Penetration Testing for Web Applications (Part One)

by Jody Melbourne and David Jorm
last updated June 16, 2003

This is the first in a series of three articles on penetration testing for Web applications. The first installment
tester with an overview of Web applications - how they work, how they interact with users, and most impo
expose data and systems with poorly written and secured Web application front-ends.

Note: It is assumed that the reader of this article has some knowledge of the HTTP protocol - specifically, t
POST requests, and the purpose of various header fields. This information is available in RFC2616.

Web applications are becoming more prevalent and increasingly more sophisticated, and as such they are c
online businesses. As with most security issues involving client/server communications, Web application vu
from improper handling of client requests and/or a lack of input validation checking on the part of the deve

The very nature of Web applications - their ability to collate, process and disseminate information over the
ways. First and most obviously, they have total exposure by nature of being publicly accessible. This make
impossible and heightens the requirement for hardened code. Second and most critically from a penetratio
process data elements from within HTTP requests - a protocol that can employ a myriad of encoding and e

Most Web application environments (including ASP and PHP, which will both be used for examples through
data elements to the developer in a manner that fails to identify how they were captured and hence what k
checking should apply to them. Because the Web "environment" is so diverse and contains so many forms
validation and sanity checking is the key to Web applications security. This involves both identifying and e
every user-definable data element, as well as a sufficient understanding of the source of all data elements
potentially user definable.

The Root of the Issue: Input Validation

Input validation issues can be difficult to locate in a large codebase with lots of user interactions, which is t
developers employ penetration testing methodologies to expose these problems. Web applications are, how
traditional forms of attack. Poor authentication mechanisms, logic flaws, unintentional disclosure of conten
and traditional binary application flaws (such as buffer overflows) are rife. When approaching a Web applic
all this must be taken into account, and a methodical process of input/output or "blackbox" testing, in addi
auditing or "whitebox" testing, must be applied.

What exactly is a Web application?

A Web application is an application, generally comprised of a collection of scripts, that reside on a Web ser
or other sources of dynamic content. They are fast becoming ubiquitous as they allow service providers an
manipulate information in an (often) platform-independent manner via the infrastructure of the Internet. S
applications include search engines, Webmail, shopping carts and portal systems.

How does it look from the users perspective?

Web applications typically interact with the user via FORM elements and GET or POST variables (even a 'Cli
FORM submission). With GET variables, the inputs to the application can be seen within the URL itself, how
often necessary to study the source of form-input pages (or capture and decode valid requests) in order to

An example HTTP request that might be provided to a typical Web application is as follows:

```
GET /sample.php?var=value&var2=value2 HTTP/1.1     | HTTP-METHOD REQUEST-URI PROTOCOL/VERSI
Session-ID: 361873127da673c                        | Session-ID Header
```

```
Host: www.webserver.com                              | Host Header
<CR><LF><CR><LF>                                     | Two carriage return line feeds
```

Every element of this request can potentially be used by the Web application processing the request. The R
of code that will be invoked along with the query string: a separated list of &variable=value pairs defining i
main form of Web applications input. The Session-ID header provides a token identifying the client's establ
form of authentication. The Host header is used to distinguish between virtual hosts sharing the same IP a
parsed by the Web server, but is, in theory, within the domain of the Web application.

As a penetration tester you must use all input methods available to you in order to elicit exception conditio
you cannot be limited to what a browser or automatic tools provide. It is quite simple to script HTTP reques
shell scripts using netcat. The process of exhaustive blackbox testing a Web application is one that involves
determining the expected input, manipulating or otherwise corrupting this input, and analysing the output
unexpected behaviour.

The Information Gathering Phase

Fingerprinting the Web Application Environment

One of the first steps of the penetration test should be to identify the Web application environment, includi
Web server software in use, and the operating system of the target server. All of these crucial details are s
Web application server through the following steps:

1. **Investigate the output from HEAD and OPTIONS http requests**

   The header and any page returned from a HEAD or OPTIONS request will usually contain a `SERVER:`
   Web server software version and possibly the scripting environment or operating system in use.

