

Web Application Report

This report includes important security information about your Web Application.

Custom Report

Samstag, 18. April 2009

Custom Report

Web Application Report

This report was created by WebScanService 3.14

Scanned Web Application: <http://hackme.webscanservice.com/>

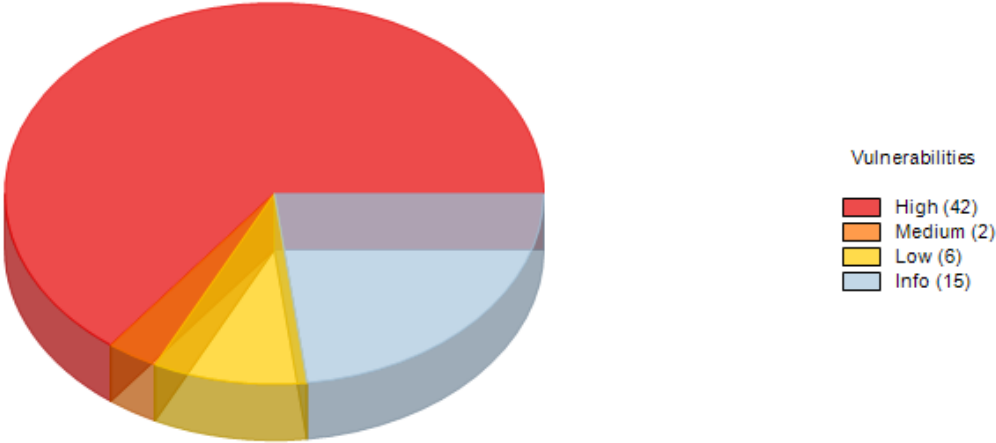
Scan ID: 4

Content

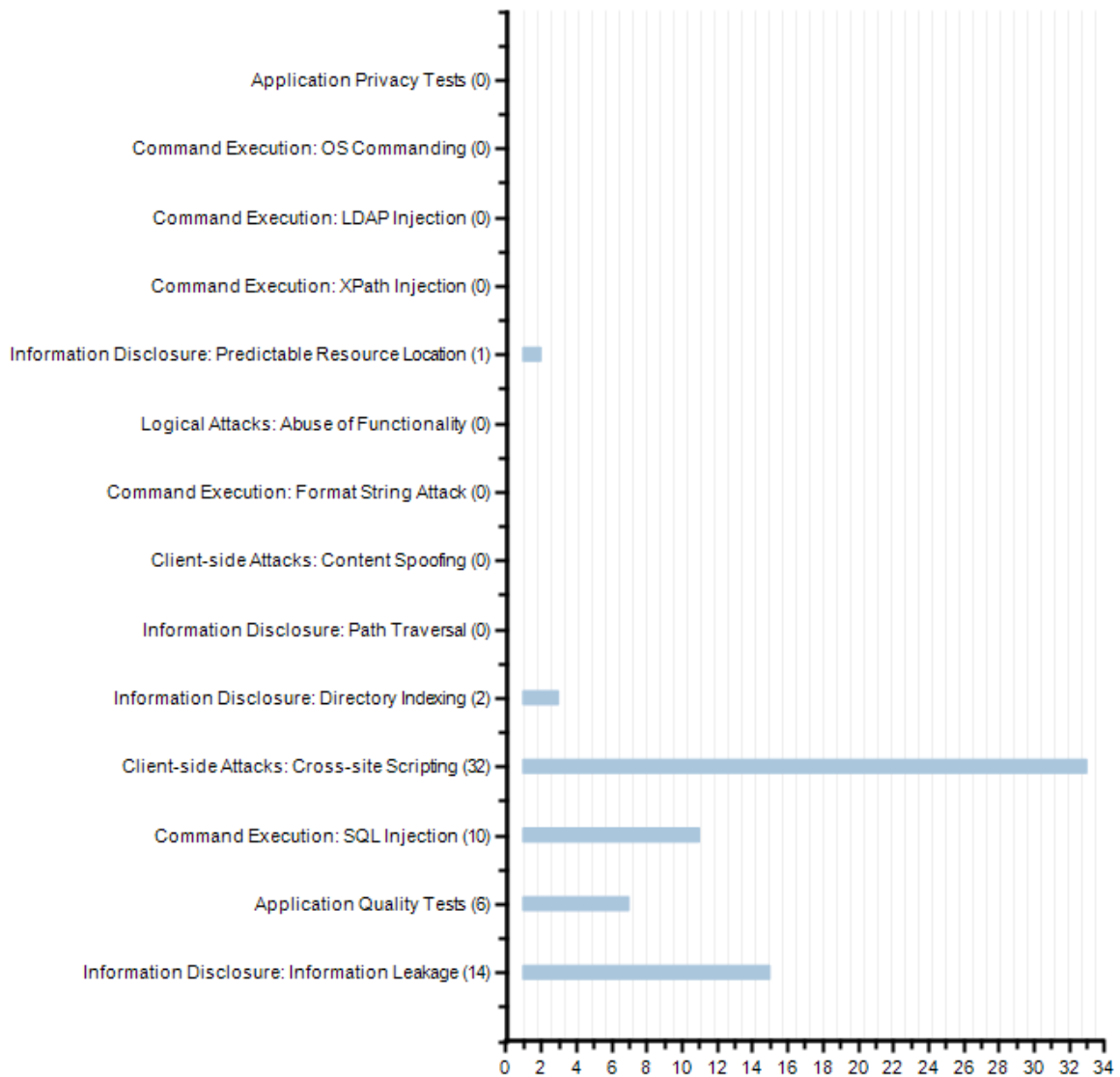
This report contains following sections:

- Executive Summary
- Detailed Security Issues
- Advisores & Fix Recommendations
- Remediation Tasks

Vulnerabilities By Severity



Vulnerabilities By Threat Classification



Executive Summary

Security Risks

Following are the security risks that appeared most often in the application. To explore which issues included these risks, please refer to the 'Detailed Security Issues' section in this report.

- It is possible to view and download the contents of certain web application virtual directories, which may contain restricted files
- It is possible to gather sensitive information about the web application such as usernames, passwords, machine name and/or sensitive file locations
- It is possible to view, modify or delete database entries and tables
- It is possible to gather sensitive debugging information
- It is possible to steal or manipulate customer session and cookies, which may be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user

Vulnerable URLs

18% of the urls had test results that included security issues

Scanned URLs

108 URLs were scanned by WebScanService

Security Issue Possible Causes

Following are the most common causes for the security issues found in the application. The causes below are those that repeated in the maximal number of issues. To explore which issues included these causes, please refer to the 'Detailed Security Issues' section in this report.

- Temporary files were left in production environment
- Insecure web application programming or configuration
- Directory browsing is enabled
- Proper bounds checking were not performed on incoming parameter values
- Debugging information was left by the programmer in web pages

URLs with the most security issues

- <http://hackme.webscanservice.com/feedback.aspx> (14)
- <http://hackme.webscanservice.com/> (13)
- <http://hackme.webscanservice.com/bank/login.aspx> (13)
- <http://hackme.webscanservice.com/subscribe.aspx> (9)
- <http://hackme.webscanservice.com/bank/> (1)

Security Issue Cause Distribution

100% Generic-related Security Issues (65 out of a total of 65 issues).

Generic application-related Security Issues can usually be fixed by application developers, as they result from defects in the application code.

0% Application and Platform Security Issues (0 out of a total 65 issues).

