Check Point
SOFTWARE TECHNOLOGIES LTD

cp<r>
CHECK POINT RESEARCH

Say Cheese

How I Ransomwared Your DSLR Camera

@EyalItkin

DEF CON

Who Am I

- Eyal Itkin
- Vulnerability Researcher
- cp<r> Check Point Research
- Focusing on embedded devices & network protocols

@EyalItkin

Background – Finding a target

- Looking for a new research idea is always tough
- Usually I try several leads
  - Examine new technologies that just got published
  - Attend conferences and hope for a good idea to pop up
  - Ask friends what software / hardware they use regularly

The Idea - Cameras

- Photography – a very common hobby
  - Many potential victims
- People buy expensive cameras, and treat them with care
  - Expensive target – good potential for a monetary gain
- Cameras store memories from special events
  - The victim is emotionally involved – adds drama

The Idea - Cameras

- Once we took over a target DSLR camera, what can we do?
  - IOT bricking – why people still do that in 2019?
  - Espionage tool – it already takes pictures ☺
  - Ransomware – lock the camera and images, and ask for cash 

The Idea - Cameras

- Once we took over a target DSLR camera, wh
  - IOT bricking – why people still do that in 2019?
  - Espionage tool – it already takes pictures ☺
  - Ransomware – lock the camera and images, and as
- Would **you** pay to get your camera back?

Getting Started

Meet our target

- Canon EOS 80D
  - Supports both USB and WiFi
- Canon controls > 50% of the DSLR market
- Canon has a modding community – Magic Lantern

ML
MAGIC LANTERN

Magic Lantern & friends

- Modding communities for Canon cameras
  - https://magiclantern.fm/
  - https://chdk.fandom.com/wiki/CHDK
- Both communities extend the original functionality
  - Researchers reverse engineer the hardware and firmware
  - Developers develop open source features for the cameras

Open Source is great

- Extensive wiki pages that document how the camera works
- Documentation includes important structs and symbols

```
/** Create a new user level task.
 *
 * The arguments are not really known yet.
 */
extern struct task *
task_create(
        const char *            name,
        uint32_t                priority,
        uint32_t                stack_size,
        void *                  entry,
        void *                  arg
);
```

Open Source is great

- Extensive wiki pages that document how the camera works
- Documentation includes important structs and symbols
- Community based, some info is still missing from the docs
- This is a gold mine for our research
- Note: at the time of the research, there is no ML port for our camera

The Attack Vector - PTP

- PTP := IEEE 1588 (Precision Time Protocol)

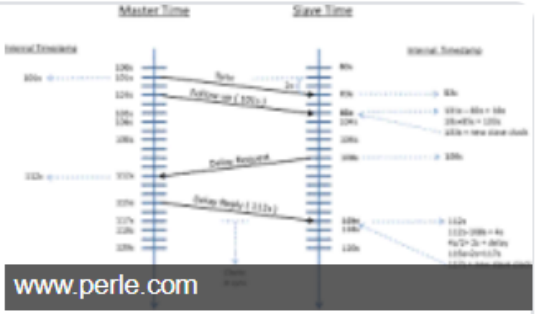
Google

ptp protocol

All Images Videos News Shopping More

About 1,550,000 results (0.58 seconds)

The Precision Time **Protocol** (**PTP**) is a **protocol** used to synchronize clocks throughout a computer network. On a local area network, it achieves clock accuracy in the sub-microsecond range, making it suitable for measurement and control systems.



Precision Time Protocol - Wikipedia

https://en.wikipedia.org/wiki/Precision_Time_Protocol

The Attack Vector - PTP

- PTP := IEEE 1588 (Precision Time Protocol)
- PTP := Picture Transfer Protocol
  - Initially used over USB
  - Now also works as PTP/IP over the WiFi
- Supports a surprisingly high amount of commands

