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

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

Inhalt

[1.1 Brute Force 0](#_Toc192167788)

[1.1 Information 0](#_Toc192167789)

[1.2 Graphische Darstellung 1](#_Toc192167790)

[1.3 Code 2](#_Toc192167791)

[1.4 Tools zum Hacken und Testen 4](#_Toc192167792)

[2 Command Injection 5](#_Toc192167793)

[2.1 Information 5](#_Toc192167794)

[2.2 Graphische Darstellung 6](#_Toc192167795)

[2.3 Code 7](#_Toc192167796)

[2.4 Tools zum Hacken und Testen 8](#_Toc192167797)

[3 CSRF 9](#_Toc192167798)

[3.1 Information 9](#_Toc192167799)

[3.2 Graphische Darstellung 9](#_Toc192167800)

[3.3 Code 10](#_Toc192167801)

[3.4 Tools zum Hacken und Testen 12](#_Toc192167802)

[4 File Inclusion 13](#_Toc192167803)

[4.1 Information 13](#_Toc192167804)

[4.2 Graphische Darstellung 15](#_Toc192167805)

[4.3 Code 16](#_Toc192167806)

[4.4 Tools zum Hacken und Testen 18](#_Toc192167807)

[5 File Upload 19](#_Toc192167808)

[5.1 Information 19](#_Toc192167809)

[5.2 Graphische Darstellung 20](#_Toc192167810)

[5.3 Code 21](#_Toc192167811)

[5.4 Tools zum Hacken und Testen 23](#_Toc192167812)

[6 Insecure CAPTCHA 24](#_Toc192167813)

[7 SQL Injection 24](#_Toc192167814)

[8 SQL Injection (blind) 24](#_Toc192167815)

[9 Weak Session IDs 24](#_Toc192167816)

[10 XSS (DOM) 24](#_Toc192167817)

[11 XSS (Reflected) 24](#_Toc192167818)

[12 XSS (Stored) 24](#_Toc192167819)

[13 CSP Byepass 24](#_Toc192167820)

[14 JavaScript 24](#_Toc192167821)

[15 Authorisation Byepass 24](#_Toc192167822)

[16 Open HTTP Redirect 24](#_Toc192167823)

[17 Cryptography 24](#_Toc192167824)

[18 TOOLS 24](#_Toc192167825)

# 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.researchgate.net/publication/379624465/figure/fig2/AS:11431281234681580@1712414089963/Work-architecture-of-Brute-Force-Attack7.ppm)

Abbildung 1BruteForce Attack https://www.researchgate.net/publication/379624465/figure/fig2/AS:11431281234681580@1712414089963/Work-architecture-of-Brute-Force-Attack7.ppm

## 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 https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT5MDilLllu\_4qBsOfOnh94\_TBFumhVfExXnpK-cXRSaTsYrXnr

## 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

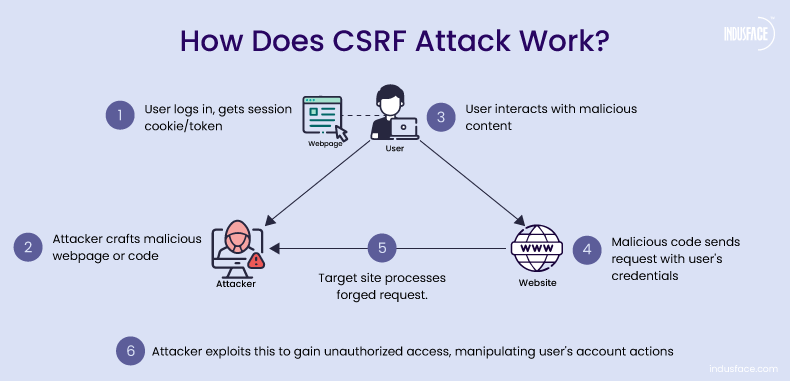


Abbildung 3 CSRF https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSNaq9pJtukr0lln4EroZ6BOW8noYucd2H4pc1KNhkS1MXflmpP

## 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

Eine der häufigsten Arten von Angriffen auf Webserver resultiert aus File-Inclusion-Schwachstellen. Diese Schwachstellen finden sich hauptsächlich in Webanwendungen, die eine Scripting-Laufzeit verwenden. Sie ermöglichen es Angreifern, auf sensible Dateien auf den Webservern zuzugreifen oder die Include-Funktion zu nutzen, um schädliche Dateien auf den Servern auszuführen. Durch den Zugriff auf unautorisierte Dateien können Angreifer sensible Informationen erlangen oder die Netzwerke des Opfers weiter kompromittieren.

