Attack Detection in a Remote Entrusting Scenario

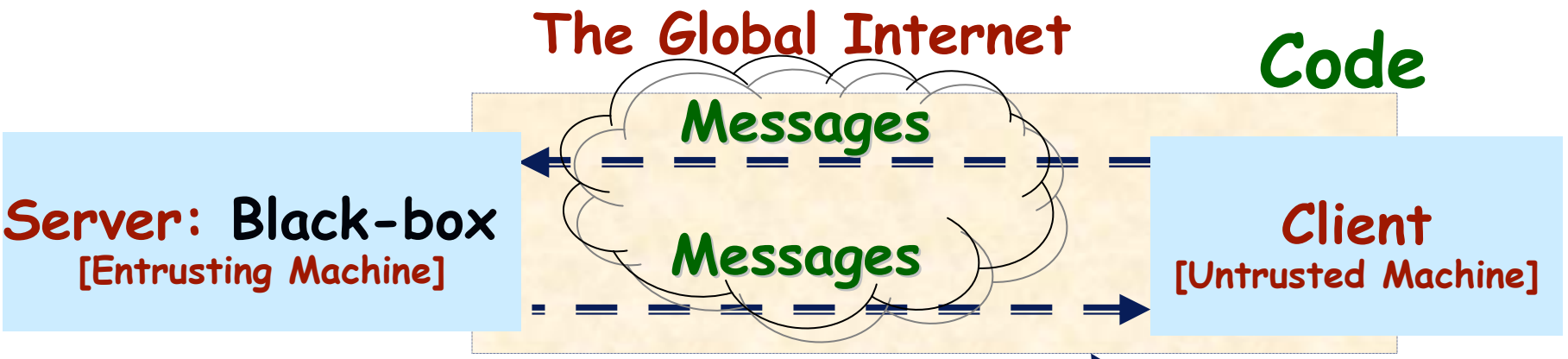
First International Workshop on Remote Entrusting
October 15-16, 2008
***Villa Madruzzo* - Trento - Italy**

Yoram OFEK (University of Trento - Italy), Anirban MAJUMDAR (University of Trento - Italy), Christian COLLBERG (University of Arizona - USA), Alessandro ZORAT (University of Trento - Italy)

Outline

- (Extended) remote entrusting model
- Attack
- Counter-Attack
- Defense against the Counter-Attack
- …
- …

(Extended) Remote Entrusting Model



- Assume: client-server scenario
  - 1. Server black box
  - 2. Constant flow of messages
- EXECUTION INTERLOCKING (assumption):
  Client unable to correctly execute the code without exchanging messages with the server black box

Client Attack

- Attack: reverse engineering on **client code**
  - Specifically: analysis of the client code by **gathering information through** observing code behavior
- Assumption: attacker is observing the code execution interactively using tools, such as:
  - Debuggers
  - Tracers
  - Etc.
- Objective: using the **gathered information** for tampering with the **client code**