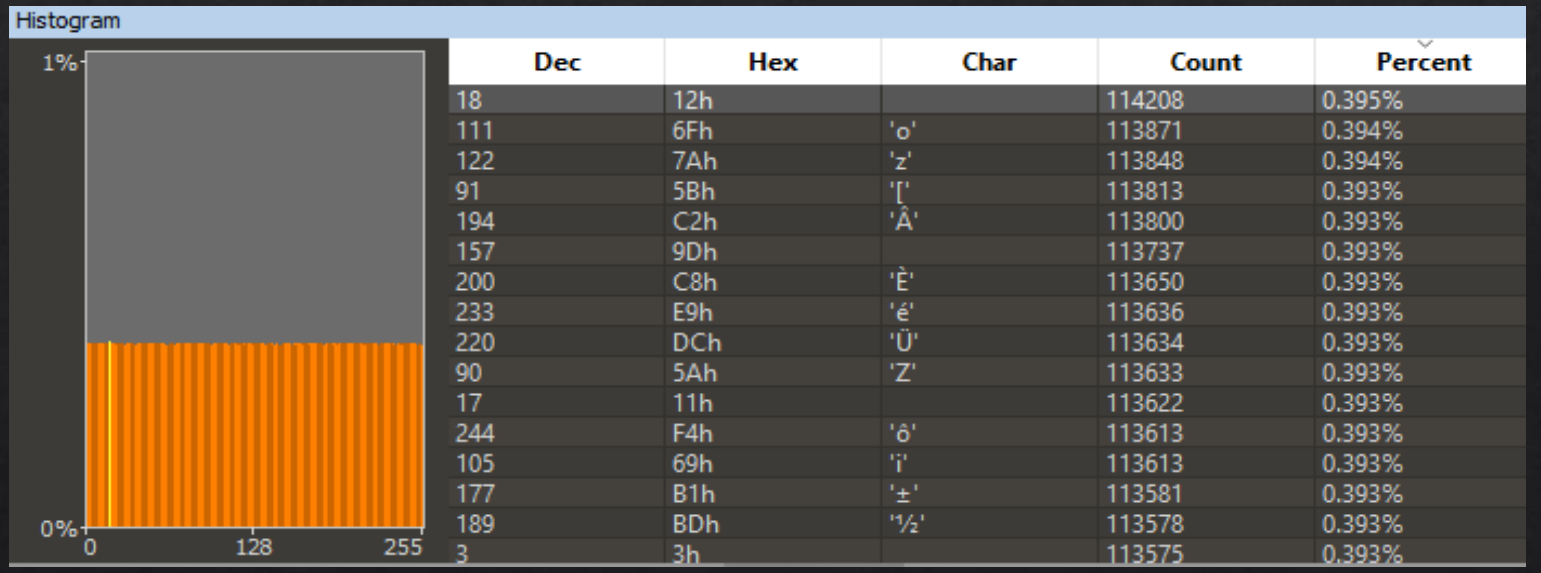
PTP - prior work

- HITB 2013 talk by Daniel Mende (ERNW)
  - https://conference.hitb.org/hitbsecconf2013ams/materials/D1T2%20-%20Daniel%20Mende%20-%20Paparazzi%20Over%20IP.pdf
- Daniel showed it is a naïve network protocol
  - The protocol has no authentication or encryption
- Let’s try to look for vulnerabilities in the implementation ☺

Analyzing the Firmware

Analyzing the firmware

- Classic case - download the `.FIR` file from the vendor's website
- On first glance, we saw a problem
- The byte histogram suggests it is encrypted / compressed



Dec	Hex	Char	Count	Percent
18	12h		114208	0.395%
111	6Fh	'o'	113871	0.394%
122	7Ah	'z'	113848	0.394%
91	5Bh	'['	113813	0.393%
194	C2h	'Â'	113800	0.393%
157	9Dh		113737	0.393%
200	C8h	'È'	113650	0.393%
233	E9h	'é'	113636	0.393%
220	DCh	'Ü'	113634	0.393%
90	5Ah	'Z'	113633	0.393%
17	11h		113622	0.393%
244	F4h	'ô'	113613	0.393%
105	69h	'i'	113613	0.393%
177	B1h	'±'	113581	0.393%
189	BDh	'½'	113578	0.393%
3	3h		113575	0.393%

Analyzing the firmware

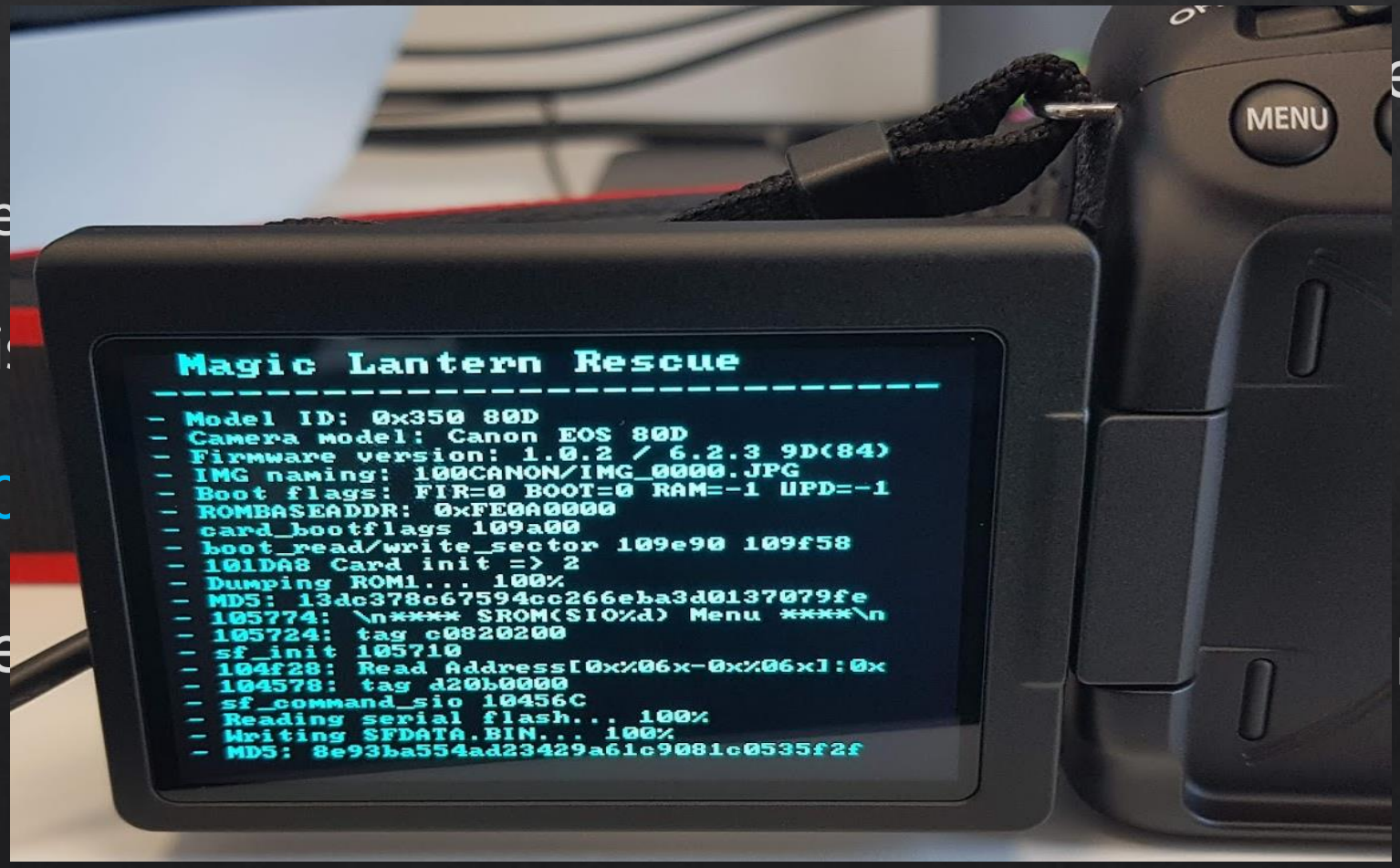
- Classic case - download the `.FIR` file from the vendor's website
- On first glance, we saw a problem
- The byte histogram suggests it is encrypted / compressed
- What Magic Lantern say about this?
  - The firmware is AES encrypted ☹
  - The key isn't available online

