

Cebrail Demir 4aWi

DVWA-Protokoll

[Brute Force 3](#_Toc159223568)

[Command Injection 3](#_Toc159223569)

[C SRF 3](#_Toc159223570)

[File Inclusion 3](#_Toc159223571)

[File Upload 3](#_Toc159223572)

[Insecure CAPTCHA 3](#_Toc159223573)

[SQL Injection 3](#_Toc159223574)

[SQL Injection (Blind) 3](#_Toc159223575)

[Weak Session IDs 3](#_Toc159223576)

[XSS (DOM) 3](#_Toc159223577)

[XSS (Reflected) 3](#_Toc159223578)

[XSS (Stored) 3](#_Toc159223579)

[CSP Bypass 3](#_Toc159223580)

[Javascript 3](#_Toc159223581)

[Authorisation Bypass 3](#_Toc159223582)

[Open HTTP Redirect 3](#_Toc159223583)

# Brute Force

// 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;

// 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();

Theorie:

Brute-Force-Angriffe beinhalten den Versuch eines Angreifers, Zugang zu einem System zu erlangen, indem er wiederholt verschiedene Kombinationen von Benutzernamen und Passwörtern ausprobiert.

Beispiel 1:

Der Angreifer könnte eine Webseite attackieren, indem er bekannte Seiten ausprobiert und überprüft, ob sie existieren.

Brute Force PHP-Schnipsel:

GET-Parameter für Benutzeranmeldeinformationen:

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

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

Unsicherheit: Die Verwendung von GET-Parametern ist riskant, da sie in der URL sichtbar sind und leicht abgefangen oder manipuliert werden können.

Direkte SQL-Zusammensetzung:

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

Unsicherheit: Die direkte Zusammensetzung von SQL-Abfragen macht die Anwendung anfällig für SQL-Injektionen, da Angriffe durch Einfügen von schädlichem SQL-Code in die Eingabefelder möglich sind.

Fehlen von Sicherheitsprüfungen:

Es gibt keine Überprüfung auf Anti-CSRF-Token oder Begrenzung fehlgeschlagener Anmeldeversuche, was die Anwendung anfälliger für Brute-Force-Angriffe macht.

# Command Injection

Command Injection ist ein Angriff, bei dem ein Angreifer versucht, beliebige Befehle auf dem Host-Betriebssystem über eine anfällige Anwendung auszuführen. Dies geschieht häufig, wenn eine Anwendung unsichere benutzerdefinierte Daten (Formulare, Cookies, HTTP-Header usw.) an eine Systemshell übergibt. Durch unzureichende Eingabeüberprüfung können Angreifer Betriebssystembefehle mit den Berechtigungen der anfälligen Anwendung ausführen. Im Gegensatz zur Code Injection erweitert der Angreifer bei der Command Injection die Standardfunktionalität der Anwendung, um Befehle auszuführen, ohne Code einzufügen.

**Beispiele:**

Ein Wrapper-Code um den UNIX-Befehl "cat" ermöglicht es, Dateiinhalte auf die Standardausgabe zu drucken. Durch Hinzufügen eines Semikolons und eines weiteren Befehls kann ein Angreifer jedoch beliebige Befehle ausführen.

Ein Programm, das Dateiinhalte anzeigt und mit Root-Rechten läuft, ermöglicht durch die unzureichende Überprüfung von Benutzereingaben das Ausführen von schädlichen Befehlen wie "rm -rf /", was zur Löschung wichtiger Systemdateien führen kann.

Ein privilegiertes Programm führt einen Initialisierungsskript ohne ausreichende Überprüfung von Umgebungsvariablen aus, was es einem Angreifer ermöglicht, schädlichen Code auszuführen.

Ein CGI-Dienst zum Ändern von Passwörtern ruft den Befehl "make" auf, ohne den vollständigen Pfad anzugeben, was es einem Angreifer ermöglicht, das Skript mit Root-Rechten auszuführen, indem er den Pfad der Umgebungsvariable $PATH ändert.

Ein C-Programm und ein PHP-Skript, das Benutzereingaben direkt an die Systembefehle übergibt, sind anfällig für Befehlsinjektion, wenn nicht ordnungsgemäß validiert wird.

Tipp:

Um solche Angriffe zu verhindern, sollten Entwickler vorhandene APIs verwenden, Eingaben auf bösartige Zeichen überprüfen und wenn möglich einen positiven Sicherheitsansatz implementieren.

**Beispiel 1:**

Command Injection PHP Snippet:

Direkte Verwendung von Benutzereingaben im Befehl:

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

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

Unsicherheit: Die Benutzereingabe wird direkt in den Shell-Befehl eingefügt, was anfällig für Command Injection ist. Ein Angreifer könnte bösartige Befehle einschleusen und auf dem Server beliebigen Code ausführen.

