Administrativia

• Please fill-in the BH13US Feedback Form - Thanks!
• The views of the authors are their own and do not represent the position of their employers or research labs
• By attending this workshop, you agree to use the tools and knowledge acquired only for legal purposes and for activities you have explicit authorization for
About – Jonas Zaddach

- PhD. candidate on "Development of novel binary analysis techniques for security applications" at EURECOM
- Co-founder of FIRMWARE.RE
- jonas@firmware.re
- jonas.zaddach@eurecom.fr
About – Andrei Costin

- PhD. candidate on "Software security in embedded systems" at EURECOM
- Co-founder of FIRMWARE.RE
- Author of MFCUK and BT5-RFID (RFID security)
- Researcher on security of: printers, ADS-B
- andrei@firmware.re
- andrei.costin@eurecom.fr
HIGH LEVEL RESEARCH

€11.3 m
Global budget with a project turnover of €5.3m

283
International scientific publications 121 cosigned with foreign institutions. A 9.2% increase compared to 2011.

18
Average H-Number

EURECOM is a Carnot Institute since 2006

104
Contracts managed in 2012 including
31 European contracts
42 National contracts
40 Industrial contracts
Table: Eurecom Research Results – Publications

<table>
<thead>
<tr>
<th>Year</th>
<th>Total No. of publ.</th>
<th>Cosigned with Ext. Labs</th>
<th>Cosigned with Intl. Labs</th>
<th>Conf.</th>
<th>Journals/Papers</th>
<th>Books/Chapters</th>
<th>Scientific Reports</th>
<th>Patents</th>
<th>H-number/Avg. Top 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>276</td>
<td>152</td>
<td>113</td>
<td>173</td>
<td>45</td>
<td>3</td>
<td>17</td>
<td>1</td>
<td>18,00 / 26,20</td>
</tr>
<tr>
<td>2011</td>
<td>240</td>
<td>156</td>
<td>108</td>
<td>160</td>
<td>35</td>
<td>19</td>
<td>14</td>
<td>0</td>
<td>16,00 / 23,40</td>
</tr>
<tr>
<td>2010</td>
<td>267</td>
<td>141</td>
<td>100</td>
<td>179</td>
<td>39</td>
<td>10</td>
<td>15</td>
<td>0</td>
<td>15,04 / 22,60</td>
</tr>
</tbody>
</table>
Workshop Roadmap

• 1st part (14:15 – 15:15)
  • Little bit of theory
  • Overview of state of the art

• 2nd part (15:30 – 16:30)
  • Encountered formats, tools
  • Unpacking end-to-end

• 3rd part (17:00 – 18:00)
  • Emulation introduction
  • Awesome exercises – find your own 0day!
What is a Firmware? (Ascher Opler)

• Ascher Opler coined the term "firmware" in a 1967 Datamation article
• Currently, in short: it’s the set of software that makes an embedded system functional
What is firmware? (IEEE)

IEEE Standard Glossary of Software Engineering Terminology, Std 610.12-1990, defines firmware as follows:

- The combination of a hardware device and computer instructions and data that reside as read-only software on that device.

Notes: (1) This term is sometimes used to refer only to the hardware device or only to the computer instructions or data, but these meanings are deprecated.

Notes: (2) The confusion surrounding this term has led some to suggest that it be avoided altogether."
Common Embedded Device Classes

- Networking – Routers, Switches, NAS, VoIP phones
- Surveillance – Alarms, Cameras, CCTV, DVRs, NVRs
- Industry Automation – PLCs, Power Plants, Industrial Process Monitoring and Automation
- Home Automation – Sensoring, Smart Homes, Z-Waves, Philips Hue
- Whiteware – Washing Machine, Fridge, Dryer
- Entertainment gear – TV, DVRs, Receiver, Stereo, Game Console, MP3 Player, Camera, Mobile Phone, Toys
- Other Devices - Hard Drives, Printers
- Cars
- Medical Devices
Common Processor Architectures