Dead End ?

- We have a camera, can we somehow dump the firmware?
  - It is a nice camera, we don’t want to break / open it
  - How ML is researching it?
- Answer: `ROM Dumper`
- ML has the keys, and they created a dumper firmware update

Dead End ?

- We have a
  - It is a nice
  - How ML i
- Answer: RC
- ML has the



Loading it to IDA

- ML even have instructions on how to load the dump into IDA
- Verified basic functions against ML's documented addresses
- Like always, had to manually improve IDA's ARM analysis
  - You don't have to do it yourself
- For this purpose we developed `Thumbs Up`

Thumbs Up

- IDA plugin that uses Machine Learning to improve the analysis
  - https://research.checkpoint.com/thumbs-up-using-machine-learning-to-improve-idas-analysis/
- Designed as a preprocess phase for Karta (matching open sources)
- Before:

- After:

DryOS

- Proprietary Real-Time operating system
- Developed by Canon especially for their cameras
- Looks like any other RTOS out there
- Pros: The camera reboots in 3 seconds !
- This is going to be useful when we debug our exploit

Breaking PTP

Phase 1 – Locate the code

- The PTP protocol is command-based (Request/Response)
  - Every command has a unique identifier (opcode)
- “Thanks to an anonymous .net programmer, here are the PTP commands from the SDK: ” (From ML’s dev group)
  - https://groups.google.com/forum/#!topic/ml-devel/EFLby-U-vy0

Phase 1 – Locate the code

- The PTP protocol is command-based (Request/Response)
  - Every command has a unique identifier (opcode)
- “Tha PTP com
  - ht

```
public const int PtpOpe_CLOSE_SESSION = 0x1003;
public const int PtpOpe_DELETE_OBJECT = 0x100b;
public const int PtpOpe_DS_CANCELTRANSFER = 0x9118;
public const int PtpOpe_DS_CREAGE_OBJECT = 0x9112;
public const int PtpOpe_DS_DELETE_OBJECT = 0x9105;
public const int PtpOpe_DS_FAPI_MESSAGERX = 0x91ff;
public const int PtpOpe_DS_FAPI_MESSAGETX = 0x91fe;
public const int PtpOpe_DS_FORMAT_STORE = 0x9106;
```

Phase 1 – Locate the code

- The PTP protocol is command-based (Request/Response)
  - Every command has a unique identifier (opcode)
- “Thanks to an anonymous .net programmer, here are the PTP commands from the SDK: ” (From ML’s dev group)
  - https://groups.google.com/forum/#!topic/ml-devel/EFLby-U-vy0
- To locate the code, just search for the constants in the firmware

Phase 1 – Locate the code

- Found a massive function with a unique code pattern
- Trace strings assured us that we found the right functions
- Recap for now:
  - 148 unique handlers (!)
  - One of them is surely going to be vulnerable

Phase 1 – Locate the code

- Found a
- Trace str ions
- Recap fo
  - 148 uni
  - One of

```
loc_FE561204
MOV             R0, R5
BL              sub_FE5630B4
LDR             R1, =(sub_FE565034+1)
MOVS            R2, #0
MOVW            R0, #0x1002 ; 0x1002 := Open Session
BL              sub_FE439A54
LDR             R1, =(sub_FE5652A6+1)
MOVS            R2, #0
MOVW            R0, #0x1003 ; 0x1003 := Close Session
BL              sub_FE439A54
LDR             R1, =(sub_FE564D7C+1)
MOVS            R2, #0
MOVW            R0, #0x1001 ; 0x1001 := Get Device Info
BL              sub_FE439A54
LDR             R1, =(sub_FE565374+1)
```

