

## Abstract

There are several challenges faced while assuring the security of an application. This white paper stresses the need for effective security testing and highlights the essence of path traversal, SQL injection and cross site scripting. No matter how well a given system may have been developed, the nature of today's complex systems with large volumes of code, complex internal interactions, interoperability with uncertain external components, unknown interdependencies coupled with vendor cost and schedule pressures, means that exploitable flaws will always be present or surface over time. Accordingly, security testing must fill the gap between the development and actual operation of these systems. Organizations that have an organized, systematic, comprehensive, on-going, and priority driven security testing regimen are in a much better position to make prudent investments to enhance the security posture of their systems.

## The Problem

The Internet has brought about many changes in the way organizations and individuals conduct business, and it would be difficult to operate effectively without the added efficiency and communications brought about by the Internet. At the same time, the Internet has brought about problems as the result of intruder attacks, both manual and automated, which can cost many organizations excessive amounts of money in damages and lost efficiency.

Thus, organizations need to find methods for achieving their mission goals in using the Internet and at the same time keeping their Internet sites secure from attack. Computer systems today are more powerful and more reliable than in the past; however they are also more difficult to manage.

## Understanding Web Application Security Testing

The black box testing method is a technique for hardening and penetration testing Web applications where the source code to the application is not available to the tester. It forces the tester to look at the Web application from a user's perspective (and therefore, an attacker's perspective). The tester, at first, tries to get a 'feel' for the application and learn its expected behavior. The term black box refers to this Input/Unknown Process/Output approach to testing.

Any strange behavior on the part of the application, in response to strange inputs, is certainly worth investigating as it may mean the developer has failed to validate inputs correctly!

## SQL Injection Vulnerabilities

Many Web application developers (regardless of the environment) do not properly strip user input of potentially "nasty" characters before using that input directly in SQL queries. Depending on the back-end database in use, SQL injection vulnerabilities lead to varying levels of data/system access for the attacker. It may be possible to not only manipulate existing queries, but to UNION in arbitrary data, use sub selects, or append additional queries. In some cases, it may be possible to read in or write out to files, or to execute shell commands on the underlying operating system.

## Locating SQL Injection Vulnerabilities

Often the most effective method of locating SQL injection vulnerabilities is by hand - studying application inputs and inserting special characters. With many of the popular backend, informative errors pages are displayed by default, which can often give clues to the SQL query in use: when attempting SQL injection attacks, you want to learn as much as possible about the syntax of database queries.

The tester attempts to elicit exception conditions and anomalous behavior from the Web application by manipulating the identified inputs - using special characters, white space, SQL keywords, oversized requests, and so forth. Any unexpected reaction from the Web application is noted and investigated. This may take the form of scripting error messages (possibly with snippets of code), server errors (HTTP 500), or half loaded pages.

## Authentications bypass using SQL injection

This is one of the most commonly used examples of SQL injection vulnerability, as it is easy to understand for non-SQL-developers and highlights the extent and severity of these vulnerabilities. One of the simplest ways to validate a user on a Web site is by providing them with a form, which prompts for a username and password. When the form is submitted to the login script (eg. login.asp), the username and password fields are used as variables within an SQL query.

In this scenario, no sanity or validity checking is being performed on the user and pass variables from our form inputs. The developer may have client-side (e.g. JavaScript) checks

on the inputs, but any attacker who understands HTML can bypass these restrictions.

### **PHP and MySQL Injection**

A vulnerable PHP Web application with a MySQL backend, despite PHP escaping numerous 'special' characters (with Magic Quotes enabled), can be manipulated in a similar manner to the above ASP application. MySQL does not allow for direct shell execution; however in many cases it is still possible for the attacker to append arbitrary conditions to queries, or use UNIONS and sub selects to access or modify records in the database.

