



# Understanding the TC model from a system architecture perspective

Loïc Duflot

French Network and Information Security Agency  
(ANSSI)

SGDN/ANSSI 51 boulevard de la Tour Maubourg 75007 Paris  
loic(dot)duflot(at)sgdn(dot)gouv(dot)fr

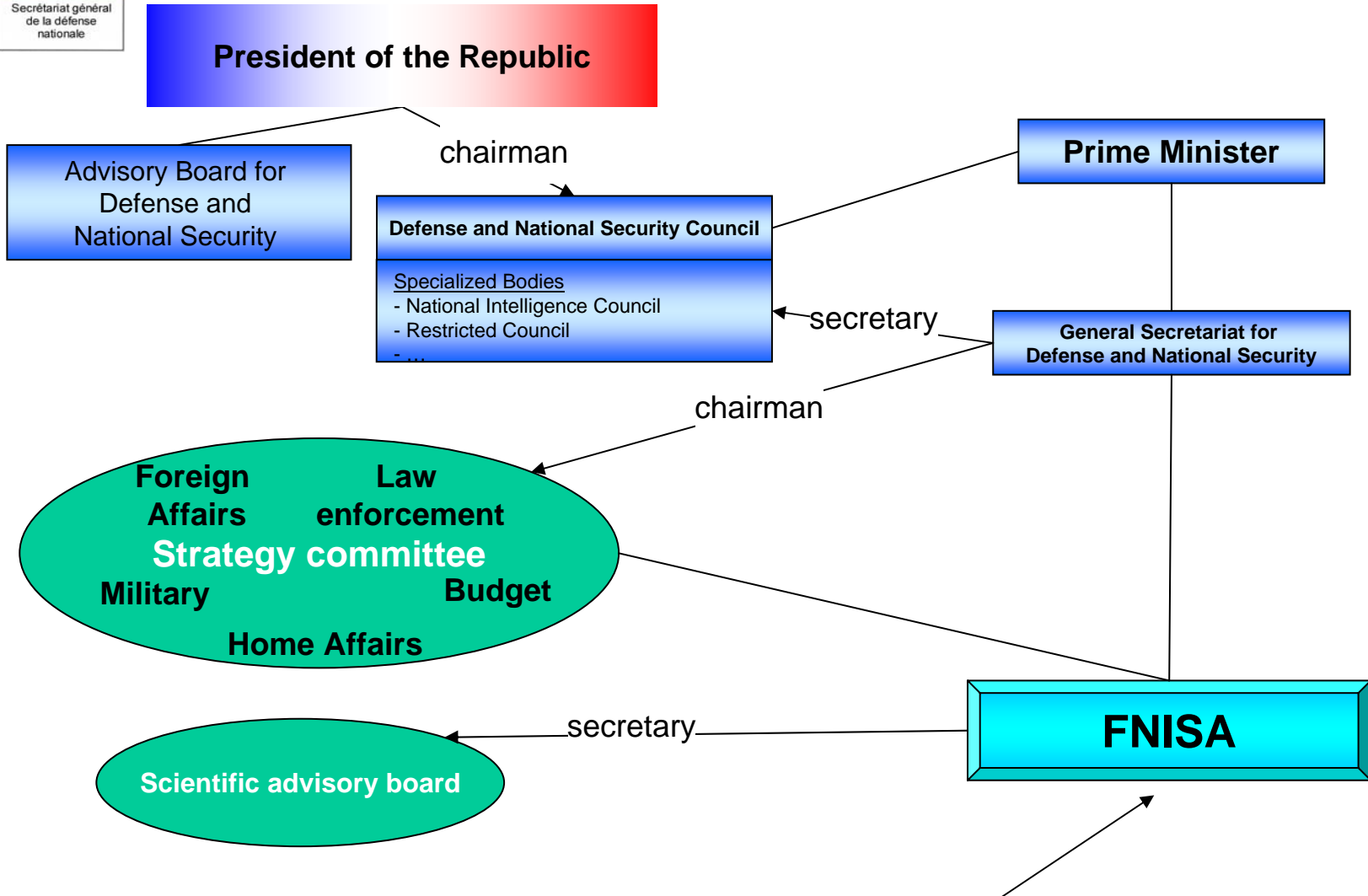


# Disclaimer

- This deck of slides has been designed to be used during the ETISS 2009 “understanding the TC model from a system architecture perspective” master class and for that purpose only.
- It will probably be meaningless without the explanation that goes with it.
- In particular, most of the examples presented should not be used in any real world design (can the reader guess why?).



# FNISA: who we are



## French Network and Information Security Agency

# Our missions

- **Alert warning response:**
  - central capacity for **early detection**:
    - of security events, of cyber-attacks.
    - sensors, correlations.
    - monitoring of the governmental gateways.
  - technical assessment.
  - cyber-crisis management.
- **High grade security products:**
  - To **develop high grade security products** for the protection of the most sensitive networks.
  - To **develop** and **operate** the most sensitive networks.
- **“Support” of customers:**
  - To **support** the departments and the Critical Infrastructure operators, in increasing their level of security.
  - To **check** the level of protection.
    - in charge of inspections (for the departments).
    - bringing technical support to inspection teams (for CI operators).
  - To spread good practice to other customers (private companies, SMEs, citizens).
- **Role of National Communication and Security Agency:**
  - Root certification authority.
  - In charge of the policy, certification body.
  - Authority for approvals.

# Introduction

- The goal of this Master Class is to show why system level considerations should be taken into account in any TC design:
  - Focusing on one aspect of TC only (TPMs for instance) is important but is not enough.
    - TC must be looked at from a system architecture perspective.