Phase 1 – Locate the code

- Found a
- Trace str ions
- Recap fo
  - 148 uni
  - One of

```
loc_FE561204
MOV             R0, R5
BL              sub_FE5630B4
LDR             R1, =(sub_FE565034+1)
MOVS            R2, #0
MOVW            R0, #0x1002 ; 0x1002 := Open Session
BL              register_ptp_handler
LDR             R1, =(sub_FE5652A6+1)
MOVS            R2, #0
MOVW            R0, #0x1003 ; 0x1003 := Close Session
BL              register_ptp_handler
LDR             R1, =(sub_FE564D7C+1)
MOVS            R2, #0
MOVW            R0, #0x1001 ; 0x1001 := Get Device Info
BL              register_ptp_handler
LDR             R1, =(sub_FE565374+1)
```

Phase 2 – Understand the API

- Each PTP handler inherits from the same API
  - Each request includes 0-5 arguments (integer type)
  - The request can also include an input data buffer
- The handlers use a `ptp_context`
  - The context contains fptrs

Phase 2 – Understand the API

- Each PTP handler inherits from the same API
  - Each request includes 0-5 arguments (ir
  - The request can also include an input da
- The handlers use a `ptp_context`
  - The context contains fptrs

```
ptp_context     struc ; (sizeof=0x2C,

handle          DCD ?
send_data_ptr   DCD ?
recv_data_ptr   DCD ?
send_resp_ptr   DCD ?
get_data_size_ptr DCD ?
send_unexpected_response DCD ?
transport_controller DCD ?
transport_controller_func_3 DCD ?
transport_controller_func_4 DCD ?
transport_controller_func_5 DCD ?
transport_controller_func_6 DCD ?
ptp_context     ends
```

Phase 2 – Understand the API

- Started with 148 unique cmd handlers
- A quick scan shows that only 38 cmds receive an input buffer
  - A massive reduction, but still better than nothing
- Fuzzing is a problem because the camera tends to crash regularly
- Decided to manually analyze the cmds for shallow vulnerabilities

Phase 3 – Finding some Vulns

- Found 3 RCE vulnerabilities on the first scan
  - CVE-2019-5994, CVE-2019-5998, CVE-2019-5999
- 2 of the vulnerabilities are in Bluetooth-related
  - Our camera model doesn't even support Bluetooth
- CVE-2019-5998 is a stack-based buffer overflow, let's exploit it

Building an Exploit

CVE-2019-5998

```
input_size = ctx->get_data_size_ptr(ctx->handle);
/* Preparing a buffer of size 0x120 */
bzero(local_buffer, 0x120);
PTP_FillUpResponse(&msg, args);
/* EI-DBG: Using input_size, with no checks against the buffer's size */
if (ctx->recv_data_ptr(ctx->handle, local_buffer, input_size) >= 0)
{
    COM_NotifyBtStatus(input_size, param1, local_buffer);
}
some_fptr = global_ptp_memory_context->adapterStatusCallback;
if ( some_fptr )
{
    some_fptr(global_ptp_memory_context->fptr_108, 12, param1);
}
result = PTP_SendResponse(ctx, &msg) < 0;
if ( result )
{
    ... // error handling
    result = 1;
}
return result;
```