- ARM (ARM7, ARM9, Cortex)
- Intel ATOM
- MIPS
- 8051
- Atmel AVR
- Motorola 6800/68000 (68k)
- Ambarella
- Axis CRIS
Common Buses

• Serial buses - SPI, I2C, 1-Wire, UART
• PCI, PCIExpress
• AMBA
Common Communication Lines

- Ethernet - RJ45
- RS485
- CAN/FlexRay
- Bluetooth
- WIFI
- Infrared
- Zigbee
- Other radios (ISM-Band, etc/)
- GPRS/UMTS
- USB
Common Directly Addressable Memory

- DRAM
- SRAM
- ROM
- Memory-Mapped NOR Flash
Common Storage

- NAND Flash
- SD Card
- Hard Drive
Common Operating Systems

- Linux
  - Perhaps most favourite and most encountered
- VxWorks
- Cisco IOS
- Windows CE/NT
- L4
- eCos
- DOS
- Symbian
- JunOS
- Ambarella
- etc.
Common Bootloaders

- U-Boot
  - Perhaps most favourite and most encountered
- RedBoot
- BareBox
- Ubicom bootloader
Common Libraries and Dev Envs

- busybox + uClibc
  - Perhaps most favourite and most encountered
- buildroot
- openembedded
- crosstool
- crossdev
What Challenges Do Firmwares Bring?

• Non-standard formats
• Encrypted chunks
• Non-standard update channels
  • Firmwares come and go, vendors quickly withdraw them from support/ftp sites
• Non-standard update procedures
  • Printer’s updates via vendor-specific PJL hacks
  • Gazillion of other hacks
Updating to a New Firmware

• Firmware Update built-in functionality
  • Web-based upload
  • Socket-based upload
  • USB-based upload

• Firmware Update function in the bootloader

• USB-boot recovery

• Rescue partition, e.g.:
  • New firmware is written to a safe space and integrity-checked before it is activated
  • Old firmware is not overwritten before new one is active

• JTAG/ISP/Parallel programming
Updating to a New Firmware – Pitfalls

- TOCTOU attacks
- Non-mutual-authenticating update protocols
- Non-signed packages
- Non-verified signatures
- Incorrectly/inconsistently verified signatures
- Leaking signature keys
Why Are Most Firmwares Outdated?

Vendor-view

- Profit and fast time-to-market first
  - Support and security comes (if at all!) as an after-thought
- Great platform variety raises compilation and maintenance effort
- Verification process is cumbersome, takes a lot of time and effort
  - E.g. for medical devices depends on national standards which require strict verification procedure, sometimes even by the state.
Why Are Most Firmwares Outdated?

Customer-view

- "If it works, don’t touch it!"
- High effort for customers to install firmwares
- High probability something goes wrong during firmware upgrades
- "Where do I put this upgrade CD into a printer – it has no keyboard nor a monitor nor an optical drive?!"
Firmware Formats
Firmware Formats – Typical Objects Inside

- Bootloader (1st/2nd stage)
- Kernel
- File-system images
- User-land binaries
- Resources and support files
- Web-server/web-interface
Firmware Formats – Components Category View

- Full-blown (full-OS/kernel + bootloader + libs + apps)
- Integrated (apps + OS-as-a-lib)
- Partial updates (apps or libs or resources or support)
Firmware Formats – Packing

Category View

- Pure archives (CPIO/Ar/Tar/GZip/BZip/LZxxx/RPM)
- Pure filesystems (YAFFS, JFFS2, extNfs)
- Pure binary formats (SREC, iHEX, ELF)
- Hybrids (any breed of above)
Firmware Formats – Flavors

- Ar
- YAFFS
- JFFS2
- SquashFS
- CramFS
- ROMFS
- UbiFS
- xFAT
- NTFS
- extNfs
- iHEX
- SREC/S19
- PJL
- CPIO/Ar/Tar/GZip/BZip/LZxxx/RPM
Firmware Analysis
Firmware Analysis – Overview

- Get the firmware
- Reconnaissance
- Unpacking
- Reuse engineering (check code.google.com and sourceforge.net)
- Localize point of interest
- Decompile/compile/tweak/fuzz/pentest/fun!
Firmware Analysis – Getting the Firmware

