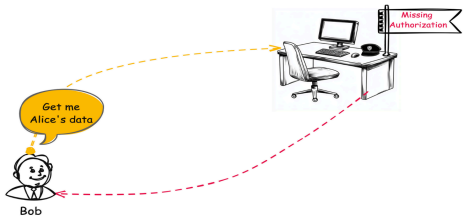
CWE-862: Missing Authorization

Weakness ID: 862
Vulnerability Mapping: ALLOWED (with careful review of mapping notes)
Abstraction: Class

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product does not perform an authorization check when an actor attempts to access a resource or perform an action.



Alternate Terms

AuthZ	"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is distinct from "AuthN" (or, sometimes, "AuthC") which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Common Consequences

Impact	Details
Read Application Data; Read Files or Directories	**Scope: Confidentiality** An attacker could read sensitive data, either by reading the data directly from a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to read the data.
Modify Application Data; Modify Files or Directories	**Scope: Integrity** An attacker could modify sensitive data, either by writing the data directly to a data store that is not restricted, or by accessing insufficiently-protected, privileged functionality to write the data.
Gain Privileges or Assume Identity; Bypass Protection Mechanism	**Scope: Access Control** An attacker could gain privileges by modifying or reading critical data directly, or by accessing privileged functionality.
DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory); DoS: Resource Consumption (Other)	**Scope: Availability** An attacker could gain unauthorized access to resources on the system and excessively consume those resources, leading to a denial of service.

Potential Mitigations

Phase(s)	Mitigation
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.

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].
Architecture and Design	**Strategy: *Libraries or Frameworks*** 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].
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.
System Configuration; Installation	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.

Relationships

Relevant to the view "Research Concepts" (View-1000)

Nature	Type	ID	Name
ChildOf	C	285	Improper Authorization
ParentOf	B	425	Direct Request ('Forced Browsing')
ParentOf	C	638	Not Using Complete Mediation
ParentOf	B	939	Improper Authorization in Handler for Custom URL Scheme
ParentOf	B	1314	Missing Write Protection for Parametric Data Values

Relevant to the view "Weaknesses for Simplified Mapping of Published Vulnerabilities" (View-1003)

Nature	Type	ID	Name
MemberOf	V	1003	Weaknesses for Simplified Mapping of Published Vulnerabilities
ParentOf	B	425	Direct Request ('Forced Browsing')

Relevant to the view "Architectural Concepts" (View-1008)

Nature	Type	ID	Name
MemberOf	C	1011	Authorize Actors

Relevant to the view "CISQ Data Protection Measures" (View-1340)

Nature	Type	ID	Name
ChildOf	P	284	Improper Access Control

Background Details

An access control list (ACL) represents who/what has permissions to a given object. Different operating systems implement (ACLs) in different ways. In UNIX, there are three types of permissions: read, write, and execute. Users are divided into three classes for file access: owner, group owner, and all other users where each class has a separate set of rights. In Windows NT, there are four basic types of permissions for files: "No access", "Read access", "Change access", and "Full control". Windows NT extends the concept of three types of users in UNIX to include a list of users and groups along with their associated permissions. A user can create an object (file) and assign specified permissions to that object.

Modes Of Introduction

Phase	Note
Architecture and Design	OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase. Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.
Implementation	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	

Applicable Platforms

Languages	Class: Not Language-Specific *(Undetermined Prevalence)*
Technologies	AI/ML *(Undetermined Prevalence)* Web Server *(Often Prevalent)* Database Server *(Often Prevalent)* Class: Not Technology-Specific *(Undetermined Prevalence)*

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

This function runs an arbitrary SQL query on a given database, returning the result of the query.

Example Language: **PHP** *(bad code)*

```
function runEmployeeQuery($dbName, $name){
   mysql_select_db($dbName,$globalDbHandle) or die("Could not open Database".$dbName);
   //Use a prepared statement to avoid CWE-89
   $preparedStatement = $globalDbHandle->prepare('SELECT * FROM employees WHERE name = :name');
   $preparedStatement->execute(array(':name' => $name));
   return $preparedStatement->fetchAll();
}
/.../

$employeeRecord = runEmployeeQuery('EmployeeDB',$_GET['EmployeeName']);
```

While this code is careful to avoid SQL Injection, the function does not confirm the user sending the query is authorized to do so. An attacker may be able to obtain sensitive employee information from the database.

