Abiva Wayne

DVWA-Sytems

Inhaltsverzeichnis

[1. Brute-Force 2](#_Toc194341616)

[1.1 Information 2](#_Toc194341617)

[1.2 Graphische Darstellungen 3](#_Toc194341618)

[1.3 Code 4](#_Toc194341619)

[1.4 Tools zum Hacken und Testen 8](#_Toc194341620)

[1.5 Umsetzung 8](#_Toc194341621)

[2. Command Injection 9](#_Toc194341622)

[2.1 Information 9](#_Toc194341623)

[2.1.1 Definition 9](#_Toc194341624)

[2.2 Grafik 9](#_Toc194341625)

[2.3 Code 10](#_Toc194341626)

[3. CSRF 13](#_Toc194341627)

[3.1 Informationen 13](#_Toc194341628)

[3.1.1 Definition 13](#_Toc194341629)

[3.1.2 Schutz vor CSRF 13](#_Toc194341630)

[3.2 Grafik 13](#_Toc194341631)

[3.3 Code 14](#_Toc194341632)

[4. File Inclusion 17](#_Toc194341633)

[4.1 Informationen 17](#_Toc194341634)

[4.1.1 Definition 17](#_Toc194341635)

[4.2 Grafik 17](#_Toc194341636)

[4.3 Code 18](#_Toc194341637)

[5. File Upload 19](#_Toc194341638)

[5.1 Informationen 19](#_Toc194341639)

[5.2 Code 19](#_Toc194341640)

# Brute-Force

## Information

**Brute Force** ist eine Cyberkriminelle Angriffsmethode, die mit **automatisierten Tools** und **leistungsstarken Hardware**, in einer kurzen Zeit Login-Daten geknackt werden können.[[1]](#footnote-1)

**Man unterscheidet zwischen mehrere Methoden:**[[2]](#footnote-2)

**Traditonell**: Alle möglichen Kombinationen zwischen Benutzernamen und Passwörter werden ausprobiert, mithilfe leistungsstarken Hardware .

**Credential Stuffing**: Bereits bekannte Login-Daten (E-Mail und Passwort) werden auf verschiedenen Diensten getestet, um von der Wiederverwendung derselben Kombination durch Benutzer zu profitieren. Botnetze beschleunigen und verschleiern diese Angriffe.

**Credential Cracking**: Teilweise bekannte Zugangsdaten (z. B. nur der Benutzername) werden durch Erraten des Passworts ergänzt, entweder mit Passwortlisten oder durch Zufallsgenerierung. Auch hier kommen Botnetze zum Einsatz.

**Rainbow-Table Angriff**: Gehasht gespeicherte Passwörter werden mit vordefinierten Hashwerten in sogenannten Rainbow Tables abgeglichen, um das ursprüngliche Passwort zu finden.

**Dictionary Angriff**: Listen mit häufig verwendeten Passwörtern und Benutzernamen werden durchprobiert, um Zugriff auf Konten zu erhalten.

**Wie schützt man sich gegen Brute-Force:[[3]](#footnote-3)**

**Für Benutzer:**

* **Starke Passwörter**: Verwenden Sie Passwörter mit mindestens 15 Zeichen, die keine leicht erratbaren Informationen wie Wörter oder Geburtsdaten enthalten. Passwort-Manager helfen bei der Verwaltung sicherer Passwörter.
* **Zwei-Faktor-Authentifizierung (2FA)**: Aktivieren Sie immer die 2FA, um einen zusätzlichen Schutz zu gewährleisten.
* **Verschlüsselte Kommunikation**: Nutzen Sie HTTPS oder andere verschlüsselte Protokolle, um die Sicherheit der Datenübertragung zu gewährleisten.
* **Passwortlose Methoden**: Setzen Sie auf Technologien wie FIDO2, die biometrische Verfahren oder Hardware-Authentifizierung verwenden.

