SOCIAL

atom feed
twitter
github

CATEGORIES

Android
Challenge
Cryptography
Development
Exploitation
Fuzzing
Hardware
Hardware, ReverseEngineering
Life at Quarkslab
Maths
PenTest
Program Analysis
Programming
ReverseEngineering
Software
Vulnerability

TAGS

Vulnerabilities in High Assurance Boot of NXP i.MX microprocessors

Date Tue 18 July 2017 By Guillaume Delugré Iván Arce Category Vulnerability. Tags vulnerability bootloader secure boot i.mx NXP ASN.1

This blog post provides details about two vulnerabilities found by Quarkslab's researchers Guillaume Delugré and Kévin Szkudłapski in the secure boot feature of the i.MX family of application processors [1] built by NXP Semiconductors.

The bugs allow an attacker to subvert the secure boot process to bypass code signature verification and load and execute arbitrary code on i.MX application processors that have the High Assurance Boot feature enabled. These bugs affect 12 i.MX processor families.

The vulnerabilities were discovered and reported to the vendor in September 2016 and the technical details included in this blogpost were disclosed in a joint Quarkslab-NXP presentation at the Qualcomm Mobile Security Summit 2017 [2] in May 19th, 2017. National computer emergency response teams (CERTs) from 4 countries were informed about the issues in March, 2017.

NXP has issued an Engineering Bulletin and two Errata documents (*EB00854*, *ERR010872* and *ERR0108873* respectively) [3] providing a brief description of both vulnerabilities, the list of affected processor models along with resolution plans and possible mitigations.

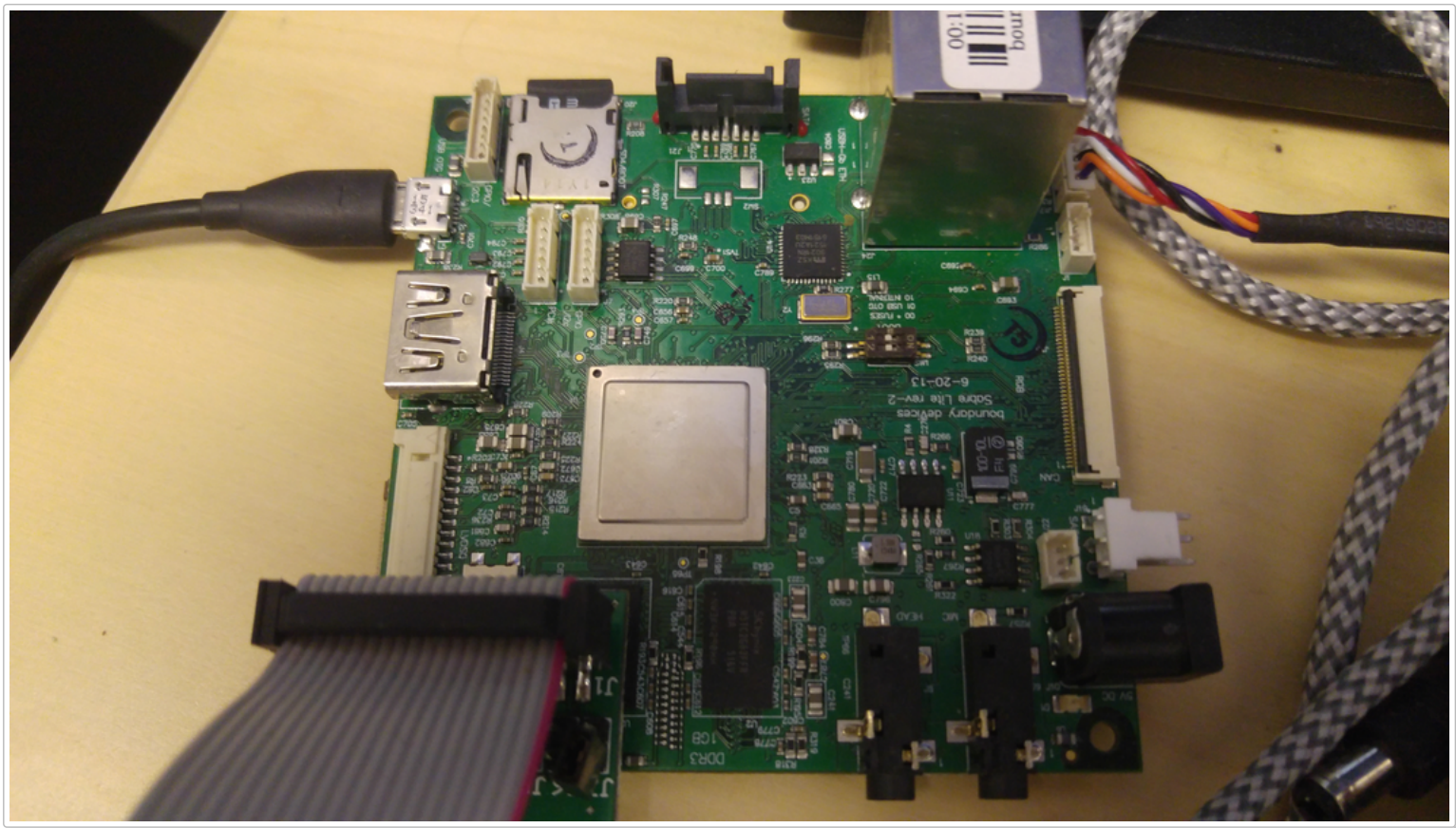
In the rest of the blogpost we describe the relevant features in i.MX processors and the vulnerabilities affecting them.

