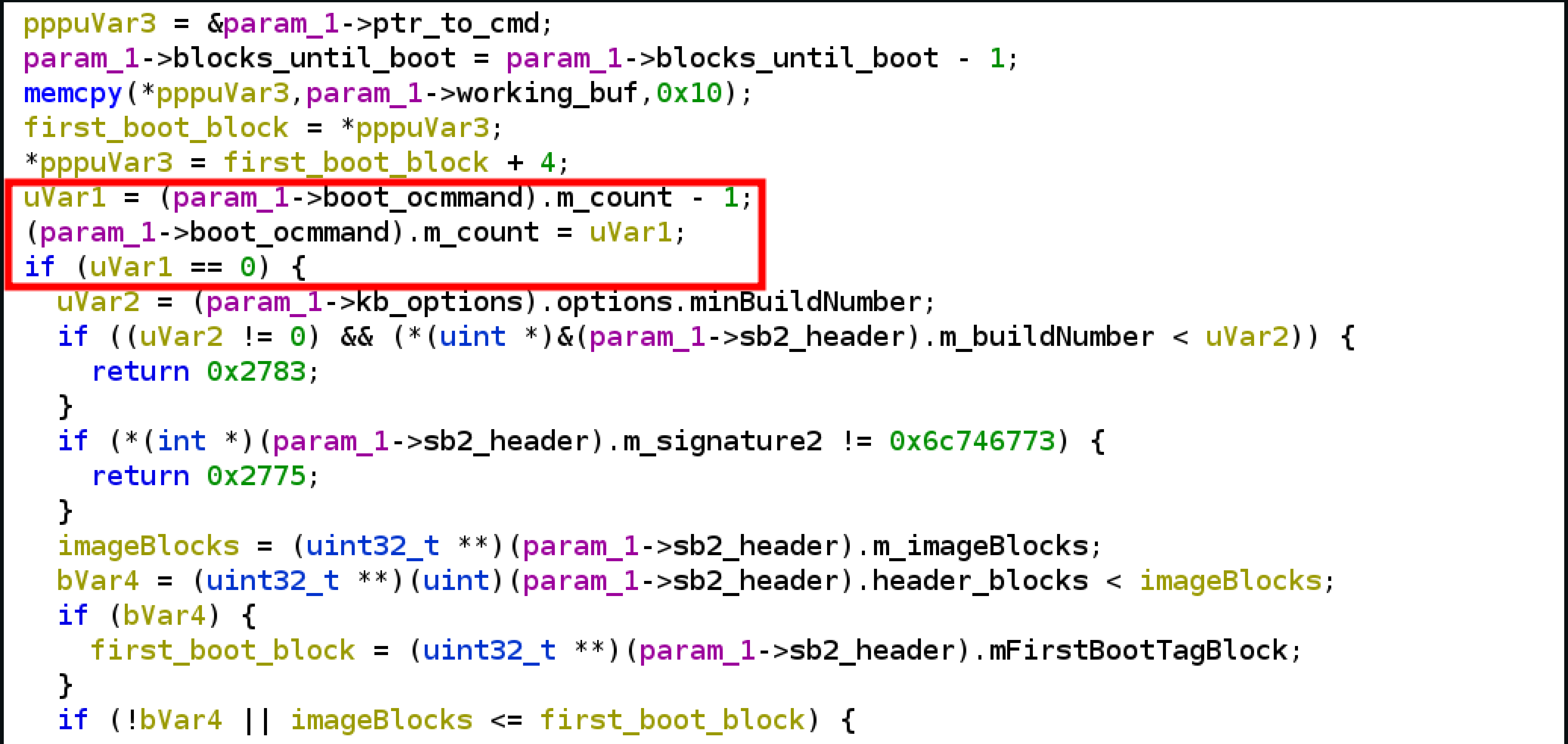
Fourth function



```
if ((key_blob_block != 0 && pvVar1 != (void *)0x0) &&
   (uVar1 = (param_1->sb2_header).mFirstBootTagBlock, key_blob_block < uVar1)) {
  param_1->blocks_until_boot = uVar1 - 4;
  param_1->key_blob_block_cnt = (uint16_t)pvVar1;
  (param_1->boot_ocmmand).m_count = (uint)(param_1->sb2_header).key_blob_block - 4;
  param_1->call_back = 0x130120d7;
  return 0;
```

