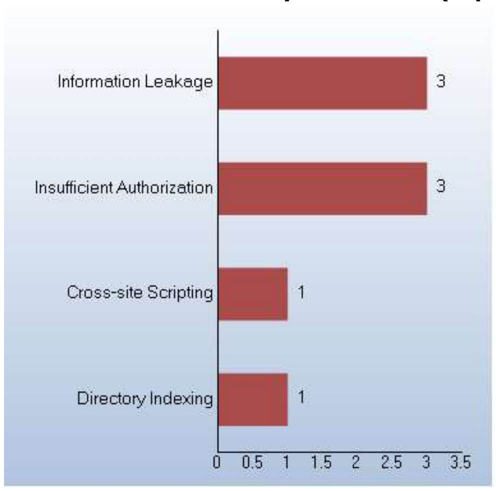
Micro Focus WebInspect

Multiple Reports

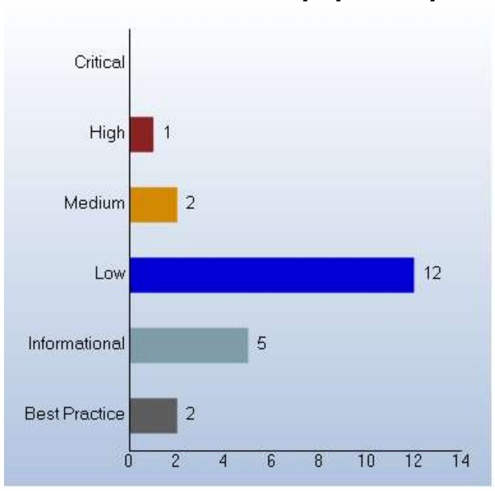
Web Application Assessment Report

Scan Name:	http://137.116.132.150/STSBidding/frontend/login-2		
Policy:	Standard	**Crawl Sessions:**	50
Scan Date:	2/19/2024 4:29:58 PM	**Vulnerabilities:**	15
Scan Version:	21.2.0.117	**Scan Duration:**	1 hour : 58 minutes
Scan Type:	Site	**Client:**	Custom

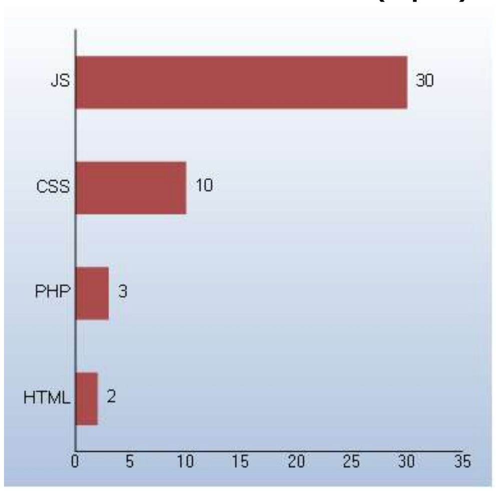
Vulnerabilities By Threat Class (Top 12)



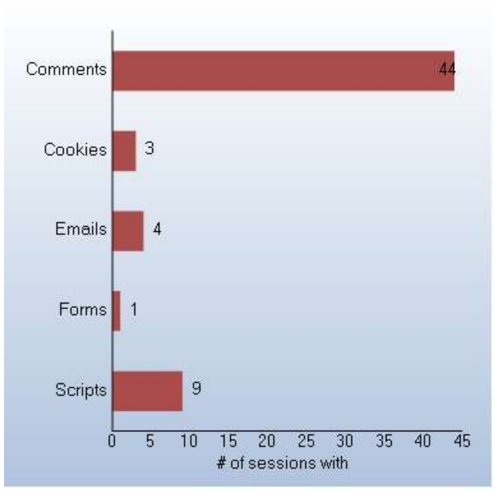
Vulnerability By Severity



Session Extensions (Top 12)



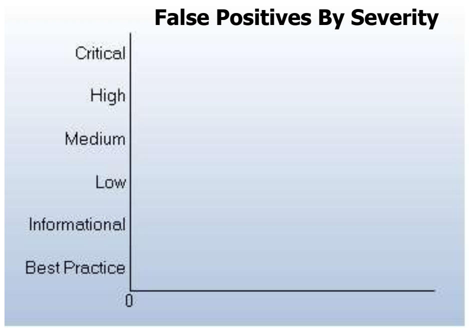
Site Structure



Scan Name:	http://137.116.132.150/STSBidding/frontend/login-2		
Policy:	Standard	**Crawl Sessions:**	50
Scan Date:	2/19/2024 4:29:58 PM	**Vulnerabilities:**	15
Scan Version:	21.2.0.117	**Scan Duration:**	1 hour : 58 minutes
Scan Type:	Site	**Client:**	Custom

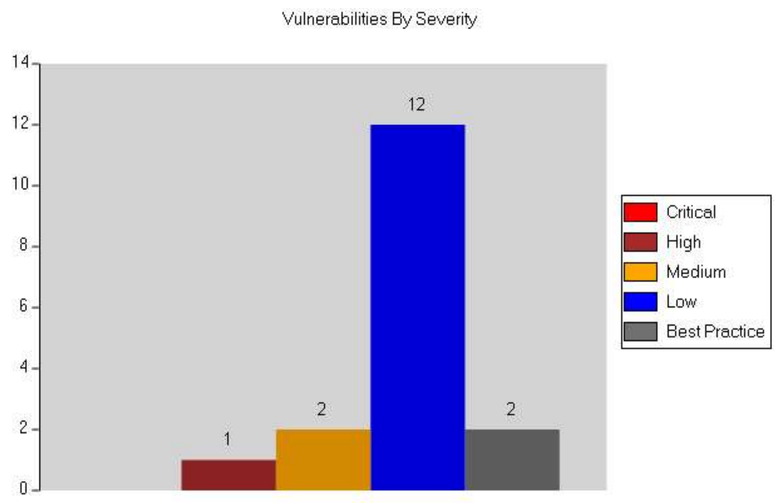
False Positive Summary

False Positive Count	0
Checks with False Positives	0
False Positive Percentage	0.00%
False Positives with Comments	0



Scan Name:	http://137.116.132.150/STSBidding/frontend/login-2		
Policy:	Standard	**Crawl Sessions:**	50
Scan Date:	2/19/2024 4:29:58 PM	**Vulnerabilities:**	15
Scan Version:	21.2.0.117	**Scan Duration:**	1 hour : 58 minutes
Scan Type:	Site	**Client:**	Custom

Server: http://137.116.132.150:80



High **Password Management: Weak Password Policy**

Summary:

Authentication is an important aspect of security. Password authentication requires users to present login credentials as evidence to validate their identity before granting them access to server resources. The reliability of the authentication process depends on the security of the login credentials. Password policies that ensure users create strong passwords are therefore crucial to deploying secure websites. Password strength is a measure of the effectiveness it provides in resisting guessing and brute force attacks. Some of the parameters that help define the password strength include password characteristics such as password length, complexity and randomness.

