DIGIC8 ORACLE

Decrypting camera updates
without knowing neither the key, nor algorithms (at first)

Laurent Clévy

Hack.lu, October 22, 2025

BIO

Laurent Clévy

- Reverse engineering / pentesting embedded systems
- Former Forensic analyst in a SOC/CERT + some reverse engineering

Free time contributions (like this talk)

- Canon RAW v2 and RAW v3 file formats reference documentations
- Following Magic Lantern (aka ML) activity and hacking since years
- Reversed in 2012 Canon Original Data Decision implementation, with python tool to recompute digital pictures signatures.
- BeerRump 2022 talk about old FIR updates version 4 (decryption and signature)
- An antivirus in 68000 assembly (https://github.com/lclevy/Uvk)

MOTIVATION

Curiosity & learning

Executing native (ARMv7) code on my camera (Canon EOS R6)

Goal: Find a way to execute native code on EOS R and recent Canon cameras, via updates (FIR format).

DISCLAIMERS

DigIC (Digital IC) : System on Chip from Canon inside their digital cameras.
Digic and EOS are Canon trademarks

There is no need to decrypt updates to access Digic 8 and Digic 10 firmware internals. Anyone without technical skills can dump firmware from EOS R / RP (Digic 8) and EOS R5/R6 (Digic X) cameras with Canon Basic scripting (DIGIC 8, DIGIC X models)

No decryption key, neither firmware dumps will be dropped with this talk.

This talk is about personal work.
Opinions are my own, not my employer, neither Magic Lantern team

DIGITAL CAMS ARE COMPUTING DEVICES

Digital Single Lens Reflex (DSLR) or Mirrorless Interchangeable Lens (MILC) cameras are complex devices

- Multiples CPUs (main, AF, peripherals, GPU, face recognition, network ops, ...)
- ARM-A9, ARM-M4 (mpu), Tensilica Xtensa (net), Takumi GV550 (gpu)
- Several instances of RTOS (DryOS)
- Wifi, Bluetooth, Ethernet, GPS, USB, HDMI
- RAW image processing at 10-30 frame/sec

Dedicated System on Chip for Canon : DigIC

And hackers have managed to run Doom on it !

https://www.youtube.com/watch?v=fAoIjXZYu70 (Doom on RP by @coon)

https://wiki.magiclantern.fm/digic (which CPUs per Digic generation)

STANDING ON GIANTS SHOULDERS

CHDK, Canon Hack DevKit (pocket cameras)

- https://chdk.fandom.com/wiki/CHDK
- Enhancing official firmware
  - RAW, LUA scripting,…
  - Loaded from sdcard, only in memory

Magic Lantern (DSLR)

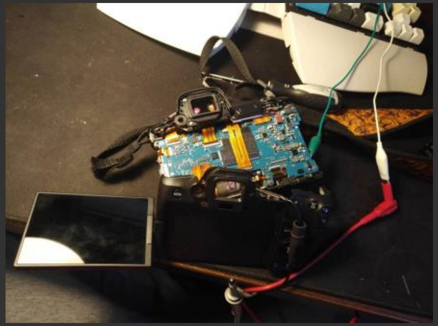
- Created by Trammell Hudson (https://trmm.net/Magic_Lantern_firmware/)
- 2010-2018 main contributors: A1ex (main dev), G3gg0 (reverse), Arm.Indy (crypto), …
  - Code execution happened via custom firmware updates (FIR format, version 4)
- 2018-2025 : Names_are_hard (main dev), Kitor (reverse), Coon, Petabyte, Turtius, …
  - Starting EOS R model : Code execution via Cbasic or UART access, then native code

LEGACY MAGIC LANTERN CODE EXECUTION BROKEN IN 2018

Before EOS R and since 2010, code execution is achieved by forging custom updates with valid hmac-sha1 signatures

which is broken since EOS R model (09/2018)

- "Cryptography of FIR format changed"
- @_kitor & @A1ex managed to execute native code then dump firmware via UART access
- but UART access is not suitable for casual ML users



Game over: R10 and R50 (2023), CBasic and UART are locked !!

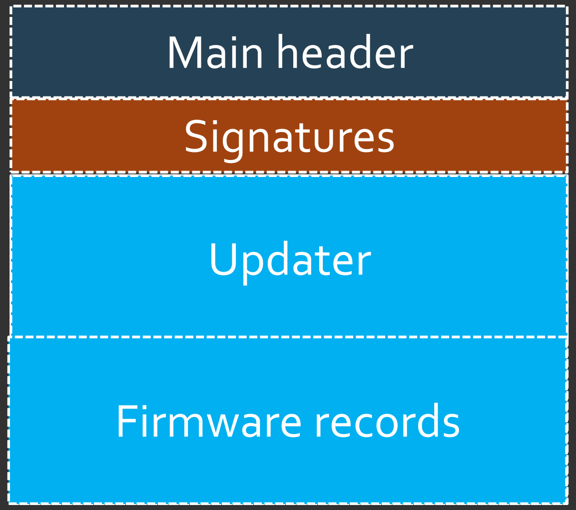
What exactly changed in 2018 within FIR format ?

FIR FORMAT AND UPDATE PROCESS

UPDATER & FIRMWARE : CHICKEN & EGG

Inside the FIR file, mainly two parts :

1. Code to apply the update : Updater
2. What to update : Firmware records



The updater is a minimal OS version able to update <u>all</u> the flash memory, including the bootcode.

