CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Weakness ID: 200
Vulnerability Mapping: DISCOURAGED
Abstraction: Class

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

Description

The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.

Extended Description

There are many different kinds of mistakes that introduce information exposures. The severity of the error can range widely, depending on the context in which the product operates, the type of sensitive information that is revealed, and the benefits it may provide to an attacker. Some kinds of sensitive information include:

- private, personal information, such as personal messages, financial data, health records, geographic location, or contact details
- system status and environment, such as the operating system and installed packages
- business secrets and intellectual property
- network status and configuration
- the product's own code or internal state
- metadata, e.g. logging of connections or message headers
- indirect information, such as a discrepancy between two internal operations that can be observed by an outsider

Information might be sensitive to different parties, each of which may have their own expectations for whether the information should be protected. These parties include:

- the product's own users
- people or organizations whose information is created or used by the product, even if they are not direct product users
- the product's administrators, including the admins of the system(s) and/or networks on which the product operates
- the developer

Information exposures can occur in different ways:

- the code **explicitly inserts** sensitive information into resources or messages that are intentionally made accessible to unauthorized actors, but should not contain the information - i.e., the information should have been "scrubbed" or "sanitized"
- a different weakness or mistake **indirectly inserts** the sensitive information into resources, such as a web script error revealing the full system path of the program.
- the code manages resources that intentionally contain sensitive information, but the resources are **unintentionally made accessible** to unauthorized actors. In this case, the information exposure is resultant - i.e., a different weakness enabled the access to the information in the first place.

It is common practice to describe any loss of confidentiality as an "information exposure," but this can lead to overuse of CWE-200 in CWE mapping. From the CWE perspective, loss of confidentiality is a technical impact that can arise from dozens of different weaknesses, such as insecure file permissions or out-of-bounds read. CWE-200 and its lower-level descendants are intended to cover the mistakes that occur in behaviors that explicitly manage, store, transfer, or cleanse sensitive information.

Alternate Terms

Information Disclosure: This term is frequently used in vulnerability advisories to describe a consequence or technical impact, for any vulnerability that has a loss of confidentiality. Often, CWE-200 can be misused to represent the loss of confidentiality, even when the mistake - i.e., the weakness - is not directly related to the mishandling of the information itself, such as an out-of-bounds read that accesses sensitive memory contents; here, the out-of-bounds read is the primary weakness, not the disclosure of the memory. In addition, this phrase is also used frequently in policies and legal documents, but it does not refer to any disclosure of security-relevant information.

Information Leak: This is a frequently used term, however the "leak" term has multiple uses within security. In some cases it deals with the accidental exposure of information from a different weakness, but in other cases (such as "memory leak"), this deals with improper tracking of resources, which can lead to exhaustion. As a result, CWE is actively avoiding usage of the "leak" term.

Common Consequences

Scope	Impact	Likelihood
Confidentiality	**Technical Impact:** *Read Application Data*	

Potential Mitigations

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 the compartmentalization allows for and reinforces privilege separation functionality. Architects and designers should rely on the principle of least privilege to decide the appropriate time to use privileges and the time to drop privileges.

Relationships

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

Nature	Type	ID	Name
ChildOf	C	668	Exposure of Resource to Wrong Sphere
ParentOf	B	201	Insertion of Sensitive Information Into Sent Data
ParentOf	B	203	Observable Discrepancy
ParentOf	B	209	Generation of Error Message Containing Sensitive Information
ParentOf	B	213	Exposure of Sensitive Information Due to Incompatible Policies
ParentOf	B	215	Insertion of Sensitive Information Into Debugging Code
ParentOf	B	359	Exposure of Private Personal Information to an Unauthorized Actor
ParentOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
ParentOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory
ParentOf	B	1258	Exposure of Sensitive System Information Due to Uncleared Debug Information
ParentOf	B	1273	Device Unlock Credential Sharing
ParentOf	B	1295	Debug Messages Revealing Unnecessary Information
CanFollow	V	498	Cloneable Class Containing Sensitive Information
CanFollow	V	499	Serializable Class Containing Sensitive Data
CanFollow	B	1272	Sensitive Information Uncleared Before Debug/Power State Transition

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

Modes Of Introduction

Phase	Note
Architecture and Design	
Implementation	

Applicable Platforms

Languages

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

Technologies

Class: Mobile *(Undetermined Prevalence)*

Likelihood Of Exploit

High

Demonstrative Examples

Example 1

The following code checks validity of the supplied username and password and notifies the user of a successful or failed login.

