CWE-522: Insufficiently Protected Credentials

Weakness ID: 522
Vulnerability Mapping: ALLOWED **(with careful review of mapping notes)**
Abstraction: Class

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

Description

The product transmits or stores authentication credentials, but it uses an insecure method that is susceptible to unauthorized interception and/or retrieval.

Common Consequences

Scope	Impact	Likelihood
Access Control	**Technical Impact:** *Gain Privileges or Assume Identity* An attacker could gain access to user accounts and access sensitive data used by the user accessible.	

Potential Mitigations

Phase: Architecture and Design

Use an appropriate security mechanism to protect the credentials.

Phase: Architecture and Design

Make appropriate use of cryptography to protect the credentials.

Phase: Implementation

Use industry standards to protect the credentials (e.g. LDAP, keystore, etc.).

Relationships

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

Nature	Type	ID	Name
ChildOf	B	668	Exposure of Resource to Wrong Sphere
ChildOf	B	1390	Weak Authentication
ParentOf	B	256	Plaintext Storage of a Password
ParentOf	B	257	Storing Passwords in a Recoverable Format
ParentOf	B	260	Password in Configuration File
ParentOf	B	261	Weak Encoding for Password
ParentOf	B	523	Unprotected Transport of Credentials
ParentOf	B	549	Missing Password Field Masking

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

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

Modes Of Introduction

Phase	Note
Architecture and Design Implementation	COMMISSION: This weakness refers to an incorrect design related to an architectural security tactic.

Applicable Platforms

Languages

Class: Not Language-Specific *(Undetermined Prevalence)*

Technologies

Class: ICS/OT *(Undetermined Prevalence)*

Demonstrative Examples

Example 1

This code changes a user's password.

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

```
$user = $_GET['user'];
$pass = $_GET['pass'];
$checkpass = $_GET['checkpass'];
if ($pass== $checkpass) {
  SetUserPassword($user, $pass);
}
```

While the code confirms that the requesting user typed the same new password twice, it does not confirm that the user requesting the password change is the same user whose password will be changed. An attacker can request a change of another user's password and gain control of the victim's account.

Example 2

The following code reads a password from a properties file and uses the password to connect to a database.

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

```
...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...
```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Example 3

The following code reads a password from the registry and uses the password to create a new network credential.

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

```
...
String password = regKey.GetValue(passKey).toString();
NetworkCredential netCred = new NetworkCredential(username,password,domain);
...
```

This code will run successfully, but anyone who has access to the registry key used to store the password can read the value of password. If a devious employee has access to this information, they can use it to break into the system

Example 4

Both of these examples verify a password by comparing it to a stored compressed version.

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

```
int VerifyAdmin(char *password) {
  if (strcmp(compress(password), compressed_password)) {
    printf("Incorrect Password!\n");
    return(0);
  }
  printf("Entering Diagnostic Mode...\n");
  return(1);
}
```

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

```
int VerifyAdmin(String password) {
  if (passwd.Equals(compress(password), compressed_password)) {
    return(0);
  }
  //Diagnostic Mode
  return(1);
}
```

Because a compression algorithm is used instead of a one way hashing algorithm, an attacker can recover compressed passwords stored in the database.

Example 5

The following examples show a portion of properties and configuration files for Java and ASP.NET applications. The files include username and password information but they are stored in cleartext.
This Java example shows a properties file with a cleartext username / password pair.

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

```
# Java Web App ResourceBundle properties file
...
webapp.ldap.username=secretUsername
webapp.ldap.password=secretPassword
...
```

The following example shows a portion of a configuration file for an ASP.Net application. This configuration file includes username and password information for a connection to a database but the pair is stored in cleartext.

Example Language: **ASP.NET** *(bad code)*

```
...
<connectionStrings>
  <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=password; dbalias=uDB;" providerName="System.Data.Odbc" />
</connectionStrings>
...
```

Username and password information should not be included in a configuration file or a properties file in cleartext as this will allow anyone who can read the file access to the resource. If possible, encrypt this information.