Application and Platform Security Issues can usually be fixed by system and network administrators as these security issues result from misconfiguration of, or defects in 3rd party products.

Test Overview

Security Tests

Following are the performed security tests.

- Application Error
- Blind SQL Injection
- Cross-Site Scripting
- Cross-Site Scripting in Path
- Directory Listing
- Directory Traversal Arbitrary File Download
- Frame Spoofing
- PHP Code Injection
- PHP Local File Inclusion
- PHP Remote File Inclusion
- SQL Injection
- Backup File Download
- Email Address Found
- HTML Comment Found
- Trojan Scripts
- XPath Injection
- LDAP Injection
- Parameter System Command Execution
- Unencrypted Login Request
- Improper URL Redirection
- CVS Directory
- Password File Detected
- PHPInfo Information Disclosure
- ASP.NET Application Trace Information Leakage
- Test Script Detected
- Unix Parameter Traversal
- Windows Parameter Traversal
- Include Files Source Disclosure
- IIS Global.asa and Global.asax Retrieval
- WS_FTP.log Information Leakage
- Oracle Log File Information Disclosure
- Oracle Error Log Found
- Apache access_log Information Disclosure

- Apache error_log Information Disclosure
- Web.config Found
- HTTP Response Splitting
- PHP Session_Start Path Disclosure
- Macromedia Dreamweaver Remote Database Scripts Information Leakage
- Web Application Source Code Disclosure Pattern Found
- Potential Order Information Found
- Potential Registration Information Found
- ASP.NET Custom Error Path Disclosure

Detailed Security Issues

<http://hackme.webscanservice.com/> (13 Security Issues)

(1 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	<a href="http://hackme.webscanservice.com/search.aspx?txtSearch=<script>alert('XSS Test Successful');</script>">http://hackme.webscanservice.com/search.aspx?txtSearch=<script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(2 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	'><script>alert('XSS Test Successful');</script>">http://hackme.webscanservice.com/search.aspx?txtSearch=>'><script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(3 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="></STYLE><STYLE>@import"javascript:alert('XSS Test Successful');</STYLE> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(4 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch=>"><script>alert("XSS Test Successful");</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(5 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="></IFRAME><script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(6 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="></style><script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(7 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="></title><script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(8 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="><SCRIPT SRC=http://appliance.webscanservice.com/tests/xss.js></SCRIPT><" (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(9 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch='><SCRIPT SRC=http://appliance.webscanservice.com/tests/xss.js></SCRIPT><<' (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(10 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch="><STYLE>@import'http://appliance.webscanservice.com/tests/xss.css';</STYLE> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(11 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/search.aspx?txtSearch=</TextArea><script>alert('XSS Test Successful');</script> (txtSearch)
Remediation Task	Filter out hazardous characters from user input

(12 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	<code>http://hackme.webscanservice.com/search.aspx?txtSearch=>"> (txtSearch)</code>
Remediation Task	Filter out hazardous characters from user input

(13 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	<code>http://hackme.webscanservice.com/search.aspx?txtSearch=--><script>alert('XSS Test Successful');</script> (txtSearch)</code>
Remediation Task	Filter out hazardous characters from user input

<http://hackme.webscanservice.com/bank/login.aspx> (13 Security Issues)

(1 of 13) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	<code>http://hackme.webscanservice.com/bank/login.aspx (uid)</code>
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(2 of 13) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (uid)
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(3 of 13) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(4 of 13) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(5 of 13) Blind SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Filter out hazardous characters from user input

(6 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (uid)
Remediation Task	Filter out hazardous characters from user input

(7 of 13) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (uid)
Remediation Task	Filter out hazardous characters from user input

(8 of 13) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx
Remediation Task	Remove html comments

(9 of 13) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (uid)
Remediation Task	Filter out hazardous characters from user input

(10 of 13) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Filter out hazardous characters from user input

(11 of 13) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Filter out hazardous characters from user input

(12 of 13) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Filter out hazardous characters from user input

(13 of 13) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx (passw)
Remediation Task	Filter out hazardous characters from user input

<http://hackme.webscanservice.com/feedback.aspx> (14 Security Issues)

(1 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(2 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(3 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(4 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(5 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(6 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(7 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(8 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(9 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(10 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(11 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(12 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(13 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

(14 of 14) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/comment.aspx (name)
Remediation Task	Filter out hazardous characters from user input

<http://hackme.webscanservice.com/bank/> (1 Security Issues)

(1 of 1) Directory Listing

Severity	Medium
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/
Remediation Task	Modify the server configuration to deny directory listing, and install the latest security patches available

<http://hackme.webscanservice.com/bank/account.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/account.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/apply.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/apply.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/main.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/main.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/mozxpath.js> (1 Security Issues)

(1 of 1) Email Address Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/mozxpath.js
Remediation Task	Remove email address

<http://hackme.webscanservice.com/bank/transaction.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/transaction.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/transfer.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/transfer.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/subscribe.aspx> (9 Security Issues)

(1 of 9) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(2 of 9) Application Error

Severity	Low
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions

(3 of 9) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(4 of 9) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(5 of 9) Cross-Site Scripting

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(6 of 9) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(7 of 9) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(8 of 9) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

(9 of 9) SQL Injection

Severity	High
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/subscribe.aspx (txtEmail)
Remediation Task	Filter out hazardous characters from user input

<http://hackme.webscanservice.com/bank/> (1 Security Issues)

(1 of 1) Directory Listing

Severity	Medium
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/
Remediation Task	Modify the server configuration to deny directory listing, and install the latest security patches available

<http://hackme.webscanservice.com/bank/account.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/account.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/apply.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/apply.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/login.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/login.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/main.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/main.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/mozxpath.js> (1 Security Issues)

(1 of 1) Email Address Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/mozxpath.js
Remediation Task	Remove email address

<http://hackme.webscanservice.com/bank/transaction.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/transaction.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/bank/transfer.aspx> (1 Security Issues)

(1 of 1) HTML Comment Found

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/bank/transfer.aspx
Remediation Task	Remove html comments

<http://hackme.webscanservice.com/test.aspx> (1 Security Issue)

(1 of 1) Test Script Detected

Severity	Info
Test Type	Generic
Vulnerable URL	http://hackme.webscanservice.com/test.aspx
Remediation Task	Remove file

Advisories & Fix Recommendations

Application Error

Generic

WASC Threat Classification

Application Quality Tests

CVE Reference

Security Risks

It is possible to gather sensitive debugging information

Possible Cause

Proper bounds checking were not performed on incoming parameter values

Proper bounds checking were not performed on incoming parameter values

Technical Description

<p>Often during a penetration test on web applications we come up against many error codes generated from applications or web servers. It's possible to cause these errors to be displayed by using a particular request, either specially crafted with tools or created manually. These codes are very useful to penetration testers during their activities because they reveal a lot of information about databases, bugs, and other technological components directly linked with web applications. </p>

General Fix Recommendations

[1] Check incoming requests for the presence of all expected parameters and values. When a parameter is missing, issue a proper error message or use default values.

[2] The application should verify that its input consists of valid characters (after decoding). For example, an input value containing the null byte (encoded as %00), apostrophe, quotes, etc. should be rejected.

[3] Enforce values in their expected ranges and types. If your application expects a certain parameter to have a value from a certain set, then the application should ensure that the value it receives indeed belongs to the set. For example, if your application expects a value in the range 10..99, then it should make sure that the value is indeed numeric, and that its value is in 10..99.

[4] Verify that the data belongs to the set offered to the client.