  - It is important to think “outside the box”.
    - Hypothesis and axioms must be verified.
- It is important to focus on the real problems.

# Some well known facts (probably not so true)



# What is this?

```
//unseals blob of size blob_size sealed for key of internal index key_index
int TPM_unseal_blob(char * blob, int blob_size, int key_index){
    int res;
    //allocate unsealed blob structure (TPM_SEALED_DATA structure)
    struct TPM_SEALED_DATA* unsealed_blob = malloc(sizeof(struct TPM_SEALED_DATA));
    //raw data is at most the size of an 256 AES key
    char sealed_key[AES_KEY_SIZE];
    if(blob_size)
    {
        //decrypt and identify fields of the TPM_SEALED_DATA structure
        res = decrypt(blob, blob_size, unsealed_blob, key_index);
        if (!res) return FAILURE;
        [...]
        strcpy(sealed_key, unsealed_blob->data);
        [...]
        return SUCCESS;
    }
    return FAILURE;
}
```

# What is this?

```
module mux4keys (y, a, auth);
  output [31:0] y;
  input  [1:0]  a;
  input                auth;
  reg    [31:0] y;

  always @(a or auth) begin
    y = 0;
    case ({auth,a}) // synopsys full case
      3'b100: y = key0;
      3'b101: y = key1;
      3'b110: y = key2;
      3'b111: y = key3;
    endcase
  end
endmodule
```

Credits: thanks to my colleague K. Khalfallah for digging that up



# What is this?

```
//unseals blob of size blob_size sealed for key of internal index key_index
int TPM_unseal_blob(char * blob, int blob_size, int key_index){
    int res;
    //allocate unsealed blob structure (TPM_SEALED_DATA structure)
    struct TPM_SEALED_DATA* unsealed_blob = malloc(sizeof(struct TPM_SEALED_DATA));
    //raw data is at most the size of an 256 AES key
    char sealed_key[AES_KEY_SIZE];
    if(blob_size)
    {
        //decrypt and identify fields of the TPM_SEALED_DATA structure
        res = decrypt(blob, blob_size, unsealed_blob, key_index);
        if (!res) return FAILURE;
        [...]
        strcpy(sealed_key, unsealed_blob->data);
        [...]
        return SUCCESS;
    }
    return FAILURE;
}
```