**Für Administratoren:**

* **Anmeldeversuche begrenzen**: Legen Sie fest, dass Konten nach einer bestimmten Anzahl fehlerhafter Logins vorübergehend gesperrt werden.
* **Standardports ändern**: Ändern Sie den Standardport (z. B. für SSH von 22 auf einen anderen zwischen 1024 und 65536), um Angreifern die Suche zu erschweren.
* **Remote Desktop-Zugriff beschränken**: Erlauben Sie den Zugang nur für ausgewählte IP-Adressen oder implementieren Sie ein Zero-Trust-Modell mit 2FA, Firewalls und Geoblocking.
* **CAPTCHA verwenden**: Verhindern Sie automatisierte Angriffe durch die Integration von CAPTCHA-Anfragen.
* **Unbenutzte Accounts löschen**: Entfernen Sie inaktive Konten, die veraltete Anmeldedaten enthalten könnten.
* **Traffic überwachen**: Überprüfen Sie Logdateien regelmäßig, um verdächtige Aktivitäten wie wiederholte Anmeldeversuche zu erkennen, und sperren Sie auffällige IP-Adressen.
* **Passwortverschlüsselung**: Speichern Sie Passwörter nur verschlüsselt (z. B. mit 256-Bit-Algorithmen), um Credential Stuffing zu verhindern.

## Brute-Force-Attacke: Definition und Funktionsweise | MyraGraphische Darstellungen

[[4]](#footnote-4)

## Code

**Low - Brute Force**

<?php

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

**Überprüft, ob der Login-Button über GET übermittelt wurde.**

// Get username

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

// Get password

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

$pass = md5( $pass );

**Holt den Benutzernamen und das Passwort aus der URL. Das Passwort wird mit MD5 gehasht.**

// Check the database

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

**Erstellt eine SQL-Abfrage, die prüft, ob es einen Benutzer mit dem angegebenen Namen und Passwort gibt. Benutzername und Passwort werden direkt in den SQL-String eingefügt.**

$result = mysqli\_query($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>' );

**Führt die Abfrage gegen die Datenbank aus. Gibt bei Fehlern eine Fehlermeldung aus.**

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

**Wenn genau ein Ergebnis gefunden wurde, ist der Login erfolgreich.**

// 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}\" />";

**Holt Benutzerdetails und zeigt den Avatar sowie eine Begrüßung an.**

}

else {

// Login failed

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

**Gibt eine Fehlermeldung aus, wenn Login fehlschlägt.**

}

((is\_null($\_\_\_mysqli\_res = mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]))) ? false : $\_\_\_mysqli\_res);  
**Verbindet sich sauber von der Datenbank.**

**Der Code erlaubt den Login-Versuch über GET-Parameter ohne jegliche Schutzmechanismen. Weder gibt es eine Begrenzung der Login-Versuche noch eine Zeitverzögerung, was Brute-Force-Angriffe sehr einfach macht. Die Eingaben werden zudem ungefiltert direkt in die SQL-Abfrage eingesetzt, was SQL-Injection-Angriffe ermöglicht. Somit bietet dieser Code keinerlei effektive Sicherheitsmaßnahmen gegen automatisierte oder manipulierte Login Versuche.**

**Impossible Brute Force:**  
<?php

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

// Check Anti-CSRF token

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

**Verhindert Cross-Site Request Forgery (CSRF)-Angriffe. Es überprüft, ob der übermittelte user\_token mit dem in der Session gespeicherten session\_token übereinstimmt. Ohne dieses Token könnte ein Angreifer das Opfer unbemerkt zum Ausführen von Aktionen zwingen (z. B. über einen manipulierten Link).**

// 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)) ? "" : ""));

**Entfernt Backslashes (\) aus der Eingabe (stripslashes).**

**Wandelt spezielle Zeichen in eine sichere Form um, um SQL-Befehle im Kontext einer Datenbankabfrage zu verhindern (mysqli\_real\_escape\_string).**

// 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 );

**Entfernt Backslashes (\) aus der Passwort-Eingabe. Wandelt Sonderzeichen um, um eine sichere Speicherung und Abfrage in der Datenbank zu ermöglichen. Verschlüsselt das Passwort mit einem Hashing-Algorithmus (md5), bevor es mit gespeicherten Werten verglichen wird.**

