 Protokoll

|  |  |  |
| --- | --- | --- |
| Thema: | **DVWA** | Klasse: 4aWI |
| Datum: |  | Lehrer: Schwärzler |
| Name: | Fatih Capar | Fach: SESD |

Inhaltsangabe:

Inhalt

[Brute Force 1](#_Toc159222533)

[Command Injection 2](#_Toc159222534)

[CSRF 3](#_Toc159222535)

[File Inclusion 3](#_Toc159222536)

[File Upload 4](#_Toc159222537)

[Insecure CAPTCHA 5](#_Toc159222538)

[SQL Injection 5](#_Toc159222539)

[SQL Injection (Blind) 5](#_Toc159222540)

[Weak Session IDs 5](#_Toc159222541)

[XSS (DOM) 5](#_Toc159222542)

[XSS (Reflected) 5](#_Toc159222543)

[XSS (Stored) 5](#_Toc159222544)

[CSP Bypass 5](#_Toc159222545)

[JavaScript 5](#_Toc159222546)

[Authorisation Bypass 5](#_Toc159222547)

[Open HTTP Redirect 5](#_Toc159222548)

# Brute Force

Theorie:  
Brute-Force-Angriffe bedeuten, dass ein Angreifer versucht, Zugang zu einem System zu erlangen, indem er wiederholt verschiedene Kombinationen von Benutzernamen und Passwörtern ausprobiert.

Beispiel 1:

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

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Brute Force PHP Snippet:

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.

Beispiel 2:

Bei der Authentifizierung könnte ein Angreifer versuchen, sich Zugang zu einem Konto zu verschaffen, indem er Listen gängiger Benutzernamen und Passwörter durchprobiert. Wenn es keine strengen Passwortregeln gibt, könnte dies erfolgreich sein. Sicherheits-Tools wie der Php-Brute-Force-Attack Detector helfen dabei, solche Angriffe zu erkennen.

**Verbessertes PHP Snippet:**

**POST-Parameter für Benutzeranmeldeinformationen:**

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

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

Sicherheit: Die Verwendung von POST-Parametern ist sicherer als GET, da POST-Daten nicht in der URL sichtbar sind.

**Verwendung von PDO und vorbereiteten Statements:**

Sicherheit: PDO und vorbereitete Statements schützen vor SQL-Injektionen, indem sie sicherstellen, dass Eingaben als Daten und nicht als Teil der Abfrage interpretiert werden.

**Implementierung von Sicherheitsmaßnahmen:**

Überprüfung von Anti-CSRF-Token und Begrenzung fehlgeschlagener Anmeldeversuche: Das Snippet enthält Funktionen wie checkToken() und die Überwachung fehlgeschlagener Anmeldeversuche, um die Anwendung vor Brute-Force-Angriffen zu schützen.

**Detaillierte Rückmeldung an den Benutzer:**

Das Snippet gibt detaillierte Informationen über den Kontostatus und mögliche Sicherheitsbedrohungen zurück, was dem Benutzer hilft, sein Verhalten anzupassen und Sicherheitsmaßnahmen zu beachten.

Sicherheits-Tipp:

Starken Passwörtern kommt eine enorme Bedeutung für die Computersicherheit zu. Oft wählen Nutzer jedoch aus Bequemlichkeit einfache Passwörter. Password-Cracker sind Tools, die entwickelt wurden, um diese schwachen Passwörter zu knacken. Systemadministratoren sollten die Benutzer über die Wichtigkeit starker Passwörter aufklären und deren Verwendung sicherstellen. Regelmäßige Passwort-Updates und die Verwendung von Passwort-Prompten sind wichtige Maßnahmen.

# Command Injection

Command Injection ist ein Angriff, bei dem das Ziel die Ausführung beliebiger Befehle auf dem Host-Betriebssystem über eine anfällige Anwendung ist. Dies geschieht oft, 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:

1. 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.
2. 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.
3. Ein privilegiertes Programm führt einen Initialisierungsskript ohne ausreichende Überprüfung von Umgebungsvariablen aus, was es einem Angreifer ermöglicht, schädlichen Code auszuführen.
4. 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.
5. 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.

