# CWE Detail – CWE-863

## Description

The product performs an authorization check when an actor attempts to access a resource or perform an action, but it does not correctly perform the check.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2021-39155: Chain: A microservice integration and management platform compares the hostname in the HTTP Host header in a case-sensitive way (CWE-178, CWE-1289), allowing bypass of the authorization policy (CWE-863) using a hostname with mixed case or other variations.

**•** CVE-2019-15900: Chain: sscanf() call is used to check if a username and group exists, but the return value of sscanf() call is not checked (CWE-252), causing an uninitialized variable to be checked (CWE-457), returning success to allow authorization bypass for executing a privileged (CWE-863).

**•** CVE-2009-2213: Gateway uses default "Allow" configuration for its authorization settings.

**•** CVE-2009-0034: Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.

**•** CVE-2008-6123: Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.

**•** CVE-2008-7109: Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.

**•** CVE-2008-3424: Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.

**•** CVE-2008-4577: ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.

**•** CVE-2006-6679: Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.

**•** CVE-2005-2801: Chain: file-system code performs an incorrect comparison (CWE-697), preventing default ACLs from being properly applied.

**•** CVE-2001-1155: Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.

## Modes of Introduction

**•** Architecture and Design: Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic. A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

**•** Operation: N/A

## Common Consequences

**•** Impact: Read Application Data, Read Files or Directories — Notes: An attacker could bypass intended access restrictions to read sensitive data, either by reading the data directly from a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.

**•** Impact: Modify Application Data, Modify Files or Directories — Notes: An attacker could bypass intended access restrictions to modify sensitive data, either by writing the data directly to a data store that is not correctly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.

**•** Impact: Gain Privileges or Assume Identity, Bypass Protection Mechanism — Notes: An attacker could bypass intended access restrictions to gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.

**•** Impact: Execute Unauthorized Code or Commands — Notes: An attacker could use elevated privileges to execute unauthorized commands or code.

**•** Impact: DoS: Crash, Exit, or Restart, DoS: Resource Consumption (CPU), DoS: Resource Consumption (Memory), DoS: Resource Consumption (Other) — Notes: An attacker could gain unauthorized access to resources on the system and excessively consume those resources, leading to a denial of service.

## Potential Mitigations

**•** Architecture and Design: Divide the product into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) [REF-229] to enforce the roles at the appropriate boundaries. Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role. (Effectiveness: N/A)

**•** Architecture and Design: Ensure that access control checks are performed related to the business logic. These checks may be different than the access control checks that are applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor [REF-7]. (Effectiveness: N/A)

**•** Architecture and Design: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, consider using authorization frameworks such as the JAAS Authorization Framework [REF-233] and the OWASP ESAPI Access Control feature [REF-45]. (Effectiveness: N/A)

**•** Architecture and Design: For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page. One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page. (Effectiveness: N/A)

**•** System Configuration: Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** The programmer expects that the cookie will only be set when getRole() succeeds. The programmer even diligently specifies a 2-hour expiration for the cookie. However, the attacker can easily set the "role" cookie to the value "Reader". As a result, the $role variable is "Reader", and getRole() is never invoked. The attacker has bypassed the authorization system.

## Notes

**•** Terminology: Assuming a user with a given identity, authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.