Keller Ruben

DVWA Dokumentation

Contents

[1. Grundlagen 2](#_Toc192839099)

[2. Brute Force 2](#_Toc192839100)

[3. Command Injection 8](#_Toc192839101)

[4. CSRF 11](#_Toc192839102)

[5. File Inclusion 15](#_Toc192839103)

[6. File Upload 18](#_Toc192839104)

[7. Insecure CAPTCHA 22](#_Toc192839105)

[8. SQL Injection 22](#_Toc192839106)

[9. SQL Injection (Blind) 22](#_Toc192839107)

[10. Weak Session Ids 22](#_Toc192839108)

[11. XSS (DOM) 22](#_Toc192839109)

[12. XSS (Reflected) 22](#_Toc192839110)

[13. XSS (Stored) 22](#_Toc192839111)

[14. CSP Bypass 23](#_Toc192839112)

[15. JavaScript 23](#_Toc192839113)

[16. Authorisation Bypass 23](#_Toc192839114)

[17. Open HTTP Redirect 23](#_Toc192839115)

# Grundlagen

Benutzername: admin  
Passwort: password

Die Damn Vulnerable Web Application (DVWA) ist eine absichtlich unsichere Webanwendung, die für Schulungszwecke und das Testen von Sicherheitstools entwickelt wurde. Sie bietet verschiedene Sicherheitsstufen, um Benutzer in den Grundlagen von Webanwendungssicherheit und Angriffstechniken zu schulen. DVWA dient als Übungsumgebung für Ethische Hacker, Entwickler und Sicherheitsexperten, um in einer kontrollierten Umgebung Schwachstellen zu verstehen und zu beheben.

Ein Bild, das Grafiken, Schrift, Text, Logo enthält.

Automatisch generierte Beschreibung

# Brute Force

Information

Ein Brute-Force-Angriff ist eine Methode, bei der ein Angreifer systematisch alle möglichen Kombinationen von Werten, wie Passwörter oder Verzeichnisnamen, ausprobiert, um unbefugten Zugang zu einem System zu erlangen oder versteckte Inhalte zu entdecken. Dies kann mit Tools wie DirBuster oder WebRoot durchgeführt werden, die Anfragen automatisieren und Serverantworten analysieren, um gültige Einträge zu finden. Der Angriff ist besonders effektiv, wenn Sicherheitsmaßnahmen wie Konto-Sperrrichtlinien oder Anforderungen an die Passwortkomplexität schwach oder nicht vorhanden sind. Gegenmaßnahmen umfassen Werkzeuge, die Scanning-Aktivitäten erkennen, oder strengere Sicherheitskontrollen, um Schwachstellen zu reduzieren.  
Quelle: <https://owasp.org/www-community/attacks/Brute_force_attack>

Ein Brute-Force-Angriff auf ein Webformular ist eine Methode, bei der ein Angreifer systematisch verschiedene Kombinationen von Benutzernamen und Passwörtern ausprobiert, um unautorisierten Zugriff zu erlangen. Tools wie THC-Hydra und Burp Suite können verwendet werden, um solche Angriffe zu automatisieren, indem sie Anfragen an den Server senden und die Antworten analysieren. Voraussetzung dafür sind eine Wortliste und die richtigen Parameter des Ziel-Webformulars, wie die Felder für Benutzername und Passwort sowie die Fehlermeldung bei falscher Eingabe. Solche Angriffe sollten nur in kontrollierten Umgebungen und mit Genehmigung durchgeführt werden, da sie andernfalls illegal sind.

Quelle: <https://www.golinuxcloud.com/brute-force-attack-web-forms>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**  
<?php

if( isset( $\_GET[ 'Login' ] ) ) { **//unbegrenzt viele Versuche möglich**

// 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>"; **//Gibt an was falsch war**

}

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

}

?>

Warum ist dieser Code unsicher: Es kann unbegrenzt versucht werden, sich anzumelden und der Angreifer kriegt direkte fehlermeldung und kann sogleich die nächste Anfrage stellen

Wie kann dieser Code sicherer gemacht werden: Begrenzte versuche pro Minute, Für bestimmte Zeit sperren, Zeige statt spezifischer Fehlermeldungen wie "Benutzername und/oder Passwort falsch" eine allgemeine Meldung wie "Ungültige Anmeldedaten" an, um Angreifern keine Hinweise zu geben was sie ändern müssen um rein zu kommen.

**Hard to hack:**

<?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; **definiert, dass bei mehr als 3 falschen Anmeldeversuchen, der Benutzer ausgesperrt wird für eine bestimmte Zeit**

// 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' ]; **Hier wird geschaut ob der Anmeldeversuch valide war**

// 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 ) ); **bei fehlerhaften Login muss der User/Hacker eine zufählige Zeit von 2 bis 4 sekunden warten, bis er es erneut versuchen kann**

// 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>"; **In der Fehlermeldung wird nicht bekannt gegeben ob der Benutzername oder das Passwort falsch ist, oder ob der Account gesperrt ist, so kann der Angreifer sich schlechter anpassen.**

// 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();

?>

Warum ist dieser Code so sicher:  
Limitierte Anmeldeversuche  
Zeitliche Verzögerungen (dauert somit länger)  
Limitierte Rückgabeinformationen bei falschen Versuchen

Tools zum Hacken

**1. John the Ripper**

* **Zweck**: Ein Passwort-Cracker.
* **Fokus**: Es versucht, gehashte Passwörter (verschlüsselte Passwörter) zu knacken.
* **Wie**: Nutzt Wörterbücher (Listen mit bekannten Passwörtern) oder Brute-Force-Angriffe (probiert alle möglichen Kombinationen aus).
* **Nutzung**: Eher vielseitig – es unterstützt viele verschiedene Hash-Typen (z. B. MD5, SHA, NTLM).
* **Typische Anwendung**: Entschlüsselung von Passwörtern aus Datenbanken oder Dateien.



Quelle <https://en.wikipedia.org/wiki/John_the_Ripper>

**2. Hydra**

* **Zweck**: Ein Passwort-Cracker für Netzwerkanwendungen.
* **Fokus**: Brute-Force-Angriffe auf **Login-Systeme**.
* **Wie**: Es versucht, sich mit verschiedenen Benutzernamen und Passwörtern bei Diensten anzumelden, z. B. SSH, FTP, HTTP, Telnet usw.
* **Nutzung**: Gut geeignet, um Schwachstellen bei Remote-Logins zu testen.
* **Typische Anwendung**: Test von Netzwerkdiensten, um schwache Passwörter zu finden.

Quelle: <https://en.wikipedia.org/wiki/Hydra_(software)>

**3. Hashcat**

* **Zweck**: Passwort-Cracking, spezialisiert auf Hashes.
* **Fokus**: Sehr schneller Passwort-Cracker, der GPU-Power nutzt.
* **Wie**: Unterstützt viele Angriffsarten, z. B. Wörterbuchangriffe, Brute-Force oder Hybridangriffe (Kombination aus Wörterbuch + Regeln).
* **Nutzung**: Sehr leistungsstark, ideal für große Datenmengen oder komplizierte Hash-Algorithmen.
* **Typische Anwendung**: Knacken komplexer Hashes, z. B. aus gehackten Datenbanken.

Quelle: <https://en.wikipedia.org/wiki/Hashcat>

**4. Aircrack-ng**

* **Zweck**: Ein Tool für WLAN-Sicherheitsüberprüfung.
* **Fokus**: Knacken von WLAN-Passwörtern (z. B. WPA, WPA2).
* **Wie**: Nutzt mitgeschnittene WLAN-Datenpakete, um Handshakes abzufangen und diese anschließend zu analysieren.
* **Nutzung**: Speziell für die WLAN-Sicherheit gedacht. Es überprüft auch die Verschlüsselung von Netzwerken und die Signalstärke.
* **Typische Anwendung**: Schwachstellen in WLAN-Netzwerken finden.



