Embedded Devices Security
Firmware Reverse Engineering

Jonas Zaddach   Andrei Costin

EURECOM
Sophia Antipolis

August 11, 2013

black hat USA 2013

About – Jonas Zaddach

- PhD. candidate on "Development of novel binary analysis techniques for security applications" at EURECOM
- Advised by Prof. Davide Balzarotti
- Co-founder of FIRMWARE.RE
- Previous works: hard-disks ”torturer”, Amazon EC2 security
- jonas@firmware.re
- jonas.zaddach@eurecom.fr

black hat USA 2013

About – Andrei Costin

- PhD. candidate on "Software security in embedded systems" at EURECOM
- Advised by Prof. Aurélien Francillon
- Co-founder of FIRMWARE.RE
- Author of MFCUK (Mifare Classic), BT5-RFID (RFID-focused livecd)
- Previous works: printers, ADS-B
- andrei@firmware.re
- andrei.costin@eurecom.fr

black hat USA 2013

Administratrivia – Legal

- 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 following the entire or part of this workshop you agree:
  - to use the tools and knowledge acquired only for legal purposes and for activities you have explicit authorization for
  - to waive off presenters any liability which might arise from applying the tools or knowledge acquired by the attendee
  - to contribute, at the best of attendee capability, back to the security research community in terms of knowledge, tools and experience

Administratrivia – Setup

Before we actually start - get the exercise instructions and start downloading the tools image to have it ready:

- Follow instructions at http://firmware.re/bh13us.php
- Or connect to WIFI:
  - SSID: firmware.re_bh13us
  - KEY: firmware.re_bh13us_380154
  - Follow instructions at ftp://anonymous@192.168.1.1/sda_part1/bh13us.html

Eurecom – About

Academic Partners:

TELECOM ParisTech

POLITECNICO DI TORINO 1859 1906

Aalto University
School of Science
and Technology

Technische Universität München

NTNU
Norwegian University of
Science and Technology

VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY

black hat USA 2013

Eurecom – About

Industrial Partners:

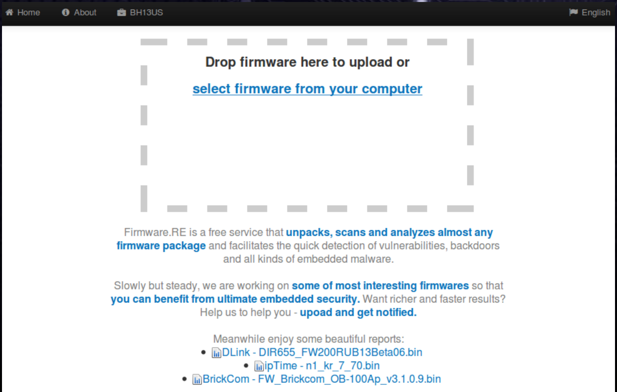
swisscom
SFR
orange
life.augmented
BMW Group
Forschung und Technik
Symantec.
Monaco telecom
SAP
IABG

Eurecom – Cote d'Azur

About – FIRMWARE.RE

Instead of relaxing at the beach, we are working to bring you

FIRMWARE.RE



Introduction

Introduction

Workshop Roadmap

- 1st part (14:15 – 15:15)
  - Little bit of theory
  - Overview of state of the art
  - Warm-up exercises
- 2nd part (15:30 – 16:30)
  - Encountered formats, tools
  - Unpacking challenges and ideas
  - Analysis and plugin-dev exercises
- 3rd part (17:00 – 18:00)
  - Emulation introduction
  - Awesome exercises – let's have some real fun!