Beispiel1:

**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.

Beispiel2:

**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.

# CSRF

Cross-Site Request Forgery (CSRF) ist ein Angriff, der es einem Angreifer ermöglicht, einen authentifizierten Benutzer dazu zu bringen, unerwünschte Aktionen auf einer Webanwendung auszuführen. Dies kann beispielsweise durch Social Engineering erfolgen, indem der Angreifer einen Link per E-Mail oder Chat an die Benutzer sendet und sie dazu bringt, Aktionen auszuführen, die vom Angreifer festgelegt wurden.

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.

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

Es gibt 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 Snippet:**

**Fehlen eines CSRF-Schutzes:**

phpCopy code

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

// ...

}

Unsicherheit: Das Snippet 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.

Beispiel2:

Verbessertes PHP Snippet:

**Überprüfung des Anti-CSRF-Tokens:**

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

Sicherheit: Das Snippet 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.

Arten des Einschlusses:

* Remote-Dateieinschluss (RFI): Der Webanwendung lädt und führt eine Remote-Datei aus, normalerweise in Form eines HTTP- oder FTP-URIs.
* Lokaler Dateieinschluss (LFI): Ähnlich wie bei RFI, jedoch werden nur lokale Dateien auf dem aktuellen Server zur Ausführung eingebunden.

Programmiersprachen:

* PHP: Hauptursache sind unvalidierte Benutzereingaben in Kombination mit Dateisystemfunktionen wie include oder require.
* JavaServer Pages (JSP): Eine Skriptsprache, die Dateien zur Laufzeit einbinden kann.
* Server Side Includes (SSI): Selten, aber auch für Remote-Code-Ausführung auf einem verwundbaren Webserver verwendet.

Beispiele:

* Ein PHP-Skript bindet Dateien basierend auf Benutzereingaben ein, was zu RFI/LFI führen kann.
* Ein JSP-Skript ist anfällig für Dateieinschlussschwachstellen, die durch Null-Byte-Injektion verursacht werden können.
* Ein SSI-Tag in HTML kann zur Remote-Dateieinschließung verwendet werden, was zu Remote-Code-Ausführung führt.
* Zur Vermeidung von Dateieinschluss-Schwachstellen sollten Entwickler Whitelists verwenden, Eingaben filtern und Switch/Case-Anweisungen bevorzugen.

Beispiel1:

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.

Beispiel2:

**Verbessertes PHP Snippet:**

**Einschränkung der erlaubten Dateien:**

// Die Seite, die wir anzeigen möchten

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

// Nur include.php oder file{1..3}.php erlauben

if ($file != "include.php" && $file != "file1.php" && $file != "file2.php" && $file != "file3.php") {

// Dies ist nicht die gewünschte Seite!

echo "FEHLER: Datei nicht gefunden!";

exit;

}

Sicherheit: Das Snippet prüft, ob die $file-Variable auf eine der erlaubten Dateien (include.php, file1.php, file2.php, file3.php) verweist. Wenn nicht, wird eine Fehlermeldung ausgegeben und das Skript beendet. Dadurch wird verhindert, dass der Benutzer auf nicht autorisierte Dateien zugreifen kann.

# 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.

Hier sind einige gängige Fehler beim Sichern von Upload-Formularen und wie Angreifer sie umgehen können:

* Keine Validierung: Ohne Validierung können Angreifer bösartigen Code hochladen, der auf dem Server ausgeführt wird.
* MIME-Typ-Validierung: Die Überprüfung des MIME-Typs allein reicht nicht aus, da Angreifer den MIME-Typ manipulieren können.
* Schwarze Liste von Dateierweiterungen: Das Blockieren bestimmter Dateierweiterungen ist unzuverlässig, da Angreifer einfach Erweiterungen ändern können.
* Doppelte Erweiterungen: Durch das Ausnutzen von doppelten Erweiterungen können Angreifer Dateiüberprüfungen umgehen.
* Überprüfung des Bildheaders: Die Überprüfung des Bildheaders ist nicht ausreichend, da Angreifer bösartigen Code in den Kommentarbereich von Bildern einbetten können.
* Absicherung des Upload-Ordners mit .htaccess: Eine .htaccess-Datei kann den Zugriff auf bestimmte Dateitypen einschränken, aber sie kann überschrieben werden.
* Clientseitige Validierung: Die Validierung auf der Clientseite ist nicht ausreichend, da sie leicht umgangen werden 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>";