Quelle: <https://en.wikipedia.org/wiki/Aircrack-ng>

# Command Injection

Information

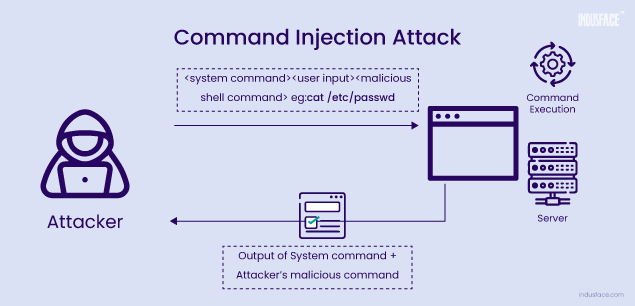
Command Injection ist eine Cyberangriffsmethode, bei der ein Hacker schädliche Befehle in ein System einschleust, indem er eine Sicherheitslücke in einer Anwendung ausnutzt. Dadurch kann der Angreifer Systembefehle ausführen, Daten stehlen, verändern oder das gesamte System übernehmen. Schutzmaßnahmen wie Eingabevalidierung und sichere Programmierung helfen, solche Angriffe zu verhindern.

Quelle: <https://www.imperva.com/learn/application-security/command-injection/>

Command Injection ist eine Sicherheitslücke, bei der ein Angreifer über unsichere Benutzereingaben Systembefehle auf einem Server ausführt. Dies kann durch spezielle Zeichen wie ;, && oder | erfolgen, um zusätzliche Befehle an das Betriebssystem weiterzugeben. Um solche Angriffe zu verhindern, sollten Entwickler Eingaben strikt validieren und direkte Systemaufrufe vermeiden.

Quelle: <https://www.vaadata.com/blog/what-is-command-injection-exploitations-and-security-best-practices/>

Visuelles Beispiel



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

KI-generierte Inhalte können fehlerhaft sein.

Code-Beispiel

**Easy to hack:**

<?php

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

// Get input

$target = $\_REQUEST[ 'ip' ]; **//Eingabe wird eins zu ein übernommen und nicht überprüft**

// Determine OS and execute the ping command.

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

// Windows

$cmd = shell\_exec( 'ping ' . $target ); **//Die Eingabe vom User wird direkt in die CMD eingefügt**

}

else {

// \*nix

$cmd = shell\_exec( 'ping -c 4 ' . $target ); **//Die Eingabe vom User wird direkt in die CMD eingefügt**

}

// Feedback for the end user

echo "<pre>{$cmd}</pre>"; **//Der Output ist ungefiltert und gibt dem Hacker alle Infos aus**

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

**//überprüfen der SessionID**

// 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];

**//prüfen ob die IP aus 4 numerischen Werten besteht**

// 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();

?>

Warum ist dieser Code so sicher:  
Weil überprüft wird ob eine IP-Adresse eingegeben wird damit keine Befehle, direkt ins Terminal eingegeben werden kann.

Tools zum Hacken

**Burp Suite**

* **Zweck:** Ein Tool zur Überprüfung der Sicherheit von Webanwendungen.
* **Fokus:** Findet Schwachstellen wie SQL-Injektionen und Cross-Site Scripting (XSS).
* **Wie:** Fängt den Datenverkehr zwischen Browser und Webanwendung ab, um ihn zu analysieren und zu verändern.
* **Nutzung:** Wird von Sicherheitsexperten verwendet, um Sicherheitslücken in Webanwendungen aufzudecken.
* **Typische Anwendung:** Testen einer Website auf Schwachstellen, die durch unsichere Benutzereingaben entstehen können.

Quelle: <https://en.wikipedia.org/wiki/Burp_Suite>

**Sqlmap**

* **Zweck:** Automatisches Erkennen und Ausnutzen von SQL-Injektions-Schwachstellen.
* **Fokus:** Spezialisiert auf SQL-Injektionen, um unberechtigten Zugriff auf Datenbanken zu verhindern.
* **Wie:** Sendet speziell gestaltete Anfragen an eine Webanwendung, um Schwachstellen in der Datenbankabfrage zu finden.
* **Nutzung:** Wird genutzt, um Sicherheitslücken in der Datenbankkommunikation von Webanwendungen aufzudecken.
* **Typische Anwendung:** Überprüfung, ob eine Webanwendung anfällig für SQL-Injektionen ist, die zu Datenlecks führen könnten.

Quelle: <https://en.wikipedia.org/wiki/Sqlmap>

**Metasploit Framework**

* **Zweck:** Ein Framework für Penetrationstests, das hilft, Sicherheitslücken zu finden und zu testen.
* **Fokus:** Bietet eine Sammlung von Tools und Exploits für verschiedene Plattformen und Dienste.
* **Wie:** Ermöglicht Testern, bekannte Schwachstellen auszunutzen, um die Sicherheit eines Systems zu bewerten.
* **Nutzung:** Wird verwendet, um Schwachstellen in Netzwerken und Systemen zu identifizieren und zu testen.
* **Typische Anwendung:** Simuliert Angriffe auf ein System, um dessen Sicherheitsniveau zu überprüfen.

Quelle: <https://en.wikipedia.org/wiki/Metasploit>

**Commix**

* **Zweck:** Automatisches Finden und Ausnutzen von Command-Injection-Schwachstellen.
* **Fokus:** Spezialisiert auf Schwachstellen, die es erlauben, beliebige Systembefehle auszuführen.
* **Wie:** Sendet speziell gestaltete Eingaben an eine Anwendung, um zu testen, ob sie anfällig für Command-Injection-Angriffe ist.
* **Nutzung:** Wird genutzt, um Schwachstellen in der Verarbeitung von Benutzereingaben aufzudecken, die zu unautorisierten Systembefehlen führen könnten.
* **Typische Anwendung:** Überprüfung einer Webanwendung auf Schwachstellen, die es ermöglichen, unberechtigte Systembefehle auszuführen.

Quelle: <https://www.broadcom.com/support/security-center/attacksignatures/detail?asid=29026>

# CSRF

Information

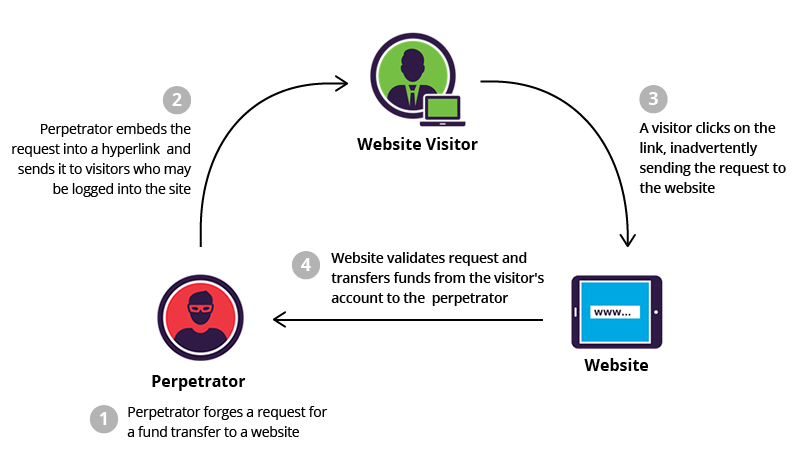
Cross-Site Request Forgery (CSRF) ist eine Web-Sicherheitslücke, bei der ein Angreifer einen Benutzer dazu bringt, unbeabsichtigt eine schädliche Aktion auf einer vertrauenswürdigen Website auszuführen, auf der er angemeldet ist. Dies funktioniert, weil die Website allein auf Sitzungscookies zur Identifikation des Nutzers vertraut und keine zusätzlichen Sicherheitsmaßnahmen wie CSRF-Token verwendet.