- The value that is being stored into `m_count` is coming straight from header

Fifth function

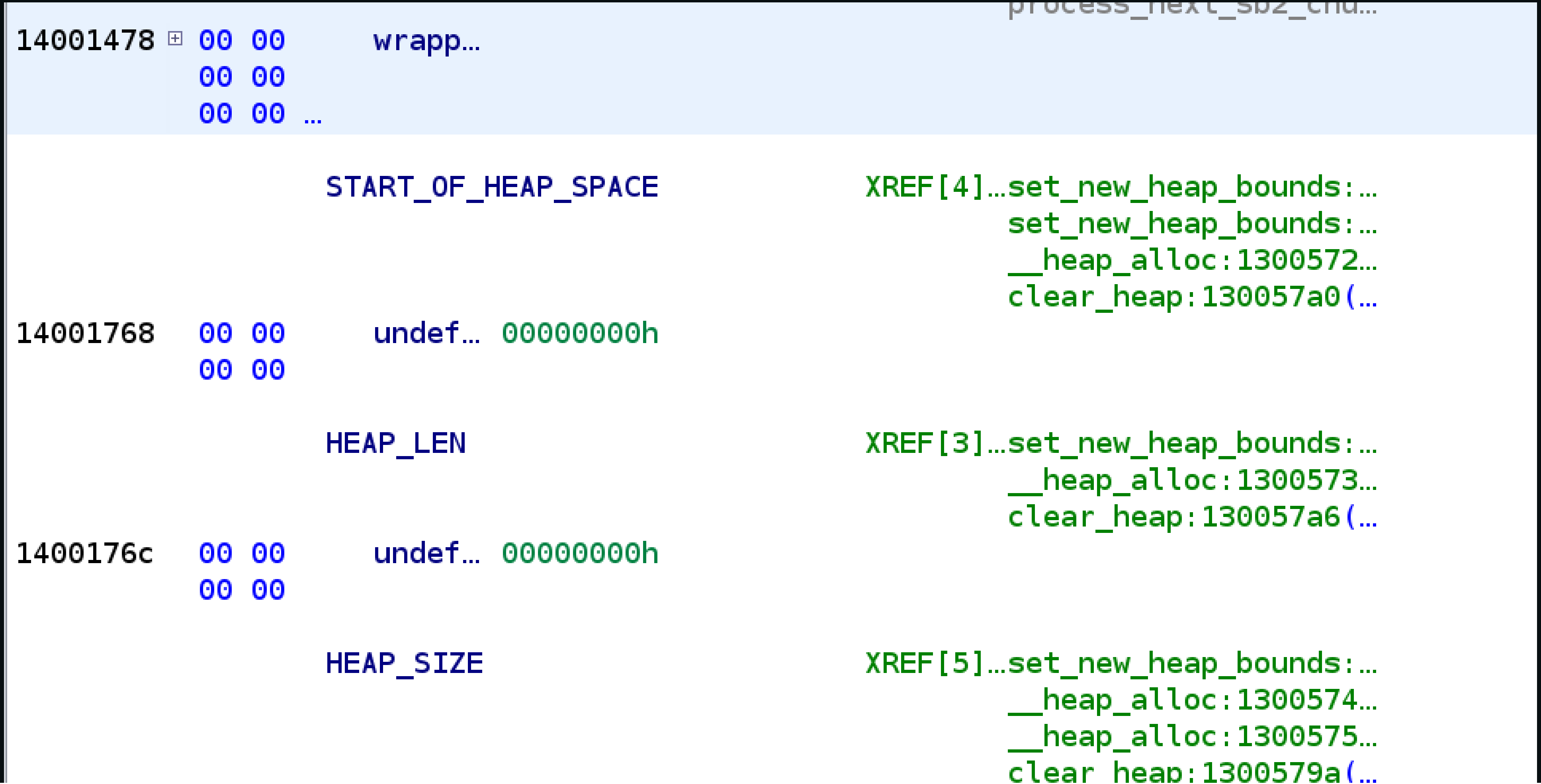


```
pppuVar3 = &param_1->ptr_to_cmd;
param_1->blocks_until_boot = param_1->blocks_until_boot - 1;
memcpy(*pppuVar3,param_1->working_buf,0x10);
first_boot_block = *pppuVar3;
*pppuVar3 = first_boot_block + 4;
uVar1 = (param_1->boot_ocmmand).m_count - 1;
(param_1->boot_ocmmand).m_count = uVar1;
if (uVar1 == 0) {
  uVar2 = (param_1->kb_options).options.minBuildNumber;
  if ((uVar2 != 0) && (*(uint *)&(param_1->sb2_header).m_buildNumber < uVar2)) {
    return 0x2783;
  }
  if (*(int *)(param_1->sb2_header).m_signature2 != 0x6c746773) {
    return 0x2775;
  }
  imageBlocks = (uint32_t **)(param_1->sb2_header).m_imageBlocks;
  bVar4 = (uint32_t **)(uint)(param_1->sb2_header).header_blocks < imageBlocks;
  if (bVar4) {
    first_boot_block = (uint32_t **)(param_1->sb2_header).mFirstBootTagBlock;
  }
  if (!bVar4 || imageBlocks <= first_boot_block) {
```