Introduction

The i.MX family of devices built by NXP Semiconductors are ARM-based application processors present in System-on-a-Chip (SoC) systems used in the automotive, industrial and consumer electronics markets. The microcontrollers, originally developed by Freescale Semiconductor which was acquired by NXP in 2015, provide a rich set of security oriented features such as secure and encrypted boot, tamper detection, hardware acceleration for various cryptographic algorithms, on and off chip secure storage, secure real-time clock and a hardware-based random number generator.

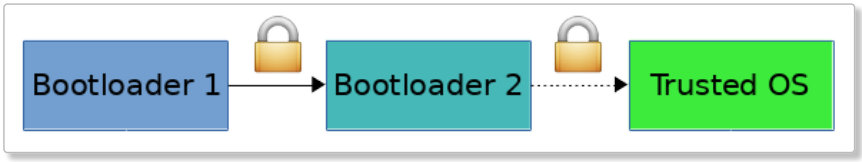
These features have to be enabled by vendors that use i.MX processors in their products and in some cases the processors are configured in a *locked* state in which the security features cannot be disabled.

There are several development boards that interested readers may use to experiment with the security features of i.MX processors. The vulnerabilities described here were discovered using a SabreLite [4] i.MX6-based development board like this:



What is secure boot?

Secure boot is the process by which a system turned off is brought up to a **known good state** when it is turned on. The process generally involves the chained execution of native code packaged in multiple binary files that are loaded, verified to be authentic and not tampered with, and executed in a sequence that ends with the system ready to run as intended by the manufacturer or user. This is generally accomplished by using digitally signed binaries that upon execution check the integrity and authenticity of the next binary to be loaded and run, and transfers control to it if the verification did not fail.



If the system is turned on in a *known good state* and, the sequence of binaries executed upon boot is verified to have valid signatures, then the resulting state after the last binary of the sequence runs is considered to be good and *trusted*. However, if a single link in the chain of binaries executed in the secure boot process fails to perform proper validation of the next binary in the sequence the resulting state of the system can no longer be trusted.