Quelle: <https://portswigger.net/web-security/csrf#:~:text=Cross%2Dsite%20request%20forgery%20(also,do%20not%20intend%20to%20perform>.

CSRF (Cross-Site-Request-Forgery) ist eine Angriffsmethode, bei der ein Angreifer eine Aktion in einer Webanwendung über den Browser eines eingeloggten Opfers ausführt, ohne dessen Wissen. Dies geschieht, weil der Browser automatisch Sitzungsdaten (z. B. Cookies) mitschickt, wodurch die Webanwendung die manipulierte Anfrage als legitim ansieht.

Quelle: <https://de.wikipedia.org/wiki/Cross-Site-Request-Forgery>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?php

if( isset( $\_GET[ 'Change' ] ) ) { **//Passwort änderung erolgt als Get-Anfrage, heißt wenn der User einen falschen Link drückt kann er ausversehen das Passwort ändern.**

// 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);

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

**//hier wird ein CSRF Token zur verifizierung verwendet.**

// 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(); **// Es werden Prepared Statements verwendet, das heißt in der URL stehen nicht das Passwort und der User sonder ersatz Platzhalter**

// 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(); **// jedes mal ein neuer Sessiontoken**

?>

Warum ist dieser Code so sicher:  
Dieser Code ist gut gegen CSRF-Angriffe geschützt, da er ein **Anti-CSRF-Token** (checkToken()) verwendet, das überprüft, ob die Anfrage legitim ist, und für jede Sitzung ein neuer Token (generateSessionToken()) generiert wird. Zusätzlich werden **Prepared Statements** genutzt, wodurch Benutzereingaben nicht direkt in die SQL-Abfragen eingefügt werden, was SQL-Injection verhindert und die Sicherheit weiter erhöht.

Tools zum Hacken

**Burp Suite:**

* **Zweck:** Ein umfassendes Framework für Penetrationstests, das Sicherheitslücken in Webanwendungen identifiziert und testet.
* **Fokus:** Bietet eine Sammlung von Tools wie Proxy, Scanner und Intruder für die Analyse und Manipulation von HTTP(S)-Anfragen und -Antworten.
* **Wie:** Erfasst und analysiert Webverkehr, ermöglicht das Modifizieren von Anfragen und das Erzeugen von Proof-of-Concepts für CSRF-Angriffe.
* **Nutzung:** Verwendet, um Sicherheitslücken zu identifizieren, zu testen und zu dokumentieren, insbesondere durch das Generieren von CSRF-Angriffsbeispielen.
* **Typische Anwendung:** Simuliert Angriffe auf Webanwendungen, um deren Sicherheitsniveau zu bewerten, einschließlich der Erstellung von CSRF-Angriffen.

Quelle: <https://en.wikipedia.org/wiki/Burp_Suite>

**OWASP ZAP (Zed Attack Proxy):**

* **Zweck:** Ein Open-Source-Tool für die Sicherheitsprüfung von Webanwendungen, das Schwachstellen erkennt und bei der Sicherheitsauditierung hilft.
* **Fokus:** Bietet Funktionen wie einen Proxy-Server, Spider, Fuzzer und automatisierte Scanner zur Identifizierung von Sicherheitslücken.
* **Wie:** Erfasst und analysiert HTTP(S)-Verkehr, identifiziert potenzielle CSRF-Schwachstellen und ermöglicht das Erzeugen von Angriffsbeispielen.
* **Nutzung:** Verwendet, um Sicherheitslücken in Webanwendungen zu erkennen, zu testen und zu beheben, einschließlich der Generierung von CSRF-Angriffsbeispielen.
* **Typische Anwendung:** Führt Sicherheitstests durch, simuliert Angriffe und bewertet die Sicherheitsstandards von Webanwendungen, einschließlich CSRF-Tests.

Quelle: <https://www.testautomatisierung.org/lexikon/zed-attack-proxy-zap/>

**XSRFProbe:**

* **Zweck:** Ein Tool zur Identifizierung von CSRF-Schwachstellen in Webanwendungen durch Analyse von HTTP-Anfragen und -Antworten.
* **Fokus:** Spezialisiert auf die Erkennung von CSRF-Schwachstellen, indem es Anfragen auf fehlende Schutzmechanismen überprüft.
* **Wie:** Analysiert HTTP-Verkehr, identifiziert potenzielle CSRF-Risiken und bietet Berichte zur Verbesserung der Sicherheit.
* **Nutzung:** Verwendet, um CSRF-Schwachstellen zu erkennen und Empfehlungen zur Absicherung von Webanwendungen bereitzustellen.
* **Typische Anwendung:** Führt gezielte Tests durch, um CSRF-Schwachstellen zu identifizieren und die Sicherheit von Webanwendungen zu erhöhen.

Quelle**:** <https://www.kali.org/tools/xsrfprobe/>

**CSRF PoC Generator:**

* **Zweck:** Ein Tool zur automatisierten Erstellung von Proof-of-Concepts für CSRF-Angriffe, um Sicherheitslücken zu demonstrieren.
* **Fokus:** Erzeugt HTML-Formulare oder Skripte, die potenzielle CSRF-Angriffe simulieren, basierend auf realen Anfragen.
* **Wie:** Analysiert HTTP-Anfragen, extrahiert relevante Parameter und generiert entsprechende Angriffsskripte zur Validierung von Schutzmechanismen.
* **Nutzung:** Verwendet, um potenzielle CSRF-Angriffe zu simulieren und die Effektivität vorhandener Schutzmaßnahmen zu testen.
* **Typische Anwendung:** Erstellt Angriffsskripte zur Demonstration und Prüfung von CSRF-Schwachstellen in Webanwendungen.

Quelle: <https://hacktify.in/csrf/>

# File Inclusion

Information

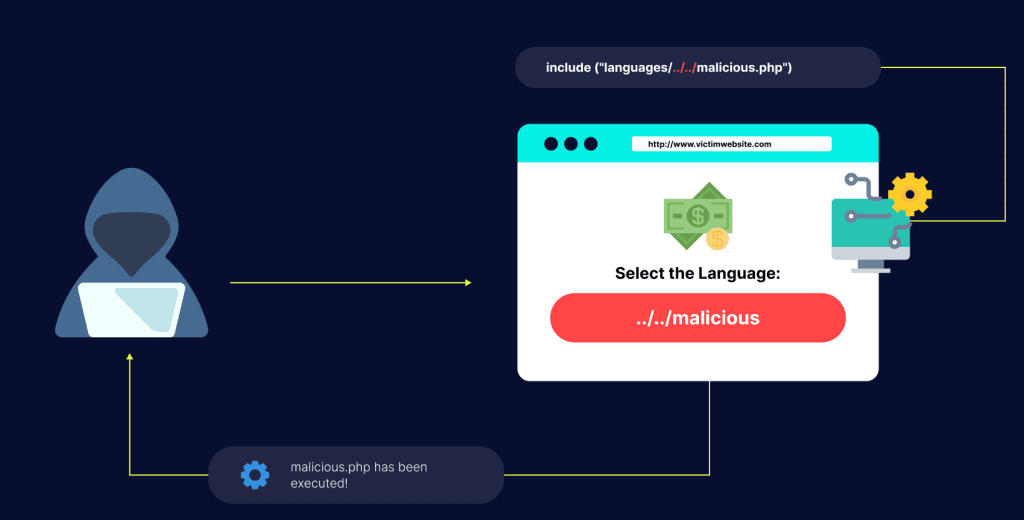
Eine File-Inclusion-Sicherheitslücke tritt auf, wenn eine Webanwendung eine Datei basierend auf einer vom Benutzer gelieferten Eingabe lädt, wodurch ein Angreifer die Kontrolle über die geladene Datei übernehmen kann. Dies kann zu einem Remote-Code-Execution-Angriff führen, bei dem bösartiger Code auf dem Webserver ausgeführt wird. Es gibt zwei Hauptarten: Remote File Inclusion (RFI), bei dem eine Datei von einem externen Server geladen wird, und Local File Inclusion (LFI), bei dem nur lokale Dateien des Servers betroffen sind.