Es gibt zwei Typen von File-Inclusion-Schwachstellen, die Sie bei der Vorbereitung auf einen Angriff kennen sollten: lokale File-Inclusion (LFI) und Remote-File-Inclusion (RFI).

Der Hauptunterschied zwischen den beiden liegt darin, wo sich die kompromittierte Datei ursprünglich befindet. LFI-Schwachstellen werden durch eine auf dem Zielserver gespeicherte Datei ausgenutzt, während RFI eine Datei aus einer Drittquelle verwendet.

**LFI involves:**

Wenn Eingaben auf einer Website oder in einer Webanwendung nicht ordnungsgemäß bereinigt werden, können lokale Dateien auf einem Server anfällig für Angriffe werden. Benutzer-Eingaben, die Pfade zu Dateien enthalten und falsch validiert werden, ermöglichen es Angreifern, auf diese Dateien zuzugreifen und sensible Dateien in anderen Verzeichnissen abzurufen.

Das folgende INFOSEC-Beispiel veranschaulicht, wie ein lokaler File-Inclusion-Angriff erfolgen kann:

http://victim\_site/abc.php?file=userinput.txt

<?php

…

include $\_REQUEST['file'];

…

?>

Angreifer können eine bösartige Eingabe einfügen, um sensible Dateien im aktuellen Verzeichnis abzurufen oder in andere Verzeichnisse zu wechseln, um Ihr System weiter zu kompromittieren.

**RFI**

Obwohl ähnlich, nutzen RFI-Schwachstellen eine externe Quelle, anstatt Dateien vom lokalen Server zuzugreifen, um einen Angriff auszuführen. Angreifer verwenden den Befehl „dynamic file include“, um schädliche externe Dateien oder Skripte einzufügen. Ohne ordnungsgemäße Bereinigung von Dateien können Angreifer Webanwendungen ausnutzen, um externe Dateien mit schädlichen Skripten einzufügen.

Hier ist ein Beispiel dafür, wie eine RFI-Schwachstelle auftreten könnte:

Bash:

[www.victim\_site.com/abc.php?testfile=example](http://www.victim_site.com/abc.php?testfile=example)

Der anfällige PHP-Code:

Shell:

$test = $\_REQUEST["testfile"];

include($test . ".php");

Der Parameter „testfile“ in diesem Beispiel wird vom Benutzer bereitgestellt, und der Code fügt den Wert von „testfile“ in die PHP-Datei ein.

In beiden Fällen resultieren die Schwachstellen direkt aus mangelhafter Eingabevalidierung, was bedeutet, dass eine der wenigen Möglichkeiten zur Vermeidung von File-Inclusion-Schwachstellen in der konsequenten Anwendung von Bereinigungspraktiken liegt.[[6]](#footnote-6)