# Which of the three is the more secure?

```
//unseals blob of size blob_size sealed for key of internal index
int TPM_unseal_blob(char * blob, int blob_size, int key_index){
    int res;
    //allocate unsealed blob structure (TPM_SEALED_DATA struct
    struct TPM_SEALED_DATA* unsealed_blob = malloc(sizeof(struct TPM_SEALED_DATA));
    //raw data is at most the size of an 256 AES key
    char sealed_key[AES_KEY_SIZE];
    if(blob_size)
    {
        //decrypt and identify fields of the TPM_SEALED_DATA structure
        res = decrypt(blob, blob_size, unsealed_blob, key_index);
        if (!res) return FAILURE;
        [...]
        strcpy(sealed_key, unsealed_blob->data);
        [...]
        return SUCCESS;
    }
    return FAILURE;
}
```

```
module mux4keys (y, a, auth);
    output [31:0] y;
    input [1:0] a;
    input auth;
    reg [31:0] y;

    always @(a or auth) begin
        y = 0;
        case ({auth,a}) // synopsis full case
            3'b100: y = key0;
            3'b101: y = key1;
            3'b110: y = key2;
            3'b111: y = key3;
        endcase
    end
endmodule
```

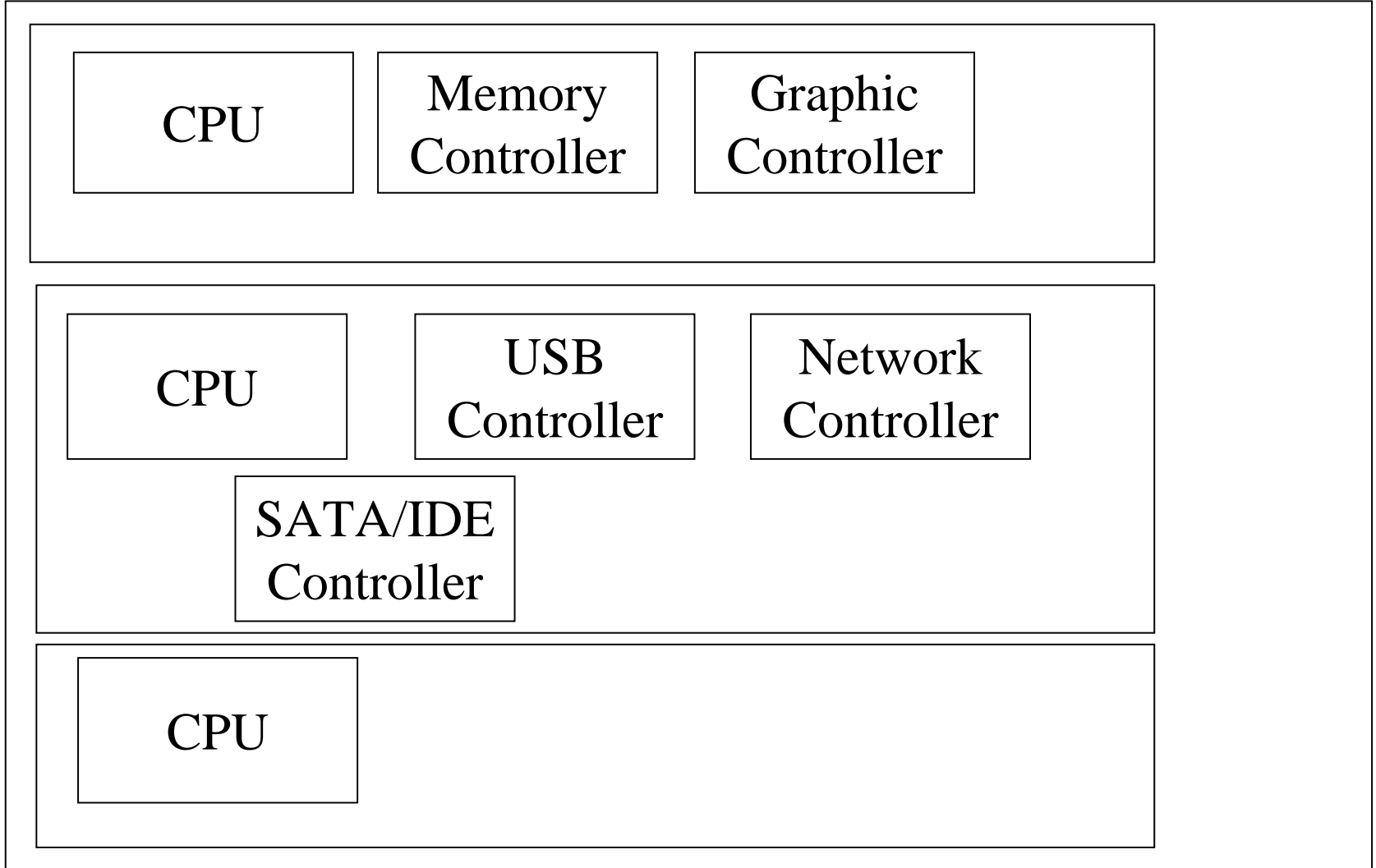
```
TA structure)
sizeof(struct TPM_SEALED_DATA));
```

```
{
    //decrypt and identify fields of the TPM_SEALED_DATA structure
    res = decrypt(blob, blob_size, unsealed_blob, key_index);
    if (!res) return FAILURE;
    [...]
    strcpy(sealed_key, unsealed_blob->data);
    [...]
    return SUCCESS;
}
return FAILURE;
}
```