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

```perl
my $username=param('username');
my $password=param('password');

if (IsValidUsername($username) == 1)
{
  if (IsValidPassword($username, $password) == 1)
  {
    print "Login Successful";
  }
  else
  {
    print "Login Failed - incorrect password";
  }
}
else
{
  print "Login Failed - unknown username";
}
```

In the above code, there are different messages for when an incorrect username is supplied, versus when the username is correct but the password is wrong. This difference enables a potential attacker to understand the state of the login function, and could allow an attacker to discover a valid username by trying different values until the incorrect password message is returned. In essence, this makes it easier for an attacker to obtain half of the necessary authentication credentials.

While this type of information may be helpful to a user, it is also useful to a potential attacker. In the above example, the message for both failed cases should be the same, such as:

(result)

```
"Login Failed - incorrect username or password"
```

Example 2

This code tries to open a database connection, and prints any exceptions that occur.

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

```php
try {
  openDbConnection();
}
//print exception message that includes exception message and configuration file location
catch (Exception $e) {
  echo 'Caught exception: ', $e->getMessage(), '\n';
  echo 'Check credentials in config file at: ', $Mysql_config_location, '\n';
}
```

If an exception occurs, the printed message exposes the location of the configuration file the script is using. An attacker can use this information to target the configuration file (perhaps exploiting a Path Traversal weakness). If the file can be read, the attacker could gain credentials for accessing the database. The attacker may also be able to replace the file with a malicious one, causing the application to use an arbitrary database.

Example 3

In the example below, the method getUserBankAccount retrieves a bank account object from a database using the supplied username and account number to query the database. If an SQLException is raised when querying the database, an error message is created and output to a log file.

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

```java
public BankAccount getUserBankAccount(String username, String accountNumber) {
  BankAccount userAccount = null;
  String query = null;
  try {
    if (isAuthorizedUser(username)) {
      query = "SELECT * FROM accounts WHERE owner = "
      + username + " AND accountID = " + accountNumber;
      DatabaseManager dbManager = new DatabaseManager();
      Connection conn = dbManager.getConnection();
      Statement stmt = conn.createStatement();
      ResultSet queryResult = stmt.executeQuery(query);
      userAccount = (BankAccount)queryResult.getObject(accountNumber);
    }
  } catch (SQLException ex) {
    String logMessage = "Unable to retrieve account information from database,\nquery: " + query;
    Logger.getLogger(BankManager.class.getName()).log(Level.SEVERE, logMessage, ex);
  }
  return userAccount;
}
```

The error message that is created includes information about the database query that may contain sensitive information about the database or query logic. In this case, the error message will expose the table name and column names used in the database. This data could be used to simplify other attacks, such as SQL injection (CWE-89) to directly access the database.

Example 4

This code stores location information about the current user:

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

```java
locationClient = new LocationClient(this, this, this);
locationClient.connect();
currentUser.setLocation(locationClient.getLastLocation());
...

catch (Exception e) {
  AlertDialog.Builder builder = new AlertDialog.Builder(this);
  builder.setMessage("Sorry, this application has experienced an error.");
  AlertDialog alert = builder.create();
  alert.show();
  Log.e("ExampleActivity", "Caught exception: " + e + " While on User:" + User.toString());
}
```

When the application encounters an exception it will write the user object to the log. Because the user object contains location information, the user's location is also written to the log.

Example 5

The following is an actual MySQL error statement:

Example Language: **SQL** *(result)*

```
Warning: mysql_pconnect(): Access denied for user: 'root@localhost' (Using password: N1nj4) in /usr/local/www/wi-data/includes/database.inc on line 4
```

The error clearly exposes the database credentials.

Example 6

This code displays some information on a web page.

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

```jsp
Social Security Number: <%= ssn %></br>Credit Card Number: <%= ccn %>
```

The code displays a user's credit card and social security numbers, even though they aren't absolutely necessary.

Example 7

The following program changes its behavior based on a debug flag.

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

```jsp
<% if (Boolean.getBoolean("debugEnabled")) {
  %>
  User account number: <%= acctNo %>
  <%
  } %>
```

The code writes sensitive debug information to the client browser if the "debugEnabled" flag is set to true .

Example 8

This code uses location to determine the user's current US State location.

