|  |
| --- |
| 4aWI |
| DVWA |
| Beer Martin |

|  |
| --- |
| Beer Martin  10.1.2025 |

Inhalt

[1.1 Brute Force 0](#_Toc188001041)

[1.1.1 Information 0](#_Toc188001042)

[1.1.2 Graphische Darstellung 0](#_Toc188001043)

[1.2 Command Injection 1](#_Toc188001044)

[1.3 CSRF 3](#_Toc188001045)

[1.4 File Inclusion 3](#_Toc188001046)

[1.5 File Upload 3](#_Toc188001047)

[1.6 Insecure CAPTCHA 3](#_Toc188001048)

[1.7 SQL Injection 3](#_Toc188001049)

[1.8 SQL Injection (blind) 3](#_Toc188001050)

[1.9 Weak Session IDs 3](#_Toc188001051)

[1.10 XSS (DOM) 3](#_Toc188001052)

[1.11 XSS (Reflected) 3](#_Toc188001053)

[1.12 XSS (Stored) 3](#_Toc188001054)

[1.13 CSP Byepass 3](#_Toc188001055)

[1.14 JavaScript 3](#_Toc188001056)

[1.15 Authorisation Byepass 3](#_Toc188001057)

[1.16 Open HTTP Redirect 3](#_Toc188001058)

[1.17 Cryptography 3](#_Toc188001059)

[1.18 TOOLS 4](#_Toc188001060)

# Brute Force

## 1.1 Information

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. Wenn diese Liste zu keinem Ergebnis führt, versucht er es mit verschiedenen Buchstabenkombinationen. Es braucht manchmal tausende von Versuchen, bis ein Passwort geknackt ist. Deshalb kommen für Brute Force, Automatisierungstools zum Einsatz, die es einem Angreifer ermöglichen, sehr viele Versuche möglichst schnell durchzuführen.[[1]](#footnote-1)

Durch einige Rechenbeispiele soll das Zusammenwirken von Länge und verwendeter Zeichen für die Sicherheit eines Passworts veranschaulicht werden. In den Rechenbeispielen wird mit einer Generierung von 2 Milliarden Schlüsseln pro Sekunde gerechnet, da dies ungefähr der Geschwindigkeit eines sehr starken Einzelrechners entspricht.

Beim Erstellen eines Passworts stehen Ihnen in der Regel folgende Zeichen zur Verfügung:

* Zahlen (10 verschiedene: 0-9)
* Buchstaben (52 verschiedene: A-Z und a-z)
* Sonderzeichen (32 verschiedene).[[2]](#footnote-2)

## 1.2 Graphische Darstellung

[](https://www.google.com/url?sa=i&url=https%3A%2F%2Frublon.com%2Fblog%2Fbrute-force-dictionary-attack-difference%2F&psig=AOvVaw0SeKz697a1BzNO8LdCtqlc&ust=1737191516353000&source=images&cd=vfe&opi=89978449&ved=0CBQQjRxqFwoTCLDYzuy0_IoDFQAAAAAdAAAAABAK)

Abbildung 1 BruceForce Attacke

## 1.3 Code

**Low Brute Force Code:**

<?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);

}

?>

**Unbegrenzte Login-Versuche**: Das Skript blockiert den Angreifer nicht nach einer bestimmten Anzahl falscher Passworteingaben.

**Schnelles Durchprobieren**: Da es kein künstliches Delay (z. B. sleep) gibt, können automatisierte Tools sehr schnell alle möglichen Kombinationen testen.

**Keine Account-Sperre**: Selbst nach vielen Fehleingaben wird der Login nicht für eine bestimmte Zeit gesperrt, sodass ein automatisiertes Ausprobieren nicht gestoppt wird.

**Impossible Brute Force Code:**

<?php

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

// 1) Anti-CSRF prüfen

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

// 2) Eingaben validieren/escapen

$user = mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], stripslashes($\_POST['username']));

$pass = md5(mysqli\_real\_escape\_string($GLOBALS["\_\_\_mysqli\_ston"], stripslashes($\_POST['password'])));

// 3) Lockout-Einstellungen

$failed\_limit = 3;

$lockout\_time = 15; // Minuten

$account\_locked = false;

// 4) Ist der Account schon gesperrt?