### **Code and Content Injection**

What is code injection? Code injection vulnerabilities occur where the output or content served from a Web application can be manipulated in such a way that it triggers server-side code execution. In some poorly written Web applications that allow users to modify server-side files (such as by posting to a message board or guestbook) it is sometimes possible to inject code in the scripting language of the application itself. This vulnerability hinges upon the manner in which the application loads and passes through the contents of these manipulated files - if this is done before the scripting language is parsed and executed, the user-modified content may also be subject to parsing and execution.

### **Path Traversal and URIs**

A common use of Web applications is the act as a wrapper for files of Web content, opening them and return in them wrapped in chunks of HTML. Once again, sanity checking is the key. If the variable being read in to specify the file to be wrapped is not checked, a relative path can be entered.

<http://www.example.com/index.php?file=../../etc/passwd>

This request would return the contents of /etc/passwd unless additional stripping of the path character (..) had been performed on the file variable.

This problem is compounded by the automatic handling of URIs by many modern Web scripting technologies, including PHP, Java and Microsoft's .NET. If this is supported on the target environment, vulnerable applications can be used as an open relay or proxy:

<http://www.example.com/index.php?file=http://www.google.com/>

This flaw is one of the easiest security issues to spot and rectify, although it remains common on smaller sites whose application code performs basic content wrapping. The problem can be mitigated in two ways. First, by implementing an internal numeric index to the documents or, by using files named in numeric sequence with a static prefix and suffix. Second, by stripping any path characters such as [/.] which attackers could use to access resources outside of the application's directory tree.

### **Cross Site Scripting**

Cross Site Scripting attack (a form of content-injection attack) differs from the many other attack methods in that it affects the client-side of the application (i.e. the user's browser). Cross Site Scripting (XSS) occurs wherever a developer incorrectly allows a user to manipulate HTML output from the application - this may be in the result of a search query, or any other output from the application where the user's input is displayed back to the user without any stripping of HTML content.

A simple example of XSS can be seen in the following URL:

<http://server.example.com/browse.cfm?categoryID=1&name=Books>

In this example the content of the 'name' parameter is displayed on the returned page. A user could submit the following request:

<http://server.example.com/browse.cfm?categoryID=1&name=<h1>Books>

If the characters < > are not being correctly stripped or escaped by this application, the "<h1>" would be returned within the page and would be parsed by the browser as valid html. A better example would be as follows:

[http://server.example.com/browse.cfm?categoryID=1&name=<script>alert\(document.cookie\);</script>](http://server.example.com/browse.cfm?categoryID=1&name=<script>alert(document.cookie);</script>)

In this case, we have managed to inject JavaScript into the resulting page. The relevant cookie (if any) for this session would be displayed in a popup box upon submitting this request.

This can be abused in a number of ways, depending on the intentions of the attacker. A short piece of JavaScript to submit a user's cookie to an arbitrary site could be placed into this URL. The request could then be hex-encoded and sent to another user, in the hope that they open the URL. Upon clicking the trusted link, the user's cookie would be submitted to the external site. If the original site relies on cookies alone for authentication, the user's account would be compromised.

In most cases, XSS would only be attempted from a reputable or widely-used site, as a user is more likely to click on a long, encoded URL if the server domain name is trusted. This kind of attack does not allow for any access to the client beyond that of the affected domain (in the user's browser security settings).

## Solution

Security testing aims to address inherent flaws and weaknesses in applications to ensure that the final product is robust, and can safeguard sensitive data without compromising data confidentiality and integrity. Depending on testing time frames, more specific, or thoroughly comprehensive testing can be performed to determine the robustness of an application in the environment within which it will eventually be deployed.

## Conclusion

Security testing has gained unparalleled significance over the years, and will continue to rise up the popularity chart given the frequent advancements that we have come to notice and appreciate in the world of technology. Moreover, the importance of data integrity and confidentiality have begun to play a significant role, as have the client demands increased in terms of having access to secure software which delivers without compromising sensitive information. Our aim is to perform thorough security testing to highlight shortcomings and weaknesses in applications to enable developers to build more reliable, and dependable applications in future.