INSECURE BOOT

Andrea Barisani – Head of Hardware Security

$ whoami

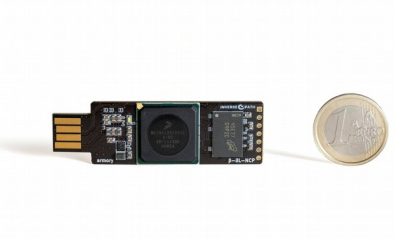
Andrea "lcars" Barisani

I am a 

Founder of INVERSE PATH now part of 

Breaking things since I got my first 

Securing  and much more since 2005.

Maker of the USB armory 

Speaker and trainer at BlackHat, CanSecWest, DEFCON, Hack In The Box, PacSec conferences among many others.

https://andrea.bio | @andreabarisani

HARDWARE SECURITY

F-Secure

For more than a decade my team has been focusing on testing hardware as well as software.

We test each and every layer.

We evaluate the attack surface obtained by physical possession and not just remote access.

We specialize in embedded systems for safety critical operations.

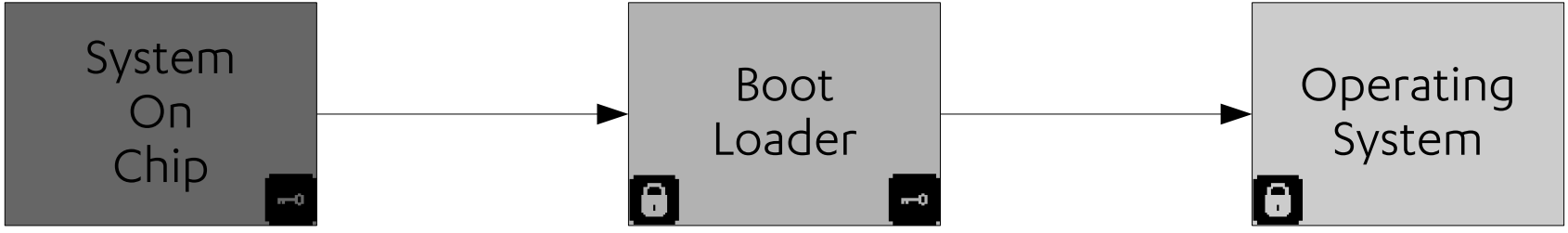
Secure Boot + Verified Boot

F-Secure

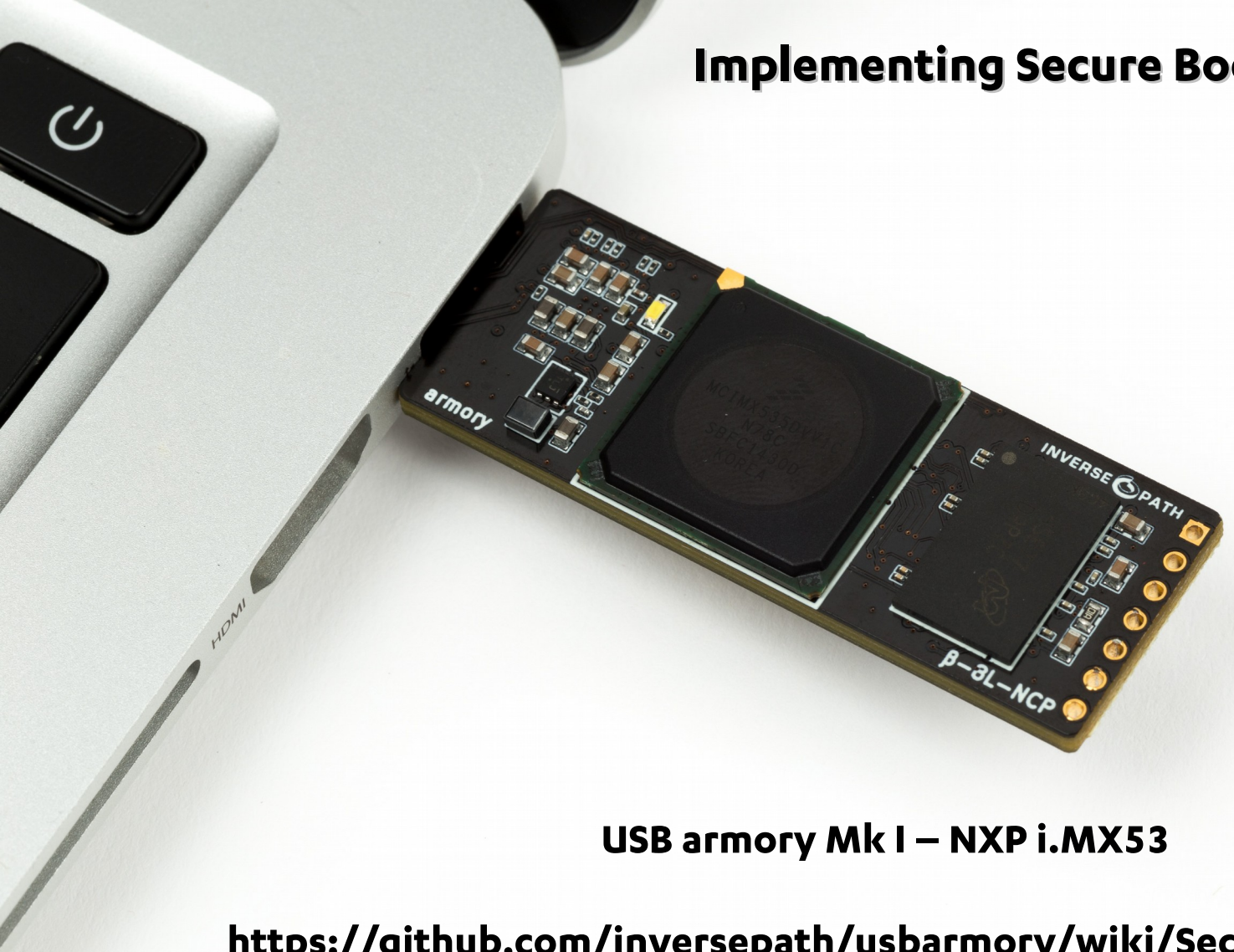
An hardware anchor (e.g. SoC boot ROM) authenticates the first user provisioned code, such as the bootloader.

The bootloader must then maintain the **chain of trust** with cryptographic verification of signed kernel images (e.g. public key verification).

The booted Operating System must also maintain the chain of trust by validating all executed code and ensuring proper lockdown.



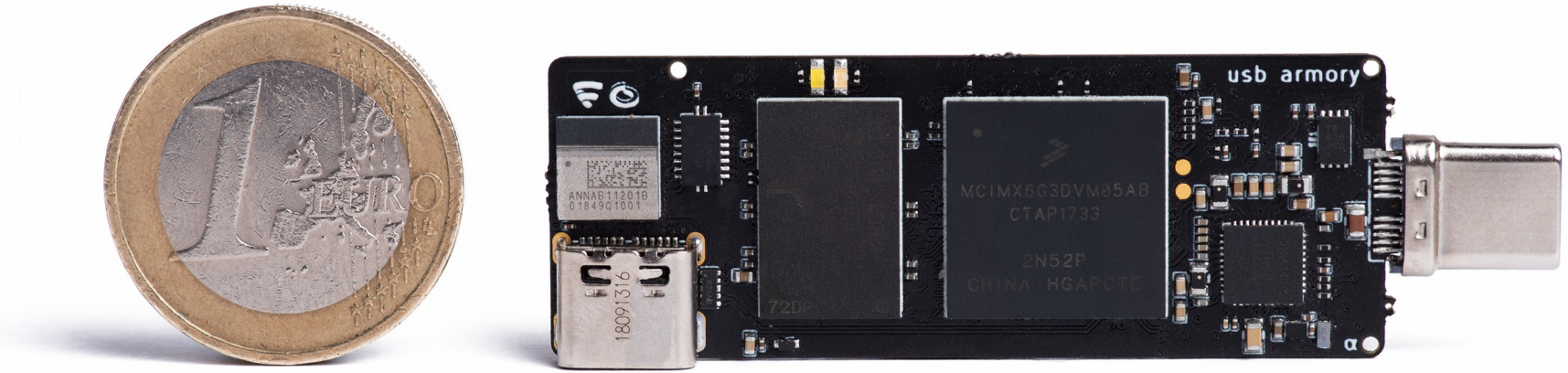
Implementing Secure Boot



USB armory Mk I – NXP i.MX53

https://github.com/inversepath/usbarmory/wiki/Secure-boot-(Mk-I)

Implementing Secure Boot



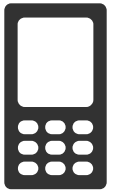
USB armory Mk II – NXP i.MX6UL / i.MX6ULZ

https://github.com/inversepath/usbarmory/wiki/Secure-boot-(Mk-II)

AUTOMOTIVE | AVIATION | INDUSTRIAL

F-Secure

ECU
Infotainment
Black box
External data logger
Car sharing add-on
...



Mobile
Gaming
Smart TV
Home automation
Routers
...

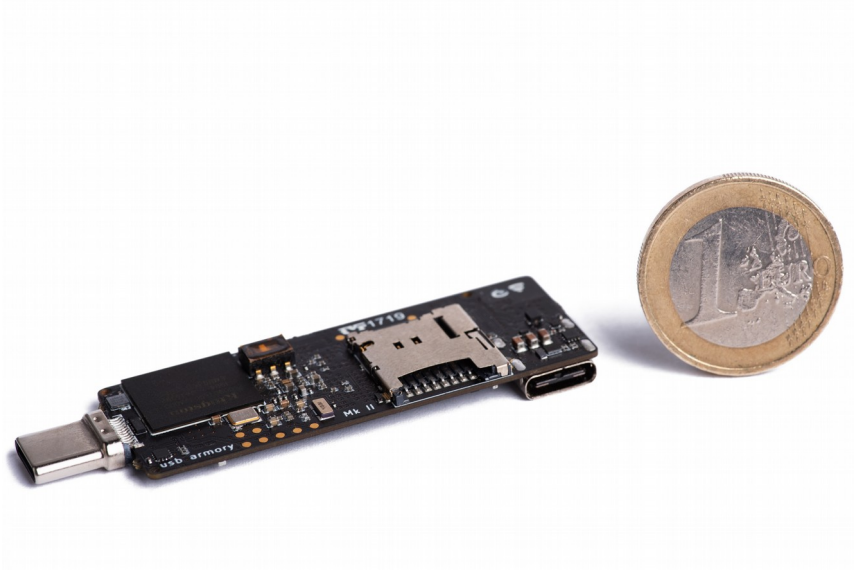
PLCs
Data logging
Networking
Alarm systems
Access controls
...