Fifth function

- Unless the callback gets updated this will call the same function again
- Callback is within the if condition
- Will call memcpy each time until it reaches `m_count`
- This is attacker controlled input! Buffer overflow!

Global Space

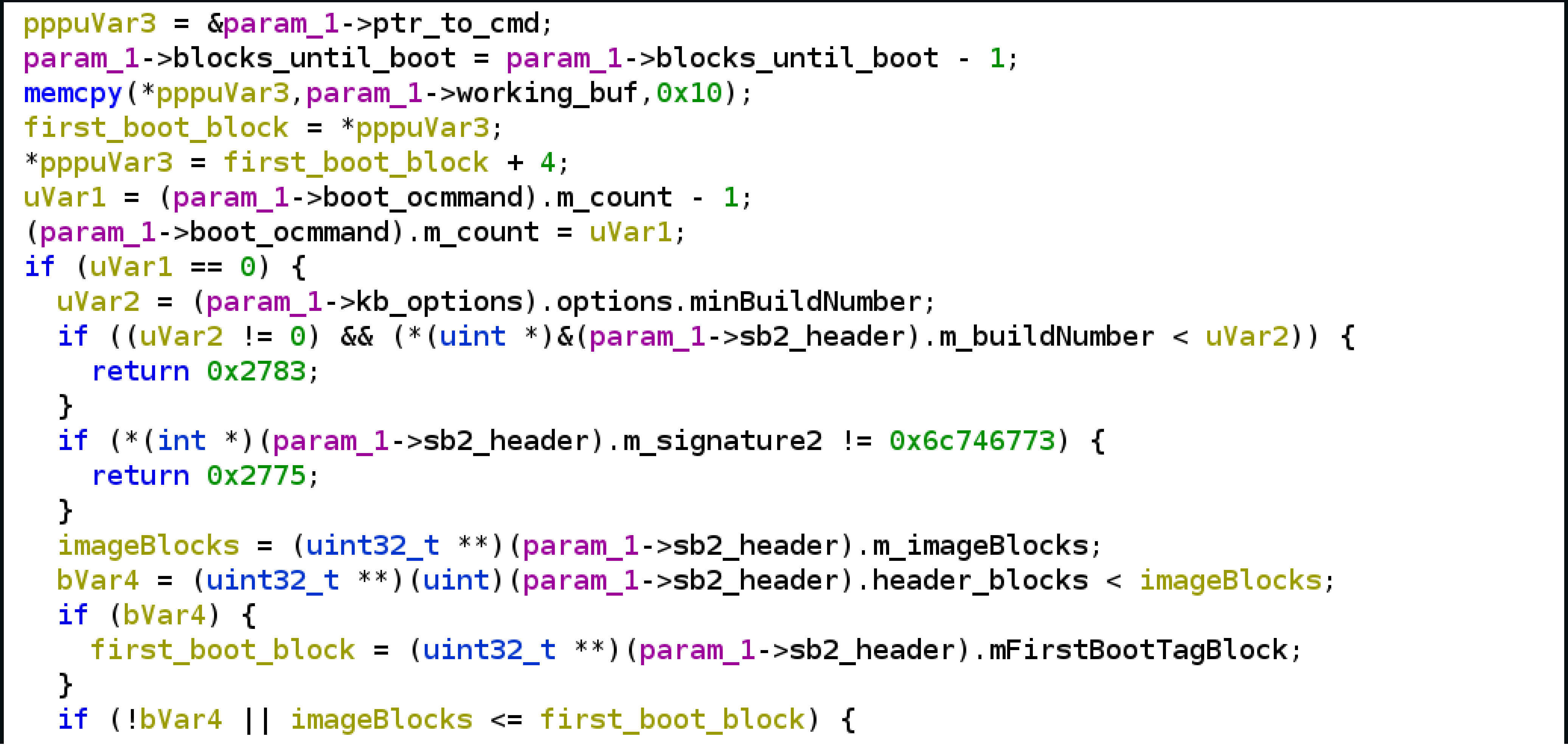


```
                                                           process_next_sb2_cmd…
14001478 ⊞ 00 00     wrapp…
           00 00
           00 00 …
                  START_OF_HEAP_SPACE            XREF[4]…set_new_heap_bounds:…
                                                         set_new_heap_bounds:…
                                                         __heap_alloc:1300572…
                                                         clear_heap:130057a0(…
14001768   00 00     undef… 00000000h
           00 00
                  HEAP_LEN                       XREF[3]…set_new_heap_bounds:…
                                                         __heap_alloc:1300573…
                                                         clear_heap:130057a6(…
1400176c   00 00     undef… 00000000h
           00 00
                  HEAP_SIZE                      XREF[5]…set_new_heap_bounds:…
                                                         __heap_alloc:1300574…
                                                         __heap_alloc:1300575…
                                                         clear_heap:130057 9a(…
```

