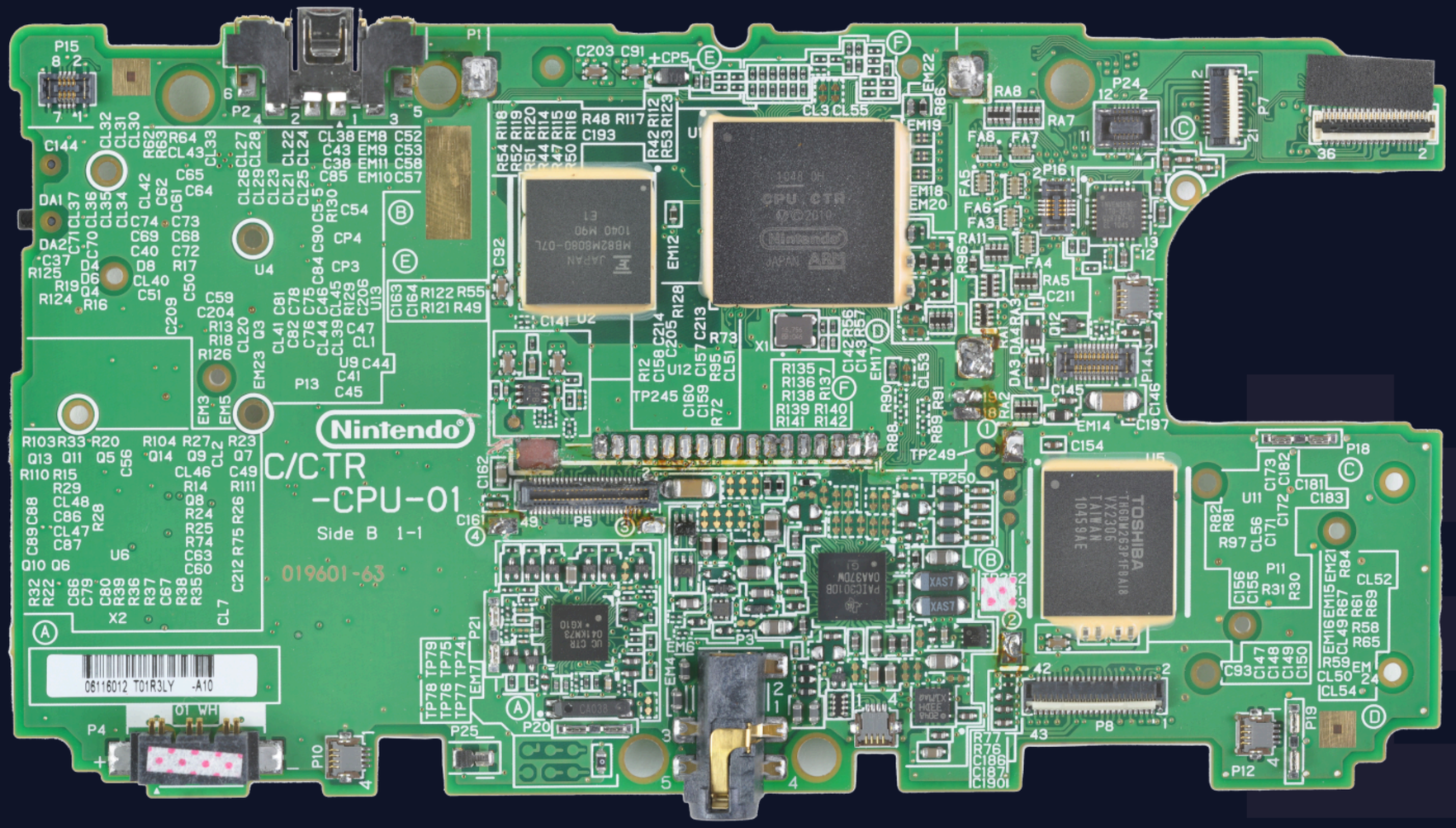
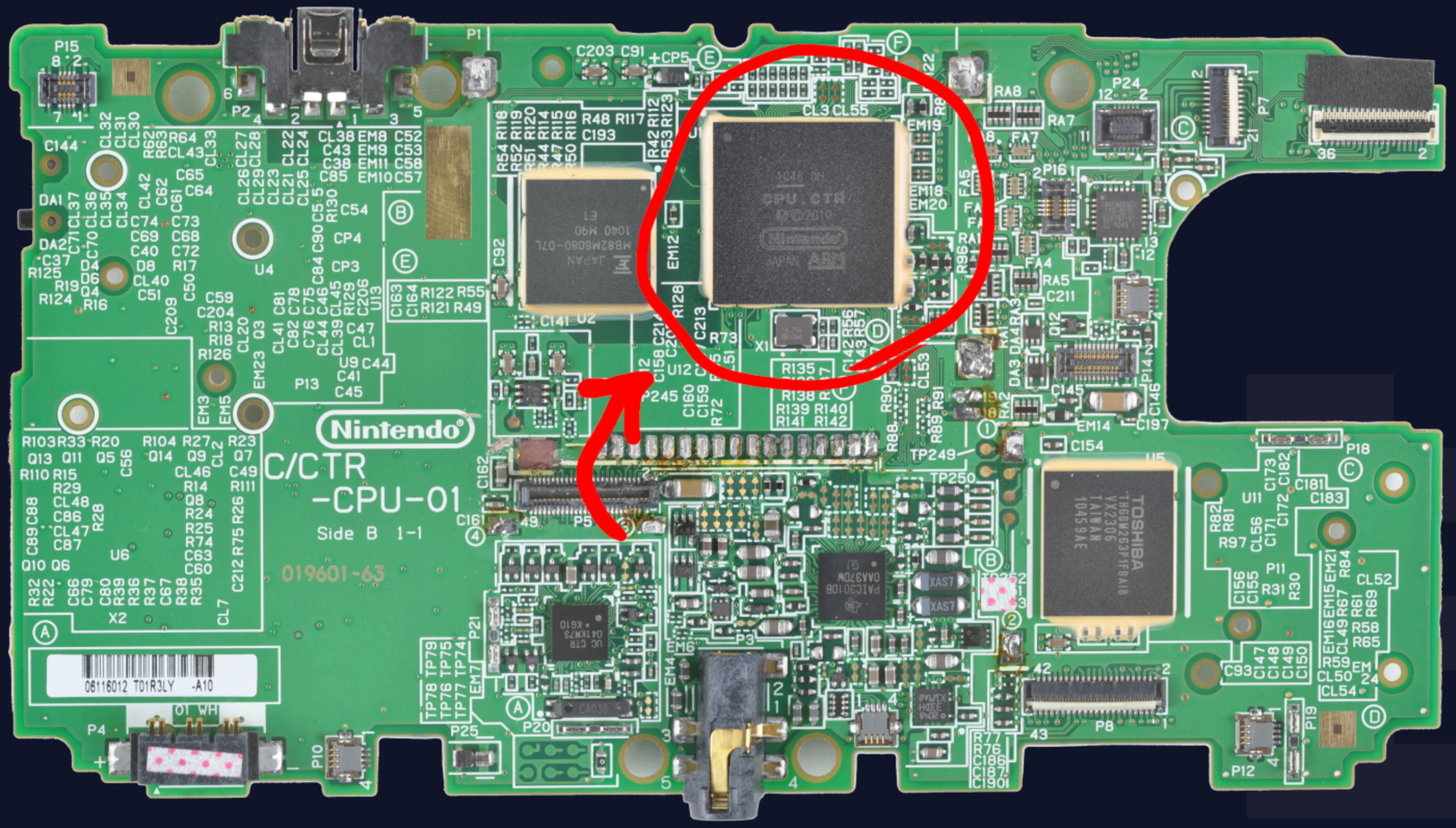
Sighax deep dive: breaking the 3DS chain of trust

