Say Cheese
How I Ransomwareed Your DSLR Camera

@EyalItkin
Who Am I

- Eyal Itkin
- Vulnerability Researcher
- cp<red> 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, what do 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 ransom
- 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
Magic Lantern & friends

✧ Modding communities for Canon cameras

✧ [https://magiclantern.fm/](https://magiclantern.fm/)

✧ [https://chdk.fandom.com/wiki/CHDK](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

```c
/** 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)

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
The Attack Vector - PTP

◊ PTP := 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)
- 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

### Table

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Count</th>
<th>Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1h</td>
<td>'a'</td>
<td>114206</td>
<td>0.393%</td>
</tr>
<tr>
<td>111</td>
<td>6f</td>
<td>'e'</td>
<td>113871</td>
<td>0.394%</td>
</tr>
<tr>
<td>122</td>
<td>7ah</td>
<td>'Z'</td>
<td>113848</td>
<td>0.394%</td>
</tr>
<tr>
<td>91</td>
<td>5bh</td>
<td>'f'</td>
<td>113013</td>
<td>0.393%</td>
</tr>
<tr>
<td>194</td>
<td>2ah</td>
<td>'A'</td>
<td>113800</td>
<td>0.393%</td>
</tr>
<tr>
<td>157</td>
<td>9dh</td>
<td>'E'</td>
<td>113747</td>
<td>0.392%</td>
</tr>
<tr>
<td>200</td>
<td>c8h</td>
<td>'E'</td>
<td>113650</td>
<td>0.393%</td>
</tr>
<tr>
<td>233</td>
<td>e9h</td>
<td>'e'</td>
<td>113636</td>
<td>0.393%</td>
</tr>
<tr>
<td>220</td>
<td>dch</td>
<td>'U'</td>
<td>113634</td>
<td>0.393%</td>
</tr>
<tr>
<td>90</td>
<td>5ah</td>
<td>'Z'</td>
<td>113633</td>
<td>0.393%</td>
</tr>
<tr>
<td>17</td>
<td>1fh</td>
<td>'o'</td>
<td>113622</td>
<td>0.393%</td>
</tr>
<tr>
<td>244</td>
<td>f4h</td>
<td>'o'</td>
<td>113613</td>
<td>0.393%</td>
</tr>
<tr>
<td>105</td>
<td>6ch</td>
<td>'o'</td>
<td>113613</td>
<td>0.393%</td>
</tr>
<tr>
<td>177</td>
<td>b1h</td>
<td>'t'</td>
<td>113591</td>
<td>0.393%</td>
</tr>
<tr>
<td>199</td>
<td>bdh</td>
<td>'t'</td>
<td>113578</td>
<td>0.393%</td>
</tr>
<tr>
<td>3</td>
<td>3h</td>
<td>'Z'</td>
<td>113575</td>
<td>0.393%</td>
</tr>
</tbody>
</table>
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 nice camera?
◊ It is a nice camera?
◊ How ML is ML?
◊ Answer: ROM
◊ ML has the update
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/](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](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)

◊ "That's how PTP commands are defined in the PTP spec"
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](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:
  ```assembly
  loc_FE561204
  MOV   R0, R5
  BL    sub_FE5630B4
  LDR   R1, =(sub_FE565034+1)
  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_FE5654D7C+1)
  MOVS  R2, #0
  MOVW  R0, #0x1001 ; 0x1001 := Get Device Info
  BL    sub_FE439A54
  LDR   R1, =(sub_FE565374+1)
  ```

- Trace strategy:

- Recap for:

- 148 units:

- One of:
Phase 1 – Locate the code

- Found a
- Trace str
- Recap for
- 148 uni
- One of

```assembly
loc_FE561204
MOV     R0, R5
BL      sub_FE5630B4
LDR     R1, =(sub_FE565034+1)
MOVS    R2, #0
MOVS    R0, #0x1002 ; 0x1002 := Open Session
MOVS    register_ptp_handler
BL      LDR
LDR     R1, =(sub_FE5652A6+1)
MOVS    R2, #0
MOVS    R0, #0x1003 ; 0x1003 := Close Session
MOVS    register_ptp_handler
BL      LDR
LDR     R1, =(sub_FE564D7C+1)
MOVS    R2, #0
MOVS    R0, #0x1001 ; 0x1001 := Get Device Info
MOVS    register_ptp_handler
BL      LDR
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 (in)
✧ The request can also include an input data buffer
✧ The handlers use a `ptp_context`
✧ The context contains fptrs

```c
struc
  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
- 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**

```c
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

A problem has been detected and Windows has been shut down to prevent damage to your computer.

IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical information:

*** STOP: 0x0000000A (0xB1B1B1B1,0x00000002,0x00000001,0x82E2CDE9)
And, it crashed

- On Windows (kernel) we will see a BSOD with a specific code.
- 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
  - 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
▷ Usually I load info
▷ Until now, use USB
▷ Scout’s TCP
▷ 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 in a loop
✧ The WiFi SoC has no 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

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 / Decryption
- Signature is HMAC
- In search of the correct keys

JACKPOT!!!
Firmware Update - Crypto

Proprietary key derivation that IDA can’t even decompile

```c
for (i = 0; i < 10; i++) {
    v30[i] = sub;
    v30[i] = sub;
    v30[8] += 0x76F;
    v9 = sub FE01A;
    v10 = sub FE01A;
    v34 = v10;
    v33 = v11;
    v12 = sub FE01A;
    v31 = HIRDWORD(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] += 0x370FDF0;
    if (v5 > 0 )
```
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

diamond Preprocess (At home):

- Firmware Update
- Compromised Camera
Firmware Update - Attack

◊ Preprocess (At home):

- Compromised Camera
- Plain Firmware Update
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

❖ Attack (CVE-2019-5995):

- Malicious Update
- Target Camera
- Compromised Camera
Firmware Update - Attack

◎ Preprocess (At home):
  - Firmware Update
  - Plain Firmware
  - Malicious Firmware
  - Malicious Update

◎ Attack (CVE-2019-5995):
  - Malicious Update
  - Target Camera
  - No User interaction Needed!
Firmware Update - Attack

◊ Preprocess (At home):
  - Compromised Camera
  - Firmware Update
  - Plain Firmware
  - Malicious Firmware
  - Malicious Update

◊ Attack (CVE-2019-5995):
  - Compromised Camera!
  - No User interaction Needed!
  - Malicious Update
Firmware Update - Attack

- Preprocessing

- Check Point Research

- Model ID: 0x350 80D
- Camera model: Canon EOS 80D
- Firmware version: 1.0.2 / 6.2.3 9D(84)
- IMG naming: 109CANON/IMG_0016.JPG
- Boot flags: FIR=0 boot=0 RAM=-1 UPD=-1
- ROMBASEADDR: 0xFEDA0000
- card_bootflags 109a00
- boot_read/write_sector 109e90 109f58

- Attack (CWE-311)

- Compromised Firmware Update

- Malicious Update

- No User interaction Needed!
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

Your pictures have been encrypted!

We are White-hat hackers, don’t worry 😊

A malicious actor would have taken over your camera, encrypting all of your images for ransom. To stay protected, update the firmware of your camera.

Payment will be released on
00:00:00

Your file will be lost on
00:00:00

@EyalItkin

eyalit@checkpoint.com