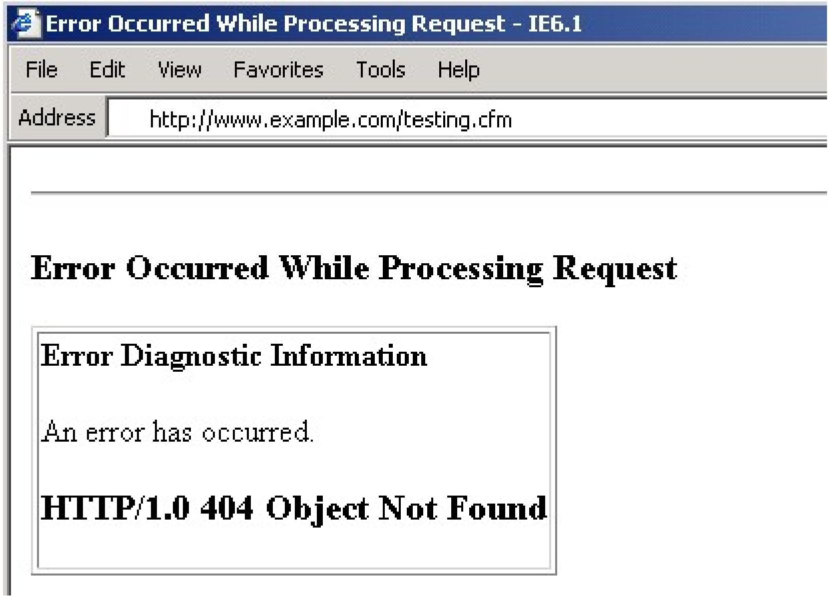
   ```
   OPTIONS / HTTP/1.0

   HTTP/1.1 200 OK
   Server: Microsoft-IIS/5.0
   Date: Wed, 04 Jun 2003 11:02:45 GMT
   MS-Author-Via: DAV
   Content-Length: 0
   Accept-Ranges: none
   DASL: <DAV:sql>
   DAV: 1, 2
   Public: OPTIONS, TRACE, GET, HEAD, DELETE, PUT, POST, COPY, MOVE, MKCOL, PROPFIND, PI
   SEARCH
   Allow: OPTIONS, TRACE, GET, HEAD, COPY, PROPFIND, SEARCH, LOCK, UNLOCK
   Cache-Control: private
   ```

2. **Investigate the format and wording of 404/other error pages**

   Some application environments (such as ColdFusion) have customized and therefore easily recogniz
   give away the software versions of the scripting language in use. The tester should deliberately requ
   alternate request methods (POST/PUT/Other) in order to glean this information from the server.

   Below is an example of a ColdFusion 404 error page:



3. Test for recognised file types/extensions/directories

Many Web services (such as Microsoft IIS) will react differently to a request for a known and support unknown extension. The tester should attempt to request common file extensions such as .ASP, .HT any unusual output or error codes.

```
GET /blah.idq HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 04 Jun 2003 11:12:24 GMT
Content-Type: text/html

<HTML>The IDQ file blah.idq could not be found.
```

4. Examine source of available pages

The source code from the immediately accessible pages of the application front-end may give clues environment.

```
<title>Home Page</title>
<meta content="Microsoft Visual Studio 7.0" name="GENERATOR">
<meta content="C#" name="CODE_LANGUAGE">
<meta content="JavaScript" name="vs_defaultClientScript">
```

In this situation, the developer appears to be using MS Visual Studio 7. The underlying environment with .NET framework.

5. Manipulate inputs in order to elicit a scripting error

In the example below the most obvious variable (ItemID) has been manipulated to fingerprint the W

6. **TCP/ICMP and Service Fingerprinting** Using traditional fingerprinting tools such as Nmap and Qu application fingerprinting tools Amap and WebServerFP, the penetration tester can gain a more accu operating systems and Web application environment than through many other methods. NMAP and (the host's TCP/IP implementation to determine the operating system and, in some cases, the kernel Application fingerprinting tools rely on data such as Server HTTP headers to identify the host's applic

Hidden form elements and source disclosure

In many cases developers require inputs from the client that should be protected from manipulation, such a dynamically generated and served to the client, and required in subsequent requests. In order to prevent u manipulating these inputs, developers use form elements with a HIDDEN tag. Unfortunately, this data is in the rendered version of the page - not within the source.