**Beispiel 2:**

Verbessertes PHP Snippet:

Überprüfung der Benutzereingabe:

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

$target = stripslashes($target);

Sicherheit: Die Benutzereingabe wird zuerst mit stripslashes() behandelt, um mögliche Escape-Sequenzen zu entfernen. Dies reduziert das Risiko von SQL-Injektionen, obwohl es keine direkte Command Injection verhindert.

Überprüfung der IP-Struktur:

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

if ((is\_numeric($octet[0])) && (is\_numeric($octet[1])) &&

(is\_numeric($octet[2])) && (is\_numeric($octet[3])) && (sizeof($octet) == 4)) {

// Valid IP

} else {

// Invalid IP

}

Sicherheit: Die Benutzereingabe wird in einzelne Oktette aufgeteilt, und jedes Oktett wird auf numerischen Wert überprüft. Dies stellt sicher, dass die Eingabe einer IP-Adresse entspricht und nicht einem bösartigen Befehl.

Verwendung von Shell-Befehlen:

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

Unsicherheit: Obwohl einige Überprüfungen durchgeführt werden, um sicherzustellen, dass die Eingabe einer IP-Adresse entspricht, besteht immer noch die Möglichkeit einer Command Injection, insbesondere wenn der Benutzer die IP manipuliert, um einen bösartigen Befehl einzuschleusen.

# C SRF

Cross-Site Request Forgery (CSRF) ist ein Angriff, bei dem ein Angreifer einen authentifizierten Benutzer dazu bringt, ungewollte Aktionen auf einer Webanwendung auszuführen. Dies kann durch Social Engineering erfolgen, indem der Angreifer einen Link per E-Mail oder Chat an die Benutzer sendet und sie dazu verleitet, Aktionen auszuführen, die der Angreifer festgelegt hat.

CSRF-Angriffe zielen auf Funktionen ab, die eine Zustandsänderung auf dem Server verursachen, wie z.B. die Änderung der E-Mail-Adresse oder des Passworts des Opfers oder den Kauf von etwas. Ein Angreifer kann auch versuchen, über einen speziellen Angriffstyp namens "Login CSRF" an private Daten des Opfers zu gelangen, indem er einen nicht authentifizierten Benutzer dazu zwingt, sich bei einem Konto anzumelden, das der Angreifer kontrolliert.

Es gibt verschiedene Bezeichnungen für CSRF-Angriffe, darunter XSRF, "Sea Surf", Session Riding, Cross-Site Reference Forgery und Hostile Linking. Microsoft bezeichnet diesen Angriffstyp als One-Click-Angriff in ihrem Bedrohungsmodellierungsprozess.

Zur Überprüfung auf CSRF-Sicherheitslücken im Code sowie zur Vorbeugung und Testung von CSRF-Anfälligkeiten gibt es verschiedene Methoden. Viele Frameworks bieten bereits eingebaute CSRF-Unterstützung, und spezielle Tools wie der OWASP CSRF Guard und das CSRFProtector-Projekt schützen Webanwendungen vor CSRF-Angriffen.

Es gibt jedoch auch einige Maßnahmen zur Verhinderung von CSRF-Angriffen, die nicht funktionieren, wie z.B. die Verwendung eines geheimen Cookies. Alle Cookies, auch die geheimen, werden bei jedem Request übermittelt, unabhängig davon, ob der Endbenutzer dazu getäuscht wurde, den Request zu senden. Daher sind geheime Cookies keine effektive Maßnahme zur Abwehr von CSRF-Angriffen.

Beispiel 1:

CSRF PHP-Schnipsel:

Fehlen eines CSRF-Schutzes:

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

// ...

}

Unsicherheit: Das Schnipsel überprüft nicht, ob die Anfrage von einem autorisierten Benutzer stammt, was es anfällig für CSRF-Angriffe macht. Ein Angreifer kann einen bösartigen Link erstellen und einen authentifizierten Benutzer dazu bringen, diesen Link zu öffnen, wodurch der Benutzer unwissentlich eine Aktion auf der Seite ausführt.

Beispiel 2:

Verbessertes PHP-Schnipsel:

Überprüfung des Anti-CSRF-Tokens:

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

Sicherheit: Das Schnipsel enthält eine Funktion checkToken, die das Anti-CSRF-Token des Benutzers überprüft, um sicherzustellen, dass die Anfrage von einem autorisierten Benutzer stammt. Dadurch wird die Anwendung vor CSRF-Angriffen geschützt.

# File Inclusion

Eine Dateieinschluss-Schwachstelle (File Inclusion) ermöglicht es einem Angreifer, Dateien auf einer Website auszuführen, die er normalerweise nicht ausführen dürfte. Diese Schwachstelle tritt auf, wenn die Website unsicher programmiert ist und es dem Angreifer erlaubt, den Pfad zu einer Datei zu manipulieren, die dann vom Webserver geladen und ausgeführt wird. Durch Ausnutzen dieser Schwachstelle kann der Angreifer bösartigen Code einschleusen, der entweder den Betrieb der Website stört, sensible Daten stiehlt oder sogar die Kontrolle über die gesamte Website übernimmt.