Server Objective

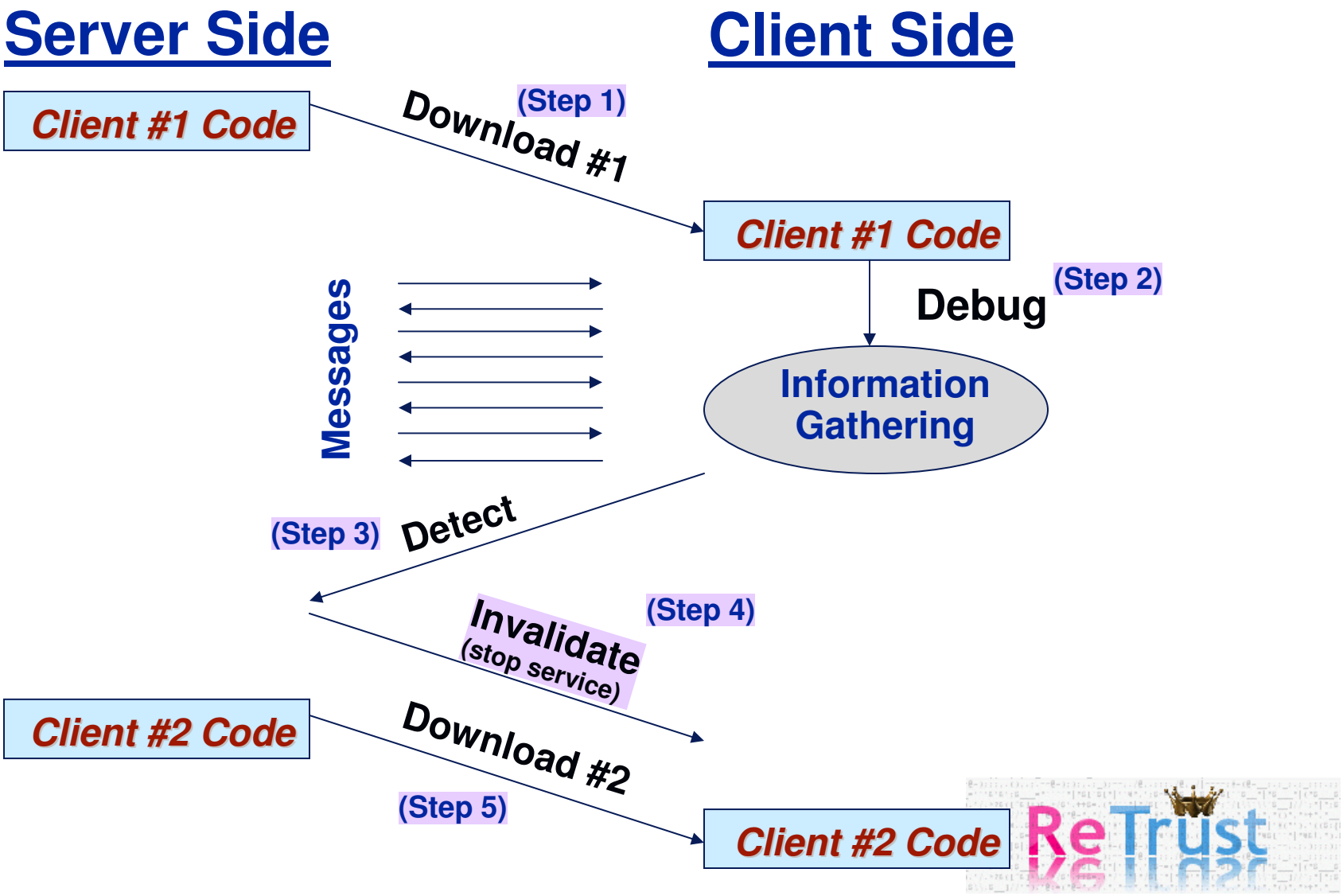
- To detect that the attacker is gathering information about the **client code**

- NOTE:
  - 1. In remote entrusting scenario attack detection was not done before
  - 2. Previously the focus was on attack prevention
    - Moving **client code** to server-side
    - Continuous replacement of selected parts of client code to slow analysis

