Webalyse

Web Application Security Report

http://demo.webalyse.nl/

Generation Date: 03/04/2014 at 09:27:34

Critical risk

Summary of discovered risks (32)

Critical	High Risk	Medium Risk	Low Risk
4	1	6	5

Vulnerabilities (16)

- Password is present in HTTP traffic unrelated to the login page
- Reflected Cross-Site Scripting (XSS) Vulnerabilities
- SQL Injection
- Browser-Specific Cross-Site Scripting (XSS)
- StaticSession ID
- Session Cookie Does Not Contain The "HTTPOnly" Attribute
- Session Cookie Does Not Contain The "secure" Attribute
- Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)
- Syntax Error Occurred
- Login Form Is Not Submitted Via HTTPS
- Slow HTTP POST vulnerability
- Cookie Does Not Contain The "HTTPOnly" Attribute
- Cookie Does Not Contain The "secure" Attribute
- Form With Potential Sensitive Content Submits Over HTTP
- Sensitive form field has not disabled autocomplete
- Unencoded characters

Information Gathered (16)

- Web Application Authentication Method
- Host Scan Time
- Flash Analysis
- Session Cookies
- Protection against Clickjacking vulnerability
- Email Addresses Collected
- Links Crawled
- File Upload Form Found
- External Links Discovered
- DNS Host Name
- Cookies Collected
- Authentication Form found
- Operating System Detected
- Scan Diagnostics
- Cookies Issued Without User Consent
- Links Discovered During User-Agent and Mobile Site Checks

Critical Risk

Password is present in HTTP traffic unrelated to the login page

URL: http://demo.webalyse.nl/users/home.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
Password is plaintext
Request:
GET http://demo.webalyse.nl/users/home.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: Matched body content.
last">
<h2>Hello ********, you got 100 Tradebuxs to spend!</h2>
Cool stuff to do:
<ul style="list-style-type:none;">
Who's got a similar name to you?
Your Uploaded Pics
Your Purchased Pics

Enter in our contest:
<object classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" width="610" height="410"
codebase="http://active.macromedia.com/flash5/

Threat

The password used for an authenticated scan was present in a request or response not directly related to the authentication process (i.e. in a location other than the login page). This demonstrates poor password handling by the web application, which should never expose a user's authentication credentials.

Impact

The web application exposes a user's password outside of the login process. A user's password should never be reused, reflected, or otherwise present in web traffic except during the authentication process.

Solution

The password should only be transferred during an authentication process. It should be stored by the web application in a hashed format using a strong hashing algorithm that includes a salt or a mechanism like PBKDF2.

A strong pseudo-random token should be used in place of the password or its hash if the value must be transferred during a transaction other than authentication.

All traffic that involves an authenticated user should use HTTPS or HTTP Strict Transport Security (HSTS).

Critical Risk

Password is present in HTTP traffic unrelated to the login page

URL: http://demo.webalyse.nl/cart/review.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php

Payloads

#1 Request:
Payload:
Password is plaintext
Request:
GET http://demo.webalyse.nl/cart/review.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: Matched body content.
elcome to your cart ********</h2>
<form action="/cart/action.php?action=delete" method="POST">
<table>
<tbody><tr>
<th>Pic name</th> <th>Sample Pic</th> <th>Price</th> <th>Delete?</th>
</tr>
<tr>
<td>Awesome Flower Pic</td> <td><img src="../upload/flowers/flowers" alt="Awesome Flower Pic" height="128" width="128"></td>
<td>26 Tradebux</td> <td><input type="checkbox" value="10" name="pics[]"> </td>
</tr>
<tr>
<td>My House!</td> <td><img src="../upload/house/My

Threat

The password used for an authenticated scan was present in a request or response not directly related to the authentication process (i.e. in a location other than the login page). This demonstrates poor password handling by the web application, which should never expose a user's authentication credentials.

Impact

The web application exposes a user's password outside of the login process. A user's password should never be reused, reflected, or otherwise present in web traffic except during the authentication process.

Solution

The password should only be transferred during an authentication process. It should be stored by the web application in a hashed format using a strong hashing algorithm that includes a salt or a mechanism like PBKDF2.

A strong pseudo-random token should be used in place of the password or its hash if the value must be transferred during a transaction other than authentication.

All traffic that involves an authenticated user should use HTTPS or HTTP Strict Transport Security (HSTS).

Critical Risk

Password is present in HTTP traffic unrelated to the login page

URL: http://demo.webalyse.nl/comments/preview_comment.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/pictures/recent.php
http://demo.webalyse.nl/pictures/view.php?picid=13

Payloads

#1 Request:
Payload:
Password is plaintext
Request:
POST http://demo.webalyse.nl/comments/preview_comment.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: Matched body content.
/view.php?userid=4">*******

<form action="/comments/delete_preview_comment.php" method="POST" style="display:inline">
<input type="hidden" name="previewid" value="310">
<input type="hidden" name="picid" value="13">
<input type="submit" value="Cancel">
</form>
<form action="/comments/add_comment.php" method="POST" style="display:inline">
<input type="hidden" name="previewid" value="310">
<input type="hidden" name="picid" value="13">
<input type="submi

Threat

The password used for an authenticated scan was present in a request or response not directly related to the authentication process (i.e. in a location other than the login page). This demonstrates poor password handling by the web application, which should never expose a user's authentication credentials.

Impact

The web application exposes a user's password outside of the login process. A user's password should never be reused, reflected, or otherwise present in web traffic except during the authentication process.

Solution

The password should only be transferred during an authentication process. It should be stored by the web application in a hashed format using a strong hashing algorithm that includes a salt or a mechanism like PBKDF2.

A strong pseudo-random token should be used in place of the password or its hash if the value must be transferred during a transaction other than authentication.

All traffic that involves an authenticated user should use HTTPS or HTTP Strict Transport Security (HSTS).

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/guestbook.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
%00<script>_q=random(@REQUESTID@)</script>
Request:
GET http://demo.webalyse.nl/guestbook.php

Request Headers:
Referer http://localhost/%00%3Cscript%3E_q%3Drandom(X150441080Y0Z)%3C%2Fscript%3E
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

th: 2

AA
 - by John
"><qss `;!--=&{()}>
 - by John
111111234
 - by

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome

message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/guestbook.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
'"><qss%20a=@REQUESTID@>
Request:
GET http://demo.webalyse.nl/guestbook.php

Request Headers:
User-Agent Mozilla'"><qss a=X150441080Y0Z>
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

" onEvent=X149337116Y2Z
 - by John
' onEvent=X149337116Y2Z
 - by John

```
<p class="comment">111111234</p>
<p> - by ' onEvent=X149337116Y1Z </p>
<p class="comment">"'><qss a=X149337116Y2Z></p>
<p> - by John </p>
<p class="comment">111111234</p>
<p> - by &amp;quot;'><qss a=X149337116Y1Z> </p>
<p class="comment"><EMBED SRC=//localhost/q.swf AllowScriptAccess=always></EMBED></p>
<p> - by John </p>
<p class="commen
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/guestbook.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:

"><qss%20a=@REQUESTID@>
Request:
GET http://demo.webalyse.nl/guestbook.php

Request Headers:
Cookie PHPSESSID=%22'%3E%3Cqss%20a%3DX150441080Y0Z%3E;
Referer http://demo.webalyse.nl/

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

```
<p class="comment">" onEvent=X149337116Y2Z </p>
<p> - by John </p>
<p class="comment">' onEvent=X149337116Y2Z </p>
<p> - by John </p>
<p class="comment">111111234</p>
<p> - by ' onEvent=X149337116Y1Z </p>
<p class="comment">"'><qss a=X149337116Y2Z></p>
<p> - by John </p>
<p class="comment">111111234</p>
<p> - by &amp;quot;'><qss a=X149337116Y1Z> </p>
<p class="comment"><EMBED SRC=//localhost/q.swf AllowScriptAccess=always></EMBED></p>
<p> - by John </p>
<p class="commen
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/pictures/search.php?query=%22'%3E%3Cqss%20a%3DX150871912Y1Z%3E&x=-

878&y=-113

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/guestbook.php

Payloads

#1 Request:
Payload:
query=%22'%3E%3Cqss%20%60%3b!--%3D%26%7b()%7d%3E&x=-878&y=-113
Request:
GET http://demo.webalyse.nl/pictures/search.php?query=%22'%3E%3Cqss%20%60%3b!--%3D%26%7b()%7d%3E&x=-878&y=-113