WebInspect has detected that the current website does not meet the basic guidelines for a secure password.

*The password value used in Login Macro **loginMacro.webmacro** fails to meet these requirements.*

The password does not meet the minimum length requirement of 8 characters.
The password contains password from a list of commonly used 10000 passwords.
The password contains sequential characters such that both the halves of the password are sequential like abcd1234.

Note: The assumption is that the username / password used during pen-testing meet the username/password requirement for the web application when deployed live.

Implication:

System security is compromised by using weak passwords that can be easily guessed or are an easy target to brute force attacks. Authentication systems fail on compromised passwords as they cannot distinguishbetween impostors and authentic users of the system. Thus, compromising the integrity and confidentiality of the system resources and data.

Fix:

The effective strength of the password can be increased by enforcing rules that make the password random and harder to guess.
As per the latest NIST guidelines, at a minimum the password policy should adhere to the following rules:

- The password should be at least 8 characters long.
- The password should not contain contextual information such as login credentials, website name etc
- In addition the password strength can be increased by enforcing rules that increase the password randomness. For example, you could require a password have a minimum of 4 tokens, where a token is defined as a set of either [letters], [numbers] or special characters. It is recommended that you implement a password policy that helps increase the password entropy and hence the password strength.
- The password should not have sequential characters like "abcd1234"
- The password should not have all same letters like "Aaaaaaaaa"

Reference:

NIST Computer Security
Estimating Password Strength
NIST Guide to Enterprise Password Management
NIST Password Guidelines
Special Publication 800-118
OWASP
Authentication Cheat Sheet
Bruce Schneier - Choosing Secure Passwords
Choosing Secure Passwords

Attack Request:

```
POST /STSBidding/service/Login/staffLogin.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: text/plain;charset=UTF-8
Origin: http://137.116.132.150
Referer: http://137.116.132.150/STSBidding/frontend/login
Content-Length: 34
Connection: keep-alive
Pragma: no-cache
Cookie: PHPSESSID=1ce78puf2pghc93vhdpgl2uhbr
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="7a143b2e";thid="161";

{"empNO":"00002","password":"123"}
```

Attack Response:

```
HTTP/1.1 404 Not Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE,PATCH
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 09:30:51 GMT
Content-Length: 71
Connection: Keep-Alive
X-BlueCoat-Authorization:
019a43ba5a07a9e5=335A2F8100000004kHUFOU5yU6TR3zeDbV4l9UmtH+ieBgAABAAAAJqqkQFAOAAAAAAAAG/AbgQAAAAA

{
  "err": "not found a employee @ number 00002",
  "status": 404
}
```

File Names: • http://137.116.132.150:80/STSBidding/service/Login/staffLogin.php

Medium **Path Manipulation: Relative Path Overwrite**

Summary:

The Relative Path Overwrite (RPO) vulnerability, also known as Path-Relative Style Sheet Import (PRSSI), can be used on some servers to overwrite the path to CSS files when the application uses relative paths to include them. This attack abuses the path handling features of some web languages and frameworks, and tricks the browsers into importing HTML content as stylesheets. The following conditions allow for a successful RPO attack:

- Usage of relative paths to import stylesheets
- Browser uses quirks mode (this may be triggered by using the meta tag or an older doctype)
- Ability to overwrite the relative path

Execution:

Consider a stylesheet being imported using a relative path such as "styles/main.css" when a request to http://example.com/index.php is made. In this case, the relative path is translated to http://example.com/styles/main.css. If an application is vulnerable to RPO, then a request to http://example.com/index.php/ will include the stylesheet from http://example.com/index.php/styles/main.css. Due to the way this request is handled by the server, the response is the same as the original HTML instead of a 404 or the contents of the actual stylesheet. This results in script being executed while handling the stylesheet inclusion.

Implication:

When the injected path is reflected in the response, it can lead to a successful cross-site scripting attack. This is made possible by using an executable CSS expression in the attack.

Fix:

It is recommended to use an absolute path while including stylesheets in web pages. Other ways of mitigating this issue could be the use of headers such as X-Frame-Options and X-Content-Type-Options to avoid mime sniffing, along with a modern doctype. However, these options will not mitigate the vulnerability with older browsers and Internet Explorer in Compatibility mode. It is therefore suggested to use the right mitigation based on the application's userbase.

Reference:

- NIST: CVE-2015-1431 - https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-1431
- Detecting and exploiting path-relative stylesheet import (PRSSI) vulnerabilities - http://blog.portswigger.net/2015/02/prssi.html
- The Spanner - http://www.thespanner.co.uk/2014/03/21/rpo/

Attack Request:

```
GET /STSBidding/src/STS00000/STS00100.php/spidir/spidir/ HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Host: 137.116.132.150
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="FB9F5CE210B233F31FD30A6F541ADB56";PSID="17CBFAB3710D70FEB53CFDA888095296";Se
ssionType="AuditAttack";CrawlType="None";AttackType="Other";OriginatingEngineID="d00c22da-e671-43f8-a4e8-
7ee258f85dee";AttackSequence="0";AttackParamDesc="";AttackParamIndex="-
1";AttackParamSubIndex="0";CheckId="11392";Engine="Relative+Path+Override";SmartMode="2";tht="11";
X-RequestManager-Memo: stid="59";stmi="0";sc="1";rid="ab3f3aa9";
X-Request-Memo: rid="9bf82299";sc="1";thid="180";
Cookie: PHPSESSID=dbft8al0dm6rl9kto0pqjsm3mt;BCSI-CS-019a43ba5a07a9e5=2
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 11:25:50 GMT
Content-Length: 9927
Connection: Keep-Alive

...TRUNCATED...al-scale=1.0" />
    <link rel="stylesheet" href="../../asset/css/googlefont-kanit.css" />
    <link rel="stylesheet" href="../../asset/...TRUNCATED...
```

