CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, November 28th, 2011

Trustworthy computing

Overview

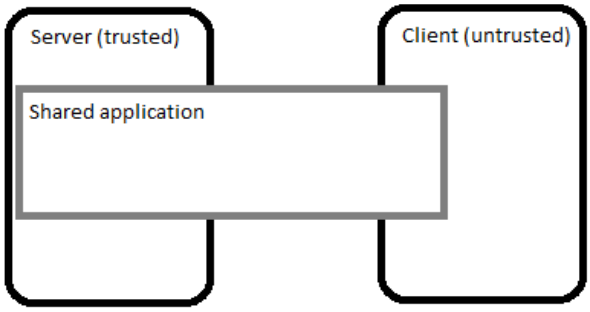
The original **motivation** behind trustworthy computing was **DRM (digital rights management)**: If I want to rent a movie to you, then once I give you the `.avi` file, how can I stop you from sharing it with other people?

If I sell you an **encrypted music file**, for your use only, then at some point you have to be able to play it, so I will have to give you the **decryption key**. The problem is, once you have the decryption key, what's to stop you from sharing the decrypted music file with other people?

Some systems used **obfuscation** in the way they stored the key to prevent people from reverse engineering the decryption process. So even if a user has the decryption key he should have a lot of trouble understanding how the decryption algorithm works. However, this is unacceptable, since once someone breaks the algorithm (and they will), everyone else will be able to share their movies.

Trustworthy computing

The idea behind trustworthy computing was that **you have a server** and **a client.** The server wants to run part of its application on the client, **but the client is not trusted by the server**. Also, **the platform owner is not trusted** (the person who owns the device on which the client runs).



Example: Imagine I want to send you an email, but I don't want you to forward it to other people. If everyone's computer had a special chip that could enforce certain rules, that are not necessarily the owner's rules, then you could implement something like this easily.

Digital signatures review

In a **digital signature scheme** the goal is for a verifier to successfully authenticate a message m sent and "signed" by a signer.

- There is a person who is the **signer** and a person who is the **verifier.**
- There is a **message** m.
- The signer has **a key-pair**, consisting of the **secret key** s and the **public key** p.
- The verifier, through some tamper-proof channel has received the public key p.
    - The public key p is actually known by everyone.
    - The secret key s is only known to the signer.
    - It is very difficult to compute s given p, despite the fact that the two are mathematically related.

The signer can compute $c = Sig_s(m)$ and she can send (m, c) to the verifier.

The verifier can take the signature c and verify it using $Vrfy(m, c)$, where $c = Sig_s(m')$

- If $m = m'$ then the the algorithm will say the signature is valid.

Observation: It is really hard without knowing the secret key s to compute $c = Sig_s(m)$ for a message m.

It's **analogous to a real signature**, because a real signature is very **hard to forge**. Also, when I sign a piece of paper then my signature is tied to the paper (non-repudiability).

Observation: Often times we hash the message m before signing it.

TCPA chip

The **TCPA chip** (also called a **TPM chip**) is the essential **piece of hardware** that enables trustworthy computing. Such a chip can be found on compliant systems such as IBM or Lenovo laptops for instance.

The TCPA has a **public-private key-pair** (S_T, P_T), which is written on the chip either at manufacturing-time or when the computer first boots. Your **OS and applications can access the public key** P_T.

Only the TCPA chip has access to the secret key S_T.

The TCPA chip contains **a special register called the PCR** (**platform configuration register**)

- Real chips contains about 24 of these

The PCR will be **used to hold information about the configuration of your system**.

Extend instruction

There is a special instruction that you can issue to the TCPA chip called **`extend(addr1, addr2)`**.

`extend(addr1, addr2)`replaces the PCR value with the hash of the current value in the PCR concatenated with the memory contents between `addr1` and `addr2`:

$$extend(pcr, addr1, addr2): pcr \leftarrow hash(pcr, memory[addr1 \dots addr2]);$$

Attest instruction

There is an instruction called **`attest(pcr)`** which signs the current PCR value using the secret key S_T of the TCPA chip.

$$attest(pcr): return\ sign_{S_T}(pcr, s);$$

One example of trustworthy computing using a TCPA chip

At manufacturing-time, Dell will turn on the computer and the TCPA chip will generate the public-private key-pair (S_T, P_T). Dell will then generate a **certificate** for the public key of the TCPA chip $C_T = Sig_{S_{Dell}}(P_T)$, where S_{Dell} is Dell's secret key.

This way **Dell is declaring that the public key P_T corresponds to one of its TCPA chips.**

Now, suppose I **want to convince someone that I am running Windows 7.**

- I will boot up Windows 7 and, at some point in the boot-up process (before system-dependent memory contents get loaded), Windows 7 will call `extend` on the loaded RAM contents.