There have been numerous examples of poorly written ordering systems that would allow users to save a l pages, edit HIDDEN variables such as price and delivery costs, and resubmit their request. The Web applica authentication or cross-checking of form submissions, and the order would be dispatched at a discounted p

```
<FORM METHOD="LINK" ACTION="/shop/checkout.htm">
<INPUT TYPE="HIDDEN" name="quoteprice" value="4.25">Quantity: <INPUT TYPE="text"
NAME="totalnum"> <INPUT TYPE="submit" VALUE="Checkout">
</FORM>
```

This practice is still common on many sites, though to a lesser degree. Typically only non-sensitive informa fields, or the data in these fields is encrypted. Regardless of the sensitivity of these fields, they are still anc by the blackbox penetration tester.

All source pages should be examined (where feasible) to determine if any sensitive or useful information ha by the developer - this may take the form of active content source within HTML, pointers to included or link file/directory permissions on critical source files. Any referenced executables and scripts should be probed,

Javascript and other client-side code can also provide many clues as to the inner workings of a Web applica when blackbox testing. Although the whitebox (or 'code-auditing') tester has access to the application's log information is a luxury which can provide for further avenues of attack. For example, take the following chu

```
<INPUT TYPE="SUBMIT" onClick="
if (document.forms['product'].elements['quantity'].value >= 255) {
   document.forms['product'].elements['quantity'].value='';
   alert('Invalid quantity');
   return false;
} else {
   return true;
```

```
}
">
```

This suggests that the application is trying to protect the form handler from quantity values of 255 of more `tinyint` field in most database systems. It would be trivial to bypass this piece of client-side validation, ins the 'quantity' GET/POST variable and see if this elicits an exception condition from the application.

Determining Authentication Mechanisms

One of the biggest shortcomings of the Web applications environment is its failure to provide a strong auth more concern is the frequent failure of developers to apply what mechanisms are available effectively. It sh that the term Web applications environment refers to the set of protocols, languages and formats - HTTP, H etc. - that are used as a platform for the construction of Web applications. HTTP provides two forms of auth These are both implemented as a series of HTTP requests and responses, in which the client requests a res authentication and the client repeats the request with authentication credentials. The difference is that *Bas* and *Digest* authentication encrypts the credentials using a `nonce` (time sensitive hash value) provided by th key.

Besides the obvious problem of clear text credentials when using *Basic*, there is nothing inherently wrong v this clear-text problem be mitigated by using HTTPS. The real problem is twofold. First, since this authentic server, it is not easily within the control of the Web application without interfacing with the Web server's au Therefore custom authentication mechanisms are frequently used. These open a veritable Pandora's box of Second, developers often fail to correctly assess every avenue for accessing a resource and then apply autl accordingly.

Given this, penetration testers should attempt to ascertain both the authentication mechanism that is being is being applied to every resource within the Web application. Many Web programming environments offer user provides a cookie or a Session-ID HTTP header containing a psuedo-unique string identifying their aut vulnerable to attacks such as brute forcing, replay, or re-assembly if the string is simply a hash or concate known elements.

Every attempt should be made to access every resource via every entry point. This will expose problems w as a main menu or portal page requires authentication but the resources it in turn provides access to do no application providing access to various documents as follows. The application requires authentication and tl documents the user is authorised to access, each document presented as a link to a resource such as:

```
http://www.server.com/showdoc.asp?docid=10
```

Although reaching the menu requires authentication, the showdoc.asp script requires no authentication itse requested document, allowing an attacker to simply insert the docid GET variable of his desire and retrieve as it sounds this is a common flaw in the wild.

Conclusions

In this article we have presented the penetration tester with an overview of web applications and how web user inputs. We have also shown the importance of fingerprinting the target environment and developing a end of an application. Equipped with this information, the penetration tester can proceed to targeted vulne next installment in this series will introduce code and content-manipulation attacks, such as PHP/ASP code Server-Side Includes and Cross-site scripting.