Request Headers:
Referer http://demo.webalyse.nl/

#1 Response

```
e.gif" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '"><qss `;!--=&{()}>'</h2>

<div class="column prepend-1 span-21 first last" style="margin-bottom: 2em;">
<h3 class="error">No pictures here...</h3>


</ul>
</div>

</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li><a hre
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: SQL Injection

SQL Injection

URL: http://demo.webalyse.nl/users/login.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
username=John%22'%3E%3Cqss%3E&password=password
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:

#1 Response
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '><qss>' and `password` = SHA1(CONCAT('password', `salt`)) limit 1' at line 1

Threat

SQL injection enables an attacker to modify the syntax of a SQL query in order to retrieve, corrupt or delete data. This is accomplished by manipulating query criteria in a manner that affects the query's logic. The typical causes of this vulnerability are lack of input validation and insecure construction of the SQL query.

Queries created by concatenating strings with SQL syntax and user-supplied data are prone to this vulnerability. If any part of the string concatenation can be modified, then the meaning of the query can be changed.

Examples:

These two lines demonstrate an insecure query that is created by appending the user-supplied data (**userid**):

```
dim strQuery as String

strQuery = "SELECT name,email FROM users WHERE userid=" + Request.QueryString("userid")
```

If no checks are performed against the **userid** parameter, then the query may be arbitrarily modified as

shown in these two examples of a completed query:

```
SELECT name,email FROM users WHERE userid=42
```

```
SELECT name,email FROM users WHERE userid=42; SHUTDOWN WITH NOWAIT
```

Impact

The scope of a SQL injection exploit varies greatly. If any SQL statement can be injected into the query, then the attacker has the equivalent access of a database administrator. This access could lead to theft of data, malicious corruption of data, or deletion of data.

Solution

SQL injection vulnerabilities can be addressed in three areas: input validation, query creation, and database security.

All input received from the Web client should be validated for correct content. If a value's type or content range is known beforehand, then stricter filters should be applied. For example, an email address should be in a specific format and only contain characters that make it a valid address; or numeric fields like a U.S. zip code should be limited to five digit values.

Prepared statements (sometimes referred to as parameterized statements) provide strong protection from SQL injection. Prepared statements are precompiled SQL queries whose parameters can be modified when the query is executed. Prepared statements enforce the logic of the query and will fail if the query cannot be compiled correctly. Programming languages that support prepared statements provide specific functions for creating queries. These functions are more secure than string concatenation for assigning user-supplied data to a query.

Stored procedures are precompiled queries that reside in the database. Like prepared statements, they also enforce separation of query data and logic. SQL statements that call stored procedures should not be created via string concatenation, otherwise their security benefits are negated.

SQL injection exploits can be mitigated by the use of Access Control Lists or role-based access within the database. For example, a read-only account would prevent an attacker from modifying data, but would not prevent the user from viewing unauthorized data. Table and row-based access controls potentially minimize the scope of a compromise, but they do not prevent exploits.

Example of a secure query created with a prepared statement:

```
PreparedStatement ps = "SELECT name,email FROM users WHERE userid=?";
ps.setInt(1, userid);
```

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/users/login.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required

Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
"><qss%20a=@REQUESTID@>
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:
Referer http://demo.webalyse.nl/

#1 Response
You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '><qss a=X150871988Y1Z>' and `password` = SHA1(CONCAT('scanner1', `salt`)) limit' at line 1

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

Password is present in HTTP traffic unrelated to the login page

URL: http://demo.webalyse.nl/users/view.php?userid=4

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php

Payloads

#1 Request:
Payload:
Password is plaintext
Request:
GET http://demo.webalyse.nl/users/view.php?userid=4

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: Matched body content.
st last">
<h2>******** doesn't have any pictures yet. </h2>

Home |
Admin |
Contact |
Terms of Service

</body></html>

Threat

The password used for an authenticated scan was present in a request or response not directly related to the authentication process (i.e. in a location other than the login page). This demonstrates poor password handling by the web application, which should never expose a user's authentication credentials.

Impact

The web application exposes a user's password outside of the login process. A user's password should never be reused, reflected, or otherwise present in web traffic except during the authentication process.

Solution

The password should only be transferred during an authentication process. It should be stored by the web application in a hashed format using a strong hashing algorithm that includes a salt or a mechanism like PBKDF2.

A strong pseudo-random token should be used in place of the password or its hash if the value must be transferred during a transaction other than authentication.

All traffic that involves an authenticated user should use HTTPS or HTTP Strict Transport Security (HSTS).

Browser-Specific Cross-Site Scripting (XSS)

URL: http://demo.webalyse.nl/piccheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
MAX_FILE_SIZE=30000&userfile=111111234&name=%3Cscript%20src%3Dhttp%3A%2F%2Flocalhost%2Fj%20
Request:
POST http://demo.webalyse.nl/piccheck.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response

```
te.gif" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>


<div class="column prepend-1 span-24 first last">
<h2>Checking your file <script src=http://localhost/j </h2>
<p>
File is O.K. to upload!
</p>
</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li><a href="/">Home</a> |</li>
<li><a href="/admin/index.php?page=login">Admin</a> |</li>
<li><a h
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contains characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in the HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Note! This specific test uses an XSS payload that takes advantage of Mozilla's HTML parsing engine. Manual confirmation of this vulnerability should use the Mozilla browser. Even though this exploits a particular Web browser, the Web application still has inadequate input filters.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code in the victim's Web browser. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash, and Java applets) can be used as part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/piccheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
MAX_FILE_SIZE=30000&userfile=111111234&name=%22%3E%3Cqss%3E
Request:
POST http://demo.webalyse.nl/piccheck.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response

```
ch_button_white.gif" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>


<div class="column prepend-1 span-24 first last">
<h2>Checking your file "><qss></h2>
<p>
File is O.K. to upload!
</p>
</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li><a href="/">Home</a> |</li>
<li><a href="/admin/index.php?page=login">Admin</a> |</li>
<li><a href="mailto
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

Critical Risk

WASC: Cross-Site Scripting

Reflected Cross-Site Scripting (XSS) Vulnerabilities

URL: http://demo.webalyse.nl/pictures/search.php?query=%22'%3E%3Cqss%20a%3DX150169648Y1Z%3E

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
query=%22%3E%3Cqss%3E
Request:
GET http://demo.webalyse.nl/pictures/search.php?query=%22%3E%3Cqss%3E

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
on_white.gif" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />

```
</form>
</div>
</div>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '"><qss>'</h2>

<div class="column prepend-1 span-21 first last" style="margin-bottom: 2em;">
<h3 class="error">No pictures here...</h3>