File Inclusion PHP Snippet:

Unbeschränkter Dateizugriff:

// Die Seite, die wir anzeigen möchten

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

Unsicherheit: Das Snippet weist keine Beschränkungen oder Validierungen für die $file-Variable auf, was es anfällig für LFI (Local File Inclusion) oder RFI (Remote File Inclusion) macht. Ein Angreifer könnte bösartige Dateien wie PHP-Skripte von einem externen Server einschleusen oder sensible Dateien auf dem Server einbeziehen.

# File Upload

Durch das Hochladen von Dateien kann ein Angreifer Schadcode auf den Server einschleusen und ausführen, was zu verschiedenen Arten von Angriffen führen kann. Dies kann zu einer Überlastung des Dateisystems oder der Datenbank, zu Angriffen auf Backend-Systeme, zu Client-seitigen Angriffen oder einfach zur Verunstaltung der Webseite führen. Die Auswirkungen dieser Sicherheitslücke sind hoch, da der Angreifer Code im Serverkontext oder auf der Client-Seite ausführen kann.

Einfacher Dateiupload:

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

// Zielverzeichnis für den Dateiupload

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

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

// Datei in das Zielverzeichnis verschieben

if( !move\_uploaded\_file( $\_FILES[ 'uploaded' ][ 'tmp\_name' ], $target\_path ) ) {

// Fehlermeldung, wenn der Dateiupload fehlschlägt

echo '<pre>Your image was not uploaded.</pre>';

}

else {

// Erfolgsmeldung nach erfolgreichem Dateiupload

echo "<pre>{$target\_path} succesfully uploaded!</pre>";

}

}

# Insecure CAPTCHA

Ein CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) ist ein Test, der verwendet wird, um festzustellen, ob ein Benutzer ein Mensch oder ein Bot ist, um Spam und andere unerwünschte Aktiv

itäten zu verhindern. Es besteht aus verzerrten Buchstaben oder Zahlen in einem Bild, das von Menschen gelöst werden kann, aber nicht von den meisten Robotern. Die Idee entstand in den 1990er Jahren, als Hacker versuchten, Text für Computer unlesbar zu machen. Heutzutage werden CAPTCHAs verwendet, um Webseiten vor Spam zu schützen, und Dienste wie Google's reCAPTCHA und hCaptcha sind weit verbreitet.

Unsichere CAPTCHA-Implementierung:

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

// Überprüfen des Anti-CSRF-Tokens

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

// CAPTCHA-Prüfung von einer Drittanbieterquelle

$resp = recaptcha\_check\_answer(

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

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

);

// CAPTCHA falsch eingegeben?

if( !$resp ) {

// Fehlermeldung bei falscher Eingabe des CAPTCHA

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

$hide\_form = false;

}

else {

// Weitere Verarbeitung und Validierung des Passworts

// ...

}

}

Sichere CAPTCHA-Implementierung mit mehrstufiger Überprüfung:

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

// CAPTCHA-Prüfung von einer Drittanbieterquelle

$resp = recaptcha\_check\_answer(

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

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

);

// CAPTCHA falsch eingegeben?

if( !$resp ) {

// Fehlermeldung bei falscher Eingabe des CAPTCHA

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

$hide\_form = false;

return;

}

else {

// Weitere Verarbeitung und Validierung des Passworts

// ...

}

}

# SQL Injection

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

// Benutzereingabe abrufen

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

// SQL-Abfrage mit direkter Verwendung der Benutzereingabe

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

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

// Ergebnisse verarbeiten und ausgeben

// ...

}

Unsicherheit: Die Benutzereingabe $id wird direkt in die SQL-Abfrage eingefügt, ohne dass sie zuerst auf SQL-Injection-Angriffe überprüft wird. Dadurch wird die Anwendung anfällig für SQL-Injection-Angriffe, bei denen ein Angreifer bösartige SQL-Codefragmente in die Benutzereingabe einfügen und die Datenbank manipulieren kann.

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

// Überprüfen des Anti-CSRF-Tokens

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

// Benutzereingabe abrufen

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

// Sicherstellen, dass eine Zahl eingegeben wurde

if(is\_numeric( $id )) {

$id = intval ($id);

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

case MYSQL:

// Vorbereitete Anweisungen und Parameterbindung für MySQL

$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();

$row = $data->fetch();

// Nur ein Ergebnis sicherstellen

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

// Ergebnisse verarbeiten und ausgeben

// ...