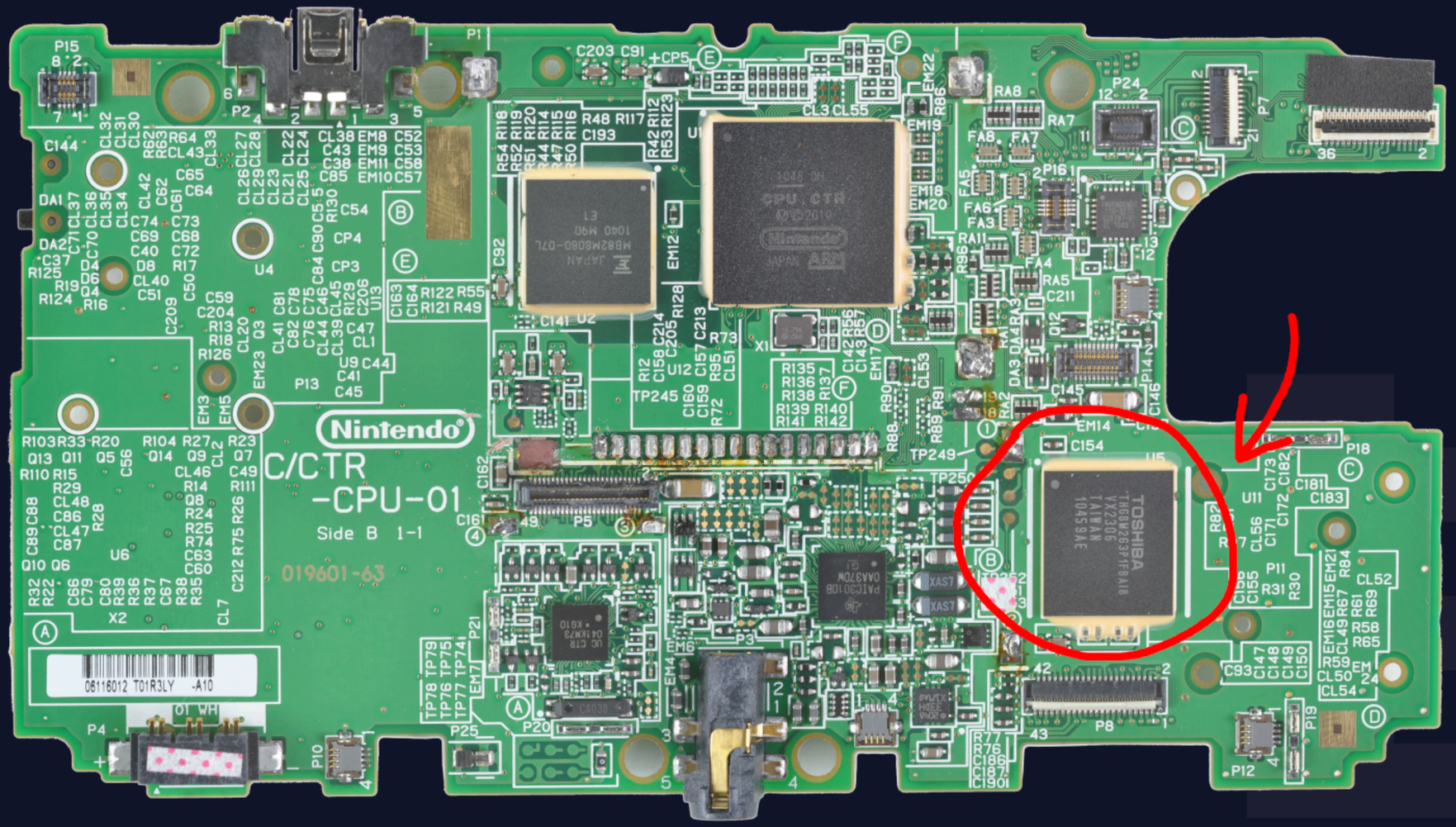
Presentation by **Cyprien Molinet** (`@cypelf`)

⚠️ Disclaimer

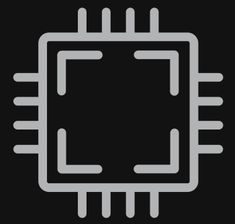
20 minutes is short.
This presentation simplifies some details.