</ul>
</div>

</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li><a href="/">
```

Threat

XSS vulnerabilities occur when the Web application echoes user-supplied data in an HTML response sent to the Web browser. For example, a Web application might include the user's name as part of a welcome message or display a home address when confirming a shipping destination. If the user-supplied data contain characters that are interpreted as part of an HTML element instead of literal text, then an attacker can modify the HTML that is received by the victim's Web browser.

The XSS payload is echoed in HTML document returned by the request. An XSS payload may consist of HTML, JavaScript or other content that will be rendered by the browser. In order to exploit this vulnerability, a malicious user would need to trick a victim into visiting the URL with the XSS payload.

Impact

XSS exploits pose a significant threat to a Web application, its users and user data. XSS exploits target the users of a Web application rather than the Web application itself. An exploit can lead to theft of the user's credentials and personal or financial information. Complex exploits and attack scenarios are possible via XSS because it enables an attacker to execute dynamic code. Consequently, any capability or feature available to the Web browser (for example HTML, JavaScript, Flash and Java applets) can be used to as a part of a compromise.

Solution

Filter all data collected from the client including user-supplied content and browser content such as Referrer and User-Agent headers.

Any data collected from the client and displayed in a Web page should be HTML-encoded to ensure the content is rendered as text instead of an HTML element or JavaScript.

High Risk

StaticSession ID

URL: http://demo.webalyse.nl/users/login.php

Detection Information

Parameter: No param has been required for detecting the information

Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:

#1 Response
Static Session ID: Ineffective Session Regeneration.

Threat

The Static Session ID or ineffective session expiration vulnerability exists if the web application does not invalidate the session ID after user clicks on a log out link or closes the browser tab with out logging out. The terminated session remains valid after logout. It helps an attacker gain unauthorized access to the victims users's session using the old but valid and active session ID.

Impact

The attacker can impersonate the victim user and misuse the account. Successful exploitation of ineffective session expiration can lead to the potential identity theft. If the web application's timeouts are not set properly. A user uses a public computer to access the application and closes the browser tab instead of clicking "logout" link/button. A malicious user uses the same browser after some time. Since the application doesn't invalidate the session properly, an attacker gets an access to the active session of the previous user.

Solution

The following are few remediation steps to mitigate Static Session ID or ineffective session expiration vulnerability:
- Regenerate the session ID per log-in request and always invalidate the session after log out, at both client and server side.
For e.g. "HttpSession.invalidate()" (J2EE), "Session.Abandon()" (ASP .NET) or "session_destroy()/unset()" (PHP).
- Session active for long time must be re-authenticated.
-Web applications must provide a visible and easily accessible logout/exit/logoff button that is available on the web application menu and reachable from every web application resource and page.
It helps user close the session manually at any given time.
-Terminate the inactive session that has not been active over a reasonable time, typically 10-15 minutes.
Example:

The Timeout property can be set in the Web.config file for an application using the timeout attribute of the SessionState configuration element:

References:

https://www.owasp.org/index.php/Session_Management

https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

http://in.php.net/session_destroy

http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx

Medium Risk

Session Cookie Does Not Contain The "HTTPOnly" Attribute

URL: http://demo.webalyse.nl/

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/

Request Headers:

#1 Response
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/ First set at URL: http://demo.webalyse.nl/

Threat

The session cookie used to identify authenticated users of the Web application does not contain the "HTTPOnly" attribute.

Impact

Cookies without the "HTTPOnly" attribute are permitted to be accessed via JavaScript. Cross-site scripting attacks can steal to session cookies which could lead to user impersonation or compromise of the application account.

Solution

If the associated risk of a compromised account is high, apply the "HTTPOnly" attribute to session cookies.

Medium Risk

Session Cookie Does Not Contain The "secure" Attribute

URL: http://demo.webalyse.nl/

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/

Request Headers:

#1 Response
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/ First set at URL: http://demo.webalyse.nl/

Threat

The session cookie used to identify authenticated users of the Web application does not contain the "secure" attribute.

Impact

Cookies with the "secure" attribute are only permitted to be sent via HTTPS. Session cookies sent via HTTP expose an unsuspecting user to sniffing attacks that could lead to user impersonation or compromise of the application account.

Solution

If the associated risk of a compromised account is high, apply the "secure" attribute to session cookies and force all sensitive requests to be sent via HTTPS.

Medium Risk

WASC: Cross-Site Request Forgery

Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)

URL: http://demo.webalyse.nl/pictures/view.php?picid=13

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/pictures/recent.php

Payloads

#1 Request:
Payload:
N/A
Request:

GET http://demo.webalyse.nl/pictures/view.php?picid=13

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=pqm3nf5rj8hnvlr0bsc2n6k2u5;

#1 Response
comment: None of the form's field values change between a user's session, which increases the chances for an attacker to predict values in order to forge a request.

N/A

Threat

An effective CSRF (Cross-Site Request Forgery) countermeasure for forms is to include a hidden field with a random value specific to the user's current session. A form was detected that did not appear to contain an anti-CSRF token. This form was tested for susceptibility to a CSRF attack and determined to be vulnerable.

Impact

CSRF vulnerabilities can be used by an attacker to force a user to submit requests to the Web application without the user's knowledge or approval. The vulnerability's impact depends on the the consequence of submitting a request within the context of the Web application.

Solution

Review the description of the CSRF vulnerability and suggested countermeasures at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). All forms that affect a user's security context should be protected from CSRF attacks.

Medium Risk

WASC: Application Misconfiguration

Syntax Error Occurred

URL: http://demo.webalyse.nl/admin/index.php?page=*%2F%3B(function()%7bqxss%7d)%3B%2F*

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
/;(function()%7bqxss%7d);/
Request:
GET http://demo.webalyse.nl/admin/index.php?page=*%2F%3B(function()%7bqxss%7d)%3B%2F*

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
The HTTP response returned an empty body. This vulnerability was solely based on 5xx response code

Threat

A test payload generated a syntax error within the Web application. This often points to a problem with input validation routines or lack of filters on user-supplied content.

Impact

A malicious user may be able to create a denial of service, serious error, or exploit depending on the error encountered by the Web application.

Solution

The Web application should restrict user-supplied data to consist of a minimal set of characters necessary for the input field. Additionally, all content received from the client (i.e. Web browser) should be validated to an expected format or checked for malicious content.

Medium Risk

WASC: Cross-Site Request Forgery

Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)

URL: http://demo.webalyse.nl/users/register.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/users/register.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=pqm3nf5rj8hnvlr0bsc2n6k2u5;

#1 Response
comment: None of the form's field values change between a user's session, which increases the chances for an attacker to predict values in order to forge a request.
The form does not appear to contain a CSRF countermeasure based on a hidden form field with a pseudo-random value.

N/A

Threat

An effective CSRF (Cross-Site Request Forgery) countermeasure for forms is to include a hidden field with a random value specific to the user's current session. A form was detected that did not appear to contain an anti-CSRF token. This form was tested for susceptibility to a CSRF attack and determined to be vulnerable.

Impact

CSRF vulnerabilities can be used by an attacker to force a user to submit requests to the Web application without the user's knowledge or approval. The vulnerability's impact depends on the the consequence of submitting a request within the context of the Web application.

Solution

Review the description of the CSRF vulnerability and suggested countermeasures at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). All forms that affect a user's security context should be protected from CSRF attacks.

Medium Risk

WASC: Cross-Site Request Forgery

Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)

URL: http://demo.webalyse.nl/cart/confirm.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php
http://demo.webalyse.nl/cart/review.php

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/cart/confirm.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=pqm3nf5rj8hnvlr0bsc2n6k2u5;

#1 Response
comment: None of the form's field values change between a user's session, which increases the chances for an attacker to predict values in order to forge a request.
The form does not appear to contain a CSRF countermeasure based on a hidden form field with a pseudo-random value.

N/A

Threat

An effective CSRF (Cross-Site Request Forgery) countermeasure for forms is to include a hidden field with a random value specific to the user's current session. A form was detected that did not appear to contain an anti-CSRF token. This form was tested for susceptibility to a CSRF attack and determined to be vulnerable.

Impact

CSRF vulnerabilities can be used by an attacker to force a user to submit requests to the Web application without the user's knowledge or approval. The vulnerability's impact depends on the the consequence of submitting a request within the context of the Web application.

Solution

Review the description of the CSRF vulnerability and suggested countermeasures at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). All forms that affect a user's security context should be protected from CSRF attacks.

Medium Risk

WASC: Cross-Site Request Forgery

Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)

URL: http://demo.webalyse.nl/passcheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/register.php

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/passcheck.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=pqm3nf5rj8hnvlr0bsc2n6k2u5;

#1 Response
comment: None of the form's field values change between a user's session, which increases the chances for an attacker to predict values in order to forge a request.
The form does not appear to contain a CSRF countermeasure based on a hidden form field with a pseudo-random value.

N/A

Threat

An effective CSRF (Cross-Site Request Forgery) countermeasure for forms is to include a hidden field with a random value specific to the user's current session. A form was detected that did not appear to contain an anti-CSRF token. This form was tested for susceptibility to a CSRF attack and determined to be vulnerable.

Impact

CSRF vulnerabilities can be used by an attacker to force a user to submit requests to the Web application without the user's knowledge or approval. The vulnerability's impact depends on the the consequence of submitting a request within the context of the Web application.

Solution

Review the description of the CSRF vulnerability and suggested countermeasures at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). All forms that affect a user's security context should be protected from CSRF attacks.

Medium Risk

Login Form Is Not Submitted Via HTTPS

URL: http://demo.webalyse.nl/users/login.php

Parameter: No param has been required for detecting the information

Detection Information

Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:

#1 Response
Login Form Is Not Submitted Via HTTPS

Threat

The login form's default action contains a link that is not submitted via HTTPS (HTTP over SSL).

Impact

Sensitive data such as authentication credentials should be encrypted when transmitted over the network. Otherwise they are exposed to sniffing attacks.

Solution

Change the login form's action to submit via HTTPS.

Medium Risk

WASC: Cross-Site Request Forgery

Form Can Be Manipulated with Cross-Site Request Forgery (CSRF)

URL: http://demo.webalyse.nl/cart/review.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/cart/action.php?action=delete