## Graphische Darstellung

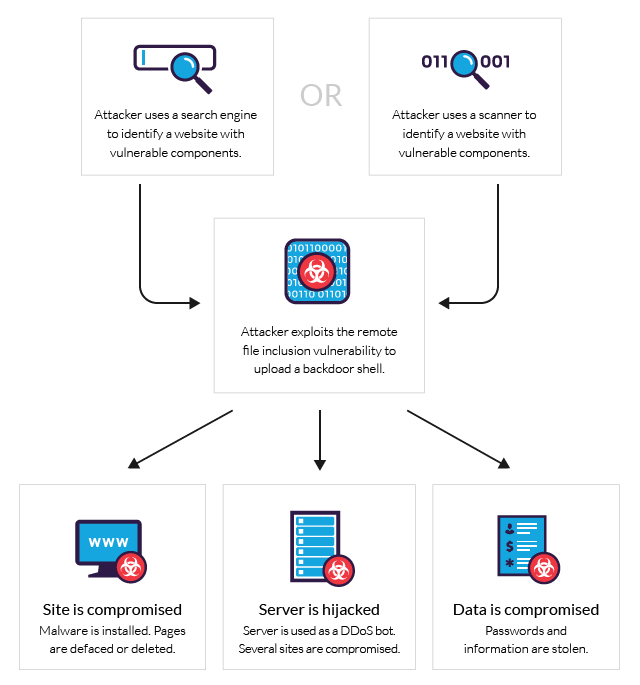


Abbildung 4 File-Inclusion <https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcSD-W7x1jQSSMl8jkIsfJl3qipeEgxk2fTBA_D8lkzumw3OlH8w>

## Code

**Low File-Inclusion Code:**

<?php

// The page we wish to display

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

?>

**Warum der "Low Security" Code unsicher ist:**

Im "Low Security" Code wird der Dateiname direkt aus der GET-Variable übernommen, ohne jegliche Überprüfung oder Validierung. Das bedeutet, dass ein Angreifer beliebige Dateinamen übergeben kann. Dadurch kann er:

* **Lokale File Inclusion (LFI)** ausnutzen, um sensible Dateien auf dem Server zu lesen.
* **Remote File Inclusion (RFI)**, sofern die Serverkonfiguration es erlaubt, ausführen, wodurch schädlicher Code von extern geladen wird.
* **Directory Traversal**-Angriffe durchführen, indem er Pfadmanipulationen (z. B. ../) einsetzt, um Dateien außerhalb des erlaubten Verzeichnisses abzurufen.

Da keinerlei Sicherheitsmechanismen wie eine Whitelist oder eine gründliche Eingabevalidierung vorhanden sind, ist der Code sehr anfällig für Angriffe.

**Impossible File-Inclusion Code:**

<?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 der "Impossible Security" Code so sicher ist:**

Im "Impossible Security" Code wird ebenfalls der Dateiname aus der GET-Variable entnommen, aber anschließend wird eine Whitelist verwendet. Nur wenn der übergebene Dateiname exakt mit einem der erlaubten Einträge (include.php, file1.php, file2.php, file3.php) übereinstimmt, wird der Code ausgeführt. Andernfalls wird ein Fehler ausgegeben und der Prozess beendet. Dadurch wird:

* **Arbiträre Dateiauswahl verhindert:** Ein Angreifer kann nur eine der explizit erlaubten Dateien anfordern.
* **File Inclusion Angriffe verhindert:** Da keine nicht erlaubten Dateien einbezogen werden können, wird die Gefahr von LFI oder RFI effektiv eliminiert.
* **Eingabevalidierung gewährleistet:** Die Anwendung überprüft die Eingabe gegen eine fest definierte Liste und stellt so sicher, dass nur legitime und vordefinierte Dateien geladen werden.

## 4.4 Tools zum Hacken und Testen

**DVWA (Damn Vulnerable Web Application):**

Eine bewusst unsichere Webanwendung, die verschiedene Sicherheitslücken, darunter File Inclusion, simuliert. DVWA ermöglicht es, Sicherheitslücken zu erkennen und zu beheben.

**bWAPP (Buggy Web Application):**

Ein weiteres Open-Source-Tool, das eine breite Palette an Schwachstellen bietet. bWAPP enthält Module, um File Inclusion-Fehler gezielt zu testen.

**OWASP WebGoat:**

Eine von OWASP bereitgestellte Lehrumgebung, in der verschiedene Angriffe und Schwachstellen, einschließlich File Inclusion, simuliert werden. Es ist ideal für interaktive Lernszenarien.

**Mutillidae II:**

Eine kostenlose, leicht zu installierende Webanwendung, die gezielt zum Testen von Web-Angriffen und Sicherheitstools entwickelt wurde. Mutillidae II enthält Szenarien zu File Inclusion und anderen typischen Angriffen.

**Google Gruyere:**

Eine webbasierte Übungsumgebung, die speziell für das Erlernen von Web-Sicherheit konzipiert wurde. Gruyere bietet praktische Übungen zu diversen Schwachstellen, darunter File Inclusion.

