NSFOCUS.CLOUD

NSFOCUS Cloud Web Vulnerability Scanning System

Vulnerability Scanning Report

Contacting us

Thank you for choosing NSFOCUS CLOUD SECURITY SERVICE. If you have any questions or suggestions about this report, please contact us at

Email: support@nsfocusglobal.com
Phone:
USA: +1-844-673-6287 or +1-844-NSFOCUS
UK: +44 808 164 0673 or +44 808 164 0NSF
Australia: +61 2 8599 0673 or +61 2 8599 0NSF
Netherlands: +31 85 208 2673 or +31 85 208 2NSF
Brazil: +55 13 4042 1673 or +55 13 4042 1NSF
Japan: +81 3-4510-8673 or +81 3-4510-8NSF
Singapore: +65 3158 3757
Hong Kong: +852 5803 2673 or +852 5803 2NSF
Middle East: +973 1619 7607

Notices

Applicability: This report presents assessment results for monitoring and diagnosis. However, the results may not be valid after 1 year or subsequent scans.
Validity Period of Data: NSFOCUS will keep the data and files involved in this report for three months after submission.
Confidentiality: This report contains security information about the web application scanned. The information herein should be maintained per the organization's security policies and any relevant compliance requirements.
Trademark: NSFOCUS and NSFOCUS Cloud are trademarks of NSFOCUS Technologies, Inc.

1 Summary

1.1 Website Overview

Scanned Website

http://demo.testfire.net/

Generation Date

2017-08-29 SGT (+0800)

Total Scanned Pages

79

Overall Security Score

20 Your website is very vulnerable on the whole.

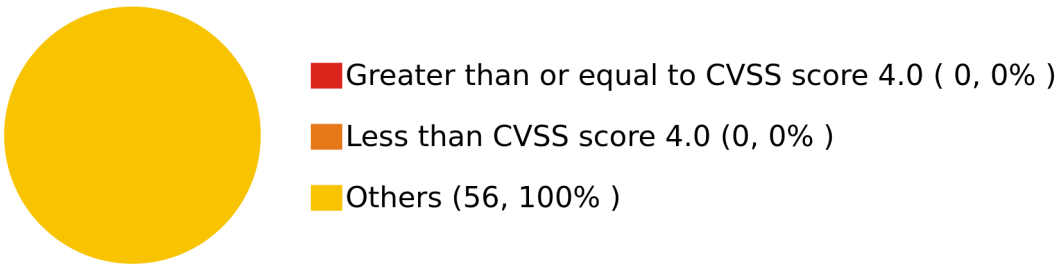
Total Discovered Vulnerabilities

56

High	15
Medium	2
Low	39

1.2 Web Vulnerabilities

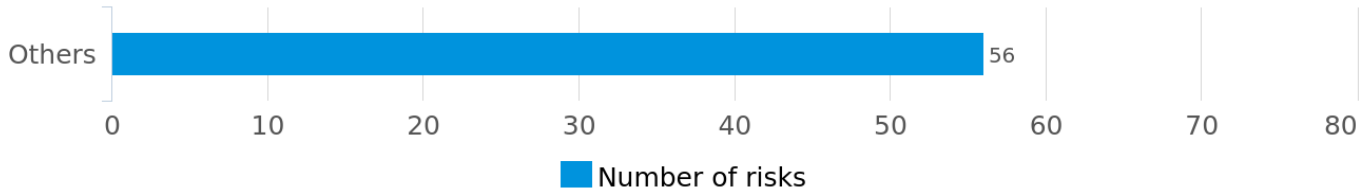
1.2.1 Statistics by CVSS Score



CVSS Score	Number of Vulnerabilities	Percentage
Greater than or equal to CVSS score 4.0	0	0.0%
Less than CVSS score 4.0	0	0.0%