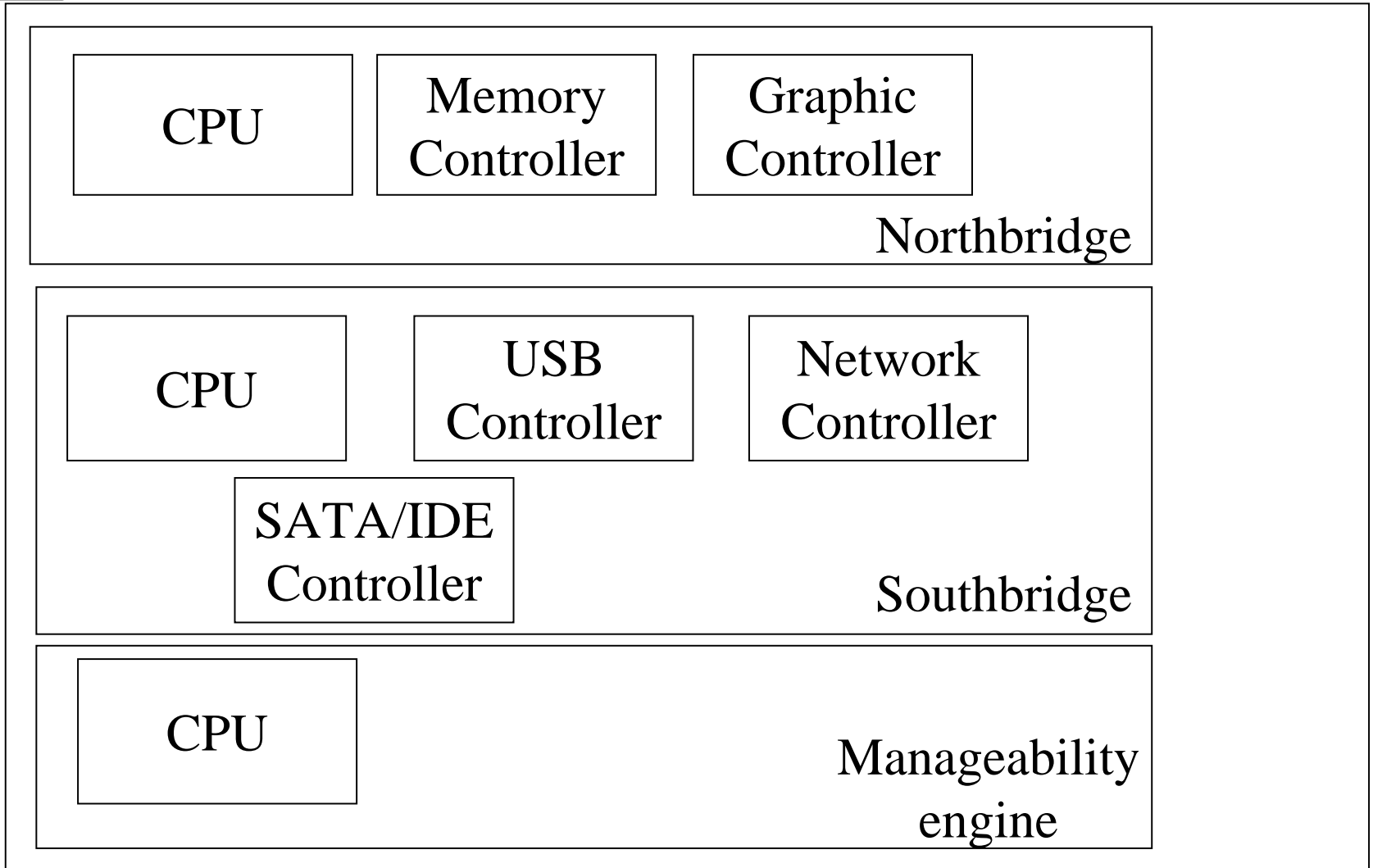
# Well known fact n°1 : hardware is secure

