

Fudging with Firmware

High Level Security Board
16-17 November 2006
Königstein, Germany

Pierre Pronchery
n.runs AG

The talker

Pierre Pronchery

I work for n.runs AG

UNIX stuff
Network security
System programming
Operating Systems internals

Agenda

- I. How does it look?
- II. First peek under the hood
- III. Identification
- IV. Is there more to it?
- V. Have some fun

Before we start

- Focusing on firmwares likely to host an Operating System
- Assumes you know how to obtain some:
 - Read your hardware documentation
 - Look for undocumented features
 - Check web sites **extensively**
 - Use your imagination...

I. How does it look?

- Unpacking
 - Presentation
 - Compression
 - Bootloaders
 - Extraction
- Storing
 - Filesystems

Presentation

- Data may just be encoded in a trivial way
- ASCII versus EBCDIC
 - Don't take anything for granted!
- ASCII-armored data transfers:
 - UUENCODE
 - Base64
 - Intel HEX...
- History (or hype) decides

XXENCODE Principles

- Groups of 3 bytes (trailing zeros)
- Split groups into four 6-bit numbers
- Apply the following translation table:

0	1	2	3	4	5	6
01234567890123456789012345678901234567890123456789012345678901234567890123						
+ - 0123456789ABCDEFGHIJKLMN0PQRSTUVWXYZabcdefghijklmnopqrstuvwxyz						

UUENCODE Principles

- Same technique, using ASCII 32 to 95:

0	1	2	3	4	5	6
0123456789012345678901234567890123456789012345678901234567890123						
!"#\$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_						

- More efficient than XXENCODE
- Seen in serial transfers (ARM platforms)

UUENCODE Example

- UUENCODE has filename and permission:

```
$ echo BOOTME > bootme.txt
```

```
$ uuencode -e bootme.txt bootme.txt
```

```
begin 640 bootme.txt
```

```
'0D]/5$U%"K\!
```

```
,
```

```
end
```

- Strings “begin” and “end”

Base64 specifications

- Uses the same 3-to-4 byte technique
- Characters used vary, often [A-Za-z0-9+/-]
- Some implementations have different names:
 - MIME
 - BinHex
 - Privacy-Enhanced Mail (PEM, as with SSL)
 - OpenPGP's Radix-64 (appends a CRC)

Base64 Example

- Output is usually as-is:

```
$ base64 -e bootme.txt
```

```
Qk9PVE1FCg==
```

- Easy to recognize: small set of characters
- Common way to obfuscate passwords
- More useful in protocol reversing (SOAP, ...)

BinHex Example

- BinHex was found on Mac OS
- Used file extensions “hex”, “hqx”, “hcx”, ...

(This file must be converted with BinHex 4.0)

```
:$f*TEQKPH#jdCA0d,R0TG!"6594%8dP8)3#3"!&m!*!%EMa6593K!!%!!!&mFNa  
KG3,r!*!$&[rr$3d,BQPZD'9i,R4PFh3!RQ+!!"AV#J#3!i!!N!@QKUjrU!#3'[q
```

- Keep an eye on restricted sets of characters!

Intel HEX Format

- From Wikipedia's definition:
 - « *Intel HEX is a file format for conveying binary information for applications like programming microcontrollers, EPROMs, and other kinds of chips. It is one of the oldest file formats available for this purpose.* »
 - Text file, line delimited (CR/LF/NUL)
 - Hexadecimal values in uppercase ASCII
- | | | | | | |
|---|-----|---------|-----|--------|-------|
| 1 | 2 3 | 4 5 6 7 | 9 a | cnt... | n-1 n |
| : | cnt | address | typ | data | sum |

Intel HEX Example

```
:10010000214601360121470136007EFE09D2190140  
:100110002146017EB7C20001FF5F16002148011988  
:00000001FF
```

(from <http://www.cs.net/lucid/intel.htm>)