# File Upload

## 5.1 Information

File-Upload-Schwachstellen treten auf, wenn ein Webserver es Benutzern erlaubt, Dateien in sein Dateisystem hochzuladen, ohne ausreichend Aspekte wie den Namen, Typ, Inhalt oder die Größe zu validieren. Werden diese Einschränkungen nicht ordnungsgemäß durchgesetzt, kann bereits eine einfache Image-Upload-Funktion dazu genutzt werden, beliebige und potenziell gefährliche Dateien hochzuladen. Dies könnte sogar serverseitige Skriptdateien einschließen, die eine Remote-Code-Ausführung ermöglichen.

In einigen Fällen reicht der Akt des Hochladens der Datei allein aus, um Schaden anzurichten. Andere Angriffe können eine nachfolgende HTTP-Anfrage für die Datei beinhalten, um deren Ausführung durch den Server zu triggern.

Die Auswirkungen von Datei-Upload-Schwachstellen hängen im Allgemeinen von zwei wesentlichen Faktoren ab:

* **Überprüfung der Dateieigenschaften:**  
  Entscheidend ist, welche Eigenschaften der hochgeladenen Datei – wie Größe, Typ oder Inhalt – von der Webanwendung nicht korrekt validiert werden.
* **Angewandte Beschränkungen:**  
  Ebenso wichtig ist, welche Einschränkungen nach dem Hochladen greifen, etwa wo und wie die Datei gespeichert oder ausgeführt wird.

Im schlimmsten Fall wird der Dateityp nicht ordnungsgemäß geprüft und der Server erlaubt, dass bestimmte Dateiformate (wie .php oder .jsp) als Code ausgeführt werden. Dadurch könnte ein Angreifer eine serverseitige Code-Datei hochladen, die als Webshell dient und ihm somit vollständige Kontrolle über den Server verschafft.

Wenn der Dateiname nicht korrekt validiert wird, besteht zudem die Gefahr, dass ein Angreifer kritische Dateien überschreibt, indem er eine Datei mit demselben Namen hochlädt. Sollte der Server außerdem anfällig für Directory-Traversal sein, kann dies dazu führen, dass Dateien an unvorhergesehenen Orten gespeichert werden.

Fehlt zudem die Prüfung, ob die Dateigröße innerhalb zulässiger Grenzen liegt, kann ein Angreifer den verfügbaren Speicherplatz füllen und so einen Denial-of-Service (DoS)-Angriff auslösen.[[7]](#footnote-7)

## 5.2 Graphische Darstellung

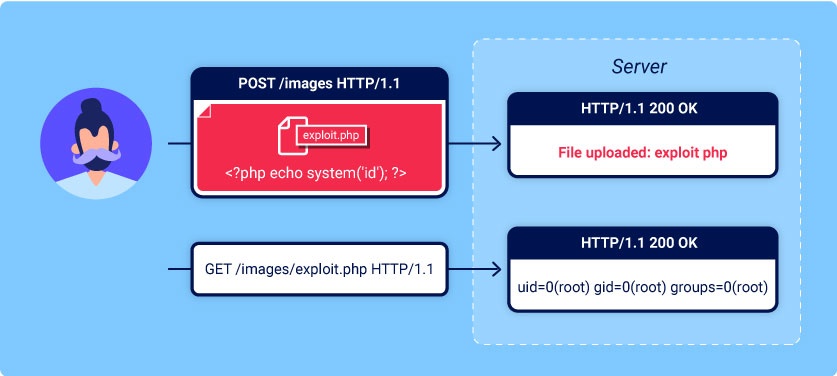


Abbildung 5 File Upload https://portswigger.net/web-security/file-upload/images/file-upload-vulnerabilities.jpg

## 5.3 Code

**Low File Upload Code:**

<?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>";

}

}

?>

**Warum der „Low Security Code“ unsicher ist:**

* **Fehlende Validierung:** Es gibt keine Überprüfung des Dateityps, der Dateigröße oder des Inhalts. Dadurch können beliebige Dateien, z. B. PHP-Skripte, hochgeladen werden.
* **Keine Anti-CSRF-Maßnahmen:** Es fehlt ein Schutzmechanismus gegen Cross-Site Request Forgery, was Manipulationen ermöglicht.
* **Keine Metadaten-Bereinigung:** Es wird keine Überprüfung oder Entfernung von potenziell schädlichen Metadaten vorgenommen.