Others	56	100.0%

1.2.2 Statistics by OWASP 2013 Top 10

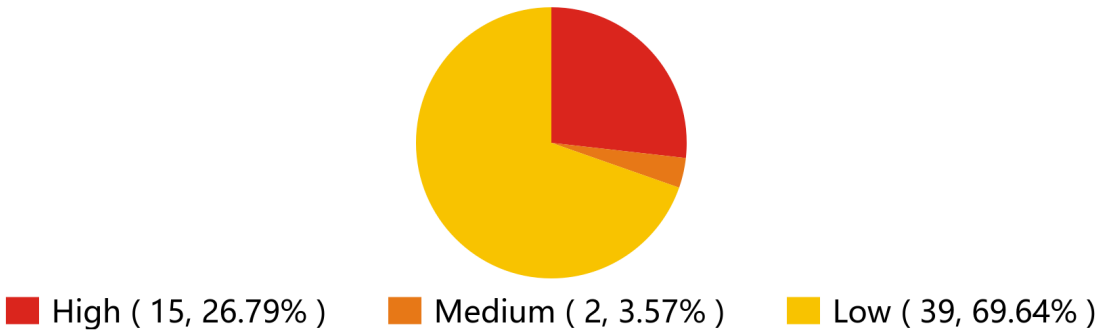


Category	Total	Percentage
Others	56	100.0%

1.2.3 Top 10 URLs with CVSS

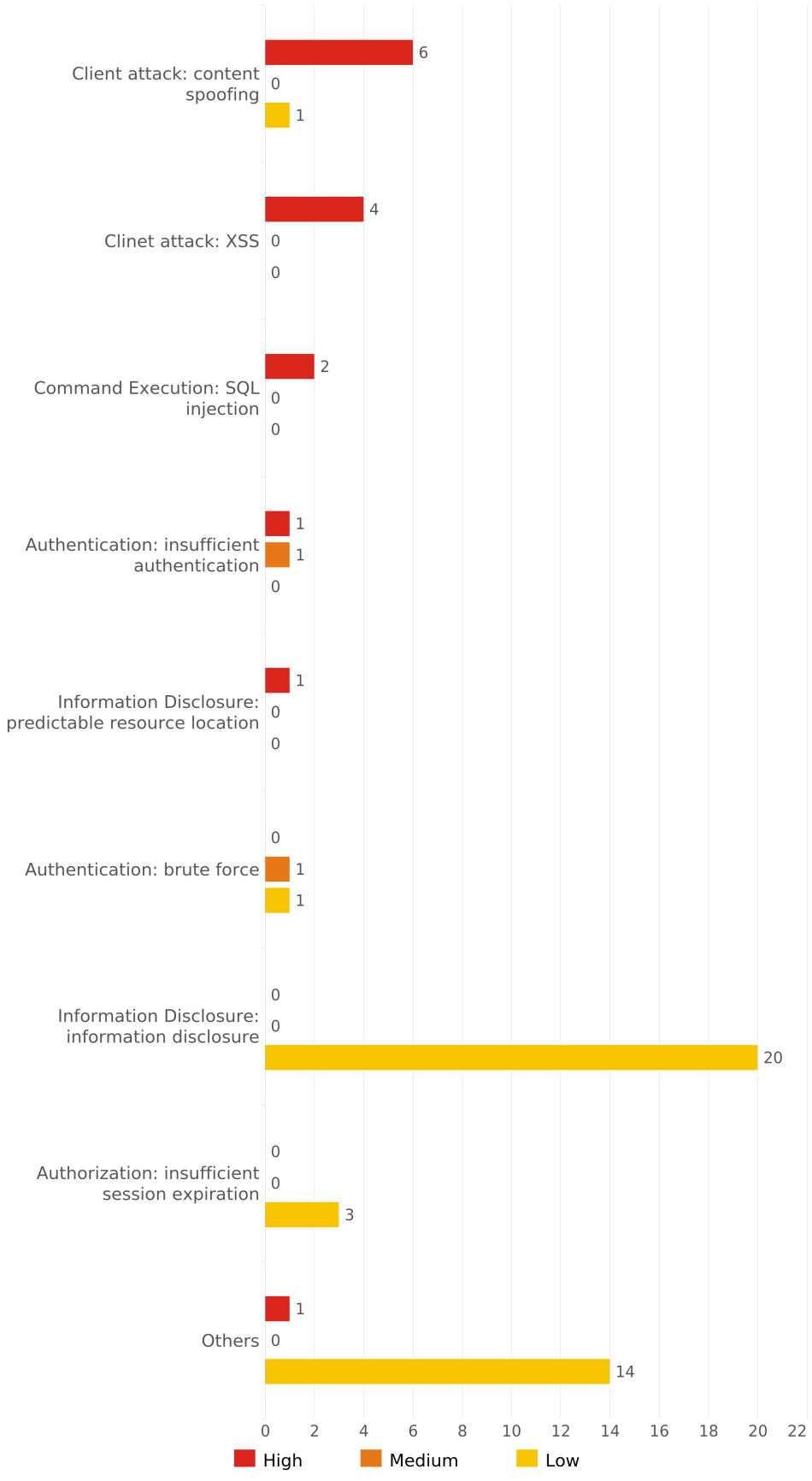
There is no data to display.

1.2.3 Statistics by Risk Level



Risk Level	Number of Vulnerabilities	Percentage
High	15	26.79%
Medium	2	3.57%
Low	39	69.64%

1.2.4 Statistics by Vulnerability Type



Vulnerability Type	High	Medium	Low	Total	Percentage
Client attack: content spoofing	6	0	1	7	12.5%
Clinet attack: XSS	4	0	0	4	7.14%
Command Execution: SQL injection	2	0	0	2	3.57%
Authentication: insufficient authentication	1	1	0	2	3.57%
Information Disclosure: predictable resource location	1	0	0	1	1.79%
Authentication: brute force	0	1	1	2	3.57%
Information Disclosure: information disclosure	0	0	20	20	35.71%
Authorization: insufficient session expiration	0	0	3	3	5.36%
Others	1	0	14	15	26.79%

1.2.5 Top 10 URLs with High-Risk Vulnerabilities

URL	Number of Vulnerabilities	Percentage
http://demo.testfire.net/comment.aspx	6	40.0%
http://demo.testfire.net/	3	20.0%
http://demo.testfire.net/bank/login.aspx	3	20.0%
http://demo.testfire.net/search.aspx?txtSearch=1	3	20.0%

2 Details of Web Vulnerabilities

2.1 High-Risk Vulnerabilities

SSL/TLS Bar Mitzvah Attack Vulnerability (High-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/
Verify Message	IP:65.61.137.117 ; PORT:443 Cipher Suites:TLS_RSA_WITH_RC4_128_SHA
Request Method	GET

Vulnerability Type

Others

Detailed Description

This vulnerability is caused by the weak and outdated RC4 cipher which reveals cipher texts in SSL/TLS-encrypted traffic, giving away the user name, password, credit card data, and other sensitive information to hackers.

Solution

1. Prohibit the use of the RC4 cipher algorithm on the server.
2. Prohibit the use of the RC4 cipher algorithm in the TLS configuration of the browser on the client.

IIS Short Filename Disclosure Vulnerability (High-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/
Verify Message	(valid)http://demo.testfire.net/*~1*\a.aspx(invalid)http://demo.testfire.net/1234567890*~1*\a.aspx
Request Method	GET

Vulnerability Type

Information Disclosure: predictable resource location

Detailed Description

Internet Information Services (IIS) is a set of Internet-based services developed by Microsoft for servers using Microsoft Windows.
Microsoft IIS is prone to a file enumeration vulnerability that allows an attacker to enumerate files in the root directory of the network server.
An attacker could exploit this vulnerability to launch a denial-of-service attack against .Net Framework in the IIS server by using the tilde (~) to guess or traverse filenames in the server.

Solution

1. Disable NTFS 8.3 file compatibility. This function is enabled by default. However, it is unnecessary for most users to enable this function.
2. For users of virtual host space, apply the following solutions:
1) Change the following registry key to 1: HKLMSYSTEMCurrentControlSetControlFileSystemNtfsDisable8dot3NameCreation. This change can prevent creation of filenames in NTFS8.3 format, while existing short filenames cannot be removed.
2) If asp.net is not required in your web environment, you choose "IIS Manager > Web Service Extensions > ASP.NET" and disable this function.
3) Upgrade Microsoft .NET Framework to 4.0 or later.
3. Copy the content in the web folder to another directory, for example, from D:www to D:www.back. Then delete the original folder and rename the new directory as the original one. Only after you have completed the copy operation, the short filenames will disappear.
For users of virtual host space, if this problem persists, contact the space provider.