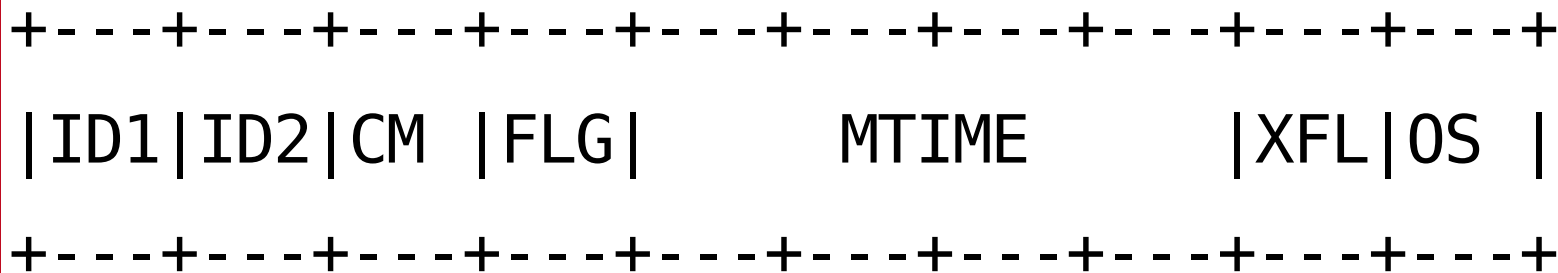
- Even more limited character set
- Redundancy in the encoding (checksum) is an interesting challenge for automation

Compression

- Consumes resources
- Known formats:
 - ZIP
 - GZIP
 - BZIP2...
- May be modified by vendors
 - Altered signatures
 - Different algorithms supported

GZIP

- Definition found in RFC1952
- Starts with \x1f\x8b
- Lots of false positives, check also compression method and level
- Includes CRC, timestamp, OS fields, and optionally a filename and a comment



Bootloaders

- On 80x86 boot sector is 512 bytes long, ends with partition table and 0xAA55
- Typically starts with a jump and stack initialization
- Will talk about assembly later...

Unpackers

- Thinking about:
 - Executable unpackers
 - Boot-time unpackers
- Look for known algorithms and signatures
- Play with checksums

Filesystems

- FAT
- Ext2 (Linux)
- Ramdisks (ROMFS, CRAMFS, ...)

File Allocation Table

- Poorly documented ...or brain-dead
- Lots of erratic implementations
- Always little-endian
- Starts jumping: 0xeb??90 or 0xe9????
- Often magic: "MSWIN4.1", "FAT", "FAT12", "FAT16", 0x55AA, 0x61417272
- Often redundant: boot sector backup, long filenames, ...

CPIO

- As seen on Linux initrd ramdisks
- Just a list of members
- File header either binary or ASCII octal
- Otherwise looks like a stat struct:
magic “070707”, dev, ino, mode, uid,
gid, nlinks, rdev, mtime, filename length
and filename

II. First peek under the hood

- Now you have a structure in mind
- Text forensics
- Binary forensics

Use the luck, forth

- Intuition matters
- “strings” is your friend
 - Common usernames
 - Common passwords
 - Operating System names, ...
- “hexdump” won't byte
- An automated tool would help a lot

Concrete example and demo

- CVE-2004-2556 Netgear WG602
super:5777364
- CVE-2004-2557 Netgear WG602 again
superman:21241036
- CVE-2006-1002 Netgear WGT624
Gearguy:Geardog

Vendors never learn...

III. Identification

- Executable formats
- Processor architectures
- Operating Systems

Executable formats

- Tell a lot about the target platform
- Two major formats:
 - Portable Executable (PE)
 - Executable and Linking Format (ELF)
- Tend to be re-used:
 - The wheel is a bit complex to reinvent
 - Flexible and complete enough
 - Many tools already support them

Portable Executable (PE)

- Inspired by UNIX's COFF
- Still compatible with MS-DOS 2.0
- “Windows is not portable” ...but CE is:
ARM, MIPS, Hitachi SH3, SH4, SH5...
- Used for:
 - Executables and object files
 - Libraries (DLL) and device drivers
 - Screensavers even

PE: Format overview

MS-DOS 2.0 EXE header	
Unused	
OEM Information	
Offset to PE header	
MS-DOS 2.0 Stub program	
Unused	

PE header	
Section headers	
Extra stuff	

PE: MS-DOS header

```

4d5a 9000 0300 0000 0400 0000 ffff 0000 MZ.....
b800 0000 0000 0000 4000 0000 0000 0000 .....@.....
0000 0000 0000 0000 0000 0000 0000 0000 .....
0000 0000 0000 0000 0000 0000 f000 0000 .....
0e1f ba0e 00b4 09cd 21b8 014c cd21 5468 .....!..L.!Th
6973 2070 726f 6772 616d 2063 616e 6e6f is program canno
7420 6265 2072 756e 2069 6e20 444f 5320 t be run in DOS
6d6f 6465 2e0d 0d0a 2400 0000 0000 0000 mode....$.