- Now the PCR will hold a hash of the RAM.
- Windows 7 will then call `attest` obtaining a signature (under the secret key of the TCPA chip) on the hashed memory contents.
- Now when I want to convince Microsoft.com that I am running Windows 7, I can send it:

$$(PCR_{current}, A_T = attest(PCR_{current}), P_T, C_T)$$

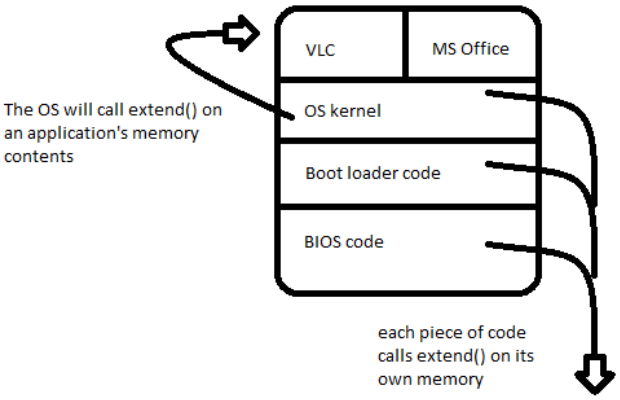
How does Microsoft.com ensure that I am not pulling any tricks?

Now, the **Microsoft.com server has the public key of Dell**, so it can verify the certificate C_T issued by Dell for P_T and this way it will know P_T is indeed the TCPA chip's public key.

Then it can check that A_T is a valid signature on $PCR_{current}$ and this way it will know that the message came from the machine with the TCPA chip and that the machine is running Windows 7.

This process is called **remote attestation.** The TCPA chip is allowing you to prove to a remote server that you are running a particular piece of software.

Software/trust stack



When your computer powers up:

- The CPU first **loads the BIOS**, then it **extends the BIOS image** into the PCR.
- Then the **BIOS loads the boot loader** and it **extends its memory code image into the PCR**.
    - Different BIOS-boot-loader pairs will result in different PCR values.
    - You can indicate what BIOS and boot loader you are using so that the server can verify your PCR values.
- Then the **boot-loader loads your OS kernel**, and part of its non-variable memory gets **extended into the PCR** too.

The TCPA chip is the **root of trust**, and propagates that trust up the stack all the way to the OS. However, if there is a **flaw in the OS, BIOS or boot-loader**, then the security of the system can be compromised.

After this is done, the **trust was propagated up** all the way to the OS-level. Therefore, **the OS can be used to attest other programs**.

Example: Windows Media Player can generate a public-private key pair and then it can ask the OS to attest that this key-pair corresponds to a WMP 10.7 process image with DRM rights management protection enabled.

CSE 409, Fall 2011, Rob Johnson, http://www.cs.stonybrook.edu/~rob/teaching/cse409-fa11/
Alin Tomescu, November 28th, 2011

Issues:

- Complicated trust
- Big trusted computing base
  - You are trusting the TCPA chip to store the secret key and provide the `extend`/`attest` interface
  - You are trusting the BIOS, boot-loader and OS code not to have any security flaws
  - You are trusting Dell's certificates not to be forgeable
- Complex to support software versions/variations/configurations
  - As a verifier like Microsoft.com, you have to store the expected PCR values for all the possible combinations of software
- Big brother
  - The public key of the TCPA chip P_T is a unique ID for your computer.
- Allows vendor lock in
  - It allows vendors to lock you to using certain applications
  - It easily enables vendors to force you to use their applications and make it hard for you to switch

Next: It'd be nice to use a system where you don't have to be locked in like this to see a movie.

Flicker and SKINIT instruction

Recent CPUs have a **special instruction called** SKINIT used by TCPA chips.

The **security objective of the SKINIT** instruction is to **establish** a **starting point for trusted operation**. More specifically, the SKINIT establishes a known execution environment, and then executes within this environment a known software object.

`SKINIT(addr, size)` will:

- disable DMA, interrupts and other stuff that would allow a software attack to tamper with it
- does a special `extend` on the buffer at the specified address `addr`
  - $PCR = hash(17, Mem[addr1, \dots, addr2])$, where 17 is some magic number.
- jumps to the address `addr`

More reading on SKINIT: http://download.microsoft.com/download/5/7/7/577a5684-8a83-43ae-9272-ff260a9c20e2/AMD_WinHEC-2003_whitepaper.doc

How can you play a movie securely?

The movie playing service will send you a small chunk of code (`play1frame`) that takes a single encrypted frame of video, decrypts it and displays it on the screen. `play1frame` will use **sealed storage** to decrypt the frame.

The OS will do an SKINIT on the chunk of code and this way that code will run securely: it will be the only thing running and will not be preempted.