The **chain of trust** implemented through Secure/Verified Boot allows to protect Intellectual Property, provides **confidentiality of sensitive assets** and **authentication of trusted code**.

It is used in automotive, aviation, industrial and consumer devices.

Secure Boot Example

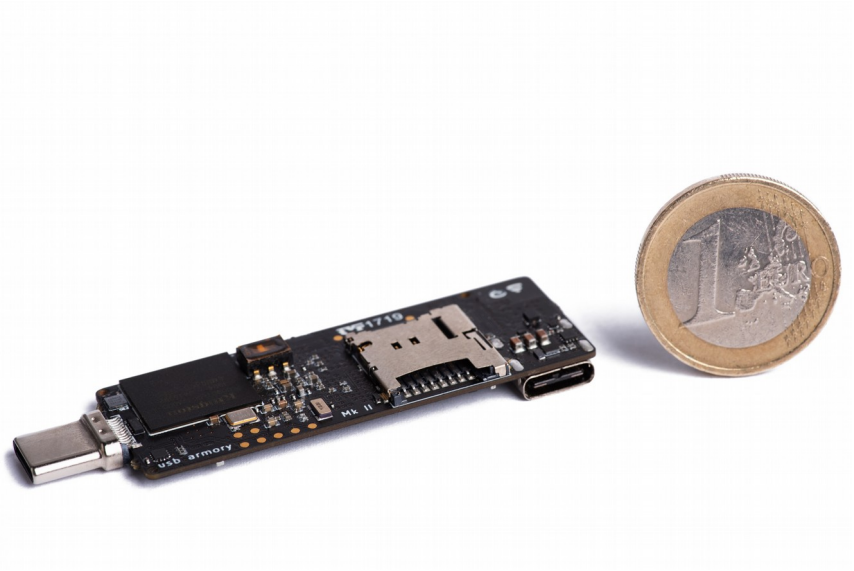
```
U-Boot 2019.07

CPU:   Freescale i.MX6UL at 900 MHz
Reset cause: POR
Board: Inverse Path USB armory Mk II
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0
In:    serial
Out:   serial
Err:   serial
Net:   CPU Net Initialization Failed
No ethernet found.
Hit any key to stop autoboot:  2
=> hab_status

Secure boot enabled

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!

=> boot
2301352 bytes read in 300 ms (7.3 MiB/s)
16670 bytes read in 178 ms (90.8 KiB/s)
## Booting kernel from Legacy Image at 70800000 ...
   Image Name:   Linux-5.2.11
```



Successfully booted signed U-Boot image.

Secure Boot Example

```
U-Boot 2019.07

CPU:   Freescale i.MX6UL at 900 MHz
Reset cause: POR
Board: Inverse Path USB armory Mk II
I2C:   ready
DRAM:  512 MiB
MMC:   FSL_SDHC: 0
In:    serial
Out:   serial
Err:   serial
Net:   CPU Net Initialization Failed
No ethernet found.
Hit any key to stop autoboot:  2
=> hab_status

Secure boot enabled

HAB Configuration: 0xf0, HAB State: 0x66

--------- HAB Event 1 -----------------
event data: ...

STS = HAB_FAILURE (0x33)
RSN = HAB_INV_SIGNATURE (0x18)
```



Failed attempt shown in verification mode, before final activation.

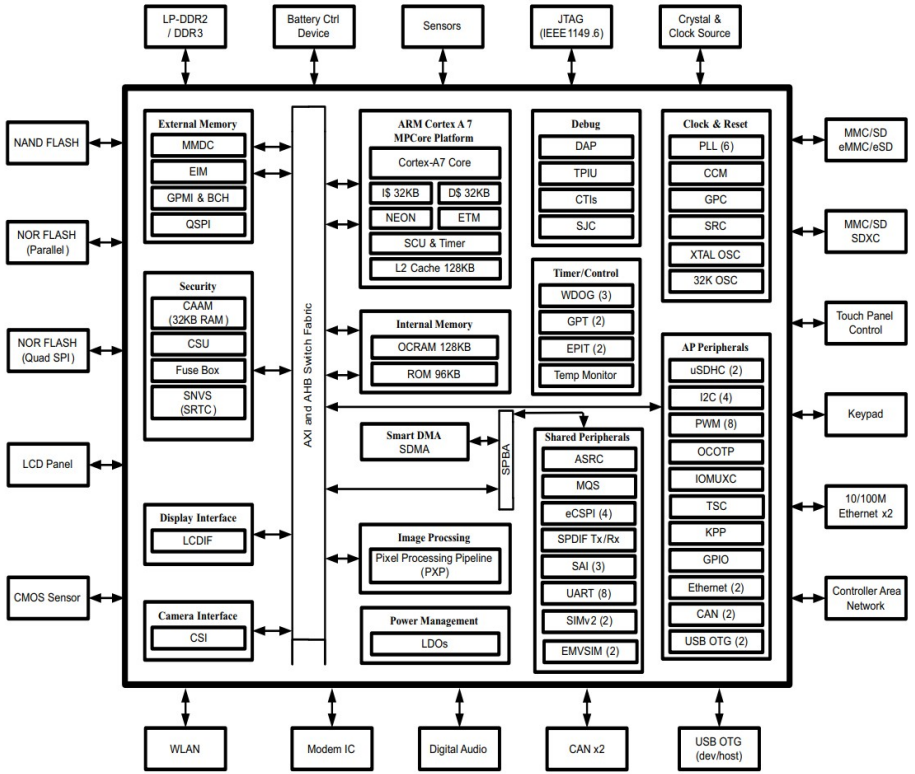
When active, failures hang the SoC without any printed information (this is a feature, not a bug).

Verified Boot

```
## Loading kernel from FIT Image at 70800000 ...
   Using 'conf@1' configuration
   Verifying Hash Integrity ... sha256,rsa2048:usbarmory+ sha256,rsa2048:usbarmory+ OK
   Trying 'kernel@1' kernel subimage
     Description:  Vanilla Linux kernel
     Type:         Kernel Image (no loading done)
     Compression:  uncompressed
     Data Start:   0x708000cc
     Data Size:    8724448 Bytes = 8.3 MiB
     Hash algo:    sha256
     Hash value:   c5949bf26f792c62fcd793f70041397240d8a17dab240f0e6b71f72dc59298b2
   Verifying Hash Integrity ... sha256+ OK
## Loading fdt from FIT Image at 70800000 ...
   Using 'conf@1' configuration
   Trying 'fdt@1' fdt subimage
     Description:  USB armory devicetree blob
     Type:         Flat Device Tree
     Compression:  uncompressed
     Data Start:   0x710521c4
     Data Size:    16670 Bytes = 16.3 KiB
     Architecture: ARM
     Hash algo:    sha256
     Hash value:   8d60182befa80d8ab8ebd4fce490a2226acd473ba815b578518867d9eeb71aaf
   Verifying Hash Integrity ... sha256+ OK
   Booting using the fdt blob at 0x710521c4
   XIP Kernel Image (no loading done) ... OK
   Loading Device Tree to 8f54d000, end 8f55411d ... OK
```

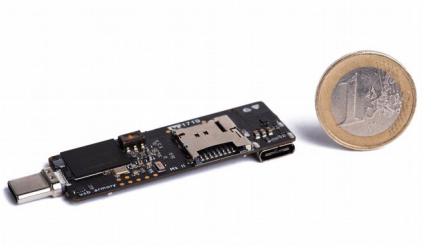
NXP – i.MX6UL /i.MX6ULZ

ARM® Cortex™-A7 528/900 MHz



Hardware security features

- High Assurance Boot (HAB 4.2.6)
- Cryptographic accelerator (CAAM/DCP)
- True Random Number Generator (TRNG)
- Bus Encryption Engine (BEE - OTF AES) (i.MX6UL only)
- Secure Non-Volatile Storage (SNVS)
- ARM® TrustZone®



NXP - Secure Non-Volatile Storage (SNVS)

The SNVS feature relies on the OTPMK, a **unique per-device hardware key**, which cannot be read directly as it can only be used via the SoC internal Cryptographic Accelerator and Assurance Module (CAAM), when secure booted.

The SNVS feature can be summarized as follows:

A random 256-bit blob encryption key (BK) is generated.

The **blob encryption key is used to encrypt the desired data** via the CAAM AES-CCM function, providing **confidentiality and integrity protection**.

The blob encryption key is AES-ECB encrypted with a key derived from the OTPMK (BKEK), using a Single-step Key-Derivation Function, resulting in the blob.

The HAB secure boot sequence, or runtime environment, can directly support authenticated decryption of arbitrary data blobs (including the bootloader image).