Example 6

In 2022, the OT:ICEFALL study examined products by 10 different Operational Technology (OT) vendors. The researchers reported 56 vulnerabilities and said that the products were "insecure by design" [REF-1283]. If exploited, these vulnerabilities often allowed adversaries to change how the products operated, ranging from denial of service to changing the code that the products executed. Since these products were often used in industries such as power, electrical, water, and others, there could even be safety implications.
Multiple vendors used cleartext transmission or storage of passwords in their OT products.

Observed Examples

Reference	Description
CVE-2022-30018	A messaging platform serializes all elements of User/Group objects, making private information available to adversaries
CVE-2022-29959	Initialization file contains credentials that can be decoded using a "simple string transformation"
CVE-2022-35411	Python-based RPC framework enables pickle functionality by default, allowing clients to unpickle untrusted data.
CVE-2022-29519	Programmable Logic Controller (PLC) sends sensitive information in plaintext, including passwords and session tokens.
CVE-2022-30312	Building Controller uses a protocol that transmits authentication credentials in plaintext.
CVE-2022-31204	Programmable Logic Controller (PLC) sends password in plaintext.
CVE-2022-30275	Remote Terminal Unit (RTU) uses a driver that relies on a password stored in plaintext.
CVE-2007-0681	Web app allows remote attackers to change the passwords of arbitrary users without providing the original password, and possibly perform other unauthorized actions.
CVE-2000-0944	Web application password change utility doesn't check the original password.
CVE-2005-3435	product authentication succeeds if user-provided MD5 hash matches the hash in its database; this can be subjected to replay attacks.
CVE-2005-0408	chain: product generates predictable MD5 hashes using a constant value combined with username, allowing authentication bypass.

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	718	OWASP Top Ten 2007 Category A7 - Broken Authentication and Session Management
MemberOf	C	724	OWASP Top Ten 2004 Category A3 - Broken Authentication and Session Management
MemberOf	V	884	CWE Cross-section
MemberOf	C	930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management
MemberOf	C	963	SFP Secondary Cluster: Exposed Data
MemberOf	C	1028	OWASP Top Ten 2017 Category A2 - Broken Authentication
MemberOf	V	1337	Weaknesses in the 2021 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	C	1348	OWASP Top Ten 2021 Category A04:2021 - Insecure Design
MemberOf	V	1350	Weaknesses in the 2020 CWE Top 25 Most Dangerous Software Weaknesses
MemberOf	C	1396	Comprehensive Categorization: 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

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2007	A7	CWE More Specific	Broken Authentication and Session Management
OWASP Top Ten 2004	A3	CWE More Specific	Broken Authentication and Session Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-102	Session Sidejacking
CAPEC-474	Signature Spoofing by Key Theft
CAPEC-50	Password Recovery Exploitation
CAPEC-509	Kerberoasting
CAPEC-551	Modify Existing Service
CAPEC-555	Remote Services with Stolen Credentials
CAPEC-560	Use of Known Domain Credentials
CAPEC-561	Windows Admin Shares with Stolen Credentials
CAPEC-600	Credential Stuffing
CAPEC-644	Use of Captured Hashes (Pass The Hash)
CAPEC-645	Use of Captured Tickets (Pass The Ticket)
CAPEC-652	Use of Known Kerberos Credentials
CAPEC-653	Use of Known Operating System Credentials

References

[REF-44] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 19: Use of Weak Password-Based Systems." Page 279. McGraw-Hill. 2010.

[REF-1283] Forescout Vedere Labs. "OT:ICEFALL: The legacy of "insecure by design" and its implications for certifications and risk management". 2022-06-20. <https://www.forescout.com/resources/ot-icefall-report/>.

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	Anonymous Tool Vendor (under NDA)	

Modifications

Page Last Updated: July 16, 2024

Use of the Common Weakness Enumeration (CWE™) and the associated references from this website are subject to the Terms of Use. CWE is sponsored by the U.S. Department of Homeland Security (DHS) Cybersecurity and Infrastructure Security Agency (CISA) and managed by the Homeland Security Systems Engineering and Development Institute (HSSEDI) which is operated by The MITRE Corporation (MITRE). Copyright © 2006–2024, The MITRE Corporation. CWE, CWSS, CWRAF, and the CWE logo are trademarks of The MITRE Corporation.