Exploiting Digital Cameras

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

Exploit Writers Team
Core Security Technologies

June 30, 2010

Introduction

This talk is about:

- How to script Canon Powershot cameras.
- How we reversed the embedded interpreter.
- What you can possibly do with this.
- 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.

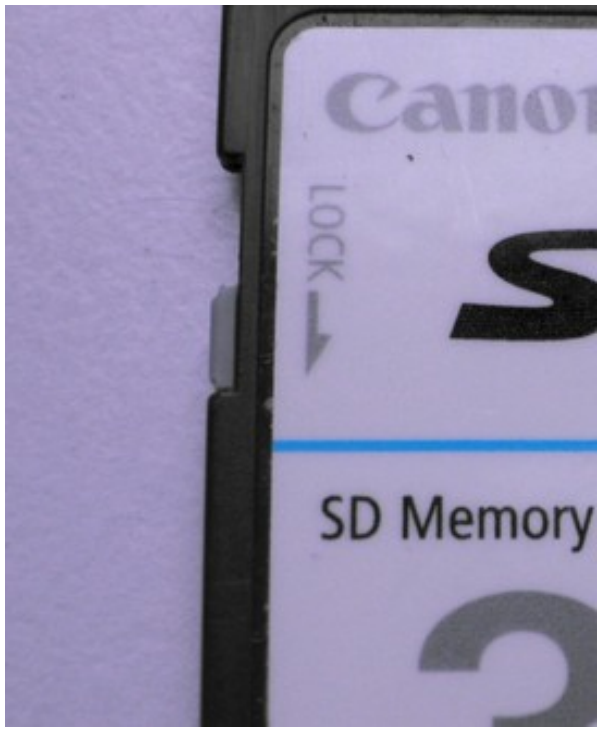
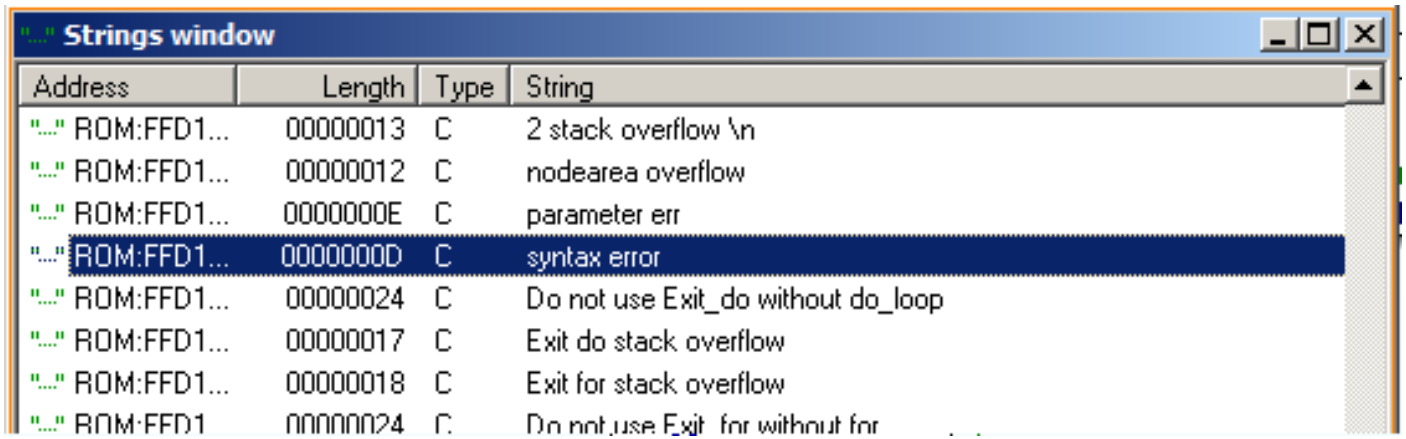


Image fuzzing

- 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

- IDA Pro was used to analyze the firmware of one camera
- 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.

