Sticky finger & KBC Custom Shop

Alexandre Gazet
Sogeti / ESEC R&D
alex(at)security-labs.org

RECON 2011
Plan

1. **Introduction**
2. **Analysis of the implementation**
   - Dissecting a BIOS update
   - Looking for the keygen code
3. **Closer to metal**
   - Hardware identification
   - Firmware analysis
   - Understanding the I/O
   - Conclusion
4. **Taking advantage of the hardware**
   - KBC Custom Shop
   - Shout at the devil
   - Conclusion
About me

- Shit happens: lost my BIOS password
- No or outdated knowledge about BIOS
- Curious on how things work...

About BIOS pwd (long history):

- BIOS
- AWARD PW
- etc.
Introduction
Analysis of the implementation
Closer to metal
Taking advantage of the hardware

Master password for sell

- BIOS master pwd derived from serial number
- Many (commercial) offers
- Many models/manufacturers are concerned
There are plenty of fish on the sea

A providential exception[3]:

Dell 2A7B Keygen

A slight modification and the keygen generates now valid passwords for Dell 2A7B serials as well as for the -595D serials.

Source Code & Binaries

Quick How-To:

1. Download the archive of the keygen from the link above. It contains two files: a C file [source code] and an executable. If you are on Windows, just unpack and double-click the executable. If you are on Mac/Linux/BSD, compile the C file:

```bash
gcc -o dell dell.o
```
Initial objectives

- Analysis of the implementation of master pwd derivation
- For my laptop: Latitude E6400 (quite important later)

A two steps talk:
1. Analysis of the implementation
2. Exploiting new findings
Plan

1. Introduction
2. Analysis of the implementation
   - Dissecting a BIOS update
   - Looking for the keygen code
3. Closer to metal
   - Hardware identification
   - Firmware analysis
   - Understanding the I/O
   - Conclusion
4. Taking advantage of the hardware
   - KBC Custom Shop
   - Shout at the devil
   - Conclusion
Dissecting an update

Static analysis

- support.dell.com
- Among the supported options:
  - -writeromfile
  - -writehdirfile
  - -writekromfile
Entry point

Where to get information?
- bios modders community
  - Features unlocking
  - Overclocking
  - ACPI tables modifications to include license information (certificate)
- Forum mydigitallife.info:
  
  *Dell bios, how to decompose/mod*[1]
HDR file: extracting modules

- A header followed by a list of module objects
- module objects format: `type-length-value`:

```ruby
class ModHeader < BinData::Record
  endian : little
  uint8 : modId
  uint32 : len
  string : data, :read_length => :len
end

mod = ModHeader.new().read(stream)
```

- Format changes according to models
- Sometimes some padding is used
- Extracted modules are compressed
RLE-like Compression

- Decompression stub present in BIOS code
- *Run-Length Encoding* with feedback? *LZ*?*
- 3 commands:
  - duplicate \( n \) times the last byte from output stream
  - copy \( n \) bytes at position \( x \) from input stream
  - copy \( n \) bytes at position \( x \) from output stream
- Ruby script: HDR/modules header parsing and decompression
Analysis

- About fifty extracted modules (non-exhaustive)
- About 4Mo of binaries
- Identification based on strings/headers:
  - 12 BMP files
  - BRCMTPMDRV-MP-SEG32 v4.0.5 Broadcom
  - LENTPMDRV-MP-SEG32
  - **CompuTrace V80.887**[6]
  - DSAT DSDT ACPI
  - IBM VGA Compatible BIOS (Intel and Nvidia)
  - Intel Boot Agent PXE Base Code
  - Jetway j3205 CHIP TCM BIOS MP Version 1.05
  - etc.
Let me see your identification

grep ‘‘2A7B’’ ⇒ only one result

Objective: retrieve functions from keygen:
1. pad (depends upon models and pwd type)
2. calcsuffix (transformations)
3. encode (MD5 + dictionary)

Candidate module analysis:
- Responsible functions easily identified (string or MD5)
- Preboot Authentication Module?
- SMM routine (instruction rsm\(^1\)) !!

\(^1\)Resume from SMM
These are not the droids you’re looking for

Problems

- `calcsuffix` is not present?
- Instructions `in/out` on ports `0x910/0x911`?  

