# CWE Detail – CWE-306

## Description

The product does not perform any authentication for functionality that requires a provable user identity or consumes a significant amount of resources.

## Extended Description

N/A

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2024-11680: File-sharing PHP product does not check if user is logged in during requests for PHP library files under an includes/ directory, allowing configuration changes, code execution, and other impacts. (KEV)

**•** CVE-2022-31260: Chain: a digital asset management program has an undisclosed backdoor in the legacy version of a PHP script (CWE-912) that could allow an unauthenticated user to export metadata (CWE-306)

**•** CVE-2022-29951: TCP-based protocol in Programmable Logic Controller (PLC) has no authentication.

**•** CVE-2022-29952: Condition Monitor firmware uses a protocol that does not require authentication.

**•** CVE-2022-30276: SCADA-based protocol for bridging WAN and LAN traffic has no authentication.

**•** CVE-2022-30313: Safety Instrumented System uses proprietary TCP protocols with no authentication.

**•** CVE-2022-30317: Distributed Control System (DCS) uses a protocol that has no authentication.

**•** CVE-2021-21972: Chain: Cloud computing virtualization platform does not require authentication for upload of a tar format file (CWE-306), then uses .. path traversal sequences (CWE-23) in the file to access unexpected files, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2020-10263: Bluetooth speaker does not require authentication for the debug functionality on the UART port, allowing root shell access

**•** CVE-2021-23147: WiFi router does not require authentication for its UART port, allowing adversaries with physical access to execute commands as root

**•** CVE-2021-37415: IT management product does not perform authentication for some REST API requests, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2020-13927: Default setting in workflow management product allows all API requests without authentication, as exploited in the wild per CISA KEV. (KEV)

**•** CVE-2002-1810: MFV. Access TFTP server without authentication and obtain configuration file with sensitive plaintext information.

**•** CVE-2008-6827: Agent software running at privileges does not authenticate incoming requests over an unprotected channel, allowing a Shatter" attack.

**•** CVE-2004-0213: Product enforces restrictions through a GUI but not through privileged APIs.

**•** CVE-2020-15483: monitor device allows access to physical UART debug port without authentication

**•** CVE-2019-9201: Programmable Logic Controller (PLC) does not have an authentication feature on its communication protocols.

## Related Attack Patterns (CAPEC)

* CAPEC-12
* CAPEC-166
* CAPEC-216
* CAPEC-36
* CAPEC-62

## Modes of Introduction

**•** Architecture and Design: OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase.

**•** Architecture and Design: Developers sometimes perform authentication at the primary channel, but open up a secondary channel that is assumed to be private. For example, a login mechanism may be listening on one network port, but after successful authentication, it may open up a second port where it waits for the connection, but avoids authentication because it assumes that only the authenticated party will connect to the port.

**•** Operation: When migrating data to the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), there is a risk of losing the protections that were originally provided by hosting on internal networks. If access does not require authentication, it can be easier for attackers to access the data from anywhere on the Internet.

## Common Consequences

**•** Impact: Gain Privileges or Assume Identity, Varies by Context — Notes: Exposing critical functionality essentially provides an attacker with the privilege level of that functionality. The consequences will depend on the associated functionality, but they can range from reading or modifying sensitive data, accessing administrative or other privileged functionality, or possibly even executing arbitrary code.

## Potential Mitigations

**•** Architecture and Design: Divide the software into anonymous, normal, privileged, and administrative areas. Identify which of these areas require a proven user identity, and use a centralized authentication capability. Identify all potential communication channels, or other means of interaction with the software, to ensure that all channels are appropriately protected, including those channels that are assumed to be accessible only by authorized parties. Developers sometimes perform authentication at the primary channel, but open up a secondary channel that is assumed to be private. For example, a login mechanism may be listening on one network port, but after successful authentication, it may open up a second port where it waits for the connection, but avoids authentication because it assumes that only the authenticated party will connect to the port. In general, if the software or protocol allows a single session or user state to persist across multiple connections or channels, authentication and appropriate credential management need to be used throughout. (Effectiveness: N/A)

**•** Architecture and Design: For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. 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. (Effectiveness: N/A)

**•** Architecture and Design: Where possible, avoid implementing custom, "grow-your-own" authentication routines and consider using authentication capabilities as provided by the surrounding framework, operating system, or environment. These capabilities may avoid common weaknesses that are unique to authentication; support automatic auditing and tracking; and make it easier to provide a clear separation between authentication tasks and authorization tasks. In environments such as the World Wide Web, the line between authentication and authorization is sometimes blurred. If custom authentication routines are required instead of those provided by the server, then these routines must be applied to every single page, since these pages could be requested directly. (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 libraries with authentication capabilities such as OpenSSL or the ESAPI Authenticator [REF-45]. (Effectiveness: N/A)

**•** Implementation: When storing data in the cloud (e.g., S3 buckets, Azure blobs, Google Cloud Storage, etc.), use the provider's controls to require strong authentication for users who should be allowed to access the data [REF-1297] [REF-1298] [REF-1302]. (Effectiveness: N/A)

## Applicable Platforms

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

## Demonstrative Examples

**•** However, there is no authentication mechanism to ensure that the user creating this bank account object has the authority to create new bank accounts. Some authentication mechanisms should be used to verify that the user has the authority to create bank account objects.

**•** Multiple vendors did not use any authentication for critical functionality in their OT products.

**•** While it was not publicly disclosed how the data was protected after discovery, multiple options could have been considered.