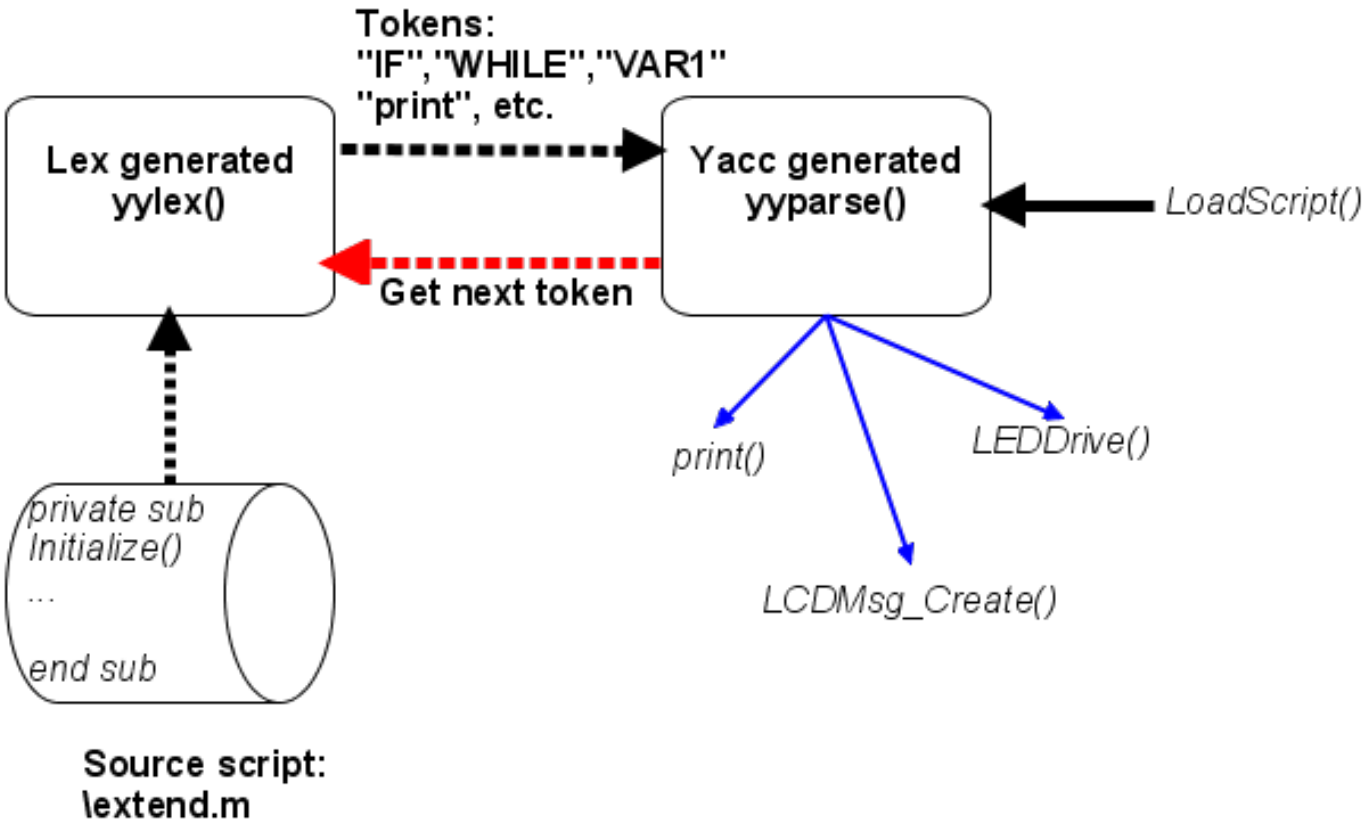
The embedded interpreter

- 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

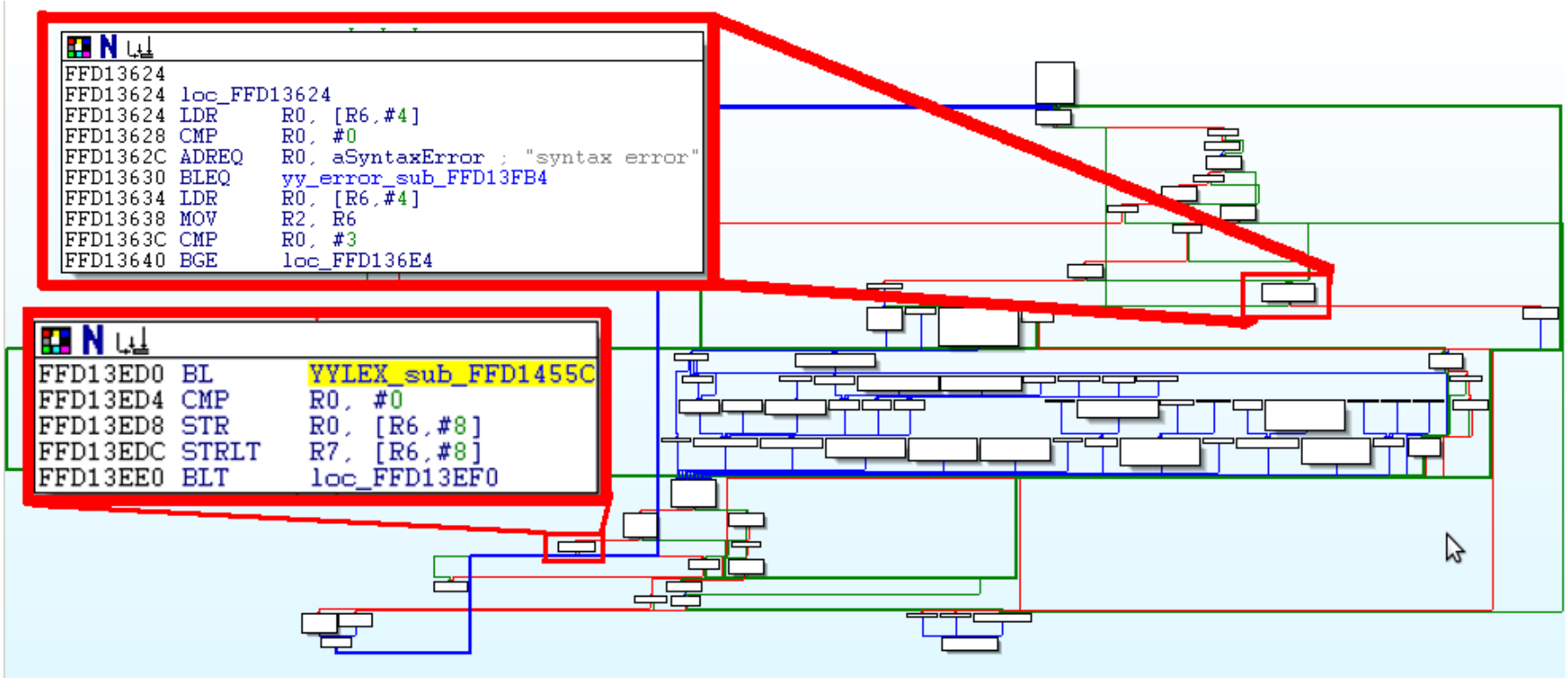
- Script file: “extend.m”
- String “for DC_scriptdisk” must be in file “script.req”
- String “SCRIPT” in offset 0x1f0 of the memory card.