```

- Lots of magic to play with
- PE header is at 0xf0

PE: Actual header at 0xf0

```

5045 0000 4c01 0300 1084 7d3b 0000 0000 PE..L.....};....
0000 0000 e000 0f01 0b01 0700 0028 0100 ..... (..
009c 0000 0000 0000 7524 0100 0010 0000 ..... u$.....
0040 0100 0000 0001 0010 0000 0002 0000 .@.....
0500 0100 0500 0100 0400 0000 0000 0000 .....
00f0 0100 0004 0000 fcd7 0100 0200 0080 .....

```

- I386, PE32 executable, linker 7.0, entrypoint at 0x00012475, created with Windows XP, requires NT 4.0, ...

Executable & Linkable Format (ELF)

- Used on most UNIX systems
- Simpler than PE
- Very easy to spot: starts with “\x7fELF”

ELF: Format overview

ELF header	
Program header table	
Section 1	
Section 2	
...	
Section n	
Section header table	

ELF: Header structure

```

typedef struct {
    unsigned char    e_ident[16]; Elf32_Half    e_type;
    Elf32_Half      e_machine;    Elf32_Word    e_version;
    Elf32_Addr      e_entry;      Elf32_Off    e_phoff;
    Elf32_Off       e_shoff;      Elf32_Word    e_flags;
    Elf32_Half      e_ehsize;     Elf32_Half    e_phentsize;
    Elf32_Half      e_phnum;      Elf32_Half    e_shentsize;
    Elf32_Half      e_shnum;      Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
  
```

Processor architectures

- 80x86
- Sparc
- ARM
- MIPS
- m68k
- PowerPC

80x86

- Known hardware:
 - Your laptop
 - Soekris, Xbox, ...
- !!! Little-Endian !!!
- We know the standard boot processes
- There are other ways:
 - Load kernel from filesystem (Cobalt RaQs)
 - ...

80x86: Assembly overview

- Variable-size instructions
- Recurrent instructions:
 - push %ebp, mov %esp, %ebp => “\x55\x89\xe5”
 - leave, ret => “\xc9\xc3”

Acorn RISC Machine

- Known hardware:
 - embedded systems (music, games, phones)
 - DEC StrongARM, Intel XScale (PDAs)
- 32-bit architecture and opcodes
- Little or Big-endian at will
- Newer support 16-bit opcodes (Thumb)

MIPS

- Known hardware:
 - Workstations: SGI, DECstation, ...
 - Networking: Alcatel Speedtouch Pro, Linksys WRT, Cisco 36*0 and 7*00, WebTV, ...
 - Game consoles: N64, PSX, PS2, PSP
- 32-bit, 64-bit and hybrid versions
- Fixed-size opcodes
- Boots either little or big endian

PowerPC / CELL

- Known hardware:
 - Legacy Apple computers
 - NCD Explora X-Terminals
 - Game consoles
 - Nintendo's GameCube
 - Sony's PlayStation 3
 - Microsoft's Xbox 360

Operating Systems

- Well known: Linux, Windows CE
- Networking classics: Cisco's IOS, JuniperOS, ...
- Real-time: QNX, VxWorks

QNX

- UNIX for embedded systems
- Also found on regular PCs
- True microkernel
- Cisco IOS-XR (high availability) is based on QNX

VxWorks

- Another RTOS POSIX system
- Goes into space: Mars Orbiter
- ...and in your hardware:
 - some Linksys WRT54G
 - LiteON DVD recorders
 - digital cameras
 - Motorola SURFboard cable modems
 - some Xerox printers...

IV. Is there more to it?

- Deciphering

Deciphering

- Encrypted content is by definition random for the eyes (without the key)
- Again looking for any clue:
 - signatures
 - patterns
- Definitely not my specialty :(
- There is encryption and encryption