During update process, the camera

1. Loads FIR file into memory (main OS)
2. Reboot into updater OS instance
3. Apply firmware records : write them into the flash
4. Reboot into main OS

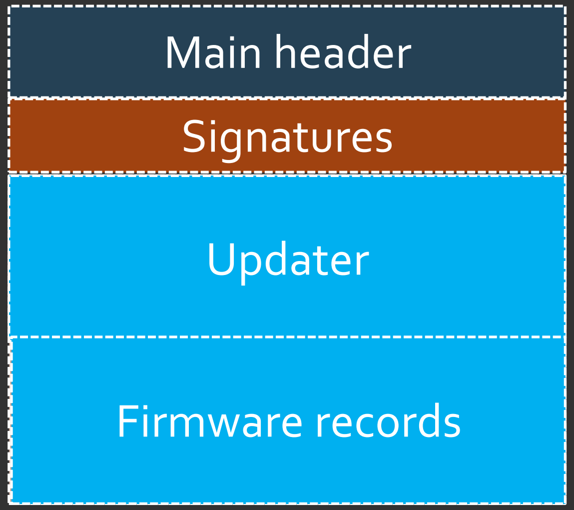
AUTHENTICITY AND CONFIDENTIALITY

Updater and firmware records are both signed and encrypted

1. Main OS is running:
   if signature_is_valid(updater) then:
   decrypt(updater)
   reboot into updater

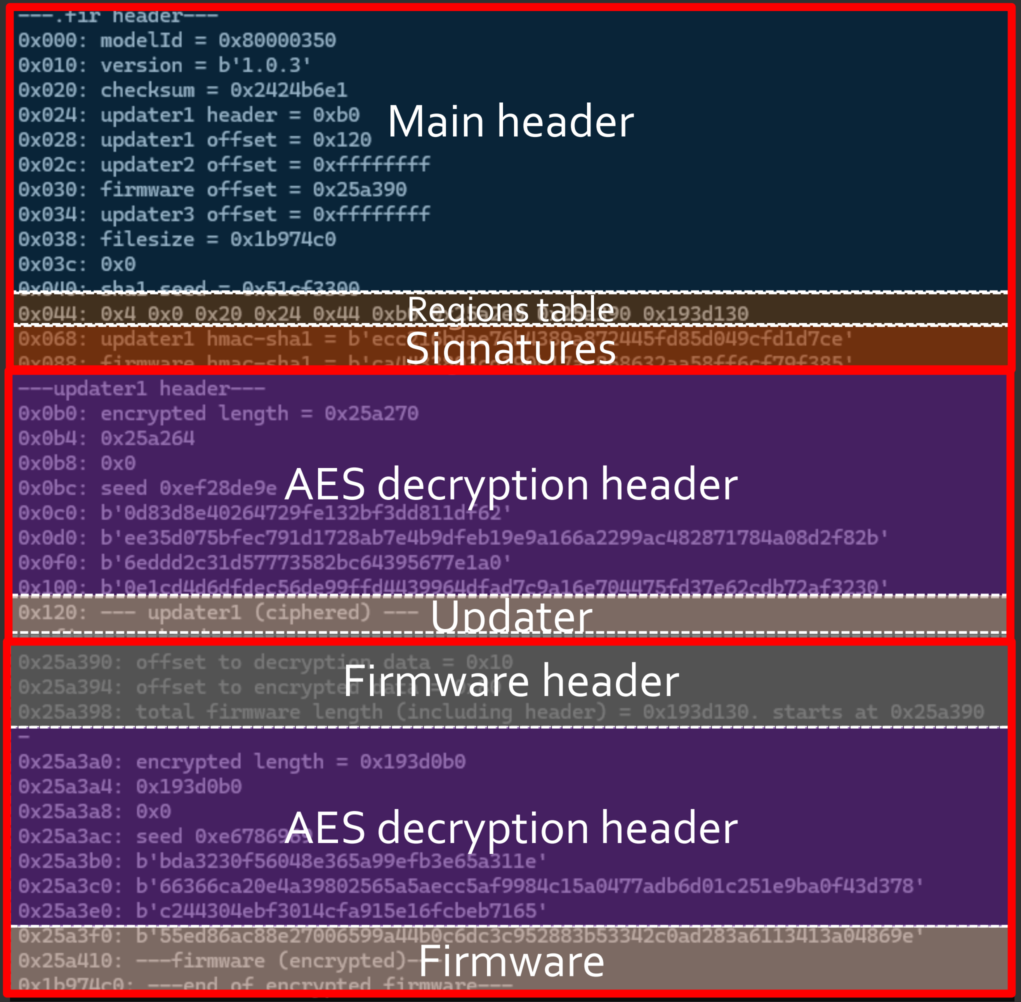
2. Updater is running:
   if signature_is_valid(firmware) then:
   decrypt(firmware)