}

}

Unsicherheit: Das Snippet führt einen einfachen Dateiupload durch, ohne ausreichende Validierung der Datei oder des Dateipfads. Dadurch wird die Anwendung anfällig für verschiedene Arten von Angriffen wie Dateiüberschreibung, Dateitransversal und Ausführung bösartiger Dateien.

**Verbesserter Dateiupload mit Validierung:**

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

// Anti-CSRF-Token überprüfen

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

// Dateiinformationen abrufen

$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' ];

// Weitere Validierungen für Dateityp, Größe und Inhaltsprüfung durchführen

// ...

// Datei in temporäres Verzeichnis verschieben und Verarbeitung durchführen

// ...

// Erfolgsmeldung oder Fehlermeldung ausgeben

// ...

// Anti-CSRF-Token generieren

generateSessionToken();

}

Sicherheit: Das Snippet führt zusätzliche Validierungen durch, um sicherzustellen, dass nur bestimmte Dateitypen, Größen und Inhalte akzeptiert werden. Es verschiebt die Datei zunächst in ein temporäres Verzeichnis, führt eine Verarbeitung durch und verschiebt sie dann in das endgültige Zielverzeichnis. Darüber hinaus wird ein Anti-CSRF-Token verwendet, um vor CSRF-Angriffen zu schützen.

Fachbegriffe:

- MIME-Typ: Ein MIME-Typ (Multipurpose Internet Mail Extensions) ist eine Kennzeichnung, die angibt, welche Art von Daten in einer Datei enthalten ist (z.B. Text, Bild, Audio usw.).

- Whitelist: Eine Liste von erlaubten Elementen, die akzeptiert oder zugelassen werden.

# 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 Aktivitäten 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

// ...

}

}

Unsicherheit: Das Snippet verwendet ein CAPTCHA von einer Drittanbieterquelle (z. B. reCAPTCHA) zur Überprüfung der Benutzereingaben. Allerdings wird die CAPTCHA-Antwort nicht ordnungsgemäß überprüft, da das Skript den Benutzer immer noch zum nächsten Schritt weiterleitet, selbst wenn das CAPTCHA falsch eingegeben wurde.

**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

// ...

}

}

Sicherheit: Das verbesserte Snippet führt eine mehrstufige Überprüfung des CAPTCHA durch. Zuerst wird überprüft, ob das CAPTCHA korrekt eingegeben wurde. Nur wenn das CAPTCHA korrekt ist, wird der Benutzer zur nächsten Stufe weitergeleitet, um die Passwortänderung abzuschließen. Dadurch wird sichergestellt, dass der Benutzer erst dann fortgefahren werden kann, wenn das CAPTCHA korrekt eingegeben wurde.

# SQL Injection

**Unsichere direkte Verwendung von Benutzereingaben in der Abfrage:**

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.

**Verwendung von vorbereiteten Anweisungen und Parameterbindung:**

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 {

// 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>';

}

}

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.

# Weak Session IDs

Schwaches Session-ID-Snippet 1:

$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)

Die beiden Code-Snippets behandeln das Thema Cross-Site Scripting (XSS) mit dem Fokus auf dem Document Object Model (DOM):

### XSS (DOM) - Snippet 1:

phpCopy code

# 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:

phpCopy code

# 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)

**XSS (Stored) - Low:**

<?php

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

// Eingaben abrufen

$message = trim( $\_POST[ 'mtxMessage' ] );

$name = trim( $\_POST[ 'txtName' ] );

// Nachrichteneingabe filtern

$message = stripslashes( $message );

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

// Nameingabe filtern

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

// Datenbank aktualisieren

$query = "INSERT INTO guestbook ( comment, name ) VALUES ( '$message', '$name' );";

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

//mysql\_close();