\(^a\)Non documented

Where it becomes interesting

- Email discussion with the author
- **A new player in the game: the keyboard controller?!**
SORRY MARIO
THE PRINCESS
IS IN ANOTHER
CASTLE
Plan

1. Introduction
2. Analysis of the implementation
   - Dissecting a BIOS update
   - Looking for the keygen code
3. Closer to metal
   - Hardware identification
   - Firmware analysis
   - Understanding the I/O
   - Conclusion
4. Taking advantage of the hardware
   - KBC Custom Shop
   - Shout at the devil
   - Conclusion
Chipset identification

- No specifications
- But schematics from a subcontractor (Quanta) for a similar model\(^a\)

\(^a\)available from a laptop repair website

Chipset MEC 5025/5035

- KBC or Super IO
- Built by SMSC
- No public specifications either...
- KBC often based on a 8051, not this one
Chipset identification

- No specifications
- But schematics from a subcontractor (Quanta) for a similar model\(^a\)

\(^a\)available from a laptop repair website

**Chipset MEC 5025/5035**

- KBC or Super IO
- Built by SMSC
- No public specifications either…
- KBC often based on a 8051, **not this one**
Processor identification

- Identifying the cpu from firmware’s binary code
- A recurrent pattern

\xC1\xC7\xC1\xC6\xC1\xC5

- Search engine:
  
  http://src.opensolaris.org/[...]/iw4965.ucode.hex

Something’s calling me back

- Intel Wifi card (ex: 4965AGN)
- Firmware distributed as binary blob
- New dead-end...
Finally, it is an ARC.

- Subject “There was a CPU...” from csdn.net$^2$
- Alias: alZou
- Hardware and software analysis on the same model

ARCompact
- 32-bit RISC architecture
- Used in embedded system
- Can interleave 16/32-bit instruction encoding

$^2$China Software Developer Network
Tools of our trade

Difficulties

- Not supported by IDA
- Configurable instruction set: undocumented encoding
- Unknown configuration registers
- Unknown I/O and memory layout

- Add ARCompact support to Metasm
- Basic switch and interruption table reconstruction
- Partial documentation from a third-party project: a Fujitsu STB

\[^{a}\]First challenge: find the instructions set’s documentation
**Tools of our trade**

**Difficulties**
- Not supported by IDA
- Configurable instruction set: undocumented encoding
- Unknown configuration registers
- Unknown I/O and memory layout

- Add ARCompact support to Metasm
- Basic *switch* and interruption table reconstruction
- Partial documentation from a third-party project: a Fujitsu STB

---

*First challenge: find the instructions set’s documentation*
Firmware analysis: *switch* example
Firmware analysis: memory read

```
COMMAND_READ_MEMORY:
  ldr  r0, [r17, -0x4]          : case 233 (0xe9)
  asl  r0, r0, 10h
  ldr  r1, [r17, -0x5h]
  asl  r1, r1, 6
  or   r0, r0, r1
  ldr  r24, [r17, -0x6h]
  or  r24, r24, r0
  brlo r24, 0x3000h, loc_0a3bah ; x:loc_0a3bah
  mov r14, -1
  b $loc_0a3bah               ; x:loc_0a3bah

// Xrefs: 0a3aeh
loc_0a3bah:
  ldr  r14, [r24, 0]

// Xrefs: 0a3b0h
loc_0a3b0h:
  mov r17, 0xffffffff
  stb r14, [r17, -0x8h]
  lsr  r0, r14, 8
  stb  r0, [r17, -0x5h]
  lsr  r0, r14, 10h
  stb  r0, [r17, -0x4h]
  lsr  r14, r14, 10h
  stb  r14, [r17, -0x3h]
  b $loc_0a795h            ; x:loc_0a795h
```

A. Gazet
Sticky finger & KBC Custom Shop
22/48
IO: CMOS analogy

Entry point:

- Analysis of interactions with the Real Time Clock (RTC)
- Access to the RTC Battery Backed Up RAM (aka CMOS)
- Pairs of registers $index/data$ ($0x70-0x76)$:

```asm
# Read
cli
mov al, index
out 0x70, al
in al, 0x71
sti

# Write
cli
mov al, index
out 0x70, al
mov al, value
out 0x71, al
sti
```

⇒ Apply the same concepts to the ports 0x910 et 0x911

---

3 Intel: Accessing the Real Time Clock Registers and the NMI Enable Bit
Protocols analysis

Communication with the KBC:

- Data:
  - Control or data bytes

- Functions:
  - Primitives: `readbyte`, `writebyte`
  - Compositions: `readbuffer`, `writebuffer`, `loopbyte`
  - Fatalities:

  `writebuffer(data) + writebyte(ctrl) + loopbyte(ctrl)`
  `readbuffer(data) ⇒ command execution (RPC-like) ??`
Calling card

- At hardware level
  - Mailbox Register Interface
  - Communication channel between the host and the ARC
  - Two ports in the IO space: 0x910 et 0x911
  - Most probably a sort of flash/memory-mapped control registers

- At software level (ARC)
  - The previously seen switch handles commands
  - Performs requested operations
  - Copy the response inside an exchange buffer
  - State the completion
When the master password is derived, the BIOS (more accurately the SMI handler):

- **pad** the serial number
- **Execute the command 78 (0x4E)**
  ⇒ **Implementation of the calcsuffix function**
- **encode** the result
The final countdown

The KBC exposes an interface

The SMI handler executes remote commands on it

Bidirectional data exchange KBC ⇔ SMM

Around 140 supported commands

SNAFU:

- master pwd derived from ServiceTag
- Efficiency of BIOS et HDD passwords “lowered”...
Overview

Main Processor
(\textit{Intel Core 2})

Operating System
(\textit{Windows 7 64 bits})

User space

Kernel space

SMM mode, SMI handler

Growing privileges

Interface
Ports 0x910/0x911

Firmware
(140 commands)

Keyboard controller
(\textit{ARCompact})
Roadmap

Current roadmap:

- Master pwd implementation analysis ✓
- KBC Communication protocol analysis ✓

Updated objectives: take advantage of the KBC
- Re-implement the protocol
- KBC takeover: code execution
Don’t talk to strangers

Talking to the KBC:

- Use of IO
- Administrative rights
- Ruby implementation of the protocol
  - Take advantage of a Windbg driver\(^4\)
  - Userland script that patches its own IOPL
  - Generate native code on the fly (Metasm/DynLdr)

What we can do:

- Dump the ROM (partially)
- Call commands from the interface

\(^4\)cf. Metasm talk RECON 2010
Interface analysis

KBC: interface and commands analysis:

- Understand few addresses (like IO addresses)
- Analysis of memory manipulations\(^5\)
- Read primitive accessible but limited (ROM access only)

\textbf{No (visible) write primitive...}

\(^5\)like memcpy
Analysis of another extracted module: bootb_dispatcher

- IO on the KBC ports (0x910/0x911) ⇐ woot ?
- “Flashing Keyboard Controller ROM.” ⇐ w00t !

Flashing procedure analysis:

- Send a 'Flash' command
- The ROM is split in blocks of 0x800 bytes
  - erase_block
  - flash_dw(addr,dw))
  - check_dw
- Update CRC and 'Finalize'
- (Hope for) reboot...
Fear of the dark

Methodology:
- Manual assembly of payloads
- Important available padding

What I’ve done:
1. Unlock read command
   $\Rightarrow$ ROM and RAM full access
2. Add a command logger (even while reboot)
3. Bonus: add a RAM write command

---

6. TODO add an ARC compiler to Metasm
Current roadmap

- Master pwd implementation analysis ✓
- KBC Communication protocol analysis ✓
- Code execution on the KBC ✓

Updated objectives:
- Backdoor (keylogger) inside the KBC ?[8]
- The KBC communicate with the SMI handler? ⇒ :)
SMM in one slide

- An SMI triggers the entry in SMM
- Execute in SMRAM
  - Normally inaccessible
  - Access control using bits D_OPEN and D_LOCK
- SAVE_STATE_MAP: save/restore the context when entering/leaving SMM
- Theoretically 16-bit code, in practice 32-bit
- Ring “-2”, under a possible hypervisor

Why one would target SMM?