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=tn3g3i5gt9drb1luo12e3c7k62;

#1 Response
comment: The form re-submission with different set of cookies is successful. This may imply that the form does not contain any CSRF countermeasures.

N/A

Threat

An effective CSRF (Cross-Site Request Forgery) countermeasure for forms is to include a hidden field with a random value specific to the user's current session. A form was detected that did not appear to contain an anti-CSRF token. This form was tested for susceptibility to a CSRF attack and determined to be vulnerable.

Impact

CSRF vulnerabilities can be used by an attacker to force a user to submit requests to the Web application without the user's knowledge or approval. The vulnerability's impact depends on the the consequence of submitting a request within the context of the Web application.

Solution

Review the description of the CSRF vulnerability and suggested countermeasures at http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF). All forms that affect a user's security context should be protected from CSRF attacks.

Medium Risk

Slow HTTP POST vulnerability

URL: http://demo.webalyse.nl/pictures/search.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:

N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/pictures/search.php

Request Headers:

#1 Response
Vulnerable to slow HTTP POST attack

Server resets timeout after accepting request data from peer.

Threat

The web application is possibly vulnerable to a "slow HTTP POST" Denial of Service (DoS) attack. This is an application-level DoS that consumes server resources by maintaining open connections for an extended period of time by slowly sending traffic to the server. If the server maintains too many connections open at once, then it may not be able to respond to new, legitimate connections. Unlike bandwidth-consumption DoS attacks, the "slow" attack does not require a large amount of traffic to be sent to the server -- only that the client is able to maintain open connections for several minutes at a time.

The attack holds server connections open by sending properly crafted HTTP POST headers that contain a Content-Length header with a large value to inform the web server how much of data to expect. After the HTTP POST headers are fully sent, the HTTP POST message body is sent at slow speeds to prolong the completion of the connection and lock up server resources. By waiting for the complete request body, the server is helping clients with slow or intermittent connections to complete requests, but is also exposing itself to abuse.

More information can be found at the in this presentation.

Impact

All other services remain intact but the web server itself becomes inaccessible.

Solution

Solution would be server-specific, but general recommendations are:
- to limit the size of the acceptable request to each form requirements
- establish minimal acceptable speed rate
- establish absolute request timeout for connection with POST request
Server-specific details can be found here.
A tool that demonstrates this vulnerability in a more intrusive manner is available here.

Low Risk

Cookie Does Not Contain The "HTTPOnly" Attribute

URL: http://demo.webalyse.nl/

Parameter: No param has been required for detecting the information

Detection Information

Authentication: Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/

Request Headers:

#1 Response
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/; domain=demo.webalyse.nl

Threat

The cookie does not contain the "HTTPOnly" attribute.

Impact

Cookies without the "HTTPOnly" attribute are permitted to be accessed via JavaScript. Cross-site scripting attacks can steal cookies which could lead to user impersonation or compromise of the application account.

Solution

If the associated risk of a compromised account is high, apply the "HTTPOnly" attribute to cookies.

Low Risk

Cookie Does Not Contain The "secure" Attribute

URL: http://demo.webalyse.nl/

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
GET http://demo.webalyse.nl/

Request Headers:

#1 Response
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/; domain=demo.webalyse.nl

Threat

The cookie does not contain the "secure" attribute.

Impact

Cookies with the "secure" attribute are only permitted to be sent via HTTPS. Session cookies sent via HTTP expose an unsuspecting user to sniffing attacks that could lead to user impersonation or compromise of the application account.

Solution

If the associated risk of a compromised account is high, apply the "secure" attribute to cookies and force all sensitive requests to be sent via HTTPS.

Low Risk

Form With Potential Sensitive Content Submits Over HTTP

URL: http://demo.webalyse.nl/comments/preview_comment.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/pictures/recent.php

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/comments/preview_comment.php

Request Headers:

#1 Response
N/A

Threat

A form has been identified as having a potential security context for the user. A security context indicates a form is only available to authenticated users, contains personal or sensitive data, or performs an action specific to the user's account.

Impact

Traffic over HTTP is unencrypted and vulnerable to sniffing attacks that can expose sensitive information about the user or the application.

Solution

Change the form's action from HTTP to HTTPS.

Low Risk

Sensitive form field has not disabled autocomplete

URL: http://demo.webalyse.nl/admin/index.php?page=login

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/admin/index.php?page=login

Request Headers:

#1 Response
Form field does not set autocomplete="off".

Threat

An HTML form that collects sensitive information (such as a password field) does not prevent the browser from prompting the user to save the populated values for late reuse. Stored credentials should not be available to anyone but their owner.

Impact

If the browser is used in a shared computing environment where more than one person may use the browser, then "autocomplete" values may be submitted by an unauthorized user. For example, if a browser saves the login name and password for a form, then anyone with access to the browser may submit the form and authenticate to the site without having to know the victim's password.

Solution

Add the following attribute to the form or input element:

autocomplete="off"

This attribute prevents the browser from prompting the user to save the populated form values for later reuse.

Low Risk

Form With Potential Sensitive Content Submits Over HTTP

URL: http://demo.webalyse.nl/users/register.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/users/register.php

Request Headers:

#1 Response
N/A

Threat

A form has been identified as having a potential security context for the user. A security context indicates a form is only available to authenticated users, contains personal or sensitive data, or performs an action specific to the user's account.

Impact

Traffic over HTTP is unencrypted and vulnerable to sniffing attacks that can expose sensitive information about the user or the application.

Solution

Change the form's action from HTTP to HTTPS.

Low Risk

Sensitive form field has not disabled autocomplete

URL: http://demo.webalyse.nl/users/register.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/users/register.php

Request Headers:

#1 Response
Form field does not set autocomplete="off".

Threat

An HTML form that collects sensitive information (such as a password field) does not prevent the browser from prompting the user to save the populated values for late reuse. Stored credentials should not be available to anyone but their owner.

Impact

If the browser is used in a shared computing environment where more than one person may use the browser, then "autocomplete" values may be submitted by an unauthorized user. For example, if a browser saves the login name and password for a form, then anyone with access to the browser may submit the form and authenticate to the site without having to know the victim's password.

Solution

Add the following attribute to the form or input element:

autocomplete="off"

This attribute prevents the browser from prompting the user to save the populated form values for later reuse.

Low Risk

Form With Potential Sensitive Content Submits Over HTTP

URL: http://demo.webalyse.nl/pictures/search.php?query=&x=-843&y=-113

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php
http://demo.webalyse.nl/cart/review.php

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/pictures/search.php?query=&x=-843&y=-113

Request Headers:

#1 Response
N/A

Threat

A form has been identified as having a potential security context for the user. A security context indicates a form is only available to authenticated users, contains personal or sensitive data, or performs an action specific to the user's account.

Impact

Traffic over HTTP is unencrypted and vulnerable to sniffing attacks that can expose sensitive information about the user or the application.

Solution

Change the form's action from HTTP to HTTPS.

Low Risk

Form With Potential Sensitive Content Submits Over HTTP

URL: http://demo.webalyse.nl/passcheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/register.php

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/passcheck.php

Request Headers:

#1 Response
N/A

Threat

A form has been identified as having a potential security context for the user. A security context indicates a form is only available to authenticated users, contains personal or sensitive data, or performs an action specific to the user's account.

Impact

Traffic over HTTP is unencrypted and vulnerable to sniffing attacks that can expose sensitive information about the user or the application.

Solution

Change the form's action from HTTP to HTTPS.

Low Risk

Sensitive form field has not disabled autocomplete

URL: http://demo.webalyse.nl/passcheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/passcheck.php

Request Headers:

#1 Response
Form field does not set autocomplete="off".

Threat

An HTML form that collects sensitive information (such as a password field) does not prevent the browser from prompting the user to save the populated values for late reuse. Stored credentials should not be available to anyone but their owner.

Impact

If the browser is used in a shared computing environment where more than one person may use the browser, then "autocomplete" values may be submitted by an unauthorized user. For example, if a browser saves the login name and password for a form, then anyone with access to the browser may submit the form and authenticate to the site without having to know the victim's password.

Solution

Add the following attribute to the form or input element:

autocomplete="off"

This attribute prevents the browser from prompting the user to save the populated form values for later reuse.

Low Risk

Sensitive form field has not disabled autocomplete

URL: http://demo.webalyse.nl/users/login.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
N/A

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:

#1 Response
Form field does not set autocomplete="off".

Threat

An HTML form that collects sensitive information (such as a password field) does not prevent the browser from prompting the user to save the populated values for late reuse. Stored credentials should not be available to anyone but their owner.

Impact

If the browser is used in a shared computing environment where more than one person may use the browser, then "autocomplete" values may be submitted by an unauthorized user. For example, if a browser saves the login name and password for a form, then anyone with access to the browser may submit the form and authenticate to the site without having to know the victim's password.

Solution

Add the following attribute to the form or input element:

autocomplete="off"

This attribute prevents the browser from prompting the user to save the populated form values for later reuse.

Low Risk

URL: http://demo.webalyse.nl/cart/action.php?action=addcoupon

Form With Potential Sensitive Content Submits Over HTTP

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/users/home.php

Payloads

#1 Request:
Payload:
N/A
Request:
POST http://demo.webalyse.nl/cart/action.php?action=delete

Request Headers:

#1 Response
N/A

Threat

A form has been identified as having a potential security context for the user. A security context indicates a form is only available to authenticated users, contains personal or sensitive data, or performs an action specific to the user's account.

Impact

Traffic over HTTP is unencrypted and vulnerable to sniffing attacks that can expose sensitive information about the user or the application.

Solution

Change the form's action from HTTP to HTTPS.

Low Risk

WASC: Improper Input Handling

Unencoded characters

URL: http://demo.webalyse.nl/pictures/search.php?query=%3C%0a%0dscript%20a%3D4%3Eqss%3D7%3C%0a%0d%2Fscript%3E&x=-878&y=-113

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Required
Access Path:
http://demo.webalyse.nl/
http://demo.webalyse.nl/guestbook.php

Payloads

#1 Request:
Payload:
query=%22'%3E%3C%3CSCRIPT%20a%3D2%3Eqss%3D7%3B%2F%2F%3C%3C%2FSCRIPT%3E&x=-878&y=-113
Request:
GET http://demo.webalyse.nl/pictures/search.php?
query=%22'%3E%3C%3CSCRIPT%20a%3D2%3Eqss%3D7%3B%2F%2F%3C%3C%2FSCRIPT%3E&x=-878&y=-113

Request Headers:
Referer http://demo.webalyse.nl/

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