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

In the news - Home Routers

- **2012-04-26 The Hacker News: More than 100,000 Wireless Routers have Default Backdoor** [3, 7]
  - Arcadyan Speedport routers allow connection with WPS "backdoor" PIN (even when WPS is disabled)
- **2013-03-15 Slashdot: Backdoor found in TP-Link routers** [15, 16]
  - HTTP request triggers download and execution of a TFTP file as root
  - CSRF attack leads to denial of service
- **2013-04-17 CNET: Top Wi-Fi routers easy to hack, says study** [9, 10]
  - Flaws allow access without authentication
  - Management sessions can be hijacked

In the news - Professional Routers

- **2010-03-02 Forbes: Cisco's Backdoor For Hackers** [17, 2]
  - Lawful interception interface prone to abuse
- **2013-03-15 Full-Disclosure: Critical SSH Backdoor in multiple Barracuda Networks Products** [11]
  - System contains backdoor user accounts that cannot be removed
  - IP access to SSH is whitelisted for Barracuda Networks and others

black hat USA 2013

In the news - Medical Devices

- **2008: Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses** [19]
  - Radio protocol discloses sensitive information
  - Operations of an ICD can be reprogrammed without authorization
- **2011-08-05 Dark Reading: Getting Root On The Human Body** [4, 21, 6]
  - Radio protocol of insuline pump is not secure
  - Injection programs can be changed

black hat USA 2013

Firmware in the news - SCADA

- **How digital detectives deciphered Stuxnet, the most menacing malware in history** - July 2010 [18, 5]
  - Stuxnet was a targeted attack against the Natanz uranium enrichment facility
  - An infected computer would send commands to exactly the there-used centrifuges that would drive them out of their specification range.
- **The lessons of Shamoon and Stuxnet ignored: US ICS still vulnerable in the same way** - 2013-01-04 [13]
- **Attacks on SCADA systems are increasing** - 2013-07-03 [8, 12]
  - Number of SQL injection attacks, spear phishing, etc against utility increased

black hat USA 2013

Firmware in the news - Cellphones

- **Kaspersky Researchers Discover Most Advanced Android Malware Yet** - June 2013 [23, 22]
  - The threat from this particular malware is low ... but
  - Sophistication of Android malware is about to reach the same level as computers
- **Millions of Sim cards are 'vulnerable to hack attack'** - July 2013 [14, 1]
  - An implementation flaw in many older SIM cards makes it possible to break the manufacturer key used to sign software updates - stay tuned for the talk here on BH!

Common Processor Architectures

LOW END

- MSP430
- 8051
- Atmel AVR

HIGH END

- ARM (ARM7, ARM9, Cortex)
- Intel ATOM
- MIPS
- Motorola 6800/68000 (68k)
- Ambarella
- Axis CRIS
- Tensilica/Xtensa

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 encoutered
- VxWorks
- Cisco IOS
- Windows CE/NT
- L4
- eCos
- DOS
- Symbian
- JunOS
- Ambarella
- etc.

Common Bootloaders

- U-Boot
  - Perhaps most favourite and most encoutered
- RedBoot
- BareBox
- Ubicom bootloader
- OpenFirmware

Common Libraries and Dev Envs

- busybox + uClibc
  - Perhaps most favourite and most encoutered
- buildroot
- openembedded
- crosstool
- crossdev

$\Rightarrow$ At the end of the build process, the vendor (or we) obtain a flashable firmware image including bootloader, operating system and applications.

First unpacking exercise

- IQeye Smart Camera Systems – IQeye 832 V3.4/5 Firmware
- Alinking IP Camera Systems – Alinking CMOS Mega Pixel Box IP Cam ALC 9153

What Challenges Do Firmwares Bring?

- Non-standard formats
- Encrypted chunks
- Non-standard or non-accessible update channels
  - Firmwares come and go, vendors quickly withdraw them from support/ftp sites
  - Industry standard update channels like ADSL with ACS
- Non-standard update procedures
  - Printer's updates via vendor-specific PJL hacks
  - Gazillion of other hacks
- Firmware update file not available at all
  - Firmware only distributed on device's flash
  - Needs to be dumped from the flash for analysis

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

black hat USA 2013

Updating to a New Firmware – Pitfalls

- TOCTTOU attacks [20]
- Non-mutual-authenticating update protocols
- Non-signed packages
- Non-verified signatures
- Incorrectly/inconsistently verified signatures
- Leaking signature keys

black hat USA 2013

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.

black hat USA 2013

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
  $\rightarrow$ Some devices do not provide recovery procedures in case something goes wrong ("Bricking")
- "Where do I put this upgrade CD into a printer – it has no keyboard nor a monitor nor an optical drive?!"

black hat USA 2013

1st Break

- Please be back in 10 minutes!