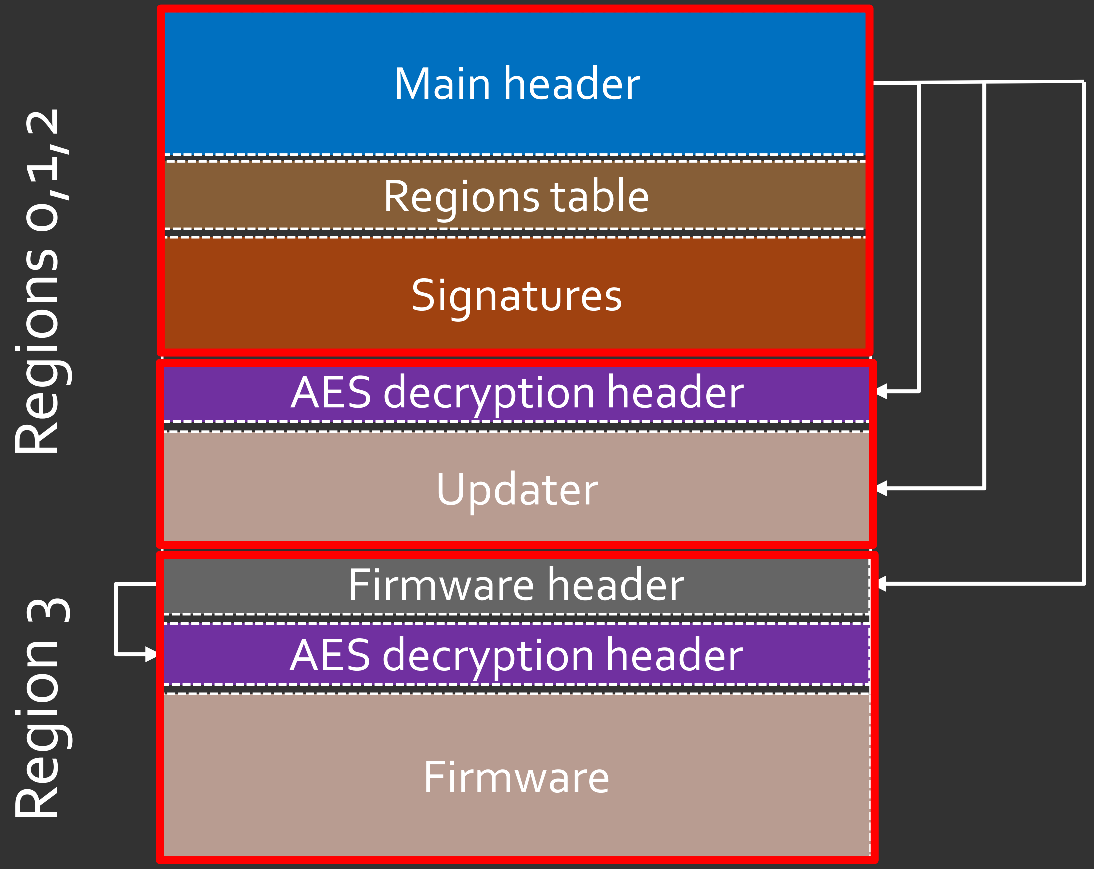
3. Apply update records, and reboot



=> if you can forge the updater signature, you obtain code execution !

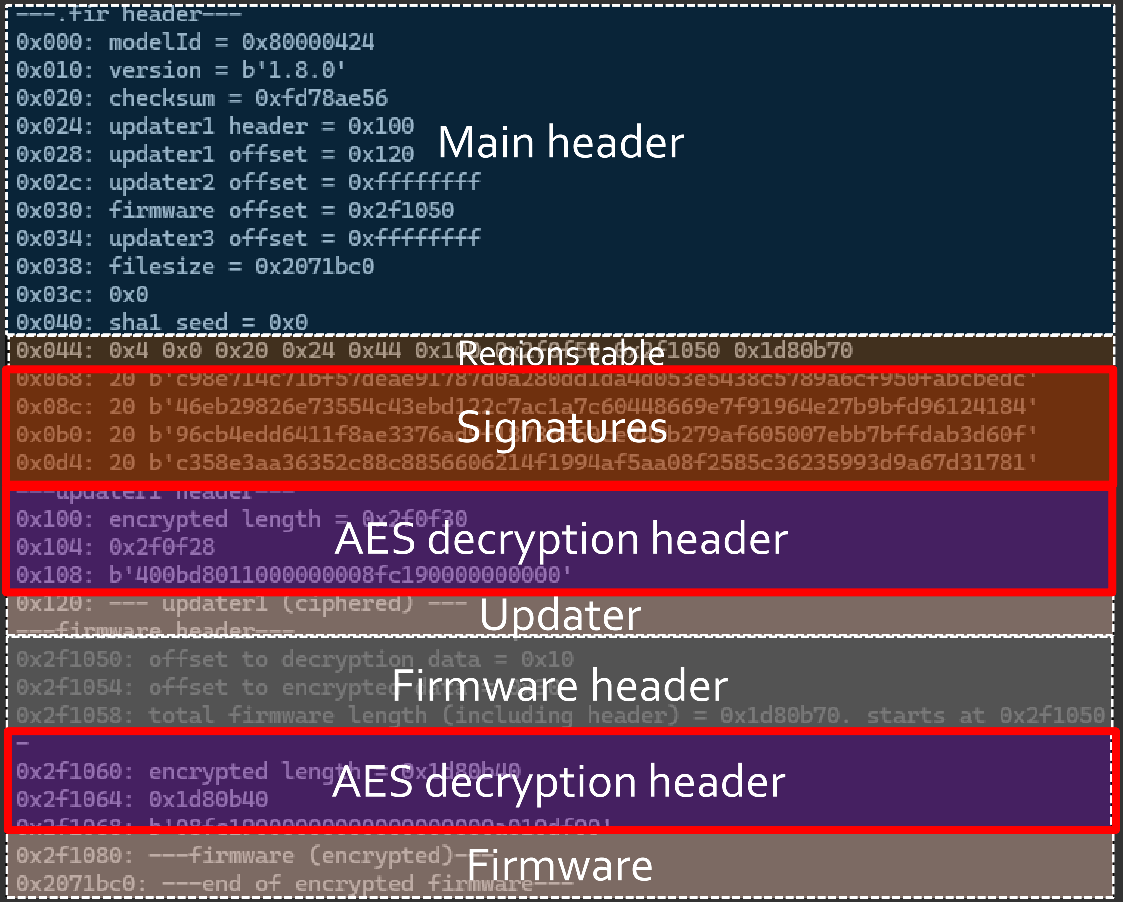
FIR V4 FORMAT, 80D 1.0.3 EXAMPLE





Regions table defines which data regions are signed

NEW FIR: EOS R AND LATER (>2018)



Very similar to FIRv4, but

Signatures section is *bigger*

- now has 4 values of 32 bytes,
- instead of 2 values of 20 bytes (hmac-sha1)

AES decryption header is *shorter*

- Now has 32 bytes
- Previously was 112 bytes

It seems <u>only</u> crypto has changed.

