CWE-295: Improper Certificate Validation

Weakness ID: 295
Vulnerability Mapping: ALLOWED
Abstraction: Base

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

Description

The product does not validate, or incorrectly validates, a certificate.

Extended Description

When a certificate is invalid or malicious, it might allow an attacker to spoof a trusted entity by interfering in the communication path between the host and client. The product might connect to a malicious host while believing it is a trusted host, or the product might be deceived into accepting spoofed data that appears to originate from a trusted host.

Common Consequences

Scope	Impact	Likelihood
Integrity Authentication	**Technical Impact:** *Bypass Protection Mechanism; Gain Privileges or Assume Identity*	

Potential Mitigations

Phases: Architecture and Design; Implementation

Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key.

Phase: Implementation

If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.

Relationships

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

Nature	Type	ID	Name
ChildOf	B	287	Improper Authentication
ParentOf	B	296	Improper Following of a Certificate's Chain of Trust
ParentOf	V	297	Improper Validation of Certificate with Host Mismatch
ParentOf	V	298	Improper Validation of Certificate Expiration
ParentOf	B	299	Improper Check for Certificate Revocation
ParentOf	V	599	Missing Validation of OpenSSL Certificate
PeerOf	B	322	Key Exchange without Entity Authentication

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

Nature	Type	ID	Name
MemberOf	C	1211	Authentication Errors

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

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

Background Details

A certificate is a token that associates an identity (principal) to a cryptographic key. Certificates can be used to check if a public key belongs to the assumed owner.

Modes Of Introduction

Phase	Note
Architecture and Design Implementation Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic. When the product uses certificate pinning, the developer might not properly validate all relevant components of the certificate before pinning the certificate. This can make it difficult or expensive to test after the pinning is complete.

Applicable Platforms

Languages

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

Technologies

Class: Mobile *(Undetermined Prevalence)*

Demonstrative Examples

Example 1

This code checks the certificate of a connected peer.

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

```
if ((cert = SSL_get_peer_certificate(ssl)) && host)
  foo=SSL_get_verify_result(ssl);

if ((X509_V_OK==foo) || X509_V_ERR_SELF_SIGNED_CERT_IN_CHAIN==foo))

  // certificate looks good, host can be trusted
```

In this case, because the certificate is self-signed, there was no external authority that could prove the identity of the host. The program could be communicating with a different system that is spoofing the host, e.g. by poisoning the DNS cache or using an Adversary-in-the-Middle (AITM) attack to modify the traffic from server to client.

Example 2

The following OpenSSL code obtains a certificate and verifies it.

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

```
cert = SSL_get_peer_certificate(ssl);
if (cert && (SSL_get_verify_result(ssl)==X509_V_OK)) {

  // do secret things
}
```

Even though the "verify" step returns X509_V_OK, this step does not include checking the Common Name against the name of the host. That is, there is no guarantee that the certificate is for the desired host. The SSL connection could have been established with a malicious host that provided a valid certificate.

Example 3

The following OpenSSL code ensures that there is a certificate and allows the use of expired certificates.

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

```
if (cert = SSL_get_peer(certificate(ssl)) {
  foo=SSL_get_verify_result(ssl);
  if ((X509_V_OK==foo) || (X509_V_ERR_CERT_HAS_EXPIRED==foo))

    //do stuff
```

If the call to SSL_get_verify_result() returns X509_V_ERR_CERT_HAS_EXPIRED, this means that the certificate has expired. As time goes on, there is an increasing chance for attackers to compromise the certificate.

Example 4

The following OpenSSL code ensures that there is a certificate before continuing execution.

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

```
if (cert = SSL_get_peer_certificate(ssl)) {

  // got a certificate, do secret things
```

Because this code does not use SSL_get_verify_results() to check the certificate, it could accept certificates that have been revoked (X509_V_ERR_CERT_REVOKED). The software could be communicating with a malicious host.

Example 5

The following OpenSSL code ensures that the host has a certificate.

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

```
if (cert = SSL_get_peer_certificate(ssl)) {

  // got certificate, host can be trusted

  //foo=SSL_get_verify_result(ssl);

  //if (X509_V_OK==foo) ...
}
```

Note that the code does not call SSL_get_verify_result(ssl), which effectively disables the validation step that checks the certificate.