File Names:
- http://137.116.132.150:80/STSBidding/src/STS00000/STS00100.php/spidir/spidir/

- http://137.116.132.150:80/STSBidding/src/STS00000/STS00200.php/spidir/spidir/

Low **Poor Error Handling: Unhandled Exception**

Summary:

A minor vulnerability has been discovered within your web application due to the the presence of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Execution:

To verify the issue, click the 'HTTP Response' button on the properties view and review the highlighted areas to determine the Unix path found.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- **Uniform Error Codes:** Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.

- **Informational Error Messages:** Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.

- **Proper Error Handling:** Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

```
GET /STSBidding/frontend/assets/index-6302bb4d.js HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://137.116.132.150/STSBidding/frontend/login
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="b904b337";thid="182";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: application/javascript
Last-Modified: Fri, 16 Feb 2024 06:21:03 GMT
Accept-Ranges: bytes
ETag: "8031f350a060da1:0"
Vary: Accept-Encoding
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 09:30:43 GMT
Content-Length: 1046156
Connection: Keep-Alive
Age: 0

...TRUNCATED...OnConfirm:!0,preConfirm:l=>fetch(`//api.github.com/users/${l}`).then(a=>{if(!a.ok)throw new Error
(a.statusT...TRUNCATED...
```

File Names:
- http://137.116.132.150:80/STSBidding/frontend/assets/index-6302bb4d.js

Low **Often Misused: File Upload**

Summary:

An indicator of file upload capability was found. File upload capability allows a web user to send a file from his or her computer to the webserver. If the web application that receives the file does not carefully examine it for malicious content, an attacker may be able to use file uploads to execute arbitrary commands on the server. Recommendations include adopting a strict file upload policy that prevents malicious material from being uploaded via sanitization and filtering.

Implication:

The exact implications depend upon the nature of the files an attacker would be able to upload. Implications range from unauthorized content publishing to aid in phising attacks, all the way to full compromise of the web server.

Fix:

For Security Operations:
This check is part of unknown application testing. Unknown application testing seeks to uncover new vulnerabilities in both custom and commercial software. Because of this, there are no specific patches or descriptions for this issue. If there is no apparent file upload capability on the page, this check may be safely ignored. You can instruct the scanner to ignore this vulnerability by right-clicking the vulnerability node on the displayed results tree and click "Ignore Vulnerability."

For QA:
This issue will need to be resolved in the production code. Notify the appropriate developer of this issue.

For Development:
Ensure that the following steps are taken to sanitize the file being received:

- Limit the types of files that can be uploaded. For instance, on an image upload page, any file other than a .jpg should be refused.
- Ensure that the web user has no control whatsoever over the name and location of the uploaded file on the server.
- Never use the name that the user assigns it.
- Never derive the filename from the web user's username or session ID.
- Do not place the file in a directory accessible by web users. It is preferable for this location to be outside of the webroot.
- Ensure that strict permissions are set on both the uploaded file and the directory it is located in.
- Do not allow execute permissions on uploaded files. If possible, deny all permission for all users but the web application user.
- Verify that the uploaded file contains appropriate content. For instance, an uploaded JPEG should have a standard JPEG file header.

Attack Request:

```
GET /STSBidding/frontend/assets/sweetalert2-cadb790f.js HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
Referer: http://137.116.132.150/STSBidding/frontend/assets/index-6302bb4d.js
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="1dccc719";thid="186";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: application/javascript
Last-Modified: Fri, 16 Feb 2024 06:20:58 GMT
Accept-Ranges: bytes
ETag: "041f84da060da1:0"
Vary: Accept-Encoding
Server: Microsoft-IIS/10.0
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 09:30:43 GMT
Content-Length: 71400
Connection: Keep-Alive
Age: 0

...TRUNCATED....concat(s.input,'" id="').concat(s.input,`" />
  <input type="file" class="`).concat(s.file,`" />
  <div class="`).c...TRUNCATED...
```

File Names:

- http://137.116.132.150:80/STSBidding/frontend/assets/sweetalert2-cadb790f.js
- http://137.116.132.150:80/STSBidding/asset/js/sweetalert2-11.7.3.js

Low **Web Server Misconfiguration: Unprotected File**

Summary:

System Environment variables log files contain information about the nature of your web application, and would allow an attacker to gain insightful information about the web system setup. Recommendations include removing this file from the affected system.

Implication:

A fundamental part of any successful attack is reconnaissance and information gathering. The primary danger from exploitation of this vulnerability is that an attacker will be able to utilize the information in launching a more serious attack. It is very simple to check for its existence, and a file most definitely on the short list of things for which a potential attacker would look.

Fix:

For Security Operations:
Remove this file from the system in question. One of the most important aspects of web application security is to restrict access to important files or directories only to those individuals who actually need to access them. Ensure that the private architectural structure of your web application is not exposed to anyone who wishes to view it as even seemingly innocuous directories can provide important information to a potential attacker.

For QA:
Notify your Security or Network Operations team of this issue.

For Development:
Notify your Security or Network Operations team of this issue.