Many times not as easy as it sounds! In order of increasing complexity of getting the firmware image:

- Present on the product CD/DVD
- Download from manufacturer FTP/HTTP site
- Many times need to register for manufacturer spam :( 
- Google Dorks
- FTP index sites (mmnt.net, ftpfiles.net)
- Wireshark traces (manufacturer firmware download tool or device communication itself)
- Device memory dump
Firmware Analysis – Reconnaissance

- strings on the firmware image/blob
  - Fuzzy string matching on a wide embedded product DB
- Find and read the specs and datasheets of device
• Did anyone pay attention to the previous section?!
Unpacking firmware from SREC/iHEX files

SREC and iHEX are much simpler binary file formats than elf - in a nutshell, they just store memory addresses and data (although it is possible to specify more information, it is optional and in most cases missing). Those files can be transformed to elf with the command

```
objcopy -I ihex -O elf32-little <input> <output>
objcopy -I srec -O elf32-little <input> <output>
```

Of course information like processor architecture, entry point and symbols are still missing, as they are not part of the original files. You will later see some tricks how to guess that information.
Firmware Emulation
Firmware Emulation – Prerequisites

- Kernel image with a superset of kernel modules
- QEMU compiled with embedded device CPU support (e.g. ARM, MIPS)
- Firmware – most usually split into smaller parts/FS-images which do not break QEMU
Debugging Embedded Systems

- JTAG
- Software debugger (e.g. GNU stub or ARM Angel Debug monitor)
- OS debug capabilities (e.g. KDB/KGDB)
Developing for Embedded Systems

• GCC/Binutils toolchain
• Cross-compilers
• Proprietary compiler
• Building the image
Reversing a Seagate HDD’s firmware file format

Task:

• Assuming you already have a memory dump of a similar firmware available
• Reverse-engineer the firmware file format
• Get help from the assembler code from the firmware update routine contained in the firmware
Obtaining a memory dump

- Seagate’s hard drives have a serial test console
- Can be accessed with a TTL (1.8V) → to UART converter cable
- The console menu (reachable via ^Z) has an online help:

All Levels CR: Rev 0011.0000, Flash, Abort
All Levels ‘/’: Rev 0001.0000, Flash, Change Diagnostic Command Level, /[Level]
All Levels ‘+’: Rev 0012.0000, Flash, Peek Memory Byte, +[AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels ‘-’: Rev 0012.0000, Flash, Peek Memory Word, -[AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels ‘=’: Rev 0011.0002, Flash, Poke Memory Byte, =[AddrHi],[AddrLo],[Data],[Opts]
All Levels ‘@’: Rev 0001.0000, Overlay, Batch File Label, @[LabelNum]
All Levels ‘|’: Rev 0001.0000, Overlay, Batch File Terminator, |
Obtaining a memory dump

- The `Peek` commands provide exactly what is needed
- One small BUT – the HDD crashes when an invalid address is specified :(
- After probing the address ranges, a python script easily dumps the memory ranges
Obtaining the firmware

Firmware Update for STM3500320AS, STM3750330AS, STM31000340AS

Firmware update information for certain Maxtor-brand DiamondMax 22 Serial ATA drives. Check to see if your model is included.

New firmware version: MX16

Which firmware is right for me?

You can verify the proper firmware revision for your drive model and serial number using the Drive Detect software.

This update applies to the following models:

<table>
<thead>
<tr>
<th>Model Number</th>
<th>Capacity</th>
<th>Firmware Download (exe)</th>
<th>Firmware Downloads (.iso Image)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM31000340AS</td>
<td>1TB</td>
<td>[download] MX1A in .exe format</td>
<td>[download] MX1A-3D4D in .iso format</td>
</tr>
<tr>
<td>STM3750330AS</td>
<td>750GB</td>
<td>[download] MX1A-3D4D in .iso format</td>
<td></td>
</tr>
<tr>
<td>STM3500320AS</td>
<td>500GB</td>
<td>[download] MX1A-2D in .iso format</td>
<td></td>
</tr>
</tbody>
</table>

