

SSED, 4aWI, Ali Dadak

Inhaltsverzeichnis

[Brute-Force 3](#_Toc192840428)

[1.1. Was ist Brute-Force: 3](#_Toc192840429)

[1.2. Graphische Darstellung 3](#_Toc192840430)

[1.3. Code: 4](#_Toc192840431)

[Command Injektion 14](#_Toc192840432)

[Was ist command injektion? 14](#_Toc192840433)

[CSRF 18](#_Toc192840434)

[Was ist CSRF? 18](#_Toc192840435)

[Code: 19](#_Toc192840436)

[File Inclusion 20](#_Toc192840437)

[Was ist File Inclusion? 20](#_Toc192840438)

[Code: 21](#_Toc192840439)

[File Upload 22](#_Toc192840440)

[Was ist File upload? 22](#_Toc192840441)

[Code: 23](#_Toc192840442)

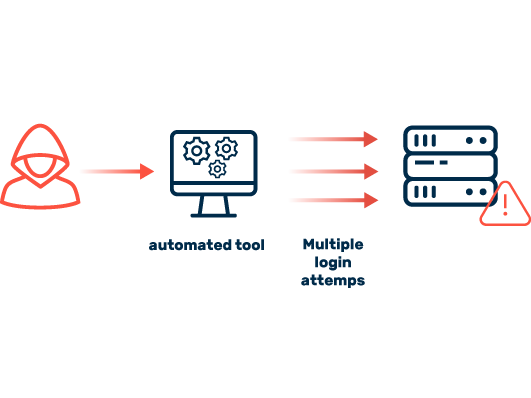
[Literaturverzeichnis 26](#_Toc192840443)

# Brute-Force

* 1. Was ist Brute-Force:  
     Ein Brute Force Angriff (Brute Force Attack auf Englisch) ist eine Methode, die Cyberkriminelle anwenden, um Passwörter und andere Zugangsdaten zu knacken. Bei einem Brute Force Angriff greift ein Angreifer auf eine Liste an häufigen Wörtern zurück und probiert sie der Reihe nach durch, bis

eines funktioniert.  (L1)

## Graphische Darstellung

  
Das Bild zeigt einen Brute-Force-Angriff: Ein Hacker nutzt ein automatisiertes Tool, um mit vielen Login-Versuchen Server anzugreifen, was zu einer Sicherheitswarnung führt.

(L2)

## Code:

**Schlechte Sicherheit**

<?php

if( isset( $\_GET[ 'Login' ] ) ) {

// Get username

$user = $\_GET[ 'username' ];

// Get password

$pass = $\_GET[ 'password' ];

$pass = md5( $pass );

// Check the database

$query = "SELECT \* FROM `users` WHERE user = '$user' AND password = '$pass';";

$result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"], $query ) or die( '<pre>' . ((is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_error($GLOBALS["\_\_\_mysqli\_ston"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' );

if( $result && mysqli\_num\_rows( $result ) == 1 ) {

// Get users details

$row = mysqli\_fetch\_assoc( $result );

$avatar = $row["avatar"]

// Login successful

echo "<p>Welcome to the password protected area {$user}</p>";

echo "<img src=\"{$avatar}\" />";

}

else {

// Login failed

echo "<pre><br />Username and/or password incorrect.</pre>";

}

((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

}

?>

Endscheidende Punkte für schlechte Sicherheit Code:

* Kein Check Loginversuche
* Kein CSRF-Token
* Keine Verhinderung über Zeiteinstieg
* Kein Check für Menschen als Bediener (Captures)
* Keine2FA

**Code mit sehr guter Sicherheit:**

<?php

if( isset( $\_POST[ 'Login' ] ) && isset ($\_POST['username']) && isset ($\_POST['password']) ) {

// Check Anti-CSRF token

checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );

// Sanitise username input

$user = $\_POST[ 'username' ];

$user = stripslashes( $user );

$user = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $user ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

// Sanitise password input

$pass = $\_POST[ 'password' ];

$pass = stripslashes( $pass );

$pass = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

$pass = md5( $pass );

// Default values

$total\_failed\_login = 3;

$lockout\_time = 15;

$account\_locked = false;

// Check the database (Check user information)

$data = $db->prepare( 'SELECT failed\_login, last\_login FROM users WHERE user = (:user) LIMIT 1;' );

$data->bindParam( ':user', $user, PDO::PARAM\_STR );

$data->execute();

$row = $data->fetch();

// Check to see if the user has been locked out.

if( ( $data->rowCount() == 1 ) && ( $row[ 'failed\_login' ] >= $total\_failed\_login ) ) {

// User locked out. Note, using this method would allow for user enumeration!