Attack Request:

```
GET /phpinfo.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Host: 137.116.132.150
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="3C49F0FD019C23669FB4655598CC492E";PSID="85EA878AB0D2D24EABA4DEEF89BEDF8C";SessionType="AuditAttack";CrawlType="None";AttackType="Probe";OriginatingEngineID="65cee7d3-561f-40dc-b5eb-c0b8c2383fcb";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="10478";Engine="Request+Modify";SmartMode="2";tht="11";
```

```
X-RequestManager-Memo: stid="59";stmi="0";sc="1";rid="f2c2d1a1";
X-Request-Memo: rid="d364d141";sc="1";thid="180";
Cookie: PHPSESSID=dbft8al0dm6rl9kto0pqjsm3mt;BCSI-CS-019a43ba5a07a9e5=2
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 11:27:07 GMT
Content-Length: 84715
Connection: Keep-Alive

...TRUNCATED...v">HTTP/1.1</td></tr>
<tr><td class="e">$_SERVER['SERVER_PORT_SECURE']</td><td class="v">0</td></tr>
<tr><td class="e">$_SERVER['SERVER_PORT']</td><td class="v">80</td></tr>
<tr><td class="e">$_SERVER['SERVER_NAME']</td><td class="v">137.116.132.150</td></tr>
<tr...TRUNCATED...
```

File Names:

- http://137.116.132.150:80/phpinfo.php

Low

Web Server Misconfiguration: Unprotected Directory

Summary:

IIS/Microsoft product directories were discovered within your web application during a Directory Enumeration scan. Risks associated with an attacker discovering a directory on your application server depend upon what type of directory is discovered, and what types of files are contained within it. The primary threat, other than accessing files containing sensitive information, is that an attacker can utilize the information discovered in that directory to perform other types of attacks. Recommendations include restricting access to important directories or files by adopting a "need to know" requirement for both the document and server root, and turning off features such as Automatic Directory Listings that provide information that could be utilized by an attacker when formulating or conducting an attack.

Fix:

For Security Operations:
You should evaluate the production requirements for the found directory. If the directory is not required for production operation, then the directory and its contents should be removed or restricted by a server access control mechanism. More information about implementing access control schemes can be found in the References. Automatic directory indexing should also be disabled, if applicable.

For Development:
This problem will be resolved by the web application server administrator. In general, do not rely on 'hidden' directories within the web root that can contain sensitive resources or web applications. Assume an attacker knows about the existence of all directories and files on your web site, and protect them with proper access controls.

For QA:
This problem will be resolved by the web application server administrator.

Reference:

Implementing Basic Authentication in IIS
http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/abbca505-6f63-4267-aac1-1ea89d861eb4.mspx

Attack Request:

```
GET /aspnet_client/ HTTP/1.1
Accept: */*
Accept-Encoding: gzip, def...TRUNCATED...
```

Attack Response:

```
HTTP/1.1 403 Forbidden
Cache-Control: private
Content-Type: ...TRUNCATED...
```

File Names: ● http://137.116.132.150:80/aspnet_client/

Low **Cookie Security: HTTPOnly not Set**

Summary:

The web application does not utilize HTTP only cookies. This is a new security feature introduced by Microsoft in IE 6 SP1 to mitigate the possibility of a successful Cross-Site scripting attack by not allowing cookies with the HTTP only attribute to be accessed via client-side scripts. Recommendations include adopting a development policy that includes the utilization of HTTP only cookies, and performing other actions such as ensuring proper filtration of user-supplied data, utilizing client-side validation of user supplied data, and encoding all user supplied data to prevent inserted scripts being sent to end users in a format that can be executed.

Reference:

References:
https://social.msdn.microsoft.com/Search/en-US?query=HTTPOnly%20Cookie&emptyWatermark=true&ac=5

Attack Request:

```
GET /STSBidding/service/Login/verify.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://137.116.132.150/STSBidding/frontend/login
Connection: keep-alive
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="7b295385";thid="186";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 404 Not Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE,PATCH
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 09:30:44 GMT
Content-Length: 99
Connection: Keep-Alive
Set-Cookie: PHPSESSID=1ce78puf2pghc93vhdpgl2uhbr; path=/

{
   "err": "ไม่เคยลงทะเบียนในระบบ",
   "st
```

File Names: ● http://137.116.132.150:80/STSBidding/service/Login/verify.php

Low **Poor Error Handling: Unhandled Exception**

Summary:

A minor vulnerability has been detected within your web application due to the discovery of a fully qualified path name to the root of your system. This most often occurs in context of an error being produced by the web application. Fully qualified server path names allow an attacker to know the file system structure of the web server, which is a baseline for many other types of attacks to be successful. Recommendations include adopting a consistent error handling scheme and mechanism that prevents fully qualified path names from being displayed.

Fix:

For Development:

Don't display fully qualified pathnames as part of error or informational messages. At the least, fully qualified pathnames can provide an attacker with important information about the architecture of web application.

For Security Operations:

The following recommendations will help to ensure that a potential attacker is not deriving valuable information from any error message that is presented.

- Uniform Error Codes: Ensure that you are not inadvertently supplying information to an attacker via the use of inconsistent or "conflicting" error messages. For instance, don't reveal unintended information by utilizing error messages such as Access Denied, which will also let an attacker know that the file he seeks actually exists. Have consistent terminology for files and folders that do exist, do not exist, and which have read access denied.
- Informational Error Messages: Ensure that error messages do not reveal too much information. Complete or partial paths, variable and file names, row and column names in tables, and specific database errors should never be revealed to the end user. Remember, an attacker will gather as much information as possible, and then add pieces of seemingly innocuous information together to craft a method of attack.
- Proper Error Handling: Utilize generic error pages and error handling logic to inform end users of potential problems. Do not provide system information or other data that could be utilized by an attacker when orchestrating an attack.

For QA:

In reality, simple testing can usually determine how your web application will react to different input errors. More expansive testing must be conducted to cause internal errors to gauge the reaction of the site.

The best course of action for QA associates to take is to ensure that the error handling scheme is consistent. Do you receive a different type of error for a file that does not exist as opposed to a file that does? Are phrases like "Permission Denied" utilized which could reveal the existence of a file to an attacker? It is often a seemingly innocuous piece of information that provides an attacker with the means to discover something else which he can then utilize when conducting an attack.