AES decryption with 2010 key is failing

DUMPING AND FINDING CRYPTO CODE

CBASIC DUMPING AND CIPHER.BIN

CBasic interpreter is available for some Canon cameras:

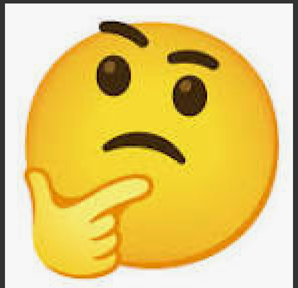
- EOS R, RP (Digic 8). EOS R6, R5 (Digic 10)
- This script will dump the camera firmware:

```
private sub Initialize()
    SaveAllRomImageToFile()
end sub
```

Let's dump EOS R firmware, grep for cipher string ☺ and AES constants

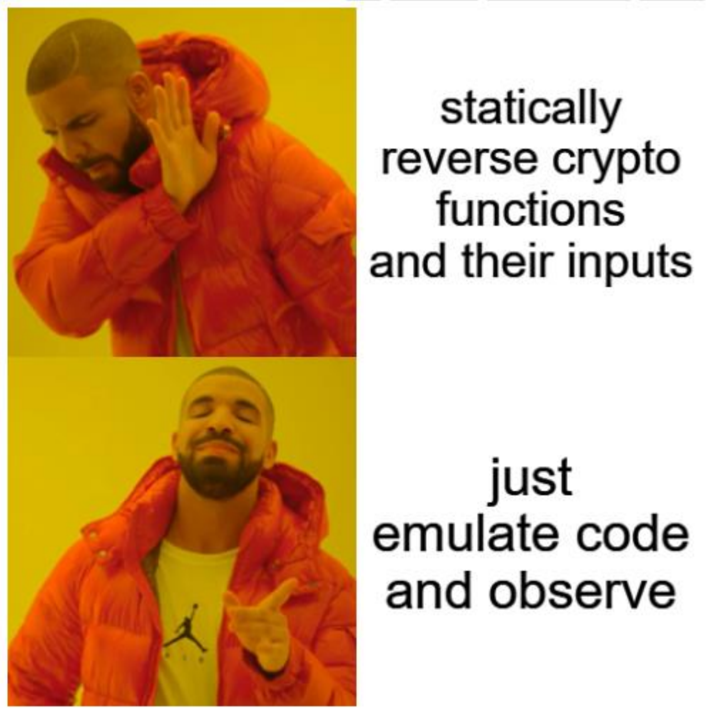
There is code called cipher.bin, copied to RAM (0x200000-0x20af00 range)
Beginning of cipher.bin code looks like this:

```
FIR_ADDRESS = 0x800000
if func1( FIR_ADDRESS, 0x205784, 0x20, 0x2057a4, 0x20 ):
          func2( FIR_ADDRESS, 0xbfe00100, 0x100, 0x2057C4, 0x10 )
```