Firmware Formats

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 I

- Pure filesystems
  - YAFFS
  - JFFS2
  - SquashFS
  - CramFS
  - ROMFS
  - UbiFS
  - xFAT
  - NTFS
  - extNfs

Firmware Formats – Packing Category View II

- Pure archives
  - CPIO
  - Ar
  - Tar
  - GZip
  - Bzip2
  - LZxxx
  - RPM/DEB
- Pure binary formats
  - iHEX
  - SREC/S19
- Hybrids (any breed of above)

Firmware Analysis

Firmware Analysis

Firmware Analysis – Overview

- Reconnaissance first – when done on device
- Get the firmware then Reconnaissance – when only firmware is available
- Unpacking
- Reuse engineering (check code.google.com and sourceforge.net)
- Localize point of interest
  - password cracking – /etc/passwd
  - web pentesting – /var/www, /etc/lighttpd
- Decompile/compile/tweak/fuzz/pentest/fun!

black hat USA 2013

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
- Google!

Firmware Analysis – Unpacking

- 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 (Altough 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

Firmware Emulation – Prerequisites

- Self-built kernel image with a superset of kernel modules
  - The device's kernel is normally not usable, since all device addresses are hardcoded (unlike in the x86 architecture).
  - The kernel serves as abstraction layer for hardware present in the emulator, compile your own and userspace programs still work
- 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

black hat USA 2013

Debugging Embedded Systems

- JTAG, Proprietary debugging connection
- Software debugger (e.g. GDB stub or ARM Angel Debug monitor) connected p.ex. over UART
- OS debug capabilities (e.g. KDB/KGDB)

Developing for Embedded Systems

- GCC/Binutils toolchain
- Cross-compilers
- Proprietary compiler
- Building the image

Firmware Exercise

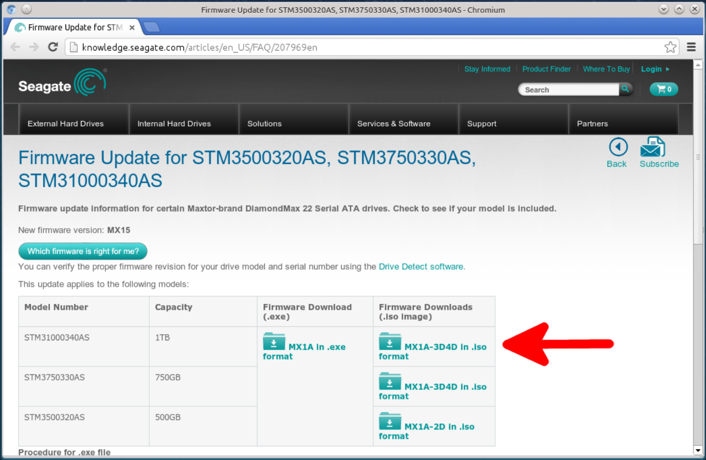
Firmware Exercise

Reversing a Seagate HDD's firmware file format

Task:

- Obtain the firmware image
- Extract the firmware file
- Reverse-engineer the firmware file format

Obtaining the firmware



Unpacking the firmware

A quite stupid and boring mechanic task:

```
$ 7z x MooseDT-MX1A-3D4D-DMax22.iso -oisoimage
$ cd isoimage
$ 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

```
$ mkdir -p /mnt2/imgimage
$ mount -o loop Bootable_1.44M.img /mnt2/imgimage
$ mkdir imgimage
$ cp -r /mnt2/imgimage/* imgimage/
$ 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   fdl464.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
```

black hat USA 2013

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
```

black hat USA 2013

Unpacking the firmware (Summary) I

- We have unpacked the various wrappers, layers, archives and filesystems of the firmware
  - ISO $\rightarrow$ DOS IMG $\rightarrow$ ZIP $\rightarrow$ 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)

Unpacking the firmware (Summary) II

- The firmware file is not in a binary format known to file and magic tools
- Sample heuristic to identify files of interest:
  - Unpack the firmware
  - Group by their *magic*
  - Flag for inspection the ones of interest or those with magic == data

→ Let's have a look at the firmware file!