$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();

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

if (time() < strtotime($row['last\_login']) + ($lockout\_time \* 60)) {

$account\_locked = true;

}

}

// 5) Prüfung der Logindaten

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

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

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

$data->execute();

// 6) Erfolgreicher Login?

if ($data->rowCount() && !$account\_locked) {

// a) Fehlversuche zurücksetzen

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

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

$upd->execute();

echo "Willkommen, $user";

} else {

// a) Verzögerung bei falschem Login

sleep(rand(2, 4));

// b) Fehlversuche hochzählen

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

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

$upd->execute();

echo "Fehler: Ungültige Login-Daten oder Account gesperrt.";

}

// 7) Letzte Loginzeit aktualisieren

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

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

$upd->execute();

}

// 8) Anti-CSRF-Token erzeugen

generateSessionToken();

?>

**Entscheidende Punkte für sicheren Code sind:**

**Anti-CSRF-Token**: Verhindert, dass Angreifer Logins von fremden Seiten auslösen.

**Eingabesanitierung**: Schützt vor SQL-Injections.

**Lockout**: Nach totalfailedlogintotal\_failed\_logintotalf​ailedl​ogin Fehlversuchen wird der Account für lockouttimelockout\_timelockoutt​ime Minuten gesperrt.

**Zeitverzögerung**: sleep(rand(2, 4)) erschwert automatisierte Angriffe.

**Zurücksetzen und Inkrementieren der**: Erlaubt dem System, erfolgreiche Logins wieder zu normalisieren und bei Fehlern konsequent zu erhöhen.

**Update der letzten Login-Zeit**: Dient zur Berechnung der Sperrzeit und zur Protokollierung.

## 1.4 Tools zum Hacken und Testen

* Hydra
* Burpsuite
* OWASP ZAP
* Klassische Programmierung: Java, JavaScript, node
* Sqlmap (Testing-Tool)

# Command Injection

## Information

Command Injection ist ein Angriff, bei dem das Ziel darin besteht, beliebige Befehle auf dem Host-Betriebssystem über eine verwundbare Anwendung auszuführen. Command Injection-Angriffe sind möglich, wenn eine Anwendung unsichere, vom Benutzer bereitgestellte Daten (wie Formulare, Cookies, HTTP-Header etc.) an eine System-Shell übergibt. Bei diesem Angriff werden die vom Angreifer gelieferten Betriebssystembefehle in der Regel mit den Privilegien der verwundbaren Anwendung ausgeführt. Command Injection-Angriffe sind vor allem auf unzureichende Eingabevalidierung zurückzuführen.

Dieser Angriff unterscheidet sich von Code Injection, bei der es dem Angreifer ermöglicht wird, eigenen Code hinzuzufügen, der dann von der Anwendung ausgeführt wird. Bei Command Injection erweitert der Angreifer die Standardfunktionalität der Anwendung, die Systembefehle ausführt, ohne dass es notwendig ist, eigenen Code einzuspritzen.[[3]](#footnote-3)

**Wie man OS-Command-Injection-Angriffe verhindert**

Der effektivste Weg, OS-Command-Injection-Schwachstellen zu verhindern, besteht darin, niemals von Anwendungscode aus Betriebssystembefehle aufzurufen. In fast allen Fällen gibt es alternative Möglichkeiten, die erforderliche Funktionalität mithilfe sichererer Plattform-APIs zu implementieren.