- Hardware conception is similar to software design:
  - Bugs can occur.
- Hardware circuits are really complex:
  - Some circuits are made of billions of transistors.
- Hardware components embed firmware:
  - Firmware is nothing but software.
  - Bugs can occur.
- So, why do we trust hardware?
  - Formal methods use is the design process?
  - In-depth analysis of the netlist or of the Hardware Description Language (HDL) code?
  - Because we have to?

# What is this?



# What is this?



# Well known fact n°2: hardware is simple

- Again, integrated circuits today are made of billions of transistors.
- Recent chipsets may encompass:
  - Several CPUs (ARC4 CPUs for instance).
  - A bunch of memory, buses and device controllers and different bridges.
  - Anti-virus/intrusion detection software.
  - A network stack.
  - A HTTP/HTTPS/SOAP server.
  - Remote administration functionalities.

# What does this do?

```
int main(void)
{
    iopl(3);
    outl(0xf, 0xb2);
    return 0;
}
```

# What does this do?

```
int main(void)
{
    iopl(3);
    outl(0xf, 0xb2);
    return 0;
}
```

# I don't know



# Explanation

- There is no way to find out (without exploiting a SMM – see later – vulnerability).
- The consequences of those two lines of code depend on the machine.
  - Feel free to open a tty and run it on your machine (with admin privileges).
  - (Disclaimer) but remember that you do that on your own will (don't blame me afterwards for the effect).
  - The truth is it will probably do exactly nothing.
- Let's find out what it does on my machine...

# AMT (Intel vPro) vulnerabilities

- Active Management Technology is a feature that allows an IT department to administrate/configure a platform even when no operating system is running.
- Therefore, some AMT components run in the chipset (the HTTPS/SOAP server I was mentioning before).
- See Alex Tereshkin and Rafal Wojtczuk Blackhat 2009 presentation (Ring -3 rootkits):
  - AMT firmware can be modified by the operating system kernel.
  - This allows rootkits to run code in the chipset.
  - Requires kernel privileges and takes advantage of a « patched » vulnerability.

