HARDCORE PHOTOGRAPHY:

HOW I HACKED MY DSLR
WHO ARE YOU?

@therealshodan
WARNING

I’M ABOUT TO DO SOMETHING REALLY STUPID
Temperature: 23.60 °

Humidity: 52.60 %

Wireless Temperature: 0 °

Last Updated: Tuesday June 24 - 08:11

+ Lounge
+ Bedroom
+ Fishtanks
- Others

Boiler ON

Boiler OFF
OH
Success.
KEEP CALM AND DON'T BE STUPID
HARDCORE PHOTOGRAPHY:

HOW I HACKED MY DSLR
WHY ON GODS EARTH HACK A CAMERA?

• BECAUSE I CAN

• ADDITIONAL FUNCTIONALITY
PRIOR ART

- CHDK
  - Low End Canon
- Magic Lantern
  - Canon DSLR
- Nikon Hacker
- Samsung Cameras Running Tizen
CHDK – CUSTOM SCRIPTING

GrAnd @ chdk.wikia.com
MAGIC LANTERN – RAW VIDEO

H264 F8 ISO6400     ML RAW F8 ISO6400

http://petapixel.com/2013/05/31/canon-raw-footage-using-magic-lantern-hack-trumps-h-264-at-high-isos/
CHDK – SUPER FAST SHUTTER SPEEDS

This file is copyrighted. The copyright holder of this file allows anyone to use it for any purpose, provided that {{|}}.
WHY BOTHER WITH PENTAX?

• 40 YEARS WORTH OF LENSES

• High end features for cheap

• In-body stabilisation
  • Self cleaning / dust removal
  • Automatic horizon leveling
  • Astrotracer
GOALS

- Don't brick camera, remember the boiler...

- Don't want to open the case
  - Ruin waterproofing
  - Most likely brick it
  - Means no JTAG shenanigans

- Run custom code from SD card
  - New code must augment functionality
  - Need to code in C
HOW DO YOU HACK EMBEDDED DEVICES?

1. **Get firmware**
2. **Run binwalk**
3. **Reverse, reverse, reverse**
4. **Fix checksum**
5. **Flash & cross fingers**
6. **Root shell, code exe, profit???
STEP 1 - GET THE FIRMWARE

• OLD FASHIONED FIRMWARE UPDATES
  • Ensure you check the T&C's!

• JTAG / EEPROM

• MITM OF WEB UPDATES

• SD CARD
HOW DO YOU HACK EMBEDDED DEVICES?

1. **Get firmware**
2. **Run binwalk**
3. **Reverse, reverse, reverse**
4. **Fix checksum**
5. **Flash & cross fingers**
6. **Root shell, code exe, profit???
STEP 2: BINWALK

- USUALLY TELLS YOU WHAT TO DO NEXT

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>3876</td>
<td>0xF24</td>
<td>Copyright string: &quot; (C) PENT</td>
</tr>
<tr>
<td>9602135</td>
<td>0x928457</td>
<td>XTF, Microsoft Xbox data</td>
</tr>
</tbody>
</table>
SO YOU'RE TELLING ME

THAT'S AN XBOX

memegenerator.net
BREAKING Obfuscation

- If it’s good crypto - game over
- Assume it’s obfuscated
- Looking for repeating patterns & hope for XOR
- Found closed source tool which did this
STEP 2: BINWALK...AGAIN