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

File Inclusion-Vulnerabilitäten entstehen, wenn eine Webanwendung es Angreifern erlaubt, beliebige Dateien zu lesen oder auszuführen, indem sie die Dateipfade manipulieren. Es gibt zwei Hauptarten: Local File Inclusion (LFI), bei dem lokale Dateien auf dem Server eingebunden werden, und Remote File Inclusion (RFI), bei dem schadhafter Code von einem externen Server eingebunden wird. Solche Angriffe können zu Datenverlust, Serverkompromittierung oder sogar vollständigem Übernahme der Website führen, wenn Eingaben nicht ausreichend validiert oder gesäubert werden.

Quelle: <https://brightsec.com/blog/file-inclusion-vulnerabilities/>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?php

// The page we wish to display

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

?> **// Der angreifer kann den Wert in der URL ändern und ein nicht öffentliches File laden.**

**Hard to hack:**

<?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 ) ) { **//Hier wird überprüft ob das File wirklich ein Bild ist und keine andere Art von File**

// 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 ); **// Metadaten werden neucodiert, damit falls Schadcode eingebettet ist im PNG file, dieser entfernt wird**

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

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

**//Files werden in einen bestimmten Ordner verschoben, dieser Ornder kann weitere Sicherheitsvorkehrungen haben.**

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 ist dieser Code so sicher:  
Dieser Code verhindert File Inclusion Angriffe, indem er sicherstellt, dass nur echte Bilddateien (JPEG/PNG) hochgeladen werden dürfen, sie durch Neucodierung von Schadcode bereinigt werden und in einen vordefinierten, potenziell abgesicherten Ordner mit zufälligem Namen gespeichert werden.

Tools zum Hacken

**Burp Suite**

* **Zweck**: Ein umfassendes Web-Sicherheitstool zur Identifikation und Analyse von Schwachstellen, einschließlich File Inclusion.
* **Fokus**: Ermöglicht das Testen und Exploitieren von Webanwendungen auf verschiedene Schwachstellen wie LFI, RFI, XSS, SQL Injection, und mehr.
* **Wie**: Burp Suite fängt HTTP-Anfragen ab, analysiert diese und ermöglicht es, Parameter zu manipulieren, um Schwachstellen zu identifizieren. Mit den Tools wie "Intruder" und "Scanner" können RFI und LFI-Angriffe simuliert werden.
* **Nutzung**: Wird verwendet, um Webanwendungen auf File Inclusion und andere Sicherheitslücken zu scannen, zu analysieren und Exploits zu testen.
* **Typische Anwendung**: Sicherheitsforschung und Penetrationstests zur Identifikation von Schwachstellen wie File Inclusion in Webanwendungen.

Quelle: <https://en.wikipedia.org/wiki/Burp_Suite>

**Nikto**

* **Zweck**: Open-Source-Webserver-Scanner zur Identifikation von bekannten Schwachstellen, einschließlich File Inclusion.
* **Fokus**: Überprüft Webserver und Webanwendungen auf eine Vielzahl von Sicherheitslücken, darunter unsichere Konfigurationen und bekannte Schwachstellen wie LFI/RFI.
* **Wie**: Nikto sendet eine Reihe vordefinierter Tests und Anfragen an Webserver, um unsichere Parameter und potenzielle Exploits wie File Inclusion zu erkennen.
* **Nutzung**: Wird verwendet, um Webserver zu scannen und File Inclusion sowie andere Sicherheitslücken zu finden.
* **Typische Anwendung**: Einsatz bei Sicherheits-Audits, um Schwachstellen auf Webservern zu identifizieren, bevor ein gezielter Angriff möglich wird.

Quelle: <https://www.cisa.gov/resources-tools/services/nikto#:~:text=Description,versions%20of%20web%20server%20software>.

**Acunetix**

* **Zweck**: Kommerzielles Sicherheits-Tool zur Durchführung automatisierter Scans von Webanwendungen, einschließlich der Erkennung von File Inclusion-Fehlern.
* **Fokus**: Scannt Webanwendungen auf eine Vielzahl von Schwachstellen, einschließlich File Inclusion, SQL-Injection, Cross-Site Scripting (XSS) und andere.
* **Wie**: Acunetix analysiert Webseiten, erkennt verwundbare Parameter und simuliert Angriffe, um mögliche LFI/RFI-Schwachstellen zu identifizieren.
* **Nutzung**: Wird verwendet, um Schwachstellen wie LFI und RFI in Webanwendungen zu finden und automatisiert zu beheben.
* **Typische Anwendung**: Einsatz zur Durchführung automatisierter Schwachstellen-Scans und zur Sicherstellung der Sicherheitsstandards von Webanwendungen.

Quelle: <https://www.acunetix.com/>

**Wapiti**

* **Zweck**: Open-Source-Webanwendungsscanner, der Sicherheitslücken wie File Inclusion identifiziert.
* **Fokus**: Überprüft Webanwendungen auf potenzielle Schwachstellen wie LFI/RFI und andere Sicherheitslücken in HTTP-Anfragen.
* **Wie**: Wapiti sendet HTTP-Anfragen und testet gezielt Parameter auf Schwachstellen, insbesondere auf unsichere Dateiinclusions.
* **Nutzung**: Wird verwendet, um Webanwendungen auf File Inclusion und andere Schwachstellen zu scannen und diese zu erkennen.
* **Typische Anwendung**: Einsatz bei Penetrationstests und Sicherheitsüberprüfungen von Webanwendungen, um potenzielle LFI/RFI-Lücken zu identifizieren.

Quelle: <https://www.hackercoolmagazine.com/complete-guide-to-wapiti/?srsltid=AfmBOoqhZUwYuAq53IVDvS7fZcR4VMiXahMwgAuTgcyp1Ho8Yp65z72U>

# File Upload

Information

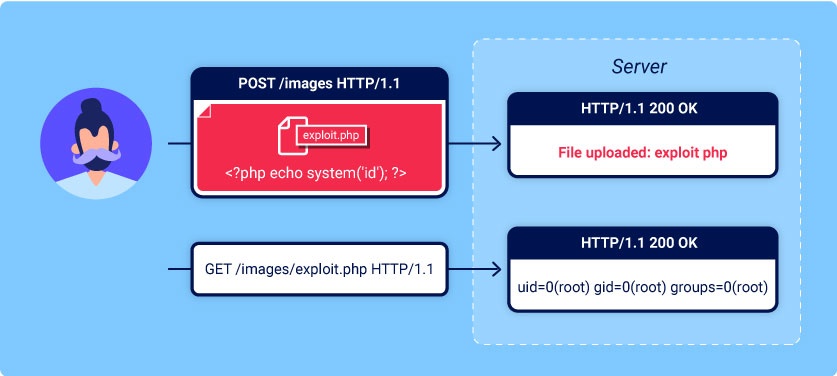
File upload vulnerabilities entstehen, wenn eine Webanwendung es Nutzern erlaubt, Dateien hochzuladen, ohne diese ausreichend zu validieren, etwa in Bezug auf Dateityp, -größe oder -inhalt. Angreifer können diese Schwächen ausnutzen, um gefährliche Dateien, wie Server-Skripte, hochzuladen, die dann vom Server ausgeführt werden können, was zu einer vollständigen Kontrolle über den Server führen kann. Die Ausnutzung solcher Lücken ermöglicht es, Daten zu exfiltrieren oder den Server für weitere Angriffe zu nutzen.