Usually the root of this chain of trust is built upon cryptographic keying material (signing and encryption keys) and a trusted initial firmware (bootrom) stored in *on-chip read only memory*

Systems that have a secure boot feature enabled and locked in a way that can not be disabled prevent tampering by end users, downstream vendors, integrators or attackers with physical access to the device. The feature is generally used to enforce digital rights management (DRM) and intellectual property protection requirements or to prevent execution of malicious or otherwise unauthorized software on the device.

High Assurance Boot (HAB)

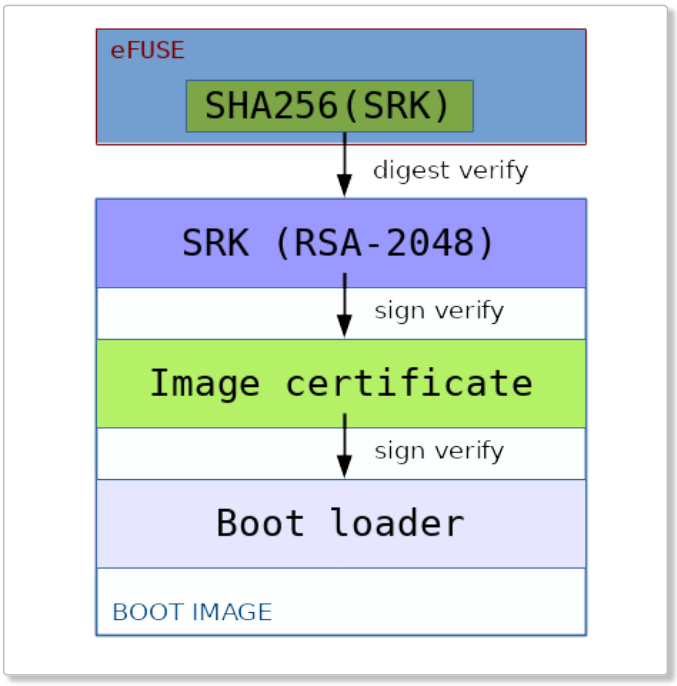
High Assurance Boot (HAB) is NXP's implementation of secure boot in i.MX processors. This is a capability built into on-chip ROM responsible for loading the initial program image, usually the first stage bootloader, from the boot medium.

HAB uses public key cryptography, specifically RSA keys, to authenticate the image executed at boot time. Image data is signed offline by the image provider using private keys and the i.MX processor verifies the signature using the corresponding public keys, which are loaded from a section of the binary to be verified.

The root of the trust chain is anchored on a set of RSA key pairs called *Super Root Keys* (SRKs) used to prevent loading and use of arbitrary public keys provided by potential attackers trying to run image data signed by themselves with the corresponding private keys. It also optimizes the use of scarce one-time-programmable hardware resources. The private keys are held by the Certification Authority issuing keys used to sign image data and a table of cryptographic hashes of the corresponding public keys is stored in one-time programmable hardware. In the case of an i.MX processor that supports secure boot, this is a masked ROM and electrically programmable fuses (eFuses).

Upon booting HAB bootrom loads the SRK table from the image's `Command Sequence File (CSF)` section, calculates the hash value and compares it to the value stored in the SRK fuses. If the value matches, the secure boot process continues by verifying that the image is properly signed with the proper key. A failure or error in the boot process may make the processor enter recovery mode, in which a boot image can be securely loaded from a UART or USB port using the Serial Download Protocol (SDP) described later on.

The public keys are provisioned in X.509 certificates using a Code Signing Tool (CST) provided by the vendor. The HAB functionality is exposed to boot image code through an API than can be called by commands in the Command Sequence File. The CSF contains all the commands that HAB executes during the secure boot, the SRK table and the certificates and signatures to validate the image to be loaded and run.



The High Assurance Boot trust chain.

The entire process is described in detail in NXP's "Secure Boot on i.MX50, i.MX53, and i.MX 6 Series using HABv4" document (AN4581) [5]

Serial Download Protocol (SDP)

The bootrom supports a recovery mode called SDP (*Serial Download Protocol*) which enables the loading of a boot image from a USB or UART port.