[5] Do not output debugging error messages and exceptions in a production environment.

.NET Fix Recommendation

You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation (for example, testing for valid dates or values within a range), plus ways to provide custom-written validation. In addition, validation controls allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page's class file, including both HTML and Web server controls.

To make sure that all the required parameters exist in a request, use the "RequiredFieldValidator" validation control. This control ensures that the user does not skip an entry in the web form.

To make sure user input contains only valid values, you can use one of the following validation controls:

[1] "RangeValidator": checks that a user's entry (value) is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.

[2] "RegularExpressionValidator": checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

Important note: validation controls do not block user input or change the flow of page processing; they only set an error state, and produce error messages. It is the programmer's responsibility to test the state of the controls in the code before performing further application-specific actions.

There are two ways to check for user input validity:

1. Test for a general error state:

In your code, test the page's IsValid property. This property rolls up the values of the IsValid properties of all the validation controls on the page (using a logical AND). If one of the validation controls is set to invalid, the page's property will return false.

2. Test for the error state of individual controls:

Loop through the page's Validators collection, which contains references to all the validation controls. You can then examine the IsValid property of each validation control.

Here are two possible ways to protect your web application against SQL injection attacks:

[1] Use a stored procedure rather than a dynamically built SQL query string. The manner in which parameters are passed to SQL Server stored procedures prevents the use of apostrophes and hyphens.

Example of how to use stored procedures in ASP.NET:

```
' Visual Basic example
Dim DS As DataSet
Dim MyConnection As SqlConnection
Dim MyCommand As SqlDataAdapter
Dim SelectCommand As String = "select * from users where username = @username"
...
MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20))
MyCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value
// C# example
String selectCmd = "select * from Authors where state = @username";
SqlConnection myConnection = new SqlConnection("server=...");
SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection);
myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20));
myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value;
```

[2] You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation (for example, testing for valid dates or values within a range), plus ways to provide custom-written validation. In addition, validation controls allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page's class file, including both HTML and Web server controls.

[PHP Fix Recommendation](#)

** Input Data Validation:

While data validations may be provided as a user convenience on the client-tier, data validation must always be performed on the server-tier. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.

A good design usually requires the web application framework to provide server-side utility routines to validate the

following:

- [1] Required field
- [2] Field data type (all HTTP request parameters are Strings by default)
- [3] Field length
- [4] Field range
- [5] Field options
- [6] Field pattern
- [7] Cookie values
- [8] HTTP Response

A good practice is to implement a function or functions that validates each application parameter. The following sections describe some example checking.

[1] Required field

Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.

Example of how to validate required fields:

```
// PHP example to validate required fields
function validateRequired($input) {
    ...
    $pass = false;
    if (strlen(trim($input))>0){
        $pass = true;
    }
    return $pass;
    ...
}
...
if (validateRequired($fieldName)) {
    // fieldName is valid, continue processing request
    ...
}
```

[2] Field data type

In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are of type String. The developer is responsible for verifying the input is of the correct data type.

[3] Field length

Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.

[4] Field range

Always ensure that the input parameter is within a range as defined by the functional requirements.

[5] Field options

Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a

malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.

[6] Field pattern

Always check that user input matches a pattern as defined by the functionality requirements. For example, if the `userName` field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:
`^[a-zA-Z0-9]+$`

[7] Cookie value

The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.

[8] HTTP Response

[8-1] Filter user input

To guard the application against cross-site scripting, the developer should sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:

`<`; `>`; `"`; `'`; `%`; `)`; `(`; `&`; `+`

PHP includes some automatic sanitation utility functions, such as `htmlspecialchars()`:

```
$input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

In addition, in order to avoid UTF-7 variants of Cross-site Scripting, you should explicitly define the Content-Type header of the response, for example:

```
<?php  
header('Content-Type: text/html; charset=UTF-8');  
?>
```

[8-2] Secure the cookie

When storing sensitive data in a cookie and transporting it over SSL, make sure that you first set the secure flag of the cookie in the HTTP response. This will instruct the browser to only use that cookie over SSL connections.

You can use the following code example, for securing the cookie:

```
<?php  
$value = 'some_value';  
$time = time()+3600;  
$path = '/application/';  
$domain = '.example.com';  
$secure = 1;  
setcookie('CookieName', $value, $time, $path, $domain, $secure, TRUE);  
?>
```

In addition, we recommend that you use the HttpOnly flag. When the HttpOnly flag is set to TRUE the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers).

The HttpOnly flag was Added in PHP 5.2.0.

REFERENCES

[1] Mitigating Cross-site Scripting With HTTP-only Cookies: <http://msdn2.microsoft.com/en-us/library/ms533046.aspx>

[2] PHP Security Consortium: <http://phpsec.org/>

[3] PHP & Web Application Security Blog (Chris Shiflett): <http://shiflett.org/>

References and Relevant Links

<http://www.wiretrip.net/rfp/txt/rfp2k01.txt>

<http://www.blackhat.com/presentations/win-usA-01/Litchfield/BHWin01Litchfield.doc>

<http://www.cert.org/advisories/CA-1997-25.html>

Blind SQL Injection

Generic

WASC Threat Classification

Command Execution: SQL Injection

http://www.webappsec.org/projects/threat/classes/sql_injection.shtml

CVE Reference

Security Risks

It is possible to view, modify or delete database entries and tables

Possible Cause

Sanitation of hazardous characters was not performed correctly on user input

Technical Description

When an attacker executes SQL Injection attacks sometimes the server responds with error messages from the database server complaining that the SQL Query's syntax is incorrect. Blind SQL injection is identical to normal SQL Injection except that when an attacker attempts to exploit an application rather than getting a useful error message they get a generic page specified by the developer instead. This makes exploiting a potential SQL Injection attack more difficult but not impossible. An attacker can still steal data by asking a series of True and False questions through sql statements.

General Fix Recommendations

There are several issues whose remediation lies in sanitizing user input.

By verifying that user input does not contain hazardous characters, it is possible to prevent malicious users from causing your application to execute unintended operations, such as launch arbitrary SQL queries, embed Javascript code to be executed on the client side, run various operating system commands etc.

It is advised to filter out all the following characters:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign) [4] \$ (dollar sign) [5] % (percent sign) [6] @ (at sign) [7] ' (single apostrophe) [8] " (quotation mark) [9] \ (backslash-escaped apostrophe) [10] \" (backslash-escaped quotation mark) [11] <> (triangular parenthesis) [12] () (parenthesis) [13] + (plus sign) [14] CR (Carriage return, ASCII 0x0d) [15] LF (Line feed, ASCII 0x0a) [16] , (comma sign) [17] \ (backslash)

The following sections describe the various issues, their fix recommendations and the hazardous characters that might trigger these issues:

SQL injection and blind SQL injection:

- Make sure the value and type (such as Integer, Date, etc.) of the user input is valid and expected by the application.
- Use stored procedures to abstract data access so that users do not directly access tables or views. When using stored procedures, use the ADO command object to implement them, so that variables are strongly typed.
- Sanitize input to exclude context-changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) [3] \ (backslash-escaped apostrophe) [4] \" (backslash-escaped quotation mark) [5]) (closing parenthesis) [6] ; (semicolon)

Cross site scripting:

- Sanitize user input and filter out JavaScript code. We suggest that you filter the following characters:

[1] <> (triangular parenthesis) [2] " (quotation mark) [3] ' (single apostrophe) [4] % (percent sign) [5] ; (semicolon) [6] () (parenthesis) [7] & (ampersand sign) [8] + (plus sign)

- For UTF-7 attacks:

[-] When possible, it is recommended to enforce a specific charset encoding (using 'Content-Type' header or <meta> tag).

HTTP response splitting:

Sanitize user input (at least, such input that is later embedded in HTTP responses).

Make sure that malicious characters are not part of the input, such as:

[1] CR (Carriage return, ASCII 0x0d) [2] LF (Line feed, ASCII 0x0a)

Remote command execution:

Sanitize input to exclude symbols that are meaningful to the operating system's command execution, such as:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign)

Shell command execution:

A. Never pass unchecked user-input to Perl commands such as: eval(), open(), sysopen(), system().

B. Make sure malicious characters are not part of the input, such as:

[1] \$ (dollar sign) [2] % (percent sign) [3] @ (at sign)

XPath injection:

Sanitize input to exclude context changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) Etc.

LDAP injection:

A. Use positive validation. Alphanumeric filtering (A..Z,a..z,0..9) is suitable for most LDAP queries.

B. Special LDAP characters which should be filtered out or escaped:

[1] A space or "#" character at the beginning of the string [2] A space character at the end of the string [3] , (comma sign) [4] + (plus sign) [5] " (quotation mark) [6] \ (backslash) [7] <> (triangular parenthesis) [8] ; (semicolon sign) [9] () (parenthesis)

[.NET Fix Recommendation](#)

Here are two possible ways to protect your web application against SQL injection attacks:

[1] Use a stored procedure rather than dynamically built SQL query string. The way parameters are passed to SQL Server stored procedures, prevents the use of apostrophes and hyphens.

Here is a simple example of how to use stored procedures in ASP.NET:

```
' Visual Basic example
Dim DS As DataSet
Dim MyConnection As SqlConnection
Dim MyCommand As SqlDataAdapter
Dim SelectCommand As String = "select * from users where username = @username"
...
MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20))
MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value
// C# example
String selectCmd = "select * from Authors where state = @username";
SqlConnection myConnection = new SqlConnection("server=...");
SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection);
myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20));
myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value;
```

[2] You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation - for example, testing for valid dates or values within a range - plus ways to provide custom-written validation. In addition, validation controls allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page's class file, including both HTML and Web server controls.

In order to make sure user input contains only valid values, you can use one of the following validation controls:

a. "RangeValidator": checks that a user's entry (value) is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.

b. "RegularExpressionValidator": checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

Important note: validation controls do not block user input or change the flow of page processing; they only set an error state, and produce error messages. It is the programmer's responsibility to test the state of the controls in the code before performing further application-specific actions.

There are two ways to check for user input validity:

1. Testing for a general error state:

In your code, test the page's IsValid property. This property rolls up the values of the IsValid properties of all the validation controls on the page (using a logical AND). If one of the validation controls is set to invalid, the page's property will return false.

2. Testing for the error state of individual controls:

Loop through the page's Validators collection, which contains references to all the validation controls. You can then examine the IsValid property of each validation control.

PHP Fix Recommendation

** Filter User Input

Before passing any data to a SQL query, it should always be properly filtered with whitelisting techniques. This cannot be over-emphasized. Filtering user input will correct many injection flaws before they arrive at the database.

** Quote User Input

Regardless of data type, it is always a good idea to place single quotes around all user data if this is permitted by the database. MySQL allows this formatting technique.

** Escape the Data Values

If you're using MySQL 4.3.0 or newer, you should escape all strings with `mysql_real_escape_string()`. If you are using an older version of MySQL, you should use the `mysql_escape_string()` function. If you are not using MySQL, you might choose to use the specific escaping function for your particular database. If you are not aware of an escaping function, you might choose to utilize a more generic escaping function such as `addslashes()`.

If you're using the PEAR DB database abstraction layer, you can use the `DB::quote()` method or use a query placeholder like `?`, which automatically escapes the value that replaces the placeholder.

REFERENCES

http://ca3.php.net/mysql_real_escape_string

http://ca3.php.net/mysql_escape_string

<http://ca3.php.net/addslashes>

<http://pear.php.net/package-info.php?package=DB>

** Input Data Validation:

While data validations may be provided as a user convenience on the client-tier, data validation must always be performed on the server-tier. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.

A good design usually requires the web application framework to provide server-side utility routines to validate the following:

- [1] Required field
- [2] Field data type (all HTTP request parameters are Strings by default)
- [3] Field length
- [4] Field range
- [5] Field options
- [6] Field pattern
- [7] Cookie values
- [8] HTTP Response

A good practice is to implement a function or functions that validates each application parameter. The following sections describe some example checking.

[1] Required field

Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.

Example of how to validate required fields:

```
// PHP example to validate required fields
function validateRequired($input) {
    ...
    $pass = false;
    if (strlen(trim($input))>0){
        $pass = true;
    }
    return $pass;
    ...
}
...
if (validateRequired($fieldName)) {
    // fieldName is valid, continue processing request
    ...
}
```

[2] Field data type

In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are of type String. The developer is responsible for verifying the input is of the correct data type.

[3] Field length

Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.

[4] Field range

Always ensure that the input parameter is within a range as defined by the functional requirements.

[5] Field options

Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.

[6] Field pattern

Always check that user input matches a pattern as defined by the functionality requirements. For example, if the userName field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:
`^[a-zA-Z0-9]+$`

[7] Cookie value

The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.

[8] HTTP Response

[8-1] Filter user input

To guard the application against cross-site scripting, the developer should sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:

`< > " ' % ;) (& +`

PHP includes some automatic sanitation utility functions, such as `htmlspecialchars()`:

```
$input = htmlspecialchars($input, ENT_QUOTES, '&#194;'UTF-8&#194;');
```

In addition, in order to avoid UTF-7 variants of Cross-site Scripting, you should explicitly define the Content-Type header of the response, for example:

```
&lt;?php
header('Content-Type: text/html; charset=UTF-8');
?&gt;
```

[8-2] Secure the cookie

When storing sensitive data in a cookie and transporting it over SSL, make sure that you first set the secure flag of the cookie in the HTTP response. This will instruct the browser to only use that cookie over SSL connections.

You can use the following code example, for securing the cookie:

```
&lt;?php
$value = '&quot;some_value&quot;;
$time = time()+3600;
```

```
$path = &quot;/application/&quot;;  
$domain = &quot;.example.com&quot;;  
$secure = 1;  
setcookie(&quot;CookieName&quot;, $value, $time, $path, $domain, $secure, TRUE);  
&gt;
```

In addition, we recommend that you use the HttpOnly flag. When the HttpOnly flag is set to TRUE the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers).

The HttpOnly flag was Added in PHP 5.2.0.

REFERENCES

[1] Mitigating Cross-site Scripting With HTTP-only Cookies: http://msdn2.microsoft.com/en-us/library/ms533046.aspx

[2] PHP Security Consortium: http://phpsec.org

[3] PHP & Web Application Security Blog (Chris Shiflett): http://shiflett.org

References and Relevant Links

<http://www.blackhat.com/presentations/win-usA-01/Litchfield/BHWin01Litchfield.doc>

<http://shh.thathost.com/text/binary-search-sql-injection.txt>

Cross-Site Scripting

Generic

WASC Threat Classification

Client-side Attacks: Cross-site Scripting

http://www.webappsec.org/projects/threat/classes/cross-site_scripting.shtml

CVE Reference

Security Risks

It is possible to steal or manipulate customer session and cookies, which may be used to impersonate a legitimate user, allowing the hacker to view or alter user records, and to perform transactions as that user

Possible Cause

Sanitation of hazardous characters was not performed correctly on user input

Technical Description

<p>Cross-site Scripting (XSS) is an attack technique that forces a web site to echo attacker-supplied executable code, which loads in a user's browser. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.</p><p>When an attacker gets a user's browser to execute his code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser.</p><p>A Cross-site Scripted user could have his account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Crosssite Scripting attacks essentially compromise the trust relationship between a user and the web site.</p><p>Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to click on any link, just simply view the web page containing the code. Many web sites host bulletin boards where registered users may post messages. A registered user is commonly tracked using a session ID cookie authorizing them to post. If an attacker were to post a message containing a specially crafted JavaScript, a user reading this message could have their cookies and their account compromised.</p>

General Fix Recommendations

There are several issues whose remediation lies in sanitizing user input.

By verifying that user input does not contain hazardous characters, it is possible to prevent malicious users from causing your application to execute unintended operations, such as launch arbitrary SQL queries, embed Javascript code to be executed on the client side, run various operating system commands etc.

It is advised to filter out all the following characters:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign) [4] \$ (dollar sign) [5] % (percent sign) [6] @ (at sign) [7] ' (single apostrophe) [8] " (quotation mark) [9] \ (backslash-escaped apostrophe) [10] \" (backslash-escaped quotation mark) [11] <> (triangular parenthesis) [12] () (parenthesis) [13] + (plus sign) [14] CR (Carriage return, ASCII 0x0d) [15] LF (Line feed, ASCII 0x0a) [16] , (comma sign) [17] \ (backslash)

The following sections describe the various issues, their fix recommendations and the hazardous characters that might trigger these issues:

SQL injection and blind SQL injection:

- A. Make sure the value and type (such as Integer, Date, etc.) of the user input is valid and expected by the application.
- B. Use stored procedures to abstract data access so that users do not directly access tables or views. When using stored procedures, use the ADO command object to implement them, so that variables are strongly typed.
- C. Sanitize input to exclude context-changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) [3] \ (backslash-escaped apostrophe) [4] \" (backslash-escaped quotation mark) [5]) (closing parenthesis) [6] ; (semicolon)

Cross site scripting:

A. Sanitize user input and filter out JavaScript code. We suggest that you filter the following characters:

[1] <> (triangular parenthesis) [2] " (quotation mark) [3] ' (single apostrophe) [4] % (percent sign) [5] ; (semicolon) [6] () (parenthesis) [7] & (ampersand sign) [8] + (plus sign)

C. For UTF-7 attacks:

[-] When possible, it is recommended to enforce a specific charset encoding (using 'Content-Type' header or <meta> tag).

HTTP response splitting:

Sanitize user input (at least, such input that is later embedded in HTTP responses).

Make sure that malicious characters are not part of the input, such as:

[1] CR (Carriage return, ASCII 0x0d) [2] LF (Line feed, ASCII 0x0a)

Remote command execution:

Sanitize input to exclude symbols that are meaningful to the operating system's command execution, such as:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign)

Shell command execution:

A. Never pass unchecked user-input to Perl commands such as: eval(), open(), sysopen(), system().

B. Make sure malicious characters are not part of the input, such as:

[1] \$ (dollar sign) [2] % (percent sign) [3] @ (at sign)

XPath injection:

Sanitize input to exclude context changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) Etc.

LDAP injection:

A. Use positive validation. Alphanumeric filtering (A..Z,a..z,0..9) is suitable for most LDAP queries.

B. Special LDAP characters which should be filtered out or escaped:

[1] A space or "#" character at the beginning of the string [2] A space character at the end of the string [3] , (comma sign) [4] + (plus sign) [5] " (quotation mark) [6] \ (backslash) [7] <> (triangular parenthesis) [8] ; (semicolon sign) [9] () (parenthesis)

[.NET Fix Recommendation](#)

[1] We recommend that you upgrade your server to .NET Framework 2.0 (or newer), which includes inherent security checks that protect against cross site scripting attacks.

[2] You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-use mechanism for all common types of standard validation (for example, tests for valid dates or values within a range). The validation controls also support custom-written validations, and allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page class file, including both HTML and Web server controls.

To make sure that user input contains only valid values, you can use one of the following validation controls:

[1] "RangeValidator": checks that a user's entry (value) is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.

[2] "RegularExpressionValidator": checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

Examples of regular expressions that may help block cross site scripting:

- A possible regular expression, which will deny the basic cross site scripting variants might be: `^[^<]|<[^\a-zA-Z]*[<]?$`
- A generic regular expression, which will deny all of the aforementioned characters might be: `^[^\<>'\\"%{};|)\(\&+]*$`

Important note: validation controls do not block user input or change the flow of page processing; they only set an error state, and produce error messages. It is the programmer's responsibility to test the state of the controls in the code before performing further application-specific actions.

There are two ways to check for user input validity:

1. Test for a general error state:

In your code, test the page's `IsValid` property. This property rolls up the values of the `IsValid` properties of all the validation controls on the page (using a logical AND). If one of the validation controls is set to invalid, the page's property will return false.

2. Test for the error state of individual controls:

Loop through the page's `Validators` collection, which contains references to all the validation controls. You can then examine the `IsValid` property of each validation control.

Finally, we recommend that the Microsoft Anti-Cross Site Scripting Library (v1.5 or higher) be used to encode untrusted user input.

The Anti-Cross Site Scripting library exposes the following methods:

- [1] `HtmlEncode` - Encodes input strings for use in HTML
- [2] `HtmlAttributeEncode` - Encodes input strings for use in HTML attributes
- [3] `JavaScriptEncode` - Encodes input strings for use in JavaScript
- [4] `UrlEncode` - Encodes input strings for use in Universal Resource Locators (URLs)
- [5] `VisualBasicScriptEncode` - Encodes input strings for use in Visual Basic Script
- [6] `XmlEncode` - Encodes input strings for use in XML
- [7] `XmlAttributeEncode` - Encodes input strings for use in XML attributes

To properly use the Microsoft Anti-Cross Site Scripting Library to protect ASP.NET Web-applications, you need to:

Step 1: Review ASP.NET code that generates output

Step 2: Determine whether output includes untrusted input parameters

Step 3: Determine the context which the untrusted input is used as output, and determine which encoding method to use

Step 4: Encode output

Example for Step 3:

Note: If the untrusted input will be used to set an HTML attribute, then the `Microsoft.Security.Application.HtmlAttributeEncode` method should be used to encode the untrusted input. Alternatively, if the untrusted input will be used within the context of JavaScript, then `Microsoft.Security.Application.JavaScriptEncode` should be used to encode.

```
// Vulnerable code
// Note that untrusted input is being treated as an HTML attribute
Literal1.Text = "<hr noshade size=[untrusted input here]>";
// Modified code
Literal1.Text = "<hr noshade size="+Microsoft.Security.Application.AntiXss.HtmlAttributeEncode([untrusted input here])+">";
```

Example for Step 4:

Some important things to remember about encoding outputs:

[1] Outputs should be encoded once.

[2] Output encoding should be done as close to the actual writing of the output as possible. For example, if an application is reading user input, processing the input and then writing it back out in some form, then encoding should happen just before the output is written.