Quelle: <https://portswigger.net/web-security/file-upload>

Unrestricted File Upload ermöglicht es Nutzern, Dateien ohne ausreichende Validierung hochzuladen, was zu Angriffen wie der Übernahme von Servern oder dem Einschleusen von Schadsoftware führen kann. Zur Absicherung sollten Dateinamen gefiltert, die Dateigröße begrenzt und Verzeichnisse ohne Ausführungsrechte eingerichtet werden.

Quelle: <https://www.invicti.com/web-vulnerability-scanner/vulnerabilities/unrestricted-file-upload/>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?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' ] );

**// File typ und Name wird nicht überprüft**

// 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>";

}

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

**//Anti-CSRF-Token wird verwendet**

// 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 ) ) {

**//Dateityp und Dateigröße wird überprüft**

// 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);

}**//Umwandlung und Neucodierung der Files um schädliche Metadaten zu entfernen**

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 ist dieser Code so sicher:

Tools zum Hacken

**Burp Suite**

* **Zweck:** Ein umfassendes Web-Sicherheitstool zur Identifikation und Analyse von Schwachstellen, einschließlich File Inclusion.
* **Fokus:** Ermöglicht das Testen und Exploitieren von Webanwendungen auf verschiedene Schwachstellen wie LFI, RFI, XSS, SQL Injection, und mehr.
* **Wie:** Burp Suite fängt HTTP-Anfragen ab, analysiert diese und ermöglicht es, Parameter zu manipulieren, um Schwachstellen zu identifizieren. Mit den Tools wie "Intruder" und "Scanner" können RFI und LFI-Angriffe simuliert werden.
* **Nutzung:** Wird verwendet, um Webanwendungen auf File Inclusion und andere Sicherheitslücken zu scannen, zu analysieren und Exploits zu testen.
* **Typische Anwendung:** Sicherheitsforschung und Penetrationstests zur Identifikation von Schwachstellen wie File Inclusion in Webanwendungen.

Quelle: <https://en.wikipedia.org/wiki/Burp_Suite>

**OWASP ZAP (Zed Attack Proxy)**

* **Zweck:** Ein Open-Source-Tool zur Durchführung von Sicherheitstests an Webanwendungen, das auf Schwachstellen wie Unrestricted File Upload und viele andere überprüft.
* **Fokus:** Schwerpunkte auf die Identifikation von Sicherheitslücken wie XSS, SQL Injection, File Uploads und anderen Web-Schwachstellen.
* **Wie:** ZAP analysiert den HTTP/HTTPS-Datenverkehr, simuliert Angriffe und scannt Webanwendungen auf Schwachstellen. Es bietet auch automatisierte und manuelle Angriffsmöglichkeiten.
* **Nutzung:** Wird verwendet, um Webanwendungen auf File Upload-Probleme und andere Schwachstellen zu testen, zu scannen und zu protokollieren.
* **Typische Anwendung:** Penetrationstests und Sicherheitsbewertungen von Webanwendungen, insbesondere im Bereich der automatisierten Schwachstellenerkennung.

Quelle: <https://www.testautomatisierung.org/lexikon/zed-attack-proxy-zap/>

**Netsparker (jetzt Invicti)**

* **Zweck:** Ein automatisiertes Web-Sicherheitsscanner-Tool, das Webanwendungen auf eine Vielzahl von Schwachstellen, einschließlich Unrestricted File Uploads, überprüft.
* **Fokus:** Vollständiger Scan von Webanwendungen, insbesondere auf kritische Schwachstellen wie File Upload, SQL Injection, XSS und viele mehr.
* **Wie:** Invicti verwendet eine Kombination aus dynamischer und statischer Analyse, um Sicherheitslücken zu erkennen und automatisch Berichte zu erstellen, die die Schwachstellen und deren Schweregrade dokumentieren.
* **Nutzung:** Wird genutzt, um die Webanwendung auf Schwachstellen wie Unrestricted File Upload zu scannen und detaillierte Berichte zur Behebung zu erstellen.
* **Typische Anwendung:** Sicherheitsanalysen und Compliance-Tests in Unternehmen, die eine automatisierte Erkennung und Berichterstattung von Sicherheitslücken benötigen.

Quelle: <https://www.invicti.com/>

**Acunetix**

* **Zweck:** Ein umfassender Web Application Scanner, der Sicherheitslücken wie Unrestricted File Uploads, XSS, SQL Injection und viele weitere Schwachstellen erkennt.
* **Fokus:** Identifikation und Scannen von Webanwendungen auf Sicherheitslücken, mit besonderem Fokus auf automatisierte Erkennung und Berichterstattung.
* **Wie:** Acunetix verwendet eine Kombination aus dynamischem Scannen und Analyse von Webanwendungen, um Schwachstellen zu identifizieren, insbesondere in Bereichen wie File Upload, Cross-Site Scripting und SQL Injection.
* **Nutzung:** Wird verwendet, um Webanwendungen auf Schwachstellen zu testen, Berichte zu erstellen und zu analysieren, um Schwachstellen zu beheben.
* **Typische Anwendung:** Penetrationstests und kontinuierliche Sicherheitsüberwachung für Webanwendungen in Unternehmen, um Sicherheitslücken wie Unrestricted File Upload zu identifizieren und zu beheben.

Quelle: <https://www.acunetix.com/>

# Insecure CAPTCHA

Information

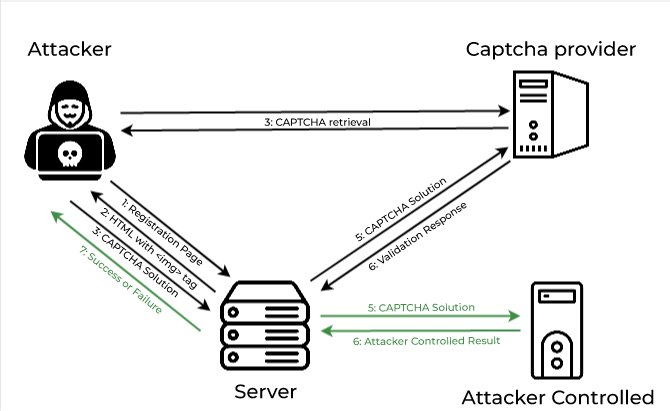
Bei einer Insecure-CAPTCHA-Attacke wird ein unsicher implementiertes CAPTCHA-System umgangen, um automatisiert Aktionen wie Passwortänderungen durchzuführen. Angreifer überspringen dabei oft die CAPTCHA-Überprüfung, manipulieren HTTP-Anfragen oder nutzen Entwicklerfehler aus.

Quelle: <https://cspanias.github.io/posts/DVWA-Insecure-CAPTCHA/>

Insecure CAPTCHA-Angriffe nutzen Schwachstellen in der Implementierung von CAPTCHA-Systemen, um die Mensch-Maschine-Prüfung zu umgehen. Dabei können Angreifer beispielsweise Parameter im Anfrageprozess manipulieren, den CAPTCHA-Status auf "erfolgreich" setzen oder versteckte Entwicklerwerte ausnutzen. Dadurch wird es möglich, automatisierte Angriffe durchzuführen, wie das Ändern von Passwörtern oder das Überlasten von Webseiten mit Anfragen, ohne eine echte CAPTCHA-Herausforderung zu bestehen.

Quelle: <https://christovitohidajat.medium.com/insecure-captcha-dvwa-hacking-for-unsafe-verification-process-8d0d938eb9e9>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?php

if( isset( $\_POST[ 'Change' ] ) && ( $\_POST[ 'step' ] == '1' ) ) {

// Hide the CAPTCHA form

$hide\_form = true;

// Get input

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

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

// Check CAPTCHA from 3rd party

$resp = recaptcha\_check\_answer(

$\_DVWA[ 'recaptcha\_private\_key'],

$\_POST['g-recaptcha-response']

);

// Did the CAPTCHA fail?

if( !$resp ) {

// What happens when the CAPTCHA was entered incorrectly

$html .= "<pre><br />The CAPTCHA was incorrect. Please try again.</pre>";

$hide\_form = false;

return;

}

**//prüft nur ob das Captcha failed, und wenn es NICHT failed dann wird automatisch erwartet, dass es true war**

else {

// CAPTCHA was correct. Do both new passwords match?