Frame Injection Vulnerability Detected in Target URL (High-Risk)

Affected URLs

Number of Affected URLs: 3

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&reset=Clear+Form&name=13800138000#*/-->'");></script></style></title></textarea></iframe><IFRAME +SRC="http://www.rfueasnm.com/ynopw.html">&email_addr=atestu@example.com&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&name=13800138000#*/-->'");></script></style></title></textarea></iframe><IFRAME+SRC="http://www.nbwrybjs.com/vbwqm.html">&email_addr=atestu@example.com&submit=Submit&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/search.aspx?txtSearch=1
Verify Message	http://demo.testfire.net/search.aspx?txtSearch=1#*/-->'");></script></style></title></textarea></iframe><IFRAME+SRC="http://www.aymanigi.com/akrig.html">
Request Method	GET

Vulnerability Type

Client attack: content spoofing

Detailed Description

An attacker could inject frame or iframe tags. Users, once browsing such tags, will be directed from the original website to a malicious website, without knowing it. Attackers could entice such users to log in to the malicious website again so as to obtain login credentials of such users.

Solution

Filter out the following characters:
[1] | (pipe)
[2] &
[3]; (semicolon)
[4] $ (dollar sign)
[5] % (percent sign)
[6] @

[7] ' (single quotation mark)
[8] " (double quotation mark)
[9] ' (backslash escape single quotation mark)
[10] \" (backslash escape double quotation mark)
[11] <> (angle brackets)
[12] () (brackets)
[13] + (plus sign)
[14] CR (carriage return, ASCII 0x0d)
[15] LF (line feed, ASCII 0x0a)
[16] , (colon)
[17] (backslash)

Link Injection Vulnerability Detected in Target URL (High-Risk)

Affected URLs

Number of Affected URLs: 3

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&reset=Clear+Form&name=13800138000#*/-->'");></iframe></script></style></title></textarea>LinkInjTest&email_addr=atestu@example.com&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&name=13800138000#*/-->'");></iframe></script></style></title></textarea>LinkInjTest&email_addr=atestu@example.com&submit=Submit&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/search.aspx?txtSearch=1
Verify Message	http://demo.testfire.net/search.aspx?txtSearch=1#*/-->'");></iframe></script></style></title></textarea>LinkInjTest
Request Method	GET

Vulnerability Type

Client attack: content spoofing

Detailed Description

Link injection is a way of altering the content on a vulnerable website by embedding the content from an external website or a URL in scripts on vulnerable websites. Embedding a malicious URL in a vulnerable website, an attacker could attack this vulnerable website, or use it as a platform for attack against other websites.
These potential attacks sometimes need users' login to websites. Attacks against the vulnerable website have a greater chance of success. This is because users are more likely to log in to this website.
The link injection vulnerability is caused due to lack of sufficient sanitization of user input. The sanitization result will be returned to the user in subsequent site responses. This allows attackers to embed malicious URLs or other potential crafted contents via advance injection of dangerous characters.

Solution

Filter out the following characters:
[1] | (pipe)
[2] &
[3]; (semicolon)
[4] $ (dollar sign)
[5] % (percent sign)
[6] @
[7] ' (single quotation mark)
[8] " (double quotation mark)
[9] ' (backslash escape single quotation mark)
[10] \" (backslash escape double quotation mark)
[11] <> (angle brackets)
[12] () (brackets)
[13] + (plus sign)
[14] CR (carriage return, ASCII 0x0d)
[15] LF (line feed, ASCII 0x0a)
[16] , (colon)
[17] (backslash)

SQL Injection Vulnerability Detected on Target URL (High-Risk)

Affected URLs

Number of Affected URLs: 2

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu&uid=atestuseru'
Request Method	POST

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu'&uid=atestuseru
Request Method	POST

Vulnerability Type

Command Execution: SQL injection

Detailed Description

SQL vulnerability exists in many Web applications. SQL injection is a type of attack that takes advantage of code defects and can be exploited in Web application parameters (such as URL parameters, post data or cookie values) that affect database query.

SQL injection attacks usually are usually caused by information obtained from error messages. However, applications can be vulnerable to SQL injection even no error message is displayed.

In general, SQL injection is an attack on Web applications rather than Web servers or operating systems. As its name indicated, SQL injection embeds unexpected SQL command into the query to cause database administrators or developers to manipulate the database through unexpected means. If the above operation succeeds, attackers can obtain, modify, embed, or delete data about the database server that is embedded with the vulnerable Web application. In certain circumstances, SQL injections can be exploited to take full control of the system.

Solution

Precautions include deploying hierarchical security measures (e.g. using parameterized query while accepting user input), ensuring that the application uses expected data, and consolidating the database server against inappropriate access data.

The following measures are recommended to fend off SQL injection attacks:

For developers

========

Compile Web applications that are not vulnerable to SQL injection attacks.

Parameterized query: SQL injection arises from the situation that an attacker controls the query data and modifies the query logic on purpose. The best way to fight against SQL injection is to separate queried logic from other data, to avoid the execution of the command injected from user input.

The disadvantages of this precaution are that it may affect the system performance (slightly), and to make the precaution fully effective, every query on this website needs to be constructed. SQL injection can be implemented even if only one query is lost. The following code shows the SQL statement that allows SQL injection:

sSql = "SELECT LocationName FROM Locations "; sSql = sSql + " WHERE LocationID = " + Request["LocationID"]; oCmd.CommandText = sSql;

But the following statement uses parameterized query and thus is not vulnerable to SQL injection:

sSql = "SELECT * FROM Locations ";

sSql = sSql + " WHERE LocationID = @LocationID"; oCmd.CommandText = sSql; oCmd.Parameters.Add("@LocationID", Request["LocationID"]);

Applications send SQL statements to the server without user input, but replace the user input with the -@LocationID- parameter. As a result, user input cannot become command executed by SQL. This can deny all input from the attacker. Although errors may occur, these are just data translation errors and cannot be exploited by attackers.

