



A Community-Developed List of Software Weakness Types

[Home](#)[About](#)[CWE List](#)[Scoring
Search](#)[Community](#)[News](#)**Weakness ID: 311****Abstraction:** Base**Structure:** Simple**Status:** Draft**Presentation Filter:** Basic ▼

▼ Description

The software does not encrypt sensitive or critical information before storage or transmission.

▼ Extended Description

The lack of proper data encryption passes up the guarantees of confidentiality, integrity, and accountability that properly implemented encryption conveys.

▼ Relationships

The table(s) below shows the weaknesses and high level categories that are related to this weakness. These relationships are defined as ChildOf, ParentOf, MemberOf and give insight to similar items that may exist at higher and lower levels of abstraction. In addition, relationships such as PeerOf and CanAlsoBe are defined to show similar weaknesses that the user may want to explore.

- ▶ **Relevant to the view "Research Concepts" (CWE-1000)**
- ▶ **Relevant to the view "Architectural Concepts" (CWE-1008)**
- ▶ **Relevant to the view "Development Concepts" (CWE-699)**

▼ Modes Of Introduction

The different Modes of Introduction provide information about how and when this weakness may be introduced. The Phase identifies a point in the software life cycle at which introduction may occur, while the Note provides a typical scenario related to introduction during the given phase.

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

▼ Applicable Platforms

The listings below show possible areas for which the given weakness could appear. These may be for specific named Languages, Operating Systems, Architectures, Paradigms, Technologies, or a class of such platforms. The platform is listed along with how frequently the given weakness appears for that instance.

Languages

Class: Language-Independent (*Undetermined Prevalence*)

▼ Common Consequences

The table below specifies different individual consequences associated with the weakness. The Scope identifies the application security area that is violated, while the Impact describes the negative technical impact that arises if an adversary succeeds in exploiting this weakness. The Likelihood provides information about how likely the specific consequence is expected to be seen relative to the other consequences in the list. For example, there may be high likelihood

that a weakness will be exploited to achieve a certain impact, but a low likelihood that it will be exploited to achieve a different impact.

Scope	Impact	Likelihood
Confidentiality	Technical Impact: <i>Read Application Data</i> If the application does not use a secure channel, such as SSL, to exchange sensitive information, it is possible for an attacker with access to the network traffic to sniff packets from the connection and uncover the data. This attack is not technically difficult, but does require physical access to some portion of the network over which the sensitive data travels. This access is usually somewhere near where the user is connected to the network (such as a colleague on the company network) but can be anywhere along the path from the user to the end server.	
Confidentiality Integrity	Technical Impact: <i>Modify Application Data</i> Omitting the use of encryption in any program which transfers data over a network of any kind should be considered on par with delivering the data sent to each user on the local networks of both the sender and receiver. Worse, this omission allows for the injection of data into a stream of communication between two parties -- with no means for the victims to separate valid data from invalid. In this day of widespread network attacks and password collection sniffers, it is an unnecessary risk to omit encryption from the design of any system which might benefit from it.	

▼ Likelihood Of Exploit

High

▼ Demonstrative Examples

Example 1

This code writes a user's login information to a cookie so the user does not have to login again later.

The code stores the user's username and password in plaintext in a cookie on the user's machine. This exposes the user's login information if their computer is compromised by an attacker. Even if the user's machine is not compromised, this weakness combined with cross-site scripting ([CWE-79](#)) could allow an attacker to remotely copy the cookie.

Also note this example code also exhibits Plaintext Storage in a Cookie ([CWE-315](#)).

Example 2

The following code attempts to establish a connection, read in a password, then store it to a buffer.

While successful, the program does not encrypt the data before writing it to a buffer, possibly exposing it to unauthorized actors.

Example 3

The following code attempts to establish a connection to a site to communicate sensitive information.

Though a connection is successfully made, the connection is unencrypted and it is possible that all sensitive data sent to or received from the server will be read by unintended actors.

▼ Potential Mitigations

Phase: Requirements

Clearly specify which data or resources are valuable enough that they should be protected by encryption. Require that any transmission or storage of this data/resource should use well-vetted encryption algorithms.

Phase: Architecture and Design

Ensure that encryption is properly integrated into the system design, including but not necessarily limited to:

- Encryption that is needed to store or transmit private data of the users of the system
- Encryption that is needed to protect the system itself from unauthorized disclosure or tampering

Identify the separate needs and contexts for encryption:

- One-way (i.e., only the user or recipient needs to have the key). This can be achieved using public key cryptography, or other techniques in which the encrypting party (i.e., the software) does not need to have access to a private key.
- Two-way (i.e., the encryption can be automatically performed on behalf of a user, but the key must be available so that the plaintext can be automatically recoverable by that user). This requires storage of the private key in a format that is recoverable only by the user (or perhaps by the operating system) in a way that cannot be recovered by others.