First the application must declare that it requires the ACCESS_FINE_LOCATION permission in the application's manifest.xml:

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

```xml
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

During execution, a call to getLastLocation() will return a location based on the application's location permissions. In this case the application has permission for the most accurate location possible:

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

```java
locationClient = new LocationClient(this, this, this);
locationClient.connect();
Location userCurrLocation;
userCurrLocation = locationClient.getLastLocation();
deriveStateFromCoords(userCurrLocation);
```

While the application needs this information, it does not need to use the ACCESS_FINE_LOCATION permission, as the ACCESS_COARSE_LOCATION permission will be sufficient to identify which US state the user is in.

Observed Examples

Reference	Description
CVE-2022-31162	Rust library leaks Oauth client details in application debug logs
CVE-2021-25476	Digital Rights Management (DRM) capability for mobile platform leaks pointer information, simplifying ASLR bypass
CVE-2001-1483	Enumeration of valid usernames based on inconsistent responses
CVE-2001-1528	Account number enumeration via inconsistent responses.
CVE-2004-2150	User enumeration via discrepancies in error messages.
CVE-2005-1205	Telnet protocol allows servers to obtain sensitive environment information from clients.
CVE-2002-1725	Script calls phpinfo(), revealing system configuration to web user
CVE-2002-0515	Product sets a different TTL when a port is being filtered than when it is not being filtered, which allows remote attackers to identify filtered ports by comparing TTLs.
CVE-2004-0778	Version control system allows remote attackers to determine the existence of arbitrary files and directories via the -X command for an alternate history file, which causes different error messages to be returned.
CVE-2000-1117	Virtual machine allows malicious web site operators to determine the existence of files on the client by measuring delays in the execution of the getSystemResource method.
CVE-2003-0190	Product immediately sends an error message when a user does not exist, which allows remote attackers to determine valid usernames via a timing attack.
CVE-2008-2049	POP3 server reveals a password in an error message after multiple APOP commands are sent. Might be resultant from another weakness.
CVE-2007-5172	Program reveals password in error message if attacker can trigger certain database errors.
CVE-2008-4638	Composite: application running with high privileges (CWE-250) allows user to specify a restricted file to process, which generates a parsing error that leaks the contents of the file (CWE-209).
CVE-2007-1409	Direct request to library file in web application triggers pathname leak in error message.
CVE-2005-0603	Malformed regexp syntax leads to information exposure in error message.
CVE-2004-2268	Password exposed in debug information.
CVE-2003-1078	FTP client with debug option enabled shows password to the screen.
CVE-2022-0708	Collaboration platform does not clear team emails in a response, allowing leak of email addresses

Weakness Ordinalities

Ordinality	Description
Primary	*(where the weakness is a quality issue that might indirectly make it easier to introduce security-relevant weaknesses or make them more difficult to detect)* Developers may insert sensitive information that they do not believe, or they might forget to remove the sensitive information after it has been processed
Resultant	*(where the weakness is a quality issue that might indirectly make it easier to introduce security-relevant weaknesses or make them more difficult to detect)* Separate mistakes or weaknesses could inadvertently make the sensitive information available to an attacker, such as in a detailed error message that can be read by an unauthorized party

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
- Inter-application Flow Analysis

Effectiveness: SOAR Partial

Dynamic Analysis with Automated Results Interpretation

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

Highly cost effective:

- Web Application Scanner
- Web Services Scanner
- Database Scanners

Effectiveness: High

Dynamic Analysis with Manual Results Interpretation

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

Cost effective for partial coverage:

- Fuzz Tester
- Framework-based Fuzzer
- Automated Monitored Execution
- Monitored Virtual Environment - run potentially malicious code in sandbox / wrapper / virtual machine, see if it does anything suspicious

Effectiveness: SOAR Partial

Manual Static Analysis - Source Code

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

Highly cost effective:

- Manual Source Code Review (not inspections)

Effectiveness: High

Automated Static Analysis - Source Code