}

break;

case SQLITE:

// Vorbereitete Anweisungen und Parameterbindung für SQLite

$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) {

// Nur ein Ergebnis sicherstellen

// Ergebnisse verarbeiten und ausgeben

// ...

}

break;

}

}

}

Sicherheit: Das verbesserte Snippet verwendet vorbereitete Anweisungen und Parameterbindung, um SQL-Injection-Angriffe zu verhindern. Die Benutzereingabe wird ordnungsgemäß validiert und als Parameter in der SQL-Abfrage verwendet, wodurch die Anwendung vor SQL-Injection-Angriffen geschützt wird.

# SQL Injection (Blind)

Unsichere direkte Verwendung von Benutzereingaben in der Abfrage:

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

// Benutzereingabe abrufen

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

$exists = false;

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

case MYSQL:

// SQL-Abfrage mit direkter Verwendung der Benutzereingabe

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

try {

$result = mysqli\_query($GLOBALS["\_\_\_mysqli\_ston"], $query );

} catch (Exception $e) {

print "There was an error.";

exit;

}

$exists = false;

if ($result !== false) {

try {

$exists = (mysqli\_num\_rows( $result ) > 0);

} catch(Exception $e) {

$exists = false;

}

}

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

break;

case SQLITE:

// SQL-Abfrage mit direkter Verwendung der Benutzereingabe

global $sqlite\_db\_connection;

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

try {

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

$row = $results->fetchArray();

$exists = $row !== false;

} catch(Exception $e) {

$exists = false;

}

break;

}

if ($exists) {

// Feedback für den Endbenutzer

echo '<pre>User ID exists in the database.</pre>';

} else {

// Benutzer wurde nicht gefunden

header( $\_SERVER[ 'SERVER\_PROTOCOL' ] . ' 404 Not Found' );

// Feedback für den Endbenutzer

echo '<pre>User ID is MISSING from the database.</pre>';

}

}

Unsicherheit: Die Benutzereingabe $id wird direkt in die SQL-Abfrage eingefügt, ohne dass sie zuerst auf SQL-Injection-Angriffe überprüft wird. Dadurch wird die Anwendung anfällig für SQL-Injection-Angriffe, bei denen ein Angreifer bösartige SQL-Codefragmente in die Benutzereingabe einfügen und die Datenbank manipulieren kann.

Verbessertes SQL-Injection (Blind):

Verwendung von vorbereiteten Anweisungen und Parameterbindung:

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

// Überprüfen des Anti-CSRF-Tokens

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

$exists = false;

// Benutzereingabe abrufen

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

// War eine Zahl eingegeben?

if(is\_numeric( $id )) {

$id = intval ($id);

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

case MYSQL:

// Vorbereitete Anweisungen und Parameterbindung für MySQL

$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();

$exists = $data->rowCount();

break;

case SQLITE:

// Vorbereitete Anweisungen und Parameterbindung für SQLite

global $sqlite\_db\_connection;

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

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

$result = $stmt->execute();

$result->finalize();

if ($result !== false) {

// Es gibt keine Möglichkeit, die Anzahl der zurückgegebenen Zeilen abzurufen

// Dies überprüft die Anzahl der Spalten (nicht Zeilen), aber es wird nicht verhindern,

// dass jemand mehrere Zeilen abruft und sie nacheinander anzeigt.

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

if ($num\_columns == 1) {

$row = $result->fetchArray();

$numrows = $row[ 'numrows' ];

$exists = ($numrows == 1);

}

}

break;

}

}

// Ergebnisse abrufen

if ($exists) {

// Feedback für den Endbenutzer

echo '<pre>User ID exists in the database.</pre>';

} else {

# Weak Session IDs

Schwaches Session-ID-Snippet:

$html = "";

if ($\_SERVER['REQUEST\_METHOD'] == "POST") {

if (!isset ($\_SESSION['last\_session\_id'])) {

$\_SESSION['last\_session\_id'] = 0;

}

$\_SESSION['last\_session\_id']++;

$cookie\_value = $\_SESSION['last\_session\_id'];

setcookie("dvwaSession", $cookie\_value);

}

Probleme:

Die Session-ID wird einfach inkrementiert, was vorhersehbar ist und angreifbar macht.

Die Session-ID wird nicht ausreichend geschützt oder verschleiert.

Es wird keine Überprüfung durchgeführt, um zu überprüfen, ob die Session-ID bereits verwendet wurde oder ob sie korrekt ist.

Die Verwendung von setcookie ohne eine geeignete Lebensdauer- oder Pfadangabe kann die Sicherheit der Session-ID weiter beeinträchtigen.