//echo "<pre><br />This account has been locked due to too many incorrect logins.</pre>";

// Calculate when the user would be allowed to login again

$last\_login = strtotime( $row[ 'last\_login' ] );

$timeout = $last\_login + ($lockout\_time \* 60);

$timenow = time();

/\*

print "The last login was: " . date ("h:i:s", $last\_login) . "<br />";

print "The timenow is: " . date ("h:i:s", $timenow) . "<br />";

print "The timeout is: " . date ("h:i:s", $timeout) . "<br />";

\*/

// Check to see if enough time has passed, if it hasn't locked the account

if( $timenow < $timeout ) {

$account\_locked = true;

// print "The account is locked<br />";

}

}

// Check the database (if username matches the password)

$data = $db->prepare( 'SELECT \* FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );

$data->bindParam( ':user', $user, PDO::PARAM\_STR);

$data->bindParam( ':password', $pass, PDO::PARAM\_STR );

$data->execute();

$row = $data->fetch();

// If its a valid login...

if( ( $data->rowCount() == 1 ) && ( $account\_locked == false ) ) {

// Get users details

$avatar = $row[ 'avatar' ];

$failed\_login = $row[ 'failed\_login' ];

$last\_login = $row[ 'last\_login' ];

// Login successful

echo "<p>Welcome to the password protected area <em>{$user}</em></p>";

echo "<img src=\"{$avatar}\" />";

// Had the account been locked out since last login?

if( $failed\_login >= $total\_failed\_login ) {

echo "<p><em>Warning</em>: Someone might of been brute forcing your account.</p>";

echo "<p>Number of login attempts: <em>{$failed\_login}</em>.<br />Last login attempt was at: <em>{$last\_login}</em>.</p>";

}

// Reset bad login count

$data = $db->prepare( 'UPDATE users SET failed\_login = "0" WHERE user = (:user) LIMIT 1;' );

$data->bindParam( ':user', $user, PDO::PARAM\_STR );

$data->execute();

} else {

// Login failed

sleep( rand( 2, 4 ) );

// Give the user some feedback

echo "<pre><br />Username and/or password incorrect.<br /><br/>Alternative, the account has been locked because of too many failed logins.<br />If this is the case, <em>please try again in {$lockout\_time} minutes</em>.</pre>";

// Update bad login count

$data = $db->prepare( 'UPDATE users SET failed\_login = (failed\_login + 1) WHERE user = (:user) LIMIT 1;' );

$data->bindParam( ':user', $user, PDO::PARAM\_STR );

$data->execute();

}

// Set the last login time

$data = $db->prepare( 'UPDATE users SET last\_login = now() WHERE user = (:user) LIMIT 1;' );

$data->bindParam( ':user', $user, PDO::PARAM\_STR );

$data->execute();

}

// Generate Anti-CSRF token

generateSessionToken();

?>

Endscheidende Punkte für sicheren Code:

* Check Loginversuche
* CSRF-Token
* Verhinderung über Zeiteinstieg
* Check für Menschen als Bediener(Captures)
* 2FA

Tools:

1. **John the Ripper  
   Logo:  
   **

**Was ist das:**John the Ripper is a tool designed to help systems administrators to find weak (easy to guess or crack through brute force) passwords, and even automatically mail users warning them about it, if it is desired.(L4)

1. **Aircrack-ng  
   Logo:  
     
     
     
   Was ist das:**Aircrack-ng is a complete suite of tools to assess WiFi network security. (L6)
2. **Hydra**  
   **Logo:**  
     
   **Was ist das:**  
   Hydra is a parallelized login cracker which supports numerous protocols to attack. It is very fast and flexible, and new modules are easy to add.