Inspecting the firmware file: hexdump

```
$ hexdump -C MX1A4d.lod
00000000  00 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 00  |................|
00000020  00 00 00 00 00 22 00 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 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 40 20 00 00  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 2d 00  10 b5 27 48 40 68 41 42  |..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  |.H@h@BpG........|
00000240  41 1a 0f 20 00 f0 5e ee  10 bd 7c b5 04 1c 20 1c  |A.. ..^...|... .|
00000250  00 21 00 90 17 a0 01 91  0c c8 00 98 00 f0 f2 ed  |.!..............|
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  |...B..|.|... ...|
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

Inspecting the firmware file: binwalk

```
$ binwalk MX1A4d.lod

DECIMAL         HEX             DESCRIPTION
--------------------------------------------------------------------------
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.
```

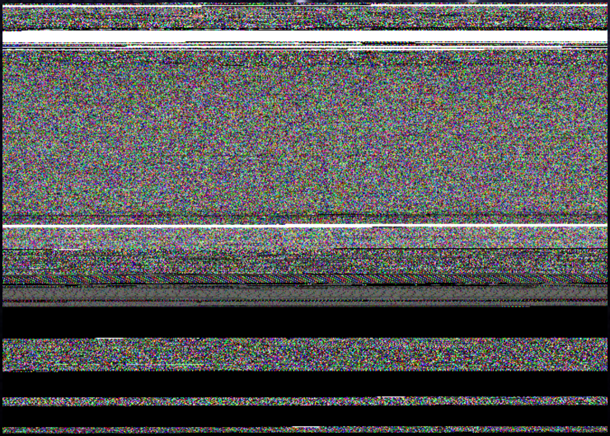
→ binwalk does not know this firmware, the contained archive was apparently a false positive.

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  00 e0 4f e1 00 50 2d e9  |..N..@-...O..P-.|
00002430  db f0 21 e3 8f 5f 2d e9  18 10 9f e5 00 00 91 e5  |..!.._-.........|
00002440  30 ff 2f e1 8f 5f bd e8  d1 f0 21 e3 00 50 bd e8  |0./.._....!..P..|
00002450  0e f0 69 e1 00 80 fd e8  44 00 00 00 08 20 fe 01  |..i.....D.... ..|
00002460  94 00 00 00 00 30 a0 e1  0c ce 9f e5 01 00 a0 e1  |.....0..........|
00002470  10 40 2d e9 14 10 93 e5  be c3 dc e1 d0 10 d1 e1  |.@-.............|
00002480  08 e0 93 e5 02 20 8c e0  92 01 01 e0 20 c0 e0 e3  |..... ...... ...|
00002490  81 22 61 e0 01 25 62 e0  42 29 a0 e1 82 0c 62 e1  |."a..%b.B)....b.|
000024a0  d8 cd 9f e5 82 11 81 e0  c6 20 51 e2 42 20 81 42  |......... Q.B .B|
000024b0  81 10 8c e0 f0 10 d1 e1  82 20 8c e0 04 c0 93 e5  |......... ......|
000024c0  f0 20 d2 e1 ac 01 2c e1  8e c2 2c e1 00 c0 83 e5  |. ....,...,.....|
000024d0  ac cd 9f e5 fc c9 dc e1  00 00 5c e3 10 40 bd a8  |..........\..@..|
000024e0  8e 1a 04 aa 10 80 bd e8  f0 41 2d e9 94 7d 9f e5  |.........A-..}..|
000024f0  80 40 a0 e1 07 00 54 e3  00 50 a0 e1 f7 6f 47 e2  |.@....T..P...oG.|
```

Let’s verify that this is indeed ARM code ...

Finding the CPU instruction set

```
$ 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        mov     r3, r0
      48:       e59fce0c        ldr     ip, [pc, #3596] ; 0xe5c
```

→ Looks good!

