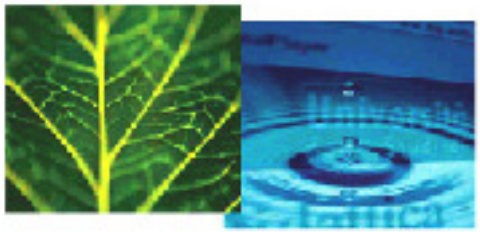



# *Hypothetical*
# Trust and Attack Models

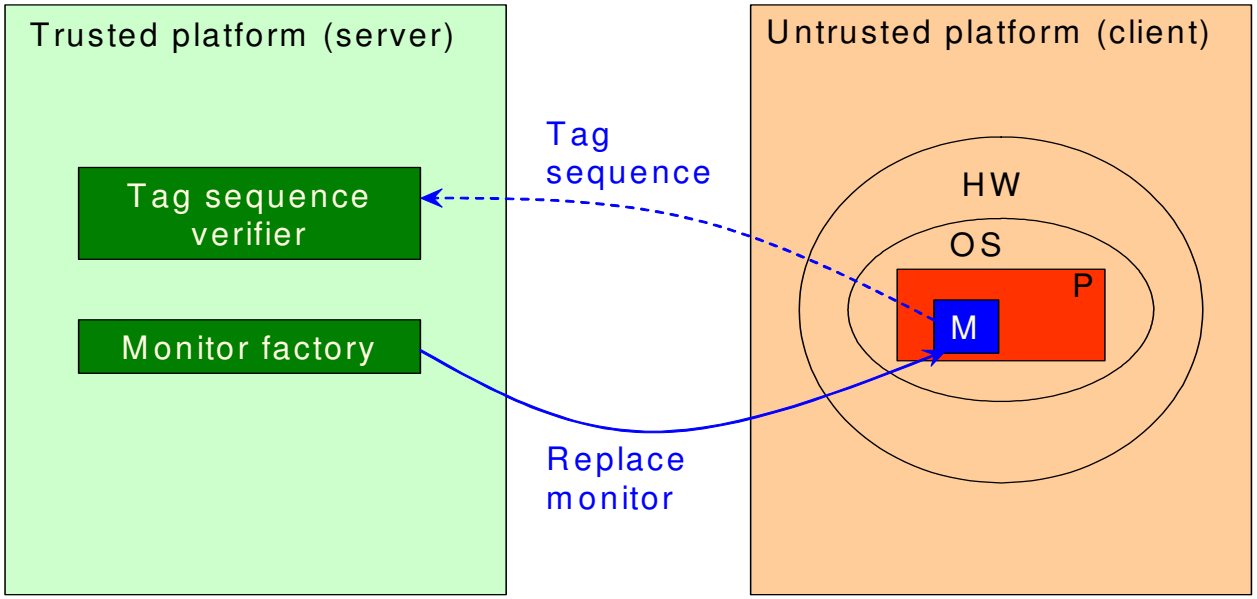
Mariano Ceccato (1), Christian Collberg (2), Paolo Tonella (1)
*(1) ITC-irst, Trento, Italy*
*(2) University of Arizona, USA*
ceccato@itc.it, tonella@itc.it, collberg@cs.arizona.edu

# The remote entrusting problem

- *Remote software authentication*: ensuring a trusted machine (server) that an untrusted host (client) is running a “healthy” version of a program ***P***:

- The program is unadulterated.
- It is executed on top of unadulterated HW/SW.
- The execution process is not manipulated externally.

- The distinctive feature of *remote* entrusting is that **the authenticated software needs to communicate over the network** with the trusted machine to work properly.