// Default values

$total\_failed\_login = 3;

$lockout\_time = 15;

$account\_locked = false;

**Setzt eine maximale Anzahl erlaubter fehlgeschlagener Anmeldeversuche auf 3. Definiert eine Sperrzeit von 15 Minuten, falls das Limit überschritten wird.**

// 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();

**Fragt die Anzahl der fehlgeschlagenen Login-Versuche und die Zeit des letzten Logins des Benutzers aus der Datenbank ab.**

// 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 />";

}

}

**Falls der Benutzer mehr als die erlaubten Fehlversuche erreicht hat, wird geprüft, ob die Sperrzeit bereits abgelaufen ist. Falls nicht, wird das Konto temporär gesperrt.**

// 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();

**Prüft, ob der Benutzername und das gehashte Passwort mit einem Datensatz in der Datenbank übereinstimmen.**

// If its a valid login...

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

**Falls der Benutzer gefunden wurde und das Konto nicht gesperrt ist, wird der Login gewährt.**

// 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}\" />";

**Zeigt eine Begrüßung und das Benutzerbild an.**

// 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>";

}

**Falls es vorher fehlgeschlagene Login-Versuche gab, wird eine Warnung ausgegeben.**

// Reset bad login count

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

**Setzt den Zähler für fehlgeschlagene Versuche zurück.**

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

$data->execute();

} else {

// Login failed

sleep( rand( 2, 4 ) );

**Verzögert die Antwortzeit um 2 bis 4 Sekunden, um Brute-Force-Angriffe zu verlangsamen.**

// 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;' );

**Erhöht den Zähler für fehlgeschlagene Login-Versuche.**

$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;' );

**Aktualisiert das Feld last\_login mit der aktuellen Zeit.**

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

$data->execute();

}

// Generate Anti-CSRF token

generateSessionToken();

**Generiert ein neues CSRF-Token für die Sitzung.**

?>

**Der Code schützt Login-Vorgänge durch mehrere sinnvolle Maßnahmen. Er prüft, ob ein CSRF-Token gültig ist, was Manipulationen über fremde Webseiten verhindert. Die Benutzereingaben für Username und Passwort werden von potenziell schädlichem Inhalt bereinigt und sicher für Datenbankabfragen vorbereitet, wodurch SQL-Injection erschwert wird. Außerdem wird das Passwort vor dem Vergleich mit der Datenbank gehasht, sodass es nicht im Klartext verarbeitet wird. Der Code überwacht die Anzahl fehlgeschlagener Login-Versuche und sperrt ein Benutzerkonto nach drei Fehlern für 15 Minuten. Durch eine zusätzliche zeitliche Verzögerung nach falschem Login wird automatisiertes Durchprobieren weiter erschwert. Diese Mechanismen sorgen dafür, dass unerlaubte Zugriffe durch Brute-Force-Angriffe deutlich schwerer umzusetzen sind.**

Entscheidende Punkte für sicheren Code sind:

1. **Check Login versuche!!**
2. **CSRF-Token**
3. **Verhinderung über Zeiteinstieg**
4. **Check auf MENSCHEN ALS Bedient (Captures)**
5. **2FA**

## Tools zum Hacken und Testen

* Hydra
* JohnTheRipper
* Burpsuite
* Klassisches Programmierung: Java, JavaScript, node …

## Umsetzung

Für die Umsetzung haben wir die Sicherheit auf Low eingestellt. Die Tools die benutzt wurden ist einfaches JavaScript Code mit Node.js.

# Command Injection[[5]](#footnote-5)

## Information

### Definition

Bei einer Command Injection handelt es sich um eine Angriffsart, bei der ein Hacker versucht beliebige Systembefehle auf dem Server auszuführen, indem er eine Schwachstelle in einer Anwendung ausgenutzt.