Attack Request:

```
GET /phpinfo.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Host: 137.116.132.150
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="3C49F0FD019C23669FB4655598CC492E";PSID="85EA878AB0D2D24EABA4DEEF89BEDF8C";S
essionType="AuditAttack";CrawlType="None";AttackType="Probe";OriginatingEngineID="65cee7d3-561f-40dc-b5eb-
c0b8c2383fcb";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="10478
";Engine="Request+Modify";SmartMode="2";tht="11";
X-RequestManager-Memo: stid="59";stmi="0";sc="1";rid="f2c2d1a1";
X-Request-Memo: rid="d364d141";sc="1";thid="180";
Cookie: PHPSESSID=dbft8al0dm6rl9kto0pqjsm3mt;BCSI-CS-019a43ba5a07a9e5=2
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/10.0
```

```
X-Powered-By: PHP/8.1.2
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 11:27:07 GMT
Content-Length: 84715
Connection: Keep-Alive

...TRUNCATED...s="e">Loaded Configuration File </td><td class="v">C:\PHP\php.ini </td></tr>
<tr><td class="e">Scan this ...TRUNCATED.../tr>
<tr><td class="e">error_log</td><td class="v">C:\Windows\Temp\php8_errors.log</td><td class="v">C:\W
indows\Temp\php8_errors.log</td></tr>
<tr><td clas...TRUNCATED...
<tr><td class="e">extension_dir</td><td class="v">C:\PHP\ext\</td><td class="v">C:\PHP\ext\</td></tr>
<tr><td class="e">fiber.stack_si...TRUNCATED...<tr><td class="e">upload_tmp_dir</td><td class="v">C:\W
indows\temp</td><td class="v">C:\Windows\temp</td></tr>
<tr><td class="e">user_dir</...TRUNCATED...lass="e">Openssl default config </td><td class="v">C:\Program
Files\Common Files\SSL/openssl.cnf </td></t...TRUNCATED...><td class="e">session.save_path</td><td class="v">C:\W
indows\temp</td><td class="v">C:\Windows\temp</td></tr>
<tr><td class="e">session.se...TRUNCATED...
```

File Names: • http://137.116.132.150:80/phpinfo.php

Low **HTML5: CORS Unsafe Methods Allowed**

Summary:

A resource on the target domain has been found to be shared using CORS. The Access-Control-Allow-Methods header, as reflected in the preflight response for the requested resource, indicates that it allows unsafe HTTP methods. An attacker can use HTTP methods such as PUT or DELETE to make unexpected modifications to shared resource and pose a security threat to the overall site security. A user agent rejects any request for this resource with an HTTP method other than the ones that are listed in the Access-Control-Allow-Methods response header.

Cross-Origin Resource Sharing, commonly referred to as CORS, is a technology that allows a domain to define a policy for its resources to be accessed by a web page hosted on a different domain using cross domain XML HTTP Requests (XHR). Historically, the browser restricts cross domain XHR requests to abide by the same origin policy. At its basic form, the same origin policy sets the script execution scope to the resources available on the current domain and prohibits any communication to domains outside this scope. Therefore, execution and incorporation of remote methods and functions hosted on domains outside of the current domain are effectively prohibited. While CORS is supported on all major browsers, it also requires that the domain correctly defines the CORS policy in order to have its resources shared with another domain. These restrictions are managed by access policies typically included in specialized response headers, such as:

- Access-Control-Allow-Origin
- Access-Control-Allow-Headers
- Access-Control-Allow-Methods

Implication:

Allowing unsafe HTTP methods such as PUT or DELETE allows opportunities that can be exploited for modification of the resource shared using CORS. An attacker can inject malicious code, backdoor, deface site, or delete and lock resources to cause denial of service attacks.

Fix:

Review your Cross-Origin-Resource-Sharing policy and consider removing unsafe http methods from Access-Control-Allow-Methods list.

Example 1:
An example of IIS server configuration for listing methods the application is allowed to communicate with.

```
<configuration>
    <system.webServer>
        <httpProtocol>
            <customHeaders>
                <add name="Access-Control-Allow-Methods" value="GET,POST,OPTIONS,HEAD" />
            </customHeaders>
        </httpProtocol>
    </system.webServer>
 </configuration>
```

Example 1 shows how to configure CORS headers at server level.

However the preferred method is to make use of the API of the language used to code the application and provide this access on each resource shared. Here are some programmatic samples by language:

- **.NET:**
  Append Header:
  Response.AppendHeader("Access-Control-Allow-Methods", "GET,POST,OPTIONS,HEAD");

  Check for cross domain XHR request:
  if((Request.Headers["X-Requested-With"] == "XMLHttpRequest") && Request.Headers["Origin"] != null))

- **Java:**
  response.addHeader("Access-Control-Allow-Methods", "GET,POST,OPTIONS,HEAD");

  check for cross domain XHR request:
  if((request.getHeader("X-Requested-With") == "XMLHttpRequest") && request.getHeader("Origin")!= null))

- **PHP:**
  header('Access-Control-Allow-Methods: GET,POST,OPTIONS,HEAD');
  ?>

  Check for cross domain XHR request:
  If(isset($_SERVER['HTTP_X_REQUESTED_WITH']) && ($_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest')
  && isset($_SERVER['Origin']))

Reference:

OWASP HTML 5 Security Cheat Sheet
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Cross-Origin Resource Sharing
http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
http://www.w3.org/TR/cors/

Same Origin Policy
http://en.wikipedia.org/wiki/Same_origin_policy

WebDAV
http://tools.ietf.org/html/rfc2518
http://www.webdav.org/

Attack Request:

```
OPTIONS /STSBidding/service/Login/verify.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Pragma: no-cache
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Pingsession
Origin: http://webinspect.microfocus.com
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="2AECAA314383B17EC165617F8758A772";PSID="2FA10BC01B7459247862701A2C3743C6";Ses
sionType="AuditAttack";CrawlType="None";AttackType="Other";OriginatingEngineID="822a8e1c-b895-4666-a9d2-
026b0a4716c9";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="1128
1";Engine="Html5+Cross+Origin+Options+Request";SmartMode="4";tht="11";
X-RequestManager-Memo: sc="1";rid="01d135f2";
X-Request-Memo: rid="56dc030d";sc="1";thid="180";
```

Attack Response:

```
HTTP/1.1 405 Method Not Allowed
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
```

```
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE,PATCH
Access-Control-Max-Age: 3600
Access-Control-Allow...TRUNCATED...
```

File Names:
- http://137.116.132.150:80/STSBidding/service/Login/verify.php

Low

HTML5: Overly Permissive CORS Policy

Summary:

A resource on the target website has been found to be shared across websites using CORS with an open access control policy.