Procedure for .exe file
Unpacking the firmware

A quite stupid and boring mechanic task:

$ 7z x MooseDT-MX1A-3D4D-DMax22.iso -o image
$ cd image
$ ls
[BOOT] DriveDetect.exe FreeDOS README.txt
$ cd \[BOOT\]/
$ ls
Bootable_1.44M.img
$ file Bootable_1.44M.img
Bootable_1.44M.img: DOS floppy 1440k, x86 hard disk boot sector
Unpacking the firmware

$ mount -o loop Bootable_1.44M.img /mnt
$ mkdir disk
$ cp -r /mnt/* disk/
$ cd disk
$ ls
AUTOEXEC.BAT COMMAND.COM CONFIG.SYS HIMEM.EXE
KERNEL.SYS MX1A3D4D.ZIP RDISK.EXE TDSK.EXE
unzip.exe
$ mkdir archive
$ cd archive
$ unzip ../MX1A3D4D.ZIP
$ ls
6_8hmx1a.txs CHOICE.EXE FDAPM.COM fd1464.exe
flash.bat LIST.COM MX1A4d.lod README.TXT
seaenum.exe
Unpacking the firmware

$ file *
6_8hmx1a.txs: ASCII text, with CRLF line terminators
CHOICE.EXE: MS-DOS executable, MZ for MS-DOS
FDAPM.COM: FREE-DOS executable (COM), UPX compressed
fdl464.exe: MS-DOS executable, COFF for MS-DOS,
            DJGPP go32 DOS extender, UPX compressed
flash.bat: DOS batch file, ASCII text, with CRLF
          line terminators
LIST.COM: DOS executable (COM)
MX1A4d.lod: data
README.TXT: ASCII English text, with CRLF line
            terminators
seaenum.exe: MS-DOS executable, COFF for MS-DOS,
             DJGPP go32 DOS extender, UPX compressed
Unpacking the firmware

$ less flash.bat
set exe=fdl464.exe
set family=Moose
set model1=MAXTOR STM3750330AS
set model2=MAXTOR STM31000340AS
rem set model3=
rem set firmware=MX1A4d.lodd
set cfgfile=6_8hmx1a.txs
set options=-s -x -b -v -a 20
...
:SEAFLASH1
%exe% -m %family% %options% -h %cfgfile%
if errorlevel 2 goto WRONGMODEL1
if errorlevel 1 goto ERROR
goto DONE
Unpacking the firmware (Summary)

- We have unpacked the various wrappers, layers, archives and filesystems of the firmware
  - ISO → DOS IMG → ZIP → LOD
- The firmware is flashed on the HDD in a DOS environment (FreeDOS)
- The update is run by executing a DOS batch file (flash.bat)
- There are
  - a firmware flash tool (fdl464.exe)
  - a configuration for that tool (6_8hmx1a.txs, encrypted or obfuscated/encoded)
  - the actual firmware (MX1A4d.lod)
- The firmware file is not in a binary format known to file and magic tools

→ Let's have a look at the firmware file!
Inspecting the firmware file: `hexdump`

```bash
$ hexdump -C MX1A4d.lod
00000000 00 00 00 00 00 00 00 00 00 00 00 00 00 07 00 |................|
00000010 80 01 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
00000020 00 00 00 00 00 22 00 00 00 00 00 00 00 00 00 |....."..........|
00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 79 dc |..............y.|
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |................|
*
000001c0 0e 10 14 13 02 00 03 10 00 00 00 00 ff 10 41 00 |................A.|
000001d0 00 20 00 00 ad 03 2d 00 13 11 15 16 11 13 07 20 |. . .---.....|
000001e0 00 00 00 00 00 40 20 00 00 00 00 00 00 00 00 |@ ................
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3f 1d |..............?.|
00000200 00 c0 49 00 00 00 00 2d 00 10 b5 27 48 40 68 41 |...I...-'H@hAB|
00000210 26 48 00 f0 78 ee 10 bd 10 b5 04 1c ff f7 f4 ff |&H...x............|
00000220 a0 42 03 d2 22 49 40 18 00 1b 10 bd 00 1b 10 bd |.B."I@...........|
00000230 1d 48 40 68 40 42 70 47 10 b5 01 1c ff f7 f8 ff |
00000240 41 1a 0f 20 00 f0 5e ee 10 bd 7c b5 04 1c 20 1c |.H@h@BpG........|
00000250 00 21 00 90 17 a0 01 91 0c c8 00 98 00 f0 f2 ed |A. ..^...|
00000260 01 da 00 f0 ed ff ff f7 cf ff 05 1c 28 1c ff f7 |
00000270 d3 ff a0 42 fa d3 7c bd 7c b5 04 1c 20 01 00 1b |...E...|....|
00000280 00 21 00 90 0b a0 01 91 0c c8 00 98 00 f0 da ed |!..............|
...
```