Das passiert, wenn eine Anwendung unsichere Benutzereingabe (z.B. aus Formularen, Cookies oder HTTP-Headern) ohne ausreichende Prüfung an die System-Shell weitergeleitet. In diesem Fall kann ein Angreifer eigenen Befehle einschleusen, die dann mit dem Rechten der betroffenen Anwendung ausgeführt werden. Die Hauptursache für diesen Angriff ist eine unzureichende Eingabeprüfung durch die Anwendung.

Es gäbe auch noch die Code Injection, wo der Angreifer eigenen Programmcode in die Anwendung einbringt und ausführt. Der Angreifer verwendet bestehende Funktionen um zusätzliche Systembefehle auszuführen ohne den eigenen Programmcode in die Anwendung einzuschleusen.

## Grafik

[[6]](#footnote-6)Ein Bild, das Screenshot, Text, Diagramm, Design enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Code

**Low – Command Injection**

<?php

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

**Prüft, ob das Formular mit dem Namen Submit über POST abgeschickt wurde.**

// Get input

$target = $\_REQUEST[ 'ip' ];

**Liest den vom Benutzer eingegebenen Wert aus dem Feld ip.**

**$\_REQUEST umfasst sowohl $\_GET als auch $\_POST.**

// Determine OS and execute the ping command.

if( stristr( php\_uname( 's' ), 'Windows NT' ) ) {

**php\_uname('s') gibt das Betriebssystem zurück (z. B. "Windows NT", "Linux").**

**stristr(..., 'Windows NT') prüft, ob es sich um ein Windows-System handelt.**

// Windows

$cmd = shell\_exec( 'ping ' . $target );

**Führt den ping-Befehl für die angegebene IP-Adresse (oder Hostname) aus, standardmäßig unter Windows.**

}

else {

// \*nix

$cmd = shell\_exec( 'ping -c 4 ' . $target );

**Führt den ping-Befehl mit der Option -c 4 aus, die 4 Ping-Pakete sendet.**

}

// Feedback for the end user

echo "<pre>{$cmd}</pre>";

**Gibt das Ergebnis des Ping-Kommandos in einem** <pre>**-Block aus (damit das Format erhalten bleibt).**

}

?>

**Der Code ist problematisch, weil er Benutzereingaben ungeprüft direkt in ein Shell-Kommando einfügt und anschließend ausführt. Dadurch kann ein Angreifer über das Eingabefeld beliebige Systembefehle einschleusen und ausführen lassen, was als Command Injection bezeichnet wird. Es fehlt jegliche Validierung oder Einschränkung der Eingabe, sodass das System durch einfache Tricks kompromittiert werden kann. Da der Befehl über** shell\_exec **läuft, erhält der Angreifer sogar direkten Zugriff auf die Rückgabe der ausgeführten Kommandos. Ohne Schutzmechanismen wie Eingabeprüfung, Whitelisting oder spezielle Funktionen zur sicheren Ausführung von Systembefehlen stellt der Code ein hohes Sicherheitsrisiko dar.**

**Impossible – Command Injection**

<?php

if( isset( $\_POST[ 'Submit' ] ) ) { **Prüft, ob das Formular mit dem Submit-Button abgeschickt wurde.**

// Check Anti-CSRF token

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

**Überprüft, ob der mitgesendete CSRF-Token mit dem im Server gespeicherten Token übereinstimmt; gleichzeitig schützt es vor Cross-Site Request Forgery.**

// Get input

$target = $\_REQUEST[ 'ip' ];

$target = stripslashes( $target );

**Liest die eingegebene IP-Adresse aus dem Formular und entfernt Backslashes aus der Eingabe.**

// Split the IP into 4 octects

$octet = explode( ".", $target );

**Teilt die IP-Adresse an jedem Punkt . in ein Array mit vier Werten (z. B. "192.168.0.1" → [192,168,0,1]).**

// 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 ) ) {

**Verifiziert, dass alle vier Werte aus der IP-Adresse gültige Zahlen sind und dass es genau vier sind.**

// If all 4 octets are int's put the IP back together.