# SMM

- System Management Mode is a mode of operation of x86 CPU.
- It is used to run power management code (SMI handler).
- SMI handlers are loaded in memory by the BIOS and protected by the chipset.
- When SMM code runs, the operating system is frozen (whole CPU context is saved and stored).
- If an attacker manages to run code in SMM, the attacker owns the machine.

# SMM vulnerabilities

- Security model at the hardware level may be flawed. SMM is a good example.
- In 2006 security features existed but were not used by BIOS vendors.
- Since then (some vendors) issued BIOS updates.
- In 2008 and 2009 various mechanisms were used to bypass those security measures.
  - See CanSecWest 2009 presentation « Getting into the SMRAM, SMM Reloaded » and (independant research) Joanna Rutkowska's blog.
    - We took advantage of a flawed repartition of security features between CPU and chipset.
    - The attack allowed a kernel level rootkit to hide code in the SMI handler supposed to be protected by the chipset.

# ACPI (seen from 10.000 miles)

- ACPI tables are BIOS-provided tables used by the OS Power Management component (OSPM).
- They are written in AML (ACPI Machine Language).
- The ACPI spec says that in order to put the first USB controller in S3 sleep state, the \\_SB.PCI0.USB0.S3 function in the DSDT table must be run.
- The function itself is specific to the machine.
- The OS has no way to find out if the function is really doing what it claims it is doing.
- Examples of ACPI rootkits: Trust 2009 paper (O. Levillain, B. Morin and myself).

# Well known fact n°3 : low level software/firmware can be trusted

- We never really know what are the exact functions that are embedded on a platform.
- Some low level firmwares cannot be verified but by the BIOS vendor itself.
  - SMM handler code.
- Again, low level software are not free of vulnerabilities:
  - Vulnerability in the BIOS BMP parser allowed an attacker to update Intel BIOS with unsigned code.
  - SMM vulnerabilities.
  - See Wojtczuk and Tereshkin Blackhat 2009 presentations.

# Impact on D-RTM based systems

- The SMM, ACPI and AMT attacks are efficient against promising architectures such as Intel TxT.
- By modifying AMT code, ACPI tables or SMM code, an attacker can find ways to leave a backdoor running on the system even after late launches are used.
  - The problem is not easy at all to solve.

# Well known fact n°4: hardware specifications are bulletproof

- Gürgens et al. ESORICS 2007 (Security Evaluation of scenarios based on the TCG's TPM specification):
  - They presented several weaknesses on the TPM interface.
- TPMs are vulnerable to offline dictionary attacks
  - Anti-hammering mechanisms exist that prevent (to some extent) an attacker from trying an active dictionary attack on a key authorization value.
  - But the attacker has enough information on the wire to carry out an offline attack.
  - For more details see M. Ryan & L. Chen Offline dictionary attack on TCG TPM weak authorization data and solution. “Future of trust in computing” 2008.
    - Or talk to Mark directly.



# Well known fact n°5 : virtualization improves the level of security

- That is true to some extent, but virtualization allows me to run on the same machines different OS/applications that I would have run on different machines otherwise.
- So even if virtualization isolation were perfect (no flaws, no vulnerabilities), the best I could hope for is the level of security I had in the first place.

# Bottom line of all this

- We need to put all those known facts to the question:
  - Is the hardware really that simple?
  - Can it be trusted?
  - Can the BIOS software be trusted?
  - What does virtualization really bring us?
- Only if we are able to answer those questions, will we really be able to trust a platform.

# So what do we do?

- We find ways to lower the impact of those axioms not being true:
  - Make it difficult for the attacker to run code on the machine:
    - Deactivate macros.
    - Do not provide compilers and development environment on production machines.
    - Restrict the number of applications allowed to be run on the machine.
    - Enforce such properties as  $W^X$ .
- Know the exact risk you are taking if one of your basic hypothesis fails to be true.

