CWE-319: Cleartext Transmission of Sensitive Information

Weakness ID: 319
Vulnerability Mapping: ALLOWED
Abstraction: Base

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

Description

The product transmits sensitive or security-critical data in cleartext in a communication channel that can be sniffed by unauthorized actors.

Extended Description

Many communication channels can be "sniffed" (monitored) by adversaries during data transmission. For example, in networking, packets can traverse many intermediary nodes from the source to the destination, whether across the internet, an internal network, the cloud, etc. Some actors might have privileged access to a network interface or any link along the channel, such as a router, but they might not be authorized to collect the underlying data. As a result, network traffic could be sniffed by adversaries, spilling security-critical data.

Applicable communication channels are not limited to software products. Applicable channels include hardware-specific technologies such as internal hardware networks and external debug channels, supporting remote JTAG debugging. When mitigations are not applied to combat adversaries within the product's threat model, this weakness significantly lowers the difficulty of exploitation by such adversaries.

When full communications are recorded or logged, such as with a packet dump, an adversary could attempt to obtain the dump long after the transmission has occurred and try to "sniff" the cleartext from the recorded communications in the dump itself. Even if the information is encoded in a way that is not human-readable, certain techniques could determine which encoding is being used, then decode the information.

Common Consequences

Scope	Impact	Likelihood
Integrity Confidentiality	**Technical Impact:** *Read Application Data; Modify Files or Directories* Anyone can read the information by gaining access to the channel being used for communication.	

Potential Mitigations

Phase: Architecture and Design

Before transmitting, encrypt the data using reliable, confidentiality-protecting cryptographic protocols.

Phase: Implementation

When using web applications with SSL, use SSL for the entire session from login to logout, not just for the initial login page.

Phase: Implementation

When designing hardware platforms, ensure that approved encryption algorithms (such as those recommended by NIST) protect paths from security critical data to trusted user applications.

Phase: Testing

Use 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. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Phase: Operation

Configure servers to use encrypted channels for communication, which may include SSL or other secure protocols.

Relationships

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

Nature	Type	ID	Name
ChildOf	C	311	Missing Encryption of Sensitive Data
ParentOf	V	5	J2EE Misconfiguration: Data Transmission Without Encryption
ParentOf	V	614	Sensitive Cookie in HTTPS Session Without 'Secure' Attribute

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	199	Information Management Errors

Relevant to the view "Hardware Design" (CWE-1194)

Nature	Type	ID	Name
MemberOf	C	1207	Debug and Test Problems

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	OMISSION: This weakness is caused by missing a security tactic during the architecture and design phase.
Architecture and Design	For hardware, this may be introduced when design does not plan for an attacker having physical access while a legitimate user is remotely operating the device.
Operation	
System Configuration	

Applicable Platforms

Languages

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

Technologies

Class: Cloud Computing *(Undetermined Prevalence)*
Class: Mobile *(Undetermined Prevalence)*
Class: ICS/OT *(Often Prevalent)*
Class: System on Chip *(Undetermined Prevalence)*
Test/Debug Hardware *(Often Prevalent)*

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

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

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

```java
try {
  URL u = new URL("http://www.secret.example.org/");
  HttpURLConnection hu = (HttpURLConnection) u.openConnection();
  hu.setRequestMethod("PUT");
  hu.connect();
  OutputStream os = hu.getOutputStream();
  hu.disconnect();
}
catch (IOException e) {
  //...
}
```

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.

Example 2

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 of sensitive information in their OT products.

Example 3

A TAP accessible register is read/written by a JTAG based tool, for internal use by authorized users. However, an adversary can connect a probing device and collect the values from the unencrypted channel connecting the JTAG interface to the authorized user, if no additional protections are employed.

Example 4

The following Azure CLI command lists the properties of a particular storage account:

Example Language: **Shell** *(informative)*

```shell
az storage account show -g {ResourceGroupName} -n {StorageAccountName}
```

The JSON result might be:

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

```json
{
  "name": "{StorageAccountName}",
  "enableHttpsTrafficOnly": false,
  "type": "Microsoft.Storage/storageAccounts"
}
```

The enableHttpsTrafficOnly value is set to false, because the default setting for Secure transfer is set to Disabled. This allows cloud storage resources to successfully connect and transfer data without the use of encryption (e.g., HTTP, SMB 2.1, SMB 3.0, etc.).

Azure's storage accounts can be configured to only accept requests from secure connections made over HTTPS. The secure transfer setting can be enabled using Azure's Portal (GUI) or programmatically by setting the enableHttpsTrafficOnly property to True on the storage account, such as:

Example Language: **Shell** *(good code)*

```shell
az storage account update -g {ResourceGroupName} -n {StorageAccountName} --https-only true
```

The change can be confirmed from the result by verifying that the enableHttpsTrafficOnly value is true:

Example Language: **JSON** *(good code)*

```json
{
  "name": "{StorageAccountName}",
  "enableHttpsTrafficOnly": true,
  "type": "Microsoft.Storage/storageAccounts"
}
```

Note: to enable secure transfer using Azure's Portal instead of the command line:

1. Open the Create storage account pane in the Azure portal.
2. In the Advanced page, select the Enable secure transfer checkbox.