$target = $octet[0] . '.' . $octet[1] . '.' . $octet[2] . '.' . $octet[3];

**Baut die IP-Adresse neu aus den geprüften Zahlen zusammen.**

// Determine OS and execute the ping command.

if( stristr( php\_uname( 's' ), 'Windows NT' ) ) {

**Erkennt, ob das Skript auf einem Windows-System läuft.**

// Windows

$cmd = shell\_exec( 'ping ' . $target );

}

else {

// \*nix

$cmd = shell\_exec( 'ping -c 4 ' . $target );

}

**Führt den Ping-Befehl mit der validierten IP-Adresse aus – je nach Betriebssystem unterschiedlich.**

// 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();

?>#

**Der Code schützt sich effektiv gegen Command Injection, indem er die Benutzereingabe strikt überprüft und nur dann ausführt, wenn es sich um eine gültige numerische IPv4-Adresse handelt. Durch die Aufteilung in vier Oktette und die explizite Prüfung auf numerische Werte wird verhindert, dass zusätzliche Shell-Befehle eingeschleust werden. Der Einsatz eines Anti-CSRF-Tokens bietet zusätzlich Schutz gegen ungewollte oder automatisierte Formularangriffe von anderen Webseiten.**

# CSRF

## Informationen

### Definition[[7]](#footnote-7)

Cross-Site Request Forgery (CSRF) ist eine Angriffsmethode, bei der ein Hacker einen Nutzer dazu bringt, in einer Webanwendung unbeabsichtigte Aktionen auszuführen, in der er gerade angemeldet ist.

Der Angreifer nutzt Social Engineering, um das Opfer dazu zu bringen, eine bestimmte Aktion auszuführen. Wenn der Nutzer bereits in der Zielanwendung angemeldet ist, werden Anfragen mit seinen Sitzungsdaten automatisch übermittelt. Die Anwendung kann nicht unterschieden, ob die Anfrage absichtlich vom Nutzer gesendet wurde oder von einem Angreifer manipuliert wurde.

### 3.1.2 Schutz vor CSRF

Viele moderne Web-Frameworks bieten eingebaute Schutzmechanismen gegen CSRF, darunter Joomla, Spring, Struts, Ruby on Rails und .NET. Entwickler können auch Tools wie OWASP CSRF Guard für Java oder CSRFProtector für PHP nutzen.

Effektive Schutzmaßnahmen.

* CSRF-Tokens verwenden: Einzigartige und nicht erratbare Token sollten in jeder Anfrage enthalten sein.
* Nur POST-Anfragen für kritische Aktionen zulassen: GET-Anfragen sollten niemals Statusänderungen bewirken.
* Referer-Header validieren: Überprüfen, ob die Anfrage von der richtigen Domain stammt (nicht 100% sicher, aber hilfreich).

## Grafik[[8]](#footnote-8)

* Ein Bild, das Text, Screenshot, Diagramm, Design enthält.

  KI-generierte Inhalte können fehlerhaft sein.Benutzer zur erneuten Authentifizierung zwingen: Besonders bei kritischen Aktionen wie Geldtransfers.

## Code

**Low - CSRF**

<?php

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

**Überprüft, ob die URL den Parameter Change enthält, was bedeutet, dass ein Passwortänderungsformular über GET übermittelt wurde.**

// Get input

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

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

**Liest die beiden Passwortfelder aus der URL: das neue Passwort und die Bestätigung.**

// Do the passwords match?

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

**Vergleicht, ob beide Eingaben identisch sind. Falls ja, wird das neue Passwort gesetzt.**

// 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 );

**Bereinigt das neue Passwort von Sonderzeichen für die Datenbankabfrage.Verschlüsselt das Passwort mit MD5.**

// 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);

}

?>

**Der Code erlaubt die Änderung des Passworts, indem er zwei Felder über GET ausliest und die Eingabe überprüft. Wenn die Passwörter übereinstimmen, wird das neue Passwort in der Datenbank gespeichert. Allerdings fehlt ein Schutzmechanismus wie ein CSRF-Token, weshalb ein externer Angreifer das Opfer durch eine präparierte URL dazu bringen könnte, unbemerkt das Passwort zu ändern. Da der Vorgang allein durch Aufruf einer URL ausgelöst werden kann, ist der Code besonders anfällig für CSRF-Angriffe.**