Nintendo
C/CTR
-CPU-01
Side B 1-1
019601-63
CPU CTR
TOSHIBA

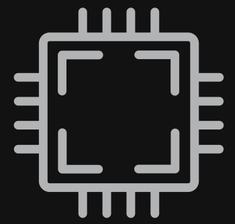

Nintendo
C/CTR
-CPU-01
Side B 1-1
019601-63
CPU_CTR
TOSHIBA


Nintendo
C/CTR
-CPU-01
Side B 1-1
019601-63
TOSHIBA

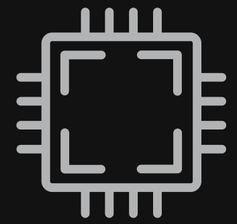
SoC CPU CTR



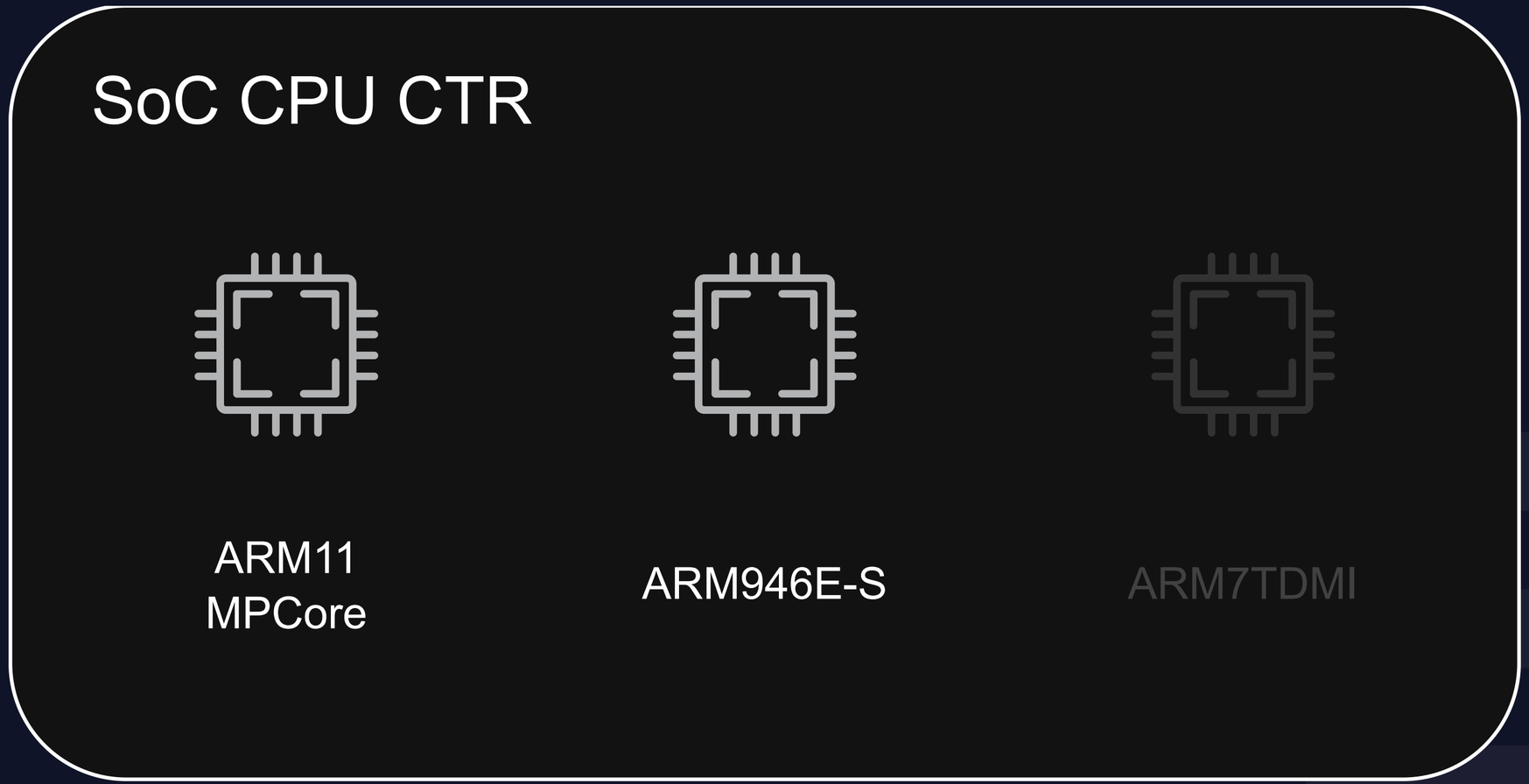
ARM11
MPCore

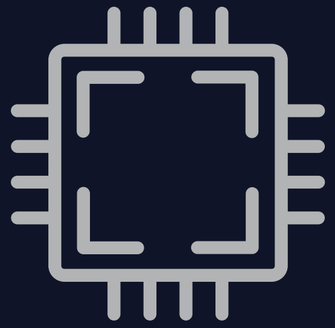


ARM946E-S

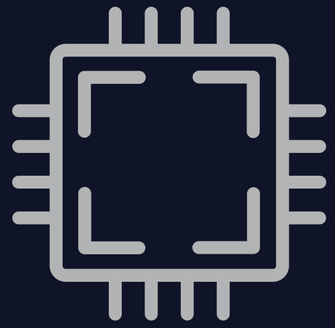


ARM7TDMI


SoC CPU CTR
ARM11
MPCore
ARM946E-S
ARM7TDMI

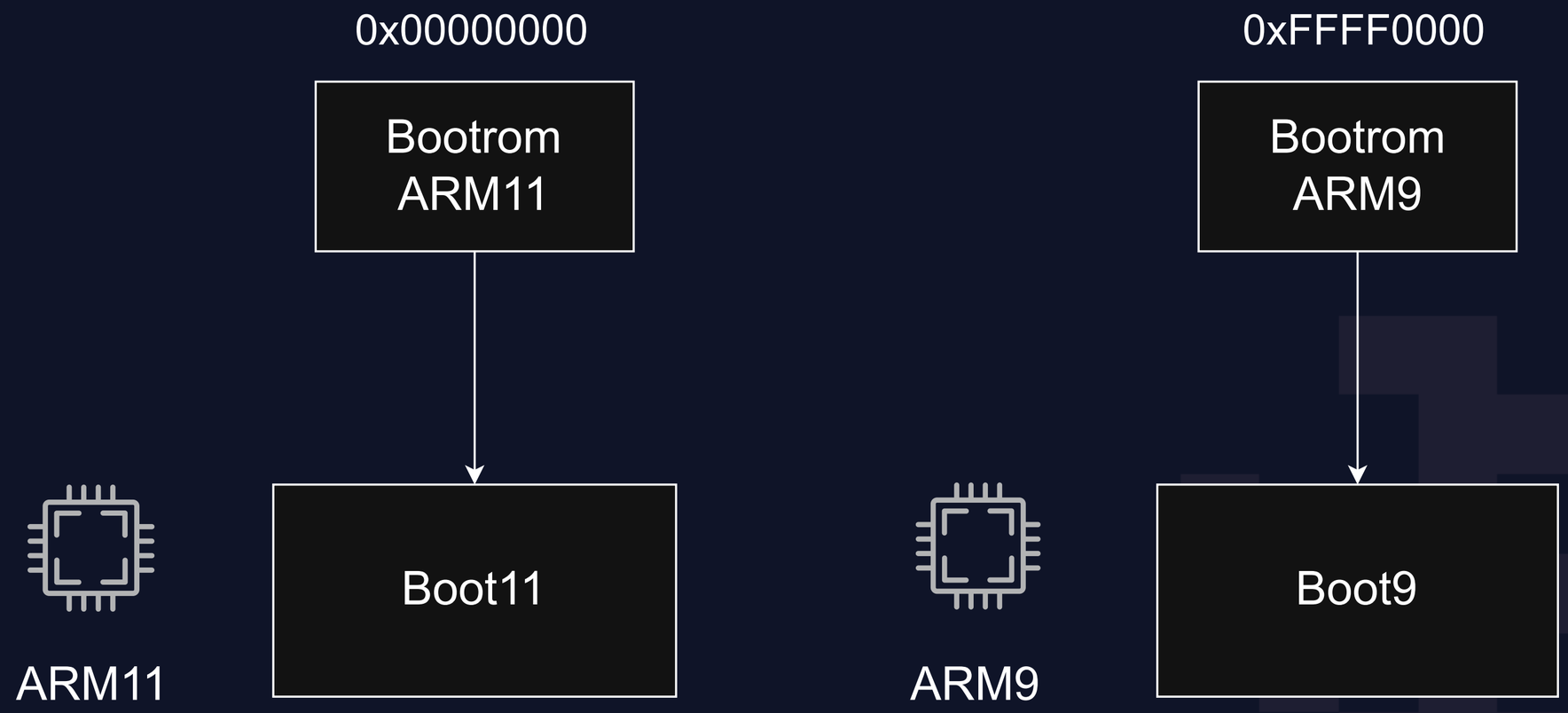


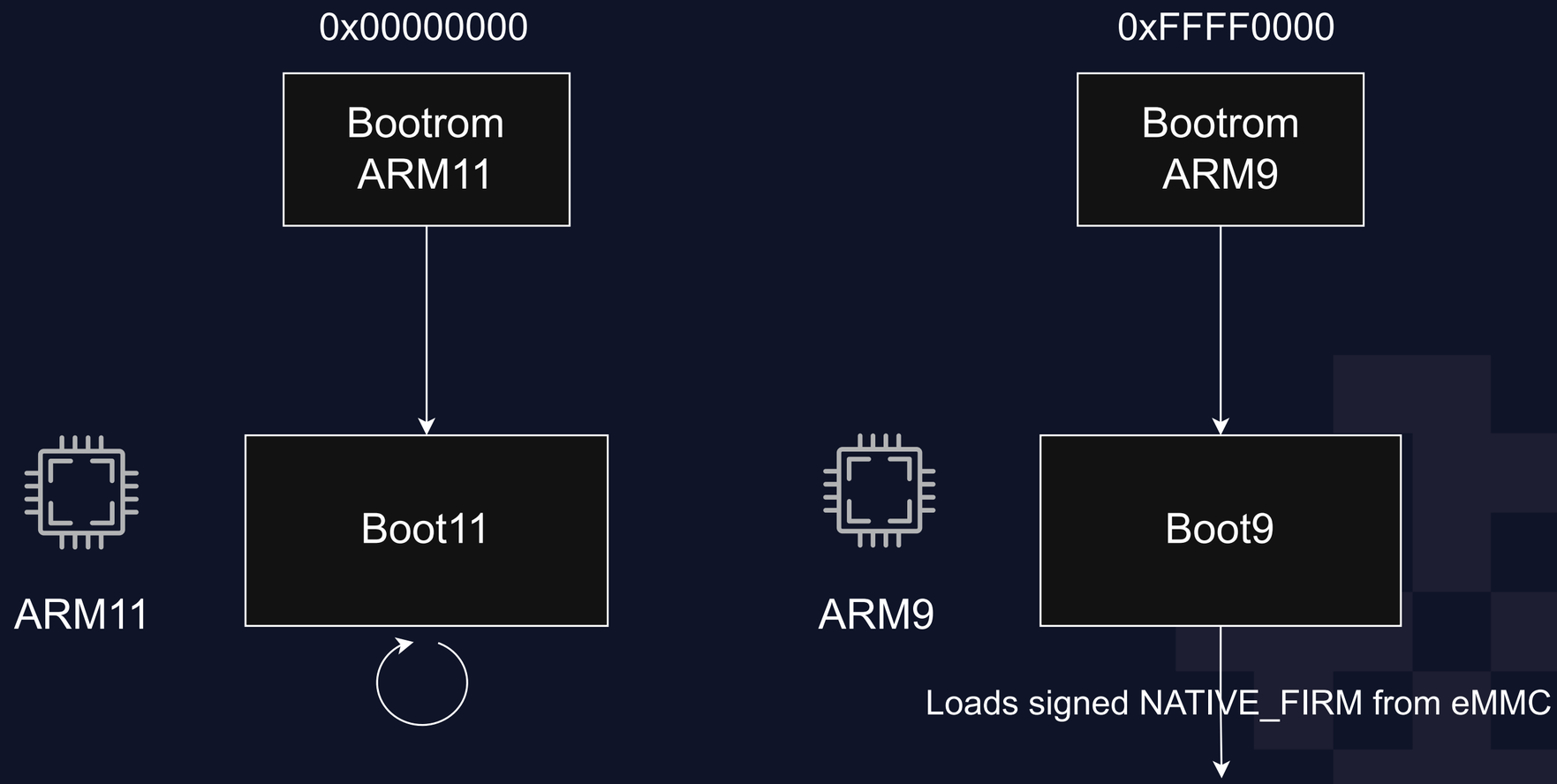
"PXI" FIFO
(IPC)