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).
$\rightarrow$ 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
220f0:      e59ff018     ldr     pc, [pc, #24]    ; 0x22110
220f4:      e59ff018     ldr     pc, [pc, #24]    ; 0x22114
220f8:      e1a00000     nop                      ; (mov r0, r0)
220fc:      e59ff018     ldr     pc, [pc, #24]    ; 0x2211c
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
```

Seagate firmware – take-aways

- Firmware unpacking takes a large amount of time and trial and error
- Unpacking can be automated to spend more of your time with actual code analysis, which is where you should spend your time
- How? See the next exercise ...

Seagate firmware – BAT plugins

- In this exercise, we are going to develop two plugins that will allow BAT to unpack the Seagate firmware into single files:
  - An ISO unpacking plugin that extracts files from a CD image
  - A Dos Floppy Image (FAT16 formatted) extractor plugin

Adding the ISO plugin to the BAT configuration file

- Add this to '/tools/gpltool/src/bruteforce-config' after the `"### unpack scans ###"` block:

```
# As we have seen, gpltool/bat by default doesn't unpack the
#'[BOOT]' section of an ISO. Let's write our ISO unpacker for this.

[iso9660_7z]
type        = unpack
module      = bat.firmware_re_bh13us
method      = searchUnpackISO7z
priority    = 3
magic       = iso9660
noscan      = text:xml:graphics:pdf:compressed:audio:video:java
description = Unpack ISO9660 (CD-ROM) file systems using 7z
enabled     = yes
```

Adding the IMG plugin to the BAT configuration file

```
# As we have seen, gpltool/bat by default doesn't
# recognize 'DOS floppy image' .img file. Let's write
# our IMG unpacker for this.

[dosfloppy]
type        = unpack
module      = bat.firmware_re_bh13us
method      = searchUnpackDosFloppyImg
priority    = 2
description = Unpack FAT16 DOS floppy .img files
enabled     = yes
```

Writing the ISO plugin to the BAT configuration file

- Edit '/tools/gpltool/src/bat/firmware_re_bh13us.py'
- Add implementation for the ISO unpacker
- Using the `7z -x <isofile> -o<output_dir>` command that we saw before

```
import subprocess
import os
import shutil
import magic

def searchUnpackISO7z(filename, tempdir=None, blacklist=[],
                      offsets=, envvars=None):
    tags = []
    counter = 1
    diroffsets = []

    # Reuse from BAT
    import fwunpack
```

Writing the ISO plugin to the BAT configuration file

```
tmpdir = fwunpack.dirsetup(tempdir, filename,
                           "iso-7z", counter)

cmd = ['7z', 'x', filename, '-o%s' % (tmpdir, ) ]
p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
                     stderr=subprocess.PIPE, close_fds=True)
(stanout, stanerr) = p.communicate()

if p.returncode != 0:
    shutil.rmtree(tmpdir)
else:
    tags.append('iso')
    diroffsets.append((tmpdir, 0, os.stat(filename).st_size))
    blacklist.append((0, os.stat(filename).st_size))

return (diroffsets, blacklist, tags)
```

Writing the IMG plugin to the BAT configuration file

- The same for the IMG unpacker plugin
- Using the mcopy -i <imgfile> -s -p -m -n ::/ -o<output_dir> command

```
def searchUnpackDosFloppyImg(filename, tempdir=None, blacklist=[],
                             offsets=, envvars=None):
    tags = []
    counter = 1
    diroffsets = []

    # Reuse from BAT
    import fwunpack
    tmpdir = fwunpack.dirsetup(tempdir, filename,
                               "dosfloppy", counter)
    cmd = ['mcopy', '-i', filename, '-s', '-p',
           '-m', '-n', '::/', tmpdir]
    p = subprocess.Popen(cmd, stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE,
                         close_fds=True)
```

Writing the IMG plugin to the BAT configuration file

```
(stanout, stanerr) = p.communicate()
if p.returncode != 0:
    shutil.rmtree(tmpdir)
else:
    tags.append('dos')

    ms = magic.open(magic.MAGIC_NONE)
    ms.load()
    mstype = ms.file(filename)
    ms.close()

    if mstype != None and 'boot' in mstype:
        tags.append('boot')

    diroffsets.append((tmpdir, 0, os.stat(filename).st_size))
    blacklist.append((0, os.stat(filename).st_size))

return (diroffsets, blacklist, tags)
```

Seagate firmware unpacking with BAT – take-aways

- We were able to automate all the unpacking by adding two plugins to BAT
- For future similar firmwares we do not need to do any work any more

2nd Break

- Please be back in 10 minutes!

Emulating a Linux-based firmware: Samsung Network HD Box Camera firmware exercise

The goal is to run the firmware of a Samsung Network HD Box Camera (SNB7000) 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 and will fail due to non-existent devices.

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
```