NXP CAAM + SNVS driver

F-Secure

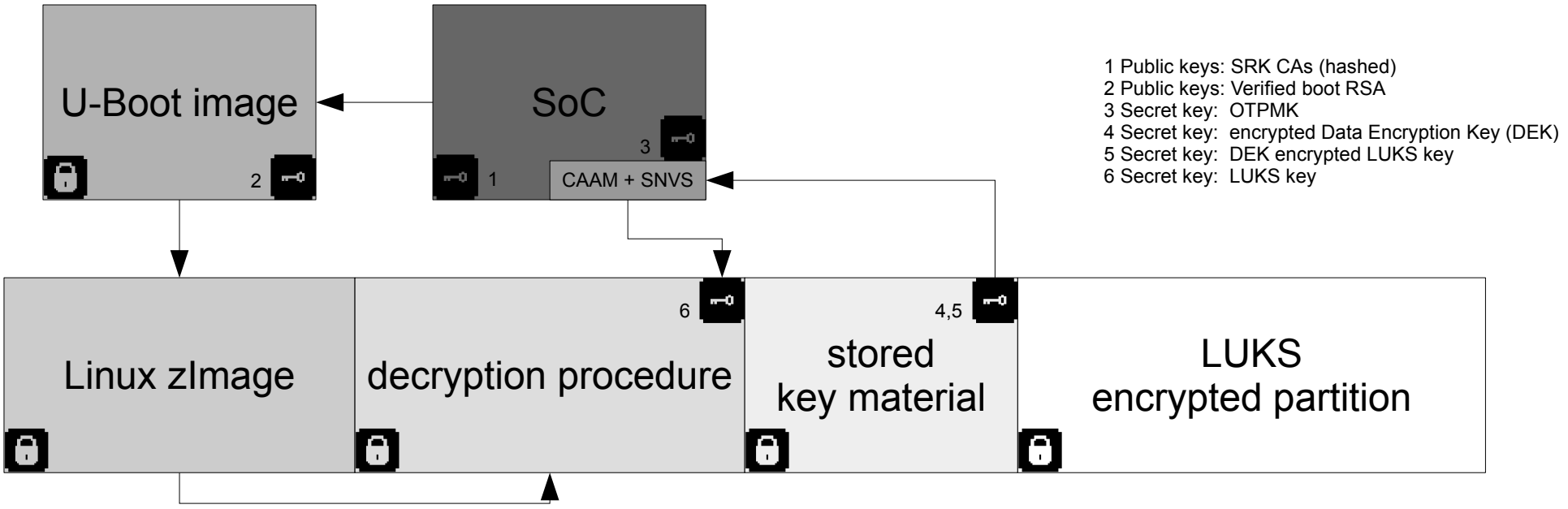
https://github.com/inversepath/caam-keyblob

```
$ sudo modprobe caam_keyblob

caam_keyblob: Secure State detected
```

Go userspace implementation:

```
$ caam_tool enc dek.bin dek_blob.bin
caam_tool: encrypting 32 bytes from dek.bin
caam_tool: caam_kb_data &{Text:0x49c000 TextLen:32 Blob:0x4a0000 BlobLen:80 Keymod:0x48c010 KeymodLen:16}
caam_tool: encrypted 80 bytes to dek.bin

$ caam_tool dec dek.bin dek_blob.bin
caam_tool: decrypting 80 bytes from dek_blob.bin
caam_tool: caam_kb_data &{Text:0x478000 TextLen:32 Blob:0x474000 BlobLen:80 Keymod:0x412140 KeymodLen:16}
caam_tool: decrypted 32 bytes to dek.bin
```

Full chain of trust example - i.MX6UL

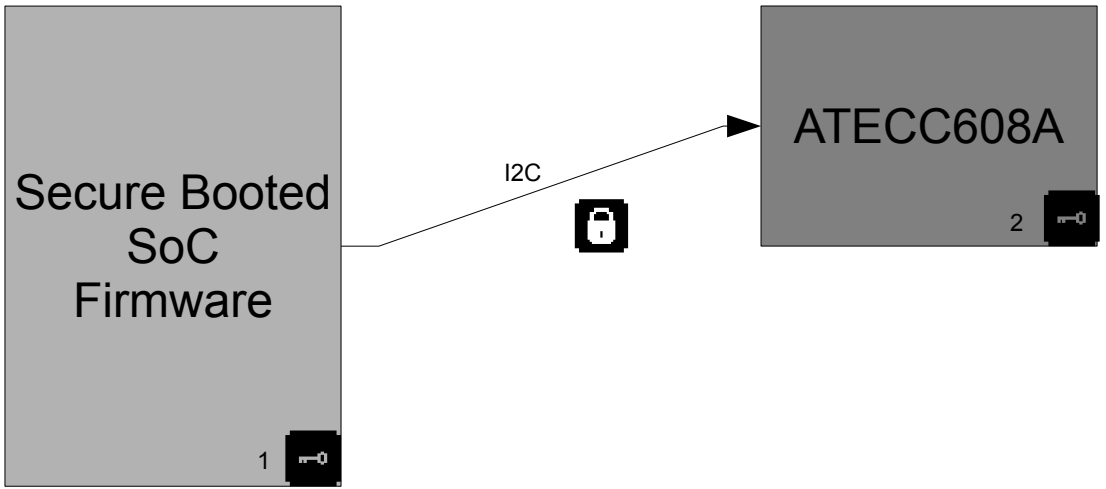


key material

authenticated + encrypted

- SoC authenticates U-Boot
- U-Boot authenticates Linux
- Linux uses SVNS decrypted key material to unlock the encrypted partition

Rollback protection + external keyring



1 Secret keys: ReadKey, WriteKey

2 Slots (16x) for key, certificates or data
High Endurance Monotonic Counters (2x)
OTPs (512-bit)

key material

authenticated + encrypted

The SoC can establish a secure session with the ATECC608A, using safely stored read and write keys, certificates or data.

This allows secure key/certificate access or use, additionally two High Endurance Monotonic Counters can be used for rollback protection.

Provides an additional hardware keyring for (partial) mitigation of further HAB issues.

ARM® TrustZone®

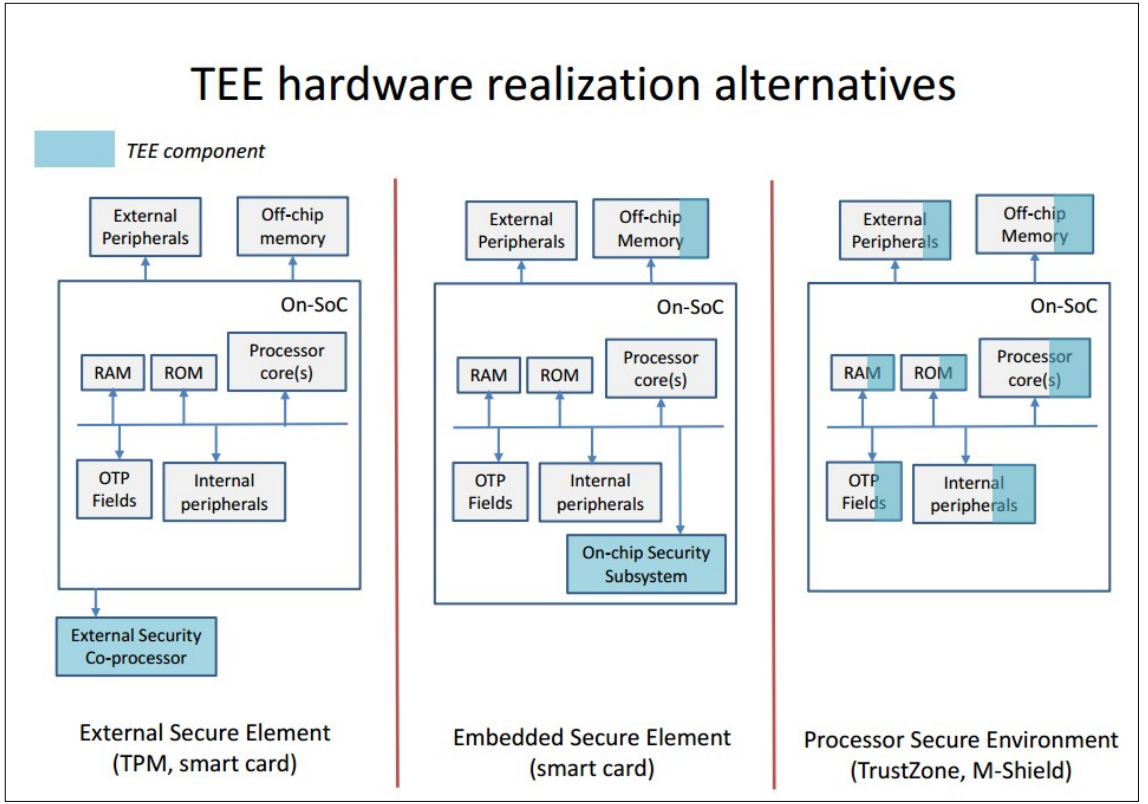
The ARM Security Extensions, marketed as TrustZone (TZ), represent a technology aimed to replace the need for a separate dedicated security core.

The extensions allow separation of the execution environment between two virtual processors, each implementing its own “world”.

Normal world | Secure world

Example use cases:

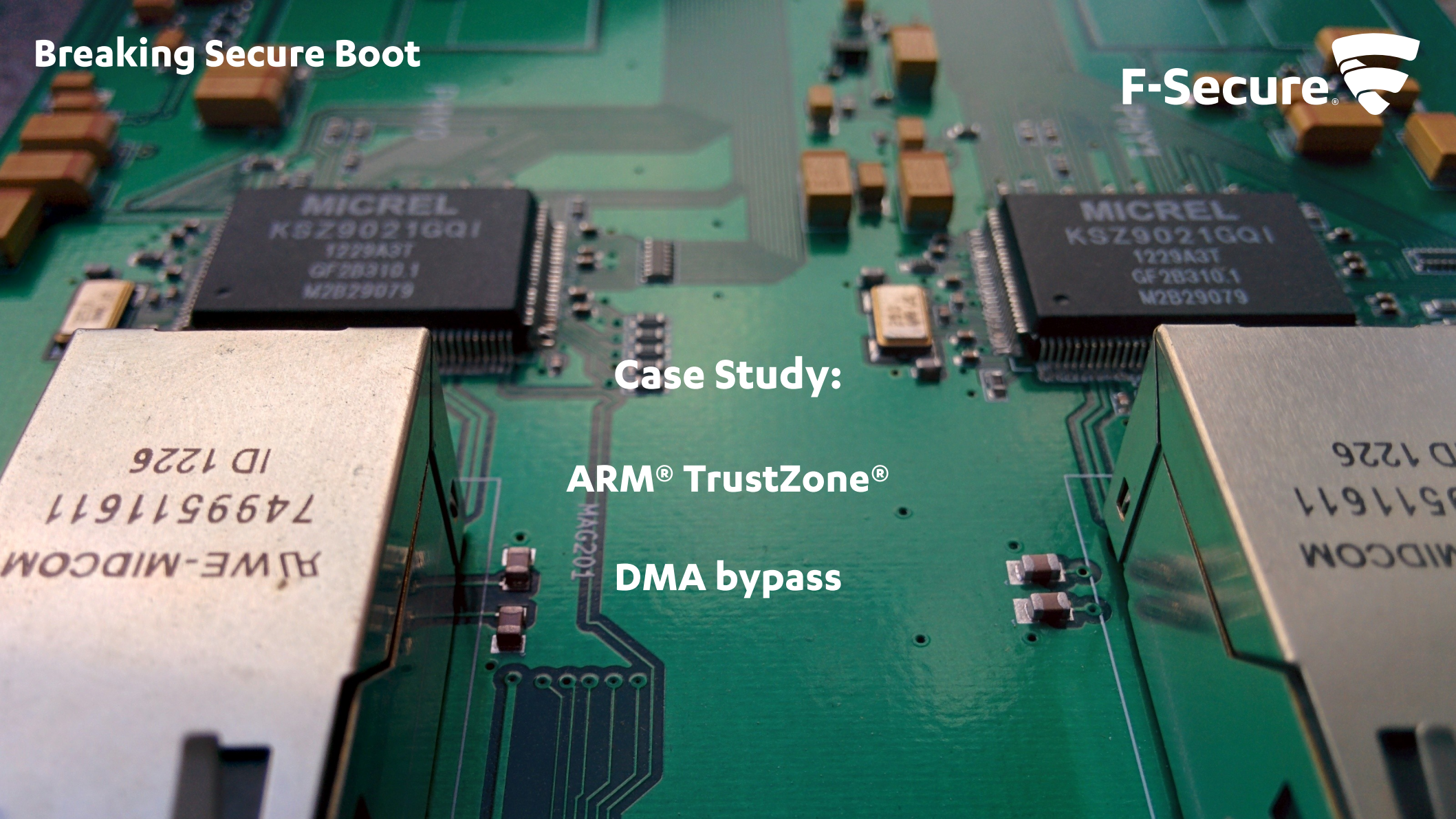
- Protected encryption/decryption for DRM purposes
- Secured user interaction (e.g. PIN)
- Keystore (e.g. Android)