The following example shows that the code allows an attacker to obtain product ID from HTTP query string and then uses the ID in SQL query. Please note that the string that contains the SELECT string and transfers to SqlComamnd is just a static string. It is not intercepted from the input. In addition, please pay attention to the way input parameters are transferred through the SqlComamnd object. The object’ s name (@pid) matches with the name used in SQL query.

C# example:

```
string connString = WebConfigurationManager.ConnectionStrings["myConn"].ConnectionString;
using (SqlConnection conn = new SqlConnection(connString))
{
conn.Open();
SqlCommand cmd = new SqlCommand("SELECT Count(*) FROM Products WHERE ProdID=@pid", conn);
SqlParameter prm = new SqlParameter("@pid", SqlDbType.VarChar, 50);
prm.Value = Request.QueryString["pid"];
cmd.Parameters.Add(prm);
int recCount = (int)cmd.ExecuteScalar();
}
```

VB.NET example:

```
Dim connString As String = WebConfigurationManager.ConnectionStrings ("myConn").ConnectionString
Using conn As New SqlConnection(connString) conn.Open()
Dim cmd As SqlCommand = New SqlCommand("SELECT Count(*) FROM Products WHERE ProdID=@pid", conn)
Dim prm As SqlParameter = New SqlParameter("@pid", SqlDbType.VarChar, 50)
prm.Value = Request.QueryString("pid")
cmd.Parameters.Add(prm)
Dim recCount As Integer = cmd.ExecuteScalar()
End Using
```

Verified input: most SQL injection attacks can be avoided through correct validation of the type and format of the user input. But the best way is through whitelist, the definition of which is to accept specific user accounts or account type for a certain field, or to accept alphabet or integers for others.

Many developers tried to validate input through blacklist or escape characters. Developers add an escape character (e.g. single quotation marks) before the known malicious data to avoid it. The following values can be then used as text values. This method is less effective than whitelist, because not all forms of malicious data are known beforehand.

For security operators:

==========

Use the following suggestions to avoid SQL injection attacks on Web applications:

Restrict application privilege: restrict user credentials to allow required privileges only. All successful SQL injection attacks run in the environment of user credentials. Although privilege restriction cannot avoid SQL injection completely, triggering this kind of attack would be more difficult.

Strong administrator password: an attacker has access to certain SQL command only if the attacker has an administrator’ s account. The attacker can perform brute force on administrator’ s weak password and increase the possibility of a successful SQL injection attack. An attacker can also create a new account for certain purpose.

Consistent error message scheme: make sure less information is provided for users when an error occurs in the database. Do not expose the entire error message. Use internet and application server to handle the error message at the same time. If a web server faults, respond the user on a general web page, or redirect the user to the standard position. Do not leak debugging information or other details that may help an attacker.

As to how to close detailed error messages at IIS, please visit the following URL:

http://www.microsoft.com/windows2000/en/server/iis/default.asp?url= /windows2000/en/server/iis/htm/core/iierrcst.htm

Use the following syntax to delete the error message from the Apache server:

Syntax: ErrorDocument <3-digit-code>

Example: ErrorDocument 500 /webserver_errors/server_error500.txt

Application servers such as WebSphere are often installed with error messages or debugging settings. As to how to delete these error messages, please refer to the documentation of the application server.

Storage process: Delete SQL storage processes such as master..Xp_cmdshell, xp_startmail, xp_sendmail, and sp_makewebtask if they are not required.

SQL injection vulnerability arises from Web application code basically. Add policies containing regular expressions to serve as an expedient in case of emergency to detect SQL injections. Although this method cannot eliminate all potential SQL injections, it is easy to implement and requires the attacker to alter its attack method. You can use the regular expression as follows:

Delete the regular expression from the SQL meta character:

/(\%27)|(')|(\-\-)|(\%23)|(#)/ix

Add the regular expression to the Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"SQL Injection-Paranoid";flow:to_server,established;uricontent:".pl";pcre:"/(\%27)|(')|(\-\-)|(%23)|(#)/i"; classtype:Web-application-attack; sid:9099; rev:5;)

Regular expression of a traditional SQL injection attack:

/\w*((\%27)|('))((\%6F)|o|(\%4F))((\%72)|r|(\%52))/ix

Delete the regular expression of a SQL injection that contains the keyword UNION:

/((\%27)|('))union/ix

(\%27)|(')

Compile a similar regular expression for other SQL queries (e.g. select, insert, update, delete and drop).

Detect the regular expression of the SQL injection attack on the MS SQL server:

/exec(\s|\+)+(s|x)p\w+/ix

For quality assurance (QA):

==========

Resolving SQL injection vulnerability requires the modification on the code. The above two parts describe information necessary for remediating this vulnerability. The following procedure introduces how to test the application manually for SQL injections.

Check the application manually for SQL injection vulnerability:

1. Open the Web application in the browser to test SQL injection vulnerabilities on it.

2. Hover your mouse over the link of the website and pay attention to the URL directed to the link on the status bar on bottom of the page. Find the URL with parameters, such as http://www.site.com/articleid.asp?id=42.

Note: If no URL is found in the status bar, click the link directly to find the URL with parameters in the address bar.

3. Click the link to enter the webpage. You can see the URL in the address bar that you previously saw in the status bar.

4. Test each parameter respectively through two methods:

Method 1. Click on the address bar to highlight the parameter value, for example "value" in "name=value", and replace the value with a single quotation mark ('). The formula should be like this "name='".

Method 2. Click on the address bar and type a single quotation mark in the middle of the value. For example, the formula "name=val'ue".

5. Click on the Go button to send the request to the Web server.

6. Analyze the error message in the response of the Web server. Error messages of most databases are as follows:

Example error 1:

Microsoft OLE DB Provider for SQL Server error '80040e14'

Unclosed quotation mark before the character string '51 ORDER BY some_name'. /some_directory/some_file.asp, line 5

Example error 2:

ODBC Error Code = S1000 (General error)

[Oracle][ODBC][Ora]ORA-00933: SQL command not properly ended

Example error 3:

Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

Message: View's SELECT and view's field list have different column counts

7. Sometimes, the error message is hidden in the source code of a page, and you need to view the HTML source code of the page and search for errors. To perform this operation on Internet Explorer, click the "View" menu, and choose "Source code" option to open a text file displaying the HTML source code. In the text file, click the Edit menu and select "Find" to display a dialog box named "Find Content". Type "Microsoft OLE DB" or [ODBC] and then click "Next".