Cross-Origin Resource Sharing, commonly referred to as CORS, is a technology that allows a domain to define a policy for its resources to be accessed by a web page hosted on a different domain using cross domain XML HTTP Requests (XHR). Historically, the browser restricts cross domain XHR requests to abide by the same origin policy. At its basic form, the same origin policy sets the script execution scope to the resources available on the current domain and prohibits any communication to domains outside this scope. While CORS is supported on all major browsers, it also requires that the domain correctly defines the CORS policy in order to have its resources shared with another domain. These restrictions are managed by access policies typically included in specialized response headers, such as:

- Access-Control-Allow-Origin
- Access-Control-Allow-Headers
- Access-Control-Allow-Methods

A domain includes a list of domains that are allowed to make cross domain requests to shared resources in Access-Control-Allow-Origin header. This header can have either list of domains or a wildcard character ("*") to allow all access. Having a wildcard is considered overly permissive policy.

Implication:

An overly permissive CORS policy can allow a malicious application to communicate with the victim application in an inappropriate way, leading to spoofing, data theft, relay and other attacks. It can open possibilities for entire domain compromise. For example, let's say a Resource is located on a private intranet and a universal access policy is created with the intent that only other intranet domains can reach it. Subsequently, an internal employee browses to an Internet resource that includes a malicious embedded JavaScript that enumerates the private resource and enables external accessibility; effectively exposing it to the Internet. If the resource discloses any sensitive information, this attack can quickly escalate into an unintentional breach of sensitive information.

Fix:

Review your Cross-Origin-Resource-Sharing policy and consider restricting access to only trusted domains. Never use wildcard open-access permissions (e.g. "*") in the Access-Control-Allow-Origin header. Additionally, do not automatically include Access -Control-Allow-Origin headers in the response unless the request is cross-domain. Alternatively, implement an allow list of known domains that are allowed to access this domain and only include domains that actually tried to access the resource. Otherwise, reject the request and reply with only host domain not exposing all allowed domains. Reserve the use of CORS for resources that cannot be shared in other ways (e.g. JavaScript can be accessed using SCRIPT tag as well as images can be accessed using IMG tag from other domains). Finally, make sure that this resource does not disclose any sensitive information and only share resources required to preserve functionality in contrast to an open domain CORS access.

Example 1:
An example of IIS server configuration for listing domains the application is allowed to communicate with.
```
<configuration>
    <system.webServer>
        <httpProtocol>
            <customHeaders>
                <add name="Access-Control-Allow-Origin" value="www.trusted.com" />
            </customHeaders>
        </httpProtocol>
    </system.webServer>
</configuration>
```

Example 1 shows how to configure CORS headers at the server level; however, the preferred method is to make use of the API of the language used to develop the application and set access permissions at the resource level.
Here are some programmatic samples by language:

- **.NET:**
Append Header:
Response.AppendHeader("Access-Control-Allow-Origin", "www.trusted.com");

Check for cross domain XHR request:
if((Request.Headers["X-Requested-With"] == "XMLHttpRequest") && Request.Headers["Origin"] != null))

- **Java:**
response.addHeader("Access-Control-Allow-Origin", "www.trusted.com");

check for cross domain XHR request:
if((request.getHeader("X-Requested-With") == "XMLHttpRequest") && request.getHeader("Origin")!= null))

- **PHP:**
header('Access-Control-Allow-Origin: www.trusted.com');
?>

Check for cross domain XHR request:
If(isset($_SERVER['HTTP_X_REQUESTED_WITH']) && ($_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest') && isset($_SERVER['Origin']))

Reference:

OWASP HTML 5 Security Cheat Sheet
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Cross-Origin Resource Sharing
http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
http://www.w3.org/TR/cors/

Same Origin Policy
http://en.wikipedia.org/wiki/Same_origin_policy

Attack Request:

```
OPTIONS /STSBidding/service/Login/verify.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Pragma: no-cache
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Pingsession
Origin: http://webinspect.microfocus.com
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="2AECAA314383B17EC165617F8758A772";PSID="2FA10BC01B7459247862701A2C3743C6";Ses
sionType="AuditAttack";CrawlType="None";AttackType="Other";OriginatingEngineID="822a8e1c-b895-4666-a9d2-
026b0a4716c9";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="1128
1";Engine="Html5+Cross+Origin+Options+Request";SmartMode="4";tht="11";
X-RequestManager-Memo: sc="1";rid="01d135f2";
X-Request-Memo: rid="56dc030d";sc="1";thid="180";
```

Attack Response:

```
HTTP/1.1 405 Method Not Allowed
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Con...TRUNCATED...
```

File Names: ● http://137.116.132.150:80/STSBidding/service/Login/verify.php

Low

HTML5: CORS Prolonged Caching of Preflight Response

Summary:

WebInspect has discovered a preflight response that is configured to be cached for a prolonged amount of time. The time a response is allowed to be cached is conveyed using an Access-Control-Max-Age response header and a value more than 30 minutes is considered to be prolonged.

Cross-Origin Resource Sharing, commonly referred to as CORS, is a technology that allows a domain to define a policy for its resources to be accessed by a web page hosted on a different domain using cross domain XML HTTP Requests (XHR). Historically, the browser restricts cross domain XHR requests to abide by the same origin policy. At its basic form, the same origin policy sets the script execution scope to the resources available on the current domain and prohibits any communication to domains outside this scope. Therefore, execution and incorporation of remote methods and functions hosted on domains outside of the current domain are effectively prohibited. While CORS is supported on all major browsers, it also requires that the domain correctly defines the CORS policy in order to have its resources shared with another domain. These restrictions are managed by access policies typically included in specialized response headers, such as:

- Access-Control-Allow-Origin
- Access-Control-Allow-Headers
- Access-Control-Allow-Methods
- Access-Control-Max-Age

The browser generates a preflight OPTIONS request whenever the cross domain request made by the web page is anything other than a simple HTTP request. A GET or POST HTTP request with no special headers or credentials is considered a simple request. A response for a preflight request exposes the server's CORS policy via specialized headers mentioned above. After examining the required permissions, the browser makes the actual request that the web page initially performed. This extra preflight request adds overhead and hence the server can configure its preflight response to be cached.