}

?>

Analyse:

* Dieses Snippet nimmt Benutzereingaben aus einem Formular entgegen und fügt sie ungeschützt in eine SQL-Abfrage ein, um sie in die Datenbank einzufügen.
* Die Eingaben werden zwar mit stripslashes und mysqli\_real\_escape\_string bearbeitet, um SQL-Injektionen zu verhindern, aber es wird keine HTML-Filterung durchgeführt, was die Anfälligkeit für XSS-Angriffe erhöht.
* Potenziell schädlicher JavaScript-Code, der in die Eingabefelder eingegeben wird, könnte in der Datenbank gespeichert und später von anderen Benutzern angezeigt werden, was zu einem Stored XSS-Angriff führen könnte.

**XSS (Stored) - Impossible:**

<?php

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

// Anti-CSRF-Token überprüfen

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

// Eingaben abrufen

$message = trim( $\_POST[ 'mtxMessage' ] );

$name = trim( $\_POST[ 'txtName' ] );

// Nachrichteneingabe filtern

$message = stripslashes( $message );

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

$message = htmlspecialchars( $message );

// Nameingabe filtern

$name = stripslashes( $name );

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

$name = htmlspecialchars( $name );

// Datenbank aktualisieren

$data = $db->prepare( 'INSERT INTO guestbook ( comment, name ) VALUES ( :message, :name );' );

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

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

$data->execute();

}

// Anti-CSRF-Token generieren

generateSessionToken();

?>

Analyse:

* Dieses Snippet führt ähnliche Operationen wie das vorherige durch, jedoch mit zusätzlicher HTML-Filterung für die Benutzereingaben.
* Die htmlspecialchars-Funktion wird verwendet, um sicherzustellen, dass HTML-Tags in den Eingabefeldern korrekt als Text behandelt werden und nicht als ausführbarer Code interpretiert werden.
* Dadurch wird das Risiko von XSS-Angriffen verringert, da potenziell gefährlicher JavaScript-Code als Text angezeigt wird und nicht ausgeführt wird.
* Außerdem wird ein Anti-CSRF-Token verwendet, um sicherzustellen, dass das Formular nur von autorisierten Benutzern gesendet wird, was zusätzlichen Schutz bietet.

# 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>

';

**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>

';

**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

**Impossible:**

<?php

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

?>

**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,n[e+10],15,-1051523),m,l,n[e+1],21,-2054922799),v=f(v,m=f(m,l=f(l,g,v,m,n[e+8],6,1873313359),g,v,n[e+15],10,-30611744),l,g,n[e+6],15,-1560198380),m,l,n[e+13],21,1309151649),v=f(v,m=f(m,l=f(l,g,v,m,n[e+4],6,-145523070),g,v,n[e+11],10,-1120210379),l,g,n[e+2],15,718787259),m,l,n[e+9],21,-343485551),l=t(l,i),g=t(g,a),v=t(v,d),m=t(m,h);return[l,g,v,m]}function a(n){var t,r="",e=32\*n.length;for(t=0;t<e;t+=8)r+=String.fromCharCode(n[t>>5]>>>t%32&255);return r}function d(n){var t,r=[];for(r[(n.length>>2)-1]=void 0,t=0;t<r.length;t+=1)r[t]=0;var e=8\*n.length;for(t=0;t<e;t+=8)r[t>>5]|=(255&n.charCodeAt(t/8))<<t%32;return r}function h(n){return a(i(d(n),8\*n.length))}function l(n,t){var r,e,o=d(n),u=[],c=[];for(u[15]=c[15]=void 0,o.length>16&&(o=i(o,8\*n.length)),r=0;r<16;r+=1)u[r]=909522486^o[r],c[r]=1549556828^o[r];return e=i(u.concat(d(t)),512+8\*t.length),a(i(c.concat(e),640))}function g(n){var t,r,e="";for(r=0;r<n.length;r+=1)t=n.charCodeAt(r),e+="0123456789abcdef".charAt(t>>>4&15)+"0123456789abcdef".charAt(15&t);return e}function v(n){return unescape(encodeURIComponent(n))}function m(n){return h(v(n))}function p(n){return g(m(n))}function s(n,t){return l(v(n),v(t))}function C(n,t){return g(s(n,t))}function A(n,t,r){return t?r?s(t,n):C(t,n):r?m(n):p(n)}"function"==typeof define&&define.amd?define(function(){return A}):"object"==typeof module&&module.exports?module.exports=A:n.md5=A}(this);

function rot13(inp) {

return inp.replace(/[a-zA-Z]/g,function(c){return String.fromCharCode((c<="Z"?90:122)>=(c=c.charCodeAt(0)+13)?c:c-26);});