To achieve this, the bootrom implements a small USB protocol over HID with some simple commands:

- Reading a word from memory (`READ REGISTER`)
- Writing a word to memory (`WRITE REGISTER`)
- Writing a boot image to memory (`WRITE FILE`)
- Writing and executing a set of DCD commands (`DCD WRITE`)
- Executing a boot image loaded in memory (`JUMP ADDRESS`)

When the device has been locked in secure mode, multiple checks are done to protect the bootrom against unallowed memory accesses and unsigned code execution:

- Memory accesses are checked against a whitelist of allowed memory ranges
- DCD memory accesses are checked against the same whitelist
- The `JUMP ADDRESS` will check the boot image signature before executing it.

The Vulnerabilities

The vulnerabilities were discovered by analyzing a compiled bootrom image.

The names used to identify functions in the code have been assigned arbitrarily based on understanding the code's functionality and do not reflect the real function names used in the original source code.

The addresses used to describe location of functions are related to a bootrom image with the following MD5 digest: `945cfc9f525a378102dd24b5eb1b41cf`.

Each vulnerability described in this document has been successfully confirmed using a functional exploit bypassing the HAB secure boot on a locked *Sabrelite* development board.

InversePath, vendor of USB Armory [6], an affected device confirmed the vulnerabilities and developed proof of concept programs to demonstrate them.

Stack-based buffer overflow in the X.509 certificate parser (*CVE-2017-7932*)

The X.509 certificate parser of the bootrom is vulnerable to a stack-based buffer overflow that can be exploited by loading a crafted certificate.

The control flow of the secure boot follows these steps:

1. Fetch the Interrupt Vector Table (IVT) from the storage device, or over USB in recovery mode
2. Execute the DCD commands
3. Execute the CSF commands, which are responsible for the secure boot integrity. When the device is in trusted mode, the CSF commands from step 3 above are usually like this:
    1. Install a RSA public key of type SRK, whose SHA256 digest must be the same as the one written in the SRK fuses
    2. Install a CSFK public key using a X.509 certificate signed by the SRK
    3. Authenticate the CSF using the CSFK
    4. Install a public key to authenticate the boot image
    5. Authenticate the boot image using the previously installed key
4. Execute the next bootloader stage

Key installation is done using the `INSTALL_KEY` CSF command. The different keys are loaded and verified, each of them being verified by a previously installed key (the root of trust being the SRK fuses).

When installing a public key of type X.509, the function `hab_csf_cmd_install_key` (at address `0xB5C0`) will locate the plugin responsible for importing X.509 certificates and call in a row:

- `mod_x509->load_key` (`x509_parse_certificate`)
- `mod_x509->verify_key` (`x509_verify_certificate_signature`)

Hence, the entire certificate is **parsed before having its signature verified**. As the `INSTALL_KEY` command can be executed before any authenticated command, a vulnerability in the X.509 parser could be triggered by an attacker with no pre-requisite other than being able to tamper with the boot image or switch the device into recovery mode.

The HAB bootrom uses its own ASN.1 and X.509 parsing library and a vulnerability exists due to an incorrect call to an ASN.1 library function when parsing X.509 certificate extensions.

The function `asn1_extract_bit_string` (at address `0xE796`) can be used to copy the contents of a *bit string* object using the internal function `asn1_deep_copy_element_data` (at `0xEF20`).

This method is called recursively to copy the contents of an ASN.1 object into an output buffer passed as an argument. One peculiar characteristic of this function is that it doesn't have an input argument that holds the value of the output buffer. Instead, the required size is returned to the caller when a `NULL` pointer is passed as the output buffer, a behavior quite similar to the Windows API.

The `asn1_extract_bit_string` function should be used by first callinf in with a `NULL` argument, allocating the required memory with `malloc` and calling the function again with the newly allocated buffer.