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

// Show next stage for the user

echo "

<pre><br />You passed the CAPTCHA! Click the button to confirm your changes.<br /></pre>

<form action=\"#\" method=\"POST\">

<input type=\"hidden\" name=\"step\" value=\"2\" />

<input type=\"hidden\" name=\"password\_new\" value=\"{$pass\_new}\" />

<input type=\"hidden\" name=\"password\_conf\" value=\"{$pass\_conf}\" />

<input type=\"submit\" name=\"Change\" value=\"Change\" />

</form>";

}

else {

// Both new passwords do not match.

$html .= "<pre>Both passwords must match.</pre>";

$hide\_form = false;

}

}

}

if( isset( $\_POST[ 'Change' ] ) && ( $\_POST[ 'step' ] == '2' ) ) {

**// Ein Angreifer kann direkt step=2 senden und das Passwort ändern, ohne das CAPTCHA auszufüllen.**

// Hide the CAPTCHA form

$hide\_form = true;

// Get input

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

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

// Check to see if both password 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 database

$insert = "UPDATE `users` SET password = '$pass\_new' WHERE user = '" . dvwaCurrentUser() . "';";

$result = mysqli\_query($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 end user

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

}

else {

// Issue with the passwords matching

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

$hide\_form = false;

}

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

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

// Hide the CAPTCHA form

$hide\_form = true;

// Get input

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

$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 );

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

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

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

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

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

$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 CAPTCHA from 3rd party

$resp = recaptcha\_check\_answer(

$\_DVWA[ 'recaptcha\_private\_key' ],

$\_POST['g-recaptcha-response']

);

// Did the CAPTCHA fail?

if( !$resp ) {

// What happens when the CAPTCHA was entered incorrectly

echo "<pre><br />The CAPTCHA was incorrect. Please try again.</pre>";

$hide\_form = false;

}**// Ist der die Captcha-Lösung ungültig so wird es abgebrochen**

else {

// Check that the current password is correct

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

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

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

$data->execute();

// Do both new password match and was the current password correct?

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

// Update the database

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

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

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

$data->execute();

// Feedback for the end user - success!

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

}

else {

// Feedback for the end user - failed!

echo "<pre>Either your current password is incorrect or the new passwords did not match.<br />Please try again.</pre>";

$hide\_form = false;

}

}

}

// Generate Anti-CSRF token

generateSessionToken();

**//Genereriert ein einzigartiges Token**

?>

Warum ist dieser Code so sicher:

Dieser Code ist sicher gegen "Insecure CAPTCHA"-Attacken, weil er ein externes reCAPTCHA-System nutzt, das serverseitig streng validiert wird, zusätzlich durch ein CSRF-Token geschützt ist und den gesamten Prozess abbricht, falls das CAPTCHA falsch oder manipuliert ist.

Tools zum Hacken

**2Captcha**

• Zweck: Ein kommerzieller Dienst, der CAPTCHAs durch menschliche Arbeiter löst, um Automatisierungstools zu unterstützen.  
**• Fokus:** Bypass von CAPTCHA-Systemen (reCAPTCHA, hCAPTCHA, Bild-CAPTCHAs) für automatisierte Angriffe wie Brute-Force oder Web-Scraping.  
**• Wie:** Nutzer senden CAPTCHA-Bilder oder Tokens an die 2Captcha-API, wo echte Menschen sie lösen. Die Lösung wird zurückgegeben und kann in Skripte eingebaut werden.  
**• Nutzung:** Wird von Angreifern genutzt, um CAPTCHA-geschützte Formulare (Login, Registrierung, Passwort-Reset) automatisiert zu überwinden.  
**• Typische Anwendung:** Massenhaftes Erstellen von Fake-Accounts, Credential-Stuffing oder Spam-Bots.

Quelle: <https://2captcha.com/about>

**DeathByCaptcha**

**• Zweck:** Ähnlich wie 2Captcha, bietet automatisierte CAPTCHA-Lösungen durch menschliche Solver und KI.  
**• Fokus:** Schnelle CAPTCHA-Entschlüsselung für Bot-Angriffe auf Webformulare.  
**• Wie:** API-basierter Dienst, der CAPTCHAs entweder durch Crowdworker oder OCR-Tools löst. Unterstützt reCAPTCHA, hCAPTCHA und Bild-CAPTCHAs.  
**• Nutzung:** Wird in Kombination mit Tools wie Selenium oder cURL verwendet, um CAPTCHAs in Echtzeit zu umgehen.  
**• Typische Anwendung:** Automatisierte Credential-Cracking-Angriffe auf Login-Seiten mit CAPTCHA-Schutz.

Ein Bild, das Grafiken, Schrift, Grafikdesign, Grün enthält.

KI-generierte Inhalte können fehlerhaft sein.Quelle: <https://blog.deathbycaptcha.com/api>

**Selenium WebDriver**  
• **Zweck**: Browser-Automatisierungstool zur Umgehung von CAPTCHA-geschützten Webformularen.  
• **Fokus**: Simulation menschlicher Browserinteraktionen zur CAPTCHA-Umgehung, besonders bei clientseitig oder schwach implementierten CAPTCHAs.  
• **Wie**: Steuert echte Browser (Chrome/Firefox) via Skripten (Python/Java/JS) und kann mit CAPTCHA-Solver-APIs gekoppelt werden.  
• **Nutzung**: Automatisierte Angriffe auf CAPTCHA-geschützte Login-/Registrierungsseiten.  
• **Typische** **Anwendung**: Credential-Stuffing-Angriffe und Massenerstellung von Fake-Accounts.

Quelle: <https://www.selenium.dev/documentation/webdriver/>

**Tesseract OCR**• **Zweck**: Open-Source-OCR-Engine zur Textextraktion aus CAPTCHA-Bildern.  
• **Fokus**: Umgehung einfacher, selbstgebauter CAPTCHAs (verzerrte Buchstaben/Zahlen).  
• **Wie**: Analysiert CAPTCHA-Bilder mittels optischer Zeichenerkennung, typischerweise in Python-Skripten integriert.  
• **Nutzung**: Effektiv gegen primitive CAPTCHAs ohne erweiterte Sicherheitsmerkmale.  
• **Typische** **Anwendung**: Angriffe auf Systeme mit statischen CAPTCHAs und automatisiertes Web-Scraping.

Quelle: <https://tesseract-ocr.github.io/tessdoc/>

# SQL Injection

Information

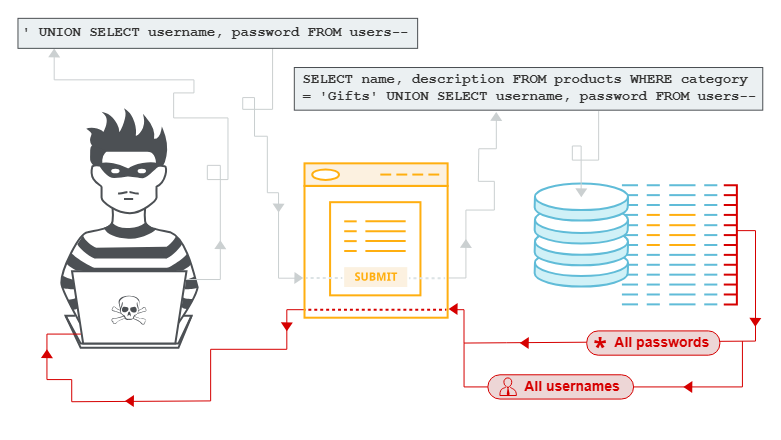
SQL Injection ist eine Angriffstechnik, bei der ein Angreifer schädliche SQL-Befehle in Eingabefelder einer Webanwendung einschleust. Diese Befehle werden dann an die Datenbank weitergeleitet und ausgeführt, da die Anwendung die Eingaben nicht ausreichend überprüft oder bereinigt. Durch diese Manipulation kann der Angreifer sensible Daten auslesen, ändern oder sogar löschen sowie administrative Befehle ausführen. Typischerweise nutzt der Angreifer Metazeichen wie Anführungszeichen, um die SQL-Abfrage zu verändern und unerwünschte Aktionen auszulösen.