This tool makes it possible for researchers and security consultants to show how easy it would be to gain unauthorized access to a system remotely. (L5)

1. **Burpsuite**  
   **Logo:  
     
     
   Was ist das:**Burp Suite is an integrated platform for performing security testing of web applications. Its various tools work seamlessly together to support the entire testing process, from initial mapping and analysis of an application’s attack surface, through to finding and exploiting security vulnerabilities. (L3)
2. **Klassische Programmierung (Java, JS, node…)**

## 

# Command Injektion

## Was ist command injektion?

**Befehlseinschleusung (Command Injection)** ist ein Angriff, bei dem ein Angreifer **beliebige Systembefehle** über eine verwundbare Anwendung auf dem **Host-Betriebssystem** ausführt. Dies geschieht, wenn unsicher bereitgestellte Benutzerdaten an eine **System-Shell** weitergeleitet werden.

Im Gegensatz zur **Code Injection**, bei dem eigenen Code in die Anwendung eingebracht wird, nutzt **Command Injection** bestehende Funktionen, um **Systembefehle auszuführen**. Die Hauptursache ist meist **unzureichende Eingabevalidierung**.  
L7

**Low security:**

|  |
| --- |
| **<?php**  **if( isset( $\_POST[ 'Submit' ] ) ) {**  **// Get input**  **$target = $\_REQUEST[ 'ip' ];**  **// Determine OS and execute the ping command.**  **if( stristr( php\_uname( 's' ), 'Windows NT' ) ) {**  **// Windows**  **$cmd = shell\_exec( 'ping ' . $target );**  **}**  **else {**  **// \*nix**  **$cmd = shell\_exec( 'ping -c 4 ' . $target );**  **}**  **// Feedback for the end user**  **echo "<pre>{$cmd}</pre>";**  **}**  **?>** |

Der PHP-Code ist unsicher, weil:

* **Benutzereingaben** ungefiltert verarbeitet werden.
* Er **Systembefehle** direkt über **shell\_exec()** ausführt.

**Impossible security:**

|  |
| --- |
| **<?php**  **if( isset( $\_POST[ 'Submit' ] ) ) {**  **// Check Anti-CSRF token**  **checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );**  **// Get input**  **$target = $\_REQUEST[ 'ip' ];**  **$target = stripslashes( $target );**  **// Split the IP into 4 octects**  **$octet = explode( ".", $target );**  **// Check IF each octet is an integer**  **if( ( is\_numeric( $octet[0] ) ) && ( is\_numeric( $octet[1] ) ) && ( is\_numeric( $octet[2] ) ) && ( is\_numeric( $octet[3] ) ) && ( sizeof( $octet ) == 4 ) ) {**  **// If all 4 octets are int's put the IP back together.**  **$target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];**  **// Determine OS and execute the ping command.**  **if( stristr( php\_uname( 's' ), 'Windows NT' ) ) {**  **// Windows**  **$cmd = shell\_exec( 'ping ' . $target );**  **}**  **else {**  **// \*nix**  **$cmd = shell\_exec( 'ping -c 4 ' . $target );**  **}**  **// Feedback for the end user**  **echo "<pre>{$cmd}</pre>";**  **}**  **else {**  **// Ops. Let the user name theres a mistake**  **echo '<pre>ERROR: You have entered an invalid IP.</pre>';**  **}**  **}**  **// Generate Anti-CSRF token**  **generateSessionToken();**  **?>** |

**Was verbessert wurde:**

* Bessere Validierung mit filter\_var() zur IP-Prüfung.
* Nutzung von escapeshellarg() zur Escapierung potenziell gefährlicher Zeichen.
* Kein OS-Leak durch php\_uname().
* HTML-Injection-Schutz mit htmlspecialchars().

## Graphische Darstellung: What Is Command Injection? | Examples, Methods & Prevention | Imperva

# CSRF

## Was ist CSRF?

Cross-Site Request Forgery (CSRF) ist ein Angriff, der einen Endbenutzer dazu zwingt, unerwünschte Aktionen auf einer Webanwendung auszuführen, bei der er derzeit authentifiziert ist. Mit ein wenig Hilfe durch Social Engineering (wie dem Versenden eines Links per E-Mail oder Chat) kann ein Angreifer die Benutzer einer Webanwendung dazu bringen, Aktionen nach dem Willen des Angreifers auszuführen. Wenn das Opfer ein normaler Benutzer ist, kann ein erfolgreicher CSRF-Angriff den Benutzer dazu zwingen, statusverändernde Anfragen auszuführen, wie z. B. Geld zu überweisen oder die E-Mail-Adresse zu ändern. Handelt es sich beim Opfer um ein Administratorkonto, kann CSRF die gesamte Webanwendung gefährden.  
L8  
  
Code:

Security Low

<?php

if( isset( $\_GET[ 'Change' ] ) ) {

// Get input

$pass\_new = $\_GET[ 'password\_new' ];

$pass\_conf = $\_GET[ 'password\_conf' ];

// Do the passwords match?

if( $pass\_new == $pass\_conf ) {

// They do!

