# CWE Detail – CWE-602

## Description

The product is composed of a server that relies on the client to implement a mechanism that is intended to protect the server.

## Extended Description

When the server relies on protection mechanisms placed on the client side, an attacker can modify the client-side behavior to bypass the protection mechanisms, resulting in potentially unexpected interactions between the client and server. The consequences will vary, depending on what the mechanisms are trying to protect.

## Threat-Mapped Scoring

Score: 0.0

Priority: Unclassified

## Observed Examples (CVEs)

**•** CVE-2022-33139: SCADA system only uses client-side authentication, allowing adversaries to impersonate other users.

**•** CVE-2006-6994: ASP program allows upload of .asp files by bypassing client-side checks.

**•** CVE-2007-0163: steganography products embed password information in the carrier file, which can be extracted from a modified client.

**•** CVE-2007-0164: steganography products embed password information in the carrier file, which can be extracted from a modified client.

**•** CVE-2007-0100: client allows server to modify client's configuration and overwrite arbitrary files.

## Related Attack Patterns (CAPEC)

* CAPEC-162
* CAPEC-202
* CAPEC-207
* CAPEC-208
* CAPEC-21
* CAPEC-31
* CAPEC-383
* CAPEC-384
* CAPEC-385
* CAPEC-386
* CAPEC-387
* CAPEC-388

## Attack TTPs

**•** T1539: Steal Web Session Cookie (Tactics: credential-access)

**•** T1528: Steal Application Access Token (Tactics: credential-access)

**•** T1134: Access Token Manipulation (Tactics: defense-evasion, privilege-escalation)

**•** T1056.004: Credential API Hooking (Tactics: collection, credential-access)

## Modes of Introduction

**•** Architecture and Design: COMMISSION: This weakness refers to an incorrect design related to an architectural security tactic.

**•** Architecture and Design: Consider a product that consists of two or more processes or nodes that must interact closely, such as a client/server model. If the product uses protection schemes in the client in order to defend from attacks against the server, and the server does not use the same schemes, then an attacker could modify the client in a way that bypasses those schemes. This is a fundamental design flaw that is primary to many weaknesses.

## Common Consequences

**•** Impact: Bypass Protection Mechanism, DoS: Crash, Exit, or Restart — Notes: Client-side validation checks can be easily bypassed, allowing malformed or unexpected input to pass into the application, potentially as trusted data. This may lead to unexpected states, behaviors and possibly a resulting crash.

**•** Impact: Bypass Protection Mechanism, Gain Privileges or Assume Identity — Notes: Client-side checks for authentication can be easily bypassed, allowing clients to escalate their access levels and perform unintended actions.

## Potential Mitigations

**•** Architecture and Design: For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. Even though client-side checks provide minimal benefits with respect to server-side security, they are still useful. First, they can support intrusion detection. If the server receives input that should have been rejected by the client, then it may be an indication of an attack. Second, client-side error-checking can provide helpful feedback to the user about the expectations for valid input. Third, there may be a reduction in server-side processing time for accidental input errors, although this is typically a small savings. (Effectiveness: N/A)

**•** Architecture and Design: If some degree of trust is required between the two entities, then use integrity checking and strong authentication to ensure that the inputs are coming from a trusted source. Design the product so that this trust is managed in a centralized fashion, especially if there are complex or numerous communication channels, in order to reduce the risks that the implementer will mistakenly omit a check in a single code path. (Effectiveness: N/A)

**•** Testing: Use dynamic tools and techniques that interact with the software using large test suites with many diverse inputs, such as fuzz testing (fuzzing), robustness testing, and fault injection. The software's operation may slow down, but it should not become unstable, crash, or generate incorrect results. (Effectiveness: N/A)

**•** Testing: Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. (Effectiveness: N/A)

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** CLIENT-SIDE (client.pl)

**•** Multiple vendors used client-side authentication in their OT products.