```
// Incorrect sequence
protected void Button1_Click(object sender, EventArgs e)
{
    // Read input
    String Input = TextBox1.Text;
    // Encode untrusted input
    Input = Microsoft.Security.Application.AntiXss.HtmlEncode(Input);
    // Process input
    ...
    // Write Output
    Response.Write("The input you gave was"+Input);
}
// Correct Sequence
protected void Button1_Click(object sender, EventArgs e)
{
    // Read input
    String Input = TextBox1.Text;
    // Process input
    ...
    // Encode untrusted input and write output
    Response.Write("The input you gave was"+
        Microsoft.Security.Application.AntiXss.HtmlEncode(Input));
}
```

PHP Fix Recommendation

** Input Data Validation:

While data validations may be provided as a user convenience on the client-tier, data validation must always be performed on the server-tier. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.

A good design usually requires the web application framework to provide server-side utility routines to validate the following:

- [1] Required field
- [2] Field data type (all HTTP request parameters are Strings by default)
- [3] Field length
- [4] Field range
- [5] Field options
- [6] Field pattern
- [7] Cookie values
- [8] HTTP Response

A good practice is to implement a function or functions that validates each application parameter. The following sections describe some example checking.

[1] Required field

Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.

Example of how to validate required fields:

```
// PHP example to validate required fields
function validateRequired($input) {
    ...
    $pass = false;
    if (strlen(trim($input))>0){
        $pass = true;
    }
    return $pass;
    ...
}
...
if (validateRequired($fieldName)) {
    // fieldName is valid, continue processing request
    ...
}
```

[2] Field data type

In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are of type String. The developer is responsible for verifying the input is of the correct data type.

[3] Field length

Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.

[4] Field range

Always ensure that the input parameter is within a range as defined by the functional requirements.

[5] Field options

Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.

[6] Field pattern

Always check that user input matches a pattern as defined by the functionality requirements. For example, if the userName field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:
`^[a-zA-Z0-9]+$`

[7] Cookie value

The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.

[8] HTTP Response

[8-1] Filter user input

To guard the application against cross-site scripting, the developer should sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:

< > " ' % ;) (& +

PHP includes some automatic sanitation utility functions, such as `htmlspecialchars()`:

```
$input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

In addition, in order to avoid UTF-7 variants of Cross-site Scripting, you should explicitly define the Content-Type header of the response, for example:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
?>
```

[8-2] Secure the cookie

When storing sensitive data in a cookie and transporting it over SSL, make sure that you first set the secure flag of the cookie in the HTTP response. This will instruct the browser to only use that cookie over SSL connections.

You can use the following code example, for securing the cookie:

```
<?php
$value = "some_value";
$time = time()+3600;
$path = "/application/";
$domain = ".example.com";
$secure = 1;
setcookie("CookieName", $value, $time, $path, $domain, $secure, TRUE);
?>
```

In addition, we recommend that you use the `HttpOnly` flag. When the `HttpOnly` flag is set to `TRUE` the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers).

The `HttpOnly` flag was Added in PHP 5.2.0.

REFERENCES

- [1] Mitigating Cross-site Scripting With HTTP-only Cookies: http://msdn2.microsoft.com/en-us/library/ms533046.aspx
- [2] PHP Security Consortium: http://phpsec.org/
- [3] PHP & Web Application Security Blog (Chris Shiflett): http://shiflett.org/

References and Relevant Links

- <http://msdn2.microsoft.com/en-us/security/aa973814.aspx>
- <http://www.cert.org/advisories/CA-2000-02.html>
- <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q252985>
- <http://www.microsoft.com/technet/archive/security/news/crssite.mspx>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGHT000004.asp>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGHT000003.asp>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/PAGHT000001.asp>

Directory Listing

Generic

WASC Threat Classification

Information Disclosure: Directory Indexing

http://www.webappsec.org/projects/threat/classes/directory_indexing.shtml

CVE Reference

Security Risks

It is possible to view and download the contents of certain web application virtual directories, which may contain restricted files

Possible Cause

Directory browsing is enabled

Technical Description

<p>Automatic directory listing/indexing is a web server function that lists all of the files within a requested directory if the normal base file(index.html/home.html/default.htm) is not present.</p><p>When a user requests the main page of a web site, they normally type in a URL such as: http://www.example - using the domain name and excluding a specific file. The web server processes this request and searches the document root directory for the default file name and sends this page to the client. If this page is not present, the web server will issue a directory listing and send the output to the client.</p><p>Essentially, this is equivalent to issuing an "ls" (Unix) or "dir" (Windows) command within this directory and showing the results in HTML form. From an attack and countermeasure perspective, it is important to realize that unintended directory listings may be possible due to software vulnerabilities (discussed in the example section below) combined with a specific web request.</p><p>When a web server reveals a directory's contents, the listing could contain information not intended for public viewing. Often web administrators rely on "Security Through Obscurity" assuming that if there are no hyperlinks to these documents, they will not be found, or no one will look for them. The assumption is incorrect. Today's vulnerability scanners, such as Nikto, can dynamically add additional directories/files to include in their scan based upon data obtained in initial probes.</p><p>By reviewing the /robots.txt file and/or viewing directory indexing contents, the vulnerability scanner can now interrogate the web server further with these new data. Although potentially harmless, Directory Indexing could allow an information leak that supplies an attacker with the information necessary to launch further attacks against the system.</p>

General Fix Recommendations

- [1] Configure the web server to deny listing of directories.
- [2] Download a specific security patch according to the issue existing on your web server or web application. Some of the known directory listing issues are listed in the "References" field of this advisory.
- [3] A Workaround from the "CERT" advisory found in the "References" field of this advisory, to fix the short filenames (8.3 DOS format) problem:
 - a. Use only 8.3-compliant short file names for the files that you want to have protected solely by the web server. On FAT file systems (16-bit) this can be enforced by enabling (setting to 1) the "Win31FileSystem" registry key (registry path: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\FileSystem\).
 - b. On NTFS (32-bit), you can disable the creation of the 8.3-compliant short file name for files with long file names by enabling (setting to 1) the "NtfsDisable8dot3NameCreation" registry key (registry path: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\FileSystem\). However, this step may cause compatibility problems with 16-bit applications.
 - c. Use NTFS-based ACLs (directory or file level access control lists) to augment or replace web server-based security.

.NET Fix Recommendation

PHP Fix Recommendation

References and Relevant Links

[Apache directory listing \(CAN-2001-0729\)](#)

[Microsoft IIS 5.0+WebDav support - directory listing](#)

[Jrun directory listing](#)

[CERT Advisory CA-98.04](#)

Email Address Found

Generic

WASC Threat Classification

Information Disclosure: Information Leakage

http://www.webappsec.org/projects/threat/classes/information_leakage.shtml

CVE Reference

Security Risks

It is possible to gather sensitive information about the web application such as usernames, passwords, machine name and/or sensitive file locations

Possible Cause

Insecure web application programming or configuration

Technical Description

Spambots crawl internet sites, set out to find email addresses in order to build mailing lists for sending unsolicited email (spam).

General Fix Recommendations

Remove any email addresses from the website.

.NET Fix Recommendation

PHP Fix Recommendation

References and Relevant Links

<http://en.wikipedia.org/wiki/Spambot>

HTML Comment Found

Generic

WASC Threat Classification

Information Disclosure: Information Leakage

http://www.webappsec.org/projects/threat/classes/information_leakage.shtml

CVE Reference

Security Risks

It is possible to gather sensitive debugging information

Possible Cause

Debugging information was left by the programmer in web pages

Technical Description

Many web application programmers use HTML comments to help debug the application when needed. Some programmers tend to leave important data, such as: filenames, old links, old code fragments, etc. An attacker who finds these comments can map the application's structure and files, expose hidden paths of the site and study the fragments of code to reverse engineer the application.

General Fix Recommendations

.NET Fix Recommendation

PHP Fix Recommendation

References and Relevant Links

SQL Injection

Generic

WASC Threat Classification

Command Execution: SQL Injection

http://www.webappsec.org/projects/threat/classes/sql_injection.shtml

CVE Reference

Security Risks

It is possible to view, modify or delete database entries and tables

Possible Cause

Sanitation of hazardous characters was not performed correctly on user input

Technical Description

SQL injection is a security vulnerability that occurs in the persistence/database layer of a web application. This vulnerability is derived from the incorrect escaping of variables embedded in SQL statements. It is in fact an instance of a more general class of vulnerabilities based on poor input validation and bad design that can occur whenever one programming or scripting language is embedded inside another.

Threat Modeling :

- SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.
- SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.
- The severity of SQL Injection attacks is limited by the attacker's skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

General Fix Recommendations

There are several issues whose remediation lies in sanitizing user input.

By verifying that user input does not contain hazardous characters, it is possible to prevent malicious users from causing your application to execute unintended operations, such as launch arbitrary SQL queries, embed Javascript code to be executed on the client side, run various operating system commands etc.

It is advised to filter out all the following characters:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign) [4] \$ (dollar sign) [5] % (percent sign) [6] @ (at sign) [7] ' (single apostrophe) [8] " (quotation mark) [9] \ (backslash-escaped apostrophe) [10] \" (backslash-escaped quotation mark) [11] <> (triangular parenthesis) [12] () (parenthesis) [13] + (plus sign) [14] CR (Carriage return, ASCII 0x0d) [15] LF (Line feed, ASCII 0x0a) [16] , (comma sign) [17] \ (backslash)

The following sections describe the various issues, their fix recommendations and the hazardous characters that might trigger these issues:

SQL injection and blind SQL injection:

- Make sure the value and type (such as Integer, Date, etc.) of the user input is valid and expected by the application.
- Use stored procedures to abstract data access so that users do not directly access tables or views. When using stored procedures, use the ADO command object to implement them, so that variables are strongly typed.
- Sanitize input to exclude context-changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) [3] \ (backslash-escaped apostrophe) [4] \" (backslash-escaped quotation mark) [5]) (closing parenthesis) [6] ; (semicolon)

Cross site scripting:

- Sanitize user input and filter out JavaScript code. We suggest that you filter the following characters:

[1] <> (triangular parenthesis) [2] " (quotation mark) [3] ' (single apostrophe) [4] % (percent sign) [5] ; (semicolon) [6] () (parenthesis) [7] & (ampersand sign) [8] + (plus sign)

B. To fix the <script> variant see MS article 821349

C. For UTF-7 attacks:

[1] When possible, it is recommended to enforce a specific charset encoding (using 'Content-Type' header or <meta> tag).

HTTP response splitting:

Sanitize user input (at least, such input that is later embedded in HTTP responses).

Make sure that malicious characters are not part of the input, such as:

[1] CR (Carriage return, ASCII 0x0d) [2] LF (Line feed, ASCII 0x0a)

Remote command execution:

Sanitize input to exclude symbols that are meaningful to the operating system's command execution, such as:

[1] | (pipe sign) [2] & (ampersand sign) [3] ; (semicolon sign)

Shell command execution:

A. Never pass unchecked user-input to Perl commands such as: eval(), open(), sysopen(), system().

B. Make sure malicious characters are not part of the input, such as:

[1] \$ (dollar sign) [2] % (percent sign) [3] @ (at sign)

XPath injection:

Sanitize input to exclude context changing symbols such as:

[1] ' (single apostrophe) [2] " (quotation mark) Etc.

LDAP injection:

A. Use positive validation. Alphanumeric filtering (A..Z,a..z,0..9) is suitable for most LDAP queries.

B. Special LDAP characters which should be filtered out or escaped:

[1] A space or "#" character at the beginning of the string [2] A space character at the end of the string [3] , (comma sign) [4] + (plus sign) [5] " (quotation mark) [6] \ (backslash) [7] <> (triangular parenthesis) [8] ; (semicolon sign) [9] () (parenthesis)

[.NET Fix Recommendation](#)

Here are two possible ways to protect your web application against SQL injection attacks:

[1] Use a stored procedure rather than dynamically built SQL query string. The way parameters are passed to SQL Server stored procedures, prevents the use of apostrophes and hyphens.

Here is a simple example of how to use stored procedures in ASP.NET:

```
' Visual Basic example
Dim DS As DataSet
Dim MyConnection As SqlConnection
Dim MyCommand As SqlDataAdapter
Dim SelectCommand As String = "select * from users where username = @username"
...
MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20))
MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value
// C# example
String selectCmd = "select * from Authors where state = @username";
SqlConnection myConnection = new SqlConnection("server=...");
SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection);
myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20));
myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value;
```

[2] You can add input validation to Web Forms pages by using validation controls. Validation controls provide an easy-to-

use mechanism for all common types of standard validation - for example, testing for valid dates or values within a range - plus ways to provide custom-written validation. In addition, validation controls allow you to completely customize how error information is displayed to the user. Validation controls can be used with any controls that are processed in a Web Forms page's class file, including both HTML and Web server controls.

In order to make sure user input contains only valid values, you can use one of the following validation controls:

- a. "RangeValidator": checks that a user's entry (value) is between specified lower and upper boundaries. You can check ranges within pairs of numbers, alphabetic characters, and dates.
- b. "RegularExpressionValidator": checks that the entry matches a pattern defined by a regular expression. This type of validation allows you to check for predictable sequences of characters, such as those in social security numbers, e-mail addresses, telephone numbers, postal codes, and so on.

Important note: validation controls do not block user input or change the flow of page processing; they only set an error state, and produce error messages. It is the programmer's responsibility to test the state of the controls in the code before performing further application-specific actions.

There are two ways to check for user input validity:

1. Testing for a general error state:

In your code, test the page's IsValid property. This property rolls up the values of the IsValid properties of all the validation controls on the page (using a logical AND). If one of the validation controls is set to invalid, the page's property will return false.

2. Testing for the error state of individual controls:

Loop through the page's Validators collection, which contains references to all the validation controls. You can then examine the IsValid property of each validation control.

PHP Fix Recommendation

** Filter User Input

Before passing any data to a SQL query, it should always be properly filtered with whitelisting techniques. This cannot be over-emphasized. Filtering user input will correct many injection flaws before they arrive at the database.

** Quote User Input

Regardless of data type, it is always a good idea to place single quotes around all user data if this is permitted by the database. MySQL allows this formatting technique.

** Escape the Data Values

If you're using MySQL 4.3.0 or newer, you should escape all strings with `mysql_real_escape_string()`. If you are using an older version of MySQL, you should use the `mysql_escape_string()` function. If you are not using MySQL, you might choose to use the specific escaping function for your particular database. If you are not aware of an escaping function, you might choose to utilize a more generic escaping function such as `addslashes()`.

If you're using the PEAR DB database abstraction layer, you can use the `DB::quote()` method or use a query placeholder like `?`, which automatically escapes the value that replaces the placeholder.

REFERENCES

http://ca3.php.net/mysql_real_escape_string