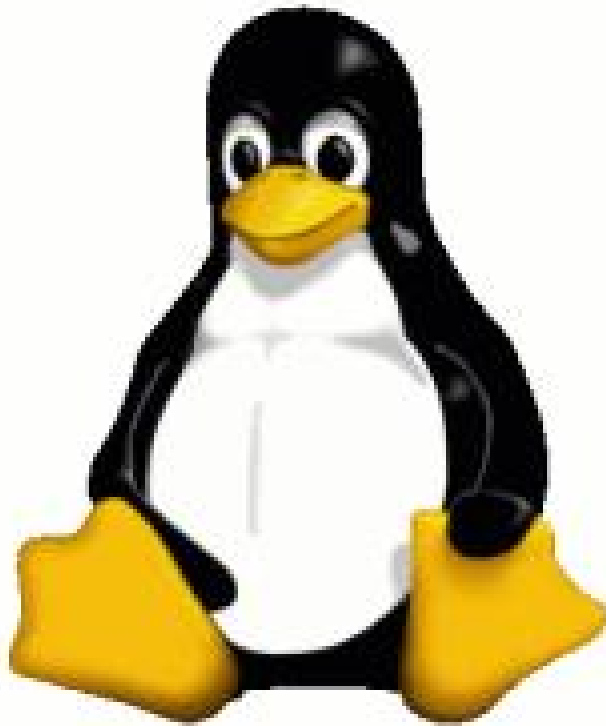
Some facts however

- Embedded systems are often slow
- Every hardware capability may not be available when booting
- If it is an algorithm, the logic is there: reverse it!
- If there is a key, it is there in the clear: find it!
(exception: if it is in the hardware, save a few million \$ for the microscope)

Cryptographic signatures

- Ask, ermm... a cryptograph
- Or Dan Kaminsky, he makes pretty pictures
- Seriously, drawing graphs helps

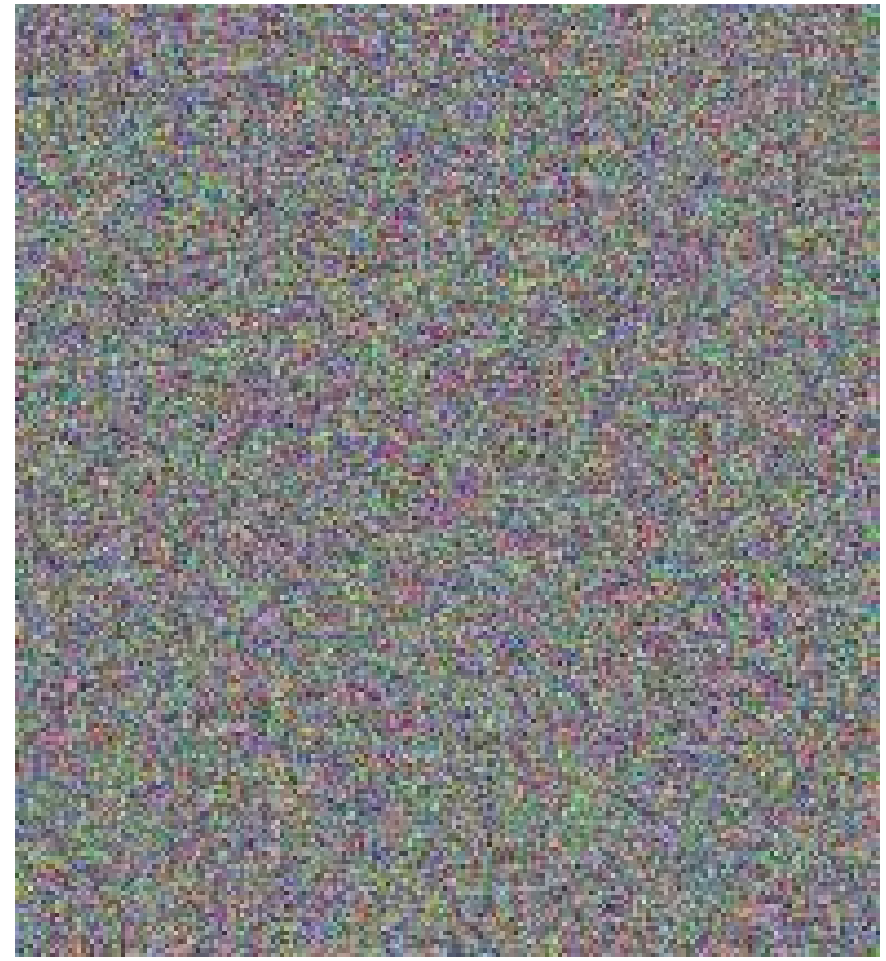
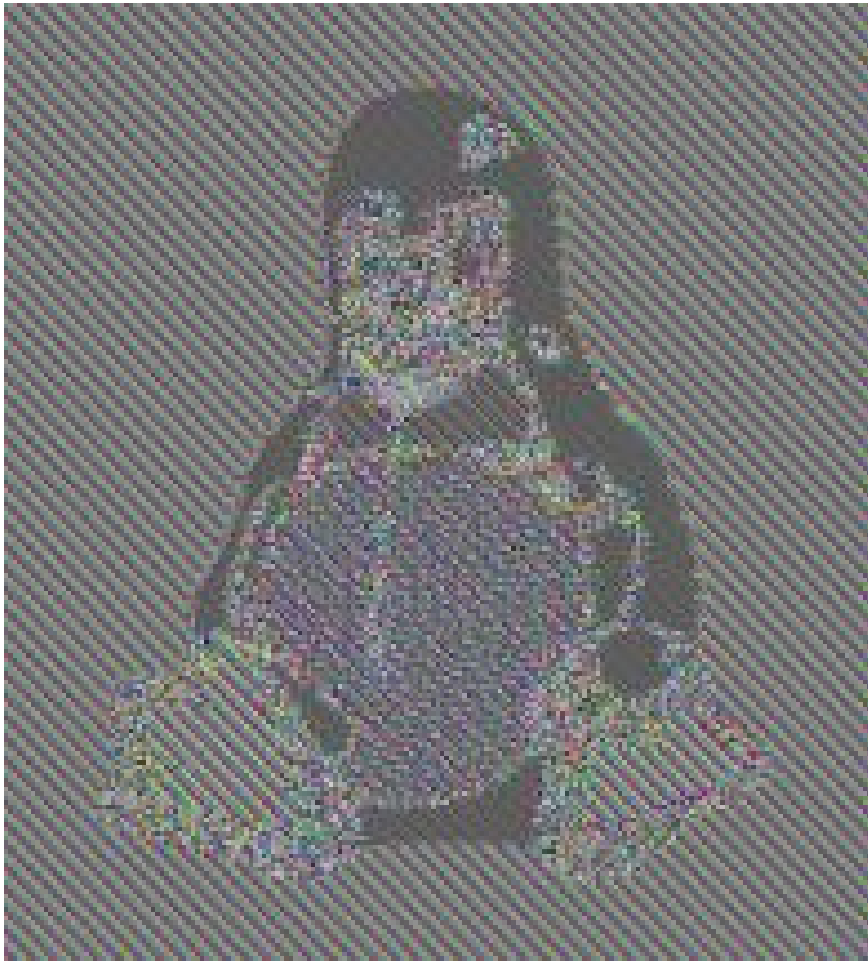
Cryptographic patterns