→ The header did not look familiar to me :(
Inspecting the firmware file: strings

$ strings MX1A4d.lod
...
XlatePhySec, h[Sec],[NumSecs]
XlatePhySec, p[Sec],[NumSecs]
XlatePlpChs, d[Cyl],[Hd],[Sec],[NumSecs]
XlatePlpChw, f[Cyl],[Hd],[Wdg],[NumWdgs]
XlateSfi, D[PhyCyl],[Hd],[Sfi],[NumSfis]
XlateWedge, t[Wdg],[NumWdgs]
ChannelTemperatureAdj, U[TweakTemperature],[Partition],[Hd],[Zone],[Opts]
WrChs, W[Sec],[NumSecs],,[PhyOpt],[Opts]
EnableDisableWrFault, u[Op]
WrLba, W[Lba],[NumLbas],,[Opts]
WrLongOrSystemChs, w[LongSec],[LongSecsOrSysSec],[SysSecs],[LongPhySecOpt],,[SysOpts]
RwPowerAsicReg, V[RegAddr],[RegValue],[WrOpt]
WrPeripheralReg, s[OpType],[RegAddr],[RegValue],[RegMask],[RegPagAddr]
WrPeripheralReg, t[OpType],[RegAddr],[RegValue],[RegMask],[RegPagAddr]
...

→ Strings are visible, meaning the program is neither encrypted nor compressed
→ We actually know these strings ... they are from the diagnostic menu’s help!
$ binwalk MX1A4d.lod

<table>
<thead>
<tr>
<th>DECIMAL</th>
<th>HEX</th>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>
| 499792  | 0x7A050 | Zip archive data, compressed size: 48028,  
uncompressed size: 785886, name: ""

$ dd if=MX1A4d.lod of=/tmp/bla.bin bs=1 skip=499792
$ unzip -l /tmp/bla.bin
 Archive: /tmp/bla.bin
   End-of-central-directory signature not found. Either this file is not
   a zipfile, or it constitutes one disk of a multi-part archive. In the
   latter case the central directory and zipfile comment will be found on
   the last disk(s) of this archive.
unzip: cannot find zipfile directory in one of /tmp/bla.bin or
   /tmp/bla.bin.zip, and cannot find /tmp/bla.bin.ZIP, period.

→ binwalk does not know this firmware, the contained archive
was apparently a false positive.
Inspecting the firmware file: Visualization

To spot different sections in a binary file, a visual representation can be helpful.

- HexWorkshop is a commercial program for Windows. Most complete featureset (Hex editor, visualisation, ...)
  http://www.hexworkshop.com/
- Binvis is a project on google code for different binary visualisation methods. Visualisation is ok, but the program seems unfinished. http://code.google.com/p/binvis/
- Bin2bmp is a very simple python script that computes a bitmap from your binary
  http://sourceforge.net/projects/bin2bmp/
Inspecting the firmware file: Visualization with bin2bmp
Identifying the CPU instruction set

- **ARM**: Look out for bytes in the form of 0xeX that occur every 4th byte. The highest nibble of the instruction word in ARM is the condition field, whose value 0xe means AL, execute this instruction unconditionally. The instruction space is populated sparsely, so a disassembly will quickly end in an invalid instruction or lots of conditional instructions.