Impossible Session-ID:

$html = "";

if ($\_SERVER['REQUEST\_METHOD'] == "POST") {

$cookie\_value = sha1(mt\_rand() . time() . "Impossible");

setcookie("dvwaSession", $cookie\_value, time()+3600, "/vulnerabilities/weak\_id/", $\_SERVER['HTTP\_HOST'], true, true);

}

Probleme:

Die Session-ID wird mit sha1(mt\_rand() . time() . "Impossible") generiert, was nicht ausreichend sicher ist. Die Verwendung von mt\_rand() ist nicht kryptographisch sicher und sha1 ist anfällig für Kollisionen.

Die Verwendung einer festen Zeichenfolge ("Impossible") in der Generierung der Session-ID trägt nicht zur Sicherheit bei.

Es wird keine Überprüfung durchgeführt, um zu überprüfen, ob die Session-ID bereits verwendet wurde oder ob sie korrekt ist.

Die Verwendung von setcookie ohne eine geeignete Lebensdauer- oder Pfadangabe kann die Sicherheit der Session-ID weiter beeinträchtigen.

# XSS (DOM)

XSS (DOM) - Snippet 1:

# Keine Schutzmaßnahmen, alles ist erlaubt

Analyse:

Dieses Snippet enthält keinerlei Schutzmaßnahmen gegen XSS-Angriffe. Es stellt sicherlich eine Schwachstelle dar, da jeglicher Benutzereingabe direkt vertraut wird, ohne dass sie gefiltert oder validiert wird.

Es ermöglicht Angreifern, bösartigen JavaScript-Code in die Seite einzufügen, der dann vom Browser des Benutzers ausgeführt wird.

Diese Art von Schwachstelle kann verwendet werden, um Session-Cookies zu stehlen, Benutzersitzungen zu übernehmen oder bösartige Aktionen im Namen des Benutzers auszuführen.

XSS (DOM) - Snippet 2:

# Es ist nicht notwendig, etwas zu tun, da der Schutz auf der Clientseite behandelt wird

Analyse:

Dieses Snippet legt nahe, dass der XSS-Schutz auf der Clientseite behandelt wird. Das bedeutet, dass JavaScript-Code auf der Benutzerseite verwendet wird, um sicherzustellen, dass Benutzereingaben ordnungsgemäß behandelt und gefiltert werden, bevor sie in das DOM eingefügt werden.

Clientseitige XSS-Schutzmaßnahmen können beispielsweise die Verwendung von Frameworks wie React.js oder Angular.js umfassen, die eingebaute XSS-Schutzfunktionen bieten.

Es ist jedoch wichtig zu beachten, dass clientseitige Schutzmaßnahmen allein nicht ausreichen, um XSS-Angriffe zu verhindern. Eine serverseitige Validierung und Filterung von Benutzereingaben ist ebenfalls erforderlich, um eine umfassende Sicherheitsstrategie gegen XSS-Angriffe zu gewährleisten.

Empfehlungen:

Implementieren Sie sowohl auf der Client- als auch auf der Serverseite Sicherheitsmaßnahmen, um XSS-Angriffe zu verhindern.

Führen Sie eine strikte Validierung und Filterung von Benutzereingaben durch, bevor sie im DOM angezeigt werden.

Verwenden Sie Content Security Policy (CSP), um zu steuern, welche Ressourcen auf einer Seite geladen werden dürfen und von wo aus sie geladen werden dürfen.

Schulen Sie Entwickler in sicheren Codierungspraktiken und Awareness im Umgang mit XSS-Risiken.

# XSS (Reflected)

**XSS (Reflected) - Impossible:**

<?php

// Gibt es eine Eingabe?

if( array\_key\_exists( "name", $\_GET ) && $\_GET[ 'name' ] != NULL ) {

// Anti-CSRF-Token überprüfen

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

// Eingabe abrufen und HTML-Sonderzeichen maskieren

$name = htmlspecialchars( $\_GET[ 'name' ] );

// Rückmeldung für den Benutzer

echo "<pre>Hello {$name}</pre>";

}

// Anti-CSRF-Token generieren

generateSessionToken();

?>

Analyse:

Das Snippet validiert die Eingabe des Benutzernamens mithilfe der htmlspecialchars-Funktion, um potenziell gefährliche Zeichen in HTML-Entities zu maskieren.

Es überprüft auch das Anti-CSRF-Token, um sicherzustellen, dass das Formular nur von autorisierten Benutzern gesendet wird.

Durch die Verwendung von htmlspecialchars wird die Eingabe vor XSS-Angriffen geschützt, da potenziell gefährlicher Code als Text behandelt wird und nicht als ausführbarer HTML-Code interpretiert wird.

**XSS (Reflected) - Low:**

<?php

// Gibt es eine Eingabe?

if( array\_key\_exists( "name", $\_GET ) && $\_GET[ 'name' ] != NULL ) {

// Anti-CSRF-Token überprüfen

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

// Eingabe abrufen und HTML-Sonderzeichen maskieren

$name = htmlspecialchars( $\_GET[ 'name' ] );

// Rückmeldung für den Benutzer

echo "<pre>Hello {$name}</pre>";

}

// Anti-CSRF-Token generieren

generateSessionToken();

?>

Analyse:

Dieses Snippet sendet den HTTP-Header "X-XSS-Protection: 0", der die XSS-Schutzfunktionen in modernen Webbrowsern deaktiviert.