- Memory card partition must be formatted to FAT16 or FAT12.
- Script is launched when ”Func. Set” key is pressed in playback mode.
- **It works even when the memory card is in unlocked mode.**
- We need to reverse the interpreter.

- Standard yacc/lex (Bison/flex) parser:



Locating yyparse()

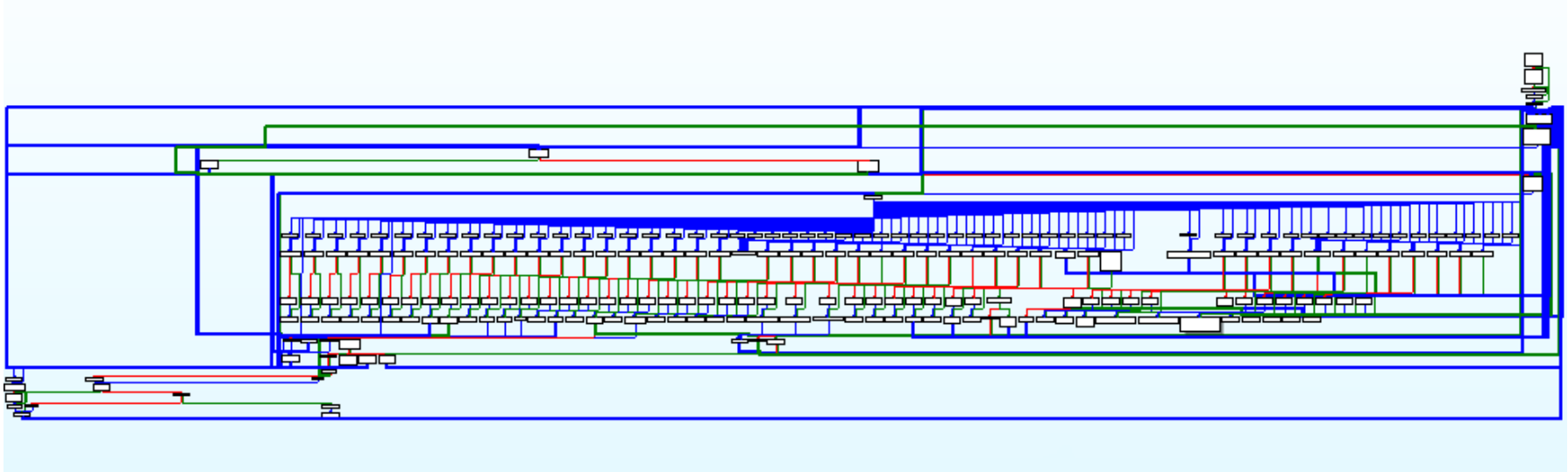
yyparse() function:



- 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.

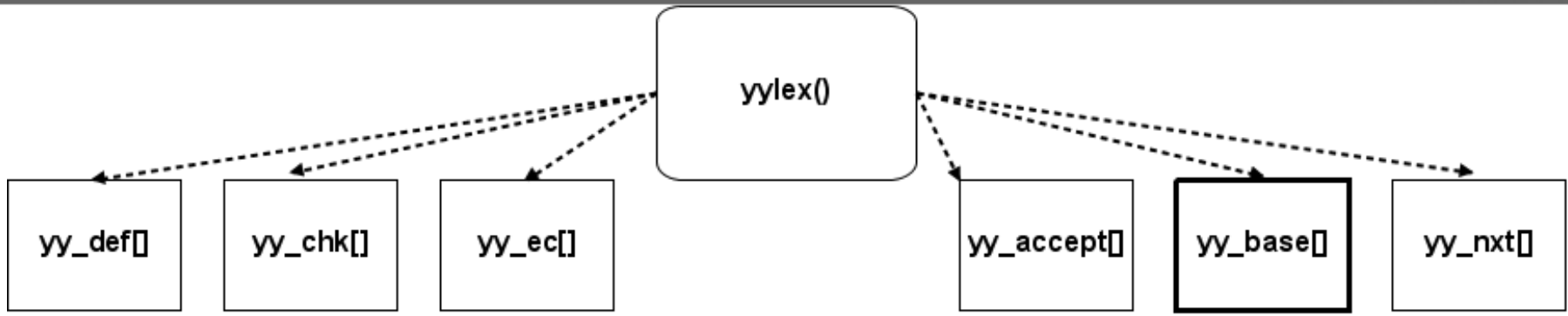
Reversing yylex()

Flex state-machine based regex parser:

```
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,  118,    0,
      161,    0,  205,    0, 2351, 2350, 2349, 2348,  254,    0,
      295,  332,  377,    0,  426,    0,  475,    0, 2368, 2373,
     2373, 2373,   43,   50,   69, 2373,    0, 2373,   67, 2373,
     2373,   74, 2341, 2325,    0, 2373, 2373,   80, 2373, 2373,
```

Location in firmware of *yy_base*[]:

```
FFE6E328  02 00 00 00 02 00 00 00  02 00 00 00 01 00 00 00  ................
FFE6E338  01 00 00 00 01 00 00 00  00 00 00 00 93 00 35 00  ..............ô.5.
FFE6E348  36 00 96 00 95 00 5B 00  8B 00 23 00 26 00 00 00  6.û.ò.[.ï.#.&...
FFE6E358  00 00 97 00 CF 00 CF 00  CF 00 CF 00 CF 00 CF 00  ..ù.─.─.─.─.─.─.
FFE6E368  00 00 CF 00 CF 00 CF 00  00 00 CF 00 CF 00 CF 00  ..─.─.─.──.─.─.─.
FFE6E378  00 00 CF 00 31 00 34 00  CF 00 34 00 80 00 36 00  ..─.1.4.─.4.Ç.6.
FFE6E388  CF 00 CF 00 CF 00 CF 00  8E 00 CF 00 CF 00 CF 00  ─.─.─.─.Ä.─.─.─.
FFE6E398  8E 00 CF 00 CF 00 CF 00  CF 00 CF 00 60 00 37 00  Ä.─.─.─.─.─.`.7.
FFE6E3A8  33 00 5F 00 65 00 5C 00  21 00 38 00 36 00 5C 00  3._.e.\.!.8.6.\.
FFE6E3B8  5B 00 56 00 00 00 00 00  00 00 4F 00 00 00 CF 00  [.V.......O...─.
FFE6E3C8  CF 00 CF 00 CF 00 CF 00  CF 00 67 00 CF 00 57 00  ─.─.─.─.─.g.─.W.
FFE6E3D8  CF 00 6B 00 5A 00 54 00  58 00 CF 00 5F 00 4F 00  ─.k.Z.T.X.─._.O.
FFE6E3E8  65 00 53 00 4B 00 51 00  00 00 5E 00 4C 00 CF 00  e.S.K.Q...^.L.─.
```