No size check:
Classic Buffer Overflow

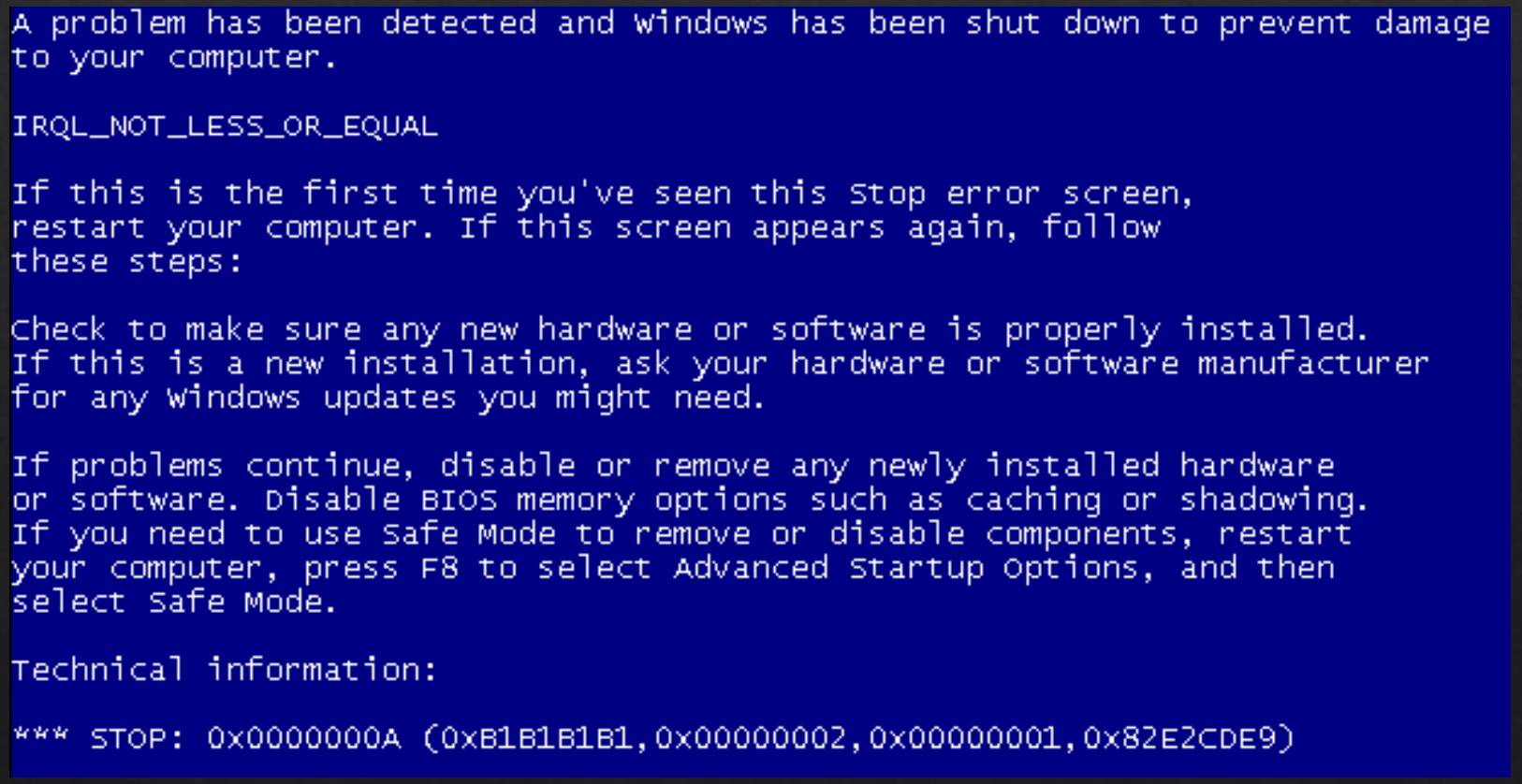
Mystery Callback

Building the Exploit

- At this point we only have a black box, with NO debugger
- In such cases I prefer to use “Sleep based” debugging
  - Add a call to sleep(5) during the exploit
  - Check if the code crashed before the sleep
  - Move this “breakpoint” around to check where the exploit failed

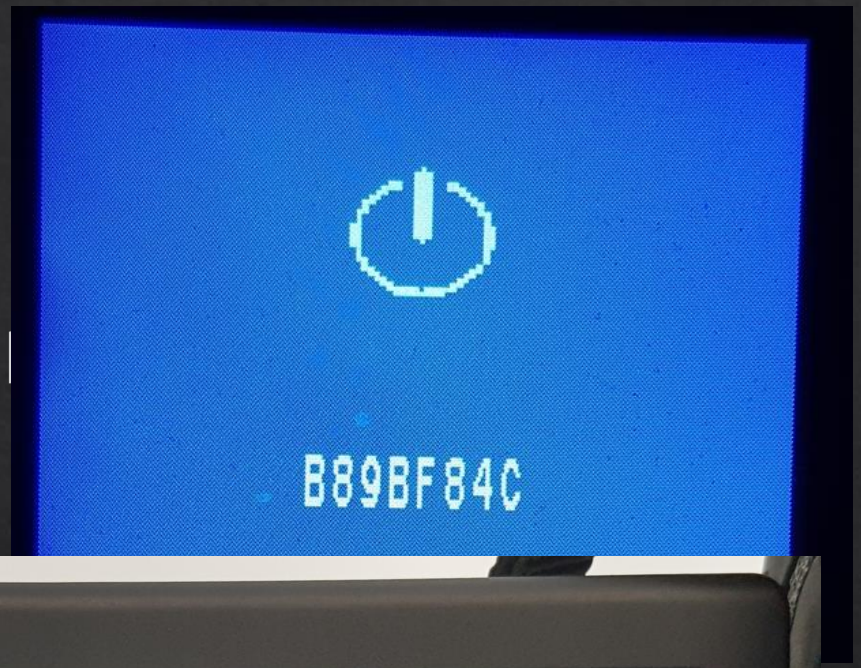
And, it crashed

- On Windows (kernel) we will see a BSOD upon crash



And, it crashed

- On Windows (kernel) we will see a BSOD u
- On IoT devices it will be similar
  - OfficeJet printer Blue Screen
  - Camera Err 70
- After each crash we issue a restart



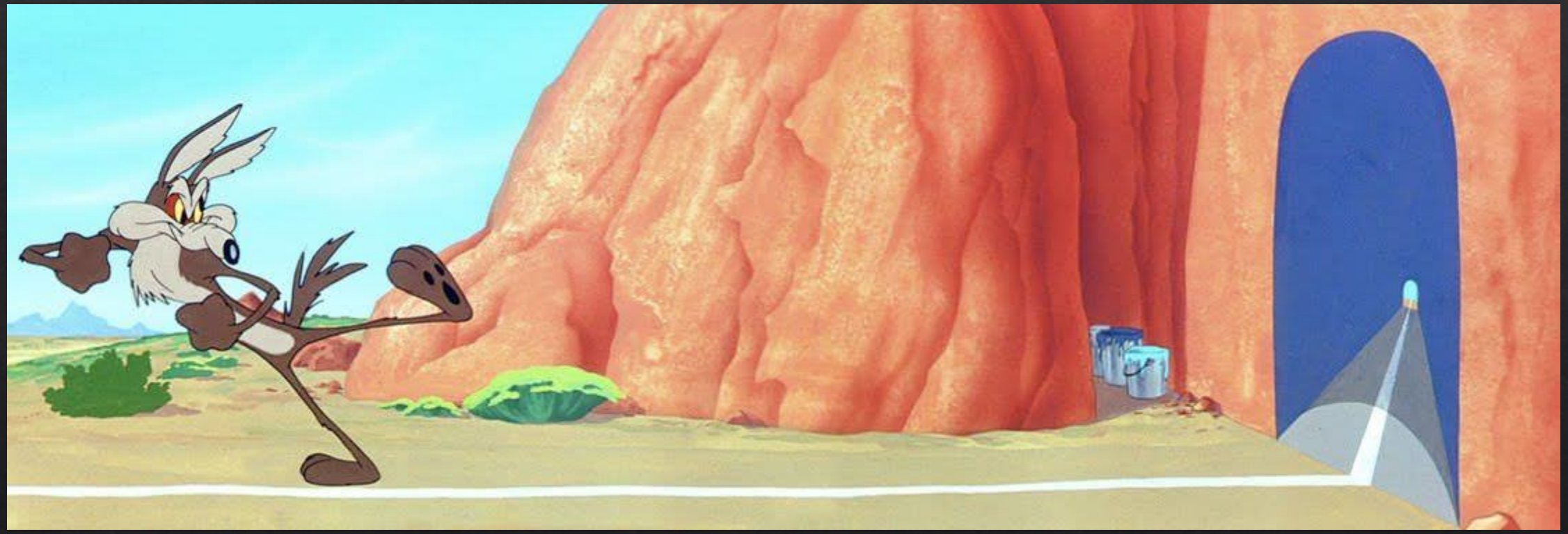
And, it hangs ?!