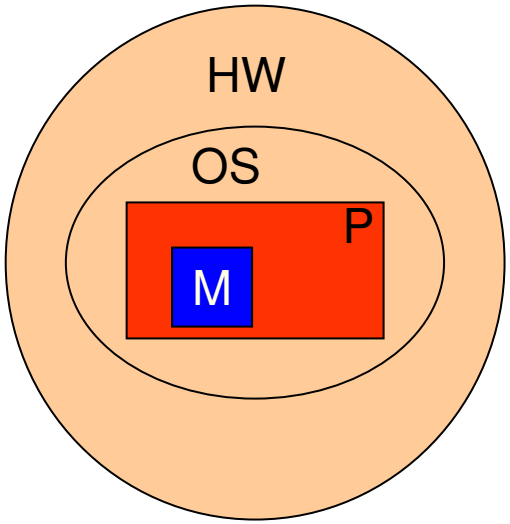


# Sources of trust

# Authenticity verification

**The monitor *M* should verify:**

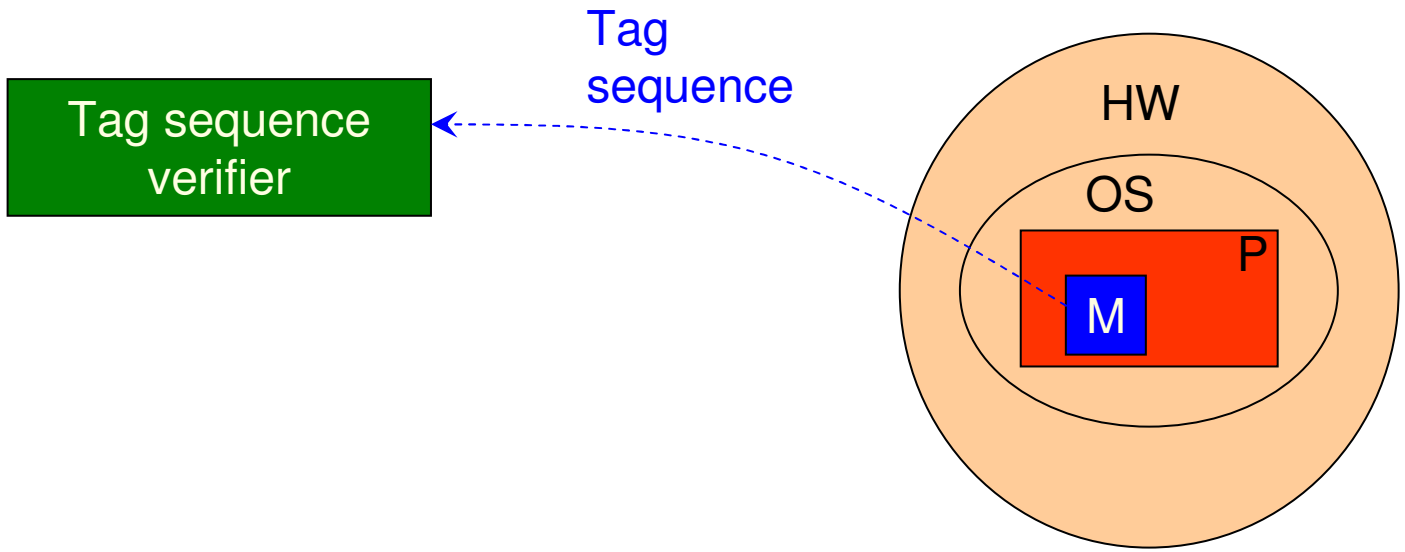
- Text and data segments of *P* as loaded in memory.
- Libraries used by *P*.
- The execution environment (HW, OS, execution process, etc.).
- Results of specific computations or assertions.



# Tag sequence generation

The monitor *M* sends the server an authenticity tag sequence as evidence of healthy execution:

- Tags have limited time validity.
- A secret key, hidden into M itself, is used to generate them.
- If no tag or an incorrect tag is received by the server, the client is considered untrusted and the service delivery is suspended as a countermeasure.



# Replacement

To give attackers a limited time to succeed, the monitor *M* is periodically replaced:

- The duration depends on the estimated reverse engineering complexity, assuming humans are necessarily involved in the process.
- The monitor factory should generate highly independent monitors.