The X.509 specification describes the `keyUsage` extension as begin a *bit string* object, composed of the following 9 bits:

```
KeyUsage ::= BIT STRING {
  digitalSignature        (0),
  nonRepudiation          (1),
  keyEncipherment         (2),
  dataEncipherment        (3),
  keyAgreement            (4),
  keyCertSign             (5),
  cRLSign                 (6),
  encipherOnly            (7),
  decipherOnly            (8) }
```

When the certificate extensions are parsed, if a field `keyUsage` is present, the function `asn1_extract_bit_string` is directly called by specifying a pointer to a 32-bits integer on the stack as the output buffer.

A *bit string* object with a size greater than 4 bytes in a `keyUsage` extension will then trigger a stack buffer overflow.

When the function `asn1_extract_bit_string` returns, the bootrom code verifies that the returned size is not greater than 4 bytes, but the copy operation has already been done. The calling function then returns and pops its return address off the stack, which allows an attacker to redirect the PC register and execute arbitrary code.

The attacker controls the size and the contents of the data being written on the stack (size and contents of the bit string contained in the certificate).

```
x509_parse_next_extension               ; CODE XREF: x509_parse_contents+166↓

unused_bits     = -0x30
bool_value      = -0x28
size            = -0x24
critical        = -0x20
_size           = -0x1C

                PUSH.W          {R4-R8,LR}
                MOVS            R7, #0
                SUB             SP, SP, #0x18
                MOV             R6, R2
                MOV             R2, R1  ; buffer_end
                MOV             R1, R0  ; buffer_start
                STR             R7, [SP,#0x30+critical]
                MOVS            R0, #ASN1_TYPE_SEQUENCE ; tag
                MOV             R8, R7
                ADD             R3, SP, #0x30+_size ; size
                BL              _asn1_basic_element_data_size
                LDR             R1, [SP,#0x30+_size]
                MOVS            R4, R0  ; address of element
                ADD.W           R5, R0, R1
                BEQ             loc_E39E
                LDR             R2, =dword_10FC4
                ADD             R3, SP, #0x30+critical
                STR             R3, [SP,#0x30+unused_bits]
                MOVS            R3, #3
                SUBS            R2, #0x44 ; 'D' ; oid_keyUsage
                MOV             R1, R5
                MOV             R0, R4
                BL              x509_get_extension
                CMP             R0, #0
                BEQ             loc_E3A0
                CMP             R0, R4
                BEQ             loc_E2F0
                CMP             R0, R5
                BEQ             loc_E2F0
                ADD             R3, SP, #0x30+size
                MOV             R1, R0  ; buffer_start
                STR.W           R8, [SP,#0x30+bool_value]
                STR             R3, [SP,#0x30+unused_bits] ; bitstr_sz
                MOVS            R4, #0
                MOVS            R0, #ASN1_TYPE_BIT_STRING ; tag
                ADD             R3, SP, #0x30+bool_value ; output
                MOV             R2, R5  ; buffer_end
                BL              asn1_extract_bit_string ; BUFFER OVERFLOW
                ADDS            R1, R0, #1
                BEQ             loc_E3D2
```

Vulnerable code that parses the X.509 certificate extensions.

Buffer overflow in SDP recovery mode (*CVE-2017-7936*)

Memory checks present in the `WRITE FILE` command handler are incorrect and enable an attacker to achieve arbitrary code execution.

The Serial Download Protocol (SDP) uses the following header for USB packets:

- `type` (command type) ;
- `address` (memory address to read from or write to) ;
- `format` (data size in bits for `READ REGISTER` and `WRITE REGISTER`) ;
- `data_count` (data size in bytes for `WRITE FILE` and `DCD WRITE`) ;
- `data` (value used for `WRITE REGISTER`).

During the handling of commands `DCD WRITE` and `WRITE FILE` in trusted mode the memory range comprised of `address` and `address + data_count` is checked by the `hab_check_target` method (at address `0x753C`).