Slide from ACM CCS 2013 TEE Tutorial

Breaking Secure Boot

F-Secure



Case Study:

ARM® TrustZone®

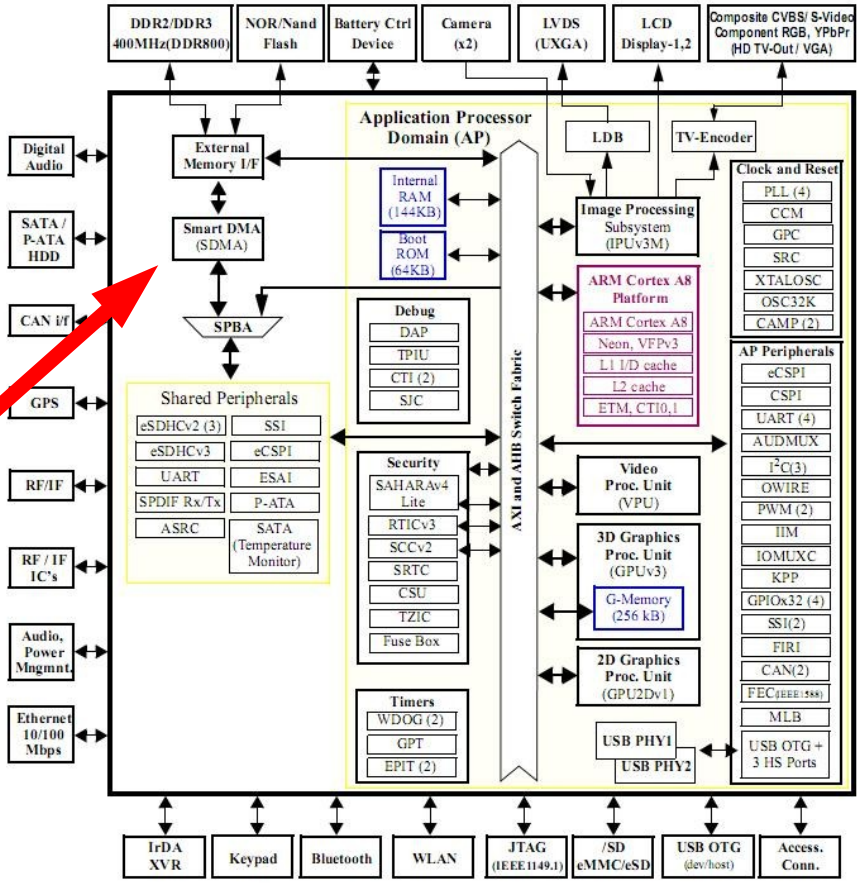
DMA bypass

Memory separation: DMA

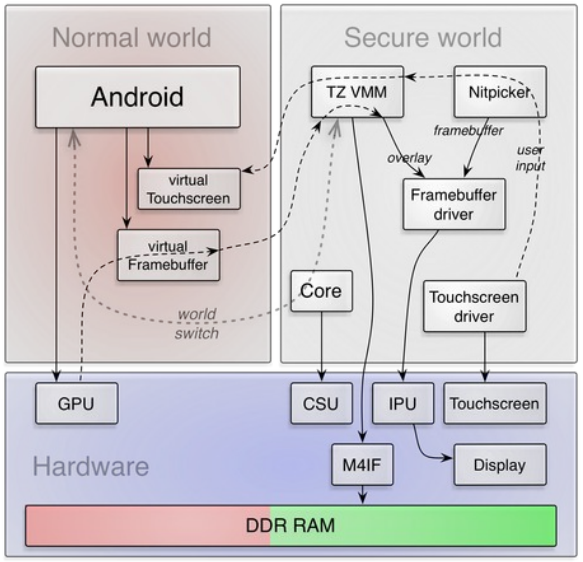
System memory access is also possible through Direct Memory Access (DMA), available controllers and features are vendor dependent.

First-party DMA (bus mastering): a device connected to the bus can initiate a transaction.

Third-party DMA (standard DMA): a system DMA controller performs the transfer.



Central Security Unit – Masters privilege policy



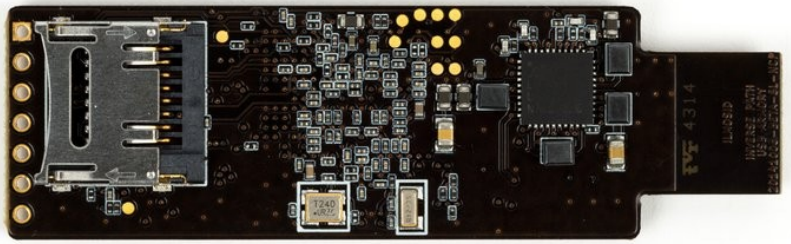
On the i.MX53 the VPU, IPU and GPU share the same Master ID, this is inconsistent with the Peripheral access policy granularity.

(flawed) separation of graphics components

```
Csu(addr_t const base) : Mmio(base)
{
...
    write<Csl24::Slave_b>(Csl00::UNSECURE); // GPU 2D
...
    write<Csl27::Slave_b>(Csl00::UNSECURE); // GPU 3D
...
     write<Csl24::Slave_a>(Csl00::SECURE);  // IPU
...
     write<Csl23::Slave_a>(Csl00::SECURE);  // VPU
...
     // GPU + IPU + VPU
     write<Master::Gpu>(Master::UNSECURE_UNLOCKED);
```

F-Secure

Case Study: HABv4



HABv4 bypass

In 2017 Quarkslab discovered critical security vulnerabilities that affect HABv4 on the entire NXP i.MX series.

The issue was reported for the i.MX6, Inverse Path immediately investigated applicability to the i.MX53.

A X.509 parsing error (ERR010873 | CVE-2017-7932) and an SDP protection bypass (ERR01872 | CVE-2017-7936) allow arbitrary code execution on SoC in Closed configuration.

The findings prevent the secure operation of unattended setups while attended setups remain protected in case of device loss (but not tampering).

NXP did not release any P/N updates for the i.MX53.

https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_QBVR2017-0001.txt

HABv4 bypass


```
Timeline
========

2017-05-18: Quarkslab presents findings at the 2017 Qualcomm Mobile Security
            Summit [9], materials are not disclosed to the public at this time.
2017-05-30: Quarkslab communicates embargo period until 2017-07-18.
2017-05-30: Inverse Path proposes preliminary advisory release on 2017-06-05.
2017-06-05: Inverse Path releases preliminary advisory.
2017-06-06: added assigned CVE numbers.
2017-07-19: Quarkslab public release of findings [4].
2017-07-19: Inverse Path release of full advisory and i.MX53 PoC [6].
2017-07-27: added link to i.MX Community post that lists affected P/Ns.
```

F-Secure prioritized announcing the existence of the issue before the full advisory release, additionally developed and released a full PoC.

The `usbarmory_csftool` is the only Open Source implementation for HABv4 signing as well as the first and only exploitation tool ;-)

“Break your own product, and break it hard”
https://labsblog.f-secure.com/2017/07/19/break-your-own-product-and-break-it-hard/

HABv4 bypass

F-Secure

```
$ usbarmory_csftool -h
Usage: usbarmory_csftool [OPTIONS]
  -A | --csf_key <private key path>  CSF private key in PEM format
  -a | --csf_crt <public  key path>  CSF public  key in PEM format
  -B | --img_key <private key path>  IMG private key in PEM format
  -b | --img_crt <public  key path>  IMG public  key in PEM format
  -I | --table   <SRK table path>    Input SRK table (see usbarmory_srktool -O)
  -x | --index   <SRK key index>     Index for SRK key (1-4)
  -i | --image   <filename>          Image file w/ IVT header (e.g. u-boot.imx)
  -o | --output  <filename>          Write CSF to file
  -s | --serial                      Serial download mode
  -S | --dcd     <address>           Serial download DCD OCRAM address
     |                                (depends on mfg tool, default: 0x00910000)
     |
  -d | --debug                       Show additional debugging information
  -T | --hab_poc                     Apply HAB bypass PoC (CVE-2017-7932)
     |
  -h | --help                        Show this help
```

Publishing PoC code encourages further investigation and testing of issues among vendors or other affected parties; it promotes security research; and it empowers other skilled parties to further verify the scope and impact of vulnerabilities.

