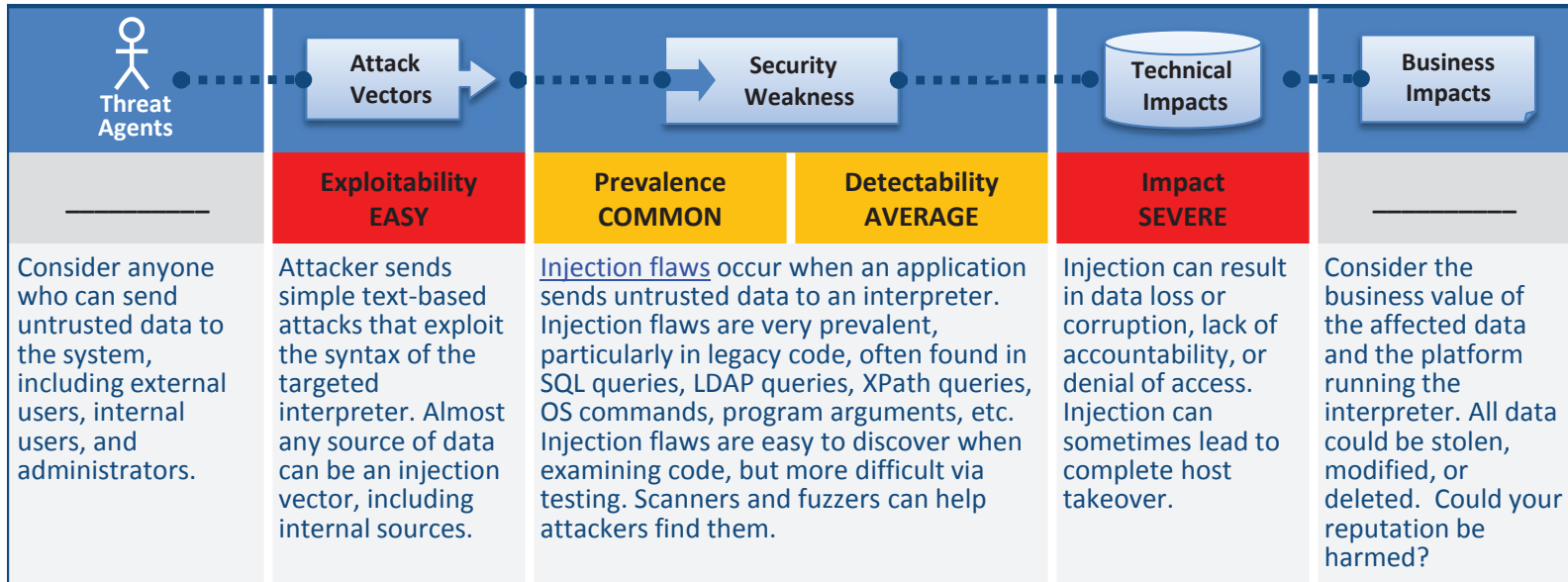


A1

Injection



Am I Vulnerable To Injection?

The best way to find out if an application is vulnerable to injection is to verify that all use of interpreters clearly separates untrusted data from the command or query. For SQL calls, this means using bind variables in all prepared statements and stored procedures, and avoiding dynamic queries.

Checking the code is a fast and accurate way to see if the application uses interpreters safely. Code analysis tools can help a security analyst find the use of interpreters and trace the data flow through the application. Penetration testers can validate these issues by crafting exploits that confirm the vulnerability.

Automated dynamic scanning which exercises the application may provide insight into whether some exploitable injection flaws exist. Scanners cannot always reach interpreters and have difficulty detecting whether an attack was successful. Poor error handling makes injection flaws easier to discover.

How Do I Prevent Injection?

Preventing injection requires keeping untrusted data separate from commands and queries.

1. The preferred option is to use a safe API which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful of APIs, such as stored procedures, that are parameterized, but can still introduce injection under the hood.
2. If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. [OWASP's ESAPI](#) has some of these [escaping routines](#).
3. Positive or "white list" input validation with appropriate canonicalization is also recommended, but is not a complete defense as many applications require special characters in their input. [OWASP's ESAPI](#) has an extensible library of [white list input validation routines](#).

Example Attack Scenario

The application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

The attacker modifies the 'id' parameter in their browser to send: ' or '1'='1. This changes the meaning of the query to return all the records from the accounts database, instead of only the intended customer's.

```
http://example.com/app/accountView?id=' or '1'='1
```

In the worst case, the attacker uses this weakness to invoke special stored procedures in the database that enable a complete takeover of the database and possibly even the server hosting the database.

References

OWASP

- [OWASP SQL Injection Prevention Cheat Sheet](#)
- [OWASP Injection Flaws Article](#)
- [ESAPI Encoder API](#)
- [ESAPI Input Validation API](#)
- [ASVS: Output Encoding/Escaping Requirements \(V6\)](#)
- [OWASP Testing Guide: Chapter on SQL Injection Testing](#)
- [OWASP Code Review Guide: Chapter on SQL Injection](#)
- [OWASP Code Review Guide: Command Injection](#)

External

- [CWE Entry 77 on Command Injection](#)
- [CWE Entry 89 on SQL Injection](#)