**Impossible – CSRF**  
<?php

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

**Prüft, ob der Parameter Change in der URL gesetzt ist, was auf einen Passwortänderungsversuch hinweist.**

// Check Anti-CSRF token

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

**Hier wird der übermittelte Token mit dem in der Sitzung gespeicherten Token verglichen. Stellt sicher, dass die Anfrage nicht von einer fremden Quelle manipuliert wurde (Schutz vor CSRF).**

// Get input

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

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

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

**Holt das aktuelle Passwort, das neue Passwort und dessen Bestätigung aus der URL.**

// 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 );

**Entfernt Backslashes und bereinigt die Eingabe für die Datenbank. Verschlüsselt das aktuelle Passwort mit MD5.**

// 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();

**Prüft, ob das eingegebene aktuelle Passwort mit dem in der Datenbank gespeicherten übereinstimmt.**

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

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

**Passwortänderung wird nur ausgeführt, wenn:**

* **Neue Passwörter übereinstimmen.**
* **Das alte Passwort korrekt war.**

// 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 );

**Bereinigt das neue Passwort und wandelt es in einen Hash um.**

// 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();

**Speichert das neue Passwort für den aktuellen Benutzer in der Datenbank.**

// 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();?>

**Der Code schützt zuverlässig vor CSRF-Angriffen, indem er einen CSRF-Token prüft, der mit der Benutzeranfrage mitgeschickt wird. Zusätzlich muss der Benutzer sein aktuelles Passwort korrekt eingeben, bevor das neue Passwort gespeichert wird. Das neue Passwort wird nur übernommen, wenn es mit der Bestätigung übereinstimmt. Alle Passworteingaben werden bereinigt und verschlüsselt, bevor sie verwendet werden. Durch diese Kombination aus Token-Prüfung, Passwort-Verifizierung und Eingabekontrolle wird ein sicherer Passwortwechsel ermöglicht.**

# File Inclusion

## Informationen

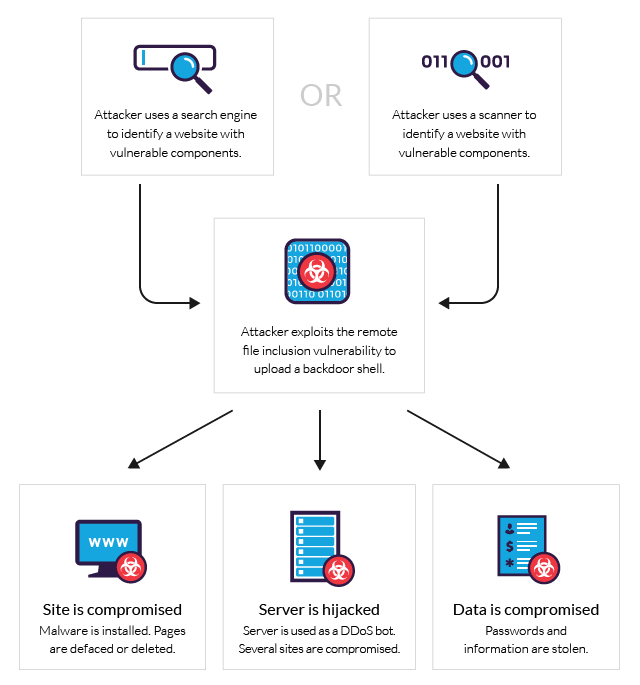
### Definition[[9]](#footnote-9)

Eine **File Inclusion**-Schwachstelle tritt auf, wenn eine Webanwendung es einem Angreifer ermöglicht, Dateien auf dem Server einzubinden und auszuführen. Dies geschieht oft durch unzureichend geprüfte Benutzereingaben, die es dem Angreifer erlauben, Pfade zu internen oder externen Dateien anzugeben. Dadurch kann er auf sensible Informationen zugreifen oder schädlichen Code ausführen. Es gibt zwei Haupttypen von File Inclusion:​