$pass\_new = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass\_new ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

$pass\_new = md5( $pass\_new );

// Update the database

$current\_user = dvwaCurrentUser();

$insert = "UPDATE `users` SET password = '$pass\_new' WHERE user = '" . $current\_user . "';";

$result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"], $insert ) or die( '<pre>' . ((is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_error($GLOBALS["\_\_\_mysqli\_ston"]) : (($\_\_\_mysqli\_res = mysqli\_connect\_error()) ? $\_\_\_mysqli\_res : false)) . '</pre>' );

// Feedback for the user

echo "<pre>Password Changed.</pre>";

}

else {

// Issue with passwords matching

echo "<pre>Passwords did not match.</pre>";

}

((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);

}

?>  
  
Gründe für die Unsicherheit:  
– Es gibt keinen Schutz gegen Cross-Site Request Forgery.  
– Jeder, der den Benutzer zu einer manipulierten URL bringt (z. B. durch einen Link oder ein verstecktes Formular), kann das Passwort ändern.

## Code:

<?php

if( isset( $\_GET[ 'Change' ] ) ) {

// Check Anti-CSRF token

checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );

// Get input

$pass\_curr = $\_GET[ 'password\_current' ];

$pass\_new = $\_GET[ 'password\_new' ];

$pass\_conf = $\_GET[ 'password\_conf' ];

// Sanitise current password input

$pass\_curr = stripslashes( $pass\_curr );

$pass\_curr = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass\_curr ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

$pass\_curr = md5( $pass\_curr );

// Check that the current password is correct

$data = $db->prepare( 'SELECT password FROM users WHERE user = (:user) AND password = (:password) LIMIT 1;' );

$current\_user = dvwaCurrentUser();

$data->bindParam( ':user', $current\_user, PDO::PARAM\_STR );

$data->bindParam( ':password', $pass\_curr, PDO::PARAM\_STR );

$data->execute();

// Do both new passwords match and does the current password match the user?

if( ( $pass\_new == $pass\_conf ) && ( $data->rowCount() == 1 ) ) {

// It does!

$pass\_new = stripslashes( $pass\_new );

$pass\_new = ((isset($GLOBALS["\_\_\_mysqli\_ston"]) && is\_object($GLOBALS["\_\_\_mysqli\_ston"])) ? mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], $pass\_new ) : ((trigger\_error("[MySQLConverterToo] Fix the mysql\_escape\_string() call! This code does not work.", E\_USER\_ERROR)) ? "" : ""));

$pass\_new = md5( $pass\_new );

// Update database with new password

$data = $db->prepare( 'UPDATE users SET password = (:password) WHERE user = (:user);' );

$data->bindParam( ':password', $pass\_new, PDO::PARAM\_STR );

$current\_user = dvwaCurrentUser();

$data->bindParam( ':user', $current\_user, PDO::PARAM\_STR );

$data->execute();

// Feedback for the user

echo "<pre>Password Changed.</pre>";

}

else {

// Issue with passwords matching

echo "<pre>Passwords did not match or current password incorrect.</pre>";

}

}

// Generate Anti-CSRF token

generateSessionToken();

?>

# File Inclusion

## Was ist File Inclusion?

Eine File-Inclusion-Schwachstelle ist eine Art von Web-Sicherheitslücke, die am häufigsten Webanwendungen betrifft, die auf einer Skript-Laufzeitumgebung basieren. Dieses Problem tritt auf, wenn eine Anwendung einen Pfad zu ausführbarem Code unter Verwendung einer vom Angreifer kontrollierten Variablen erstellt, sodass der Angreifer bestimmen kann, welche Datei zur Laufzeit ausgeführt wird.Eine File-Inclusion-Schwachstelle unterscheidet sich von einem allgemeinen Directory-Traversal-Angriff, da Directory-Traversal den unautorisierten Zugriff auf das Dateisystem ermöglicht, während eine File-Inclusion-Schwachstelle die Art und Weise untergräbt, wie eine Anwendung Code zur Ausführung lädt.Ein erfolgreicher Angriff auf eine File-Inclusion-Schwachstelle führt zu einer Remote-Code-Ausführung auf dem Webserver, auf dem die betroffene Webanwendung läuft. Ein Angreifer kann die Remote-Code-Ausführung nutzen, um eine Webshell auf dem Webserver zu erstellen, die zur Verunstaltung von Webseiten (Website Defacement) verwendet werden kann.