```
type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '"><<SCRIPT a=2>qss=7;//<</SCRIPT>'</h2>

<div class="column prepend-1 span-21 first last" style="margin-bottom: 2em;">
<h3 class="error">No pictures here...</h3>


</ul>
</div>

</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li
```

Threat

The web application reflects potentially dangerous characters such as single quotes, double quotes, and angle brackets. These characters are commonly used for HTML injection attacks such as cross-site scripting (XSS).

Impact

No exploit was determined for these reflected characters. The input parameter should be manually analyzed to verify that no other characters can be injected that would lead to an HTML injection (XSS) vulnerability.

Solution

Review the reflected characters to ensure that they are properly handled as defined by the web application's coding practice. Typical solutions are to apply HTML encoding or percent encoding to the characters depending on where they are placed in the HTML. For example, a double quote might be encoded as " when displayed in a text node, but as %22 when placed in the value of an href attribute.

Low Risk

WASC: Improper Input Handling

Unencoded characters

URL: http://demo.webalyse.nl/users/login.php

Parameter: No param has been required for detecting the information

Detection Information

Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
username=%22'%3E%3C%3CSCRIPT%20a%3D2%3Eqss%3D7%3B%2F%2F%3C%3C%2FSCRIPT%3E&password=password
Request:
POST http://demo.webalyse.nl/users/login.php

Request Headers:

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

You have an error in your SQL syntax; check the manual that corresponds to your MariaDB server version for the right syntax to use near '><<SCRIPT a=2>qss=7;//<</SCRIPT>' and `password` = SHA1(CONCAT('password', `sal' at line 1

Threat

The web application reflects potentially dangerous characters such as single quotes, double quotes, and angle brackets. These characters are commonly used for HTML injection attacks such as cross-site scripting (XSS).

Impact

No exploit was determined for these reflected characters. The input parameter should be manually analyzed to verify that no other characters can be injected that would lead to an HTML injection (XSS) vulnerability.

Solution

Review the reflected characters to ensure that they are properly handled as defined by the web application's coding practice. Typical solutions are to apply HTML encoding or percent encoding to the characters depending on where they are placed in the HTML. For example, a double quote might be encoded as " when displayed in a text node, but as %22 when placed in the value of an href attribute.

Low Risk

WASC: Improper Input Handling

Unencoded characters

URL: http://demo.webalyse.nl/piccheck.php

Detection Information

Parameter: No param has been required for detecting the information
Authentication: Not Required
Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
MAX_FILE_SIZE=30000&userfile=111111234&name=%22'%3E%3C%3CSCRIPT%20a%3D2%3Eqss%3D7%3B%2F%2F%3C%3C%2FSCRIPT%3E
Request:
POST http://demo.webalyse.nl/piccheck.php

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

```
e.gif" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>


<div class="column prepend-1 span-24 first last">
<h2>Checking your file '"><<SCRIPT a=2>qss=7;//<</SCRIPT></h2>
<p>
File is O.K. to upload!
</p>
</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li><a href="/">Home</a> |</li>
<li><a href="/admin/index.php?page=login">Admin</a> |</li>
<li><a
```

Threat

The web application reflects potentially dangerous characters such as single quotes, double quotes, and angle brackets. These characters are commonly used for HTML injection attacks such as cross-site scripting (XSS).

Impact

No exploit was determined for these reflected characters. The input parameter should be manually analyzed to verify that no other characters can be injected that would lead to an HTML injection (XSS) vulnerability.

Solution

Review the reflected characters to ensure that they are properly handled as defined by the web application's coding practice. Typical solutions are to apply HTML encoding or percent encoding to the characters depending on where they are placed in the HTML. For example, a double quote might be encoded as " when displayed in a text node, but as %22 when placed in the value of an href attribute.

Low Risk

WASC: Improper Input Handling

Unencoded characters

URL: http://demo.webalyse.nl/pictures/search.php?query=%3C%0a%0dscript%20a%3D4%3Eqss%3D7%3C%0a%0d%2Fscript%3E

Parameter: No param has been required for detecting the information
Authentication: Not Required

Detection Information

Access Path:
http://demo.webalyse.nl/

Payloads

#1 Request:
Payload:
query=%3C%0a%0dscript%20a%3D4%3Eqss%3D7%3C%0a%0d%2Fscript%3E
Request:
GET http://demo.webalyse.nl/pictures/search.php?
query=%3C%0a%0dscript%20a%3D4%3Eqss%3D7%3C%0a%0d%2Fscript%3E

Request Headers:
Referer http://demo.webalyse.nl/
Cookie PHPSESSID=41nd6hub3aho2ia4vucfuv4e22;

#1 Response
comment: A significant portion of the XSS test payload appeared in the web page, but the page's DOM was not modified as expected for a successful exploit. This result should be manually verified to determine its accuracy.