Full control, stealthiness
Deeper analysis of the SMI handler

- From commands logs
- Auditing exchanges with the KBC

One specific function manipulates a buffer

- **Loaded/Saved** from/to the KBC
- Read/write a 0x10 bytes buffer
- Using 4 registers from the SAVE_STATE_MAP
Abusing the protocol

- Read in a fixed stack buffer
- Buffer size is hardcoded on both sides (0x10)
- By default exchange 8 bytes chunks
- Desynchronization by modifying the size of chunks sent by the KBC

**Judy is a punk**

1. *Integer underflow* on the size (type *short*)
2. *Stack overflow*, controlled by the KBC
3. No protection

⇒ *Possible code execution*
Code path analysis leads to a **new interface**

Check of the last instruction executed before entering SMM[^7]

⇒ out ax, 0xB2 [^4]

- al: command number
- ah: first argument

⇒ A new interface[^8]

---

[^7]: Use CR3 from the SAVE_STATE_MAP
[^8]: Dozens of commands
Updated overview

Main Processor
(Intel Core 2)

Operating System
(Windows 7 64 bits)

User space

Kernel space

Interface Ports 0x910/0x911

SMM mode, SMI handler

Keyboard controller
(ARCompact)

Firmware
(140 commands)

Growing privileges
Attack summary

Main Processor
(Intel Core 2)

Operating System
(Windows 7 64 bits)

User space

Kernel space

Interface
Out 0xB2

SMM mode, SMI handler

Interface Ports 0x910/0x911

Keyboard controller
(ARC\text{compact})

Firmware
(140 commands)

1. Flash the KBC
2. Write payload
3. Call trigger
4. Initiate read buffer
5. Send payload SMRAM +RWX
Attack summary

1. Flash the KBC
2. Write payload
3. Call trigger
4. Initiate read buffer
5. Send payload SMRAM +RWX
Attack summary

1. Flash the KBC
2. Write payload
3. Call trigger
4. Initiate read buffer
5. Send payload SMRAM +RWX
**Attack summary**

1. Flash the KBC
2. Write payload
3. Call trigger
4. Initiate read buffer
5. Send payload SMRAM +RWX

---

**Diagram Description:**

- **Main Processor** (Intel Core 2)
  - Operating System (Windows 7 64 bits)
    - User space
    - Kernel space
  - Interface Ports 0x910/0x911
  - SMM mode, SMI handler

- **Keyboard controller (ARCompact)**
  - Firmware (140 commands)
Payload: add functions $^9$ to the 0xB2 interface to expose the SMRAM

$^9$read, write, execute
Sometimes you hit the bar

- A classical problem of trust:
  - The SMM trust the KBC
  - But the KBC is accessible with different rights
  - ⇒ SMRAM access control bypassed

- Many independent computing units:
  - Main processor[4][7]
  - Network card[2][5]
  - Keyboard controller[8]
  - …

- Can we trust our hardware?
  ⇒ Taken into consideration in security models?
  ⇒ Integrity check?
  ⇒ Firmware auditing?
At the end, what’s the point?

- For most of us, fun but mostly useless in practice
- Require administrative rights/physical access
- For extremists: (very) targeted backdoor
  - The vulnerable function is called when the laptop reboots:
    - ⇒ Stealthiness, all-in-memory attack
    - ⇒ Resilience (even in case of OS reinstallation)

About this talk:

- No more than a simple PoC
- Concepts are quite generic
- But very strong hardware and software dependencies
High fives all around

Special thanks the following people for their precious help:

- Damien Aumaitre
- @IvanleF0u
- Yoann Guillot

Bruce Dang and Rolf Rolles for sharing inspiring thoughts.

---

10 twitter whore
Thanks for listening
Any questions?

Metasm: http://metasm.cr0.org/

Contact: #metasm at Freenode
References I

Apokrif.
Dell bios, how to decompose/mod, 2010.

Guillaume Delugré.
Closer to metal: Reverse engineering the broadcom netextreme’s firmware.

dogber1.
Dell 2a7b keygen, 2010.

Loic Duflot, Benjamin Morin, Olivier Levillain, and Olivier Grumelard.
Getting into the smram: Smm reloaded.
In CanSecWest, 2009.
Loïc Duflot, Guillaume Valadon, and Olivier Levillain.
*Quelques éléments en matière de sécurité des cartes réseau.*
In *SSTIC*, 2010.
http://www.sstic.org/media/SSTIC2010/SSTIC-actes/Peut_on_faire_confi...valadon_duflot_levillain_perez__.pdf.

Alfredo Ortega and Anibal Sacco.
*Deactivate the rootkit*, 2009.

Joanna Rutkowska.
*A quest to the core.*
In *BlackHat*, 2009.
http://www.invisiblethingslab.com/resources/misc09/Quest%20To%20The%20Core%20public%29.pdf.
References III