Es gibt eine Eingabe direkt aus der GET-Anfrage aus, ohne sie zu validieren oder zu filtern, was eine potenzielle Sicherheitslücke für Reflected XSS-Angriffe darstellt.

Ohne XSS-Schutzmechanismen könnten Angreifer bösartigen JavaScript-Code über die URL einschleusen, der dann vom Browser des Opfers ausgeführt wird.

# XSS (Stored)

Das bereitgestellte Code-Snippet ist ein PHP-Skript, das Daten aus einem HTML-Formular entgegennimmt, diese filtert und dann in einer Datenbank speichert. Hier ist eine Analyse des Codes:

Eingabenvalidierung und Filterung:

Die trim()-Funktion wird verwendet, um führende und nachfolgende Leerzeichen aus den Eingabestrings zu entfernen.

stripslashes() wird verwendet, um Escape-Zeichen, insbesondere Slashes, aus den Eingaben zu entfernen.

mysqli\_real\_escape\_string() wird verwendet, um Zeichen zu escapen, die spezielle Bedeutung in SQL-Anweisungen haben könnten, und so vor SQL-Injection-Angriffen zu schützen.

htmlspecialchars() wird verwendet, um HTML-Sonderzeichen in ihre entsprechenden HTML-Entities umzuwandeln. Dies ist entscheidend, um XSS-Angriffe zu verhindern, indem verhindert wird, dass HTML- und JavaScript-Code in die Seite eingefügt und vom Browser des Benutzers ausgeführt wird.

Datenbankaktualisierung:

Das vorbereitete SQL-Statement wird verwendet, um sicherzustellen, dass die Benutzereingaben sicher in die Datenbank eingefügt werden, ohne dass dies zu SQL-Injection-Angriffen führt.

Die Benutzereingaben werden als Parameter an das vorbereitete Statement gebunden, um sicherzustellen, dass sie sicher in die Datenbank eingefügt werden, ohne dass SQL-Injection-Angriffe möglich sind.

Anti-CSRF-Token:

Ein Anti-CSRF-Token wird generiert und überprüft, um sicherzustellen, dass das Formular nur von autorisierten Benutzern gesendet wird. Dies hilft, Cross-Site Request Forgery (CSRF)-Angriffe zu verhindern.

Zusammenfassend ist das bereitgestellte Snippet ein gutes Beispiel für sichere Praktiken im Umgang mit Benutzereingaben und Datenbankinteraktionen in PHP-Anwendungen. Es implementiert Filterung, Escape-Mechanismen und CSRF-Schutz, um die Sicherheit der Anwendung zu verbessern und Angriffe wie SQL-Injection und XSS zu verhindern.

# CSP Bypass

**CSP Bypass - Impossible:**

<?php

$headerCSP = "Content-Security-Policy: script-src 'self';";

header($headerCSP);

?>

<?php

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

$page[ 'body' ] .= "

" . $\_POST['include'] . "

";

}

$page[ 'body' ] .= '

<form name="csp" method="POST">

<p>Unlike the high level, this does a JSONP call but does not use a callback, instead it hardcodes the function to call.</p><p>The CSP settings only allow external JavaScript on the local server and no inline code.</p>

<p>1+2+3+4+5=<span id="answer"></span></p>

<input type="button" id="solve" value="Solve the sum" />

</form>

<script src="source/impossible.js"></script>

';

?>

**Text für CSP Bypass - Impossible:**

Analyse:

Dieses Snippet setzt die CSP-Richtlinie auf 'self', was bedeutet, dass nur Skripte von derselben Domain geladen werden dürfen.

Es gibt jedoch ein Formular, das es Benutzern ermöglicht, einen beliebigen Wert als URL einzugeben und ihn dann ohne Filterung oder Validierung in die Seite einzufügen. Dadurch können Benutzer JavaScript-Code von externen Quellen einschließen, was einen CSP-Bypass ermöglicht.

**CSP Bypass - Low:**

<?php

$headerCSP = "Content-Security-Policy: script-src 'self' https://pastebin.com hastebin.com www.toptal.com example.com code.jquery.com https://ssl.google-analytics.com https://digi.ninja ;"; // allows js from self, pastebin.com, hastebin.com, jquery, digi.ninja, and google analytics.

header($headerCSP);

?>

<?php

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

$page[ 'body' ] .= "

<script src='" . $\_POST['include'] . "'></script>

";

}

$page[ 'body' ] .= '

<form name="csp" method="POST">

<p>You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:</p>

<input size="50" type="text" name="include" value="" id="include" />

<input type="submit" value="Include" />

</form>

<p>

As Pastebin and Hastebin have stopped working, here are some scripts that may, or may not help.