* **Local File Inclusion (LFI):** Hierbei nutzt der Angreifer lokale Dateien auf dem Server aus, um beispielsweise Konfigurationsdateien oder Passwörter auszulesen.​
* **Remote File Inclusion (RFI):** In diesem Fall bindet der Angreifer externe Dateien von entfernten Servern ein, was zur Ausführung von Schadcode auf dem Zielserver führen kann.​

Um solche Schwachstellen zu verhindern, sollten Entwickler Benutzereingaben stets validieren und nur explizit erlaubte Dateien einbinden.

## Grafik[[10]](#footnote-10)



## Code

**Low – File Inclusion**

<?php

// The page we wish to display

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

?>

**Die Variable $file erhält direkt den Wert aus der URL (z. B. ?page=include.php). Es gibt keine Einschränkung oder Prüfung, was bedeutet: Jede Datei, deren Pfad angegeben wird, kann eingebunden oder verarbeitet werden, inklusive potenziell schädlicher Dateien.**

**Impossible-File Inclusion**

<?php

// The page we wish to display

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

**Auch hier wird die Datei über die URL gesteuert. Allerdings wird der angeforderte Dateiname gegen eine Whitelist geprüft:**

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

$configFileNames = [

'include.php',

'file1.php',

'file2.php',

'file3.php',

];

**Auch hier wird die Datei über die URL gesteuert. Allerdings wird der angeforderte Dateiname gegen eine Whitelist geprüft:**

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

**Nur wenn der Dateiname genau einer der zugelassenen Werte ist, wird das Skript weiter ausgeführt.**

// This isn't the page we want!

echo "ERROR: File not found!";

exit;

**Diese Validierung verhindert das Einbinden unerwünschter oder gefährlicher Dateien.**

}

?>

**Der Code im „Low – File Inclusion“-Beispiel lässt uneingeschränkten Zugriff auf beliebige Dateien zu, da keine Validierung der Benutzereingabe erfolgt. Dadurch ist die Anwendung stark anfällig für File-Inclusion-Angriffe. Im Gegensatz dazu schützt die „Impossible – File Inclusion“-Variante gezielt durch eine Whitelist nur definierte, sichere Dateien. Der Zugriff auf andere Dateien wird blockiert, wodurch File-Inclusion-Schwachstellen effektiv verhindert werden.**

# File Upload

## Informationen

### Definition[[11]](#footnote-11)

**Ein Datei-Upload-Angriff tritt auf, wenn eine Webanwendung es Benutzern ermöglicht, Dateien hochzuladen, ohne diese ausreichend zu validieren oder einzuschränken. Angreifer können diese Schwachstelle ausnutzen, um bösartige Dateien auf den Server hochzuladen, was zu unautorisiertem Zugriff, Datenmanipulation oder der Ausführung von Schadcode führen kann. Solche Angriffe sind besonders kritisch, da sie die Integrität und Sicherheit der gesamten Anwendung gefährden können.**

## Grafik[[12]](#footnote-12)

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Code

**Low – File Upload**

<?php

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

**Der Code prüft, ob das Upload-Formular mit dem Button Upload abgeschickt wurde.**

// Where are we going to be writing to?

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

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

**Definiert das Verzeichnis, in das die hochgeladene Datei gespeichert werden soll. Fügt den ursprünglichen Dateinamen an den Pfad an, damit die Datei unter demselben Namen gespeichert wird.**

// 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>";

**Versucht, die temporär gespeicherte Datei in den Zielordner zu verschieben und es gibt eine Erfolgsmeldung oder eine Fehlermeldung aus – je nachdem, ob der Vorgang funktioniert hat.**

}

}

?>

**Dieser Code erlaubt das Hochladen von Dateien auf den Server, ohne irgendwelche Sicherheitsprüfungen. Weder der Dateityp, die Größe noch die Dateiendung werden kontrolliert. Dadurch kann ein Angreifer potenziell gefährliche Dateien wie .php-Skripte hochladen, was zu einer vollständigen Kompromittierung des Servers führen kann. Das macht diesen Code stark anfällig für File-Upload-Angriffe.**