- Different encryption mechanisms
 - ECB Electronic Code Book
 - CBC and PCBC
 - CTR
 - ...

Cryptographic patterns

ECB versus CBC: patterns may still be found



V. Have some fun

- Disassembly
- Emulation
- Debugging

Disassembly software

- nasm
- GNU binutils
- OllyDbg
- IDA Pro

Disassembly: IDA Pro

Supported architectures:

- IA-32
- MIPS
- ARM... PDP-11

Supported operating systems:

- Windows
- Linux
- OS/2...

Emulation software

- VMWare, VirtualPC, Plex86, ...
- Bochs, QEMU, ...
- PearPC
- GXEmul, SimOS, MipsSim, vmips, ...
- Cisco 7200 Emulator

Emulator: Bochs

- Available on Windows, Linux, *BSD, ...
- Free software (GPL)
- Emulates IA-32 and AMD64
- Software simulation only
- Includes a debugging system
- Works with gdb

Emulator: QEMU

- Now for Linux, *BSD, MacOS X, Windows
- Free software (GPL)
- Emulates many platforms:
 - IA-32 and AMD64, up to 255 processors
 - Sparc (sun4m/32 bits and sun4u/64 bits)
 - ARM (ARM926E and ARM1026E)
 - MIPS...
- Runs binaries cross platform (Linux host)

References

- http://en.wikipedia.org/wiki/Cipher_block_chaining
- <http://en.wikipedia.org/wiki/QNX>
- Wikipedia is great
- <http://www.netbsd.org/Ports/emulators.html>
- Assembly reference books
- The “specifications” part of your users manuals
- And so on...

Contact information

- <http://www.nruns.com/>
We make the network run!



Pierre Pronchery

Solutions Consultant
Security

n.runs AG, Nassauer Straße 60, D-61440 Oberursel
phone +49 6171 699-0, fax +49 6171699-199
pierre.pronchery@nruns.com