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>3876</td>
<td>0xF24</td>
<td>Copyright string: &quot; (C) PENTAX RICOH IMAGING COMPANY</td>
</tr>
<tr>
<td>2878618</td>
<td>0x2BEC9A</td>
<td>mcrypt 2.2 encrypted data, algorithm: blowfish-448,</td>
</tr>
<tr>
<td>2879898</td>
<td>0x2BF19A</td>
<td>mcrypt 2.2 encrypted data, algorithm: blowfish-448,</td>
</tr>
<tr>
<td>2885018</td>
<td>0x2C059A</td>
<td>mcrypt 2.2 encrypted data, algorithm: blowfish-448,</td>
</tr>
<tr>
<td>2906304</td>
<td>0x2D58C0</td>
<td>mcrypt 2.2 encrypted data, algorithm: blowfish-448,</td>
</tr>
<tr>
<td>2966853</td>
<td>0x2D8E5</td>
<td>LZMA compressed data, properties: 0x9A, dictionary</td>
</tr>
<tr>
<td>2967041</td>
<td>0x2D4601</td>
<td>LZMA compressed data, properties: 0x6D, dictionary</td>
</tr>
<tr>
<td>2967149</td>
<td>0x2D466D</td>
<td>LZMA compressed data, properties: 0x9A, dictionary</td>
</tr>
<tr>
<td>3123374</td>
<td>0x2FABAE</td>
<td>LZMA compressed data, properties: 0x8D, dictionary</td>
</tr>
<tr>
<td>3579857</td>
<td>0x369FD1</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>4180582</td>
<td>0x35CA66</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>4182685</td>
<td>0x3FD29D</td>
<td>TIFF image, little-endian</td>
</tr>
<tr>
<td>4182715</td>
<td>0x3FD2BB</td>
<td>TIFF image, little-endian</td>
</tr>
<tr>
<td>4210134</td>
<td>0x403E8A</td>
<td>Copyright string: &quot; (c) 2010, 2011 by FotoNation.</td>
</tr>
<tr>
<td>4210344</td>
<td>0x403EA8</td>
<td>LZMA compressed data, properties: 0x40, dictionary</td>
</tr>
<tr>
<td>4638725</td>
<td>0x46C805</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>4821763</td>
<td>0x499317</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6602658</td>
<td>0x64BFA2</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6609074</td>
<td>0x64DBB2</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6625546</td>
<td>0x69190A</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6628466</td>
<td>0x6E582A</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6661678</td>
<td>0x65A62E</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6713718</td>
<td>0x667176</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6713838</td>
<td>0x6671EE</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6738098</td>
<td>0x66D0B2</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>6758998</td>
<td>0x672256</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>8900101</td>
<td>0x87CE05</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>10556454</td>
<td>0xA11426</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>10629494</td>
<td>0xA23176</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
<tr>
<td>10641382</td>
<td>0xA25FE6</td>
<td>JPEG image data, JFIF standard 1.02</td>
</tr>
</tbody>
</table>
**STEP 2: BINWALK...OPCODES**

<table>
<thead>
<tr>
<th>Address</th>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x10512160</td>
<td>0xA06720</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10512568</td>
<td>0xA068B8</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10513264</td>
<td>0xA06B70</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10517928</td>
<td>0xA07DA8</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10521504</td>
<td>0xA08BA0</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10522212</td>
<td>0xA08E64</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10524196</td>
<td>0xA09624</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10524340</td>
<td>0xA096B4</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10533540</td>
<td>0xA0BAA4</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10540304</td>
<td>0xA0D510</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10541720</td>
<td>0xA0DA98</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10542124</td>
<td>0xA0DC2C</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10549520</td>
<td>0xA0F910</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
<tr>
<td>0x10552500</td>
<td>0xA104B4</td>
<td><strong>ARMEB</strong> instructions, function epilogue</td>
</tr>
</tbody>
</table>
HOW DO YOU HACK EMBEDDED DEVICES?

1. Get firmware
2. Run binwalk
3. **Reverse, reverse, reverse**
4. Fix checksum
5. **Flash & Cross fingers**
6. **Root shell, code exe, profit??**
STEP 3 – REVERSE ENGINEERING - TOOLING

- IDA PRO ADVANCED - $$
- HEX RAYS ARM DECOMPILER - $$$
- 010 HEX EDITOR - $
- QEMU
- DEVELOPMENT UNIT / INSTRUMENTED HARDWARE - $$
- IDA PRO FREE
- HOPPER - $
- SOMETHING ON SOURCEFORGE (XVI32)
- QEMU
- ???
A BIT ABOUT THE CODEBASE

• There is NO operating system – this isn’t Linux

• Firmware appears to be made of three blobs
  • CPU1, CPU2, Debug functionality

• Compiler doesn’t feel like GCC or CLANG
  • Is stupid

• Very few strings!

• Heavy use of C++
  • No STL – thank God!
WHY HARDCORE?