Heap allocation?

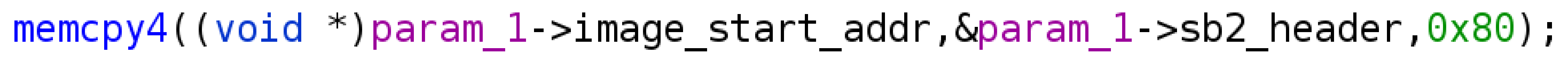
- Very simple bump allocator
- ROM is very simple, nearly everything is stack allocated
- Exceptions are certificate parsing -- library seems to be a form of mbedTLS!

Heap Allocation



```
pppuVar3 = &param_1->ptr_to_cmd;
param_1->blocks_until_boot = param_1->blocks_until_boot - 1;
memcpy(*pppuVar3,param_1->working_buf,0x10);
first_boot_block = *pppuVar3;
*pppuVar3 = first_boot_block + 4;
uVar1 = (param_1->boot_ocmmand).m_count - 1;
(param_1->boot_ocmmand).m_count = uVar1;
if (uVar1 == 0) {
  uVar2 = (param_1->kb_options).options.minBuildNumber;
  if ((uVar2 != 0) && (*(uint *)&(param_1->sb2_header).m_buildNumber < uVar2)) {
    return 0x2783;
  }
  if (*(int *)(param_1->sb2_header).m_signature2 != 0x6c746773) {
    return 0x2775;
  }
  imageBlocks = (uint32_t **)(param_1->sb2_header).m_imageBlocks;
  bVar4 = (uint32_t **)(uint)(param_1->sb2_header).header_blocks < imageBlocks;
  if (bVar4) {
    first_boot_block = (uint32_t **)(param_1->sb2_header).mFirstBootTagBlock;
  }
  if (!bVar4 || imageBlocks <= first_boot_block) {
```

Heap Allocation



```
memcpy4((void *)param_1->image_start_addr,&param_1->sb2_header,0x80);
```

- We copy the header to the heap

What do we have

- Can overwrite address of heap
- Header gets copied to an address we choose
- How can we get code exec?

We have a convenient callback!

- Address of callback is at offset `0x4` of the global parsing structure
- Offset `0x4` of our header contains nonce data
- If we put an address in place of our nonce we can control the address