- For some reason, the camera usually hangs instead of crashing
- It means we can’t see our breakpoint
- We want a trace point that triggers an event we can see
- Let’s switch roles

```
0x000022B6
```

  - New breakpoint method: call an address that always crashes
  - If we see nothing, we hanged before our breakpoint

And, it hangs ?!



- If we see nothing, we hanged before our breakpoint

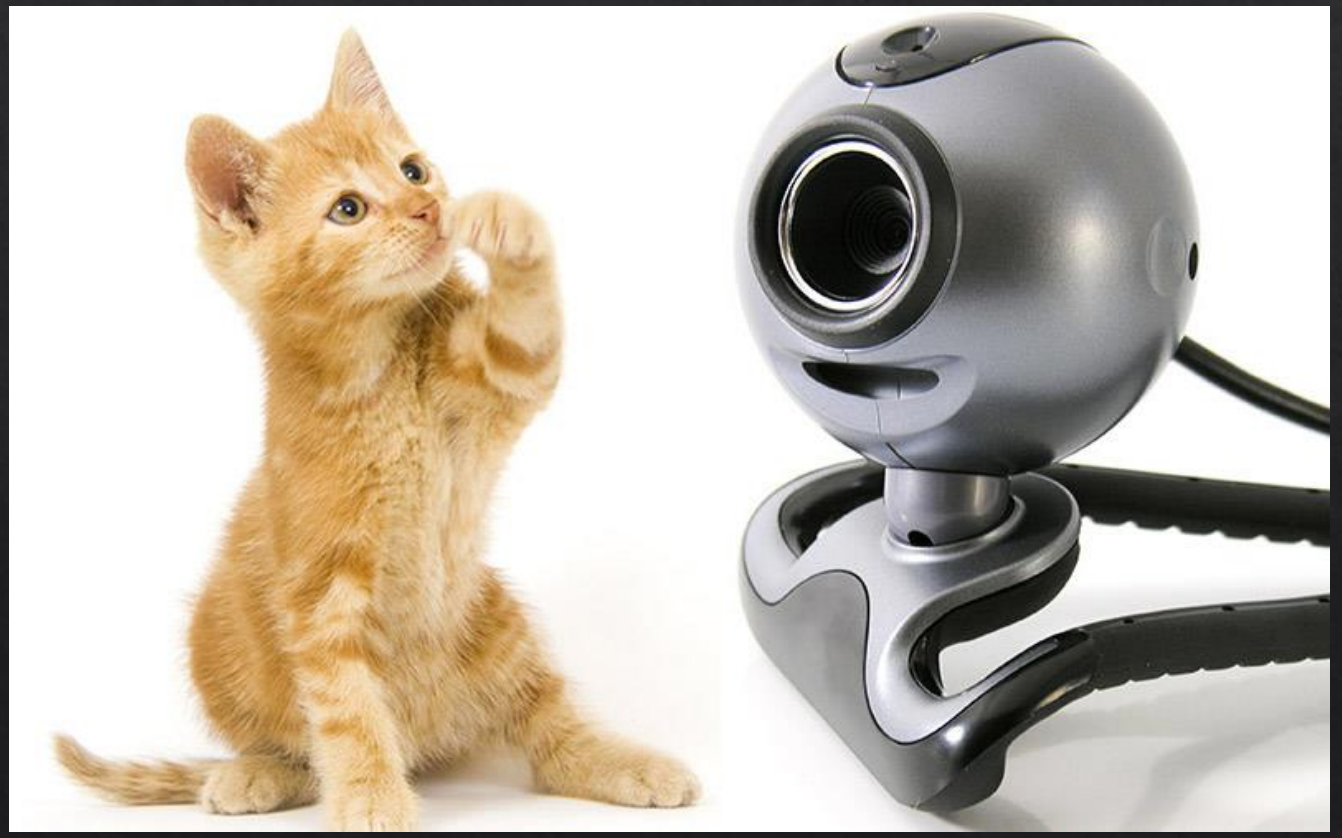
Deploying Scout

- Blind crash-based debugging worked, we have an RCE ☺
- Usually I load Scout debugger to start dynamic RE
- Until now, used PTP over USB
- Scout's TCP doesn't work, it seems like it is USB ^ WiFi
  - We can't use TCP if the USB is connected ☹

Deploying Scout

- Blind crash- CE ☺
- Usually I loa
- Until now, u
- Scout's TCP iFi
  - We can't use TCP if the USB is connected ☹

Migrating to WiFi



PTPy needs help

- Up until now I've used PTPy
  - https://github.com/Parrot-Developers/sequoia-ptpy
- Didn't work out-of-the-box, but was better than nothing
- For the WiFi it needed additional fixes
- We have a PTP vulnerability, it should work over WiFi right?