- **Thumb**: Look out for words with the pattern 0xF000F000 (bl/blx), 0xB500BD00 ("pop XXX, pc" followed by "push XXX, lr"), 0x4770 (bx lr). The Thumb instruction set is much denser than the ARM instruction set, so a disassembly will go for a long time before hitting an invalid instruction.
Identifying the CPU instruction set

- i386
- x86_64
- MIPS

In general, you should either know the processor already from the reconnaissance phase, or you try to disassemble parts of the file with a disassembler for the processor you suspect the code was compiled for. In the visual representation, executable code should be mostly colorful (dense instruction sets) or display patterns (sparse instruction sets).
Identifying the CPU instruction set

In our firmware, searching for "e?" in the hexdump leads us to:

00002420 04 e0 4e e2 00 40 2d e9
00002430 db f0 21 e3 8f 5f 2d e9
00002440 30 ff 2f e1 8f 5f bd e8
00002450 0e f0 69 e1 00 80 fd e8
00002460 94 00 00 00 30 a0 e1
00002470 10 40 2d e9 14 10 93 e5
00002480 08 e0 93 e5 02 20 8c e0
00002490 81 22 61 e0 01 25 62 e0
000024a0 d8 cd 9f e5 82 11 81 e0
000024b0 81 10 8c e0 f0 10 d1 e1
000024c0 f0 20 d2 e1 ac 01 2c e1
000024d0 ac cd 9f e5 fc c9 dc e1
000024e0 8e 1a 04 aa 10 80 bd e8
000024f0 80 40 a0 e1 07 00 54 e3

Let's verify that this is indeed ARM code ...
Finding the CPU instruction set

```bash
$ dd if=MX1A4d.lod bs=1 skip=$(( 0x2420 )) > /tmp/bla.bin
$ arm-none-eabi-objdump -b binary -m arm -D /tmp/bla.bin

/tmp/bla.bin:   file format binary

Disassembly of section .data:

00000000 <.data>:

  0: e24ee004  sub    lr, lr, #4
  4: e92d4000  stmfd  sp!, lr
  8: e14fe000  mrs    lr, SPSR
 c:  e92d5000  push   ip, lr
10: e321f0db  msr    CPSR_c, #219    ; 0xdb
14: e92d5f8f  push   r0, r1, r2, r3, r7, r8, r9, sl, fp, ip, lr
18: e59f1018  ldr    r1, [pc, #24]    ; 0x38
1c: e5910000  ldr    r0, [r1]
20: e12fff30  blx    r0
24: e8bd5f8f  pop    r0, r1, r2, r3, r7, r8, r9, sl, fp, ip, lr
28: e321f0d1  msr    CPSR_c, #209    ; 0xd1
2c: e8bd5000  pop    ip, lr
30: e169f00e  msr    SPSR_fc, lr
34: e8fd8000  ldm    sp!, pc^  
38: 00000044  andeq  r0, r0, r4, asr #32
3c: 01fe2008  mvnseq r2, r8
40: 00000094  muleq  r0, r4, r0
44: e1a03000  movq   r3, r0
48: e59fce0c  ldr    ip, [pc, #3596]    ; 0xe5c

→ Looks good!

Andrei Costin/Jonas Zaddach  www.firmware.re  58/78
Navigating the firmware

At the very beginning of a firmware, the stack needs to be set up for each CPU mode. This typically happens in a sequence of "msr CPSR_c, XXX" instructions, which switch the CPU mode, and assignments to the stack pointer. The msr instruction exists only in ARM mode (not true for Thumb2 any more ... :( ) Very close you should also find some coprocessor initializations (mrc/mcr).