L9

Code:

Security low:  
<?php

// The page we wish to display

$file = $\_GET[ 'page' ];

?>

Warum?

**Ungeprüfte Benutzereingabe**: Direktes Verwenden von $\_GET['page'] ohne Validierung.

**Local File Inclusion (LFI)**: Einbinden von lokalen Dateien, z. B. hochgeladene bösartige Skripte.

**Remote File Inclusion (RFI)**: Ausführen externer Dateien, wenn allow\_url\_include aktiviert ist.

**Fehlende Zugriffsbeschränkung**: Keine Kontrolle, welche Dateien eingebunden werden dürfen.

Impossible:  
  
<?php

// The page we wish to display

$file = $\_GET[ 'page' ];

// Only allow include.php or file{1..3}.php

$configFileNames = [

'include.php',

'file1.php',

'file2.php',

'file3.php',

];

if( !in\_array($file, $configFileNames) ) {

// This isn't the page we want!

echo "ERROR: File not found!";

exit;

}

?>  
Warum?

Nur bestimmte Dateien werden erlaubt.

Code exited nach Fehler.

# File Upload

## Was ist File upload?

Hochgeladene Dateien stellen ein erhebliches Sicherheitsrisiko für Webanwendungen dar. In vielen Angriffen ist der erste Schritt, Schadcode auf das Zielsystem zu übertragen. Sobald der Code hochgeladen ist, muss der Angreifer nur noch einen Weg finden, ihn auszuführen. Datei-Uploads erleichtern diesen ersten Schritt.

Die Folgen eines unkontrollierten Datei-Uploads können schwerwiegend sein – von der vollständigen Übernahme des Systems über die Überlastung des Dateisystems oder der Datenbank bis hin zu Angriffen auf Backend-Systeme, clientseitigen Angriffen oder der Manipulation von Webseiten (Defacement). Die Gefahr hängt davon ab, wie die Anwendung die hochgeladene Datei verarbeitet und wo sie gespeichert wird.

Es gibt zwei Hauptproblembereiche:

1. **Dateimetadaten (Pfad, Dateiname):** Manipulierte Metadaten können die Anwendung dazu bringen, wichtige Dateien zu überschreiben oder an unsicheren Orten zu speichern. Daher müssen diese Informationen sorgfältig geprüft werden.
2. **Dateigröße und -inhalt:** Je nach Nutzung der Datei können verschiedene Angriffe möglich sein. Es ist wichtig, alle Dateiverarbeitungsprozesse genau zu analysieren und sicherzustellen, dass keine unsicheren Interpreter oder Verarbeitungsmethoden verwendet werden.

Um sich zu schützen, sollten Dateien und ihre Metadaten streng validiert und alle Verarbeitungsschritte sicher gestaltet werden.  
L10

Code:  
  
Low:

<?php

if( isset( $\_POST[ 'Upload' ] ) ) {

// Where are we going to be writing to?

$target\_path = DVWA\_WEB\_PAGE\_TO\_ROOT . "hackable/uploads/";

$target\_path .= basename( $\_FILES[ 'uploaded' ][ 'name' ] );

// Can we move the file to the upload folder?

if( !move\_uploaded\_file( $\_FILES[ 'uploaded' ][ 'tmp\_name' ], $target\_path ) ) {

// No

echo '<pre>Your image was not uploaded.</pre>';

}

else {

// Yes!

echo "<pre>{$target\_path} succesfully uploaded!</pre>";

}

}

?>  
**Fehlende Dateityp-Prüfung**: Beliebige Dateitypen (z. B. .php) können hochgeladen und ausgeführt werden.

