Exploiting Digital Cameras

Oren Isacson - oren]at[coresecurity.com
Alfredo Ortega - aortega]at[coresecurity.com

Exploit Writers Team
Core Security Technologies

August 1, 2010
This talk is about:

- How to script Canon Powershot cameras.
- How we reversed the embedded interpreter.
- What are the possibilities of this?
- What are the security consequences?
Architecture of Powershot cameras

- ARM type Processor (ARM946E-S based)
- Memory Protection Unit (MPU)
- No Memory Mapping Unit
- Exception handlers
- SD Memory Card
- Debugging Support
- Proprietary OS (DryOS)

We used MPU’s registers to find the memory regions
And exception handlers for debugging.
Previous works: CHDK

- CHDK is an unofficial firmware enhancement.
- Can be booted from the memory card.
- Loads as a firmware update but it doesn’t make permanent changes.
- It doesn’t automatically load unless the Memory Card is locked.
- But users don’t normally have Memory Card in locked mode.
- So it’s safe (not a good attack vector).

We used it for researching the firmware.
Image fuzzing

- Can we run code exploiting the image parsers?
- The camera crashes when processing some malformed images.
- We wrote an exception handler to examine the crashes.
- Even if we can exploit this bugs, the exploit would be model specific.
Firmware analysis

- CHDK project provided IDA dumps of some powershot cameras (Thanks!).
- Some Interesting strings:
  - "Syntax Error", "yacc stack overflow", "input in flex scanner failed", etc.
  - It appears that there is an embedded interpreter.
  - Flex lexical scanner and yacc or bison parser generator were used.
The embedded interpreter

- We are not the first ones to find the interpreter
- But there is no public documentation on the language
- Invalid scripts make the camera shut down.
- And there are no helpful error messages.
Running a script

- Script file: “extend.m”
- String “for DC_scriptdisk” must be in file “script.req”
- String “SCRIPT” in offset 0x1f0 of the memory card.
- Memory card can be either FAT32, FAT16 or FAT12 (unlike in the firmware update boot method).
- Script is launched when ”Func. Set” key is pressed in playback mode.
- **It works even when the memory card is in unlocked mode.**
- We need to reverse the interpreter.
• Standard yacc/lex (Bison/flex) parser:

```
private sub Initialize()
...
end sub
PRINT
Script \extend.m
print()
LEDDrive()
IFWHILE  
yylex() yyparse()
(yacc generated)(flex generated)
getch() Get next token
LoadScript("\extend.m")
```
Locating `yyparse()`

- `yyparse()` is the grammatical parser, it calls the lexer `yylex()`.
Locating yylex()

yylex() function:

- Flex is a regex-based tokenizer (Lexical analyzer)
- The regex implementation is a table-based state machine
- Over 220 states and 50 different tokens.
Reversing `yylex()`

Flex state-machine based regex parser:

```c
yy_match:
    do
        {
            register YY_CHAR yy_c = yy_ec[YY_SC_TO_UI(*yy_cp)];
            if ( yy_accept[yy_current_state] )
                {
                    yy_last_accepting_state = yy_current_state;
                    yy_last_accepting_cpos = yy_cp;
                }
            while ( yy_chk[yy_base[yy_current_state] + yy_c] != yy_current_state )
                {
                    yy_current_state = (int) yy_def[yy_current_state];
                    if ( yy_current_state >= 76 )
                        yy_c = yy_meta[(unsigned int) yy_c];
                }
            yy_current_state = yy_nxt[yy_base[yy_current_state] + (unsigned int) yy_c];
            ++yy_cp;
        }
    while ( yy_base[yy_current_state] != 271 );
```

- Let's emulate it in x86! we only need to find the tables.
Finding Flex tables

```
static const short int yy_base[517] = {
0,
0, 0, 43, 46, 49, 50, 69, 0,
161, 0, 205, 0, 2351, 2350, 2349, 2348,
295, 332, 377, 0, 426, 0, 475, 0,
};
```

Location in firmware of `yy_base[]`:

<table>
<thead>
<tr>
<th>Location</th>
<th>FF6E328</th>
<th>FF6E338</th>
<th>FF6E348</th>
<th>FF6E358</th>
<th>FF6E368</th>
<th>FF6E378</th>
<th>FF6E388</th>
<th>FF6E398</th>
<th>FF6E3A8</th>
<th>FF6E3B8</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF6E328</td>
<td>02 00 00 00 02 00 00 00 02 00 00 00 01 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E338</td>
<td>01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E348</td>
<td>95 00 5B 00 00 00 00 00 8B 00 23 00 26 00 00 00 00 60 00 37 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E358</td>
<td>00 00 00 97 00 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E368</td>
<td>00 00 00 00 00 00 00 00 00 8B 00 23 00 26 00 00 00 60 00 37 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E378</td>
<td>00 00 00 31 00 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E388</td>
<td>00 00 00 00 00 00 00 00 00 8B 00 23 00 26 00 00 00 60 00 37 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E398</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E3A8</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FF6E3B8</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 93 00 35 00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Exploiting Digital Cameras
Dumping Tokens

- Find all the tables and rebuild the equivalent Flex parser
- Try all different combination of inputs (Exit on “unknown” token)
- Brute force time!