Compiling a Linux kernel for Qemu

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
```

black hat USA 2013

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.ta
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)

black hat USA 2013

Samsung Network HD Box Camera firmware exercise

Samsung SNB-7000 - Network HD Box Camera 1 / 2.8"

by Samsung

Be the first to review this item

List Price: ~~$672.00~~

Price: $621.01

You Save: $50.99 (8%)

Only 2 left in stock.

Ships from and sold by **Beach Audio**.

- SNB7000

15 new from $590.11

Samsung Network HD Box Camera firmware

- Get the firmware from Samsung
- Unpack the firmware with BAT:
  ```
  /tools/firmware.re_unpack.sh
  snb7000_Series_2.00_121004.zip /mnt/tmp
  ```

Samsung Network HD Box Camera firmware

- Inconveniently the kernel cannot mount the JFFS2 image directly, since it expects a mtd device
- An easy solution to circumvent this problem is to convert the JFFS2 image to an ext2 image

Samsung Network HD Box Camera firmware

```
dd if=/dev/zero bs=1M count=300 \
    of=/mnt/tmp/snb7000_ext2.img
sudo losetup /dev/loop1 /mnt/tmp/snb7000_ext2.img
sudo mkfs.ext2 /dev/loop1
```

(Note: If you do this on your own machine, double-check that you use the same loop device in both cases! If you use HDD encryption, then you might erase your drive by using the wrong command!)

```
sudo mkdir -p /mnt2/snb
sudo mount /dev/loop1 /mnt2/snb
cp -fr ./data/snb7000_Series_2.00_121004.zip-zip-1/ \
    snb7000_Series_2.00_121004.img-gzip-1/  \
    tmp1hLfhz-tar-1/work_snb7000.dm365-jffs2-1/* /mnt2/s
sudo umount /mnt2/snb
sudo losetup -d /dev/loop1
```

black hat USA 2013

Start Qemu with Samsung Network HD Box Camera firmware