This method takes 3 arguments:

1. The type of memory (`0x0F` for memory, `0xF0` for hardware registers, `0x55` for any kind of memory)
2. The base address of the memory to be accessed
3. The size of the memory range in **bytes**

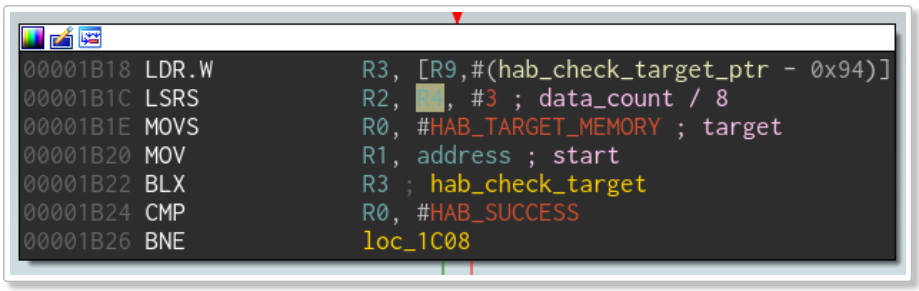
Depending on the kind of memory accessed, the memory range is checked against a whitelist of safe memory ranges.

However the method is not called correctly because the size argument (from the `data_count` field) is divided by 8 as if it were expressed in bits. The `data_count` field is actually expressed in bytes, which consequently leads to checking only a small subrange of the memory to be accessed.

This bug is possibly caused by a confusion with the `READ/WRITE REGISTER` commands that use the `format` field expressed in bits.

As a consequence, the bootrom checks that `[ address : address + data_count / 8 ]` is correct, whereas the data is actually copied to `[ address, address + data_count ]`.

This bug leads to a buffer overflow when the data is sent by the host over USB.



The memory size passed into R2 is expressed in bits instead of bytes.

Disclosure timeline

- **Sep 1 2016** : Vulnerabilities reported to NXP
- **Sep 2 2016** : NXP acknowledges receiving the report
- **Sep 8 2016** : NXP request for proof-of-concept of X.509 vulnerability
- **Sep 9 2016** : Exploit targeting i.MX6 bootrom X.509 overflow sent to NXP
- **Sep 16 2016** : NXP request for proof-of-concept of SDP vulnerability
- **Sep 16 2016** : Exploit targeting i.MX6 USB recovery mode sent to NXP
- **Feb 20 2017** : Abstract with a proposal for a Quarkslab presentation submitted to the Qualcomm Mobile Security Summit 2017
- **Mar 11 2017** : Qualcomm asks if Quarkslab is willing to do a joint presentation with NXP. The answer is yes.
- **Mar 21 2017** : Abstract of the QMSS 2017 presentations published disclosing the existence of secure boot bugs in i.MX processors.
- **Mar 22 2017** : InversePath informs Quarkslab of the partial disclosure of the bugs
- **Mar 22 2017** : Mail to NXP PSIRT, CERT-FR (France), CERT-NL (Netherlands), CERT-Bund (Germany) and ICS-CERT (USofA) notifying them of the vulnerabilities and the partial disclosure
- **Mar 23 2017** : InversePath confirmed the x.509 vulnerability in i.MX53 and published a draft security advisory [7].
- **Mar 23 2017** : Abstract of presentation at QMSS 2017 removed from website for corrections
- **Mar 24 2017** : InversePath pulled down its security advisory
- **Mar 23 2017** : NXP sends a corrected abstract that doesnt mention i.MX processors to QMSS
- **Mar 24 2017** : The corrected abstract is published on the QMSS 2017 website
- **Mar 24 2017** : Quarkslab mails InversePath, CERTs and NXP to coordinate future disclosure steps. Informs all parties its plan to publish details in a blogpost after the QMSS presentation on May 19th but may do so ealier if the issues become public.
- **May 1 2017** : ICS-CERT asks if Quakslab is willing to delay publication for 60 days to accomodate dissemmination of an NXP advisory through its Homeland Security Informaiton Network
- **May 2 2017** : Quarkslab responds that it sees no benefit in a 60-day embargo on public information about issues disclosed to a private list with thousands of suscribers. Asks ICS-CERT what is the intended effect of the embargo.
- **May 5 2017** : ICS-CERT responds that NXP would like to give customers time to mitigate affected products while disclosure of the vulnerabilities is more controlled.