Bluetooth ^ WiFi

- Earlier we exploited a vulnerability in a Bluetooth related message
  - Now it hangs instead of executing code
  - The WiFi SoC hangs when we remind him he doesn’t support Bluetooth

Bluetooth ^ WiFi

- Earlier we exploited a vulnerability in a Bluetooth related message
  - Now it hangs i
  - The WiFi SoC h support Bluetooth

We need more vulnerabilities

- Time for a second scan over the PTP handlers
  - The first scan only identified shallow bugs
  - Maybe we missed something
- Turns out we did
  - CVE-2019-6000
  - CVE-2019-6001

We need more vulnerabilities

- Time fo
  - The fir
  - Maybe
- Turns o
  - CVE-2
  - CVE-2

```
input_size = ctx->get_data_size_ptr(ctx->handle);
/* Size check - expecting a fixed size message */
if ( input_size != 100 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      3,
      " PTP_SendHostInfo ReceiveSizeError [%x]",
      input_size);
    /* EI-DBG: No return? illegal packets will be logged, and that's it! */
}
ctx->recv_data_ptr(ctx->handle, local_msg_buffer, input_size);
COM_SendHostInfo(local_buffer);
if ( PTP_SendResponse(ctx, &msg) < 0 )
{
    dbg_printf(
      global_ptp_memory_context->some_mem_id,
      6,
      "PTP_SendHostInfo USB Send Error");
}
return result;
```

Vulnerability Recap

- Found 5 vulnerable PTP handlers
  - All PoCs work over USB
  - Only 3 of them work over WiFi
- We demonstrated an RCE from USB and WiFi
- The End?

Goal: Ransomware

- Attackers are profit maximizers
  - We want to develop a ransomware
  - We don’t want to implement the crypto on our own
  - Don’t implement your own crypto – steal it from someone else ☺
- The firmware update mentioned something about AES
- Time to use Scout, debug the camera, and find the crypto funcs

Looking for Crypto

Firmware Update - Design

- The firmware has two **symmetric** (AES) keys
  - Sign / Verification key
  - Encryption / Decryption key
- Signature is HMAC based (again, symmetric)
- In search of the crypto functions, we also found the **keys**

Firmware Update - Design

- The firmware has
  - Sign / Verification
  - Encryption / Decry
- Signature is HMA
- In search of the cr d the **keys**

Firmware Update - Crypto

- Proprietary key derivation that IDA can't even decompile

```
for ( i = 0; i
  v30[i] = sub_
v30[0] += 0x76F
v9 = sub_FE01A4
v10 = sub_FE01A
v34 = v10;
v33 = v11;
v12 = sub_FE01A
v31 = HIDWORD(v
v32 = v12;
v13 = sub_FE019
v14 = sub_FE019
v30[v14] += 0x6
v15 = sub_FE01A
v16 = sub_FE019
v17 = sub_FE019
v30[v17] += 0x7
v18 = sub_FE019
v19 = sub_FE019
v20 = sub_FE019
v22 = 0;
v30[v20] += 0x378FFDF0;
if ( v5 > 0 )
{
```

```
const_1 = 1;
const_2 = 2;
EI_Crypt_Sha1_init(&hash_ctx);
EI_Crypt_Sha1_digest(&hash_ctx, buffer, length);
EI_Crypt_Sha1_digest(&hash_ctx, &const_1, 1);
EI_Crypt_Sha1_finalize(pOutput, &hash_ctx);
EI_Crypt_Sha1_init(&hash_ctx);
EI_Crypt_Sha1_digest(&hash_ctx, buffer, length);
EI_Crypt_Sha1_digest(&hash_ctx, &const_2, 1);
EI_Crypt_Sha1_finalize(&hash_digest, &hash_ctx);
EI_Crypt_memcpy(pOutput + 20, &hash_digest, 0xCu);
```

```
25);

+ __CFSHL__(result, 25) + v2;
```

Firmware Update - Crypto

- Proprietary key derivation that IDA can’t even decompile
- Encryption: OFB’ mode of operation
  - Looks like Output Feedback Block (OFB)
  - Doesn’t match python’s crypto library…
- Trying to implement this simply felt wrong

Firmware Update - Attack

- Implemented a Scout instruction that calculates the crypto
  - No need to understand all the proprietary key derivation
  - Just call their functions and relax
- The camera will calculate the correct signature for us
  - Also implemented a command to encrypt / decrypt the update

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Malicious Update

Compromised Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Malicious Update

Compromised Camera

- Attack (CVE-2019-5995):

Malicious Update

Target Camera

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Malicious Update

Compromised Camera

- Attack (CVE-2019-5995):

Malicious Update

Target Camera

No User interaction Needed!

Firmware Update - Attack

- Preprocess (At home):

Firmware Update

Plain Firmware

Malicious Firmware

Malicious Update

Compromised Camera

- Attack (CVE-2019-5995):

Malicious Update

Compromised Camera!