}

function generate\_token() {

var phrase = document.getElementById("phrase").value;

document.getElementById("token").value = md5(rot13(phrase));

}

generate\_token();

</script>

EOF;

?>

Das erste Code-Snippet fügt ein JavaScript-Skript mit dem Namen "high.js" hinzu, das sich in einem bestimmten Verzeichnis auf der Webseite befindet.

Das zweite Code-Snippet fügt ein Inline-JavaScript-Skript ein, das einige Funktionen definiert:

rot13(inp): Eine Funktion, die den ROT13-Verschlüsselungsalgorithmus auf den Eingabetext anwendet.

generate\_token(): Eine Funktion, die aufgerufen wird, um einen Token zu generieren. Sie nimmt den Wert aus einem HTML-Eingabefeld mit der ID "phrase", wendet ROT13 darauf an und dann die MD5-Verschlüsselung und setzt das Ergebnis in ein anderes HTML-Eingabefeld mit der ID "token".

Das Skript ruft dann die generate\_token()-Funktion sofort auf, um den Token beim Laden der Seite zu generieren.

Es ist wichtig zu beachten, dass die Verwendung von Inline-JavaScript in Verbindung mit CSP (Content Security Policy) Risiken birgt und vermieden werden sollte, wenn möglich. Stattdessen sollte externes JavaScript geladen werden, das den CSP-Richtlinien entspricht.

# Authorisation Bypass

**Impossible :**

<?php

/\*

Only the admin user is allowed to access this page

\*/

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

print "Unauthorised";

http\_response\_code(403);

exit;

}

?>

**Low :**

<?php

/\*

Nothing to see here for this vulnerability, have a look

instead at the dvwaHtmlEcho function in:

\* dvwa/includes/dvwaPage.inc.php

\*/

?>

Im ersten Code-Snippet wird überprüft, ob der aktuelle Benutzer "admin" ist. Wenn dies nicht der Fall ist, wird die Meldung "Unauthorised" ausgegeben und der HTTP-Statuscode 403 (Forbidden) zurückgegeben, um anzuzeigen, dass der Zugriff verweigert wurde.

Im zweiten Code-Snippet wird erklärt, dass es für diese spezifische Schwachstelle nichts zu sehen gibt. Stattdessen wird empfohlen, die Funktion dvwaHtmlEcho in der Datei dvwa/includes/dvwaPage.inc.php zu überprüfen. Dies deutet darauf hin, dass die Schwachstelle möglicherweise in der Art und Weise besteht, wie Daten auf der Seite ausgegeben werden, und nicht in der Autorisierung selbst. Es wird empfohlen, diese Funktion zu überprüfen, um nach potenziellen Sicherheitslücken zu suchen.

# 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.

**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 beiden Code-Snippets wird ein Parameter namens "redirect" überprüft. Wenn dieser Parameter vorhanden ist, leitet der Server den Benutzer auf die darin enthaltene URL weiter. Das Problem dabei ist, dass der Benutzer diese URL beliebig ändern kann, um auf eine andere Website weitergeleitet zu werden. Das bedeutet, dass ein Angreifer den Benutzer auf eine bösartige Website weiterleiten könnte, indem er einfach die URL ändert.

Diese Schwachstelle wird "Open Redirect" genannt, und sie ermöglicht es einem Angreifer, Benutzer auf bösartige Websites umzuleiten. Um sie zu beheben, muss der Server sicherstellen, dass der Weiterleitungsziel nur auf vordefinierte, vertrauenswürdige URLs beschränkt ist.