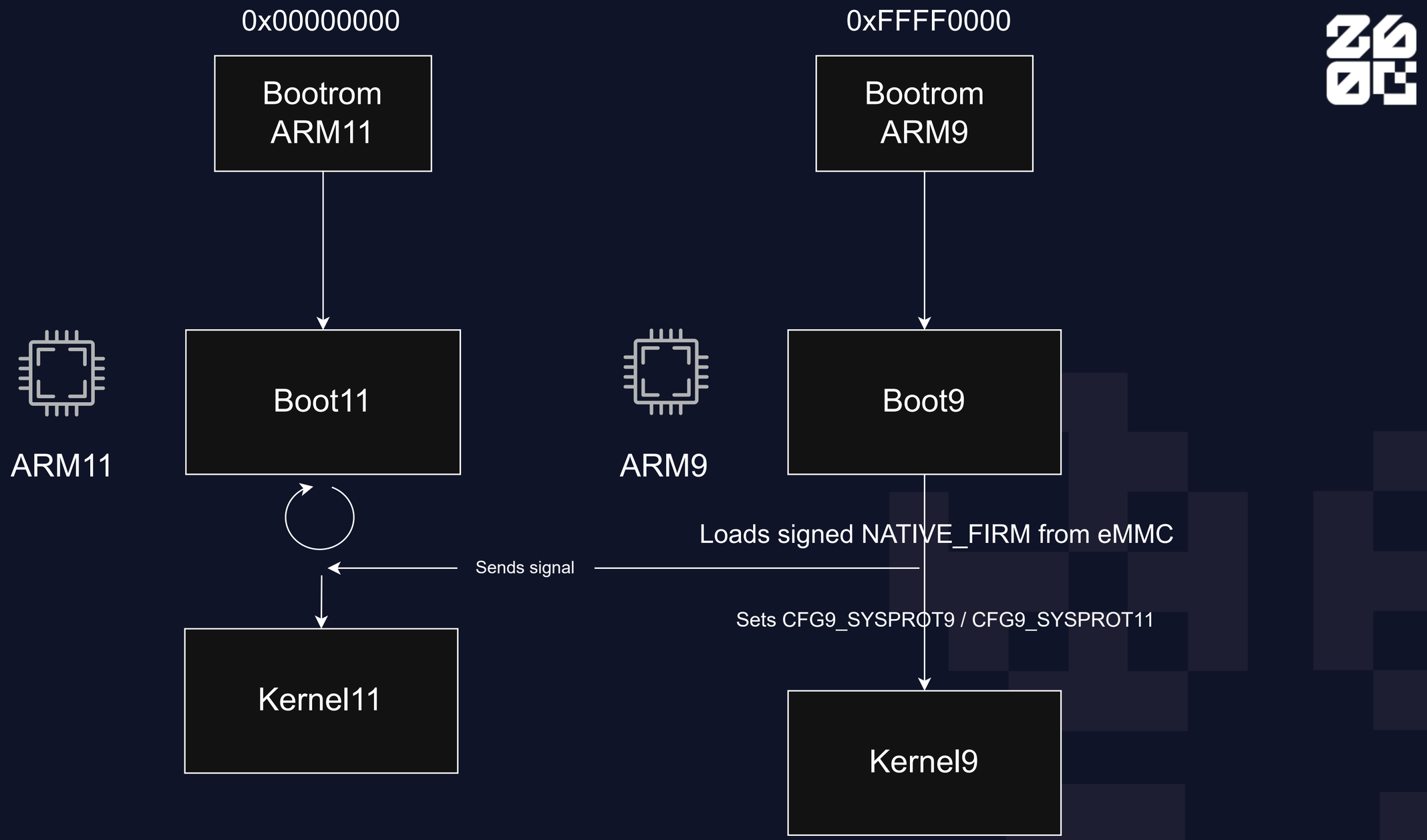


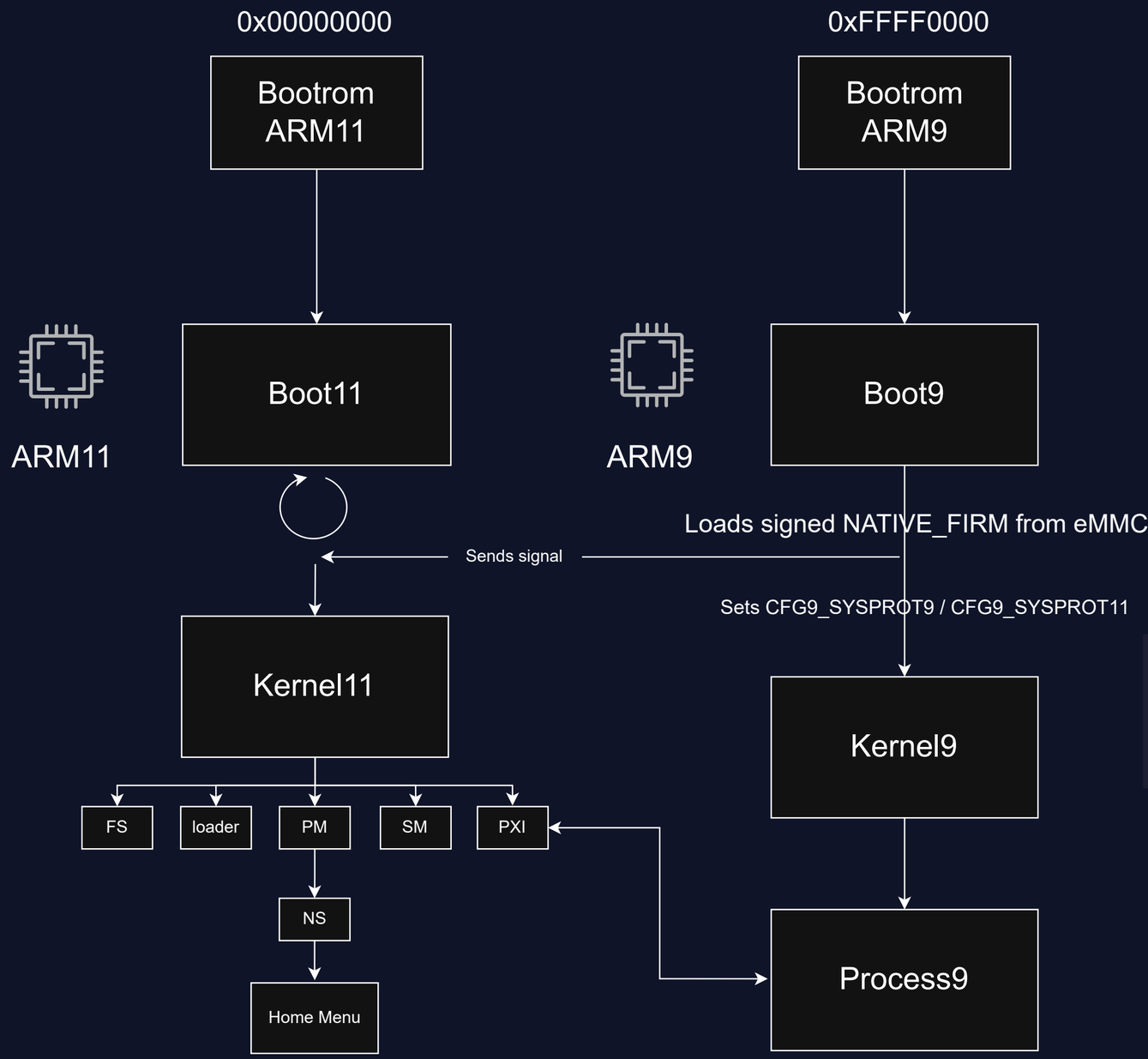
ARM11
MPCore

ARM946E-S


0x00000000
Bootrom
ARM11
Boot11
ARM11
0xFFFF0000
Bootrom
ARM9
Boot9
ARM9


0x00000000
Bootrom
ARM11
Boot11
ARM11
0xFFFF0000
Bootrom
ARM9
Boot9
ARM9
Loads signed NATIVE_FIRM from eMMC


0x00000000
Bootrom
ARM11
0xFFFF0000
Bootrom
ARM9
ARM11
Boot11
ARM9
Boot9
Loads signed NATIVE_FIRM from eMMC
Sends signal
Sets CFG9_SYSPROT9 / CFG9_SYSPROT11
Kernel11
Kernel9


0x00000000
0xFFFF0000
Bootrom ARM11
Bootrom ARM9
ARM11
Boot11
ARM9
Boot9
Loads signed NATIVE_FIRM from eMMC
Sends signal
Sets CFG9_SYSPROT9 / CFG9_SYSPROT11
Kernel11
Kernel9
FS
loader
PM
SM
PXI
NS
Home Menu
Process9

Context

Assuming the ARM11 and ARM9 cores are already fully compromised.
Including userland and both kernels.

Goal: dump the bootroms + get early code execution!

FIRM file format

FIRM Header

OFFSET	SIZE	DESCRIPTION
0x000	4	Magic 'FIRM'
0x004	4	Boot priority (highest value = max prio), this is normally zero.
0x008	4	ARM11 Entrypoint
0x00C	4	ARM9 Entrypoint
0x010	0x030	Reserved
0x040	0x0C0 (0x030*4)	Firmware Section Headers
0x100	0x100	RSA-2048 signature of the FIRM header's SHA-256 hash. The signature is checked when bootrom/Process9 are doing FIRM-launch (with the public key being hardcoded in each). The signature is not checked when installing FIRM to the NAND firm0/firm1 partitions.

Firmware Section Headers

OFFSET	SIZE	DESCRIPTION
0x000	4	Byte offset
0x004	4	Physical address where the section is loaded to.
0x008	4	Byte-size. While loading FIRM this is the field used to determine whether the section exists or not, by checking for value 0x0.
0x00C	4	Copy-method (0 = NDMA, 1 = XDMA, 2 = CPU mem-copy), Process9 ignores this field. Boot9 doesn't immediately throw an error when this isn't 0..2. In that case it will jump over section-data-loading which then results in the hash verification with the below hash being done with the hash already stored in the SHA hardware.
0x010	0x020	SHA-256 Hash of Firmware Section

FIRM file format

FIRM Header

OFFSET	SIZE	DESCRIPTION
0x000	4	Magic 'FIRM'
0x004	4	Boot priority (highest value = max prio), this is normally zero.
0x008	4	ARM11 Entrypoint
0x00C	4	ARM9 Entrypoint
0x010	0x030	Reserved
0x040	0x0C0 (0x030*4)	Firmware Section Headers
0x100	0x100	RSA-2048 signature of the FIRM header's SHA-256 hash. The signature is checked when bootrom/Process9 are doing FIRM-launch (with the public key being hardcoded in each). The signature is not checked when installing FIRM to the NAND firm0/firm1 partitions.

Firmware Section Headers

OFFSET	SIZE	DESCRIPTION
0x000	4	Byte offset
0x004	4	Physical address where the section is loaded to.
0x008	4	Byte-size. While loading FIRM this is the field used to determine whether the section exists or not, by checking for value 0x0.
0x00C	4	Copy-method (0 = NDMA, 1 = XDMA, 2 = CPU mem-copy), Process9 ignores this field. Boot9 doesn't immediately throw an error when this isn't 0..2. In that case it will jump over section-data-loading which then results in the hash verification with the below hash being done with the hash already stored in the SHA hardware.
0x010	0x020	SHA-256 Hash of Firmware Section

Reusing code

The factory version of NATIVE_FIRM had **issues** in its RSA verification and ASN.1 parsing.