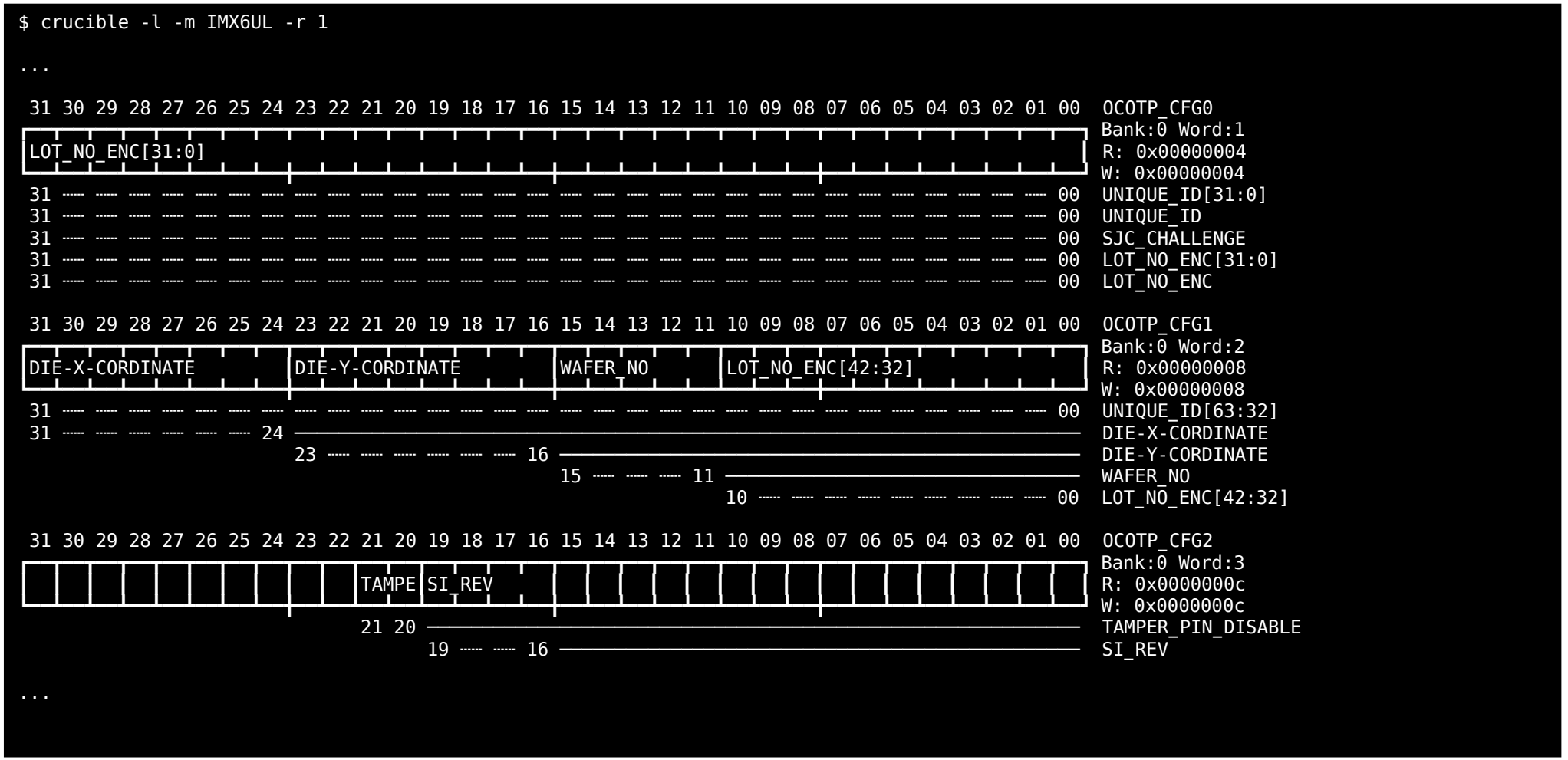
The most important and compelling reason to take this approach, however, is this: In scenarios where detailed technical information has already been made public, the lack of a working PoC does not, and should not, constitute any form of “protection.”

One-Time-Programmable (OTP) fuses

F-Secure

https://github.com/inversepath/crucible

OTP fuse management with crucible



crucible - i.MX6UL example

F-Secure

Blow a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 blow MAC1_ADDR 0x001f7b1007e3
IMX6UL ref:1 op:blow addr:0x88 off:0 len:48 val:0xe307107b1f000000
```

Read a fuse

```
$ sudo crucible -m IMX6UL -r 1 -b 16 read MAC1_ADDR
IMX6UL ref:1 op:read addr:0x88 off:0 len:48 val:0x001f7b1007e3
```

Read a fuse (minimal output for batch operations)

```
$ sudo crucible -s -m IMX6UL -r 1 -b 16 read MAC1_ADDR
001f7b1007e3
```

```
registers:
  OCOTP_LOCK:
    bank: 0
    word: 0
    fuses:
      TESTER_LOCK:
        offset: 0
        len: 2
      BOOT_CFG_LOCK:
        offset: 2
        Len: 2
...
  OCOTP_MAC0:
    bank: 4
    word: 2
    fuses:
      MAC1_ADDR:
        offset: 0
        len: 48
  OCOTP_MAC1:
    bank: 4
    word: 3
  OCOTP_MAC:
    bank: 4
    word: 4
```

crucible - i.MX6UL lock down

F-Secure

It is vital to reduce the low-level SoC attack surface as much as possible and ensure lock down of all relevant fuses.

```
# set device in Closed Configuration (IMX6ULRM Table 8-2, p245)
crucible -m IMX6UL -r 1 -b 2 -e big blow SEC_CONFIG 0b11

# disable NXP reserved mode (IMX6ULRM 8.2.6, p244)
crucible -m IMX6UL -r 1 -b 2 -e big blow DIR_BT_DIS 1

# Disable debugging features (IMX6ULRM Table 5-9, p216)
# * disable Secure JTAG controller
# * disable JTAG debug mode
# * disable HAB ability to enable JTAG
# * disable tracing
crucible -m IMX6UL -r 1 -b 2 -e big blow SJC_DISABLE 1
crucible -m IMX6UL -r 1 -b 2 -e big blow JTAG_SMODE 0b11
crucible -m IMX6UL -r 1 -b 2 -e big blow JTAG_HEO 1
crucible -m IMX6UL -r 1 -b 2 -e big blow KTE 1

# To further reduce the attack surface:
#  * disable Serial Download Protocol (SDP) READ_REGISTER command (IMX6ULRM 8.9.3, p310)
#  * disable SDP over UART (IMX6ULRM 8.9, p305)
crucible -m IMX6UL -r 1 -b 2 -e big blow SDP_READ_DISABLE 1
crucible -m IMX6UL -r 1 -b 2 -e big blow UART_SERIAL_DOWNLOAD_DISABLE 1
```

i.MX6UL - Security audit

F-Secure

Against Silicon Revision 1.2 and HAB 4.1 or greater, meaning P/Ns “AB” or greater, with patched HABv4.

Completed as an internal + third party security audit for HABv4 as well as our buildroot chain of trust implementation.

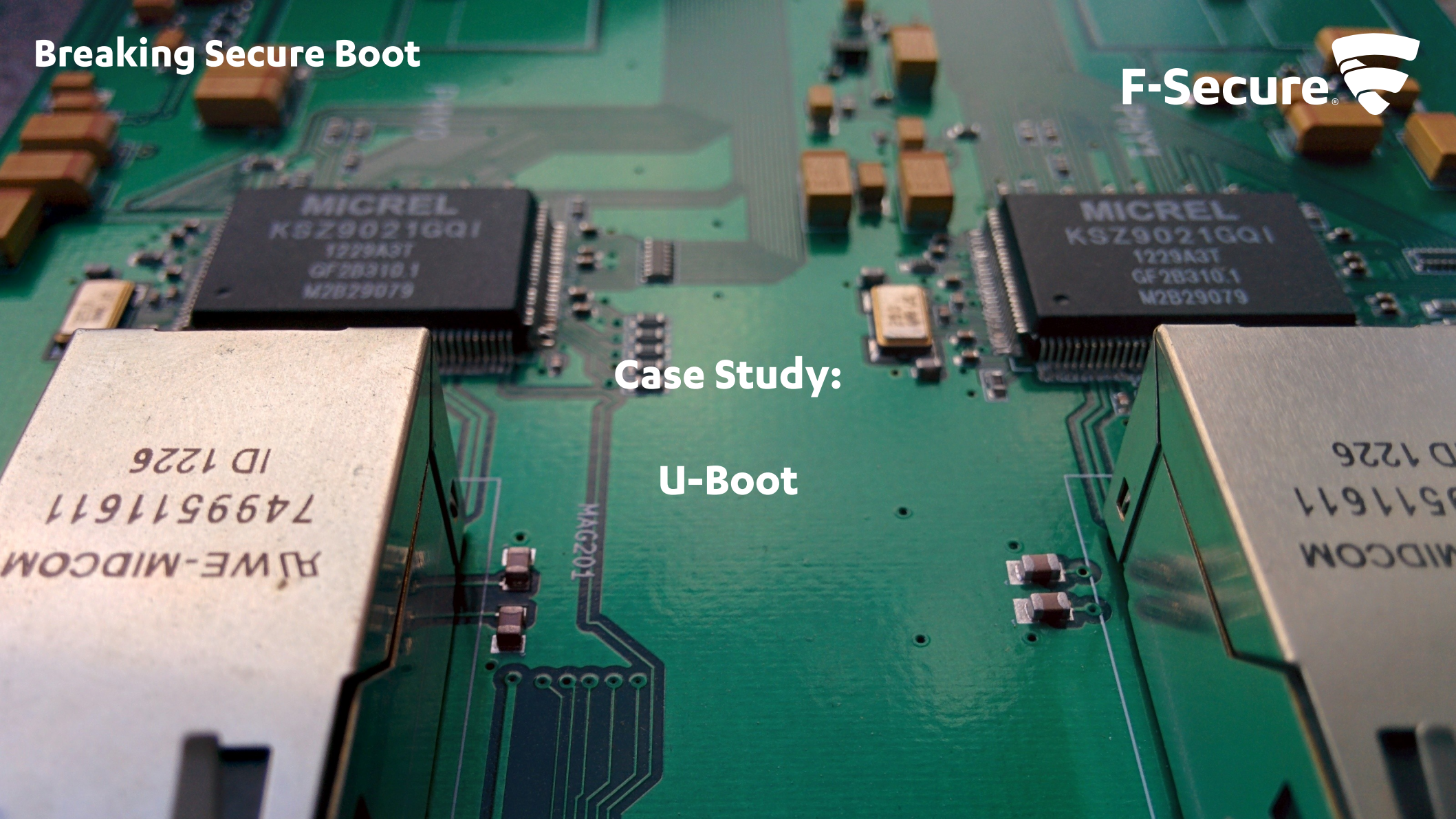
Conclusions

No further issues have been identified in the patched boot ROM.

Freescale/NXP kernel module issues (invalid error values, NULL pointer exceptions, various operational errors), all resolved in our own driver ports.

Several U-Boot issues identified and resolved (CVE-2018-18439, CVE-2018-18440)

F-Secure



Case Study:

U-Boot

U-Boot Verified Boot bypass

Multiple techniques allow execution of arbitrary code, bypassing secure boot and/or verified boot.

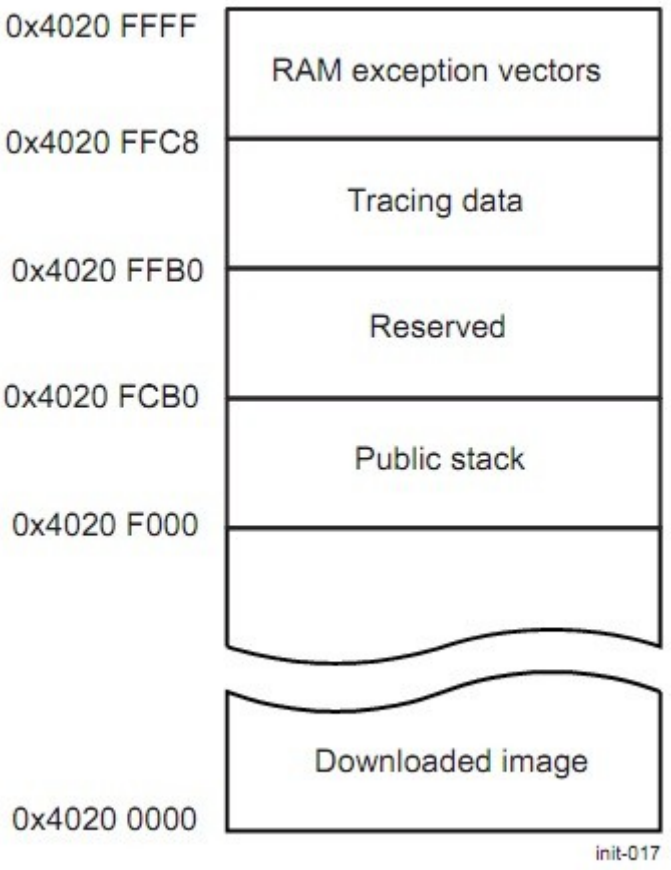
U-Boot lacks any automatic memory allocation protection in relation to its own code location in memory.

CVE-2018-18440

Lack of boundary checks in filesystem image load.

CVE-2018-18439

Lack of boundary checks in network image boot.



https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_IPVR2018-0001.txt

CVE-2018-18440

F-Secure