```
if" type="image" style="border: 0pt none ; position: relative; top: 0px;vertical-align:middle;margin-left: 1em;" />
</form>
</div>
</div>

<div class="column prepend-1 span-24 first last">
<h2>Pictures that are tagged as '<
script a=4>qss=7<
/script>'</h2>

<div class="column prepend-1 span-21 first last" style="margin-bottom: 2em;">
<h3 class="error">No pictures here...</h3>


</ul>
</div>

</div>


<div class="column span-24 first last" id="footer" >
<ul>
<li
```

Threat

The web application reflects potentially dangerous characters such as single quotes, double quotes, and angle brackets. These characters are commonly used for HTML injection attacks such as cross-site scripting (XSS).

Impact

No exploit was determined for these reflected characters. The input parameter should be manually analyzed to verify that no other characters can be injected that would lead to an HTML injection (XSS) vulnerability.

Solution

Review the reflected characters to ensure that they are properly handled as defined by the web application's coding practice. Typical solutions are to apply HTML encoding or percent encoding to the characters depending on where they are placed in the HTML. For example, a double quote might be encoded as " when displayed in a text node, but as %22 when placed in the value of an href attribute.

Information Gathered

Web Application Authentication Method

Response Data

Response:

Auth successful against form: http://demo.webalyse.nl/users/login.php
Web Application Authentication Record: Form Auth Record Demo Audit Auth, #34855
User Name: scanner1

Threat

Web application authentication was performed for the scan. The Results section includes a list of authentication credentials used.

Impact

N/A

Solution

N/A

Information Gathered

Host Scan Time

Response Data

Response:

Scan duration: 491 seconds

Start time: Thu, Apr 03 2014, 09:17:57 GMT

End time: Thu, Apr 03 2014, 09:26:08 GMT

Threat

The Host Scan Time is the period of time it takes the scanning engine to perform the vulnerability assessment of a single target host. The Host Scan Time for this host is reported in the Result section below.

The Host Scan Time does not have a direct correlation to the Duration time as displayed in the Report Summary section of a scan results report. The Duration is the period of time it takes the service to perform a scan task. The Duration includes the time it takes the service to scan all hosts, which may involve parallel scanning. It also includes the time it takes for a scanner appliance to pick up the scan task and transfer the results back to the service's Secure Operating Center. Further, when a scan task is distributed across multiple scanners, the Duration includes the time it takes to perform parallel host scanning on all scanners.

Impact

N/A

Solution

N/A

Information Gathered

Flash Analysis

Response Data

Response:

SWF file: http://demo.webalyse.nl/action.swf
Version: 6
Extracted links: 0

Threat

This check provides various details on Flash analysis including problems encountered while handling SWF files.

Impact

N/A

Solution

No action is required.

Information Gathered

Session Cookies

Response Data

Response:

Total cookies: 1
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/ First set at URL: http://demo.webalyse.nl/

Threat

The cookies listed in the Results section were identified as affecting the user's authenticated session to the Web application.

Impact

Session cookies are used to uniquely identify a user to a Web application. A compromised session cookie can lead to user impersonation.

Solution

Review the session cookies and verify that they do not contain sensitive values. Randomly generated session cookies should not exhibit a predictable pattern.

Information Gathered

Protection against Clickjacking vulnerability

Response Data

Response:

http://demo.webalyse.nl/
http://demo.webalyse.nl/admin/index.php?page=login
http://demo.webalyse.nl/calendar.php
http://demo.webalyse.nl/guestbook.php
http://demo.webalyse.nl/pictures/recent.php
http://demo.webalyse.nl/tos.php
http://demo.webalyse.nl/users/home.php
http://demo.webalyse.nl/users/login.php
http://demo.webalyse.nl/users/register.php
http://demo.webalyse.nl/users/sample.php?userid=1

Threat

The URIs listed have a protection against Clickjacking. The protection is implemented by use of X-Frame-Options header.

Impact

X-Frame-Options header is used to prevent framing of the page.

Solution

Another techniques of prevention against Clickjacking is the "framekiller" JavaScript.

Information Gathered

Email Addresses Collected

Response Data

Response:

Number of emails: 1
contact@wackopicko.com first seen at http://demo.webalyse.nl/

Threat

The email addresses listed in the Results section were collected from the returned HTML content during the crawl phase.

Impact

Email addresses may help a malicious user with brute force and phishing attacks.

Solution

Review the email list to see if they are all email addresses you want to expose.

Information Gathered

Links Crawled

Response Data

Response:

Duration of crawl phase (seconds): 225.00
Number of links: 73
(This number excludes form requests and links re-requested during authentication.)

http://demo.webalyse.nl/
http://demo.webalyse.nl/action.swf
http://demo.webalyse.nl/admin/index.php?page=login
http://demo.webalyse.nl/calendar.php
http://demo.webalyse.nl/calendar.php?date=1396603162
http://demo.webalyse.nl/calendar.php?date=1396689562
http://demo.webalyse.nl/calendar.php?date=1396775962
http://demo.webalyse.nl/calendar.php?date=1396862362
http://demo.webalyse.nl/calendar.php?date=1396948762
http://demo.webalyse.nl/calendar.php?date=1397035162
http://demo.webalyse.nl/calendar.php?date=1397121562
http://demo.webalyse.nl/calendar.php?date=1397207962
http://demo.webalyse.nl/calendar.php?date=1397294362
http://demo.webalyse.nl/calendar.php?date=1397380762
http://demo.webalyse.nl/calendar.php?date=1397467162
http://demo.webalyse.nl/calendar.php?date=1397553562
http://demo.webalyse.nl/cart/action.php?action=add&picid=10
http://demo.webalyse.nl/cart/action.php?action=add&picid=11
http://demo.webalyse.nl/cart/action.php?action=add&picid=12
http://demo.webalyse.nl/cart/action.php?action=add&picid=13
http://demo.webalyse.nl/cart/action.php?action=add&picid=14
http://demo.webalyse.nl/cart/action.php?action=add&picid=15
http://demo.webalyse.nl/cart/action.php?action=add&picid=7
http://demo.webalyse.nl/cart/action.php?action=add&picid=8
http://demo.webalyse.nl/cart/action.php?action=add&picid=9
http://demo.webalyse.nl/cart/action.php?action=addcoupon
http://demo.webalyse.nl/cart/action.php?action=delete

http://demo.webalyse.nl/cart/confirm.php
http://demo.webalyse.nl/cart/review.php
http://demo.webalyse.nl/comments/delete_preview_comment.php
http://demo.webalyse.nl/comments/preview_comment.php
http://demo.webalyse.nl/error.php?msg=Error%2C%20need%20to%20provide%20a%20query%20to%20search
http://demo.webalyse.nl/error.php?msg=Error,%20need%20to%20provide%20a%20query%20to%20search
http://demo.webalyse.nl/guestbook.php
http://demo.webalyse.nl/passcheck.php
http://demo.webalyse.nl/piccheck.php
http://demo.webalyse.nl/pictures/highquality.php?picid=10&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=11&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=12&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=13&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=14&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=15&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=7&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=8&key=highquality
http://demo.webalyse.nl/pictures/highquality.php?picid=9&key=highquality
http://demo.webalyse.nl/pictures/purchased.php
http://demo.webalyse.nl/pictures/recent.php
http://demo.webalyse.nl/pictures/search.php
http://demo.webalyse.nl/pictures/search.php?query=&x=-843&y=-113
http://demo.webalyse.nl/pictures/search.php?query=&x=-878&y=-113
http://demo.webalyse.nl/pictures/upload.php
http://demo.webalyse.nl/pictures/view.php?picid=
http://demo.webalyse.nl/pictures/view.php?picid=10
http://demo.webalyse.nl/pictures/view.php?picid=11
http://demo.webalyse.nl/pictures/view.php?picid=12
http://demo.webalyse.nl/pictures/view.php?picid=13
http://demo.webalyse.nl/pictures/view.php?picid=14
http://demo.webalyse.nl/pictures/view.php?picid=15
http://demo.webalyse.nl/pictures/view.php?picid=7
http://demo.webalyse.nl/pictures/view.php?picid=8
http://demo.webalyse.nl/pictures/view.php?picid=9
http://demo.webalyse.nl/tos.php
http://demo.webalyse.nl/users/home.php
http://demo.webalyse.nl/users/login.php
http://demo.webalyse.nl/users/register.php
http://demo.webalyse.nl/users/sample.php?userid=1
http://demo.webalyse.nl/users/similar.php
http://demo.webalyse.nl/users/view.php?userid=1
http://demo.webalyse.nl/users/view.php?userid=10
http://demo.webalyse.nl/users/view.php?userid=11
http://demo.webalyse.nl/users/view.php?userid=2
http://demo.webalyse.nl/users/view.php?userid=4