**Impossible – File Upload**

<?php

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

// Check Anti-CSRF token

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

**Überprüft, ob das Formular abgeschickt wurde. Verifiziert das CSRF-Token, um Angriffe durch fremde Webseiten zu verhindern (Cross-Site Request Forgery).**

// 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' ];

**Liest Name, Dateiendung, Größe, MIME-Typ und temporären Pfad der hochgeladenen Datei aus.**

// 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;

**Erstellt zufällige Dateinamen mit Hash zur Verhinderung von Überschreibungen. Definiert temporären Speicherort und Zielverzeichnis.**

$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 ) ) {

**Stellt sicher, dass die Datei eine echte JPEG oder PNG-Bilddatei ist. Prüft zusätzlich Größe, MIME-Typ und Bildstruktur.**

// 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 );

**Öffnet das Bild, speichert es neu und entfernt dabei potenziell eingebettete Metadaten oder Schadcode.**

// 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>';

}

**Verschiebt die bereinigte Datei vom temporären Speicher ins Upload-Verzeichnis und gibt eine Rückmeldung über Erfolg oder Misserfolg.**

// Delete any temp files

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

unlink( $temp\_file );

**Löscht eventuell verbliebene temporäre Dateien.**

}

else {

// Invalid file

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

}

}

// Generate Anti-CSRF token

generateSessionToken();

?>

**Dieser Code stellt einen sicheren Datei-Upload-Prozess dar. Er schützt durch CSRF-Token vor unerlaubten Formularaktionen, prüft, ob die hochgeladene Datei wirklich ein JPEG oder PNG Bild ist, überprüft Größe und Typ, validiert die Bildstruktur mit getimagesize() und entfernt alle Metadaten durch Neu-Speicherung. Die Dateinamen werden zufällig generiert, um Überschreibungen zu verhindern. Dadurch wird das Hochladen gefährlicher oder manipulierten Dateien effektiv verhindert.**

1. [Brute Force Angriffe: Methoden, Tools und Schutzmaßnahmen | Proofpoint DE](https://www.proofpoint.com/de/threat-reference/brute-force-attack#:~:text=Ein%20Brute%20Force%20Angriff%20(Brute,nach%20durch%2C%20bis%20eines%20funktioniert.), 17.01,2025 [↑](#footnote-ref-1)
2. <https://www.myrasecurity.com/de/brute-force-attacke/>, 17.01.2025 [↑](#footnote-ref-2)
3. <https://www.experte.de/it-sicherheit/brute-force-angriff>, 17.01.2025 [↑](#footnote-ref-3)
4. <https://www.myrasecurity.com/de/brute-force-attacke/>, 24.01.2025 [↑](#footnote-ref-4)
5. <https://owasp.org/www-community/attacks/Command_Injection>, 07.03.2025 [↑](#footnote-ref-5)
6. <https://www.ubikasec.com/en/posts/command-injection-how-to-counter-them-with-ubika-cloud-protector/> [↑](#footnote-ref-6)
7. <https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html>, 07.03.2025 [↑](#footnote-ref-7)
8. <https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/>, 07.03.2025 [↑](#footnote-ref-8)
9. <https://www.sumologic.com/glossary/file-inclusion/>, 31.03.2025 [↑](#footnote-ref-9)
10. <https://evbn.org/what-is-rfi-remote-file-inclusion-example-mitigation-methods-imperva-1657240244/>, 31,03.2025 [↑](#footnote-ref-10)
11. <https://br4ind3ad.medium.com/port-swigger-file-upload-vulnerability-lab-1-1e0dd4c54cf>, 31.03.2025 [↑](#footnote-ref-11)
12. <https://sarnisyakir01.medium.com/file-upload-vulnerability-5bfc7df7066>, 31.03.2025 [↑](#footnote-ref-12)