</p>

<ul>

<li>https://digi.ninja/dvwa/alert.js</li>

<li>https://digi.ninja/dvwa/alert.txt</li>

<li>https://digi.ninja/dvwa/cookie.js</li>

<li>https://digi.ninja/dvwa/forced\_download.js</li>

<li>https://digi.ninja/dvwa/wrong\_content\_type.js</li>

</ul>

<p>

Pretend these are on a server like Pastebin and Hastebin and try to work out why some work and some do not work. Check the help for an explanation if you get stuck.

</p>

';

?>

**Text für CSP Bypass - Low:**

Analyse:

Dieses Snippet erlaubt Skripte von mehreren vertrauenswürdigen Domänen, einschließlich pastebin.com und hastebin.com.

Es gibt ein Formular, das es Benutzern ermöglicht, eine URL einzugeben, von der sie Skripte einfügen möchten. Das eingefügte Skript wird dann als <script>-Tag auf der Seite geladen.

Wenn pastebin.com und hastebin.com nicht funktionieren, werden einige alternative Skript-URLs zur Verfügung gestellt, die von digi.ninja stammen.

# Javascript

<?php

$page[ 'body' ] .= '<script src="' . DVWA\_WEB\_PAGE\_TO\_ROOT . 'vulnerabilities/javascript/source/high.js"></script>';

?>

**Impossible:**

<?php

$page[ 'body' ] .= '<script src="' . DVWA\_WEB\_PAGE\_TO\_ROOT . 'vulnerabilities/javascript/source/high.js"></script>';

?>

Das erste Code-Snippet fügt ein JavaScript-Skript mit dem Namen "high.js" hinzu, das sich in einem bestimmten Verzeichnis auf der Webseite befindet. Dies ist eine gängige Methode, um externe JavaScript-Dateien in eine Webseite einzubinden, und kann sicher sein, wenn die eingebundene Datei vertrauenswürdig ist.

**Low**

<?php

$page[ 'body' ] .= <<<EOF

<script>

/\*

MD5 code from here

https://github.com/blueimp/JavaScript-MD5

\*/

!function(n){"use strict";function t(n,t){var r=(65535&n)+(65535&t);return(n>>16)+(t>>16)+(r>>16)<<16|65535&r}function r(n,t){return n<<t|n>>>32-t}function e(n,e,o,u,c,f){return t(r(t(t(e,n),t(u,f)),c),o)}function o(n,t,r,o,u,c,f){return e(t&r|~t&o,n,t,u,c,f)}function u(n,t,r,o,u,c,f){return e(t&o|r&~o,n,t,u,c,f)}function c(n,t,r,o,u,c,f){return e(t^r^o,n,t,u,c,f)}function f(n,t,r,o,u,c,f){return e(r^(t|~o),n,t,u,c,f)}function i(n,r){n[r>>5]|=128<<r%32,n[14+(r+64>>>9<<4)]=r;var e,i,a,d,h,l=1732584193,g=-271733879,v=-1732584194,m=271733878;for(e=0;e<n.length;e+=16)i=l,a=g,d=v,h=m,g=f(g=f(g=f(g=f(g=c(g=c(g=c(g=c(g=u(g=u(g=u(g=u(g=o(g=o(g=o(g=o(g,v=o(v,m=o(m,l=o(l,g,v,m,n[e],7,-680876936),g,v,n[e+1],12,-389564586),l,g,n[e+2],17,606105819),m,l,n[e+3],22,-1044525330),v=o(v,m=o(m,l=o(l,g,v,m,n[e+4],7,-176418897),g,v,n[e+5],12,1200080426),l,g,n[e+6],17,-1473231341),m,l,n[e+7],22,-45705983),v=o(v,m=o(m,l=o(l,g,v,m,n[e+8],7,1770035416),g,v,n[e+9],12,-1958414417),l,g,n[e+10],17,-42063),m,l,n[e+11],22,-1990404162),v=o(v,m=o(m,l=o(l,g,v,m,n[e+12],7,1804603682),g,v,n[e+13],12,-40341101),l,g,n[e+14],17,-1502002290),m,l,n[e+15],22,1236535329),v=u(v,m=u(m,l=u(l,g,v,m,n[e+1],5,-165796510),g,v,n[e+6],9,-1069501632),l,g,n[e+11],14,643717713),m,l,n[e],20,-373897302),v=u(v,m=u(m,l=u(l,g,v,m,n[e+5],5,-701558691),g,v,n[e+10],9,38016083),l,g,n[e+15],14,-660478335),m,l,n[e+4],20,-405537848),v=u(v,m=u(m,l=u(l,g,v,m,n[e+9],5,568446438),g,v,n[e+14],9,-1019803690),l,g,n[e+3],14,-187363961),m,l,n[e+8],20,1163531501),v=u(v,m=u(m,l=u(l,g,v,m,n[e+13],5,-1444681467),g,v,n[e+2],9,-51403784),l,g,n[e+7],14,1735328473),m,l,n[e+12],20,-1926607734),v=c(v,m=c(m,l=c(l,g,v,m,n[e+5],4,-378558),g,v,n[e+8],11,-2022574463),l,g,n[e+11],16,1839030562),m,l,n[e+14],23,-35309556),v=c(v,m=c(m,l=c(l,g,v,m,n[e+1],4,-1530992060),g,v,n[e+4],11,1272893353),l,g,n[e+7],16,-155497632),m,l,n[e+10],23,-1094730640),v=c(v,m=c(m,l=c(l,g,v,m,n[e+13],4,681279174),g,v,n[e],11,-358537222),l,g,n[e+3],16,-722521979),m,l,n[e+6],23,76029189),v=c(v,m=c(m,l=c(l,g,v,m,n[e+9],4,-640364487),g,v,n[e+12],11,-421815835),l,g,n[e+15],16,530742520),m,l,n[e+2],23,-995338651),v=f(v,m=f(m,l=f(l,g,v,m,n[e],6,-198630844),g,v,n[e+7],10,1126891415),l,g,n[e+14],15,-1416354905),m,l,n[e+5],21,-57434055),v=f(v,m=f(m,l=f(l,g,v,m,n[e+12],6,1700485571),g,v,n[e+3],10,-1894986606),l,g