Observed Examples

Reference	Description
CVE-2019-12496	A Go framework for robotics, drones, and IoT devices skips verification of root CA certificates by default.
CVE-2014-1266	chain: incorrect "goto" in Apple SSL product bypasses certificate validation, allowing Adversary-in-the-Middle (AITM) attack (Apple "goto fail" bug). CWE-705 (Incorrect Control Flow Scoping) -> CWE-561 (Dead Code) -> CWE-295 (Improper Certificate Validation) -> CWE-393 (Return of Wrong Status Code) -> CWE-300 (Channel Accessible by Non-Endpoint).
CVE-2021-22909	Chain: router's firmware update procedure uses curl with "-k" (insecure) option that disables certificate validation (CWE-295), allowing adversary-in-the-middle (AITM) compromise with a malicious firmware image (CWE-494).
CVE-2008-4989	Verification function trusts certificate chains in which the last certificate is self-signed.
CVE-2012-5821	Web browser uses a TLS-related function incorrectly, preventing it from verifying that a server's certificate is signed by a trusted certification authority (CA)
CVE-2009-3046	Web browser does not check if any intermediate certificates are revoked.
CVE-2011-0199	Operating system does not check Certificate Revocation List (CRL) in some cases, allowing spoofing using a revoked certificate.
CVE-2012-5810	Mobile banking application does not verify hostname, leading to financial loss.
CVE-2012-3446	Cloud-support library written in Python uses incorrect regular expression when matching hostname.
CVE-2009-2408	Web browser does not correctly handle '\0' character (NUL) in Common Name, allowing spoofing of https sites.
CVE-2012-2993	Smartphone device does not verify hostname, allowing spoofing of mail services.
CVE-2012-5822	Application uses third-party library that does not validate hostname.
CVE-2012-5819	Cloud storage management application does not validate hostname.
CVE-2012-5817	Java library uses JSSE SSLSocket and SSLEngine classes, which do not verify the hostname.
CVE-2010-1378	chain: incorrect calculation allows attackers to bypass certificate checks.
CVE-2005-3170	LDAP client accepts certificates even if they are not from a trusted CA.
CVE-2009-0265	chain: DNS server does not correctly check return value from the OpenSSL EVP_VerifyFinal function allows bypass of validation of the certificate chain.
CVE-2003-1229	chain: product checks if client is trusted when it intended to check if the server is trusted, allowing validation of signed code.
CVE-2002-0862	Cryptographic API, as used in web browsers, mail clients, and other software, does not properly validate Basic Constraints.
CVE-2009-1358	chain: OS package manager does not check properly check the return value, allowing bypass using a revoked certificate.

Detection Methods

Automated Static Analysis - Binary or Bytecode

According to SOAR, the following detection techniques may be useful:

Cost effective for partial coverage:

- Bytecode Weakness Analysis - including disassembler + source code weakness analysis
- Binary Weakness Analysis - including disassembler + source code weakness analysis

Effectiveness: SOAR Partial

Manual Static Analysis - Binary or Bytecode

According to SOAR, 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, the following detection techniques may be useful:

Cost effective for partial coverage:

- Web Application Scanner

Effectiveness: SOAR Partial

Dynamic Analysis with Manual Results Interpretation

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Man-in-the-middle attack tool

Effectiveness: High

Manual Static Analysis - Source Code

According to SOAR, the following detection techniques may be useful:

Highly cost effective:

- Focused Manual Spotcheck - Focused manual analysis of source
- Manual Source Code Review (not inspections)

Effectiveness: High

Automated Static Analysis - Source Code

According to SOAR, 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, the following detection techniques may be useful:

Highly cost effective:

- Inspection (IEEE 1028 standard) (can apply to requirements, design, source code, etc.)

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management
MemberOf	C	1029	OWASP Top Ten 2017 Category A3 - Sensitive Data Exposure
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors
MemberOf	C	1353	OWASP Top Ten 2021 Category A07:2021 - Identification and Authentication Failures
MemberOf	C	1382	ICS Operations (& Maintenance): Emerging Energy Technologies
MemberOf	C	1396	Comprehensive Categorization: Access Control

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.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-459	Creating a Rogue Certification Authority Certificate
CAPEC-475	Signature Spoofing by Improper Validation

References

[REF-243] Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith and Lars Baumgärtner, Bernd Freisleben. "Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security". 2012-10-16. <http://www2.dcsec.uni-hannover.de/files/android/p50-fahl.pdf>.

[REF-244] M. Bishop. "Computer Security: Art and Science". Addison-Wesley. 2003.

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	CWE Community Submitted by members of the CWE community to extend early CWE versions	

Modifications

Previous Entry Names

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.