# One additional quick question



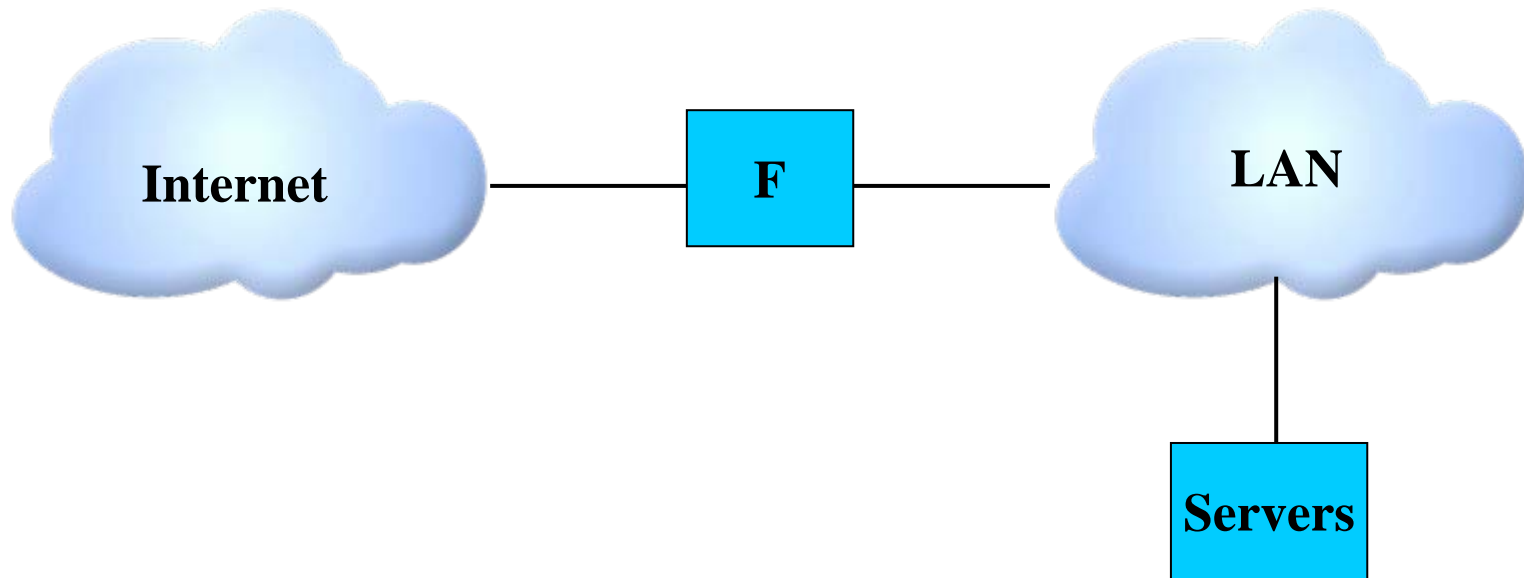
# Is my hard drive encrypted?

- There exists quite a lot of different seamless hard drive encryption products:
  - Microsoft's « Bitlocker Drive Encryption ».
  - Truecrypt.
  - Native Linux functions dm\_crypt.
  - Etc...
- But how can I, as a regular user, make sure that my hard drive is really encrypted?
  - If I could, encryption would not be that seamless.