Putting it all together

- Craft a custom header with `keyBlobBlock` set to the amount to write
- Header has address to jump to at offset 0x4
- Pad out bytes to overwrite heap address with start of global structure
- After `memcpy`ing the header on the next loop it will jump to our address
- Winner!

```
0x14001478 +-----------------------------+ ----+
           |        Next Fn              |     |
           +-----------------------------+     |
           |                             |     |
           |                             |     |
           |                             |     |
           +-----------------------------+     |  Global Parsing state
           |                             |     |
           |       SB2 Header            |     |
           |                             |     |
           +-----------------------------+     |
           |                             |     |
           |                             |     |
           +-----------------------------+ ----+
           |       Heap Address          |     |
           +-----------------------------+     |  Heap State
           |                             |     |
           +-----------------------------+ ----+
0x14001478 +-----------------------------+ ----+
           |        Next Fn              |     |
           +-----------------------------+     |
           |                             |     |
           |                             |     |
           |                             |     |
           +---+-------------------+-----+     |  Global Parsing state
           |   |                   |     |     |
           |   |   SB2 Header      |     |     |
           |   |                   |     |     |
           +---+-------------------+-----+     |
           |   |                   |     |     |
           |   |                   |     |     |
           +---+-------------------+-----+ ----+
           |   v   0x14001478      v     |     |
           +-----------------------------+     |  Heap State
           |                             |     |
           +-----------------------------+ ----+
0x14001478 +-----------------------------+ ----+
           |   Value from our header     |     |
           +-----------------------------+     |
           |                             |     |
           |                             |     |
           |                             |     |
           +---+-------------------+-----+     |  Global Parsing state
           |   |                   |     |     |
           |   |   SB2 Header      |     |     |
           |   |                   |     |     |
           +---+-------------------+-----+     |
           |   |                   |     |     |
           |   |                   |     |     |
           +---+-------------------+-----+ ----+
           |   v   0x14001478      v     |     |
           +-----------------------------+     |  Heap State
           |                             |     |
           +-----------------------------+ ----+
```

Not full execution

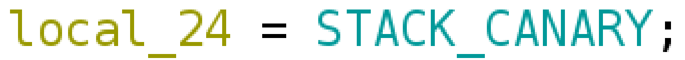
- Only gets access to ROM addresses
- SAU/MPU protections are enabled.
    - As a hacker I am saddened.
    - As a product developer I am thrilled

Previous work on the ROM patcher

- ROM patcher can insert `svc` instructions to trigger a system call
- The point of ROM patching is that the data isn't in ROM
- Also must be executable
- Where does the table live? A region at towards the end of SRAM

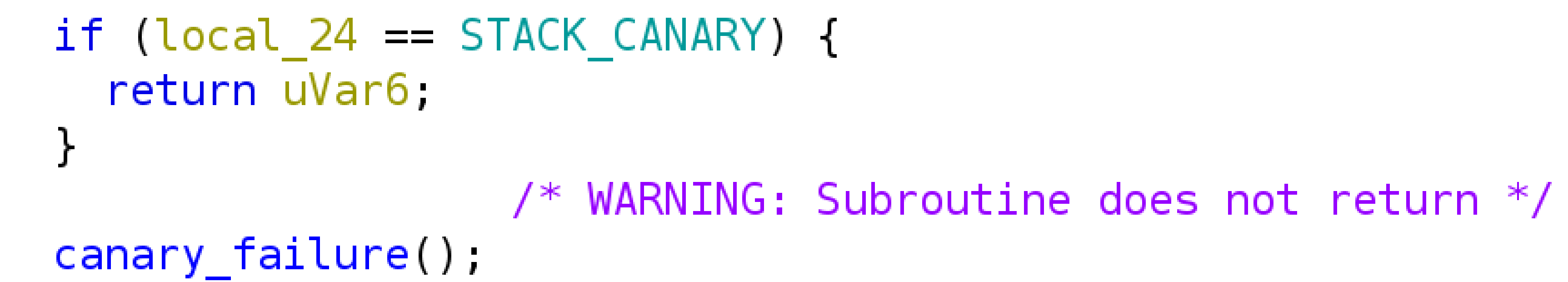
```
+------------------------------+
|                              |
|                              |
|                              |
|                              |
|     Global Parsing State     |
|                              |
|                              |
|                              |
|                              |
+------------------------------+
|                              |  |
|       Heap State             |  |
|                              |  |
+------------------------------+  |
|                              |  |
|                              |  |
|                              |  |
|      Other Stuff             |  |
|                              |  |
|                              |  |   Keep writing!
|                              |  |
|       We (mostly)            |  |
|      don't care about this   |  |
|                              |  |
|                              |  |
+------------------------------+  |
|                              |  |
|  Executable ROM Patch Area   |  v
|                              |
+------------------------------+
```