- **May 10 2017** : Quarkslab tells ICS-CERT that it considers enforcing a 60-day embargo detrimental to the goal of helping vulnerable organizations mitigate risk and it plans to publish technical details in a blog post within a week of the joint presentation at QMSS 2017 scheduled for May 19th.
- **May 19 2017** : Quarkslab and NXP present at QMSS 2017
- **May 19 2017** : In a face to face meeting at QMSS 2017, NXP asks Quarkslab for a 60-day embargo on publication of technical details about the bugs. NXP says customers have asked for such delay and informs Quarkslab that it had published a notification about the bugs to all customers, that errata will be available for all affected i.MX products. Quarkslab asks NXP to provide errata documents and evidence of the notification to customer to evalute the request. Also asks to forward to Quarkslab any request for a 60-day embargo from an NXP customer.
- **May 22 2017** : NXP says that it published a notification to customers on April 27th, informed the Auto ISAC organization with a security bulletin on April 21st, helped ICS CERT to craft bulletin for its Homeland security Information Network. It provides a screenshot of customer notificationa, and the engineering bulletin and errata documents mentioned in this blog post.
- **May 30 2017** : After reviewing all documents, Quarkslab agrees to postpone this blogpost until July 18th 2017 and informs all parties (CERTs, NXP and InversePath) of the decision.
- **Jun 5 2017** : InversePath published the preliminary advisory originally published in March 23rd.
- **Jun 6 2017** : ICS-CERT provide the CVE IDs assigned to identify the vulnerabilities
- **Jul 19 2017** : Blog article posted. C'est la fin.

Acknowledgements

- Quarkslab colleagues for proofreading this article and for their feedback.

References

[1] http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/i.mx-applications-processors:IMX_HOME
[2] https://qct-qualcomm.secure.force.com/QCTConference/GenericSitePage?eventname=2017Security&page=Summit%20Information
[3] http://www.nxp.com/support/support/documentation:DOCUMENTATION
[4] https://boundarydevices.com/product/sabre-lite-imx6-sbc/
[5] http://www.nxp.com/docs/en/application-note/AN4581.pdf
[6] https://inversepath.com/usbarmory
[7] https://github.com/inversepath/usbarmory/blob/master/software/secure_boot/Security_Advisory-Ref_QBVR2017-0001.txt

Comments

Bob Sun • 2 years ago

Thanks for your research. Where can I get the binary of boot ROM of iMX to dissemble and study? I saw some disassembled code in your blog pointing out the vulnerabilities.

Reply • Share ›

Matthijs van Duin • 2 years ago

Excellent demonstration that complexity is the enemy of security. The ROM code that implements the root of trust should be absolutely as minimal as possible and very thoroughly analyzed. It should definitely not include something as horrible as an X.509 certificate parser. (It is in fact quite possible to avoid public-key crypto entirely in the ROM code, pushing it to the second stage instead, which can be replaced if crippling bugs are found without having to produce new silicon.)

Also, "secure boot" is a mistake in general, since it puts way too much code within the trust boundary. Allowing a measured launch environment to be established after untrusted boot (TCG calls this a "dynamic root of trust") would push all boot/recovery code out of the TCB and have prevented the second vulnerability from having any impact.

Reply • Share ›

J. Rachs • 2 years ago

Will NXP release a new sillicon revision of the component that fixes the vulnerabilities ? There is a mention of a "how they were able to put in place two fixes in the bootrom code" in the Qualcomm conference information page.

Reply • Share ›

Iván Arce ➚ J. Rachs • 2 years ago

NXP indicated that they will release new silicon for affected