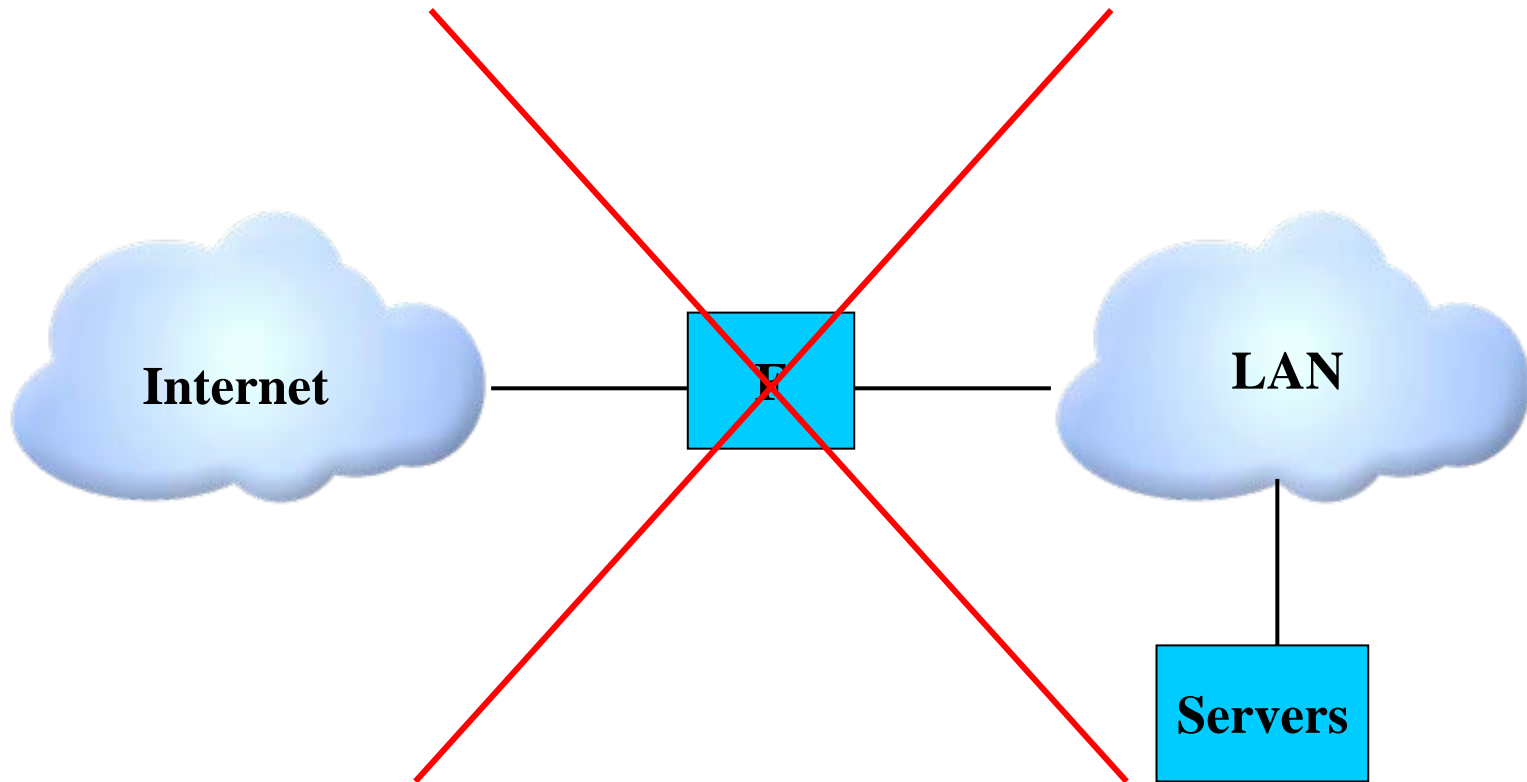
# Taking the system aspects into account



# Where should I put my IDS?



# Where should I put my IDS?



There is no good solution as the network architecture is a really bad one that should not be used in practice



# What is the best password?

- Bonjour
- Bonjour123
- 3!337\_Pwd@r00t
- AAAAAAAAAAAAAAAAAA

**Quoted from/Credits: Nicolas Ruff, SSTIC 2009**

# Software evaluation challenges

- Security evaluation of cryptographic components (Smart Cards, TPMs) can be done for instance in the Common Criteria scheme.
  - FNISA hosts the French Certification Body.
- Software products can also be evaluated this way.
- But what about PC platforms?
  - How can we evaluate them?
  - What about software updates? Do we have to go through another certification product if we update our BIOS?
  - What kind of assurance level can we get to?

# Conclusions

This page is intentionally left blank