8. If step 6 or 7 succeeds, the website contains a SQL injection vulnerability.

XSS Vulnerability Detected On Target URL (High-Risk)

Affected URLs

Number of Affected URLs: 4

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&reset=Clear+Form&name=<ScRipt>izjfzv(6123);</ScRipt>&email_addr=atestu@example.com&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)cfile=comments.txt&name=<ScRipt>izjfzv(6823);</ScRipt>&email_addr=atestu@example.com&submit=Submit&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu&uid=" onmouseover=izjfzv(6584)
Request Method	POST

URL	http://demo.testfire.net/search.aspx?txtSearch=1
Verify Message	http://demo.testfire.net/search.aspx?txtSearch=<ScRipt>izjfzv(6281);</ScRipt>
Request Method	GET

Vulnerability Type

Clinet attack: XSS

Detailed Description

Cross site scripting (XSS) steals information from users through exploitation of website vulnerabilities. Users often click links on the page while browsing a website, using Instant Messaging software or reading emails. Attackers can embed malicious code into the link and then steal user information or execute malicious code on the terminal user system.

Main problems caused by XSS attacks include:

Account hijacking—attackers can hijack user sessions before session cookie expires, and operate with privileges of a login user, such as issuing database query and viewing result.

Malicious script execution—users can mistakenly execute JavaScript, VBScript, ActiveX, HTML and even Flash content that are embedded into the dynamically generated web page.

Worm spread—through Ajax applications, XSS can be spread like a virus. XSS loading can automatically embed itself to a page, and then easily embed itself to the same host, without any manual refresh of the page. Therefore, XSS can send multiple requests in the complicated HTTP mode, and spread itself invisibly.

Information theft—attackers can connect users to the malicious server of the attacker through website redirection and forgery, and obtain any information a user has typed.

Denial of service—attackers use malformed requests on the website containing XSS vulnerabilities to cause the website to perform self queries again and again, leading to denial of service.

Browser redirection—on some websites that use frames, users may have already been redirected to a malicious website without notice, because the address in the address bar does not change. The reason is that not the whole page is redirected, but the JavaScript frame is executed.

Control user setting—attackers can change user settings on purpose.

Solution

Recommended measures include using secure encoding techniques, filtering user-supplied data, compiling all user-supplied data to avoid sending injection script to end users in executable format.

For developers

========

Stop XSS attacks by carefully validating all user input and properly encoding all output. Use standard ASP.NET validation control or validate the code directly with a strict template.

The output code should have done correct HTML encoding on any scriptable content before sending the content to the client. Function ttpUtility.HtmlEncode helps as in the following example:

Label2.Text = HttpUtility.HtmlEncode(input)

Take all path that may be used into consideration. For example, if the data is input by a user, stores it in the database and display it again, to make sure that it can be correctly encoded every time a search is implemented. If free text input is required (e.g. in a message board), and HTML format is also expected, allow a small list of security tags, as follows:

C# example:

```
StringBuilder sb = new StringBuilder(HttpUtility.HtmlEncode(htmlInputTxt.Text));
sb.Replace("&lt;b&gt;", "<b>");
sb.Replace("&lt;/b&gt;", "</b>");
sb.Replace("&lt;i&gt;", "<i>");
sb.Replace("&lt;/i&gt;", "</i>");
Response.Write(sb.ToString());
```

VB.NET example:

```
Dim sb As StringBuilder = New StringBuilder( _
HttpUtility.HtmlEncode(input))
sb.Replace("&lt;b&gt;", "<b>");
sb.Replace("&lt;/b&gt;", "</b>");
sb.Replace("&lt;i&gt;", "<i>");
sb.Replace("&lt;/i&gt;", "</i>");
Response.Write(sb.ToString());
```

Java example:

```
public static String HTMLEncode(String aTagFragment){ final StringBuffer result = new
StringBuffer(); final StringCharacterIterator iterator = new StringCharacterIterator
(aTagFragment);
char character = iterator.current();
while (character != StringCharacterIterator.DONE){
if (character = = '<') {
result.append("&lt;");
}
else if (character = = '>') {
result.append("&gt;");
}
else if (character = = '\"') {
result.append("&quot;");
}
else if (character = = '&') {
result.append("&amp;");
}
else {
//the char is not a special one
//add it to the result as is
result.append(character);
}
character = iterator.next();
}
return result.toString();
}
```

The following suggestions help construct Web applications that can fight against XSS attacks.

Define permitted content. Make sure web applications validate all input parameters (e.g. cookies, headers, query strings, spreadsheets, and hidden fields).

Check whether responses to POST and GET requests are expected and valid.

Delete conflicting characters, brackets and double or single quotation marks from user input by encoding the input data. This could stop sending injection scripts to end users.

Restrict data supplied by the client to letters or numerals. If a user input <script>alert('aaa') </script>, it will be reduced to scriptalert('aaa')script. If characters other than letters and numerals are required, encode the character into HTML entity before using it. Then, the characters will not be used to modify the structure of an HTML document.

Use dual authentication mechanism instead of single authentication mechanism.

Check the source of a script before using or modifying it.

Do not absolutely trust scripts from others while you are coding, whether from internet or someone familiar.

Most server-side scripting language provides embedding mode to convert the input variable to a correct unexplainable HTML. Filter all input before showing them to the client.

PHP: string htmlspecialchars (string string [, int quote_style])

ASP / ASP.NET: Server.HTMLEncode (strHTML String)

For security operators:

==========

Server side encoding indicates to send all dynamic content by encoding functions, and use code in the selected character set to replace scripting tags, which could fend off XSS attacks. The disadvantage of this encoding is that it is resource consuming and may have some negative effect on performance of some web servers.

If HTML tags are required, such as bulletin board with format tags, restrict available tags. Create a list of available tags, such as bold, italic characters or underlines. Forbid any other tags. The followings are some regular expressions that help detect XSS attacks.