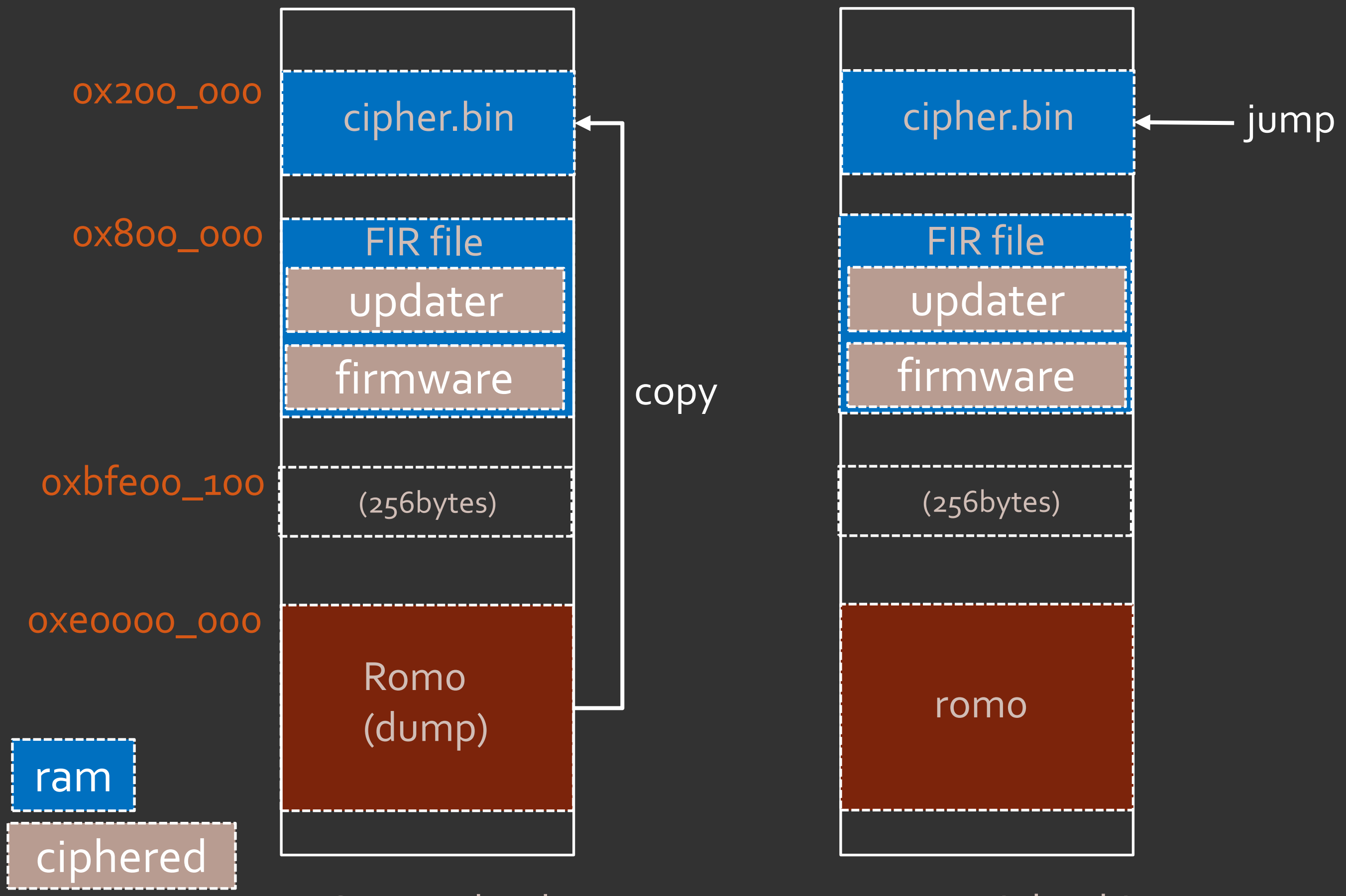


0x2057xx data is inside *cipher.bin* region, so required inputs are:
0x100 bytes at 0xbfe00100 and FIR content at 0x800000
thus, why not testing dumped code as Oracle through emulation ?

UNICORN EMULATION



EMULATION SETUP FOR UPDATER DECRYPTION



1. Copy cipher.bin to ram

2. Jump to cipher.bin to decrypt updater

1. loading FIR data at 0x800_000 (RAM)

2. setup ROM0 at 0xe0000_000 (only cipher.bin will be used)

3. copy cipher.bin code from flash (0xe0039_000) to RAM (0x200_000)

4. setup data at 0xbfe00_100

5. setup stack and memory space for malloc()/free()

6. jump to 0x200_001, as this is Thumb code

D8_ORACLE.PY SCRIPT

Unicorn

The Ultimate CPU emulator

It worked ! Because **cipher.bin** has been designed to be moved to RAM, interactions with DryOS are restricted to malloc/free. We are very lucky !

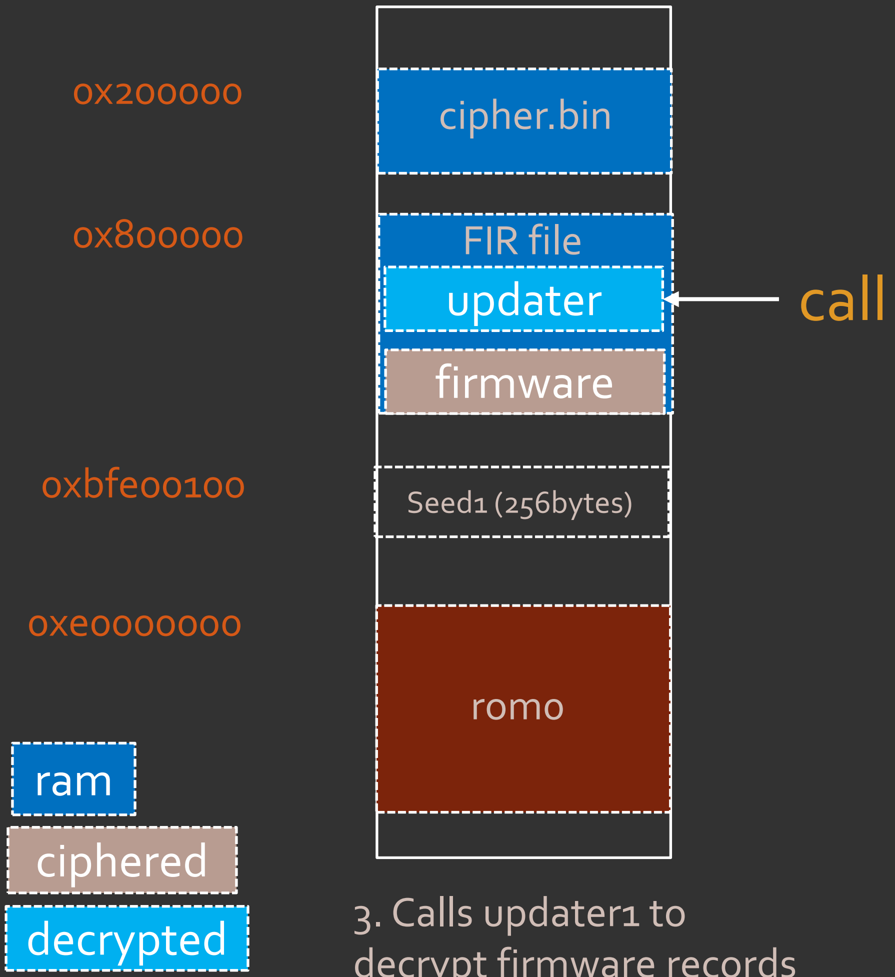
And, if the decryption key is unique to all digic8 cameras (like previously), we can write a python tool to decrypt updater from a camera using a dump from another (Digic8) camera:

```
>python d8_oracle.py -u -r roms\eosr_110.BIN fir\EOSRP160.FIR
Input is update file fir\EOSRP160.FIR
allocating 0x20c6400 bytes at 0x800000 for FIR file
Oracle is rom file roms\eosr_110.BIN loaded at 0xe0000000
Emulating cipher.bin at 0x200000. Code copied from 0xe0039000
dumping verified and decrypted updater1 (0x800120-0xae5030) to file 80000433_1.6.0_updater1.bin
```