No User interaction Needed!

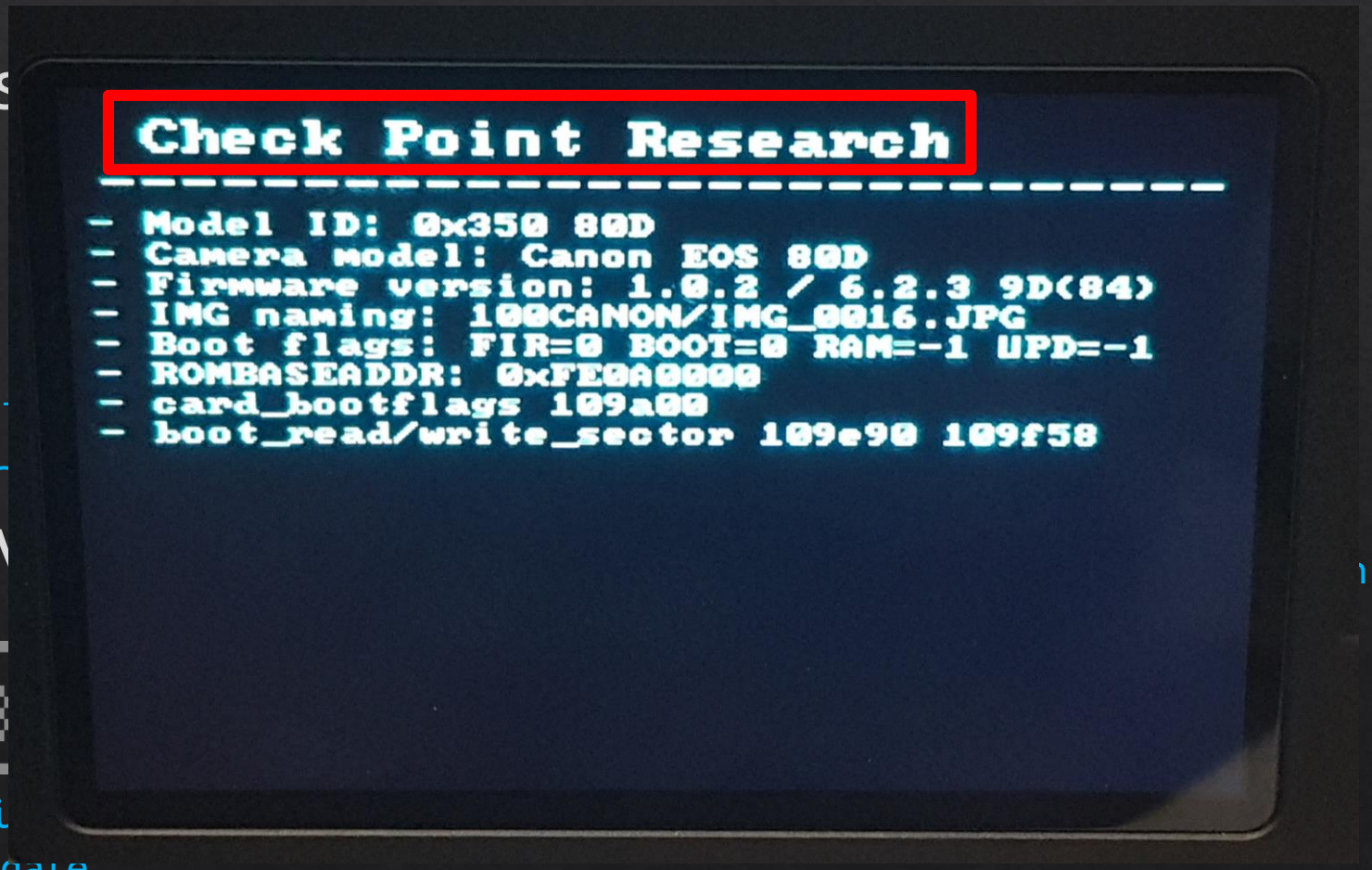
Firmware Update - Attack

- Preproces

Firmware Update

- Attack (CV

Mali Update



Connecting the dots

- Once we finished with the firmware update, we returned back to our ransomware
- Implemented the logic to encrypt the pictures on the SD-Card
- Time for a demo ☺

Ransomware Demo

Responsible Disclosure

- All vulnerabilities were reported to Canon on 31/03/2019
- Canon confirmed the vulnerabilities and issued a patch
  - Full details & Advisory in our blog post:
  - https://research.checkpoint.com/say-cheese-ransomware-ing-a-dslr-camera
- Please don’t ask me for the crypto keys
  - I don’t want ML to work hard on extracting new ones

Conclusions

- Found many vulnerabilities in the PTP implementation
  - Might also apply for other vendors
- The PTP protocol has no network level protection
  - Anyone can send messages and try to attack the camera
- Canon implemented proprietary key derivation for their crypto
  - Bypassing it was easy – used our debugger to invoke their functions

Shameless Self Promotion

- Karta – IDA Plugin for matching open sources in binaries
  - https://github.com/CheckPointSW/Karta
- Thumbs Up – Using Machine Learning to improve IDA's analysis
  - https://github.com/CheckPointSW/Karta/tree/master/src/thumbs_up
- Scout – Embedded instruction-based debugger
  - https://github.com/CheckPointSW/Scout

That’s all folks

cp<r>
CHECK POINT RESEARCH

@EyalItkin

eyalit@checkpoint.com