```
U-Boot 2018.09-rc1 (Oct 10 2018 - 10:52:54 +0200)

DRAM:  128 MiB
Flash: 128 MiB
MMC:   MMC: 0

=> bdinfo
arch_number = 0x000008E0
boot_params = 0x60002000
DRAM bank   = 0x00000000
-> start    = 0x60000000
-> size     = 0x08000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000004
eth0name    = smc911x-0
ethaddr     = 52:54:00:12:34:56
current eth = smc911x-0
ip_addr     = <NULL>
baudrate    = 38400 bps
TLB addr    = 0x67FF0000
relocaddr   = 0x67F96000
reloc off   = 0x07796000
irq_sp      = 0x67EF5EE0
sp start    = 0x67EF5ED0
```

```
=> ext2load mmc 0 0x60000000 fitimage.itb

(gdb) p gd
$28 = (volatile gd_t *) 0x67ef5ef8
(gdb) p *gd
$27 = {bd = 0x7f7f7f7f, flags = 2139062143, baudrate = 2139062143, ... }
(gdb) x/300x 0x67ef5ef8
0x67ef5ef8: 0x7f7f7f7f  0x7f7f7f7f  0x7f7f7f7f  0x7f7f7f7f
```

CVE-2018-18439

F-Secure

```
U-Boot 2018.09-rc1 (Oct 10 2018 - 10:52:54 +0200)

DRAM:  128 MiB
Flash: 128 MiB
MMC:   MMC: 0

=> bdinfo
arch_number = 0x000008E0
boot_params = 0x60002000
DRAM bank   = 0x00000000
-> start    = 0x60000000
-> size     = 0x08000000
DRAM bank   = 0x00000001
-> start    = 0x80000000
-> size     = 0x00000004
eth0name    = smc911x-0
ethaddr     = 52:54:00:12:34:56
current eth = smc911x-0
ip_addr     = <NULL>
baudrate    = 38400 bps
TLB addr    = 0x67FF0000
relocaddr   = 0x67F96000
reloc off   = 0x07796000
irq_sp      = 0x67EF5EE0
sp start    = 0x67EF5ED0
```

```
=> setenv loadaddr 0x60000000
=> dhcp
smc911x: MAC 52:54:00:12:34:56
smc911x: detected LAN9118 controller
smc911x: phy initialized
smc911x: MAC 52:54:00:12:34:56
BOOTP broadcast 1
DHCP client bound to address 10.0.0.20 (1022 ms)
Using smc911x-0 device
TFTP from server 10.0.0.1; our IP address is 10.0.0.20
Filename 'fitimage.bin'.
Load address: 0x60000000
Loading: ##################################################################
...
         ###################################