Implication:

Prolonged caching of a preflight response can pose a security threat as the policy can be updated on the server while a browser will still allow unauthorized access to resources based on the original cached policy.

Fix:

Review your Cross-Origin-Resource-Sharing policy and consider keeping Access-Control-Max-Age value to be under 30 minutes.

Reference:

OWASP HTML 5 Security Cheat Sheet
https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet

Cross-Origin Resource Sharing
http://en.wikipedia.org/wiki/Cross-origin_resource_sharing
http://www.w3.org/TR/cors/

Same Origin Policy
http://en.wikipedia.org/wiki/Same_origin_policy

Attack Request:

```
OPTIONS /STSBidding/service/Login/verify.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Pragma: no-cache
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-Pingsession
Origin: http://webinspect.microfocus.com
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="2AECAA314383B17EC165617F8758A772";PSID="2FA10BC01B7459247862701A2C3743C6";Ses
sionType="AuditAttack";CrawlType="None";AttackType="Other";OriginatingEngineID="822a8e1c-b895-4666-a9d2-
026b0a4716c9";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="1128
1";Engine="Html5+Cross+Origin+Options+Request";SmartMode="4";tht="11";
X-RequestManager-Memo: sc="1";rid="01d135f2";
X-Request-Memo: rid="56dc030d";sc="1";thid="180";
```

Attack Response:

```
HTTP/1.1 405 Method Not Allowed
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE,PATCH
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type, Access...TRUNCATED...
```

File Names:
- http://137.116.132.150:80/STSBidding/service/Login/verify.php

---

Low

Web Server Misconfiguration: Insecure Content-Type Setting

Summary:

Almost all browsers are designed to use a mime sniffing technique to guess the content type of the HTTP response instead of adhering to the Content-Type specified by the application in specific cases or ignoring the content when no mime type is specified. Inconsistencies introduced by the mime sniffing techniques could allow attackers to conduct Cross-Site Scripting attacks or steal sensitive user data. WebInspect has determined that the application fails to instruct the browser to strictly enforce the Content-Type specification supplied in the response.

Web server misconfiguration can cause an application to send HTTP responses with the missing Content-Type header or specify a mime type that does not match up accurately with the response content. When a browser receives such a response, it attempts to programmatically determine the mime type based on the content returned in the response. The mime type derived by the browser, however, might not accurately match the one intended by the application developer. Such inconsistencies have historically allowed attackers to conduct Cross-Site Scripting or data theft using Cascading Style Sheets (CSS) by letting them bypass server-side filters using mime type checking and yet have the malicious payload with misleading mime type specification executed on the client-side due to the browser mime sniffing policies.

Microsoft Internet Explorer (IE) introduced the X-Content-Type-Options: nosniff specification that application developers can include in all responses to ensure that mime sniffing does not occur on the client-side. This protection mechanism is limited to Microsoft Internet Explorer versions 9 and above.

Execution:

. Build a test page that includes a reference to an external JavaScript or CSS resource

. Configure the server to return the external resource with an incorrect mime type specification

. Visit the test page using an old version of Microsoft's Internet Explorer (version IE 8) browser

. Interpretation of the external content as JavaScript or CSS by the browser despite the misleading mime type specification indicates a potential for compromise.

Implication:

By failing to dictate the suitable browser interpretation of the response content, application developers can expose their users to Cross-Site Scripting or information stealing attacks.

Fix:

Configure the web server to always send the X-Content-Type-Options: nosniff specification in the response headers. In addition, ensure that following safety precautions are also put in place:

. Verify that the web server configuration will send the accurate mime type information in the Content-Type header of each HTTP response

. Configure the server to send a default Content-Type of text-plain or application/octet-stream to tackle failure scenarios

. Ensure that appropriate Character Set is specified in the Content-Type header

. Configure the server to send Content-Disposition: attachment; filename=name; for content without an explicit content type specification.

Reference:

Microsoft Internet Explorer:
MIME-Handling Change: X-Content-Type-Options: nosniff
MIME-Handling Changes in Internet Explorer

OWASP:
OWASP Testing Guide Appendix D: Encoded Injection
List of Useful HTTP Headers

CSS Data Theft:
CVE-2010-0654

Attack Request:

```
GET /STSBidding/frontend/login HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="9af2cb75";thid="185";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Last-Modified: Fri, 16 Feb 2024 06:21:07 GMT
Acce...TRUNCATED...
```

File Names: • http://137.116.132.150:80/STSBidding/frontend/login

Low **Cookie Security: Missing SameSite Attribute**

Summary:

The SameSite attribute protects cookies from Cross-Site Request Forgery (CSRF) attacks. The browser automatically appends cookies to every HTTP request made to the site that sets the cookie. Cookies might store sensitive data like session ID and authorization token or site data that is shared between different requests to the same site during a session. An attacker can perform an impersonation attack by generating a request to the authenticated site from a third-party site page loaded on the client machine because the browser automatically appended the cookie to the request.
The SameSite attribute on a cookie allows sites to control that behaviour and prevents browsers from appending the cookie to request if the request is generated from a third-party site page load. The SameSite attribute can have the following three values:

- Strict: When set to Strict, cookies are only sent along with requests upon top level navigation .
- Lax: When set to Lax, cookies are sent with top level navigation from the same host as well as GET requests originated to the host from third-party sites (for example, in iframe, link, href, and so on and the form tag with GET method only).
- None: Cookies are sent in all requests made to the site within the path and domain scope set for the cookie. Requests generated due to form submissions using the POST method are also allowed to send cookies with request.