Falls es unumgänglich ist, Betriebssystembefehle mit benutzerdefinierten Eingaben auszuführen, muss eine strenge Eingabevalidierung erfolgen. Einige Beispiele für eine effektive Validierung sind:

* Validierung anhand einer Positivliste (Whitelist) zugelassener Werte.
* Überprüfung, ob die Eingabe eine Zahl ist.
* Validierung, dass die Eingabe ausschließlich alphanumerische Zeichen enthält, ohne weitere Syntax oder Leerzeichen.[[4]](#footnote-4)

## Graphische Darstellung

Ein Bild, das Text, Diagramm, Entwurf, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung 2 Command Injection

## Code

**Low Command Injection Code:**

<?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 "Low Command Injection Code" ist unsicher, weil er den vom Benutzer eingegebenen Wert direkt in den Shell-Befehl einfügt, ohne diesen zu validieren oder zu bereinigen. Dadurch kann ein Angreifer zusätzliche Befehle einschleusen (z. B. durch das Einfügen von Semikolons oder anderen Shell-Metazeichen), was zu einer Ausführung beliebiger Systembefehle führen kann.

**Impossible Command Injection Code:**

<?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();

?>

Der Impossible Command Injection Code hingegen ergreift mehrere Sicherheitsmaßnahmen:

* **Strenge Validierung:** Der eingegebene IP-Wert wird mittels explode in vier Oktette aufgeteilt. Anschließend wird überprüft, ob jedes Oktett numerisch ist und ob tatsächlich genau vier Teile vorliegen. Das stellt sicher, dass nur gültige IPv4-Adressen akzeptiert werden.
* **Anti-CSRF Token:** Durch den Einsatz eines Anti-CSRF Tokens wird zusätzlich verhindert, dass Angreifer über Cross-Site-Request-Forgery-Angriffe schädliche Anfragen stellen können.

## Tools zum Hacken und Testen

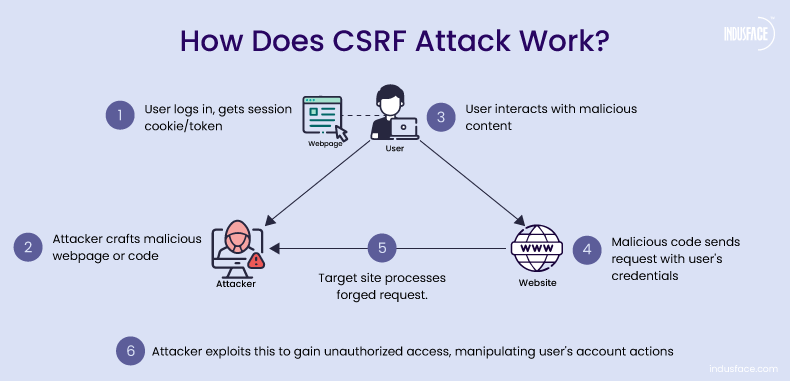
* Commix (COMMand Injection eXploiter)
* Metasploit Framework
* Burpsuit
* Nuclei
* Custom Scripts (Python, Bash, …)

# CSRF

## Information

CSRF ist ein Angriff, der das Opfer dazu bringt, eine bösartige Anfrage abzusenden. Dabei übernimmt der Angriff die Identität und Privilegien des Opfers, um in dessen Namen eine unerwünschte Funktion auszuführen (wobei zu beachten ist, dass dies bei Login-CSRF, einer speziellen Form des Angriffs, wie im Folgenden beschrieben, nicht zutrifft). Bei den meisten Webseiten beinhalten Browseranfragen automatisch alle mit der Seite verbundenen Zugangsdaten, wie das Sitzungscookie des Benutzers, die IP-Adresse, Windows-Domänenanmeldeinformationen und so weiter. Daher hat die Webseite, wenn der Benutzer gerade authentifiziert ist, keine Möglichkeit zu unterscheiden, ob es sich um eine gefälschte Anfrage des Opfers oder um eine legitime Anfrage handelt.[[5]](#footnote-5)

## Graphische Darstellung



## Code

**Low CSRF Code:**

<?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);

}

?>

**Übertragung sensibler Daten über GET:**  
Die neuen Passwörter werden per GET übermittelt, was dazu führt, dass diese Informationen in Browser-Historien und Server-Logs landen können.