Observed Examples

Reference	Description
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-2002-1949	Passwords transmitted in cleartext.
CVE-2008-4122	Chain: Use of HTTPS cookie without "secure" flag causes it to be transmitted across unencrypted HTTP.
CVE-2008-3289	Product sends password hash in cleartext in violation of intended policy.
CVE-2008-4390	Remote management feature sends sensitive information including passwords in cleartext.
CVE-2007-5626	Backup routine sends password in cleartext in email.
CVE-2004-1852	Product transmits Blowfish encryption key in cleartext.
CVE-2008-0374	Printer sends configuration information, including administrative password, in cleartext.
CVE-2007-4961	Chain: cleartext transmission of the MD5 hash of password enables attacks against a server that is susceptible to replay (CWE-294).
CVE-2007-4786	Product sends passwords in cleartext to a log server.
CVE-2005-3140	Product sends file with cleartext passwords in e-mail message intended for diagnostic purposes.

Detection Methods

Black Box

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process, trigger the feature that sends the data, and look for the presence or absence of common cryptographic functions in the call tree. Monitor the network and determine if the data packets contain readable commands. Tools exist for detecting if certain encodings are in use. If the traffic contains high entropy, this might indicate the usage of encryption.

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	751	2009 Top 25 - Insecure Interaction Between Components
MemberOf	C	818	OWASP Top Ten 2010 Category A9 - Insufficient Transport Layer Protection
MemberOf	C	858	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 15 - Serialization (SER)
MemberOf	C	859	The CERT Oracle Secure Coding Standard for Java (2011) Chapter 16 - Platform Security (SEC)
MemberOf	V	884	CWE Cross-section
MemberOf	C	934	OWASP Top Ten 2013 Category A6 - Sensitive Data Exposure
MemberOf	C	963	SFP Secondary Cluster: Exposed Data
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
MemberOf	C	1148	SEI CERT Oracle Secure Coding Standard for Java - Guidelines 14. Serialization (SER)
MemberOf	C	1346	OWASP Top Ten 2021 Category A02:2021 - Cryptographic Failures
MemberOf	C	1366	ICS Communications: Frail Security in Protocols
MemberOf	C	1402	Comprehensive Categorization: Encryption

Vulnerability Mapping Notes

Usage: ALLOWED *(this CWE ID could be used to map to real-world vulnerabilities)*

Reason: Acceptable-Use

Rationale:
This CWE entry is at the Base level of abstraction, which is a preferred level of abstraction for mapping to the root causes of vulnerabilities.

Comments:
Carefully read both the name and description to ensure that this mapping is an appropriate fit. Do not try to 'force' a mapping to a lower-level Base/Variant simply to comply with this preferred level of abstraction.

Notes

Maintenance

The Taxonomy_Mappings to ISA/IEC 62443 were added in CWE 4.10, but they are still under review and might change in future CWE versions. These draft mappings were performed by members of the "Mapping CWE to 62443" subgroup of the CWE-CAPEC ICS/OT Special Interest Group (SIG), and their work is incomplete as of CWE 4.10. The mappings are included to facilitate discussion and review by the broader ICS/OT community, and they are likely to change in future CWE versions.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Plaintext Transmission of Sensitive Information
The CERT Oracle Secure Coding Standard for Java (2011)	SEC06-J		Do not rely on the default automatic signature verification provided by URLClassLoader and java.util.jar
The CERT Oracle Secure Coding Standard for Java (2011)	SER02-J		Sign then seal sensitive objects before sending them outside a trust boundary
Software Fault Patterns	SFP23		Exposed Data
ISA/IEC 62443	Part 3-3		Req SR 4.1
ISA/IEC 62443	Part 4-2		Req CR 4.1B

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-102	Session Sidejacking
CAPEC-117	Interception
CAPEC-383	Harvesting Information via API Event Monitoring
CAPEC-477	Signature Spoofing by Mixing Signed and Unsigned Content
CAPEC-65	Sniff Application Code

References

[REF-271] OWASP. "Top 10 2007-Insecure Communications". 2007. <http://www.owasp.org/index.php/Top_10_2007-A9>.

[REF-7] Michael Howard and David LeBlanc. "Writing Secure Code". Chapter 9, "Protecting Secret Data" Page 299. 2nd Edition. Microsoft Press. 2002-12-04. <https://www.microsoftpressstore.com/store/writing-secure-code-9780735617223>.

[REF-44] Michael Howard, David LeBlanc and John Viega. "24 Deadly Sins of Software Security". "Sin 22: Failing to Protect Network Traffic." Page 337. McGraw-Hill. 2010.

[REF-172] Chris Wysopal. "Mobile App Top 10 List". 2010-12-13. <https://www.veracode.com/blog/2010/12/mobile-app-top-10-list>. *URL validated: 2023-04-07.*

[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/>.

[REF-1307] Center for Internet Security. "CIS Microsoft Azure Foundations Benchmark version 1.5.0". Sections 3.1 and 3.10. 2022-08-16. <https://www.cisecurity.org/benchmark/azure>. *URL validated: 2023-01-19.*

[REF-1309] Microsoft. "Require secure transfer to ensure secure connections". 2022-07-24. <https://learn.microsoft.com/en-us/azure/storage/common/storage-require-secure-transfer>. *URL validated: 2023-01-24.*

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	PLOVER	

Contributions

Contribution Date	Contributor	Organization
2023-01-24	Accellera IP Security Assurance (IPSA) Working Group Submitted original contents of CWE-1324 and reviewed its integration into this entry.	Accellera Systems Initiative

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024