```
qemu-system-arm  -M versatilepb -cpu arm1176 -m 256 \
    -serial tcp::1235,server,nowait \
    -kernel zImage_3.10.2 -hda ramdisk_snb7000.dm365 \
    -hdb snb.ext2 -net nic -net user \
    -redir tcp:8000::1022 \
    -redir tcp:8001::80 \
    -redir tcp:8002::443 \
    -redir tcp:8003::554
    -append "root=/dev/sda \
        console=ttyAMA0,115200 console=tty \
        init=/bin/sh \
        ip=10.0.2.15:::255.255.255.0:snb:eth0:off"
```

black hat USA 2013

Running the Samsung Network HD Box Camera firmware

- Qemu starts up the system, which greets you with `sh-4.1#`
- The shell inside is a little fragile, i.e. Ctrl+C does not work to interrupt a command
- So the first goal is to get the SSH server running
- The second goal will be to get the Web Server running and access some web applications

Summary and Take-aways I

- We took a firmware of an embedded device of interest
- For various reasons, we might not have or not want to have a device at hand
- We heavily used automation to unpack it
- We have briefly analyzed it for important components, files and keywords
- We have compiled a stock ARM kernel for embedded device emulation
- We have compiled QEMU for ARM devices emulation
- We have automated (scriptable steps) firmware loading into emulator
- We have successfully logged into the shell of the emulated device, over SSH

Summary and Take-aways II

- We have successfully logged into the web-interface of the emulated device, over HTTP and HTTPS
- We have successfully extracted PEM private key used for SSL – it was protected by an EMPTY passphrase
- We have successfully decrypted the SSL traffic, including ”secure over SSL” web-login of the admin
- We have successfully found the username and password by analyzing strings of the firmware and running strings-based HTTP basic-auth bruteforce script

Summary and Take-aways III

- 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?

Visit, share and support our project:

- FIRMWARE.RE
- Upload, upload, upload... We eat firmwares for breakfast, lunch and five o'clock tea!

Contact us at (for *trainings* or general queries):

- contact@firmware.re
- jonas@firmware.re
- andrei@firmware.re

K THX C U BY

K THX C U BY

References I

- Rooting sim cards.
  https://srlabs.de/rooting-sim-cards/.
- Cisco's backdoor for hackers, March 2010.
  http://www.forbes.com/2010/02/03/hackers-networking-equipment-technology-security-cisco.html.
- Brute forcing wi-fi protected setup, November 2011.
  http://sviehb.files.wordpress.com/2011/12/viehboeck_wps.pdf.
- Getting root on the human body, August 2011.
  http://www.darkreading.com/vulnerability/getting-root-on-the-human-body/231300312.
- How digital detectives deciphered stuxnet, the most menacing malware in history, July 2011.
  http://arstechnica.com/tech-policy/2011/07/how-digital-detectives-deciphered-stuxnet-the-most-menacing-malware

References II

- Medical device security under fire at black hat, defcon, August 2011.
  http://www.darkreading.com/evil-bytes/medical-device-security-under-fire-at-bl/231500306.
- The hacker news: More than 100,000 wireless routers have default backdoor, April 2012.
  http://thehackernews.com/2012/04/more-than-100000-wireless-routers-have.html.
- Attacks on scada systems are increasing, July 2013.
  http://www.h-online.com/security/news/item/Attacks-on-SCADA-systems-are-increasing-1910302.html.
- Cnet: Top wi-fi routers easy to hack, says study, April 2013.
  http://news.cnet.com/8301-1009_3-57579981-83/top-wi-fi-routers-easy-to-hack-says-study/.

black hat USA 2013

References III

- Exploiting soho routers, April 2013.
  http://securityevaluators.com//content/case-studies/routers/soho_router_hacks.jsp.
- [full-disclosure] sec consult sa-20130124-0 :: Critical ssh backdoor in multiple barracuda networks products, January 2013.
  http://archives.neohapsis.com/archives/fulldisclosure/2013-01/0221.html.
- Ics-cert monitor, June 2013.
  http://ics-cert.us-cert.gov/sites/default/files/ICS-CERT_Monitor_April-June2013_3.pdf.
- The lessons of shamoon and stuxnet ignored: Us ics still vulnerable in the same way, January 2013.
  http://www.infosecurity-magazine.com/view/30058/the-lessons-of-shamoon-and-stuxnet-ignored-us-ics-still-vulnerable-in-

References IV

- Millions of sim cards are 'vulnerable to hack attack', July 2013.
  http://www.bbc.co.uk/news/technology-23402988.
- Report: Backdoor found in tp-link routers, March 2013.
  http://habrahabr.ru/post/172799/.
- Slashdot: Backdoor found in tp-link routers, March 2013.
  http://tech.slashdot.org/story/13/03/15/1234217/backdoor-found-in-tp-link-routers.
- Tom Cross.
  Exploiting lawful intercept to wiretap the internet, March 2010.
  http://www.blackhat.com/presentations/bh-dc-10/Cross_Tom/BlackHat-DC-2010-Cross-Attacking-LawfulI-Intercept-wp.pdf.

black hat USA 2013

References V

- Nicolas Falliere, Liam O Murchu, and Eric Chien. W32.stuxnet dossier, February 2011. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf.
- D. Halperin, T.S. Heydt-Benjamin, B. Ransford, S.S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W.H. Maisel. Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 129–142, 2008. http://www.secure-medicine.org/public/publications/icd-study.pdf.

black hat USA 2013

References VI

- Collin Mulliner and Benjamin Michéle.
  Read it twice! a mass-storage-based tocttou attack.
  In *Proceedings of the 6th USENIX conference on Offensive Technologies*, pages 11–11. USENIX Association, 2012.
- Jerome Radcliffe.
  Hacking medical devices for fun and insulin: Breaking the human scada system, August 2011.
  http://cs.uno.edu/~dbilar/BH-US-2011/materials/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf.
- Roman Unucheck.
  The most sophisticated android trojan, June 2013.
  http://www.securelist.com/en/blog/8106/The_most_sophisticated_Android_Trojan.

References VII

Ryan Whitwam.
Kaspersky researchers discover most advanced android malware yet, June 2013.
http://www.androidpolice.com/2013/06/07/kaspersky-researchers-discover-most-advanced-android-malware-yet/.