Server Detection Model

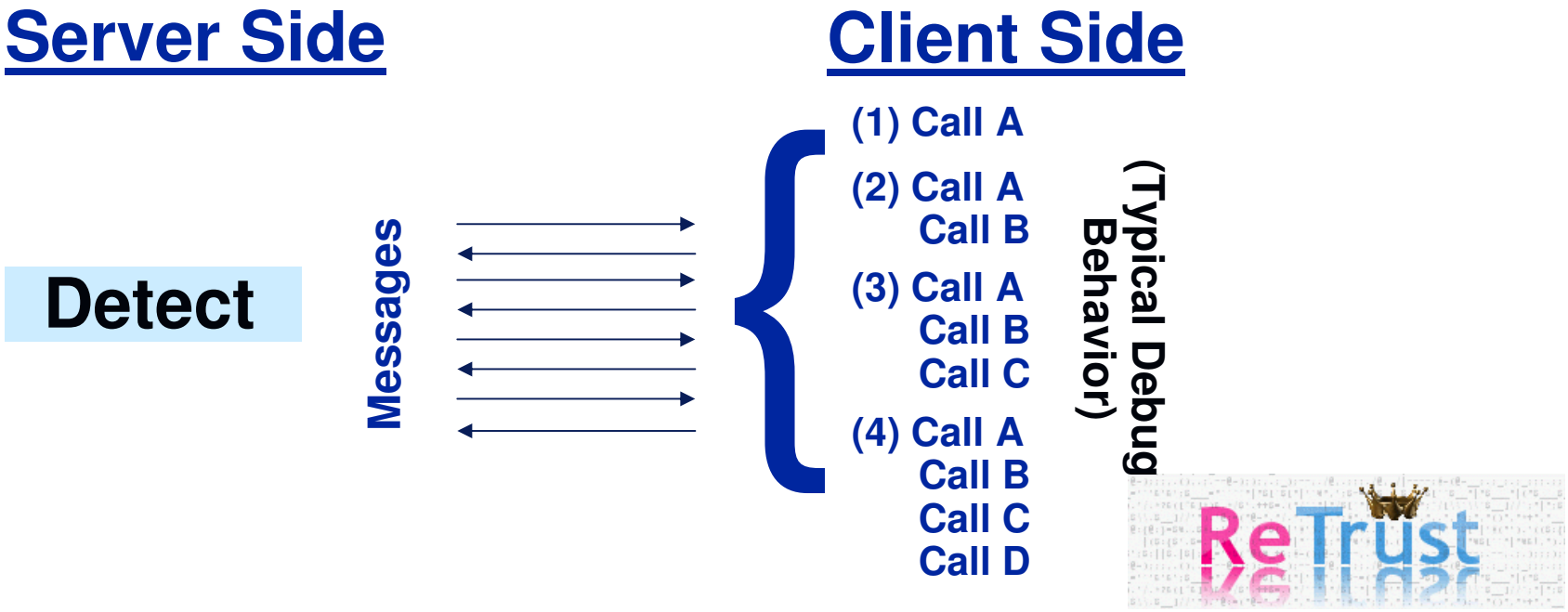
- Server with detection module, which the attacker has no access to: **black-box**
  - **Step 1:** Client download copy of the code
  - **Step 2:** The attacker gathering and analyzing information (e.g., using a debugger)
  - **Step 3:** The server detects some **anomalies** of the client behavior **from the exchanged messages**
  - **Step 4:** Then the server invalidates the client copy of the code
  - **Step 5:** The client must download a new copy and the attack will have to start all over again (due to diversification)

Server Detection Model (2)



Scenario 1 – Server Detection

- The server detection module "knows" what to expect, and consequently, detect **anomalous behavior** by inspecting the flow/sequence of messages
  - By, for example, constructing regular expressions (similar to intrusion detection?)