Above, we decrypt EOS RP update v1.6.0 using a cipher.bin from EOS R 1.1.0 without knowing the key and algorithm (I.E. we use dumped code as Oracle)

What about decrypting firmware records now ?

LET'S DECRYPT FIRMWARE RECORDS



3. Calls updater1 to
decrypt firmware records

We have to locate
the decryption call in updater

1. Look for crypto constants : AES detected
2. Look for AES expansion function (sbox table)
3. …
4. Found references to 0xbfe00_100 and 0x2057c4 again !
5. Identify required memory / registers context

DECRYPTION CALL IN UPDATER

This is where decryption is called in EOS R 1.8.0 updater:

```
0081fef0  movs r0,#0x10
0081fef2  ldr r3,[PTR_DAT_00820024] ; 0x2057C4
0081fef4  lsls r2,r0,#0x4           ; 0x10<<4 == 0x100
0081fef6  ldr r1,[PTR_DAT_00820028] ; 0xbfe00100
0081fef8  str r0,[sp,#0x0]=>local_b0 ; stack+0, 0x10
0081fefa  mov r0,r7
0081fefc  bl decrypt
```

It seems AES key is derived from 2 seeds values at 0xbfe00100 and 0x2057c4

```
decrypt( FIR_ADDRESS, seed1_data=0xbfe00100, seed1_size=0x100, seed2_data=0x2057C4, seed2_size=0x10)
              r0            r1                     r2                 r3                   stack+0
```

Unicorn decrypt() arguments setup:
emulation starts at 0x81fef4
so r1 and stack+0 are filled my emulation

```
mu.reg_write(UC_ARM_REG_R3, 0x2057C4 )
mu.reg_write(UC_ARM_REG_R0, 0x10 )   # 0x100 in R2
mu.reg_write(UC_ARM_REG_R7, FIR_ADDRESS )   // R0

EMU_START_ADDRESS = 0x81fef4
```

LET'S IMPROVE D8_ORACLE.PY

Let's decrypt firmware records for another Digic8 camera (250d), using EOS R dump

```
>python d8_oracle.py fir\250d_CCF20101.FIR
...
Oracle is rom file eosr_110/ROM0.BIN loaded at 0xe0000000
Emulating cipher.bin at 0x200000. Code copied from 0xe0039000
dumping verified and decrypted updater1 (0x800120-0xaf0df0) to file 80000436_1.0.1_updater1.bin
found decryption function called around 0x82c200-0x82c20c
Emulating AES decryption at 0x82c200 within updater1
dumping 80000436_1.0.1_firmware.bin (0xaf0e20-0x2a57160)
```

...then records table can be displayed using dump_fir.py (from ML project)

```
     + tag  + foffset  +   size   + moffset  +    ?
 ----------------------------------------------------------
 0x01: 0x0100 0x000000f8 0x015638a8 0xe0040000 0x00026979
 0x02: 0x0100 0x015639a0 0x001784e0 0xe1bb0000 0x00002666
 0x03: 0x0100 0x016dbe80 0x00063db4 0xe1f50000 0x00000f5c
 0x04: 0x0100 0x0173fc38 0x00000014 0xf0000000 0x00000019
 0x05: 0x0100 0x0173fc50 0x00000364 0xf0350000 0x00000189
 0x06: 0x0100 0x0173ffb8 0x007caecc 0xf05a0000 0x0000ee14
 0x07: 0x0102 0x01f0ae88 0x0000011c 0x00000000 0x00000064
 0x08: 0x0200 0x01f0afa8 0x00000153 0x00000000 0x00000066
 0x09: 0x0200 0x01f0b100 0x0005b237 0x00000000 0x0001999a
```

Main code loaded at 0xe0040_000

DIGIC8 DECRYPTION

LOCATE INTERESTING CRYPTO FUNCTIONS

Looking for crypto constants, and where they are used

- aes_sbox used by key_expansion()
- sha256_k used by sha256_update()

Data Constants

Name	Family	Flags	Address
Rijndael_sbox	AES	0x63-0x7c-0x77-0x7b	0x205288
SHA256_h	SHA256	0x6a-0x09-0xe6-0x67	0x205600
SHA256_K	SHA256	0x42-0x8a-0x2f-0x98	0x205680

This allows identifying these functions which high probability, and arguments could be:

- At 0x204ad0 : Sha256_update(context, data_ptr, data_size)
- At 0x2042dc : Aes_key_expansion(key, key_size, expanded_key)

Let's trace where these functions are called and their arguments values with Unicorn hooking