Works every time
It’s Basic!

<table>
<thead>
<tr>
<th>#</th>
<th>Token</th>
<th>#</th>
<th>Token</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>/</td>
<td>8</td>
<td>^</td>
</tr>
<tr>
<td>9</td>
<td>&gt;&gt;</td>
<td>10</td>
<td>&lt;&lt;</td>
</tr>
<tr>
<td>11</td>
<td>==</td>
<td>13</td>
<td>&gt;=</td>
</tr>
<tr>
<td>15</td>
<td>&lt;=</td>
<td>16</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>19</td>
<td>(</td>
<td>20</td>
<td>)</td>
</tr>
<tr>
<td>23</td>
<td>%MEMORY_L</td>
<td>24</td>
<td>%MEMORY_M</td>
</tr>
<tr>
<td>25</td>
<td>%MEMORY_S</td>
<td>26</td>
<td>&quot;</td>
</tr>
<tr>
<td>42</td>
<td>if</td>
<td>42</td>
<td>If</td>
</tr>
<tr>
<td>42</td>
<td>IF</td>
<td>43</td>
<td>sub</td>
</tr>
<tr>
<td>43</td>
<td>Sub</td>
<td>43</td>
<td>SUB</td>
</tr>
<tr>
<td>44</td>
<td>function</td>
<td>44</td>
<td>Function</td>
</tr>
<tr>
<td>45</td>
<td>do</td>
<td>45</td>
<td>Do</td>
</tr>
<tr>
<td>45</td>
<td>DO</td>
<td>47</td>
<td>for</td>
</tr>
<tr>
<td>47</td>
<td>For</td>
<td>47</td>
<td>FOR</td>
</tr>
</tbody>
</table>
Emulating the parser

Now we know (most of) the Tokens. We need to emulate the parser. We used:

- QEMU: processor emulator with ARM support
- GNU Binutils: for working with memory images
- Our exception handler: for dumping camera memory
- CHDK: for loading our exception handler and writing to memory card.
- GDB: debugger for setting initial CPU state and monitoring.
Dumping the memory

- QEMU can’t emulate the whole camera.
- So we need a memory dump at the parser entry point.
- We can’t set breakpoints.
- but we can force a memory address exception
- Setting the static variable yy_start to 0xA0A0A0A0, the last line raises an exception:

```c
static yy_start = 1;

...;

yy_current_state = yy_start;

do
{
    YY_CHAR yy_c = yy_ec[*yy_cp];
    if ( yy_accept[yy_current_state] )
```
Dumping the memory

Exception handler:

- Stores CPU registers
- Stores memory image
- MPU registers tells us memory regions

<table>
<thead>
<tr>
<th>offset</th>
<th>size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>32 MB</td>
</tr>
<tr>
<td>0x0</td>
<td>2 GB</td>
</tr>
<tr>
<td>0x2000</td>
<td>8 KB</td>
</tr>
<tr>
<td>0x10000000</td>
<td>32 MB</td>
</tr>
<tr>
<td>0x40000000</td>
<td>4 KB</td>
</tr>
<tr>
<td>0xc0000000</td>
<td>16 MB</td>
</tr>
<tr>
<td>0xffc00000</td>
<td>4 MB</td>
</tr>
</tbody>
</table>

- We only need 2 areas: Main memory at offset 0 (32MB) and ROM at offset 0xffff0000 (4MB)
Loading the emulator

- QEMU can load ELF format files.
- We used Binutils objcopy and objdump to make ELF file from memory dump.
- QEMU has an internal GDB server.
- We use it for setting initial register state.
- We fix the invalid variable so it doesn’t generate an exception inside the emulator.
Running the emulator

- As not all hardware is emulated, we can’t allow the emulated code to make system calls.
- The flex generated scanner uses a macro to read input: YY_INPUT.
- Using GDB python integration, it’s easy to replace this macro.
- The lexical scanner will continue to call YY_INPUT until it returns 0.
- Or until an error is found.
- We used this to find out the camera script syntax.
Emulating scripts

Sample emulation runs:

Dim a as Long ← Error
a=1

Dim a
a=1 ← Error

Dim a=1 No Errors
Emulating scripts

sub test() ← Error
end sub
test()

private sub test()
end sub
test() ← Error

declare test()

private sub test2()
test()
end sub No Errors
Entry point function has to be called “Initialize“.
HelloWorld script:

```vba
private sub sayHello ()
    a=LCDMsg_Create ()
    LCDMsg_SetStr (a, "Hello World!")
end sub

private sub Initialize ()
    UI.CreatePublic ()
    sayHello ()
end sub
```
• We found over 500 functions controlling all aspects of the camera
• We documented some of them and made a (very incomplete) user guide
• Exploiting_Digital_Cameras_IOBasic.pdf
Launching exploits from the camera

- Example 1: Launch common auto-run exploits against the SD (MS08-038, CVE-2010-2568)
- Example 2: Activate the microphone!
- Example 3: Output data via Exif Tags
Countermeasures

- Check that there are no .REQ or .BIN files in the SD card before inserting into the camera.
- Camera can’t be infected by using USB-PTP, malware can’t access root filesystem.
- AntiVirus can’t scan cameras by USB-PTP.
The end

Thank you!