232 bytes from the end and we hit a snag!



```
local_24 = STACK_CANARY;
```

232 bytes from the end and we hit a snag!



```
  if (local_24 == STACK_CANARY) {
    return uVar6;
  }
                    /* WARNING: Subroutine does not return */
  canary_failure();
```

Canaries

- I <3 stack canaries this is a good thing!
- We're overwriting the global part of the stack canary
- Doesn't get detected in the SB2 parsing, further up in the ISP code
- Reverse stack canary -- we're not detecting a stack smash

Workaround

Putting it all together

- Custom header with `keyBlobBlock` set to the length we need to write and offset `0x4` set to our executable region of SRAM.
- Overwrite our heap address with the address of parsing global state
- Continue writing right up to the stack canary
- Overwrite the stack canary + executable area in one 512 byte chunk
- Executable area contains a small payload to turn off SAU/MPU, do a jump wherever
- Finish our overflow, copy our header to the heap address (i.e. global state)
- Next time around the parsing loop we execute our executable function!

Demo!

Product implications

- Worth discussing what this can't get you
    - Can't unseal anything (CMPA/NMPA)
    - If sealed, CMPA and keystore cannot be changed.
    - Region of flash covered by a signed image can't be changed

Product Implications

- Unprogramed pages == open for business
- Image rollback!
- CFPA, lead to DoS
- Running with an unsealed device really in trouble

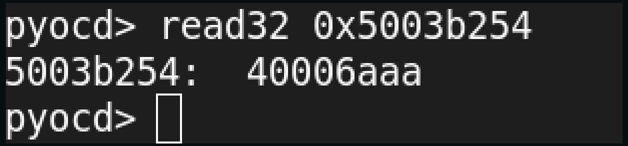
DICE

```
+--------------------+
|Device Secret       +------+
+--------------------+      |
                           +v-----------+
                           |Identity    |
                           +^-----------+
+--------------------+      |
|Hash of Running code+------+
+--------------------+

    ID = KDF(UDS, HASH(CODE))
```

PUF

```
+----------------+
|Chip Fingerprint+-----------+
+----------------+           |
                             |
+----------------+           |      +---------------+
| Key Index      +-----------+----> |Key Code       |
+----------------+           |      +---------------+
                             |
+----------------+           |
| Key Data       +-----------+
+----------------+
```

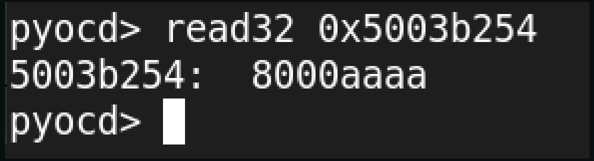
PUF Registers -- Before



```
pyocd> read32 0x5003b254
5003b254:  40006aaa
pyocd> 
```

- ROM designates key #15 for the UDS

PUF Registers -- After



```
pyocd> read32 0x5003b254
5003b254:  8000aaaa
pyocd> 
```

- A consequence of how DICE works: cannot be locked until after the image is booted, at the time of image update we do not know the image!

Fixes?

- This code is in ROM
- We need new hardware
- Semicoductor shortage? :shrug:

Hey aren't you trying to build a product out of this

- I promise my job is not just vuln hunting
- Oxide encourages this research and also really wish there were fewer bugs for me to find
- This is the second vulnerability we've found in this chip!

Why not switch chips?

- There were very few other candidates out there that met our requirements
- Even before the shortage could not get hands on actual silicon
- Need to do another cycle of review and validation
- Could find even more problems

Workaround: Can you validate the update?

- Theoretically yes!
- What does the validation now becomes part of our trusted base. How much do we trust the validation code?
- If we weren't building a Root of Trust this might be different!

Workaround: Siganture checking?

- Changes the threat model
- Signed code tells you nothing about correctness
- A signature only tells you the code came from a particular source
- If we weren't building a Root of Trust this might be different!

Oxide Answer: don't use this code at all

- Only using it because some engineer decided she didn't want to write update code
- Positive side: don't have to write SB2 parsing code

Takeaways

Validate your input

- Obviously
- Especially in ROMs (give us your ROM source)

Needed to get several things right

- MPU/SAU == Good
- Stack canary == Good
- Got lucky with convenient layout in the global space
- Make it hard for attackers

No single right answer for your product

- "It depends" is an annoying answer
- Alternate universe: we ran into other issues and had time to swap out the chip.
- If the product is focused around the LPC55 that also changes consideration

Thank you!