SHA256 USAGE: 2 CASES SPOTTED

```
FIR_ADDRESS = 0x800000
if func1( FIR_ADDRESS, 0x205784, 0x20, 0x2057a4, 0x20 ):
    func2( FIR_ADDRESS, 0xbfe00100, 0x100, 0x2057C4, 0x10 )
```

verify

decrypt

seed1

seed2

```
Input is update file fir\EOSR0180.FIR
  allocating 0x2071c00 bytes at 0x800000 for FIR file
Oracle is rom file roms\eosr_110.BIN loaded at 0xe0000000
Emulating cipher.bin at 0x200000. Code copied from 0xe0039000
  204ad0: sha256_update  R1/data=800000 R2/size=20 R0/ctx=f000000
  204ad0: sha256_update  R1/data=800024 R2/size=44 R0/ctx=f000000
  204ad0: sha256_update  R1/data=800100 R2/size=2f0f50 R0/ctx=f000000
  204d74: decrypt  R1=bfe00100 R2=100 R0=800000 R3=2057c4
  204ad0: sha256_update  R1/data=bfe00100 R2/size=100 R0/ctx=f13d7d8
  204ad0: sha256_update  R1/data=2057c4 R2/size=10 R0/ctx=f13d7d8
  2042dc: aes_key_expansion  R1=100e64 R2=10 R0=2057d4
  Updater decrypted ? True
  dumping verified and decrypted updater1 (0x800120-0xaf1050) to file 80000424_1.8.0_updater1.bin
  found decryption function called around 0x82b2ac-0x82b2b8
Emulating AES decryption at 0x82b2ac within updater1
  dumping 80000424_1.8.0_firmware.bin (0xaf1080-0x2871bc0)
  decryption successful ? True
```

Inside func1()

Inside func2/decrypt()

Offset and size from regions table : for regions #0 to #2 : likely used by **verify()** function

AES key generation ?

HOW DECRYPTION IS WORKING (FIRV5/DIGIC8) ?

- Like previously : AES128 CTR, for updater and firmware records (Dmit, 2009).
- D8_key = sha256(bfe00100_seed + 2057c4_seed)[:16]
- IV is at offset +8 in encryption headers (FIR file):

```
---updater1 header---
0x100: encrypted length = 0x2f0c20
0x104: 2f0c20
0x108: b'303ad6011000000020fc180000000000'
0x120: ---encrypted content---
---firmware header---
0x2f0d40: offset to encryption header = 0x10
0x2f0d44: offset to ciphertext = 0x30
0x2f0d48: length, including header(s) (from 0x2f0d40) = 0x1d63a60
-
0x2f0d50: encrypted length = 0x1d63a30
0x2f0d54: 1d63a30
0x2f0d58: b'20fc18000000000000000000900d9800'
0x2f0d70: ---encrypted content---
0x20547a0: --end of encrypted content---
```

text inside updater code:
Verify & Decrypt V5

```
"V&D Updater V5"
```

SIGNATURE VERIFICATION

CAN WE FORGE VALID SIGNATURES ?
SIGNATURE SCHEME IS ECDSA WITH SECP256R1

Ponguin user from ML forums first cited secp256r1 algorithm about XF605

```
00205188  char param_r[0x20] = "\xff\xff\xff\xff\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff"
002051a8  char param_a[0x20] = "\xff\xff\xff\xff\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xff\xfc"
002051c8  char param_b[0x20] = "Z\xc65\xd8\xaa:\x93\xe7\xb3\xeb\xbdUv\x98\x86\xbce\x1d\x06\xb0\xccS\xb0\xf6;\xce<>\'\xd2`K"
002051e8  char param_g[0x41] = "\x04k\x17\xd1\xf2\xe1,BG\xf8\xbc\xe6\xe5c\xa4@\xf2w\x03}\x81-\xeb3\xa0\xf4\xa19E\xd8\x98\xc2\x96O\xe3B\xe2\xfe\x1a\x7f\x9b\x8e\xe7"
002051e8      "\xebJ|\x0f\x9e\x16+\xce3Wk1^\xce\xcb\xb6@h7\xbfQ\xf5"
00205229  char data_205229[0x3] = "\x00\x00", 0
0020522c  char param_n[0x20] = "\xff\xff\xff\xff\x00\x00\x00\x00\xff\xff\xff\xff\xff\xff\xff\xff\xbc\xe6\xfa\xad\xa7\x17\x9e\x84\xf3\xb9\xca\xc2\xfcc%Q"
0020524c  int32_t data_20524c = 0x20
00205250  void* data_205250 = param_r
00205254  int32_t data_205254 = 0x20
00205258  void* data_205258 = param_a
0020525c  int32_t data_20525c = 0x20
00205260  void* data_205260 = param_b
00205264  int32_t data_205264 = 0x41
00205268  void* data_205268 = param_g
0020526c  int32_t data_20526c = 0x20
00205270  void* data_205270 = param_n
```

secp256r1 parameters in EOS R cipher.bin

Start of cipher.bin :

```
FIR_ADDRESS = 0x800000
if verify( FIR_ADDRESS, pubkey_x=0x205784, size_x=0x20, pubkey_y=0x2057a4, size_y=0x20 ):
            decrypt( FIR_ADDRESS, 0xbfe00100, 0x100, 0x2057C4, 0x10 )
```

secp256r1 | Standard curve database

Canon XF605: Digic DV 7 == Digic X rebranded for camcorders?