R00=7f7f7f7f R01=67fedf6e R02=00000000 R03=7f7f7f7f
R04=7f7f7f7f R05=7f7f7f7f R06=7f7f7f7f R07=7f7f7f7f
R08=7f7f7f7f R09=7f7f7f7f R10=0000d677 R11=67fef670
R12=00000000 R13=67ef5cd0 R14=02427f7f R15=7f7f7f7e
PSR=400001f3 -Z-- T S svc32
```

Workarounds (< v2019.04)

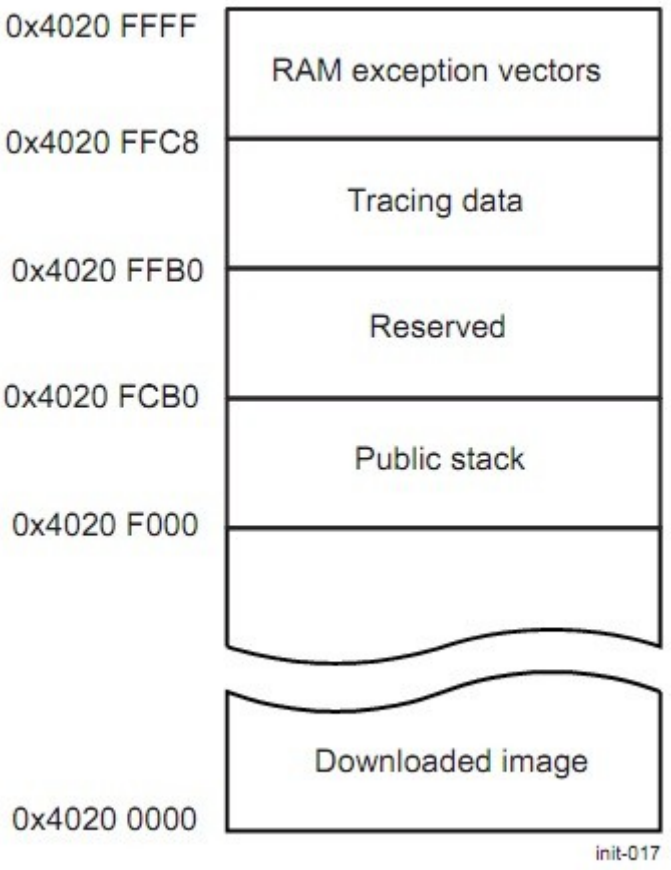
It must be emphasized these two cases only represent two possible occurrences of such architectural limitation, other U-Boot image loading functions are extremely likely to suffer from the same validation issues.

CVE-2018-18440

The optional bytes argument can be passed to all load commands to restrict the maximum size of retrieved data. A value consistent with memory regions mapping can be passed.

CVE-2018-18439

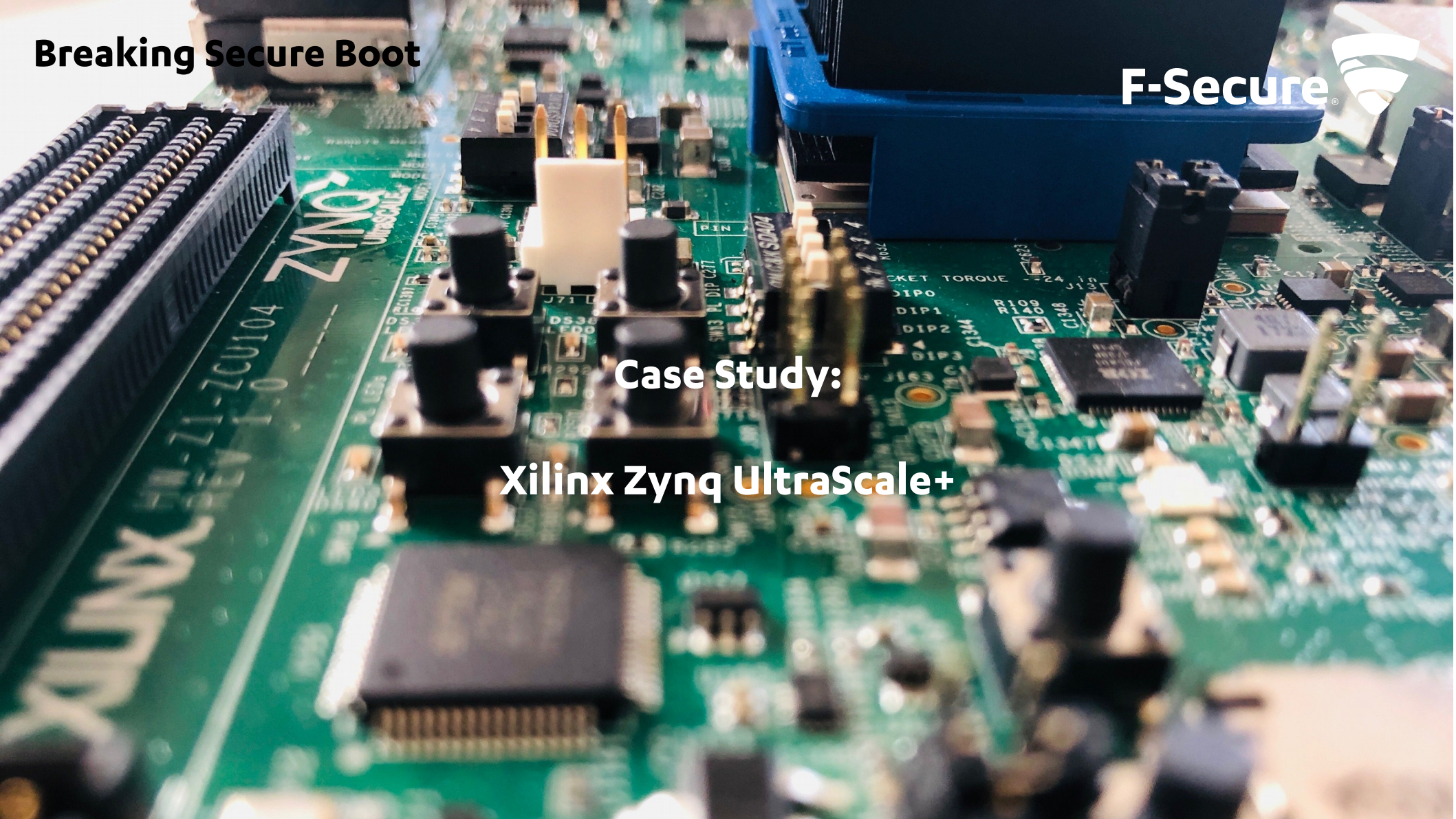
Disable all network loading commands.



https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_IPVR2018-0001.txt

Breaking Secure Boot

F-Secure



Case Study:

Xilinx Zynq UltraScale+

Xilinx Zynq UltraScale+ Secure Boot bypass

F-Secure

Two design flaws have been identified from documentation resources, and later verified on actual hardware (**CVE-2019-5478**).

The highlighted pointers **lack authentication** on **Encrypt Only Secure Boot**.

By design this metadata is not encrypted, therefore it cannot be authenticated as authentication is only carried through encryption in this mode.

Arbitrary code execution is achieved by tampering Images.

Table 11-4: **Boot Header Format**

Offset	Description	Details
0x000 – 0x01C	Reserved for interrupts	This field is used in case of XIP boot mode when the default 0x01F interrupt vectors are changed in the LQSPI address space.
0x020	Width detection	Quad-SPI width description.
0x024	Image identification	Boot image identification string.
0x028	Encryption status	This field is used to identify the AES key source. 0000_0000h: Unencrypted. 3A5C_3C5Ah: Red key in BBRAM. A35C_7CA5h: Obfuscated key in boot header. A35C_7C53h: Black key in boot header. A5C3_C5A3h: Red key in eFUSE. A5C3_C5A5h: Black key in eFUSE (PUF key). A5C3_C5A7h: eFUSE (family key). A3A5_C3C5h: User key. [illegible] ingle partition boot images.
0x02C	FSBL execution address	FSBL execution start address.
0x030	[illegible]	[illegible] ress.
0x034	PMU FW image length	PMU FW original image length.
0x038	Total PMU FW image length	PMU FW total image length. This includes the complete PMU firmware image block size, AES key, AES IV, and GCM tag (in case of an encrypted image). This field size must be ≤128 KB.
0x03C	FSBL image length	FSBL original image length.
0x040	Total FSBL image length	Total FSBL image length.
0x044	Image attributes	Image attributes are described in Table 11-5.
0x048	Header checksum	Header checksum from 0x20 to 0x44.
0x04C–0x068	Obfuscated key	256-bit obfuscated key. Only valid when 0x028 (encryption status) is A35C_7CA5h.
0x06C	Reserved	
0x070–0x09C	FSBL/User defined	How to use the FSBL/user defined areas is explained in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3].
0x0A0–0x0A8	Secure header initialization vector	Initialization vector for a secure header for both PMU FW and FSBL.

Table 16-8: **Partition Headers (Offsets, Parameters, and Description)**

Address Offset	Parameter	Description
0x00	Partition Data Word Length (x4)	Encrypted partition length.
0x04	Extracted Data Word Length (x4)	Unencrypted partition data length.
0x08	Total Partition Word Length (Includes Authentication Certificate) (x4)	The total encrypted + padding + expansion +authentication length.
0x0C	Next Partition Header Offset	Location of next partition header
0x10	Destination Execution Address LO	The executable address of this partition after loading.
0x14	Destination Execution Address HI	The executable address of this partition after loading.
0x18	Destination Load Address LO	The RAM address into which this partition is to be loaded.
0x1C	Destination Load Address HI	The RAM address into which this partition is to be loaded.
0x20	Actual Partition Word Offset	The position of the partition data relative to the start of the boot image.
0x24	Attributes	See Table 16-9.
0x28	Reserved	For internal use.
0x2C	Checksum Word Offset (x4)	The location of the checksum word in the boot image.
0x30	Reserved	For internal use.
0x34	Authentication Certificate Word Offset (x4)	The location of the Authentication Certification in the boot image.

https://github.com/inversepath/advisories/blob/master/Security_Advisory-Ref_FSC-HWSEC-VR2019-0001-Xilinx_ZU%2B-Encrypt_Only_Secure_Boot_bypass.txt

Xilinx Zynq UltraScale+ Secure Boot bypass

Before:

If there is a decryption failure, via the GCM-tag check, XilSecure passes the failure status to U-Boot. An important consideration of the encrypt only secure boot mode is that the partitions are not authenticated prior to decryption. The symmetric authentication occurs at the end of the decryption cycle. This means that the device is subject to a DPA random-data attack. Consequently, it is incumbent on the user to provide system-level protections if the DPA attack vector is a concern. For more information see DPA Resistance.

After:

There are two important considerations of the encrypt only secure boot mode that may require you to provide system-level protections. First, the partitions are not authenticated before decryption. The symmetric authentication occurs at the end of the decryption cycle. This means that the device is subject to a DPA random-data attack. Hence, you should provide system-level protections if the DPA attack vector is a concern. For more information, see DPA Resistance. Second, the boot and partition headers are not authenticated. Without authentication of these headers, anyone with access to the boot image, can modify the control fields resulting in incorrect secure boot behavior. One such example, is modification of the destination execution address. This address represents the start instruction address for a loaded partition. Anyone with access to the boot image could modify the address, causing the device to jump to an arbitrary memory location to modify or bypass the secure boot process. Hence, you should provide system-level protections if the lack of authentication of the boot and partition headers is a concern.

https://github.com/inversepath/advisories/blob/master/Security_Advisory-Ref_FSC-HWSEC-VR2019-0001-Xilinx_ZU%2B-Encrypt_Only_Secure_Boot_bypass.txt

KEY TAKEWAYS

F-Secure

It is vital to **account for all phases** of the **chain of trust** for design and implementation flaws.

The strongest hardware anchor can be compromised by the **smallest mistake**, whether hardware or software, throughout the chain of trust.

Low level **boot ROM** issues have **critical impact** due to their **lack of mitigation** in most cases, requiring P/N replacement (if available)...there are **no remote hardware upgrades**!

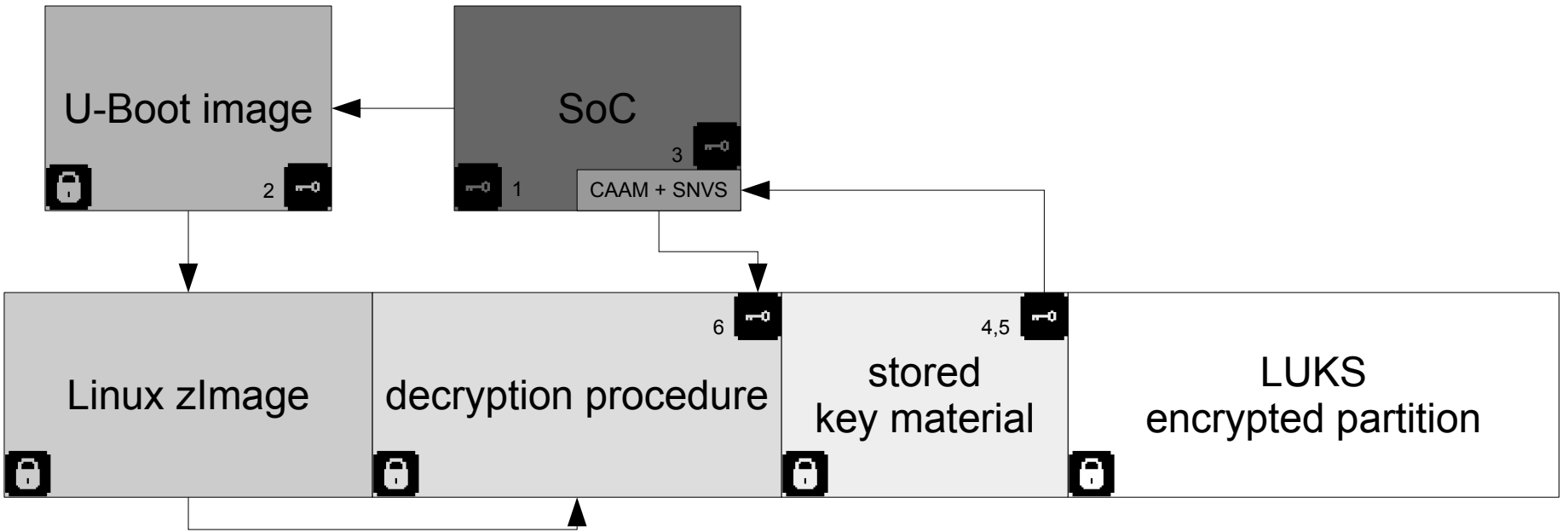
Public responsible vulnerability disclosure of hardware findings remains complicated with most customers and is most of the times not allowed. Such findings affect the entire community, lack of public findings does not mean that a P/N is immune.

Most SoC boot ROMs are either **vulnerable** or likely to suffer from critical Secure Boot findings.

Auditing secure boot software must go hand-in-hand with hardware auditing.

Understanding the convergence between software and hardware is paramount.

Reducing the attack surface

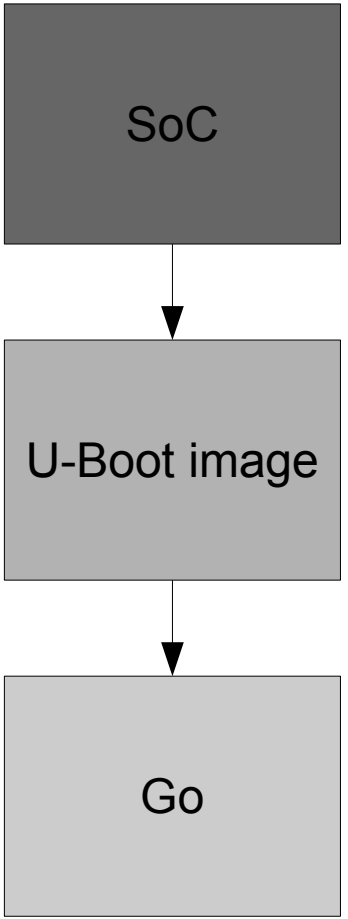


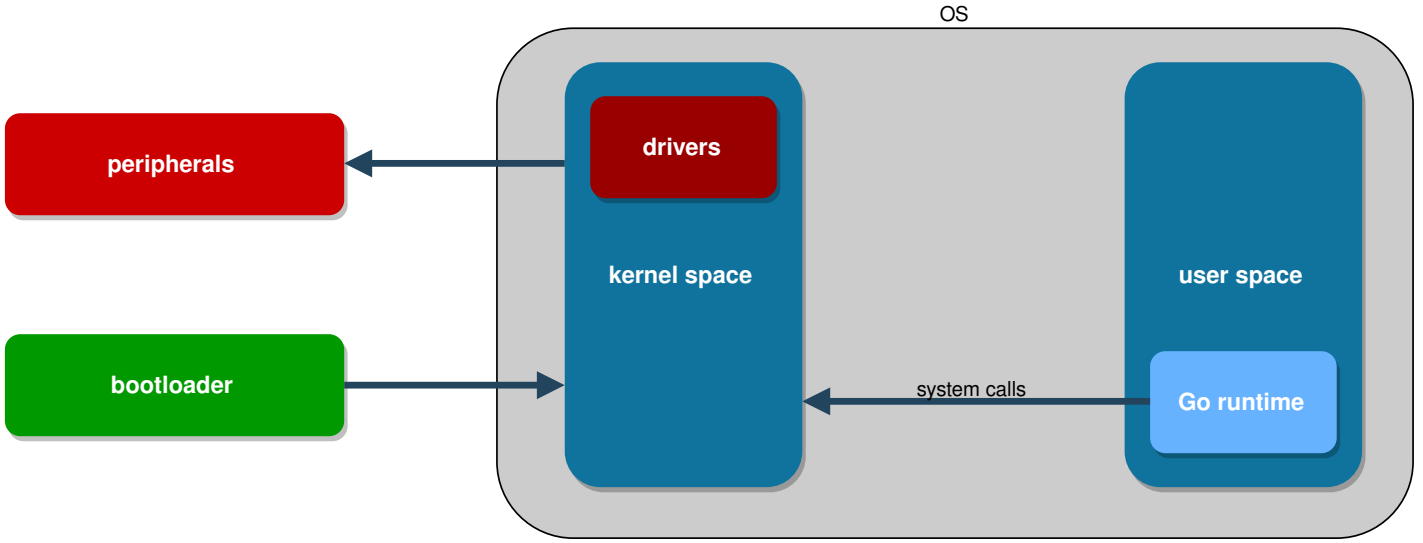
When building single purpose firmware images the attack surface, and maintenance burden, of the Linux kernel and runtime is considerable.

We introduce a development project that ultimately aims to reduce it as much as possible.

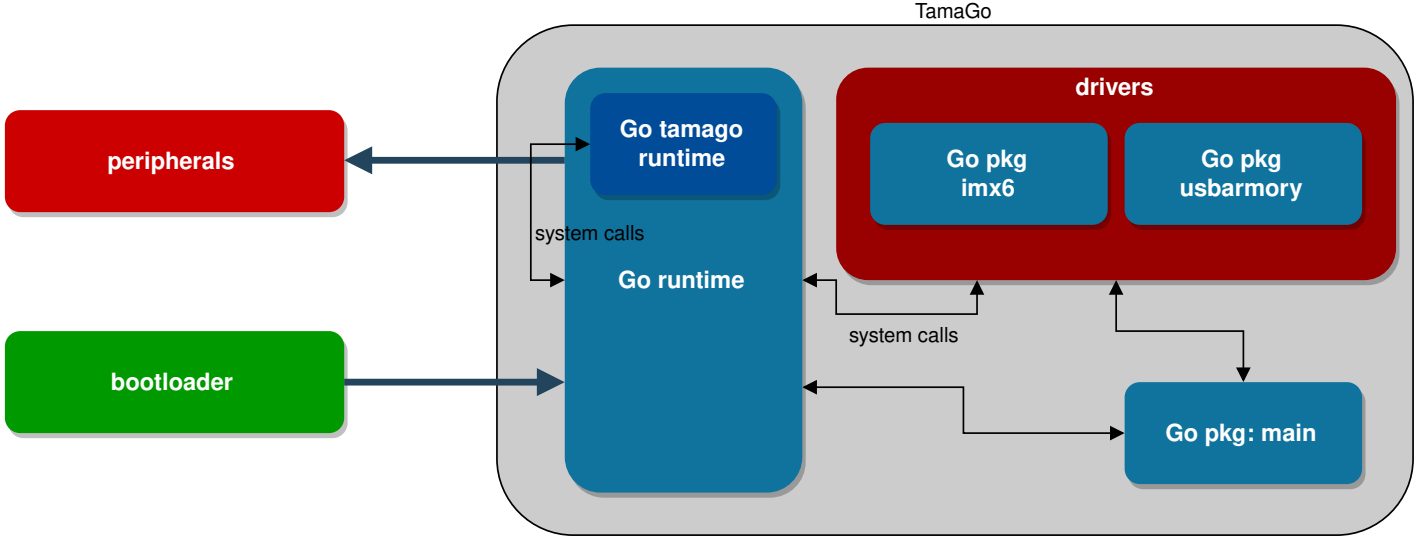
Reducing the attack surface with TamaGo