Regular expression of simple XSS attack:

/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/ix

Add the above regular expression to a new Snort rule as follows:

alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS (msg:"NIICross-Site Scripting attempt"; flow:to_server,established;pcre:"/((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)/i";classtype:Web-application-attack; sid:9000; rev:5;)

Regular expression of XSS attacks:

/((\%3C)|<)[^\n]+((\%3E)|>)/I

This signature only searches for start HTML tag and corresponding hex character. Characters behind it cannot be a line break. And the one behind it is end tag or corresponding hex character. Some false negative might be generated, depending on the structure of the web application and web server. But this method can ensure that every attack is captured, even remote attacks like XSS. For public users, you can add more training classes for them and help them defend against online fraud that is exploited in account hijacking and other forms of identity theft.

For quality assurance (QA)

==========

Remediating XSS vulnerability requires remediation of the code. Procedures described above provide developers with information required to remediate these problems. The following procedure introduces the method for manually test applications against XSS attacks.

Step 1. Open arbitrary website in the browser, find the position that requires user input, such as spreadsheet or login page. Type “test” in the search box and send it to the web server.

Step 2. Find the web server that returns messages like “Your search for 'test' did not find any items” or “Invalid login test”. If the search result contains “test”, please proceed.

Step 3. To test XSS, type the <script>alert('hello')</script> string in the search box or login box and send it to the web server.

Step 4. If the prompt of the server displays “hello”, the website is vulnerable to XSS attack.

Step 5. Even if step 4 fails and no message is returned, a risk may still exist. Click the “View source code” option in the browser to view the actual HTML code on the webpage. Now find the <script> string sent to the server. If a whole <script>alert('hello')</script> text is seen in the source code, then the web server is vulnerable to XSS attack.

SSLv3 Serious Design Defect Vulnerability (CVE-2014-3566) (High-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/
Verify Message	IP:65.61.137.117 ; PORT:443
Request Method	GET

Vulnerability Type

Authentication: insufficient authentication

Detailed Description

An SSLv3 vulnerability (CVE-2014-3566) affects all implementation of SSLv3. By exploiting this vulnerability, an attacker can obtain transfer data (such as cookies) via man-in-the-middle attacks (as long as the both ends of the hijacked session use SSL 3.0).
To avoid exploitation of this vulnerability, disable SSLv3 for both the server and client.

Solution

Currently, the vendor has not provided any patches to fix this issue.
==========
Workaround:
Modify the Apache configuration file to disable SSLv3:
/etc/apache2/vhosts.d/00_default_ssl_vhost.conf
Add the following lines prior to SSLCipherSuite HIGH:!ADH:!aNULL:
SSLProtocol all -SSLv2 -SSLv3
SSLHonorCipherOrder on
Save the configuration file and restart Apache.

2.2 Medium-Risk Vulnerabilities

Detection of Weak Password on Target Web Application Form (Medium-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	('admin', 'admin')
Request Method	GET

Vulnerability Type

Authentication: brute force

Detailed Description

Attackers guess the website's user name and password by repeatedly attempting to conduct form-based login using user name and password in the dictionary.

User name and password enumeration threatens websites that have no restriction on failed login attempts.

This vulnerability is commonly seen on web applications.

Solution

1. Restrict login failures to lock the account when certain amount of login failures has reached.
2. Increase the complexity of the password.

(1) The user-registered password must meet the intended complexity requirements. For example, the password must be a combination of no less than 8 characters of the following: lowercase letters, uppercase letters, digits, and special characters ~!@#$%^&*()-+_. For users with unqualified passwords, the registration should not be allowed.

(2) Users are advised to change the default passwords occurring during installation to make them satisfy the preceding requirements.

SQL Injection Login Restriction Bypass Vulnerability on Target Website (Medium-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	("hoWlq' or 'a'='a'-- ", 'hoWlq')
Request Method	GET

Vulnerability Type

Authentication: insufficient authentication

Detailed Description

Many Web applications save username and password in databases, and check the user through database query when logging in. If insufficient filter is performed on the database query statement used for handling login requirements, SQL injection vulnerability may occur, allowing attackers to log in to the website without username and password, and finally log in as administrator.

Solution

Handle user inputs and data used for login strictly, including submitted data like hidden field.

2.3 Low-Risk Vulnerabilities

Microsoft ASP.NET Debug Enabled on Target URL (Low-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/
Verify Message	http://demo.testfire.net/
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Microsoft ASP.NET is quite vulnerable to information disclosure attacks. Attackers can send a malicious request notifying whether to support debugging.
Attackers can send malicious requests through the word "DEBUG".

Solution

To disable debugging in ASP.NET, please edit your web.config file so that it has the following property:
<compilation
debug="false"
/>

Application Error Exists on Target Server (Low-Risk)

Affected URLs

Number of Affected URLs: 3

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx(POST)reset=Clear +Form&name=13800138000&email_addr=atestu@example.com&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/comment.aspx
Verify Message	http://demo.testfire.net/comment.aspx (POST)name=13800138000&email_addr=atestu@example.com&submit=Submit&comments=atestu&subject=atestu
Request Method	POST

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw="())[]]{}}'/@^*$;#,&uid=atestuseru
Request Method	POST

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

If the attacker detects an application (such as the following examples) by forging requests containing parameters or parameter values that are not expected by the application, then the application may enter the vulnerable state. An attacker could obtain useful information from the application's response to the request, and exploit this information to identify weaknesses in the application.

For example, if the parameter field should be a string enclosed in single quotes (as in the ASP script or SQL query), then injected single quotation marks will be terminate the string stream early, and thus changing the normal flow/grammar of the script.

Another reason for disclosing important information in the error message is because of the configuration error of the scripting engine, Web server or database.

The following are a number of different variants:

[1] Exclude parameters

[2] Exclude parameter values

[3] Set parameter value to null

[4] Set parameter value to numeral overflow (+/- 99999999)
[5] Set parameter value to dangerous character, such as ' " ' \") ;
[6] Add a string to a numeral parameter value

Solution

[1] Check the networking request to check whether all the expected parameters and values exist. When the parameter is missing, send an appropriate error message, or use the default value.
[2] The application should verify that their input is composed of (decoded) valid characters. For example, input values containing empty bytes (encoded as% 00), single quotation marks, quotation marks, etc. should be rejected.
[3] Ensure that the scope and type of the value are in line with expectations. If the application expects that a specific parameter contain values in a collection, then the application should ensure that the received value actually belongs to the collection. For example, if the expected value is in the range [10-99], then the value should indeed be numbers in the range [10-99].
[4] Verify that the data belongs to the collection provided to the client.
[5] Do not generate debug error messages and anomalies in the production environment.