```
<a href="http://ca.php.net/mysql_escape_string">http://ca.php.net/mysql_escape_string</a>
<a href="http://ca.php.net/addslashes">http://ca.php.net/addslashes</a>
<a href="http://pear.php.net/package-info.php?package=DB">http://pear.php.net/package-info.php?package=DB</a>
```

** Input Data Validation:

While data validations may be provided as a user convenience on the client-tier, data validation must always be performed on the server-tier. Client-side validations are inherently insecure because they can be easily bypassed, e.g. by disabling Javascript.

A good design usually requires the web application framework to provide server-side utility routines to validate the following:

- [1] Required field
- [2] Field data type (all HTTP request parameters are Strings by default)
- [3] Field length
- [4] Field range
- [5] Field options
- [6] Field pattern
- [7] Cookie values
- [8] HTTP Response

A good practice is to implement a function or functions that validates each application parameter. The following sections describe some example checking.

[1] Required field

Always check that the field is not null and its length is greater than zero, excluding leading and trailing white spaces.

Example of how to validate required fields:

```
// PHP example to validate required fields
function validateRequired($input) {
    ...
    $pass = false;
    if (strlen(trim($input))>0){
        $pass = true;
    }
    return $pass;
    ...
}
...
if (validateRequired($fieldName)) {
    // fieldName is valid, continue processing request
    ...
}
```

[2] Field data type

In web applications, input parameters are poorly typed. For example, all HTTP request parameters or cookie values are

of type String. The developer is responsible for verifying the input is of the correct data type.

[3] Field length

Always ensure that the input parameter (whether HTTP request parameter or cookie value) is bounded by a minimum length and/or a maximum length.

[4] Field range

Always ensure that the input parameter is within a range as defined by the functional requirements.

[5] Field options

Often, the web application presents the user with a set of options to choose from, e.g. using the SELECT HTML tag, but fails to perform server-side validation to ensure that the selected value is one of the allowed options. Remember that a malicious user can easily modify any option value. Always validate the selected user value against the allowed options as defined by the functional requirements.

[6] Field pattern

Always check that user input matches a pattern as defined by the functionality requirements. For example, if the userName field should only allow alpha-numeric characters, case insensitive, then use the following regular expression:
`^[a-zA-Z0-9]+$`

[7] Cookie value

The same validation rules (described above) apply to cookie values depending on the application requirements, e.g. validate a required value, validate length, etc.

[8] HTTP Response

[8-1] Filter user input

To guard the application against cross-site scripting, the developer should sanitize HTML by converting sensitive characters to their corresponding character entities. These are the HTML sensitive characters:
< > " ' % ;) (& +

PHP includes some automatic sanitation utility functions, such as `htmlspecialchars()`:

```
$input = htmlspecialchars($input, ENT_QUOTES, 'UTF-8');
```

In addition, in order to avoid UTF-7 variants of Cross-site Scripting, you should explicitly define the Content-Type header of the response, for example:

```
&lt;?php  
header('Content-Type: text/html; charset=UTF-8');  
?&gt;
```

[8-2] Secure the cookie

When storing sensitive data in a cookie and transporting it over SSL, make sure that you first set the secure flag of the cookie in the HTTP response. This will instruct the browser to only use that cookie over SSL connections.

You can use the following code example, for securing the cookie:

```
&lt;$php
$value = &quot;some_value&quot;;
$time = time()+3600;
$path = &quot;/application/&quot;;
$domain = &quot;.example.com&quot;;
$secure = 1;
setcookie(&quot;CookieName&quot;, $value, $time, $path, $domain, $secure, TRUE);
?&gt;
```

In addition, we recommend that you use the HttpOnly flag. When the HttpOnly flag is set to TRUE the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectly help to reduce identity theft through XSS attacks (although it is not supported by all browsers).

The HttpOnly flag was Added in PHP 5.2.0.

REFERENCES

- [1] Mitigating Cross-site Scripting With HTTP-only Cookies: http://msdn2.microsoft.com/en-us/library/ms533046.aspx
- [2] PHP Security Consortium: http://phpsec.org/
- [3] PHP & Web Application Security Blog (Chris Shiflett): http://shiflett.org/

References and Relevant Links

<http://www.blackhat.com/presentations/win-usA-01/Litchfield/BHWin01Litchfield.doc>

Test Script Detected

Generic

WASC Threat Classification

Information Disclosure: Predictable Resource Location

http://www.webappsec.org/projects/threat/classes/predictable_resource_location.shtml

CVE Reference

Security Risks

It is possible to gather sensitive debugging information

Possible Cause

Temporary files were left in production environment

Technical Description

Sometimes developers forget to remove certain debugging or test pages from production environments. These pages may include sensitive information that should not be accessed by web users.

General Fix Recommendations

.NET Fix Recommendation

PHP Fix Recommendation

References and Relevant Links

Remediation Tasks

URL	Remediation Tasks	Addressed Security Issues
http://hackme.webscanservice.com/ (13)	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch
	Filter out hazardous characters from user input (High)	Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

Filter out hazardous characters from user input (High)

Parameter: txtSearch

<http://hackme.webscanservice.com/bank/login.aspx> (13)

Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)

Parameter: uid

Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)

Parameter: uid

Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)	Parameter: passw
Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)	Parameter: passw
Filter out hazardous characters from user input (High)	Parameter: passw
Filter out hazardous characters from user input (High)	Parameter: uid
Filter out hazardous characters from user input (High)	Parameter: uid
Remove html comments (Info)	-
Filter out hazardous characters from user input (High)	Parameter: uid
Filter out hazardous characters from user input (High)	Parameter: passw

Filter out hazardous characters from user input (High)

Parameter: passw

Filter out hazardous characters from user input (High)

Parameter: passw

Filter out hazardous characters from user input (High)

Parameter: passw

<http://hackme.webscanservice.com/feedback.aspx> (14)

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

Filter out hazardous characters from user input (High)

Parameter: name

<http://hackme.webscanservice.com/bank/> (1)

Modify the server configuration to deny directory listing, and install the latest security patches available (Medium) -

<http://hackme.webscanservice.com/bank/account.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/apply.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/main.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/mozxpath.js> (1)

Remove email address (Info) -

<http://hackme.webscanservice.com/bank/transaction.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/transfer.aspx> (1)

Remove html comments (Info) -

Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)	Parameter: txtEmail
Verify that parameter values are in their expected ranges and types. Do not output debugging error messages and exceptions (Low)	Parameter: txtEmail
Filter out hazardous characters from user input (High)	Parameter: txtEmail
Filter out hazardous characters from user input (High)	Parameter: txtEmail
Filter out hazardous characters from user input (High)	Parameter: txtEmail
Filter out hazardous characters from user input (High)	Parameter: txtEmail
Filter out hazardous characters from user input (High)	Parameter: txtEmail

Filter out hazardous characters from user input (High)

Parameter: txtEmail

Filter out hazardous characters from user input (High)

Parameter: txtEmail

<http://hackme.webscanservice.com/bank/> (1)

Modify the server configuration to deny directory listing, and install the latest security patches available (Medium) -

<http://hackme.webscanservice.com/bank/account.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/apply.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/login.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/main.aspx> (1)

Remove html comments (Info) -

<http://hackme.webscanservice.com/bank/mozxpath.js> (1)

Remove email address (Info) -

<http://hackme.webscanservice.com/bank/transaction.aspx> (1)

Remove html comments (Info)

-

<http://hackme.webscanservice.com/bank/transfer.aspx> (1)

Remove html comments (Info)

-

<http://hackme.webscanservice.com/test.aspx> (1)

Remove file (Info)

Test Script Detected