**Pfad-Manipulation (Directory Traversal)**: Durch spezielle Dateinamen kann der Speicherort manipuliert werden.

**Fehlende Dateigrößenprüfung**: Große Dateien können den Server überlasten (Denial of Service).

**Kein Schutz vor doppelten Dateinamen**: Wichtige Dateien könnten überschrieben werden.

**Fehlende Fehlerprotokollierung:** Angriffsversuche werden nicht erfasst, was die Erkennung erschwert.  
  
Impossible:  
<?php

if( isset( $\_POST[ 'Upload' ] ) ) {

// Check Anti-CSRF token

checkToken( $\_REQUEST[ 'user\_token' ], $\_SESSION[ 'session\_token' ], 'index.php' );

// File information

$uploaded\_name = $\_FILES[ 'uploaded' ][ 'name' ];

$uploaded\_ext = substr( $uploaded\_name, strrpos( $uploaded\_name, '.' ) + 1);

$uploaded\_size = $\_FILES[ 'uploaded' ][ 'size' ];

$uploaded\_type = $\_FILES[ 'uploaded' ][ 'type' ];

$uploaded\_tmp = $\_FILES[ 'uploaded' ][ 'tmp\_name' ];

// Where are we going to be writing to?

$target\_path = DVWA\_WEB\_PAGE\_TO\_ROOT . 'hackable/uploads/';

//$target\_file = basename( $uploaded\_name, '.' . $uploaded\_ext ) . '-';

$target\_file = md5( uniqid() . $uploaded\_name ) . '.' . $uploaded\_ext;

$temp\_file = ( ( ini\_get( 'upload\_tmp\_dir' ) == '' ) ? ( sys\_get\_temp\_dir() ) : ( ini\_get( 'upload\_tmp\_dir' ) ) );

$temp\_file .= DIRECTORY\_SEPARATOR . md5( uniqid() . $uploaded\_name ) . '.' . $uploaded\_ext;

// Is it an image?

if( ( strtolower( $uploaded\_ext ) == 'jpg' || strtolower( $uploaded\_ext ) == 'jpeg' || strtolower( $uploaded\_ext ) == 'png' ) &&

( $uploaded\_size < 100000 ) &&

( $uploaded\_type == 'image/jpeg' || $uploaded\_type == 'image/png' ) &&

getimagesize( $uploaded\_tmp ) ) {

// Strip any metadata, by re-encoding image (Note, using php-Imagick is recommended over php-GD)

if( $uploaded\_type == 'image/jpeg' ) {

$img = imagecreatefromjpeg( $uploaded\_tmp );

imagejpeg( $img, $temp\_file, 100);

}

else {

$img = imagecreatefrompng( $uploaded\_tmp );

imagepng( $img, $temp\_file, 9);

}

imagedestroy( $img );

// Can we move the file to the web root from the temp folder?

if( rename( $temp\_file, ( getcwd() . DIRECTORY\_SEPARATOR . $target\_path . $target\_file ) ) ) {

// Yes!

echo "<pre><a href='{$target\_path}{$target\_file}'>{$target\_file}</a> succesfully uploaded!</pre>";

}

else {

// No

echo '<pre>Your image was not uploaded.</pre>';

}

// Delete any temp files

if( file\_exists( $temp\_file ) )

unlink( $temp\_file );

}

else {

// Invalid file

echo '<pre>Your image was not uploaded. We can only accept JPEG or PNG images.</pre>';

}

}

// Generate Anti-CSRF token

generateSessionToken();

?>

warum?  
**Anti-CSRF-Schutz** – Verhindert **CSRF-Angriffe** durch Token-Überprüfung.

**Dateityp-Prüfung** – Erlaubt nur **JPEG-** und **PNG-Dateien**.

**Größenbeschränkung** – Maximal **100 KB**.

**Metadaten-Entfernung** – Entfernt **bösartigen Code** durch Neu-Kodierung.

**Löschung temporärer Dateien** – **Aufräumen** nach Upload schützt vor Datenlecks.

# Captcha

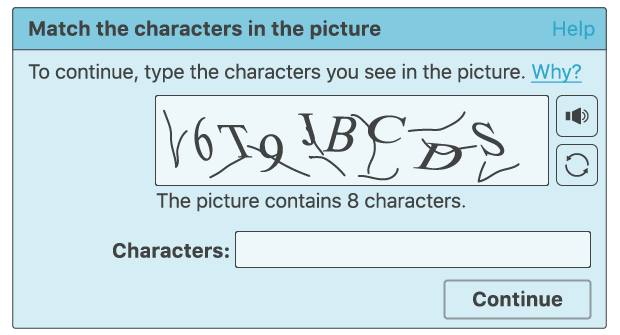
## Was ist ein Captcha?