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

#	Token	#	Token
1	+	2	-
4	/	8	^
9	>>	10	<<
11	==	13	>=
15	<=	16	<>
19	(	20	)
23	%%MEMORY_L	24	%%MEMORY_M
25	%%MEMORY_S	26	"
42	if	42	If
42	IF	43	sub
43	Sub	43	SUB
44	function	44	Function
45	do	45	Do
45	DO	47	for
47	For	47	FOR

Emulating the parser

Now we know (most of) the Tokens. Now we are going 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.
- But we can't set breakpoints.
- But we can force an memory address exception
- Setting the static variable yy_start to 0xA0A0A0A0, the last line raises an exception:

```
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:

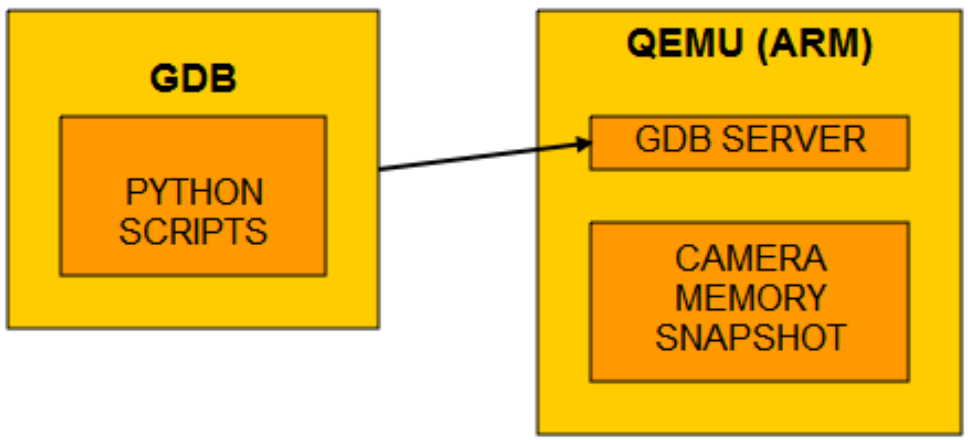
- Stores CPU registers
- Stores memory image
- MPU registers tells us memory regions

offset	size
0x0	32 MB
0x0	2 GB
0x2000	8 KB
0x10000000	32 MB
0x40000000	4 KB
0xc0000000	16 MB
0xffc00000	4 MB

- 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.

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
```

```
private sub test()
end sub
private sub test2()
      test()
end sub No Errors
```

Hello World

Entry point function must be called “Initialize“.
HelloWorld script:

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

private sub Initialize()
    UI.CreatePublic()
    sayHello()
end sub
```

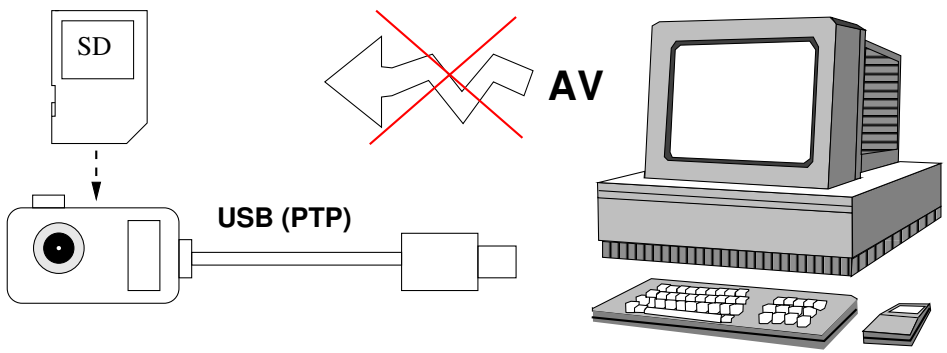
- We found over 200 functions controlling all aspects of the camera
- We documented some of them and made a (very incomplete) user guide
- Exploiting_Digital_Cameras_IOBasic.pdf

Launching exploits from the camera

- We have complete control, but limited reach
- Example 1: Launch common auto-run exploits against the SD (MS08-038)
- 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

Thanks you!