What is Injection

This page describes in-depth what an injection attack is.

Contents:

Injection

> **OWASP Definition of Injection**
>
> OWASP defines Injection flaws as follows:
>
> A1:2017 – **Injection**
>
> Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Injection is a broad concept that covers many different security risks.

What's common to all of them is that **interpreters running in an app's background can be intentionally cheated to run code that can be exploited by an attacker**.

Depending on the underlying interpreter, injection flaws can occur on SQL-engines, LDAP engines, OS command-interpreters, XML interpreters, etc.

Tainted Data Flow

A common injection root cause is to send **untrusted** (potentially tainted) data to an interpreter as part of a command or query.

Source locations are those places in the code from where data comes in, that can be potentially controlled by the user (or the environment) and must consequently be presumably considered as **tainted** (as they may be used to build injection attacks).

- User input should always be considered as **untrusted** (you will have no way to know if a user is an attacker or a normal app user).

Sink locations are those places where consumed data must be **untainted**.

- Data used by an interpreter (a sink) must always be **trusted** (must not be controlled by a threat), i.e. sensitive data sinks rely on trusted (**untainted**) data

Your app contains an **injection point** (an injection vulnerability) wherever a data flow exists where any sink consumes input data which is not being properly neutralized. Kiuwan scans your source code to find any injection point:

> For all possible sinks, prove that tainted data will never be used where untainted data is expected.

Injection Prevention Rules

As in any other security matter, there is not a unique protection mechanism and the best approach is combining more than one.

To prevent Injection flaws we can consider the following complementary approaches:

1. The first line of defense consists of **using the interpreter through a safe API** (i.e an API that avoids the use of the interpreter entirely or provides a parameterized interface).
    - Even so, be careful of APIs, such as stored procedures that are parameterized, but can still introduce injection under the hood.
2. If you cannot use a safe API (or even using it), you should **always perform adequate input validation**.
    - By input validation, we mean accepting only what is known to be good (whitelist), rejecting what is known to be bad (backlist) and escaping special characters using the specific escape syntax for the interpreter.

This is a general approach to prevent injection flaws. But, depending on the interpreter, there can be further prevention possibilities.

When Kiuwan scans your source code, all the vulnerabilities of the same type are grouped under a Kiuwan rule, indicating how many files are affected (and where), how many vulnerabilities were found, and providing specific remediation clues based on the specifics of the programming language or interpreter being used.

What follows is a description of the main types of injection attacks, providing references to further detailed documentation and the detection coverage provided by Kiuwan.

Injection types

Kiuwan provides out-of-the-box rules to detect Injection points in the following programming languages:

- Abap, ASP.NET, C/C++, C#, Cobol, Java, JavaScript, JSP, Objective-C, Oracle Forms, PHP, PL/SQL, Python, RPG IV, Swift and Transact-SQL

This list is continuously growing, if you miss any programming language, please contact Kiuwan Support.

It follows an explanation of the **most common injection types** (referred by their CWE ID#). You can visit https://cwe.mitre.org/ for further information on every injection type.

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command (**'SQL Injection**')
- CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('**LDAP Injection'**)
- CWE-91: **XML Injection** (aka Blind XPath Injection)
- CWE-643: Improper Neutralization of Data within XPath Expressions (**'XPath Injection'**)
- CWE-611: Improper Restriction of XML External Entity Reference (**'XXE'**)
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement (**'Expression Language Injection'**)
- CWE-78: Improper Neutralization of Special Elements used in an OS Command ('**OS Command Injection'**)

Nevertheless, although we will concentrate on this subset, **there are many other injection attacks also covered by Kiuwan.**

Below you can find a list of injection attacks not covered in this paper but controlled by Kiuwan.

- CWE-15: External Control of System or Configuration Setting
- CWE-20: Improper Input Validation
- CWE-88: Argument Injection or Modification
- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-564: SQL Injection: Hibernate
- CWE-345: Insufficient Verification of Data Authenticity
- CWE-93: Improper Neutralization of CRLF Sequences ('CRLF Injection')
- CWE-94: Improper Control of Generation of Code ('Code Injection')
- CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')
- CWE-114: Process Control
- CWE-917: Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')
- CWE-99: Improper Control of Resource Identifiers ('Resource Injection')
- CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')
- CWE-494: Download of Code Without Integrity Check
- CWE-117: Improper Output

- Neutralization for Logs
- CWE-134: Use of Externally-Controlled Format String
- CWE-159: Failure to Sanitize Special Element
- CWE-180: Incorrect Behavior Order: Validate Before Canonicalize
- CWE-183: Permissive Whitelist
- CWE-185: Incorrect Regular Expression
- CWE-235: Improper Handling of Extra Parameters
- CWE-346: Origin Validation Error
- CWE-352: Cross-Site Request Forgery (CSRF)
- CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')
- CWE-472: External Control of Assumed-Immutable Web Parameter
- CWE-473: PHP External Variable Modification
- CWE-501: Trust Boundary Violation
- CWE-502: Deserialization of Untrusted Data
- CWE-601: URL Redirection to Untrusted Site ('Open Redirect')
- CWE-776: Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')
- CWE-652: Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')
- CWE-915: Improperly Controlled Modification of Dynamically-Determined Object Attributes
- CWE-918: Server-Side Request Forgery (SSRF)