**Impossible File Upload Code:**

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

// Anti-CSRF-Prüfung

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

// Dateiinformationen

$uploaded\_ext = strtolower(pathinfo($\_FILES['uploaded']['name'], PATHINFO\_EXTENSION));

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

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

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

// Überprüfungen

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

($uploaded\_size < 100000) &&

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

getimagesize($uploaded\_tmp)) {

// Re-Encoding zur Metadaten-Entfernung

$img = ($uploaded\_type == 'image/jpeg') ? imagecreatefromjpeg($uploaded\_tmp) : imagecreatefrompng($uploaded\_tmp);

$temp\_file = sys\_get\_temp\_dir() . DIRECTORY\_SEPARATOR . md5(uniqid() . $\_FILES['uploaded']['name']) . '.' . $uploaded\_ext;

($uploaded\_type == 'image/jpeg') ? imagejpeg($img, $temp\_file, 100) : imagepng($img, $temp\_file, 9);

imagedestroy($img);

// Sicherer Dateiname

$target\_file = md5(uniqid() . $\_FILES['uploaded']['name']) . '.' . $uploaded\_ext;

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

if(rename($temp\_file, $target\_path)) {

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

} else {

echo 'Your image was not uploaded.';

}

} else {

echo 'Your image was not uploaded. We only accept JPEG or PNG images.';

}

}

generateSessionToken();

**Warum der „Impossible Security Code“ so sicher ist:**

* **Anti-CSRF-Token:** Durch die Überprüfung des Anti-CSRF-Tokens wird sichergestellt, dass nur authentische Upload-Anfragen verarbeitet werden.
* **Umfassende Dateiprüfung:**
  + **Erweiterungs- und MIME-Typ-Check:** Nur JPEG- oder PNG-Dateien werden akzeptiert.
  + **Größenprüfung:** Dateien, die zu groß sind, werden abgelehnt.
  + **Validierung durch getimagesize:** Dies stellt sicher, dass die Datei tatsächlich ein Bild ist.
* **Metadaten-Entfernung:** Durch das Re-Encoding der Bilder werden schädliche Metadaten entfernt.
* **Sicherer Dateiname:** Der Dateiname wird zufällig generiert (mit md5 und uniqid), um Manipulationen zu verhindern.

## 5.4 Tools zum Hacken und Testen

**OWASP ZAP:** Open-Source-Security-Tool zum Auffinden von Schwachstellen.

**Burp Suite (Community Edition):** Ermöglicht das Abfangen und Manipulieren von Upload-Anfragen.

**DVWA:** Speziell entwickelte Übungsumgebung für File-Upload-Angriffe.

**OWASP Juice Shop:** Realistische Web-App mit diversen Sicherheitslücken zum Üben.

# 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 1BruteForce Attack https://www.researchgate.net/publication/379624465/figure/fig2/AS:11431281234681580@1712414089963/Work-architecture-of-Brute-Force-Attack7.ppm 1](#_Toc192167848)

[Abbildung 2 Command Injection https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcT5MDilLllu\_4qBsOfOnh94\_TBFumhVfExXnpK-cXRSaTsYrXnr 6](#_Toc192167849)

[Abbildung 3 CSRF https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcSNaq9pJtukr0lln4EroZ6BOW8noYucd2H4pc1KNhkS1MXflmpP 9](#_Toc192167850)

[Abbildung 4 File-Inclusion https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcSD-W7x1jQSSMl8jkIsfJl3qipeEgxk2fTBA\_D8lkzumw3OlH8w 15](#_Toc192167851)

[Abbildung 5 File Upload https://portswigger.net/web-security/file-upload/images/file-upload-vulnerabilities.jpg 20](#_Toc192167852)

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)
6. <https://www.sumologic.com/glossary/file-inclusion/> [↑](#footnote-ref-6)
7. <https://portswigger.net/web-security/file-upload> [↑](#footnote-ref-7)