Injection Flaws

Description

Injection flaws allow attackers to relay malicious code through an application to another system. These attacks include calls to the operating system via system calls, the use of external programs via shell commands, as well as calls to backend databases via SQL (i.e., SQL injection). Whole scripts written in Perl, Python, and other languages can be injected into poorly designed applications and executed. Any time an application uses an interpreter of any type there is a danger of introducing an injection vulnerability.

Many web applications use operating system features and external programs to perform their functions. Sendmail is probably the most frequently invoked external program, but many other programs are used as well. When a web application passes information from an HTTP request through as part of an external request, it must be carefully scrubbed. Otherwise, the attacker can inject special (meta) characters, malicious commands, or command modifiers into the information and the web application will blindly pass these on to the external system for execution.

SQL injection is a particularly widespread and dangerous form of injection. To exploit a SQL injection flaw, the attacker must find a parameter that the web application passes through to a database. By carefully embedding malicious SQL commands into the content of the parameter, the attacker can trick the web application into forwarding a malicious query to the database. These attacks are not difficult to attempt and more tools are emerging that scan for these flaws. The consequences are particularly damaging, as an attacker can obtain, corrupt, or destroy database contents.

Injection vulnerabilities can be very easy to discover and exploit, but they can also be extremely obscure. The consequences of a successful injection attack can also run the entire range of severity, from trivial to complete system compromise or destruction. In any case, the use of external calls is quite widespread, so the likelihood of an application having an injection flaw should be considered high.

Environments Affected

Every web application environment allows the execution of external commands such as system calls, shell commands, and SQL requests. The susceptibility of an external call to command injection depends on how the call is made and the specific component that is being called, but almost all external calls can be attacked if the web application is not properly coded.

Examples

1. A malicious parameter could modify the actions taken by a system call that normally retrieves the current user's file to access another user's file (e.g., by including path traversal `../` characters as part of a filename request). Additional commands could be tacked on to the end of a parameter that is passed to a shell script to execute an additional shell command (e.g., `; rm -r \*`) along with the intended command.
2. SQL queries could be modified by adding additional 'constraints' to a where clause (e.g., `OR 1=1`) to gain access to or modify unauthorized data.

How to Determine If You Are Vulnerable

The best way to determine if your applications are vulnerable to injection attacks is to search the source code for all calls to external resources (e.g., system, exec, fork, Runtime.exec, SQL queries, or whatever the syntax is for making requests to interpreters in your environment). Note that many languages have multiple ways to run external commands. Developers should review their code and search for all places where input from an HTTP request could possibly make its way into any of these calls. You should carefully examine each of these calls to be sure that the protection steps outlined below are followed.

How to Protect Yourself

The simplest way to protect against injection is to avoid accessing external interpreters wherever possible. For many shell commands and some system calls, there are language specific libraries that perform the same functions. Using such libraries does not involve the operating system shell interpreter, and therefore avoids a large number of problems with shell commands.

For those calls that you must still employ, such as calls to backend databases, you must carefully validate the data provided to ensure that it does not contain any malicious content. You can also structure many requests in a manner that ensures that all supplied parameters are treated as data, rather than potentially executable content. The use of stored procedures or prepared statements will provide significant protection, ensuring that supplied input is treated as data. These measures will reduce, but not completely eliminate the risk involved in these external calls. You still must always validate such input to make sure it meets the expectations of the application in question. For more details on how to specifically defend against SQL Injection, please refer to OWASP's SQL Injection Prevention Cheat Sheet.

Another strong protection against injection attacks is to ensure that the web application runs with only the privileges it absolutely needs to perform its function. So you should not run the webserver as root or access a database as DBADMIN, otherwise an attacker can abuse these administrative privileges granted to the web application. Some of the J2EE environments allow the use of the Java sandbox, which can prevent the execution of system commands.

If an external command must be used, any user information that is being inserted into the command should be rigorously checked. Mechanisms should be put in place to handle any possible errors, timeouts, or blockages during the call. All output, return codes and error codes from the call should be checked to ensure that the expected processing actually occurred. At a minimum, this will allow you to determine that something has gone wrong. Otherwise, the attack may occur and never be detected.

The OWASP Filters project is producing reusable components in several languages to help prevent many forms of injection.

References

- OWASP SQL Injection Prevention Cheat Sheet
- NOSQL-injection