They were patched in version 1.0.0-0, released in January 2011.

What about boot9?

What about boot9?

ASN.1 structure

```
DigestInfo ::= SEQUENCE {
    digestAlgorithm     DigestAlgorithmIdentifier,
    digest              Digest
}

DigestAlgorithmIdentifier ::= SEQUENCE {
    algorithm           OBJECT IDENTIFIER,
    parameters          ANY DEFINED BY algorithm OPTIONAL
}

Digest ::= OCTET STRING
```

ASN.1 actual values

```
SEQUENCE (2 elem)
  SEQUENCE (2 elem)
    OBJECT IDENTIFIER 2.16.840.1.101.3.4.2.1 (sha-256)
    NULL
  OCTET STRING (32 byte) [SHA256 hash value]
```

DER encoding

TLV encoding for ASN.1.

ASN.1 :

```
example ::= INTEGER
```

DER :

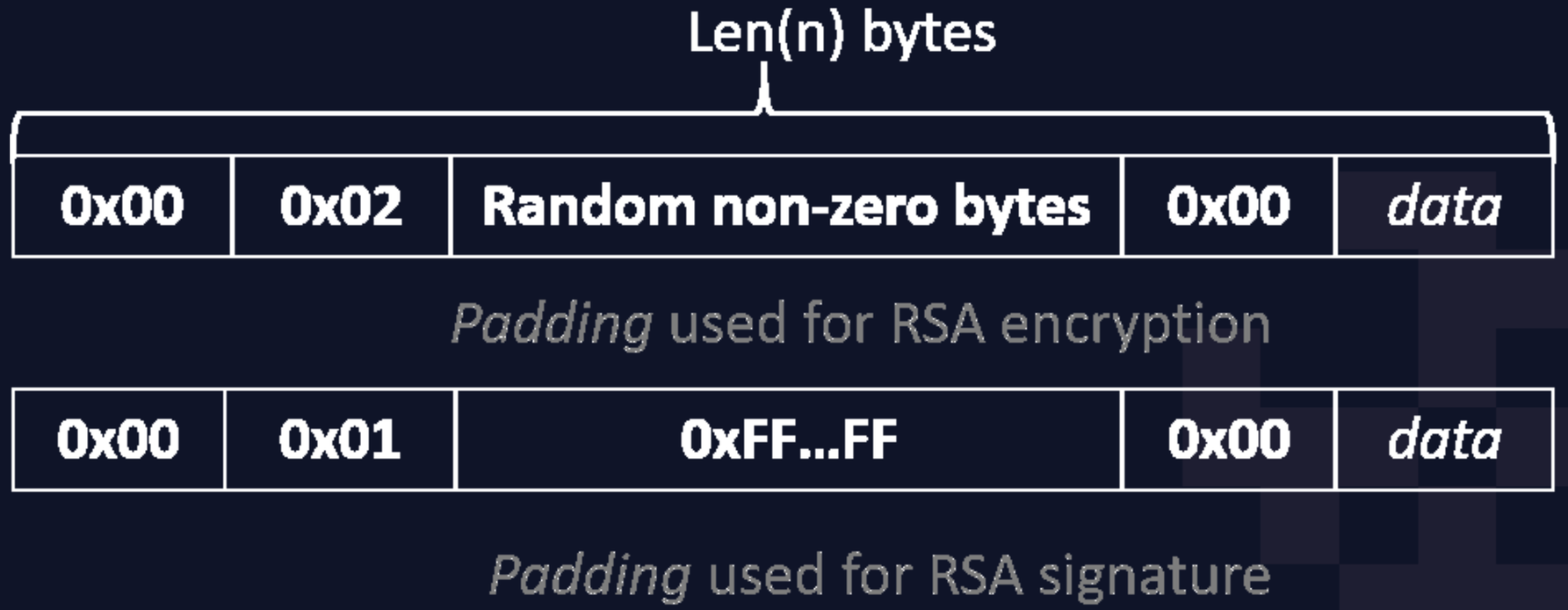
```
02 01 01
```

DER encoding for SHA256

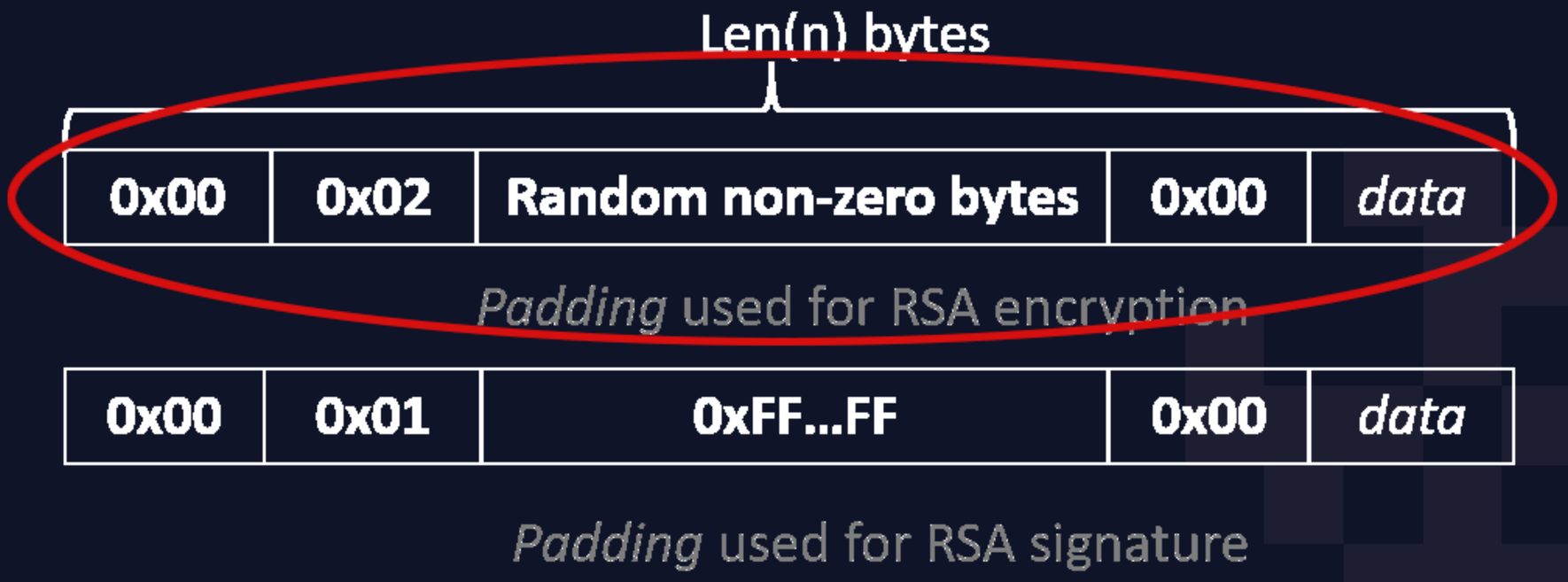
```
30 31 30 0d 06 09 60 86 48 01 65 03 04 02 01 05 00 04 20 [SHA256 hash value]
```