VERIFICATION, USING SECP256R1 (V5 AND V6)

Hash1 = sha256(regions #0 to #2)

Hash2 = sha256(regions #3)

Signature (R1,S1) at offsets 0x6c and 0x90

Signature (R2,S2) at offsets 0xb4 and 0xd8

Pk = public key for digic 8 or 10

If ecdsa_verification(pk, hash1, r1+s1) then *cipher.bin* will decrypt Updater1

If ecdsa_verification(pk, hash2, r2+s2) then *updater1* will decrypt Firmware records

VERIFICATION TOOL: D810_VERIF.PY

EOS R, v1.8.0, digic8

```
E:\perso\d8_oracle>python d810_verif.py fir\EOSR0180.FIR
{
    "model_id": 2147484708,
    "digic": 8,
    "version": "1.8.0",
    "checksum": 4252544598,
    "l1": 32,
    "r1": 91166556806707211677561475345499150438537835926616945616903020740357461425884,
    "l2": 32,
    "s1": 32077394949076650054306415366813385072401846531199938319407393560046924349828,
    "l3": 32,
    "r2": 68206141546541855554507743411351545857086902995789214269167472289672162694671,
    "l4": 32,
    "s2": 88358059300757836676753194441676221921316375016539663271750803666707972036481,
    "h1": "c7bece906b07711ca31996667e908f545935a3ea45cadde396f4cb95b5cd4e9c",
    "h2": "e85456e6bfd83b1213abb341bbb573e04b0c98d84f099f0d2500bd7e3891046a",
    "v1": true,
    "v2": true
}
```

EOS R6, v1.2.0, digic10

```
F:\d8_oracle>python d810_verif.py fir\EOSR6120.FIR
{
    "model_id": 2147484755,
    "digic": 10,
    "version": "1.2.0",
    "checksum": 2616006900,
    "l1": 32,
    "r1": 47792633328137182841597573660849596701047492904304937443793298288611044409372,
    "l2": 32,
    "s1": 42667954688985036105556263335159907808619858313438828049114817024814805705529,
    "l3": 32,
    "r2": 37498477888346825957785337362870872913532388735764960821251675480355619508103,
    "l4": 32,
    "s2": 57760211935964340613755306752130085042128545048977959178103793819193124656744,
    "h1": "67749f5cb22f937ab9b1a329f4df631f5f1701e4ac7c72a153b57e526f2eb262",
    "h2": "56a0b625912f793968671f025ca61f428ca61b8ccd094488cc4bb55f64531dc1",
    "v1": true,
    "v2": true
}
```

This tool extracts and verifies ECDSA/SEC256R1 values for Digic 8 and Digic 10 cameras

- v1 and v2 are verification results

of respectively signatures r1+s1 (header+updaters) and r2+s2 (firmware records)

- h1 and h2 are sha256 values
- l1 to l4 are length of r and s values (seen 31)

250d, version 1.0.1

```
{
    "model_id": 2147484726,
    "digic": 8,
    "version": "1.0.1",
    "checksum": 4000511107,
    "l1": 31,
    "r1": 431034614637480789116775637149267034282128259901249387417281580330575904535,
    "l2": 32,
    "s1": 68530585671853584657428157948998879673772189405141080406342755968204684937504,
    "l3": 32,
    "r2": 33119336651409145883008707454585610900633303581515523407942885403180151926064,
    "l4": 32,
    "s2": 37023851437049816882720123366972611280525313523883516883016746360987088522119,
    "h1": "6c5ab250f016b8e279dc87f8656450a3b2c9665bfedc11c042b5a3fe7d664533",
    "h2": "5681ae68c9872ed0e90c44c9eb9f0d1edba22317eab101f50c92523786bb7148",
    "v1": true,
    "v2": true
}
```

CONCLUSION

Unicorn
The Ultimate CPU emulator

- Unicorn emulation enables decryption of recent digic8 camera updates, given a camera dump from the same Digic generation (because a unique key is used), by using dumped code as Oracle.
  Open source script *d8_oracle.py* demonstrate this (see my github).
- but, we were lucky with emulating the whole cipher.bin, it is usually more difficult.

- We described and experimented version 5 of signature and decryption schemes:
  Canon moved to asymmetric signing scheme : ECDSA/secp256r1 (FIRv4 was HMAC).
  No one can forge FIR signatures anymore without private keys to obtain code exec.
- 2020, EOS R5 release (Digic 10) : Canon changed the secp256r1 pairs and AES key(s).
- AFAIK, Digic 8 decryption key is valid with models R, RP, 250d, G7x m3, 90d.

PREVIOUS WORK AND REFERENCES

- « AES-128 in CTR mode », https://chdk.setepontos.com/index.php?topic=111.msg40313#msg40313, Dmit, February 2009
- FIR format, https://magiclantern.fandom.com/wiki/Firmware_file, Arm.Indy, April 2010
- State of the Latern: 1 year anniversary, Trammel Hudson, June 2010
- Dmitry Sklyarov and co. crack Canon's "image verification" anti-photoshopping tool - Boing Boing, Nov 2010
- EOS firmware in QEMU - development and reverse engineering guide, A1ex https://foss.heptapod.net/magic-lantern/magic-lantern/-/blob/branch/qemu/contrib/qemu/HACKING.rst
- About EOS R encryption, Canon EOS R / RP - Page 2, Alex, Feb 2019
- Update process, Firmware Update/Downdate? - Page 3, Alex, April 2020
- Canon basic examples, GitHub - lclevy/cbasic_examples, January 2022
- Cryptographie et exécution de code sur appareil photo, Laurent Clévy, BeeRump, Sept 2022 https://www.rump.beer/2022/slides/camera_jailbreak_v2_green.pdf

- Magic Lantern discord : https://discord.com/invite/uaY8akC and WWW: Magic Lantern | Home
- Latest news by ML team: News

THANK YOU !

@lorenzo2472.bsky.social

[GitHub - lclevy/d8_oracle: digic8 decryption experiments using emulation]