```
18a2c: e3a000d7  mov     r0, #215      ; 0xd7
18a30: e121f000  msr     CPSR_c, r0
18a34: e59fd0cc  ldr     sp, [pc, #204] ; 0x18b08
18a38: e3a000d3  mov     r0, #211      ; 0xd3
18a3c: e121f000  msr     CPSR_c, r0
18a40: e59fd0c4  ldr     sp, [pc, #196] ; 0x18b0c
18a44: ee071f9a  mcr     15, 0, r1, cr7, cr10, 4
18a48: e3a00806  mov     r0, #393216 ; 0x60000
18a4c: ee3f1f11  mrc     15, 1, r1, cr15, cr1, 0
18a50: e1801001  orr     r1, r0, r1
18a54: ee2f1f11  mcr     15, 1, r1, cr15, cr1, 0
```
Navigating the firmware

In the ARMv5 architecture, exceptions are handled by ARM instructions in a table at address 0. Normally these have the form "ldr pc, XXX" and load the program counter with a value stored relative to the current program counter (i.e. in a table from address 0x20 on).

→ The exception vectors give an idea of which addresses are used by the firmware.

```
arm-none-eabi-objdump -b binary -m arm -D MX1A4d.lod \ 
| grep -E 'ldr\s+pc' | less
```
Navigating the firmware

We get the following output from `arm-none-eabi-objdump`

```
220e4:  e59ff018  ldr   pc, [pc, #24] ; 0x22104
220e8:  e59ff018  ldr   pc, [pc, #24] ; 0x22108
220ec:  e59ff018  ldr   pc, [pc, #24] ; 0x2210c
22100:  e59ff018  ldr   pc, [pc, #24] ; 0x22120
22104:  0000a824  andeq sl, r0, r4, lsr #16
22108:  0000a8a4  andeq sl, r0, r4, lsr #17
2210c:  0000a828  andeq sl, r0, r8, lsr #16
22110:  0000a7ec  andeq sl, r0, ip, ror #15
22114:  0000a44c  andeq sl, r0, ip, asr #8
22118:  00000000  andeq r0, r0, r0
2211c:  0000a6ac  andeq sl, r0, ip, lsr #13
22120:  00000058  andeq r0, r0, r8, asr r0
```
Emulating a Linux-based firmware

The goal is to run a firmware with as much functionality as possible in a system emulator (Qemu)
Emulating a Linux-based firmware

- We need a new Linux kernel. Why?
- Because the existing one is not compiled for the peripherals emulated by Qemu.
Compiling a Linux kernel for Qemu

Following this tutorial to build the kernel:
http://xecdesign.com/compiling-a-kernel/

```
sudo apt-get install git libncurses5-dev gcc-arm-linux-gnueabihf ia32-libs
git clone https://github.com/raspberrypi/linux.git
wget http://xecdesign.com/downloads/linux-qemu/linux-arm.patch
patch -p1 -d linux/ < linux-arm.patch
cd linux
make ARCH=arm versatile_defconfig
make ARCH=arm menuconfig
```
Change the following kernel options:

General Setup ---> Cross-compiler tool prefix = (arm-linux-gnueabihf-)
System Type ---> [*] Support ARM V6 processor
System Type ---> [*] ARM errata: Invalidation of the Instruction Cache operation can fail
Floating point emulation ---> [*] VFP-format floating point maths
Kernel Features ---> [*] Use ARM EABI to compile the kernel
Kernel Features ---> [*] Allow old ABI binaries to run with this kernel
Bus Support ---> [*] PCI Support
Device Drivers ---> SCSI Device Support ---> [*] SCSI Device Support
Device Drivers ---> SCSI Device Support ---> [*] SCSI Disk Support
Device Drivers ---> SCSI Device Support ---> [*] SCSI CDROM support
Device Drivers ---> SCSI Device Support ---> [*] SCSI low-lever drivers ---> [*] SYM53C8XX Version 2 SCSI support
Device Drivers ---> Generic Driver Options ---> [*] Maintain a devtmpfs filesystem to mount at /dev
Device Drivers ---> Generic Driver Options ---> [*] Automount devtmpfs at /dev, after the kernel mounted the root
File systems ---> Pseudo filesystems ---> [*] Virtual memory file system support (former shm fs)
Device Drivers ---> Input device support ---> [*] Event interface
General Setup ---> [*] Kernel .config support
General Setup ---> [*] Enable access to .config through /proc/config.gz
Device Drivers ---> Graphics Support ---> Console display driver support ---> [ ] Select compiled-in fonts
File systems ---> Select all file systems
Compiling a Linux kernel for Qemu

make ARCH=arm -j8
cp arch/arm/boot/zImage ../

... or just download the kernel that we prepared for you here
Get or compile Qemu

wget http://wiki.qemu-project.org/download/qemu-1.5.1.tar.bz2