Email Address Model Exists in Target URL (Low-Risk)

Affected URLs

Number of Affected URLs: 4

URL	http://demo.testfire.net/robots.txt
Verify Message	test@test.com
Request Method	GET

URL	http://demo.testfire.net/index.aspx
Verify Message	test@test.com
Request Method	GET

URL	http://demo.testfire.net/bank/mozxpath.js
Verify Message	km0ti0n@gmail.com
Request Method	GET

URL	http://demo.testfire.net/default.htm
Verify Message	skipfish@example.com
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Searching Internet websites, Spambot begins to find email address to compose an address list for sending emails (spam).
Responses from one or more emails address can be exploited to send spam.
Some of the email addresses are for special use and are not accessible for others.

Solution

Delete the email address from the Website so that malicious users can not exploit it.

Permanent Cookies Detected in Target (Low-Risk)

Affected URLs

Number of Affected URLs: 3

URL	http://demo.testfire.net/bank/account.aspx
Verify Message	expires=Tue, 08-Aug-2017 11:05:26 GMT;
Request Method	GET

URL	http://demo.testfire.net/bank/transfer.aspx
Verify Message	expires=Tue, 08-Aug-2017 11:05:20 GMT;
Request Method	GET

URL	http://demo.testfire.net/bank/transaction.aspx
Verify Message	expires=Tue, 08-Aug-2017 11:05:13 GMT;
Request Method	GET

Vulnerability Type

Authorization: insufficient session expiration

Detailed Description

Permanent cookies are found in the target. This allows attackers to obtain users' session credentials in other ways. At this time, exploiting cross-site scripting (XSS) vulnerabilities to obtain cookies or hijack sessions becomes quite effective. By analysis the authentication by the application, an attacker, via malicious tricks, could permanently obtain the users' ID even if users exit applications, and take control of such user for a long time or permanently.

Solution

Don't set cookies to be permanent. Modify the validity attribute of cookies to ensure that cookies are invalid in a specified period or when sessions are closed.

Session Cookies Detected to Be Without HttpOnly Attribute (Low-Risk)

Affected URLs

Number of Affected URLs: 2

URL	http://demo.testfire.net/
Verify Message	http://demo.testfire.net/
Request Method	GET

URL	http://demo.testfire.net/bank/customize.aspx
Verify Message	http://demo.testfire.net/bank/customize.aspx
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Session cookies lack the HttpOnly attribute, which allows an attacker to obtain user cookies via applications (such as JavaScript and Applet). This causes user cookie disclosure and raises the threat of cross-site scripting attacks.

The HttpOnly attribute is an extension by Microsoft to cookies. It specifies whether cookies can be accessed via client script. Microsoft Internet Explorer 6 Service Pack 1 and later support the HttpOnly attribute of cookies.

Cookies may be stolen if the HttpOnly attribute not set to true. These stolen cookies may contain sensitive information (such as ASP.NET session ID or Forms authentication ticket) that identifies site users. An attacker could replay stolen cookies to be disguised as users or obtain sensitive information to launch cross-site scripting attacks.

If the HttpOnly attribute is set to true, the client cannot, via applications (such as JavaScript and Applet), read cookies received by the compatible browser. This will mitigate the threat of cross-site scripting.

Solution

Add the HttpOnly attribute for all session cookies.

Java example:

```
HttpServletResponse response2 = (HttpServletResponse)response;
response2.setHeader( "Set-Cookie", "name=value; HttpOnly");
```

Example:

```
HttpCookie myCookie = new HttpCookie("myCookie");
myCookie.HttpOnly = true;
Response.AppendCookie(myCookie);
```

VB.NET example:

```
Dim myCookie As HttpCookie = new HttpCookie("myCookie")
myCookie.HttpOnly = True
Response.AppendCookie(myCookie)
```

Autocomplete Attribute Detected to Be "on" for Password Type in Target Web Application Form (Low-Risk)

Affected URLs

Number of Affected URLs: 2

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx
Request Method	GET

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu&uid=atestuseru
Request Method	POST

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

For web application forms, if the autocomplete attribute of the <input> tag is not specified as "off\ the value of this attribute is the default one, "on". If the value of the autocomplete attribute is "on" for the <input> tag of the password type included in the web application form, when a user submits a new user name and password, the browser asks the user whether to save the user name and password. If the user chooses to save, the user name and password will be automatically filled in the corresponding text boxes when the web application form is displayed. In this case, attackers could obtain the plain-text password from the local browser cache, causing disclosure of sensitive user information.

Solution

Change the value of the autocomplete attribute to "off" for the <input> tag of the password type in the web application form.
Example:
(1) If the <input> tag of the password type lacks the autocomplete attribute:
<input type="password" name="password">
Add the autocomplete attribute that is set to "off":
<input type="password" name="password" "autocomplete"="off">
(2) If the autocomplete attribute is set to "on" for the <input> tag of the password type:
<input type="password" name="password" "autocomplete"="on">
Change the value of the autocomplete attribute to "off":
<input type="password" name="password" "autocomplete"="off">

Invalid Links Detected on Target Network (Low-Risk)

Affected URLs

Number of Affected URLs: 14

URL	http://demo.testfire.net/servererror.asp
Verify Message	http://demo.testfire.net/trace.axd
Request Method	

URL	http://demo.testfire.net/bank/login.asp
Verify Message	http://demo.testfire.net/bank/main.aspx
Request Method	

URL	http://demo.testfire.net/bank/queryxpath.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/main.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/apply.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/login.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/transaction.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/logout.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/transfer.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/account.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/bank.master
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/bank.master.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/customize.aspx.cs
Verify Message	http://demo.testfire.net/bank/
Request Method	

URL	http://demo.testfire.net/bank/URL=login.asp
Verify Message	http://demo.testfire.net/bank/transaction.aspx
Request Method	

Vulnerability Type

Others

Detailed Description

Invalid links on the page direct to resources that do not exist.

