 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.

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.

# SQL Injection

# SQL Injection (Blind)

# Weak Session IDs

# XSS (DOM)

# XSS (Reflected)

# XSS (Stored)

# CSP Bypass

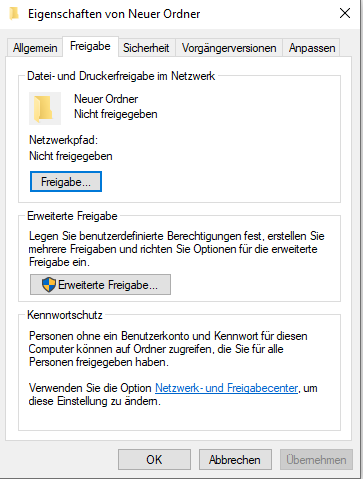
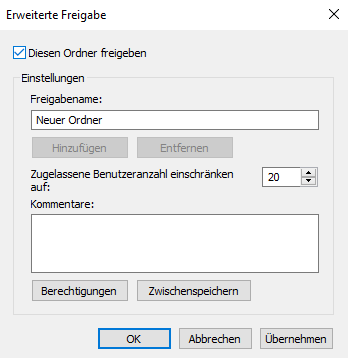
# JavaScript

# Authorisation Bypass

# Open HTTP Redirect

Protokoll 1

1. Zu beginn haben wir gelernt, wie man eine Datei für andere PC’s freigeben kann bzw. wie man einen Ordner verstecken kann (bei Freigabename ein $ hinzufügen)

Ordner freigeben:

2. Danach haben wir über die Komponenten eines PC’s, also CPU, Grafikkarte, Festplatte, Netzteil, CD-ROM, Ram, Mainboard, gesprochen und auch über die Eigenschaften von den genannten gerededet

