System Security
Solution: Trusted Computing and Software-Based Attestation

George Cojocar

Distribution: 07.12.2015
Hand in: 17.12.2015, 13:15

1 Static and Dynamic Root of Trust

Consider the secure cloud service provider "TrustedCloud", located in Libya. TrustedCloud offers secure computing services by enabling on all their servers TPM version 1.2. For computations performed on TrustedCloud, cloud customers can verify system attestations using either a **static root of trust** or a **dynamic root of trust**. The attestations customers receive are signed by a key that is certified by Infineon (a reliable TPM manufacturer) to belong to a valid TPM chip.

Does such a system provide cloud clients with secure cloud computing? Provide your solution by answering the following points:

(a) Does the static root of trust system provide cloud customers with secure cloud computing? Describe what security properties one can achieve through static root of trust.

Solution:

Not, because the static root of trust measures the integrity of boot sequence and typically the operating system of cloud instances is virtualized. Most virtualization technologies do not require a boot loader on the virtual machines (e.g. Xen). Also normally the cloud services are long running applications which would require a restart before doing any sensitive operations.

The security property one can achieve through static root of trust is the integrity of the platform at load time. It verifies the code at initial loading time, but does no provide protection against the exploits at run-time. Such kind of exploits at run-time can compromise the system state.

(b) Does the dynamic root of trust system provide cloud customers with secure cloud computing? Describe what security properties one can achieve through dynamic root of trust.

Solution:

The dynamic root of trust provides more security guaranties for cloud customers but it is not entirely secure. It eliminates the BIOS/Bootloader from the chain of trust when launching a Virtual Machine or a sensitive operation on a untrusted system.

Using dynamic root of trust, remote attestation can be done on the status of the Virtual Machines. The Intel Trusted Execution Technology guarantees that a Virtual Machine accessing a confident information is unmodified.

(c) If you believe that the static or dynamic root of trust system is not secure, briefly describe an attack that TrustedCloud could mount.

Solution:

In case of static root of trust, the TrustedCloud could mount a *reset attack* by trying to zero out the PCRs without resetting the machine. Typically the PCRs resides on a *Low Pin Count (LPC)* bus. The LPC bus supports a ground driven reset line. This means when this particular line on the bus is driven to ground every device on this bus is supposed to reset.

Also the dynamic root of trust can be attacked. A known attack of Intel's TXT modifies the *SMM (System Management Mode) Memory* before late lunched. The SMM has access to whole physical memory and is the most privileged mode on the CPU. The SMM Memory is protected by the chipset, but however on some chipsets bugs allow to bypass those protections. In this way one can execute arbitrary code in the context of the "late launched" Virtual Machines.

2 `BIOS` compromise

Assume Malta passes a law where all wireless access routers have to be equipped with a TPM. A manufacturer of cheap hardware, D-Linksys, realizes that there is a critical bug in the BIOS of their router.

(a) Explain whether the TPM can help circumvent the bug. Your solution must explain either **why** the TPM cannot help circumvent the bug, or **how** the TPM can help.

Solution:

The TPM can monitor the boot process by measuring the integrity of BIOS code executed during the startup. Also it allows to D-Linksys to verify (attest) remotely the platform configuration. These measures might protect against some exploits which alter the code or configuration of BIOS, but in general do no provide any security against software vulnerabilities resulted from design or coding errors.

(b) Eventually, D-Linksys decides to update all BIOSes of their hardware. Rather than recalling all products, the update procedure is carried online as follows:

- For configuration purposes, the wireless router runs a web server with SSL. The administrator user connects to the web server, authenticates it with SSL, and clicks on a button to update the BIOS. Note that the web server is not accessible outside the private network.

- Once the update button has been pressed, the wireless access router connects, via HTTP, to the manufacturer's web server `update.d-linksys.com` and downloads the file `BIOS.COM`.
- The wireless router re-flashes the BIOS with `BIOS.COM`, and reboots. The update procedure is, at this point, complete.

No other steps are being carried out. Is the update procedure secure? Justify your answer, by either arguing for its security, or by giving an example of an attack against it.

Solution:

This update procedure is not secure. The file `BIOS.COM` can be modified in transit by an attacker because is transferred in clear via HTTP. As a result, the router is going to be re-flashed with a malicious version of BIOS. In this case a TPM would help to verify the integrity of `BIOS.COM` file and would allow to D-Linksys to attest remotely that the correct version of BIOS was upgraded.

3 Verifiable code execution via software-based attestation

Consider the following steps of verifiable code execution, from the Pioneer paper discussed in class:

1. Initialization function, receive nonce from verifier
2. Checksum code
3. Post processing
4. Send result to verifier
5. Compute hash function
6. Send result to verifier
7. Execute

Reason about the following:

(a) Which parts 1-7 need to be included in the checksum computation and why? Which parts in the hash computation and why?

Solution:

The checksum code computes a checksum over the entire verification function which consists of parts 1, 2, 3, 4 and 5. It sets up an execution environment in which the send function, the hash function and the executable are guaranteed to run untampered by any malicious software on the untrusted platform.

The part 7, execute, is included in hash function computation. The hash function is used to perform integrity measurement of the executable.

(b) The Pioneer paper states that the code to replace exception handlers cannot be placed in the checksum loop, and also not in the initialization code, because the attacker could simply skip these instructions. However, since the verification function is computed over the checksum initializaton, loop, and epilog code, how can you explain that the attacker can skip these exception handler replacement instructions without the result being visible in the checksum?

Solution:

The attacker can skip the instructions which replace the exception handlers in the initialization code by setting an execution break-point in the code. The processor will generate a debug exception when trying to execute the code. In this way, the adversary can control the execution flow of the program.

The instructions replacing the exception handlers (e.g *iret* on x86) in the checksum loop do not affect the value of the checksum. For this reason, the adversary can simply remove these instructions and still compute the correct checksum within the expected time.

(c) The paper states, however, that the handler replacement instructions can be placed in the epilog code. In this setting, consider an attacker sets a debug breakpoint between steps 2 and 3, so right after the checksum is computed, the attacker gains control. Thus, it seems the attacker can skip the instructions as before. What could you do about it?

Solution:

A break point right after the checksum computation, it will increase the time within the verifier receives the checksum. The verifier detects such a situation by measuring the time elapsed from sending the nonce until receiving the checksum. If this time exceeds a certain threshold, the verifier skips the checksum verification and exits with a failure.

(d) The verifier wants to periodically perform verifications. What security properties do checksum verification requests need to satisfy? How could you perform this efficiently?

Solution:

The security verification requests need to satisfy the checksum *freshness*. The checksum must depend on a unpredictable *challenge* sent by the verifier. This measure prevents that the adversary pre-computes the checksum before making changes to the verification function, and replies old checksum values. This is achieved in two ways. First, the checksum code uses the challenge to seed a Pseudo-Random Number Generator that generates input for computing the checksum. Second, the challenge is also used to initialise the checksum variable to deterministic yet unpredictable value.