# Authorisation Bypass

<?php

/\*

Only the admin user is allowed to access this page

\*/

if (dvwaCurrentUser() != "admin") {

print "Unauthorised";

http\_response\_code(403);

exit;

}

?>

Das erste Code-Snippet überprüft, ob der aktuelle Benutzer als "admin" authentifiziert ist. Falls nicht, wird die Meldung "Unauthorised" ausgegeben und der HTTP-Statuscode 403 (Forbidden) gesendet, um den Zugriff zu verweigern.

**Low:**

<?php

/\*

Nothing to see here for this vulnerability, have a look

instead at the dvwaHtmlEcho function in:

\* dvwa/includes/dvwaPage.inc.php

\*/

?>

Das zweite Code-Snippet erklärt, dass für diese spezifische Schwachstelle nichts angezeigt wird. Es wird stattdessen empfohlen, die Funktion dvwaHtmlEcho in der Datei dvwa/includes/dvwaPage.inc.php zu überprüfen. Dies legt nahe, dass die Schwachstelle möglicherweise in der Art und Weise besteht, wie Daten auf der Seite ausgegeben werden, und nicht in der Autorisierung selbst. Entwickler sollten diese Funktion überprüfen, um potenzielle Sicherheitslücken zu identifizieren.

# Open HTTP Redirect

**Impossible:**

<?php

$target = "";

if (array\_key\_exists ("redirect", $\_GET) && is\_numeric($\_GET['redirect'])) {

switch (intval ($\_GET['redirect'])) {

case 1:

$target = "info.php?id=1";

break;

case 2:

$target = "info.php?id=2";

break;

case 99:

$target = "https://digi.ninja";

break;

}

if ($target != "") {

header ("location: " . $target);

exit;

} else {

?>

Unknown redirect target.

<?php

exit;

}

}

?>

Missing redirect target.

In diesem Code-Snippet wird der Parameter "redirect" überprüft, und je nach Wert wird der Benutzer auf eine bestimmte URL weitergeleitet. Das Problem dabei ist, dass die Überprüfung des Parameters nicht ausreichend ist, da der Benutzer immer noch eine beliebige numerische ID eingeben kann, um auf eine andere Seite weitergeleitet zu werden. Dies ermöglicht potenziell das sogenannte "Open Redirect"-Angriffsszenario, bei dem ein Angreifer den Benutzer auf eine bösartige Website umleitet.

**Low:**

<?php

if (array\_key\_exists ("redirect", $\_GET) && $\_GET['redirect'] != "") {

header ("location: " . $\_GET['redirect']);

exit;

}

http\_response\_code (500);

?>

<p>Missing redirect target.</p>

<?php

exit;

?>

In diesem Code-Snippet wird ebenfalls der Parameter "redirect" überprüft, und wenn er vorhanden ist, wird der Benutzer auf die darin enthaltene URL weitergeleitet. Das Problem hier ist, dass keine Validierung des Parameters stattfindet, was es einem Angreifer ermöglicht, die Weiterleitungs-URL zu manipulieren und den Benutzer auf eine bösartige Website umzuleiten. Dies ist ebenfalls anfällig für Open-Redirect-Angriffe.