Solution

Clickjacking: X-Frame-Options Not Configured (Low-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/
Verify Message	http://demo.testfire.net/
Request Method	GET

Vulnerability Type

Client attack: content spoofing

Detailed Description

Clickjacking is a kind of visual deceptive means. Attackers can put a transparent and invisible iframe into a web page and trick an innocent user into clicking the transparent iframe page. By adjusting the position of the iframe page, attackers can tempt users to click some function buttons on the iframe page.

The X-Frame-Options field in the HTTP response header specifies whether or not a browser should render a page in an <iframe> tag. If X-Frame-Options is absent from the header of the response from the server, the website is vulnerable to Clickjacking attacks. By setting X-Frame-Options, you can prevent your website pages from being embedded with other pages, thereby eliminating the possibility of Clickjacking attacks.

Solution

Modify the web server configuration by adding the X-Frame-Options response header. It has three values:
1. DENY: The page can never be embedded into a <frame> or <iframe> tag.
2. SAMEORIGIN: The page can be embedded into an <iframe> or <frame> tag on the same origin as the page itself.
3. ALLOW-FROM uri: The page can be embedded into a frame on the specified origin.
For example:
The http.conf file can be configured on the Apache:
<IfModule headers_module>
Header always append X-Frame-Options "DENY"
</IfModule>

Database Error Information Disclosed (Low-Risk)

Affected URLs

Number of Affected URLs: 2

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu&uid=atestuseru'
Request Method	POST

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu'&uid=atestuseru
Request Method	POST

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Responses from the server contain error information about the database, because the web application fails to properly handle user input and database anomalies. Database error information can help attackers get to know the type of the backend database, and even the database structure, offering efficient information for further SQL injection attacks.

Solution

For developers, code in a secure way to capture anomalies instead of displaying error information. Filter the user input effectively, specify the data type, and define the maximum and minimum lengths of the data to be received.
Close the error information server of the database.

Default Application Test Files Discovered (Low-Risk)

Affected URLs

Number of Affected URLs: 2

URL	http://demo.testfire.net/test.html
Verify Message	http://demo.testfire.net/test.html
Request Method	GET

URL	http://demo.testfire.net/test.aspx
Verify Message	http://demo.testfire.net/test.aspx
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Test applications discovered on the target website. This type of file is usually left on the server when developers or network administrators test certain function of the web application. This kind of file may contain sensitive information, such as a validated session ID, username or password, etc. If an attacker obtains this sensitive information, he can further obtain other sensitive information.

Solution

Delete this kind of file or restrict access privilege to this file.

Detection of Brue-Force Attack on Forms of Target Web Application (Low-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx (POST)btnSubmit=Login&passw=atestpwdu&uid=atestuseru
Request Method	POST

Vulnerability Type

Authentication: brute force

Detailed Description

A brute-force attack is a common approach to try guesses repeatedly for the password. In such an attack, the attacker attempts login by using every combination of letters, digits, and special characters until the right one is found.
If a login page does not have any protection for such attacks, an attacker could use the exhaustive attack method to crack user passwords.

Solution

Limit the number of failed login attempts and lock the account when a user's number of login attempts reaches the specified value.

System Directory Disclosure on Target Server (Low-Risk)

Affected URLs

Number of Affected URLs: 3

URL	http://demo.testfire.net/bank/queryxpath.aspx
Verify Message	c:\downloads\AltoroMutual_v6\website\bank\bank.master.cs
Request Method	GET

URL	http://demo.testfire.net/bank/customize.aspx
Verify Message	c:\downloads\AltoroMutual_v6\website\bank\bank.master.cs
Request Method	GET

URL	http://demo.testfire.net/feedback.aspx
Verify Message	L:\backup\website\oldfiles
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

Server responses may contain system directories, such as /home, /var, or c:\ and others, which is generally because the target web application does not handle error messages properly and lead to directory disclosure.
If an attacker obtains this information, he can get the target server's directory structure, bringing convenience to the attack.

Solution

Delete the information on pages that contain directory information.
Shield error messages of the application if they contain directory information.

Login Request Based on HTTP Connection Detected (Low-Risk)

Affected URLs

Number of Affected URLs: 1

URL	http://demo.testfire.net/bank/login.aspx
Verify Message	http://demo.testfire.net/bank/login.aspx
Request Method	GET

Vulnerability Type

Information Disclosure: information disclosure

Detailed Description

The target application is detected to receive the login request from the client through HTTP connection. If the login request is not encrypted, an attacker may successfully sniff client-submitted data that generally contains username and password, leading to information disclosure.

This vulnerability is common among Web applications.

Solution

Encrypt the requests before submitting them. Send login requests through HTTPS connection.

Appendix A: Website Security Level Metrics

Security Level	Security Score Range
Very vulnerable	0<= Score <45
Vulnerable	45<= Score <61
Minimally vulnerable	61<= Score <85
No significant vulnerabilities	85<= Score <=100

Appendix B: Web Vulnerability Risk Level Metrics

Risk Level	Risk Value Range	Description
High	7 <= Value <= 10	An attacker can remotely manipulate system files, gain read/ write access to the background database, execute arbitrary commands, or cause a remote denial of service.
Medium	4 <= Value < 7	An attacker can exploit websites to attack other users, thereby reading system files or data in the background data.
Low	1 <= Value < 4	An attacker can obtain certain system or file information or impersonate a legitimate user.

Risk Value	Description
1	An attacker can remotely obtain version information of web server components.
2	The target web server provides unnecessary services.
3	An attacker can remotely access certain files outside the directory tree or read the source code of dynamic scripts of the server.
4	The issue of session management may be remotely exploited to impersonate a legitimate user.
5	An attacker can remotely exploit an affected web server to attack other users who are visiting the website.
6	An attacker can remotely read system files or data in the background database.

7	An attacker can remotely read and write system files or to manipulate the background database.
8	An attacker can impersonate a legitimate user to remotely execute commands or launch denial-of-service attacks.
9	An attacker can remotely execute commands as an administrative user (such vulnerabilities are not easily exploitable as the exploitation is restricted).
10	An attacker can remotely execute commands as an administrative user (such vulnerabilities are easily exploitable as the exploitation is unrestricted).