- The RE pyramid of pain

- **HARDCORE**
  - C++ (no strings)
  - C (no strings)
  - C (with strings), ALL Perl
  - Java, C#
  - Python, JavaScript, PHP
v38 = a1;
v39 = a2;
v37 = 0;
v21 = sub_A0007F7C(1024);
v22 = sub_A0007F7C(128);
nullsub_15(&v36);
v23 = sub_A000983C(&v36);
v24 = sub_A0009848(&v36);
v25 = sub_A000FC10(&v32, (const char *)dword_A08E4C04, v24, v23);
v26 = sub_A0146C18(v25);
v27 = sub_A02AC398(v26);
sub_A015C7A0(&v33);
v28 = sub_A015D208(v27, &v32, &v33);
if ( v28 )
{
    v29 = sub_A02B0344(&v33);
v35 = 0;
v28 = (*int (__fastcall **)(int, int *, signed int, int *))(*DWORD *)v27 + 20)(v27);
if ( v28 )
{
    v34 = sub_A0007F7C(v29 + 1);
v30 = *(DWORD *)v27;
*(WORD *)(v2 + v29) = v22;
*(WORD *)(v30 + v3) = v29;}
HARDCORE REVERSE ENGINEERING

- **No quick solutions**

- **Tips**
  - Look for paired functions, New & delete
  - Spend the time understanding all functionality you have any clue about
  - Don't trust IdA for ARM
  - Don't trust IdA for C++

- **Write IdA scripts**
  - Automatically find functions from C++ vtables
  - Gather strings
UNLOCKING IT’S SECRETS – DEBUG BLOB

• The firmware contains another menu system

• Debug functionality to be run in unit testing

• Reverse engineered
  • Camera requires two magic files
  • One filename is scrambled!
HIDDEN DEBUG MENUS

Development Menu1
- DEBUG MODE <EN>
- CARDDOOR OPEN <DIS>
- WRITE CARD VNDR <DIS>
- SCRIPT EN MOD <DIS>
- AUTO TEST MODE <DIS>
- USR DAT CLEAR <OO>
- REGION CODE <00>

[MENU] Cancel [OK] OK

Development Menu2
- SWITCH TEST
- RELEASE AGING <AUTO>
- LENS COM CHECK
- AF PINT DISP
- CAMERA LOG DISP
- BOOT BLOCK INFO

[MENU] Cancel [OK] OK

Development Menu3
- CPU ADJ DAT CAMERA->SD
- CPU ADJ DAT SD=>CAMERA
- DSP ADJ DAT CAMERA->SD
- DSP ADJ DAT SD=>CAMERA
- CAMERA LOG CAMERA->SD

[MENU] Cancel [OK] OK

Development Menu4
- LENS ROM DAT SD=>CAMERA
- LENS ROM DAT CAMERA=>SD
- LENS ROM EDIT R/W
- LENS CPU FWUP

[MENU] Cancel [OK] OK

Development Menu5
- BOARD VER
- DSP CHIP VER
- DUST REDUCTION TEST
- MEDIA TEST
- DISPLAY TEST
- MEMORY TEST
- LV-TEST: DIS N-CLOSE: DIS

[MENU] Cancel [OK] OK

SWS SWR PRV AF GREEN
FLPOP DOOR [CARD] RAW XV
MENU ERASE INFO DISP
4WR 4WL 4WD 4WU 4WOK

FDL:00,00 RDL:00,00
MODE:0E-AP POS:01-U
AFMD:01-AFS
MPIN:52(H), 10010100(B)
this = this_param;
debugSettings = settings_param;
v23 = 0;
buf1 = memory_alloc(1024);
buf2 = memory_alloc(128);
nullsub_15(&v22);
cameraDebugId = GetCameraDebugId();
anotherDebugNumber = GetProductNumber();
sprintf(&fileName, "C:\\%08ld.%03ld", anotherDebugNumber, cameraDebugId);
v6 = fileAPInit();
fileAPI = getFileAPI(v6);
file allocating(&fileBuffer);
wasSuccessful = loadFile(fileAPI, &fileName, &fileBuffer);
if ( wasSuccessful ) {
    fileSize = GetFileSizeFromFileStruct(&fileBuffer);
    settings = 0;
    wasSuccessful = (fileAPI->vtable->OpenFileHandle)(fileAPI, &fileName, 1, &settings);
    if ( wasSuccessful ) {
        newBuffer = memory_alloc(fileSize + 1);
        wasSuccessful = (fileAPI->vtable->ReadFile)(fileAPI, settings, newBuffer, fileSize)
        (fileAPI->vtable->CloseFileHandle)(fileAPI, settings);
HOW DO YOU HACK EMBEDDED DEVICES?