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

Highly cost effective:

- Context-configured Source Code Weakness Analyzer

Cost effective for partial coverage:

- Source code Weakness Analyzer

Effectiveness: High

Architecture or Design Review

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

Highly cost effective:

- Formal Methods / Correct-By-Construction

Cost effective for partial coverage:

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

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	V	635	Weaknesses Originally Used by NVD from 2008 to 2016
MemberOf	C	717	OWASP Top Ten 2007 Category A6 - Information Leakage and Improper Error Handling
MemberOf	C	963	SFP Secondary Cluster: Exposed Data
MemberOf	V	1200	Weaknesses in the 2019 CWE Top 25 Most Dangerous Software Errors
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	C	1417	Comprehensive Categorization: Sensitive Information Exposure

Vulnerability Mapping Notes

Usage: DISCOURAGED *(this CWE ID should not be used to map to real-world vulnerabilities)*

Reason: Frequent Misuse

Rationale:

CWE-200 is commonly misused to represent the loss of confidentiality in a vulnerability, but confidentiality loss is a technical impact - not a root cause error. As of CWE 4.9, over 400 CWE entries can lead to a loss of confidentiality. Other options are often available. [REF-1287].

Comments:

If an error or mistake causes information to be disclosed, then use the CWE ID for that error. Consider starting with improper authorization (CWE-285), insecure permissions (CWE-732), improper authentication (CWE-287), etc. Also consider children such as Insertion of Sensitive Information Into Sent Data (CWE-201), Observable Discrepancy (CWE-203), Insertion of Sensitive Information into Externally-Accessible File or Directory (CWE-538), or others.

Notes

Maintenance

As a result of mapping analysis in the 2020 Top 25 and more recent versions, this weakness is under review, since it is frequently misused in mapping to cover many problems that lead to loss of confidentiality. See Mapping Notes, Extended Description, and Alternate Terms.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Information Leak (information disclosure)
OWASP Top Ten 2007	A6	CWE More Specific	Information Leakage and Improper Error Handling
WASC	13		Information Leakage

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-116	Excavation
CAPEC-13	Subverting Environment Variable Values
CAPEC-169	Footprinting
CAPEC-22	Exploiting Trust in Client
CAPEC-224	Fingerprinting
CAPEC-285	ICMP Echo Request Ping
CAPEC-287	TCP SYN Scan
CAPEC-290	Enumerate Mail Exchange (MX) Records
CAPEC-291	DNS Zone Transfers
CAPEC-292	Host Discovery
CAPEC-293	Traceroute Route Enumeration
CAPEC-294	ICMP Address Mask Request
CAPEC-295	Timestamp Request
CAPEC-296	ICMP Information Request
CAPEC-297	TCP ACK Ping
CAPEC-298	UDP Ping
CAPEC-299	TCP SYN Ping
CAPEC-300	Port Scanning
CAPEC-301	TCP Connect Scan
CAPEC-302	TCP FIN Scan
CAPEC-303	TCP Xmas Scan
CAPEC-304	TCP Null Scan
CAPEC-305	TCP ACK Scan
CAPEC-306	TCP Window Scan
CAPEC-307	TCP RPC Scan
CAPEC-308	UDP Scan
CAPEC-309	Network Topology Mapping
CAPEC-310	Scanning for Vulnerable Software
CAPEC-312	Active OS Fingerprinting
CAPEC-313	Passive OS Fingerprinting
CAPEC-317	IP ID Sequencing Probe
CAPEC-318	IP 'ID' Echoed Byte-Order Probe
CAPEC-319	IP (DF) 'Don't Fragment Bit' Echoing Probe
CAPEC-320	TCP Timestamp Probe
CAPEC-321	TCP Sequence Number Probe
CAPEC-322	TCP (ISN) Greatest Common Divisor Probe
CAPEC-323	TCP (ISN) Counter Rate Probe
CAPEC-324	TCP (ISN) Sequence Predictability Probe
CAPEC-325	TCP Congestion Control Flag (ECN) Probe
CAPEC-326	TCP Initial Window Size Probe
CAPEC-327	TCP Options Probe
CAPEC-328	TCP 'RST' Flag Checksum Probe
CAPEC-329	ICMP Error Message Quoting Probe
CAPEC-330	ICMP Error Message Echoing Integrity Probe
CAPEC-472	Browser Fingerprinting
CAPEC-497	File Discovery
CAPEC-508	Shoulder Surfing
CAPEC-573	Process Footprinting
CAPEC-574	Services Footprinting
CAPEC-575	Account Footprinting
CAPEC-576	Group Permission Footprinting
CAPEC-577	Owner Footprinting
CAPEC-59	Session Credential Falsification through Prediction
CAPEC-60	Reusing Session IDs (aka Session Replay)
CAPEC-616	Establish Rogue Location
CAPEC-643	Identify Shared Files/Directories on System
CAPEC-646	Peripheral Footprinting
CAPEC-651	Eavesdropping
CAPEC-79	Using Slashes in Alternate Encoding

References

[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-1287] MITRE. "Supplemental Details - 2022 CWE Top 25". Details of Problematic Mappings. 2022-06-28. <https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25_supplemental.html#problematicMappingDetails>.

Content History

Submissions

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

Contributions

Contribution Date	Contributor	Organization
2022-07-11	Nick Johnston. Identified incorrect language tag in demonstrative example.	

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024