



WHITE PAPER

Common web application security problems and how to identify them

Part 6 Injection flaws

By

Lee J Lawson
Lead Penetration Tester, **dns**.

Introduction

This is **part 6** of a series of papers designed to raise the level of security knowledge regarding common security flaws in web applications. All papers revolve around the top 10 list of web application security issues as defined by the Open Web Application Security Project (OWASP). The last issue described the vulnerability known as **Buffer overflows**. This paper concentrates on **Injection flaws**, the identification and resolution of this flaw.

Here is a list of the most common security flaws at the time of writing taken directly from the OWASP site.

1. **Unvalidated input**
2. **Broken access control**
3. **Broken authentication and session management**
4. **Cross site scripting**
5. **Buffer overflow**
6. **Injection flaws**
7. **Improper error handling**
8. **Insecure storage**
9. **Application denial of service**
10. **Insecure configuration management**

It should be noted that this paper is not intended to be used as a replacement to professional penetration testing. Rather it is aimed toward raising the level of security knowledge in the community. Professional penetration testing requires years of experience to be able to apply worthwhile interpretation to results from testing tools and gain the skills required to perform manual assurance testing.

Injection flaws

Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.

The Problem

Part 1 of this series titled “**Unvalidated Input**” described a security flaw where input to an application is not verified that it is safe before execution. In that paper, SQL injection was used as a means to explain the problem, yet SQL injection is really an attack against an injection flaw in an application. Cross-Site Scripting (XSS) and buffer overflow attacks can also be attributed to injection flaws.

an injection flaw in an application will allow an attacker to prefix, embed or append commands onto a value being passed to an application

An injection flaw in an application will allow an attacker to prefix, embed or append commands onto a value being passed to an application. The primary problem with this is that the injected command will be executed under the security context of the application rather than the user. This may allow an attacker to execute commands as a privileged account, depending on how the application is installed.

A common error among developers is the use of privileged accounts in which to run their programs. This makes the complicated handling of permissions moot as the application has ‘full power’ over the system. This, though being a critical oversight by the developer, would be fine if the application was 100% bullet proof. Unfortunately this is very difficult to achieve. As applications get more and more functionality ‘plugged’ in to them, the likelihood of introducing a vulnerability increases.

Identification

Identification of an injection flaw will depend almost entirely on what program or service that the application is communicating with. For example, a web application that passes a date of birth to the Unix Cal program so that the day of the week a user is born is displayed, will respond differently to an application that communicates with a database server.

there are some common characters that can be entered into an application that may generate an unexpected response

There are some common characters that can be entered into an application that may generate an unexpected response. The aim of this exercise is to generate a server error message. When a server error message is displayed back to the browser, the attacker knows that whatever they typed into the web page has broken the system and caused it to do something it was not designed for.

Characters mean something to certain applications. The ‘pipe’ character | means that the output of one tool can be used as the input to another. The ‘greater than’ sign means that the output of one tool can be redirected somewhere else, such as into a text file. The semi-colon ; generally means that the preceding command is complete and it should execute.

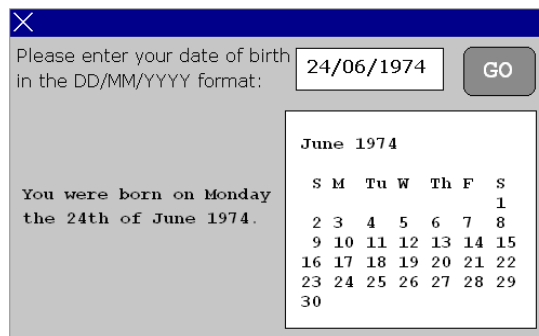
an attacker is trying to get the application to do something unexpected

When an application passes some kind of input data, whether that is direct input within a forms or a box, or indirect input via a drop down box or button, that data can be manipulated and these characters can be injected into the data stream. It can be as easy as entering a single quote into a username field of a web application or adding a semi-colon after another input.

Essentially, an attacker is trying to get the application to do something unexpected. When the correct character sequence is found, the attacker will attempt to get the application to do something other than it was designed for.

Again, an example will work best for explaining this vulnerability.

Consider a program which passes a date of birth to the 'cal' program for *nix systems.



The entered value "24/06/1974" is passed as a variable and then the result is calculated.

What would happen if an attacker entered the following two examples into the input box?

24/06/1974;shutdown -h now

Or...

24/06/1974&&rm -rf /*.*

The semi-colon and double ampersand characters instruct the system to execute both commands. By appending an extra UNIX command onto the end of the input, the attacker is able to pass those commands to the operating systems shell interpreter. These appended commands are then executed by the shell (command prompt within a Windows system) under the security context of the application that passed them and not as the logged on user.

This presents obvious security problems, especially if the application has been installed and runs as root.

To identify if any of your applications are vulnerable to an injection flaw, attempt to input the list of characters below and record the result.

|/\<>&& ();'+

If the application returns some form of error, or shows in some way that the process failed, then you may be vulnerable and further analysis is required.

one of the most effective methods of determining the security stature of an application is to analyse the source code

If the application handles the inputted characters in a safe way and returns some message stating that the input is invalid, then you are probably not vulnerable. That being said, there are a number of automated tools freely available that can input lots of different types of data in the hope that one of them will result in an application crash/failure.

One of the most effective methods of determining the security stature of an application is to analyse the source code. This high level view is very effective in finding potential vulnerabilities which can then be confirmed by actually injecting against the application.

Countermeasures

To prevent this type of attack will require a solid and robust Software Development Life Cycle (SDLC) that incorporates secure coding practises as well as security testing. This two pronged approach has been found to be very effective in eradicating all kinds of vulnerabilities from bespoke applications.

a 'white list' approach should be taken to filtering input to applications.

A 'white list' approach should be taken to filtering input to applications. Instead of filtering out a large list of known or potentially unsafe character strings ('black list'), only allow known safe characters that relate to the input in question. For instance, if you are entering a date of birth in the DD/MM/YYYY format, only allow numbers 0-9 and forward slashes. This will drastically reduce the probability that any attack will work.

Filtering, combined with secure coding practises where ALL input is validated and verified that it is safe prior to being applied to the application, should result in a robust and secure application.

The next part of this series concentrates on **Improper error handling** and how they are used in exploitation.