Using threat modeling or other techniques, assume that data can be compromised through a separate vulnerability or weakness, and determine where encryption will be most effective. Ensure that data that should be private is not being inadvertently exposed using weaknesses such as insecure permissions ([CWE-732](#)). [[REF-7](#)]

Phase: Architecture and Design

Strategy: Libraries or Frameworks

When there is a need to store or transmit sensitive data, use strong, up-to-date cryptographic algorithms to encrypt that data. Select a well-vetted algorithm that is currently considered to be strong by experts in the field, and use well-tested implementations. As with all cryptographic mechanisms, the source code should be available for analysis.

For example, US government systems require FIPS 140-2 certification.

Do not develop custom or private cryptographic algorithms. They will likely be exposed to attacks that are well-understood by cryptographers. Reverse engineering techniques are mature. If the algorithm can be compromised if attackers find out how it works, then it is especially weak.

Periodically ensure that the cryptography has not become obsolete. Some older algorithms, once thought to require a billion years of computing time, can now be broken in days or hours. This includes MD4, MD5, SHA1, DES, and other algorithms that were once regarded as strong. [[REF-267](#)]

Phase: Architecture and Design

Strategy: Separation of Privilege

Compartmentalize the system to have "safe" areas where trust boundaries can be unambiguously drawn. Do not allow sensitive data to go outside of the trust boundary and always be careful when interfacing with a compartment outside of the safe area.

Ensure that appropriate compartmentalization is built into the system design and that the compartmentalization serves to allow for and further reinforce privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide when it is appropriate to use and to drop system privileges.

Phases: Implementation; Architecture and Design

When using industry-approved techniques, use them correctly. Don't cut corners by skipping resource-intensive steps ([CWE-325](#)). These steps are often essential for preventing common attacks.

Phase: Implementation

Strategy: Attack Surface Reduction

Use naming conventions and strong types to make it easier to spot when sensitive data is being used. When creating structures, objects, or other complex entities, separate the sensitive and non-sensitive data as much as possible.

Effectiveness: Defense in Depth

Note: This makes it easier to spot places in the code where data is being used that is unencrypted.

Memberships

This MemberOf Relationships table shows additional CWE Categories and Views that reference this weakness as a member. This information is often useful in understanding where a weakness fits within the context of external information sources.

Nature	Type	ID	Name
MemberOf		719	OWASP Top Ten 2007 Category A8 - Insecure Cryptographic Storage
MemberOf		720	OWASP Top Ten 2007 Category A9 - Insecure Communications
MemberOf		729	OWASP Top Ten 2004 Category A8 - Insecure Storage
MemberOf		803	2010 Top 25 - Porous Defenses
MemberOf		816	OWASP Top Ten 2010 Category A7 - Insecure Cryptographic Storage
MemberOf		818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection
MemberOf		861	The CERT Oracle Secure Coding Standard for Java (2011). Chapter 18 - Miscellaneous (MSC).
MemberOf		866	2011 Top 25 - Porous Defenses
MemberOf		930	OWASP Top Ten 2013 Category A2 - Broken Authentication and Session Management
MemberOf		934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure
MemberOf		963	SFP Secondary Cluster: Exposed Data
MemberOf		1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
MemberOf		1152	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 49. Miscellaneous (MSC).

Notes

Relationship

There is an overlapping relationship between insecure storage of sensitive information ([CWE-922](#)) and missing encryption of sensitive information ([CWE-311](#)). Encryption is often used to prevent an attacker from reading the sensitive data. However, encryption does not prevent the attacker from erasing or overwriting the data.

More information is available — Please select a different filter.

Use of the Common Weakness Enumeration and the associated references from this website are subject to the [Terms of Use](#). For more information, please email cwe@mitre.org.

CWE is sponsored by [US-CERT](#) in the office of [Cybersecurity and Communications](#) at the [U.S. Department of Homeland Security](#). Copyright © 2006-2019, The MITRE Corporation. CWE, CWSS, CWRAF, and the CWE logo are trademarks of [The MITRE Corporation](#).

[Privacy Policy](#)
[Terms of Use](#)
[Site Map](#)
[Contact Us](#)