1. Get firmware
2. Run binwalk
3. Reverse, reverse, reverse
4. Fix checksum
5. Flash & cross fingers
6. Root shell, code exe, profit???
STEP 4 - DEFEATING THE CHECKSUM

- **Reverse engineered algorithm**
  - Treats firmware as int[], adds everything together
  - Valid if total == 0

- **Somewhere is a 4 byte checksum value**
  - Couldn’t find it

- **Used the unused space in the firmware**
  - Copyright strings are long and never used
OH NOOOOOOOOOOOOOOOOOO!!!
HOW DO YOU HACK EMBEDDED DEVICES?

1. **Get firmware**
2. **Run binwalk**
3. **Reverse, reverse, reverse**
4. **Fix checksum**
5. **Flash & cross fingers**
6. **Root shell, code exe, profit???
BREAKING OBFUSCATION...AGAIN

• FOUND DEOBFUCATION CODE

• CODE THAT GENERATES THE XOR KEY IS VERY COMPLEX

• IF ONLY I COULD RUN THE DAMN CODE...
NEED THAT DEV UNIT!!!

- IDA Pro Advanced - $$
- Hex rays ARM decompiler - $$$
- 010 hex editor - $
- QEMU
- Development unit / instrumented hardware - $$
- IDA Pro free
- Hopper - $
- Something on SourceForge (XVI32)
- QEMU
- ???

Exists but hard to get
## NEED A DEVELOPMENT BOARD

<table>
<thead>
<tr>
<th>Pentax K30</th>
<th>Raspberry Pi</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM CPU (big endian)</td>
<td>ARM CPU (little endian)</td>
</tr>
<tr>
<td>Lots of RAM</td>
<td></td>
</tr>
<tr>
<td>SD card</td>
<td></td>
</tr>
<tr>
<td>GPIO (buttons...)</td>
<td></td>
</tr>
<tr>
<td>16Mpx CMOS sensor</td>
<td></td>
</tr>
<tr>
<td>Rugged case</td>
<td></td>
</tr>
</tbody>
</table>
GLUE IT TOGETHER

- **TAKE ASSEMBLY FROM IDA, PUT HEADER ON IT, COMPILIE WITH GCC**

- **WRITE C WRAPPER**
  - **LOADS FILE**
  - **CREATES FICTITIOUS POINTERS / PARAMETERS**
  - **DON’T FORGET BIG ENDIAN DATA**

- **YES THIS REALLY WORKS!**
HOW DO YOU HACK EMBEDDED DEVICES?

1. Get firmware
2. Run binwalk
3. Reverse, reverse, reverse
4. Fix checksum
5. Flash & cross fingers
6. Root shell, code exe, profit???
MORE ADVANCED – ARBITRARY STRINGS
REMEMBER THE BOILER

WE STILL HAVEN'T CHANGED ANY ACTUAL CODE
THE PENTAX COMMUNITY TO THE RESCUE

- Ronald L Beckett
- Lawson Stone
- Mark Banas
- Matthew Weinel
- Robert Smith
- Michal Karski
- Verena Reich
- Jason Doss
- Johan Le-Nobel
- John Morris
- Thomas Locke

- Leonids Morozs
- Darren Moon
- Stephan Wehrheim
- Joshua Abbott
- Andy Chien
- Jaroslav Sladek
- Stephan Munsch
- Stefan Herberth
- Mehrdad Momeny Sarrazdeh
- Alessandro Di Vito
- Peter Patoschka
- Francois SEVE
HOW DO YOU HACK EMBEDDED DEVICES?

