Keller Ruben

DVWA Dokumentation

Contents

[1. Grundlagen 2](#_Toc188004119)

[2. Brute Force 2](#_Toc188004120)

[3. Command Injection 3](#_Toc188004121)

[4. CSRF 3](#_Toc188004122)

[5. File Inclusion 3](#_Toc188004123)

[6. File Upload 3](#_Toc188004124)

[7. Insecure CAPTCHA 3](#_Toc188004125)

[8. SQL Injection 3](#_Toc188004126)

[9. SQL Injection (Blind) 3](#_Toc188004127)

[10. Weak Session Ids 4](#_Toc188004128)

[11. XSS (DOM) 4](#_Toc188004129)

[12. XSS (Reflected) 4](#_Toc188004130)

[13. XSS (Stored) 4](#_Toc188004131)

[14. CSP Bypass 4](#_Toc188004132)

[15. JavaScript 4](#_Toc188004133)

[16. Authorisation Bypass 4](#_Toc188004134)

[17. Open HTTP Redirect 4](#_Toc188004135)

# 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 (pro und contra) courier new code

**Easy to hack:**  
<?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);

}

?>

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

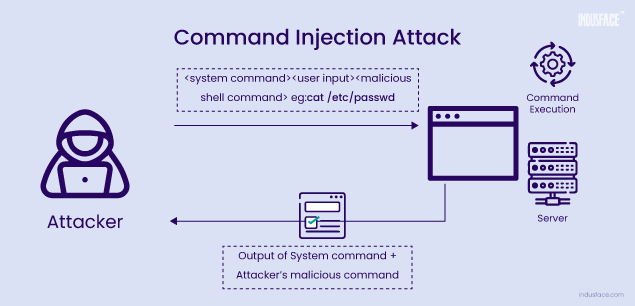
Command Injection ist eine Cyberangriffsmethode, bei der ein Hacker schädliche Befehle in ein System einschleust, indem er eine Sicherheitslücke in einer Anwendung ausnutzt. Dadurch kann der Angreifer Systembefehle ausführen, Daten stehlen, verändern oder das gesamte System übernehmen. Schutzmaßnahmen 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



Code-Beispiel (pro und contra) courier new code

**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:

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 es 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

Quelle:

Visuelles Beispiel

Code-Beispiel (pro und contra) courier new code

Warum ist dieser Code so sicher:

Tools zum Hacken

# File Inclusion

Information

Quelle:

Visuelles Beispiel

Code-Beispiel (pro und contra) courier new code

Warum ist dieser Code so sicher:

Tools zum Hacken

# File Upload

Information

Quelle:

Visuelles Beispiel

Code-Beispiel (pro und contra) courier new code

Warum ist dieser Code so sicher:

Tools zum Hacken

# Insecure CAPTCHA

# SQL Injection

# SQL Injection (Blind)

# Weak Session Ids

# XSS (DOM)

# XSS (Reflected)

# XSS (Stored)

# CSP Bypass

# JavaScript

# Authorisation Bypass

# Open HTTP Redirect