- `30 31` = **SEQUENCE** of length 0x31
- `30 0d` = **SEQUENCE** of length 0x0d
- `06 09` = **OID** of length 9
- `60 86 48 01 65 03 04 02 01` = OID value for **SHA256**
- `05 00` = **NULL type** of length 0
- `04 20` = **OCTET STRING** of length 0x20 (32 bytes)

PKCS#1 v1.5


Len(n) bytes
0x00
0x02
Random non-zero bytes
0x00
data
Padding used for RSA encryption
0x00
0x01
0xFF...FF
0x00
data
Padding used for RSA signature

PKCS#1 v1.5


Len(n) bytes
0x00
0x02
Random non-zero bytes
0x00
data
Padding used for RSA encryption
0x00
0x01
0xFF...FF
0x00
data
Padding used for RSA signature

Bad ASN.1 parser 101

```
int extract_asn1_from_rsa(uint8_t* signature, uint8_t** hash) {
    int result, length_field;
    uint8_t type_field;
    uint8_t* sig_ptr;

    sig_ptr = signature;
    if (signature
        && parse_asn1_field(&sig_ptr, &type_field, &length_field) && type_field == 0x30
        && parse_asn1_field(&sig_ptr, &type_field, &length_field) && type_field == 0x30)
    {
        sig_ptr += length_field;
        parse_asn1_field(&sig_ptr, &type_field, &length_field);
        if (hash)
            *hash = sig_ptr;
        result = 1;
    }
    else
        result = 0;

    return result;
}
```

*code reproduced from decompilation of factory firmware ASN.1 parser

Salut les cocos

The goal

The computed FIRM hash is **probably below on the stack**.
We can bruteforce "perfect" signatures!

GTX 1080 Ti goes brrrrr!

Structure of a sighax signature:

```
00 02 B3 13 31 C7 10 41 23 33 A5 87 89 0F 9C F0
B6 A8 6E 71 C8 A7 8F 96 B7 60 82 90 3B 3E 54 EA  Flag bytes
9A B9 35 97 8B BF 24 93 BB 82 9E 9A 5A 60 60 B0  Padding
C7 81 18 81 17 6B CF 9F E8 B1 C5 C5 E0 A9 53 27  ASN.1 Type Field
DB 8B 52 EC 17 8A 88 4A D9 CF 28 DB 8B BF 29 22  ASN.1 Length Field
C0 5F D0 34 AC 81 BD 23 1A EB 0C BE F6 F7 DE 6F  Added Length
3A 30 81 2B 9F 9A 83 BF 33 25 18 91 BF A1 8F A3
8A 64 C6 FF 5F 77 DB E1 1C 37 80 C2 3E A9 F6 D0
0F 9C 01 D6 FC 8A 87 85 91 D3 6C 4F 64 AC A6 B8
D1 1B BE B2 14 76 10 3C 6E 86 FF 21 96 D4 65 BA
4D B7 8F 81 F1 D3 BC CA 18 6B DD D5 67 39 A1 2D
D3 61 22 F3 F5 B3 DD 51 8D DA C4 FA 29 39 5E A4
CD 9D FD 80 AF 8A 39 99 90 F4 FD D3 CD 6B 07 EC
21 22 43 7C CF C3 B6 2B 1D 14 93 A7 DB B4 42 00
30 62 30 1A C0 A5 D8 7E 1E 31 A4 02 0F 0B EA EC
26 99 4D 25 80 32 4E 60 C6 CE AB A6 53 9A C8 14
{ Calculated hash located on stack right here }
```

Sighax

Any FIRM can be **signed** now.
Can't be patched.

Dumping the bootroms

`CFG9_SYSPROT9` and `CFG9_SYSPROT11` : set by boot9 **just before** going to FIRM entrypoints.

But **after** copying FIRM sections in memory!

FIRM arbitrary write

FIRM Header

OFFSET	SIZE	DESCRIPTION
0x000	4	Magic 'FIRM'
0x004	4	Boot priority (highest value = max prio), this is normally zero.
0x008	4	ARM11 Entrypoint
0x00C	4	ARM9 Entrypoint
0x010	0x030	Reserved
0x040	0x0C0 (0x030*4)	Firmware Section Headers
0x100	0x100	RSA-2048 signature of the FIRM header's SHA-256 hash. The signature is checked when bootrom/Process9 are doing FIRM-launch (with the public key being hardcoded in each). The signature is not checked when installing FIRM to the NAND firm0/firm1 partitions.

Firmware Section Headers

OFFSET	SIZE	DESCRIPTION
0x000	4	Byte offset
0x004	4	Physical address where the section is loaded to.
0x008	4	Byte-size. While loading FIRM this is the field used to determine whether the section exists or not, by checking for value 0x0.
0x00C	4	Copy-method (0 = NDMA, 1 = XDMA, 2 = CPU mem-copy), Process9 ignores this field. Boot9 doesn't immediately throw an error when this isn't 0..2. In that case it will jump over section-data-loading which then results in the hash verification with the below hash being done with the hash already stored in the SHA hardware.
0x010	0x020	SHA-256 Hash of Firmware Section

FIRM arbitrary write

FIRM Header

OFFSET	SIZE	DESCRIPTION
0x000	4	Magic 'FIRM'
0x004	4	Boot priority (highest value = max prio), this is normally zero.
0x008	4	ARM11 Entrypoint
0x00C	4	ARM9 Entrypoint
0x010	0x030	Reserved
0x040	0x0C0 (0x030*4)	Firmware Section Headers
0x100	0x100	RSA-2048 signature of the FIRM header's SHA-256 hash. The signature is checked when bootrom/Process9 are doing FIRM-launch (with the public key being hardcoded in each). The signature is not checked when installing FIRM to the NAND firm0/firm1 partitions.

Firmware Section Headers

OFFSET	SIZE	DESCRIPTION
0x000	4	Byte offset
0x004	4	Physical address where the section is loaded to.
0x008	4	Byte-size. While loading FIRM this is the field used to determine whether the section exists or not, by checking for value 0x0.
0x00C	4	Copy-method (0 = NDMA, 1 = XDMA, 2 = CPU mem-copy), Process9 ignores this field. Boot9 doesn't immediately throw an error when this isn't 0..2. In that case it will jump over section-data-loading which then results in the hash verification with the below hash being done with the hash already stored in the SHA hardware.
0x010	0x020	SHA-256 Hash of Firmware Section

⚠️ Boot9 blacklist

FIRM sections can't be written to ARM9 SRAM.