Example 2

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

Example Language: **Perl** *(bad code)*

```
sub DisplayPrivateMessage {
   my($id) = @_;
   my $Message = LookupMessageObject($id);
   print "From: " . encodeHTML($Message->{from}) . "<br>\n";
   print "Subject: " . encodeHTML($Message->{subject}) . "\n";
   print "<hr>\n";
   print "Body: " . encodeHTML($Message->{body}) . "\n";
}

my $q = new CGI;
# For purposes of this example, assume that CWE-309 and


# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
   ExitError("invalid username or password");
}

my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other

users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Selected Observed Examples

Note: this is a curated list of examples for users to understand the variety of ways in which this weakness can be introduced. It is not a complete list of all CVEs that are related to this CWE entry.

Reference	Description
CVE-2024-6845	chatbot Wordpress plugin does not perform authorization on a REST endpoint, allowing retrieval of an API key
CVE-2025-2224	AI-enabled WordPress plugin has a missing capability check for a particular function, allowing changing public status of posts
CVE-2022-24730	Go-based continuous deployment product does not check that a user has certain privileges to update or create an app, allowing adversaries to read sensitive repository information
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.
CVE-2009-2282	Terminal server does not check authorization for guest access.
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms.
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files.
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files.
CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations.
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-2005-1036	Chain: Bypass of access restrictions due to improper authorization (CWE-862) of a user results from an improperly initialized (CWE-909) I/O permission bitmap
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files.
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.
CVE-2020-17533	Chain: unchecked return value (CWE-252) of some functions for policy enforcement leads to authorization bypass (CWE-862)

Weakness Ordinalities

Ordinality	Description
Primary	*(where the weakness exists independent of other weaknesses)*

Detection Methods

Method	Details
Automated Static Analysis	Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries. Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives. **Effectiveness: Limited**
Automated Dynamic Analysis	Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic.
Manual Analysis	This weakness can be detected using 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. Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms. **Effectiveness: Moderate** **Note:** These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.
Manual Static Analysis - Binary or Bytecode	According to SOAR [REF-1479], the following detection techniques may be useful: Cost effective for partial coverage: • Binary / Bytecode disassembler - then use manual analysis for vulnerabilities & anomalies **Effectiveness: SOAR Partial**
Dynamic Analysis with Automated Results Interpretation	According to SOAR [REF-1479], the following detection techniques may be useful: Cost effective for partial coverage: • Web Application Scanner • Web Services Scanner • Database Scanners **Effectiveness: SOAR Partial**

Dynamic Analysis with Manual Results Interpretation	According to SOAR [REF-1479], the following detection techniques may be useful: Cost effective for partial coverage: • Host Application Interface Scanner • Fuzz Tester • Framework-based Fuzzer **Effectiveness: SOAR Partial**
Manual Static Analysis - Source Code	According to SOAR [REF-1479], the following detection techniques may be useful: Cost effective for partial coverage: • Focused Manual Spotcheck - Focused manual analysis of source • Manual Source Code Review (not inspections) **Effectiveness: SOAR Partial**
Automated Static Analysis - Source Code	According to SOAR [REF-1479], the following detection techniques may be useful: Cost effective for partial coverage: • Source code Weakness Analyzer • Context-configured Source Code Weakness Analyzer **Effectiveness: SOAR Partial**
Architecture or Design Review	According to SOAR [REF-1479], the following detection techniques may be useful: Highly cost effective: • Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.) • Formal Methods / Correct-By-Construction **Effectiveness: High**

Memberships

Nature	Type	ID	Name
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References
MemberOf	C	817	OWASP Top Ten 2010 Category A8 - Failure to Restrict URL Access
MemberOf	C	866	2011 Top 25 - Porous Defenses
MemberOf	V	884	CWE Cross-section
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	C	1345	OWASP Top Ten 2021 Category A01:2021 - Broken Access Control
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	V	1387	Weaknesses in the 2022 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	C	1396	Comprehensive Categorization: Access Control
MemberOf	V	1425	Weaknesses in the 2023 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	V	1430	Weaknesses in the 2024 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	V	1435	Weaknesses in the 2025 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	C	1436	OWASP Top Ten 2025 Category A01:2025 - Broken Access Control

Vulnerability Mapping Notes

Usage	**ALLOWED-WITH-REVIEW** *(this CWE ID could be used to map to real-world vulnerabilities in limited situations requiring careful review)*
Reason	Abstraction
Rationale	This CWE entry is a Class and might have Base-level children that would be more appropriate
Comments	Examine children of this entry to see if there is a better fit

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
ISA/IEC 62443	Part 2-1		Req 4.3.3.7
ISA/IEC 62443	Part 3-3		Req SR 2.1
ISA/IEC 62443	Part 4-2		Req CR 2.1

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-665	Exploitation of Thunderbolt Protection Flaws

References

[REF-229]	NIST. *"Role Based Access Control and Role Based Security".* <https://csrc.nist.gov/projects/role-based-access-control>. (*URL validated: 2023-04-07*)
[REF-7]	Michael Howard and David LeBlanc. *"Writing Secure Code".* Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft Press. 2002-12-04. <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223>.
[REF-231]	Frank Kim. *"Top 25 Series - Rank 5 - Improper Access Control (Authorization)".* SANS Software Security Institute. 2010-03-04. <https://www.sans.org/blog/top-25-series-rank-5-improper-access-control-authorization>. (*URL validated: 2023-04-07*)
[REF-45]	OWASP. *"OWASP Enterprise Security API (ESAPI) Project".* <https://owasp.org/www-project-enterprise-security-api/>. (*URL validated: 2025-07-24*)
[REF-233]	Rahul Bhattacharjee. *"Authentication using JAAS".* <https://javaranch.com/journal/2008/04/authentication-using-JAAS.html>. (*URL validated: 2023-04-07*)
[REF-62]	Mark Dowd, John McDonald and Justin Schuh. *"The Art of Software Security Assessment".* Chapter 2, "Common Vulnerabilities of Authorization", Page 39. 1st Edition. Addison Wesley. 2006.
[REF-1479]	Gregory Larsen, E. Kenneth Hong Fong, David A. Wheeler and Rama S. Moorthy. *"State-of-the-Art Resources (SOAR) for Software Vulnerability Detection, Test, and Evaluation".* 2014-07. <https://www.ida.org/-/media/feature/publications/s/st/stateoftheart-resources-soar-for-software-vulnerability-detection-test-and-evaluation/p-5061.ashx>. (*URL validated: 2025-09-05*)

Content History

Submissions

Submission Date	Submitter	Organization
2011-05-24 *(CWE 1.13, 2011-06-01)*	CWE Content Team	MITRE

Contributions

Contribution Date	Contributor	Organization
2023-04-25	"Mapping CWE to 62443" Sub-Working Group *Suggested mappings to ISA/IEC 62443.*	CWE-CAPEC ICS/OT SIG
2024-02-29 *(CWE 4.16, 2024-11-19)*	Abhi Balakrishnan *Provided diagram to improve CWE usability*	

Submissions

Modifications