```
vec_start 0x81000000
vec_size 0x40
bootstack start 0x9ff00000
exception stack_start 0x9ff20000
exception stack_size 0x10000
g0.stackguard0 0x9fef0000
g0.stackguard1 0x9fef0000
g0.stack.lo 0x9fef0000
g0.stack.hi 0x9ff00000
mmu_map: 0x80000000 0x20000000
------------------------------------------------------------------------
imx6_rng: self-test...done
imx6_rng: seeding...done
imx6_soc: i.MX6ULL (0x65, 0.1) @ freq:900 MHz - native:true
Hello from tamago/arm! (epoch 3038767125)
launched 6 test goroutines
-- i.mx6 dcp -----------------------------------------------------------
imx6_dcp: derived SNVS key 7d5fec1ee57ab31e4ca75a59c568ffb6
-- file ----------------------------------------------------------------
read /tamago-test/tamago.txt (22 bytes)
-- sleep ---------------------------------------------------------------
sleeping 100ms @ 72238750
   slept 100ms @ 174514000
-- rng -----------------------------------------------------------------
   51682759bfbeee914df905b69d37a3504f94ab2d40dbdd3f5a51e1c7c061a285
-- ecdsa ---------------------------------------------------------------
ECDSA sign and verify with p224 ... done (114.507375ms)
ECDSA sign and verify with p256 ... done (47.94475ms)
-- btc -----------------------------------------------------------------
Script Hex: 76a914128004ff2fcaf13b2b91eb654b1dc2b674f7ec6188ac
Script Disassembly: OP_DUP OP_HASH160 128004ff2fcaf13b2b91eb654b1dc2b674f7ec61 OP_EQUALVERIFY OP_CHECKSIG
Script Class: pubkeyhash
Addresses: [12gpXQVcCL2qhTNQgyLVdCFG2Qs2px98nV]
Required Signatures: 1
Transaction successfully signed
------------------------------------------------------------------------
completed 6 goroutines
Goodbye from tamago/arm (5.7927555s)
exit with code 0 halting
```




OS
peripherals
bootloader
drivers
kernel space
user space
system calls
Go runtime

F-Secure


TamaGo
peripherals
bootloader
Go tamago runtime
system calls
Go runtime
drivers
Go pkg imx6
Go pkg usbarmory
system calls
Go pkg: main

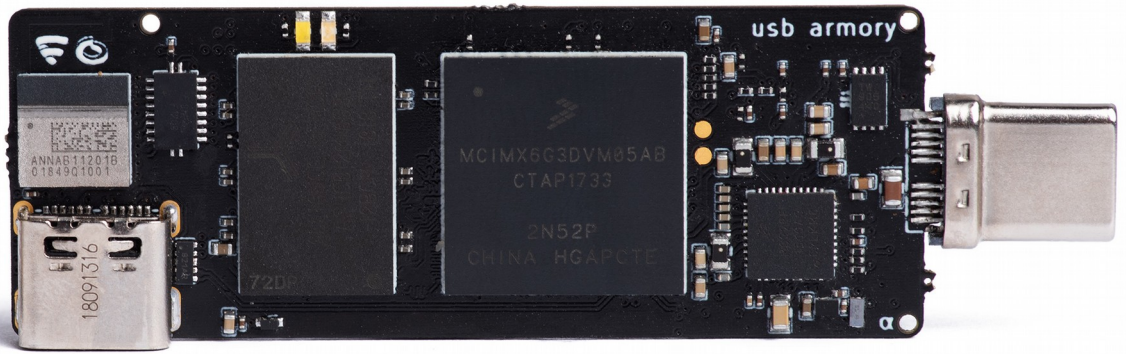
F-Secure

```
// Sample hardware key derivation on USB armory Mk II with TamaGo
//
// +build tamago,arm

package main

import (
	"imx6"
	// required to initialize board
	_ "usbarmory/mark-two"
)

func main() {
	key, err := imx6.DCP.DeriveKey(diversifier, iv)

	// do stuff...
	...
}
```

https://github.com/inversepath/tamago/wiki

どうもありがとうございます

Questions?



Andrea Barisani
Head of Hardware Security
andrea.barisani@f-secure.com

https://github.com/inversepath/advisories
https://www.f-secure.com/documents/10192/2157381/fsecure-hardware-security-services-overview.pdf