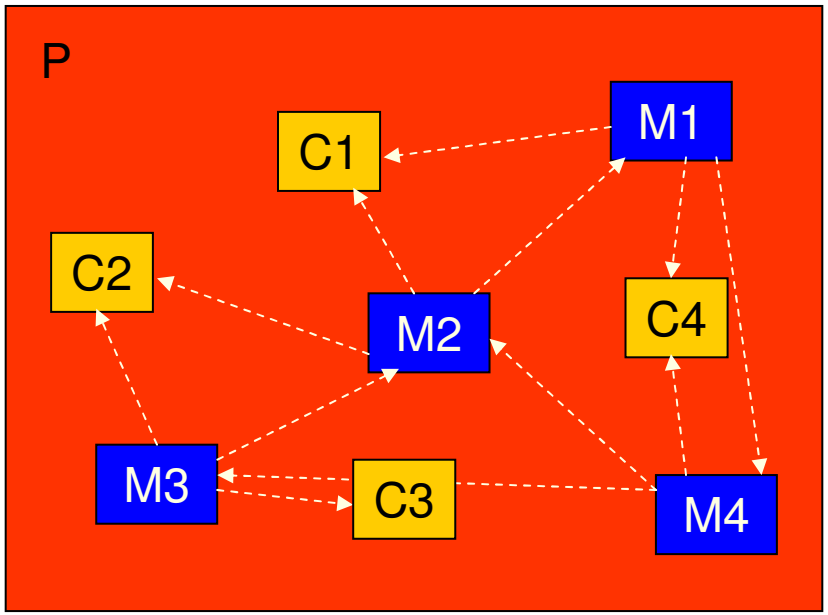
# Code obfuscation

To increase the resistance to reverse engineering, the code is obfuscated:

- Opaque predicates based on conditions that are hard to analyze statically (e.g., involving pointer structures) could be used.

# Additional sources of trust

- Self checking monitor: M checks itself before checking P.
- Tags include data verified by server: authenticity verification is no longer local to M.
- Server sends challenge C to client: tag generation and authenticity verification depend on C.
- Network of trust:



# Attacks

# Assumptions on attacker

A malicious user can:

- Give wrong information to the server about its hardware.
- Install any software on the client.
- Read and write memory locations, processor registers and files.
- Observe and modify the network traffic.
- Modify P and M, both on disk and in memory.
- Use any available code analysis tool.
- Take advantage of tracers, emulators and debuggers.
- Tamper with libraries, operating system and hardware.

A malicious user cannot:

- Access and tamper with the trusted server.
- Know the software/hardware configuration of the server.

# Classes of attacks

1. Reverse engineering attack.
2. Execution environment attack.
3. Cloning attack.
4. Differential analysis attack

# Reverse engineering attacks

Important functionalities and data structures are located and altered maliciously in *P* and *M*:

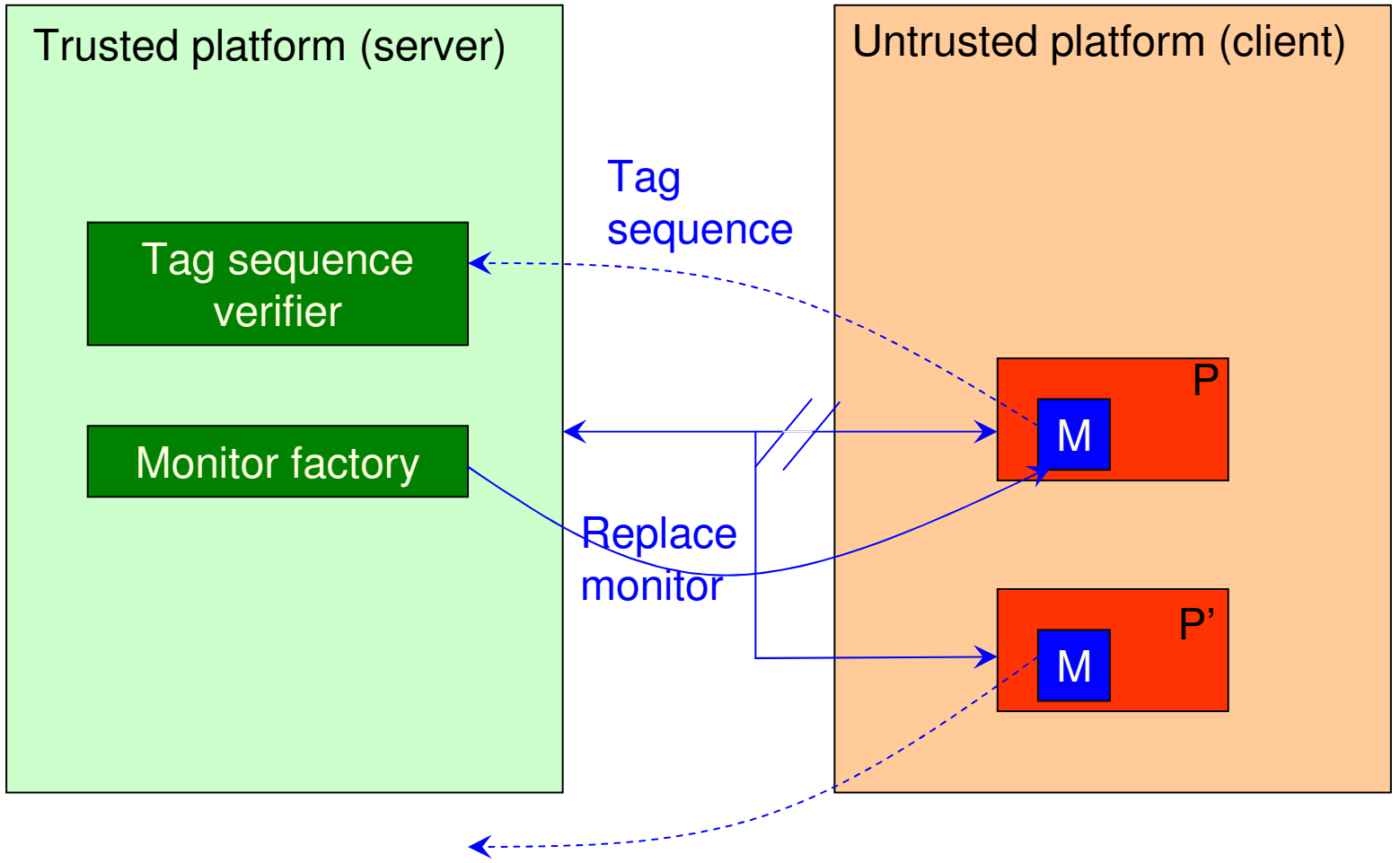
- Tag sequence generator.
- Authenticity checking functions.
- Secret keys.
- Input data (e.g., passed to checking functions).
- Output data (e.g., returned by checking functions).

# Execution environment attacks

*P* is run on an emulator, in debug mode or is interpreted by an adulterated virtual machine:

- Memory locations, call stack, program counter and parameters can be altered dynamically.
- Dynamic libraries can be altered maliciously.
- Input and output values can be replaced on-the-fly.

# Cloning attack




This attack is ineffective if tag sequence includes computation data.

# Differential analysis attack

The attacker gathers information about *M* by comparing the sequence of monitors delivered by the monitor factory in the past:

- If the strategy used by the monitor factory is (even only partially) understood, the time necessary to break new monitors might be reduced, eventually allowing the attacker to break a still valid monitor.

# Analysis of attack resistance

| Sources of trust | Attacks | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reverse engineering attacks | | | | | | | Execution environment attacks | | | | |
| | P is tampered with (1) | Replace checking function (2) | Replace tag sequence generator (3) | Modify input before call on M/P (4) | Modify input before call on env. (5) | Modify output before return on M/P (6) | Modify output before return on env. (7) | Replace HW/OS (8) | Replace dynamic libraries (9) | Tampered execution (debug mode) (10) | Cloning attack (11) | Differential analysis (12) |
| (1) M checks P text and data segment | X | | | | | | | | | | | |
| (2) M self checks itself before checking P | | X | X | X | | X | | | | | | |
| (3) M checks libraries used by P | | | | | | | | | X | | | |
| (4) M checks execution environment | | | | | X | | X | | | X | | |
| (5) M checks the OS and the HW | | | | | | | | X | | | | |
| (6) M checks results of computation | X | | | X | X | X | X | X | X | X | | |
| (7) Secret key used to generate the tag sequence | | | X | | | | | | | | | |
| (8) Monitor replacement | | X | X | X | X | X | X | | | | | X |
| (9) Rev-eng resistance (code obfuscation) | X | X | X | X | X | X | X | X | X | X | X | X |
| (10) Network of trust (self-checking implementation) | | X | X | X | | X | | | | | | |
| (11) Tags include (portion of) output | | X | | | | X | X | | | | X | |
| (12) Bi-directional communication (challenge from the server) | | X | X | X | X | X | X | | | | | |