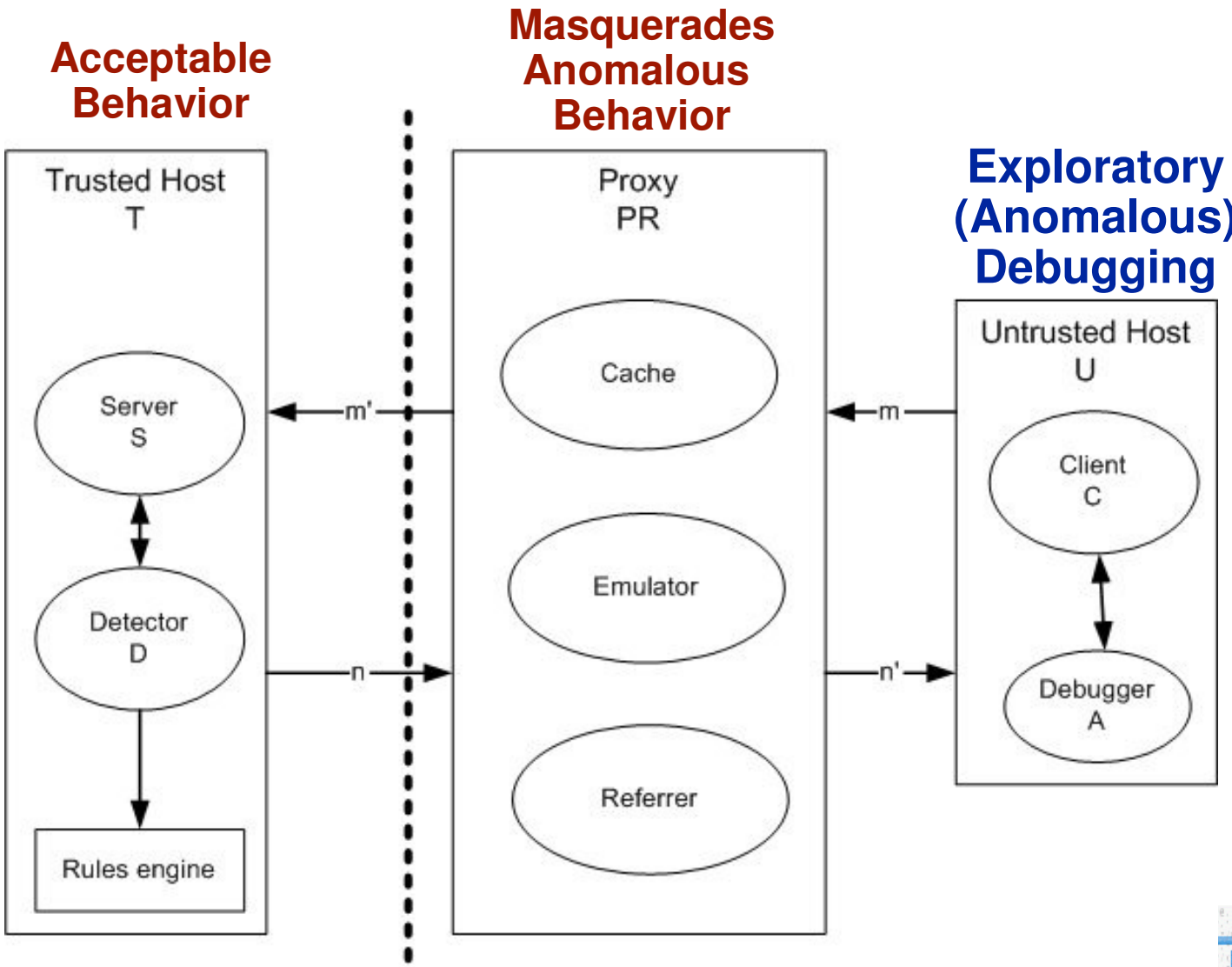
Scenario 2 – Attacker Counter-Measures

- The attacker learn what is typical behavior by observing correct execution of the client code
- Then the attacker

  masquerades anomalous behavior

  as **acceptable behavior** to the server by, for example:
  - Proxy
  - Simulation
  - Playback
  - Caching
  - … …

Scenario 2 – Attacker Counter-Measures



Scenario 3 – Server Counter- Measures

- **Goal #1**: make it infeasible for the client to learn acceptable **message pattern**
  - Add extraneous messages to the code – **message obfuscation**
    - Increasing the client-server REQUIRED interaction
- **Goal #2**: to detect exploratory debugging
  - Add illegal (bogus copies) execution paths
  - **Direct client to previously "unused" code modules – "new" execution paths (cloned copies)**
    - Expending the execution space
      - E.g., by copying transformation
    - Detect execution paths that should not be taken

Scenario 2* - Attacker Counter-Measures

- By using a **clone** (copy) of the **code**
  - Synchronize execution
- Then the attacker

  masquerade anomalous behavior

  as **acceptable behavior by sending messages from the clone code**

Scenario 3* – Server Counter-Measures

- Goal: infeasible to clone by
  adding randomization to the code
- …
- …
- …

Summary: General Methodology

- Code + transformation
  - Indirect
  - Copy (cloned and bogus)
  - Mapping (e.g., encryption)
  - Replacement
  - … …
- Storing critical transformation information in the server **black-box**
  - **Required for correct execution**
- Such that:
  learning and/or cloning is not possible