Please note that cookies that have the SameSite attribute with the value of None must be set with the Secure attribute otherwise the browser rejects the cookies. Additionally, a few specific browser versions reject the SameSite cookie with the None value for example, Chrome versions 51 to 66, versions of the UC Browser on Android prior to version 12.13.2, versions of Safari and embedded browsers on macOS 10.14, and all browsers on iOS 12 reject cookies set with SameSite=None. A suggested workaround for this issue is to set an alternate cookie with a prefix or suffix such as *Legacy*appended to cookiename . Sites can look for this legacy cookie if it does not find a cookie that was set with SameSite=None.

Execution:

Inspect the highlighted cookie value in the HTTP response in the vulnerable session. The cookie is missing the SameSite attribute.

Implication:

Sites that set cookies without the SameSite attribute have an increased risk of CSRF attacks. Using CSRF, an attacker can impersonate a valid user and gain unauthorized access to application functionality. Furthermore, the recent versions of browsers might reject the cookie that is not set with the SameSite attribute.

Fix:

Add the SameSite attribute to all cookies. Starting February 2020, the Chrome browser made SameSite a mandatory attribute for all cookies. Any cookie without the SameSite attribute is either rejected or treated with the default behaviour, which is the same as setting the attribute value to Lax. Therefore, any cookie that must be sent regardless the origination of the request (for example, analytics cookies) should have the SameSite attribute value of none.
Furthermore, we recommend that developers continue to add traditional CSRF mitigations to the site along with SameSite attribute. Many site users might still use older browser versions to access the site. Older browser versions do not understand the SameSite attribute.

Reference:

https://tools.ietf.org/html/draft-ietf-httpbis-rfc6265bis-05
https://www.chromium.org/updates/same-site/incompatible-clients
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite

Attack Request:

```
GET /STSBidding/service/Login/verify.php HTTP/1.1
Host: 137.116.132.150
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://137.116.132.150/STSBidding/frontend/login
Connection: keep-alive
X-WIPP: AscVersion=21.2.0.117
X-RequestManager-Memo: stid="23";stmi="0";Category="EventMacro.StartMacro";MacroName="loginMacro.webmacro";
X-Request-Memo: rid="7b295385";thid="186";
Pragma: no-cache
```

Attack Response:

```
HTTP/1.1 404 Not Found
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Type: application/json; charset=UTF-8
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Vary: Origin
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: OPTIONS,GET,POST,PUT,DELETE,PATCH
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 09:30:44 GMT
Content-Length: 99
Connection: Keep-Alive
Set-Cookie: PHPSESSID=1ce78puf2pghc93vhdpgl2uhbr; path=/

{
  "err": "ไม่เคยลงทะเบียนในระบบ",
  "st
```

File Names: • http://137.116.132.150:80/STSBidding/service/Login/verify.php

Best Practice **Weak Cryptographic Hash**

Summary:

A string of hexadecimal digits matching the length of a cryptographic hash from the MD family was detected. Cryptographic hashes are often used to protect passwords, session information, and other sensitive data. There are multiple hashing algorithms in the MD family. By far the most commonly used algorithm is MD5, though MD4 and MD2 are still used with various public key and digital certificate systems. There are known attacks against MD5, MD4, and MD2. These hashes are also susceptible to Rainbow table attacks unless the input is properly salted. As such the MD family of cryptographic hashing functions should not be considered secure and should only be used in certain situations.

Implication:

Hashes produced by the MD family should only be used for short-lived uses where the hash and/or hashed data is not highly security sensitive, or for uses where uniqueness is not a critical requirement. MD Hashes should not be used for any type of long term application such as verifying the integrity of a file or for password storage.

Fix:

For Development:
The application should only use cryptographically secure hashing algorithms, such as SHA-224, SHA-256, SHA-384, or SHA-512. Hashes representing sensitive data should be salted to reduce the effectiveness of rainbow tables.

For Security Operations:
Implement a security policy that precludes the use of MD5, MD4, or MD2 for cryptographic functionality.

For QA:
Make sure that the application is not relying on MD5, MD4, or MD2 for cryptographic functionality.

Reference:

MD5
http://en.wikipedia.org/wiki/MD5
Cryptographic Salting
http://en.wikipedia.org/wiki/Salt_%28cryptography%29

Attack Request:

```
GET /phpinfo.php HTTP/1.1
Accept: */*
Accept-Encoding: gzip, deflate
Pragma: no-cache
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:83.0) Gecko/20100101 Firefox/83.0
Host: 137.116.132.150
Connection: Keep-Alive
X-WIPP: AscVersion=21.2.0.117
X-Scan-Memo:
Category="Audit.Attack";SID="3C49F0FD019C23669FB4655598CC492E";PSID="85EA878AB0D2D24EABA4DEEF89BEDF8C";S
essionType="AuditAttack";CrawlType="None";AttackType="Probe";OriginatingEngineID="65cee7d3-561f-40dc-b5eb-
c0b8c2383fcb";AttackSequence="0";AttackParamDesc="";AttackParamIndex="0";AttackParamSubIndex="0";CheckId="10478
";Engine="Request+Modify";SmartMode="2";tht="11";
X-RequestManager-Memo: stid="59";stmi="0";sc="1";rid="f2c2d1a1";
X-Request-Memo: rid="d364d141";sc="1";thid="180";
Cookie: PHPSESSID=dbft8al0dm6rl9kto0pqjsm3mt;BCSI-CS-019a43ba5a07a9e5=2
```

Attack Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Server: Microsoft-IIS/10.0
X-Powered-By: PHP/8.1.2
X-Powered-By: ASP.NET
Date: Mon, 19 Feb 2024 11:27:07 GMT
Content-Length: 84715
Connection: Keep-Alive

...TRUNCATED...s="v">Category=&quot;Audit.Attack&quot;;SID=&quot;3C49F0FD019C23669FB4655598CC492E
&quot;;PSID=&quot;85EA878AB0D2D24EABA4DEEF89BEDF8C
&quot;;SessionType=&quot;AuditAttack&quot;;CrawlTy...TRUNCATED...
```

File Names:

- http://137.116.132.150:80/phpinfo.php
- http://137.116.132.150:80/STSBidding/src/Template/JSSweetAlert.js