http://demo.webalyse.nl/users/view.php?userid=9

Threat

The list of unique links crawled by the Web application scanner appear in the Results section. This list may contain fewer links than the maximum threshold defined at scan launch. The maximum links to crawl includes links in this list, requests made via HTML forms, and requests for the same link made as an anonymous and authenticated user.

Impact

N/A

Solution

N/A

Information Gathered

File Upload Form Found

Response Data

Response:

http://demo.webalyse.nl/

Threat

A file upload form found while crawling the application.

Impact

N/A

Solution

N/A

Information Gathered

External Links Discovered

Response Data

Response:

Number of links: 6
http://active.macromedia.com/flash5/cabs/swflash.cab
http://localhost/j
http://localhost/q.swf
javascript:qss=7
mailto:contact@wackopicko.com
http://www.macromedia.com/go/getflashplayer

Threat

The external links discovered by the Web application scanning engine are provided in the Results section. These links were present on the target Web application, but were not crawled.

Impact

N/A

Solution

N/A

Information Gathered

DNS Host Name

Response Data

Response:

#table
IP_address Host_name

178.79.129.186 nuvini.com

Threat

The fully qualified domain name of this host, if it was obtained from a DNS server, is displayed in the RESULT section.

Impact

Solution

Information Gathered

Cookies Collected

Response Data

Response:

Total cookies: 1
PHPSESSID=41nd6hub3aho2ia4vucfuv4e22; path=/ First set at URL: http://demo.webalyse.nl/

Threat

The cookies listed in the Results section were received from the web application during the crawl phase.

Impact

Cookies may contain sensitive information about the user. Cookies sent via HTTP may be sniffed.

Solution

Review cookie values to ensure that sensitive information such as passwords are not present within them.

Information Gathered

Authentication Form found

Response Data

Response:

Authentication form found at: http://demo.webalyse.nl/users/login.php
Action uri: http://demo.webalyse.nl/users/login.php
Fields: username, password, WasNoName_S_2_loginAuthentication form found at: http://demo.webalyse.nl/admin/index.php?page=login
Action uri: http://demo.webalyse.nl/admin/index.php?page=login
Fields: adminname, password,

Threat

Authentication Form was found during the web application crawling.

Impact

N/A

Solution

N/A

Information Gathered

Operating System Detected

Response Data

Response:

#table cols="3"
Operating_System Technique ID

Ubuntu_/_Linux_2.6.x TCP/IP_Fingerprint U4856:80

Threat

Several different techniques can be used to identify the operating system (OS) running on a host. A short description of these techniques is provided below. The specific technique used to identify the OS on this host is included in the RESULTS section of your report.

1) **TCP/IP Fingerprint**: The operating system of a host can be identified from a remote system using TCP/IP fingerprinting. All underlying operating system TCP/IP stacks have subtle differences that can be seen in their responses to specially-crafted TCP packets. According to the results of this "fingerprinting" technique, the OS version is among those listed below.

Note that if one or more of these subtle differences are modified by a firewall or a packet filtering device between the scanner and the host, the fingerprinting technique may fail. Consequently, the version of the OS may not be detected correctly. If the host is behind a proxy-type firewall, the version of the operating system detected may be that for the firewall instead of for the host being scanned.

2) **NetBIOS**: Short for Network Basic Input Output System, an application programming interface (API) that augments the DOS BIOS by adding special functions for local-area networks (LANs). Almost all LANs for PCs are based on the NetBIOS. Some LAN manufacturers have even extended it, adding additional network capabilities. NetBIOS relies on a message format called Server Message Block (SMB).

3) **PHP Info**: PHP is a hypertext pre-processor, an open-source, server-side, HTML-embedded scripting language used to create dynamic Web pages. Under some configurations it is possible to call PHP functions like phpinfo() and obtain operating system information.

4) **SNMP**: The Simple Network Monitoring Protocol is used to monitor hosts, routers, and the networks to which they attach. The SNMP service maintains Management Information Base (MIB), a set of variables (database) that can be fetched by Managers. These include "MIB_II.system.sysDescr" for the operating system.

Impact

Not applicable.

Solution

Not applicable.

Information Gathered

Scan Diagnostics

Response Data

Response:

Loaded 0 blacklist entries.
Loaded 0 whitelist entries.
No more requeues, redundant link threshold has been surpassed.
Collected 78 links overall.
Number of times authentication needed to be re-verified: 1
Path manipulation: estimated time < 1 minute (115 tests, 84 inputs)
Path manipulation: 115 vulnsigs tests, completed 1836 requests, 6 seconds. All tests completed.
WS enumeration: estimated time < 1 minute (9 tests, 78 inputs)
WS enumeration: 9 vulnsigs tests, completed 54 requests, 1 seconds. All tests completed.
Batch #1 URI parameter manipulation (no auth): estimated time < 1 minute (46 tests, 0 inputs)
Batch #1 URI parameter manipulation (no auth): 46 vulnsigs tests, completed 138 requests, 1 seconds. No tests to execute.
Batch #1 Login form parameter manipulation (no auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #1 Login form parameter manipulation (no auth): 46 vulnsigs tests, completed 184 requests, 1 seconds. No tests to

execute.
Batch #1 URI blind SQL manipulation (no auth): estimated time < 1 minute (19 tests, 0 inputs)
Batch #1 URI blind SQL manipulation (no auth): 19 vulnsigs tests, completed 57 requests, 0 seconds. No tests to execute.
Batch #1 Login form blind SQL manipulation (no auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #1 Login form blind SQL manipulation (no auth): 19 vulnsigs tests, completed 76 requests, 1 seconds. All tests completed.
URI parameter time-based tests (no auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #1 URI parameter time-based tests (no auth): 8 vulnsigs tests, completed 24 requests, 1 seconds. No tests to execute.
Batch #1 URI parameter manipulation (auth): estimated time < 1 minute (46 tests, 0 inputs)
Batch #1 URI parameter manipulation (auth): 46 vulnsigs tests, completed 184 requests, 1 seconds. No tests to execute.
Batch #1 Form parameter manipulation (auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #1 Form parameter manipulation (auth): 46 vulnsigs tests, completed 690 requests, 6 seconds. All tests completed.
Batch #1 Login form parameter manipulation (auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #1 Login form parameter manipulation (auth): 46 vulnsigs tests, completed 184 requests, 0 seconds. No tests to execute.
Batch #1 URI blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 0 inputs)
Batch #1 URI blind SQL manipulation (auth): 19 vulnsigs tests, completed 76 requests, 1 seconds. No tests to execute.
Batch #1 Form blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #1 Form blind SQL manipulation (auth): 19 vulnsigs tests, completed 285 requests, 3 seconds. All tests completed.
Batch #1 Login form blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #1 Login form blind SQL manipulation (auth): 19 vulnsigs tests, completed 76 requests, 1 seconds. All tests completed.
URI parameter time-based tests (auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #1 URI parameter time-based tests (auth): 8 vulnsigs tests, completed 32 requests, 0 seconds. No tests to execute.
Form field time-based tests (auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #1 Form field time-based tests (auth): 8 vulnsigs tests, completed 120 requests, 3 seconds. No tests to execute.
Batch #2 URI parameter manipulation (no auth): estimated time < 1 minute (46 tests, 2 inputs)
Batch #2 URI parameter manipulation (no auth): 46 vulnsigs tests, completed 230 requests, 1 seconds. All tests completed.
Batch #2 URI blind SQL manipulation (no auth): estimated time < 1 minute (19 tests, 2 inputs)
Batch #2 URI blind SQL manipulation (no auth): 19 vulnsigs tests, completed 95 requests, 1 seconds. All tests completed.
URI parameter time-based tests (no auth): estimated time < 1 minute (8 tests, 2 inputs)
Batch #2 URI parameter time-based tests (no auth): 8 vulnsigs tests, completed 40 requests, 0 seconds. All tests completed.
Batch #2 URI parameter manipulation (auth): estimated time < 1 minute (46 tests, 2 inputs)
Batch #2 URI parameter manipulation (auth): 46 vulnsigs tests, completed 276 requests, 1 seconds. All tests completed.
Batch #2 Form parameter manipulation (auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #2 Form parameter manipulation (auth): 46 vulnsigs tests, completed 46 requests, 1 seconds. All tests completed.