ARM9 memory layout

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

ARM9 memory layout

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

ARM9 memory layout

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

NDMA engine

IO Registers

Overview

Old3DS	A9/ A11	Category	Physaddr	Used by	Comments
Yes	A9	CONFIG9 Registers	0x10000000	Boot9, Process9	
Yes	A9	IRQ Registers	0x10001000	Boot9, Process9, Kernel9	ARM9 Interrupt Masking
Yes	A9	NDMA Registers	0x10002000	Boot9, Process9	AHB DMA Engine
Yes	A9	TIMER Registers	0x10003000	Boot9, Process9	
Yes	A9	CTRCARD Registers	0x10004000 / 0x10005000	Process9	
Yes	A9	EMMC Registers	0x10006000 / 0x10007000	Boot9, Process9, NewKernel9Loader	SD(IO) controller 1 and 3. 3 is normally mapped to ARM11.
Yes	A9	PXI Registers	0x10008000	Boot9, Process9	
Yes	A9	AES Registers	0x10009000	Boot9, Process9, NewKernel9Loader	
Yes	A9	SHA Registers	0x1000A000	Boot9, Process9, NewKernel9Loader	
Yes	A9	RSA Registers	0x1000B000	Boot9, Process9	
Yes	A9	XDMA Registers	0x1000C000	Boot9, Kernel9	CoreLink™ DMA-330 r0p0 (AXI busmaster, two channels, uses 32-bit bus width instead of 64).
Yes	A9	SPICARD Registers	0x1000D800	Process9	
Yes	A9	CONFIG Registers	0x10010000	Process9	

NDMA engine

IO Registers

Overview

Old3DS	A9/A11	Category	Physaddr	Used by	Comments
Yes	A9	CONFIG9 Registers	0x10000000	Boot9, Process9	
Yes	A9	IRQ Registers	0x10001000	Boot9, Process9, Kernel9	ARM9 Interrupt Masking
Yes	A9	NDMA Registers	0x10002000	Boot9, Process9	AHB DMA Engine
Yes	A9	TIMER Registers	0x10003000	Boot9, Process9	
Yes	A9	CTRCARD Registers	0x10004000 / 0x10005000	Process9	
Yes	A9	EMMC Registers	0x10006000 / 0x10007000	Boot9, Process9, NewKernel9Loader	SD(IO) controller 1 and 3. 3 is normally mapped to ARM11.
Yes	A9	PXI Registers	0x10008000	Boot9, Process9	
Yes	A9	AES Registers	0x10009000	Boot9, Process9, NewKernel9Loader	
Yes	A9	SHA Registers	0x1000A000	Boot9, Process9, NewKernel9Loader	
Yes	A9	RSA Registers	0x1000B000	Boot9, Process9	
Yes	A9	XDMA Registers	0x1000C000	Boot9, Kernel9	CoreLink™ DMA-330 r0p0 (AXI busmaster, two channels, uses 32-bit bus width instead of 64).
Yes	A9	SPICARD Registers	0x1000D800	Process9	
Yes	A9	CONFIG Registers	0x10010000	Process9	

💔 The blacklist becomes useless!

NDMA gives us **unrestricted arbitrary copy**.

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers
2. NDMA triggers a copy from ARM9 bootrom to ARM9 SRAM

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers
2. NDMA triggers a copy from ARM9 bootrom to ARM9 SRAM
3. `CFG9_SYSPROT9` gets set by boot9, bootrom is locked

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers
2. NDMA triggers a copy from ARM9 bootrom to ARM9 SRAM
3. `CFG9_SYSPROT9` gets set by boot9, bootrom is locked
4. FIRM ARM9 entrypoint starts

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers
2. NDMA triggers a copy from ARM9 bootrom to ARM9 SRAM
3. `CFG9_SYSPROT9` gets set by boot9, bootrom is locked
4. FIRM ARM9 entrypoint starts
5. The bootrom copy is accessible in ARM9 memory!

Dumping boot9

1. Write a FIRM with a section copied to the NDMA registers
2. NDMA triggers a copy from ARM9 bootrom to ARM9 SRAM
3. `CFG9_SYSPROT9` gets set by boot9, bootrom is locked
4. FIRM ARM9 entrypoint starts
5. The bootrom copy is accessible in ARM9 memory!

What’s next?

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

ARM9 exception vectors

Stored at `0x08000000` in ARM9 memory.
Writable using **NDMA**!

Triggering an exception

Create a FIRM section that copies to **NULL**.

Triggering an exception

Create a FIRM section that copies to **NULL**.

Data abort!

Getting boot9 code execution

1. Copy an ARM9 payload from FIRM to ARM9 memory

Getting boot9 code execution

1. Copy an ARM9 payload from FIRM to ARM9 memory
2. Trigger NDMA to replace ARM9 data abort vector

Getting boot9 code execution

1. Copy an ARM9 payload from FIRM to ARM9 memory
2. Trigger NDMA to replace ARM9 data abort vector
3. Copy FIRM section to NULL

Getting boot9 code execution

1. Copy an ARM9 payload from FIRM to ARM9 memory
2. Trigger NDMA to replace ARM9 data abort vector
3. Copy FIRM section to NULL
4. ARM9 payload starts executing!

Getting boot9 code execution

1. Copy an ARM9 payload from FIRM to ARM9 memory
2. Trigger NDMA to replace ARM9 data abort vector
3. Copy FIRM section to NULL
4. ARM9 payload starts executing!

Getting boot11 code execution

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

Getting boot11 code execution

ARM9

Old 3DS	Address	Size	Description
Yes	0x00000000	0x08000000	Instruction TCM, repeating each 0x8000 bytes.
Yes	0x01FF8000	0x00008000	Instruction TCM (Accessed by the kernel and process by this address)
Yes	0x07FF8000	0x00008000	Instruction TCM (Accessed by bootrom by this address)
Yes	0x08000000	0x00100000	ARM9-only internal memory (ARM7's internal regions are mapped here as well)
No	0x08100000	0x00080000	New_3DS ARM9-only extension, only enabled when a certain CONFIG register is set.
Yes	0x10000000	0x08000000	IO memory
Yes	0x18000000	0x00600000	VRAM (divided in two banks, VRAM and VRAMB)
Yes	0x1FF00000	0x00080000	DSP memory
Yes	0x1FF80000	0x00080000	AXI WRAM
Yes	0x20000000	0x08000000	FCRAM
No	0x28000000	0x08000000	New_3DS FCRAM extension
Yes	0xFFF00000	0x00004000	Data TCM (Mapped during bootrom). Enabled at the time Boot9 jumps to FIRM, however Kernel9+arm9loader disables it.
Yes	0xFFFF0000	0x00010000	Bootrom, the main region is at +0x8000, which is disabled during system boot.

Getting boot11 code execution

1. Copy an ARM11 payload from FIRM to AXI WRAM

Getting boot11 code execution

1. Copy an ARM11 payload from FIRM to AXI WRAM
2. Replace a boot11 function pointer in AXI WRAM

Getting boot11 code execution

1. Copy an ARM11 payload from FIRM to AXI WRAM
2. Replace a boot11 function pointer in AXI WRAM
3. ARM11 payload starts executing!

Getting boot11 code execution

1. Copy an ARM11 payload from FIRM to AXI WRAM
2. Replace a boot11 function pointer in AXI WRAM
3. ARM11 payload starts executing!

Dumping boot11

Code execution is gained before bootrom lockdown.
It can still be read directly.

Boot9strap

FIRM getting boot9 / boot11 code execution.
Starts another FIRM from SD or NAND.

https://github.com/SciresM/boot9strap

Used to start CFW!

Sources and resources

- https://www.3dbrew.org
- https://arxiv.org/pdf/1802.00359
- https://sciresm.github.io/33-and-a-half-c3/
- https://www.copetti.org/writings/consoles/nintendo-3ds

Special thanks

- Noé Coste (NoXe) (͡° ͜ʖ ͡°)

Thank you for your attention!

Feel free to reach out and ask questions after the talk.

Avdray is back!