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

SQL Injection (SQLi) ist eine Sicherheitsschwachstelle, die es Angreifern ermöglicht, bösartige SQL-Befehle in Eingabefelder einer Webanwendung einzuschleusen. Dadurch können sie Datenbankabfragen manipulieren, um unbefugt sensible Daten auszulesen, zu ändern oder sogar administrative Befehle auszuführen. Typische Angriffe nutzen spezielle Zeichen (wie Anführungszeichen oder Kommentarzeichen), um die ursprüngliche SQL-Abfrage zu umgehen. Effektiver Schutz bietet die Verwendung von parametrisierten Abfragen (Prepared Statements), die Benutzereingaben sicher behandeln.

Quelle: <https://portswigger.net/web-security/sql-injection>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?php

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

// Get input

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

**//Fehlende Überprüfung**

switch ($\_DVWA['SQLI\_DB']) {

case MYSQL:

// Check database

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

**// Führt automatisch alles im String aus ohne zu überprüfen**

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

// Get results

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

// Get values

$first = $row["first\_name"];

$last = $row["last\_name"];

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]);

break;

case SQLITE:

global $sqlite\_db\_connection;

#$sqlite\_db\_connection = new SQLite3($\_DVWA['SQLITE\_DB']);

#$sqlite\_db\_connection->enableExceptions(true);

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

#print $query;

try {

$results = $sqlite\_db\_connection->query($query);

} catch (Exception $e) {

echo 'Caught exception: ' . $e->getMessage();

exit();

}

if ($results) {

while ($row = $results->fetchArray()) {

// Get values

$first = $row["first\_name"];

$last = $row["last\_name"];

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

} else {

echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

}

break;

}

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

// Get input

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

// Was a number entered?

if(is\_numeric( $id )) {

**//erzwingt ein INT**

$id = intval ($id);

switch ($\_DVWA['SQLI\_DB']) {

case MYSQL:

// Check the database

$data = $db->prepare( 'SELECT first\_name, last\_name FROM users WHERE user\_id = (:id) LIMIT 1;' );

$data->bindParam( ':id', $id, PDO::PARAM\_INT );

$data->execute();

**//Trennt Query-Struktur von Nutzdaten → Keine direkte Code-Einschleusung möglich.**

$row = $data->fetch();

// Make sure only 1 result is returned

if( $data->rowCount() == 1 ) {

// Get values

$first = $row[ 'first\_name' ];

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

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

break;

case SQLITE:

global $sqlite\_db\_connection;

$stmt = $sqlite\_db\_connection->prepare('SELECT first\_name, last\_name FROM users WHERE user\_id = :id LIMIT 1;' );

$stmt->bindValue(':id',$id,SQLITE3\_INTEGER);

$result = $stmt->execute();

$result->finalize();

if ($result !== false) {

// There is no way to get the number of rows returned

// This checks the number of columns (not rows) just

// as a precaution, but it won't stop someone dumping

// multiple rows and viewing them one at a time.

$num\_columns = $result->numColumns();

if ($num\_columns == 2) {

$row = $result->fetchArray();

// Get values

$first = $row[ 'first\_name' ];

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

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

}

break;

}

}

}

// Generate Anti-CSRF token

generateSessionToken();

?>

Warum ist dieser Code so sicher:

Tools zum Hacken

**SQLmap**

**• Zweck:** Automatisierte Erkennung und Ausnutzung von SQL-Injection-Schwachstellen.  
**• Fokus:** Unterstützt klassische SQLi und Blind SQLi (Time-Based, Boolean-Based).  
**• Wie:** Sendet manipulierte Anfragen, erkennt Datenbanktypen und extrahiert Daten automatisch.  
**• Nutzung:** Effektiv gegen unsichere Webformulare, Login-Felder und API-Endpunkte.  
**• Typische Anwendung:** Automatisierte Datenbank-Exfiltration (z. B. Passwörter, Benutzerdaten).  
**Quelle:** [sqlmap.org](https://sqlmap.org/)

**Burp Suite**

**• Zweck:** Manuelle und halbautomatische Schwachstellenerkennung, inkl. SQLi.  
**• Fokus:** Analyse von HTTP-Anfragen für klassische und Blind SQLi (via Time-Delays/Error-Verhalten).  
**• Wie:** Intercept von Requests, Manipulation von Parametern und Beobachtung der Serverantworten.  
**• Nutzung:** Ideal für komplexe Angriffe, wo SQLmap versagt (z. B. WAF-Umgehung).  
**• Typische Anwendung:** Penetrationstests gegen Web-Apps mit individueller Logik.  
**Quelle:** [portswigger.net/burp](https://portswigger.net/burp)

**Havij**

**• Zweck:** GUI-basiertes Tool für einfache SQL-Injection-Angriffe.  
**• Fokus:** Automatisierte Datenbank-Abfragen (auch Blind SQLi via Boolescher Logik).  
**• Wie:** Punkt-&-Klick-Oberfläche zum Injizieren von Payloads und Auslesen von Tabellen.  
**• Nutzung:** Schnelle Angriffe auf verwundbare Websites mit minimalem Aufwand.  
**• Typische Anwendung:** Einsteigerfreundliche Datenbank-Exploitation (z. B. CMS wie WordPress).

Quelle: <https://isc.sans.edu/diary/11011>

**OWASP ZAP (Zed Attack Proxy)**

**• Zweck:** Open-Source-Sicherheitstool für automatisierte/manuelle SQLi-Tests.  
**• Fokus:** Erkennt klassische und Blind SQLi durch Fuzzing und aktive Scans.  
**• Wie:** Sendet gezielte Payloads und prüft auf Anomalien in Responses (Fehler/Verzögerungen).  
**• Nutzung:** Integriert in CI/CD-Pipelines oder manuelle Pentests.  
**• Typische Anwendung:** Sicherheitsaudits von Web-Apps mit API-Schnittstellen.  
**Quelle:** [owasp.org/zap](https://owasp.org/zap/)

# SQL Injection (Blind)

Information

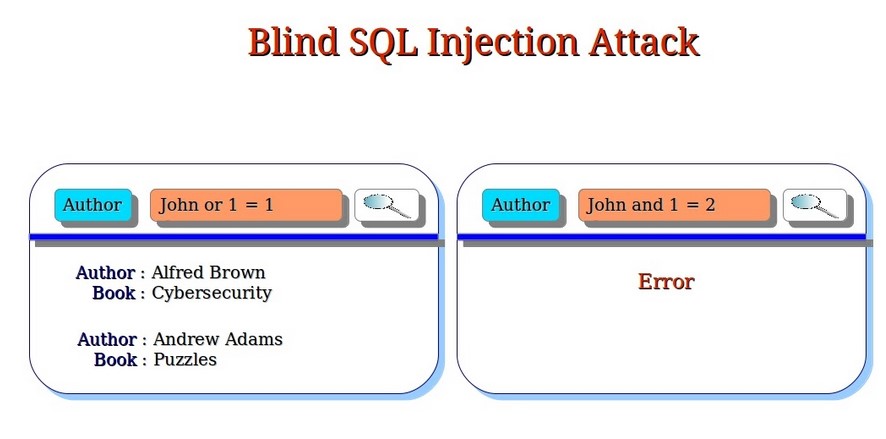
**SQL Injection Blind** ist eine Angriffstechnik, bei der ein Angreifer schädliche SQL-Befehle an eine Datenbank sendet, ohne direkte Rückmeldungen zu erhalten. Im Gegensatz zur klassischen SQL-Injection zeigt die Anwendung keine Fehlermeldungen oder Daten an, sondern reagiert nur indirekt, z. B. durch unterschiedliche Antwortzeiten oder Verhaltensweisen. Der Angreifer muss durch systematisches Ausprobieren („Blindes Erraten“) auf die Daten schließen. Diese Methode ist besonders tückisch, da sie schwerer zu erkennen ist.