```
tar xf qemu-1.5.1.tar.bz2
cd qemu-1.5.1
./configure --target-list=arm-softmmu
make -j8
```

or install the package of your distribution, if it is recent
(qemu-kvm-extras in Ubuntu 12.04)
Exercise –
DIR655_FW200RUB13Beta06.bin

- DLink DIR-655
- Wireless N Gigabit Router
Exercise –
DIR655_FW200RUB13Beta06.bin

- Getting DIR655_FW200RUB13Beta06.bin
- Unpacking DIR655_FW200RUB13Beta06.bin
  - Classic way
  - Firmware.RE way
- Exploring DIR655_FW200RUB13Beta06.bin
Exercise – 51110.2.1800.96.bin

- Vicon IPCAM 960 series
- IP/Network based cameras for CCTV surveillance
Exercise – 51110.2.1800.96.bin

- Getting 51110.2.1800.96.bin
- Unpacking 51110.2.1800.96.bin
  - "$\text{VICON}\_\text{JFFS2}$ is the unpacked JFFS2 image inside 51110.2.1800.96.bin"
- Exploring 51110.2.1800.96.bin web-interface
  - "$\text{VICON}\_\text{JFFS2}/\text{etc}/\text{lighttpd}/\text{lighttpd}.\text{conf}$"
  - "$\text{VICON}\_\text{JFFS2}/\text{mnt}/\text{www}.\text{nf}$"
Web-interface of 51110.2.1800.96.bin

- first, quick-explore the web-interface
- lighttpd-based
  - `sudo apt-get install lighttpd php5-cgi`
  - `sudo lighty-enable-mod fastcgi`
  - `sudo lighty-enable-mod fastcgi-php`
  - `sudo service lighttpd force-reload`
- then, we want to emulate the web-interface on a PC
  - requires tweaking `$VICON_JFFS2/etc/lighttpd/lighttpd.conf`
  - requires some minor development and fixes
Tweaking $VICON_JFFS2/etc/lighttpd/lighttpd.conf

- correct document-root
- replace /mnt/www.nf with $VICON_JFFS2/mnt/www.nf
- set port to 1337
- set errorlog and accesslog
- create plain basic-auth password file
- set auth.backend.plain.userfile
- replace all .fcgi files with a generic action.bottle.fcgi.py
- enable .py as FastCGI in $VICON_JFFS2/etc/lighttpd/lighttpd.conf
Writing a stub action.bottle.fcgi.py

- sudo apt-get install python-pip python-setuptools
- sudo pip install bottle
Running and debugging web-interface of 51110.2.1800.96.bin

- iterative-fixing approach
- `sudo lighttpd -D -f $VICON_JFFS2/etc/lighttpd/lighttpd.conf`
- check lighttpd logs for startup errors
- check Firefox web-developer console for client/server errors
  - console shows we need to define `INFO_SWVER` inside `info.js`
  - start from above by restarting lighttpd
Embedded devices and firmware security is an awesome topic :)
Nevertheless, security is totally missing :(  
Reversing firmwares used to be hard  
Now it is much cheaper, easier, faster  
Virtually any component of a firmware is vulnerable  
This includes web-interface, crypto PKI/IPSEC, unpatched/outdated dependencies/kernels  
Backdooring is still there and is a real problem
Questions?

• Ask right here right now

• Visit, share and support (by uploading firmwares) our project:
  • FIRMWARE.RE

• Contact us at:
  • contact@firmware.re
  • jonas@firmware.re
  • andrei@firmware.re