1. **Get firmware**
2. **Run binwalk**
3. **Reverse, reverse, reverse**
4. **Fix checksum**
5. **Flash & cross fingers**
6. **Root shell, **code exe**, profit???
WHAT CODE TO WRITE

- **Something to help with debugging / RE**

- **Firmware update only gives a small % of what’s going on**

- **Memory dump will help a lot with RE**

- **Could overwrite a debug menu function**
  - **Should be very safe**
```
int dumpMem(void* unusedA, void* unusedB)
{
    unsigned char* buffer = malloc(BUFFER_SIZE);

    FILEREF fileRef = initFileAPI();
    FILECLASS *fileAPI = getFileAPI(fileRef);
    FILEHANDLE handle = 0;

    fileAPI->api->OpenFileHandle(fileAPI, "C:\DUMP.BIN", FILE_WRITE, &handle);

    for (unsigned int memLocation = BASE_ADDR;
         memLocation < END_ADDR;
         memLocation += BUF_SIZE)
    {
        // inline memcpy
        fileAPI->api->WriteFile(fileAPI, handle, buffer, BUFFER_SIZE);
    }

    fileAPI->api->CloseFileHandle(fileAPI, handle);
    return 1;
}
```
NEED A COMPILER

• Need to really understand the CPU
  • ARM isn’t enough

• Vendors hide chips under their own names
  • Pentax PRIME, Nikon Expeed are the same CPU different software

• IDA Python Script goes through every instruction
  • Github in a few days

• Need some ld tweaks to produce a valid binary
COMPILES AND WORKS FIRST TIME.

WHAT DID I DO WRONG?
WHERE NEXT

• **CAN RUN CODE FROM THE DEBUG MENU**
  • CAN’T ACTUALLY USE CAMERA FOR PHOTOS!

• **NEED TO TAKE OVER A MENU DURING NORMAL OPERATION**

• **LOAD CODE FROM SD AND EXECUTE**

• **IF ONLY THERE WAS ANOTHER DEBUG MENU IN CAMERA MODE**
TEST MODE

CAPTURE/SYSTEM TEST
AF TEST
STRINGS DISP TEST
Icon Test
DEBUG MODE DISABLE

MENU Exit
SOFTWARE DEBUG MENU

- **Hidden during normal use**
- **Turned on via magic file on sd**
- **Pointless functionality, much better as our entry point**
- **Patch fw so our code is called instead of icon test**
TEST MODE
CAPTURE/SYSTEM TEST
AF TEST
STRINGS DISP TEST
Run PHDK
DEBUG MODE DISABLE

MENU Exit
PHDK – AVAILABLE TODAY!

• **Full control, via C API**

• **Functionality**
  • Change camera modes
  • Control buttons
  • Access the SD card
  • Display text on screen

• **Available on GitHub**
  • Google PHDK GitHub
PHDK – THE FUTURE

- Arbitrary memory read/write over the USB
  - Looks like Pentax implement other debugging interfaces

- GPS support
  - Mobile phone instead of expensive adapter

- Custom intervalometer
SOME THINGS I’VE LEARNT
DEBUGGING FOR DIFFICULT DEVICES

• **Differential Analysis of Memory Dumps**
  1. **Take Dump**
  2. **Change Feature (ISO etc)**
  3. **Take Dump, Binary Compare, IDA**

• **Dump Registers and Stack**
  • **GCC allows you to alias registers to variables**

```c
register void* r13v asm ("r13"); // sp
void* r13 = r13v;
void* r13 = r13v;
saveFile ("C:\STACK.BIN", r13 - LENGTH, LENGTH);
```
NEVER TRUST YOUR TOOLS COMPLETELY

- **Binwalk results** have a lot of false positives

- **IDA Pro** support for non Intel is patchy
  - ARM / THUMB confusion

- **Decompilers (ASM -> C)** have tons of bugs and always will

- **3rd party scripts / plugins** never just work
  - Always choke on large files / take hours to run
  - Get to know your API's
  - Endian assumptions
QUESTIONS

ONE DOES NOT SIMPLY LEAVE WITHOUT ONE LAST MEME