CAPTCHA-Tests unterscheiden zwischen Menschen und Bots, indem sie automatisierte Herausforderungen stellen. Sie erscheinen an bestimmten Stellen auf Websites und genehmigen oder blockieren Nutzer automatisch. Trotz ihrer Funktion zur Bot-Abwehr sind CAPTCHAs selbst automatisierte Systeme.

L11

## Wie Funktionieren Captchas?

Klassische CAPTCHAs fordern Nutzer auf, verzerrte Buchstaben zu identifizieren, um Bots zu blockieren. Menschen können diese Buchstaben meist erkennen, während Bots scheitern oder durch Zufallsversuche wenig Erfolg haben. Da fortgeschrittene Bots solche Tests zunehmend umgehen können, werden sie durch komplexere Methoden wie Google reCAPTCHA ersetzt.  
L11



## Vor- und Nachteile:

Vorteile:

* Effektive-Bot-Abwehr
* Vielfältige Einsatzmöglichkeiten

Nachteile:

* Nicht 100% sicher (KI-Modelle können Captchas umgehen)
* Zeitaufwand für Nutzer

# SQL-Injection

## Was ist SQL-Injection?

Ein **SQL-Injection-Angriff** erfolgt durch das Einschleusen von SQL-Befehlen über die Eingabedaten einer Anwendung. Erfolgreiche Angriffe können sensible Daten auslesen, Datenbankeinträge verändern oder löschen, administrative Befehle ausführen, Dateien aus dem Datenbanksystem abrufen und sogar Betriebssystembefehle ausführen. SQL-Injection ist eine spezielle Form der **Injection-Angriffe**, bei denen manipulierte SQL-Befehle die reguläre Ausführung von Datenbankabfragen beeinflussen.  
L12

## Code

**Security Low:**

$id = $\_GET['id'];

$query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id'";

$result = mysqli\_query($db, $query);

while ($row = mysqli\_fetch\_assoc($result)) {

echo "Name: " . $row['first\_name'] . " " . $row['last\_name'];

}

**Unsicher da:**

Der Parameter „id“ wird direkt in die SQL- Abfrage eingefügt

SELECT first\_name, last\_name FROM users WHERE user\_id = '1' OR '1'='1'  
Da 1=1 immer wahr ist, gibt die Datenbank alle Benutzer aus.

**Security Impossible**

if (isset($\_GET['id'])) {

$id = $\_GET['id'];

// Datenbankverbindung

$db = new mysqli("localhost", "user", "password", "dvwa");

// Prepared Statement verwenden

$query = $db->prepare("SELECT first\_name, last\_name FROM users WHERE user\_id = ?");

$query->bind\_param("i", $id); // "i" steht für Integer, um nur Zahlen zu akzeptieren

$query->execute();

$result = $query->get\_result();

while ($row = $result->fetch\_assoc()) {

echo "Name: " . htmlspecialchars($row['first\_name']) . " " . htmlspecialchars($row['last\_name']);

}

// Ressourcen schließen

$query->close();

$db->close();

}

**Sicher, weil:**Es wird verhindert, dass SQL-Befehle manipuliert werden

Es wird verhindert, dass der Benutzer Strings eingibt.

# Literaturverzeichnis

LINKS:  
  
L1  
<https://www.proofpoint.com/de/threat-reference/brute-force-attack#:~:text=Ein%20Brute%20Force%20Angriff%20(Brute,nach%20durch%2C%20bis%20eines%20funktioniert.>  
  
L2

<https://www.myrasecurity.com/de/brute-force-attacke/>

L3  
<https://www.kali.org/tools/burpsuite/>

L4  
<https://www.kali.org/tools/john/>

L5  
<https://www.kali.org/tools/hydra/>

L6  
<https://www.aircrack-ng.org/>

L7

<https://owasp.org/www-community/attacks/Command_Injection>

L8

<https://owasp.org/www-community/attacks/csrf>

L9

<https://en.wikipedia.org/wiki/File_inclusion_vulnerability>

L10  
<https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload>

L11  
<https://www.cloudflare.com/de-de/learning/bots/how-captchas-work/>

L12  
<https://owasp.org/www-community/attacks/SQL_Injection>