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

Bei **Blind SQL Injection** nutzt ein Angreifer Sicherheitslücken in Webanwendungen aus, ohne dass die Datenbank direkte Ergebnisse preisgibt. Stattdessen werden boolesche Abfragen (TRUE/FALSE) oder Zeitverzögerungen genutzt, um Informationen abzuleiten. Beispielsweise kann eine langsame Antwortzeit auf eine erfolgreiche Injection hindeuten. Diese Technik erfordert Geduld und viele Anfragen, ist aber effektiv, wenn Standard-SQL-Injection blockiert wird. Prävention erfolgt durch Prepared Statements und Input-Validierung.

Quelle: <https://portswigger.net/web-security/sql-injection/blind>

Visuelles Beispiel



Code-Beispiel

**Easy to hack:**

<?php

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

// Get input

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

**// Fehlende Überprüfung**

switch ($\_DVWA['SQLI\_DB']) {

case MYSQL:

// Check database

$query = "SELECT first\_name, last\_name FROM users WHERE user\_id = '$id';";  
 **// Führt automatisch alles im String aus ohne zu überprüfen**

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

// Get results

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

// Get values

$first = $row["first\_name"];

$last = $row["last\_name"];

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

mysqli\_close($GLOBALS["\_\_\_mysqli\_ston"]);

break;

case SQLITE:

global $sqlite\_db\_connection;

#$sqlite\_db\_connection = new SQLite3($\_DVWA['SQLITE\_DB']);

#$sqlite\_db\_connection->enableExceptions(true);

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

#print $query;

try {

$results = $sqlite\_db\_connection->query($query);

} catch (Exception $e) {

echo 'Caught exception: ' . $e->getMessage();

exit();

}

if ($results) {

while ($row = $results->fetchArray()) {

// Get values

$first = $row["first\_name"];

$last = $row["last\_name"];

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

} else {

echo "Error in fetch ".$sqlite\_db->lastErrorMsg();

}

break;

}

}

?>

**Hard to hack:**

<?php

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

// Check Anti-CSRF token

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

// Get input

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

// Was a number entered?

if(is\_numeric( $id )) {

**//erzwingt ein INT**

$id = intval ($id);

switch ($\_DVWA['SQLI\_DB']) {

case MYSQL:

// Check the database

$data = $db->prepare( 'SELECT first\_name, last\_name FROM users WHERE user\_id = (:id) LIMIT 1;' );

$data->bindParam( ':id', $id, PDO::PARAM\_INT );

$data->execute();

**//Trennt Query-Struktur von Nutzdaten → Keine direkte Code-Einschleusung möglich.**

$row = $data->fetch();

// Make sure only 1 result is returned

if( $data->rowCount() == 1 ) {

// Get values

$first = $row[ 'first\_name' ];

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

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

break;

case SQLITE:

global $sqlite\_db\_connection;

$stmt = $sqlite\_db\_connection->prepare('SELECT first\_name, last\_name FROM users WHERE user\_id = :id LIMIT 1;' );

$stmt->bindValue(':id',$id,SQLITE3\_INTEGER);

$result = $stmt->execute();

$result->finalize();

if ($result !== false) {

// There is no way to get the number of rows returned

// This checks the number of columns (not rows) just

// as a precaution, but it won't stop someone dumping

// multiple rows and viewing them one at a time.

$num\_columns = $result->numColumns();

if ($num\_columns == 2) {

$row = $result->fetchArray();

// Get values

$first = $row[ 'first\_name' ];

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

// Feedback for end user

echo "<pre>ID: {$id}<br />First name: {$first}<br />Surname: {$last}</pre>";

}

}

break;

}

}

}

// Generate Anti-CSRF token

generateSessionToken();

?>

Warum ist dieser Code so sicher:

Tools zum Hacken

**SQLmap**

**• Zweck:** Automatisierte Erkennung und Ausnutzung von SQL-Injection-Schwachstellen.  
**• Fokus:** Unterstützt klassische SQLi und Blind SQLi (Time-Based, Boolean-Based).  
**• Wie:** Sendet manipulierte Anfragen, erkennt Datenbanktypen und extrahiert Daten automatisch.  
**• Nutzung:** Effektiv gegen unsichere Webformulare, Login-Felder und API-Endpunkte.  
**• Typische Anwendung:** Automatisierte Datenbank-Exfiltration (z. B. Passwörter, Benutzerdaten).  
**Quelle:** [sqlmap.org](https://sqlmap.org/)

**Burp Suite**

**• Zweck:** Manuelle und halbautomatische Schwachstellenerkennung, inkl. SQLi.  
**• Fokus:** Analyse von HTTP-Anfragen für klassische und Blind SQLi (via Time-Delays/Error-Verhalten).  
**• Wie:** Intercept von Requests, Manipulation von Parametern und Beobachtung der Serverantworten.  
**• Nutzung:** Ideal für komplexe Angriffe, wo SQLmap versagt (z. B. WAF-Umgehung).  
**• Typische Anwendung:** Penetrationstests gegen Web-Apps mit individueller Logik.  
**Quelle:** [portswigger.net/burp](https://portswigger.net/burp)

**Havij**

**• Zweck:** GUI-basiertes Tool für einfache SQL-Injection-Angriffe.  
**• Fokus:** Automatisierte Datenbank-Abfragen (auch Blind SQLi via Boolescher Logik).  
**• Wie:** Punkt-&-Klick-Oberfläche zum Injizieren von Payloads und Auslesen von Tabellen.  
**• Nutzung:** Schnelle Angriffe auf verwundbare Websites mit minimalem Aufwand.  
**• Typische Anwendung:** Einsteigerfreundliche Datenbank-Exploitation (z. B. CMS wie WordPress).

Quelle: <https://isc.sans.edu/diary/11011>

**OWASP ZAP (Zed Attack Proxy)**

**• Zweck:** Open-Source-Sicherheitstool für automatisierte/manuelle SQLi-Tests.  
**• Fokus:** Erkennt klassische und Blind SQLi durch Fuzzing und aktive Scans.  
**• Wie:** Sendet gezielte Payloads und prüft auf Anomalien in Responses (Fehler/Verzögerungen).  
**• Nutzung:** Integriert in CI/CD-Pipelines oder manuelle Pentests.  
**• Typische Anwendung:** Sicherheitsaudits von Web-Apps mit API-Schnittstellen.  
**Quelle:** [owasp.org/zap](https://owasp.org/zap/)

# Weak Session Ids

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# XSS (DOM)

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# XSS (Reflected)

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# XSS (Stored)

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# CSP Bypass

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# JavaScript

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# Authorisation Bypass

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

# Open HTTP Redirect

Information

Quelle:

Visuelles Beispiel

Code-Beispiel

**Easy to hack:**

**Hard to hack:**

Warum ist dieser Code so sicher:

Tools zum Hacken

Test lernen  
Tools:  
Burpsuite, OWASP ZAP, Metasploit, SQLMAP,