**Fehlender CSRF-Schutz:**  
Es wird kein Anti-CSRF-Token verwendet, sodass ein Angreifer das Opfer dazu bringen könnte, ungewollt das Passwort zu ändern.

**Unzureichende SQL-Sicherheit:**  
Obwohl bei $pass\_new eine Escape-Funktion genutzt wird, wird der Benutzername direkt in die SQL-Anfrage eingebettet. Dadurch entsteht ein potenzielles Risiko für SQL-Injection, da keine Prepared Statements verwendet werden.

**Veraltetes Hashing-Verfahren:**  
Die Verwendung von MD5 zum Hashen der Passwörter gilt als unsicher, da MD5 anfällig für Kollisionen und Brute-Force-Angriffe ist.

**Mangelhafte Fehlerbehandlung:**  
Durch den direkten Einsatz von die() können im Fehlerfall interne Details an den Angreifer preisgegeben werden.

**Impossible CSRF Code:**

<?php

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

// Anti-CSRF-Überprüfung

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

// Eingaben abrufen und sanitizen

$pass\_curr = md5(mysqli\_real\_escape\_string($db\_connection, stripslashes($\_GET['password\_current'])));

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

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

// Aktuelles Passwort mit Prepared Statement prüfen

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

$current\_user = dvwaCurrentUser();

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

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

$stmt->execute();

// Prüfung: Ist das aktuelle Passwort korrekt und stimmen die neuen Passwörter überein?

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

$pass\_new = md5(mysqli\_real\_escape\_string($db\_connection, stripslashes($pass\_new)));

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

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

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

$update->execute();

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

} else {

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

}

}

// Neuer Anti-CSRF Token für die Session

generateSessionToken();

?>

**Wesentliche Sicherheitsaspekte:**

* **Anti-CSRF-Überprüfung:** Mit checkToken() wird sichergestellt, dass die Anfrage von einem berechtigten Benutzer stammt.
* **Input-Sanitization:** Durch stripslashes() und mysqli\_real\_escape\_string() werden Eingaben bereinigt, um SQL-Injection zu verhindern.
* **Prepared Statements:** Die Verwendung von Prepared Statements mit bindParam() verhindert SQL-Injection, da Benutzereingaben nicht direkt in die SQL-Abfrage eingebettet werden.
* **Passwort-Hashing:** Die Passwörter werden mittels md5() gehasht, um sie nicht im Klartext zu speichern (obwohl md5 heutzutage als veraltet gilt).
* **Token-Erneuerung:** Mit generateSessionToken() wird nach der Anfrage ein neuer Anti-CSRF-Token erstellt, um die Sicherheit der Session zu gewährleisten.

## 3.4 Tools zum Hacken und Testen

* BurpSuite
* OWASP ZAP (Zed Attack Proxy)
* Postman
* NoCSRF
* CSRF PoC Generator
* WebGoat
* BeEF

# File Inclusion

## Information

## 4.2 Graphische Darstellung

## 4.3 Code

## 4.4 Tools zum Hacken und Testen

# 5 File Upload

# 6 Insecure CAPTCHA

# 7 SQL Injection

# 8 SQL Injection (blind)

# 9 Weak Session IDs

# 10 XSS (DOM)

# 11 XSS (Reflected)

# 12 XSS (Stored)

# 13 CSP Byepass

# 14 JavaScript

# 15 Authorisation Byepass

# 16 Open HTTP Redirect

# 17 Cryptography

# 18 TOOLS

[Abbildung 1 BruceForce Attacke 1](#_Toc188001649)

1. <https://www.proofpoint.com/de/threat-reference/brute-force-attack> [↑](#footnote-ref-1)
2. https://www.password-depot.de/de/know-how/brute-force-angriffe.htm [↑](#footnote-ref-2)
3. https://owasp.org/www-community/attacks/Command\_Injection [↑](#footnote-ref-3)
4. https://portswigger.net/web-security/os-command-injection [↑](#footnote-ref-4)
5. https://owasp.org/www-community/attacks/csrf [↑](#footnote-ref-5)