



Exploiting Digital Cameras

Oren Isacson - [oren\]at\[coresecurity.com](mailto:oren]at[coresecurity.com)
Alfredo Ortega - [aortega\]at\[coresecurity.com](mailto: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?

- 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.



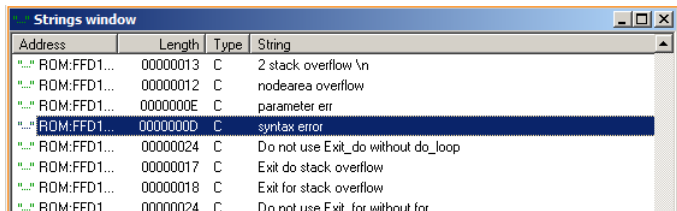
- CHDK is an unofficial firmware enhancement.
- Can be booted from the memory card.
- Loads as a firmware update but it doesn't make permanent changes.
- It doesn't automatically load unless the Memory Card is locked.
- But users don't normally have Memory Card in locked mode.
- So it's safe (not a good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

We used it for researching the firmware.



- 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.

- CHDK project provided IDA dumps of some powershot cameras (Thanks!).
- Some Interesting strings:



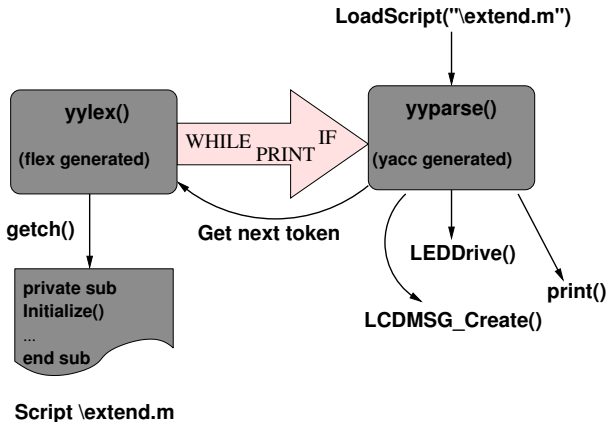
Address	Length	Type	String
ROM:FFD1...	00000013	C	2 stack overflow \n
ROM:FFD1...	00000012	C	nodearea overflow
ROM:FFD1...	0000000E	C	parameter err
ROM:FFD1...	0000000D	C	syntax error
ROM:FFD1...	00000024	C	Do not use Exit_do without do_loop
ROM:FFD1...	00000017	C	Exit do stack overflow
ROM:FFD1...	00000018	C	Exit for stack overflow
ROM:FFD1...	00000024	C	Do not use Exit for without for

- “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.

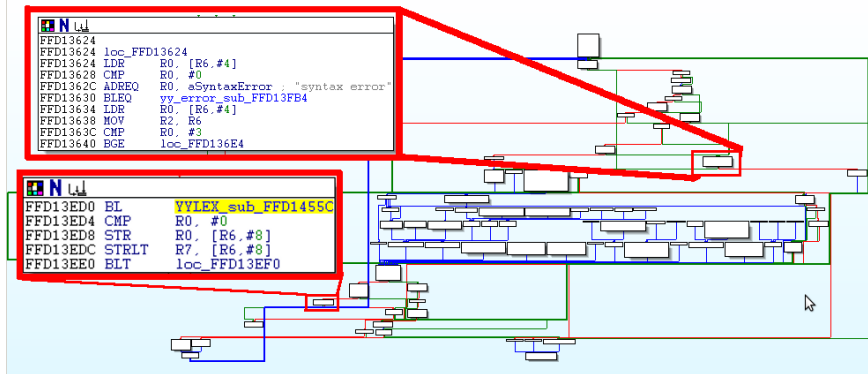
- We are not the first ones to find the interpreter
- But there is no public documentation on the language
- Invalid scripts make the camera shut down.
- And there are no helpful error messages.

- 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:

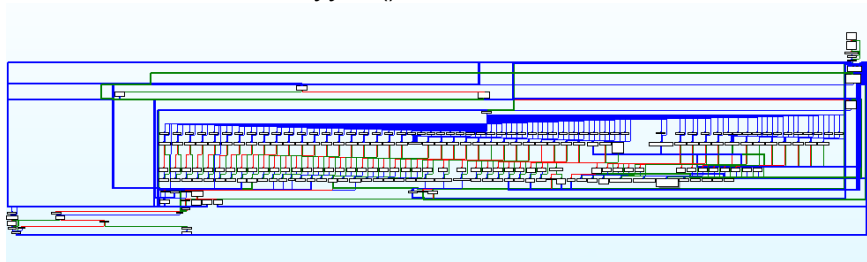


yyparse() function:



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

yylex() function:

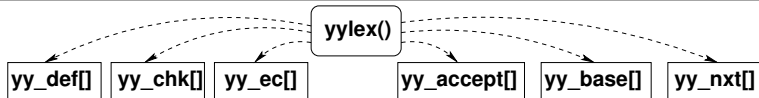


- Flex is a regex-based tokenizer (Lexical analyzer)
- The regex implementation is a table-based state machine
- Over 220 states and 50 different tokens.

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.



```

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[]:

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 00ô.5.
FFE6E348	36 00 96 00 95 00 5B 00	8B 00 23 00 26 00 00 00	6.â.ò.[.i.#.&...
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....-

#	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

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.

- 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:

```
static yy_start=1;
[...]  
yy_current_state = yy_start;  
do  
{  
    YY_CHAR yy_c = yy_ec[*yy_cp];  
    if ( yy_accept[yy_current_state] )
```

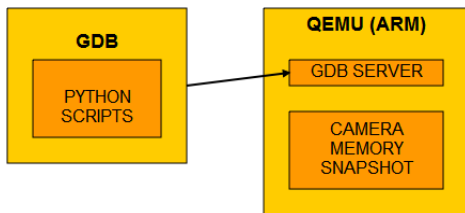
Exception handler:

- 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)

- 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.



- 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
```

```
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
```

Entry point function has to be called "Initialize".

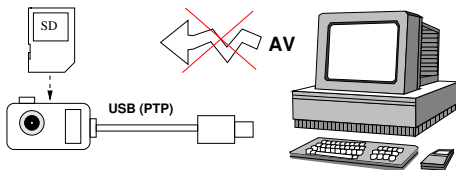
HelloWorld script:

```
private sub sayHello ()  
    a=LCDMsg_Create ()  
    LCDMsg_SetStr (a, " Hello_World! ")  
end sub  
  
private sub Initialize ()  
    UI.CreatePublic ()  
    sayHello ()  
end sub
```

- We found over 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

- 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

- 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!