Batch #2 URI blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 2 inputs)
Batch #2 URI blind SQL manipulation (auth): 19 vulnsigs tests, completed 114 requests, 1 seconds. All tests completed.
Batch #2 Form blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #2 Form blind SQL manipulation (auth): 19 vulnsigs tests, completed 19 requests, 1 seconds. All tests completed.
URI parameter time-based tests (auth): estimated time < 1 minute (8 tests, 2 inputs)
Batch #2 URI parameter time-based tests (auth): 8 vulnsigs tests, completed 48 requests, 1 seconds. All tests completed.
Form field time-based tests (auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #2 Form field time-based tests (auth): 8 vulnsigs tests, completed 8 requests, 91 seconds. No tests to execute.
Batch #3 URI parameter manipulation (no auth): estimated time < 1 minute (46 tests, 0 inputs)
Batch #3 URI parameter manipulation (no auth): 46 vulnsigs tests, completed 46 requests, 0 seconds. No tests to execute.
Batch #3 URI blind SQL manipulation (no auth): estimated time < 1 minute (19 tests, 0 inputs)
Batch #3 URI blind SQL manipulation (no auth): 19 vulnsigs tests, completed 19 requests, 0 seconds. No tests to execute.
URI parameter time-based tests (no auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #3 URI parameter time-based tests (no auth): 8 vulnsigs tests, completed 8 requests, 1 seconds. No tests to execute.
Batch #3 URI parameter manipulation (auth): estimated time < 1 minute (46 tests, 2 inputs)
Batch #3 URI parameter manipulation (auth): 46 vulnsigs tests, completed 230 requests, 1 seconds. All tests completed.
Batch #3 Form parameter manipulation (auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #3 Form parameter manipulation (auth): 46 vulnsigs tests, completed 92 requests, 1 seconds. All tests completed.
Batch #3 URI blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 2 inputs)
Batch #3 URI blind SQL manipulation (auth): 19 vulnsigs tests, completed 95 requests, 2 seconds. All tests completed.
Batch #3 Form blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #3 Form blind SQL manipulation (auth): 19 vulnsigs tests, completed 38 requests, 0 seconds. All tests completed.
URI parameter time-based tests (auth): estimated time < 1 minute (8 tests, 2 inputs)
Batch #3 URI parameter time-based tests (auth): 8 vulnsigs tests, completed 40 requests, 1 seconds. All tests completed.
Form field time-based tests (auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #3 Form field time-based tests (auth): 8 vulnsigs tests, completed 16 requests, 1 seconds. No tests to execute.
Batch #4 File Upload analysis: estimated time < 1 minute (1 tests, 1 inputs)
Batch #4 File Upload analysis: 1 vulnsigs tests, completed 0 requests, 0 seconds. All tests completed.
HTTP call manipulation: estimated time < 1 minute (33 tests, 0 inputs)
HTTP call manipulation: 33 vulnsigs tests, completed 0 requests, 0 seconds. No tests to execute.
Open Redirect analysis: estimated time < 1 minute (1 tests, 0 inputs)
Open Redirect analysis: 1 vulnsigs tests, completed 0 requests, 0 seconds. No tests to execute.
CSRF: estimated time < 1 minute (2 tests, 13 inputs)
CSRF: 2 vulnsigs tests, completed 0 requests, 91 seconds. All tests completed.
StaticSessionID: estimated time < 1 minute (1 tests, 1 inputs)
StaticSessionID: 1 vulnsigs tests, completed 0 requests, 3 seconds. All tests completed.
File Inclusion analysis: estimated time < 1 minute (1 tests, 84 inputs)
File Inclusion analysis: 1 vulnsigs tests, completed 0 requests, 0 seconds. All tests completed.
Cookie manipulation: estimated time < 1 minute (37 tests, 1 inputs)
Cookie manipulation: 37 vulnsigs tests, completed 949 requests, 7 seconds. XSS optimization removed 1752 links. Completed 949 requests of 2701 estimated requests (35%). All tests completed.
Header manipulation: estimated time < 1 minute (37 tests, 73 inputs)
Header manipulation: 37 vulnsigs tests, completed 1752 requests, 16 seconds. XSS optimization removed 1752 links. Completed

1752 requests of 5402 estimated requests (32%). All tests completed.
Login Brute Force manipulation estimated time: no tests enabled
Batch #4 URI parameter manipulation (auth): estimated time < 1 minute (46 tests, 0 inputs)
Batch #4 URI parameter manipulation (auth): 46 vulnsigs tests, completed 0 requests, 0 seconds. No tests to execute.
Batch #4 Form parameter manipulation (auth): estimated time < 1 minute (46 tests, 6 inputs)
Batch #4 Form parameter manipulation (auth): 46 vulnsigs tests, completed 46 requests, 1 seconds. All tests completed.
Batch #4 URI blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 0 inputs)
Batch #4 URI blind SQL manipulation (auth): 19 vulnsigs tests, completed 0 requests, 0 seconds. No tests to execute.
Batch #4 Form blind SQL manipulation (auth): estimated time < 1 minute (19 tests, 6 inputs)
Batch #4 Form blind SQL manipulation (auth): 19 vulnsigs tests, completed 19 requests, 0 seconds. All tests completed.
Form field time-based tests (auth): estimated time < 1 minute (8 tests, 0 inputs)
Batch #4 Form field time-based tests (auth): 8 vulnsigs tests, completed 8 requests, 0 seconds. No tests to execute.
Batch #4 URI parameter manipulation (no auth): estimated time < 1 minute (46 tests, 0 inputs)
Batch #4 URI parameter manipulation (no auth): 46 vulnsigs tests, completed 0 requests, 0 seconds. No tests to execute.
Batch #4 Form parameter manipulation (no auth): estimated time < 1 minute (46 tests, 6 inputs)

Threat

This check provides various details of the scan's performance and behavior. In some cases, this check can be used to identify problems that the scanner encountered when crawling the target Web application.

Impact

The scan diagnostics data provides technical details about the crawler's performance and behavior. This information does not necessarily imply problems with the Web application.

Solution

No action is required.

Information Gathered

Cookies Issued Without User Consent

Response Data

Response:

Total cookies: 1
PHPSESSID=d453c6rb55gl6opvrc89lrg503; path=/ First set at URL: http://demo.webalyse.nl/

Threat

The cookies listed in the Results section were issued from the web application during the crawl without accepting any opt-in dialogs.

Impact

Cookies may be set without user explicitly agreeing to accept them.

Solution

Review the application to ensure that all cookies listed are supposed to be issued without user opt-in. If the EU Cookie law is applicable for this web application, ensure these cookies require user opt-in or have been classified as exempt by your organization.

Information Gathered

Links Discovered During User-Agent and Mobile Site Checks

Response Data

Response:

Unique content discovered during user-agent and common mobile device specific subdomains and paths manipulation: User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3 (.NET CLR 3.5.30729)
http://demo.webalyse.nl/
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:7.0.1) Gecko/20100101 Firefox/7.0.1
http://demo.webalyse.nl/
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.04506.30; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729)
http://demo.webalyse.nl/
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.835.202 Safari/535.1
http://demo.webalyse.nl/
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/531.22.7 (KHTML, like Gecko) Version/4.0.5 Safari/531.22.7
http://demo.webalyse.nl/
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/534.50 (KHTML, like Gecko) Version/5.1 Safari/534.50
http://demo.webalyse.nl/
User-Agent: Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_3_3 like Mac OS X; en-us) AppleWebKit/533.17.9 (KHTML, like Gecko) Version/5.0.2 Mobile/8J2 Safari/6533.18.5
http://demo.webalyse.nl/
User-Agent: Opera/9.80 (IPhone; Opera Mini/5.0.019802/886; U; en) Presto/2.4.15
http://demo.webalyse.nl/
User-Agent: BlackBerry9700/5.0.0.405 Profile/MIDP-2.1 Configuration/CLDC-1.1 VendorID/102
http://demo.webalyse.nl/

Threat

Links were discovered via requests using an alternate User-Agent or guessed based on common mobile device URI patterns. The scanner attempts to determine if the Web application changes its behavior when accessed by mobile devices. These checks are based on modifying the User-Agent, changing the domain name, and appending common directories.

The extra links discovered by the Web application scanner during User-Agent manipulation are provided in the Results section.

Impact

The Web application should apply consistent security measures irrespective of browser platform, type or version used to access the application. If the Web application fails to apply security controls to alternate representations of the site, then it may be exposed to vulnerabilities like cross-site scripting, SQL injection, or authorization-based attacks.

Solution

No specific vulnerability has been discovered that requires action to be taken. These links are provided to ensure that a review of the web application includes all possible access points.