



CSEC 471 - Penetration Testing

Penetration Testing On DASH

Spring 2025

Group 7

Student Name	UID Number
Alissar Salman	386004938
Sufian Al Hattab	377004971
Ahmed Abu'shar	386005103
Kassem Darawcha	377004267

Dr. Kevser Ovaz Akpinar

April 24, 2025

Table Of Contents

Contents

1. DOCUMENT PROPERTIES	4
2. VERSION CONTROL.....	4
3. EXECUTIVE SUMMARY	5
3.1 SCOPE OF THE RISK ASSESSMENT	5
3.2 PROJECT OBJECTIVE	7
3.3 ASSUMPTIONS	7
3.4 TIMELINE.....	8
3.5 SUMMARY OF FINDINGS	9
3.6 SUMMARY OF RECOMMENDATIONS	11
4. METHODOLOGY	13
Methodology Phases:	13
5.1 Objective	15
5.2 Details Provided.....	15
5.3 Steps Taken	16
5.4 Key Findings.....	19
6 DETAILED STEPS FOR EXPLOITATION.....	20
6.1 AUTHENTICATION BYPASS – SQLi.....	20
6.1.1 Way of Exploitation.....	20
6.1.2 The reason for the vulnerability	22
6.1.3 Tools used	22
6.1.4 Risk Analysis.....	23
6.1.5 Recommendation.....	24
6.2 BROKEN ACCESS CONTROL	25
6.2.1 Way of Exploitation.....	25
6.2.2 The reason for the vulnerability	26
6.2.3 Tools used	27
6.2.4 Risk Analysis.....	27
6.2.5 Recommendation.....	28
6.3 INSECURE DIRECT OBJECT REFERENCE (IDOR).....	29

6.3.1 Way of Exploitation.....	29
6.3.2 The reason for the vulnerability	29
6.3.3 Tools used	30
6.3.4 Risk Analysis.....	30
6.3.5 Recommendation.....	31
6.4 CROSS-SITE REQUEST FORGERY (CSRF)	32
6.4.1 Way of Exploitation.....	32
6.4.2 The reason for the vulnerability	35
6.4.3 Tools used	35
6.4.4 Risk Analysis.....	36
6.4.5 Recommendation.....	37
7 INNOVATION JOURNEY	38
References	40

1. DOCUMENT PROPERTIES

Property	Details
Title	DASH Web Application Penetration Testing Report
Version	V1.0
Author	Group 7 : Kassem Darawcha - Sufian Al Hattab - Alissar Salman - Ahmed Abu'shar
Pen-Tester	Group 7 : Kassem Darawcha - Sufian Al Hattab - Alissar Salman - Ahmed Abu'shar
Reviewed By	Kevser Ovaz Akpinar
Approved By	Kevser Ovaz Akpinar
Classification	Confidential

2. VERSION CONTROL

Version	Date	Author	Description
V1.0	April 23, 2025	Group 7	First Draft

3. EXECUTIVE SUMMARY

This report involves simulating a real world cyber attack against a virtual custom built network and a web application called DASH. In the first phase, we created a virtual organization which simulates a real world small office environment that includes many different features such as users, different services, and the addition of vulnerabilities to test our environment against attacks. These vulnerabilities included simple and default passwords, no encryption, and other misconfigurations in the network which allowed us to have access to the network, view some private files, and even gain control over the network.

In the second phase, we performed an attack on another group's web application which was called DASH. We found many security vulnerabilities present, since we were able to log in without a real password (SQL injection), and we were also able to view data of other users just by changing the URL which is a method called IDOR. This shows that it is necessary to implement strong security features to avoid system break ins and access of sensitive data.

3.1 SCOPE OF THE RISK ASSESSMENT

The scope of this risk assessment includes two main sections which include a virtual organization that simulates a small real life office environment which was created by our group, and the second being an external web application called DASH which was created by another group. The main goal of this was to simulate real life cyberattacks to test both our group's environment and the other group's. Through the test, we were able to identify vulnerabilities and decide how critical their impact can be on the security of the network.

The first part of the assessment included dividing the network into 2 VLANS (VLAN 10 which was for employees and VLAN 20 which was used for the supervisor and for file sharing). In this part, our group intentionally left out vulnerabilities by setting up weak configurations for Telnet, SNMP, RDP, and SMB in order to be able to test how easily an attacker can exploit the network.

The goal of the second part of the assessment included testing DASH vulnerabilities, the web application which was created by the other group. We found the DASH web application to be vulnerable to things such as SQL injection, and IDOR which lets us view user data by modifying the URL. All of this shows us that a lack of proper access control and poorly configured input handling can seriously compromise the systems security.

The assessment included the following:

- Identifying and then exploiting the vulnerabilities which included weak and default credentials, insecure protocols, misconfigured ACL's, SQL injection, and IDOR.
- Scanning the target IP addresses (192.168.99.X, 192.168.98.X, 192.168.192.131) in order to find out what ports are open in addition to running services.
- Analyzing and checking the impact of a successful exploitation in both parts, the virtual network which simulated a real small office environment, and the web application 'DASH'.
- Providing recommendations which will help with mitigating the found vulnerabilities in order to strengthen the security of the systems.

All the testing that was performed was done in a controlled environment to safely explore vulnerabilities and avoid affecting any real live systems. The results that we found are all based on how the state of the systems were during the time of testing.

3.2 PROJECT OBJECTIVE

The objective of this project is to perform a test on 'DASH' web application created by another group. We aimed to identify 4 web vulnerabilities in 'DASH' such as SQL injection and BAC, IDOR and CSRF and then check how these vulnerabilities can be used to access sensitive information. We checked how well the current security controls work, tested how attacks could affect confidentiality, integrity, and availability (CIA), and gave recommendations to fix the risks. The main goal was to help improve the organization's security and highlight important weaknesses. Moreover, the main goal is to understand how attackers think in terms of exploiting vulnerabilities, and then learn what mitigations can be applied to avoid compromising the security of the systems and user data, in addition to how they can be implemented.

3.3 ASSUMPTIONS

These are all assumptions that were made during the penetration testing process:

- The virtual network we created accurately represents a real world small office environment since it also includes user, services, and configurations.
- All the vulnerabilities that are present in the virtual network were intentionally added for the sake of performing penetration testing.
- 'DASH' web application which was created by the other group (external party) is considered to be functional and also an intentionally vulnerable system for the purpose of external testing.
- All of the provided IP addresses, configurations, in addition to credentials were all correct and active during the process of the assessment.

- We can assume that vulnerabilities that were found during the assessment process all exist unless only it is proven otherwise.
- No live systems were affected since the tests performed were done in an isolated environment.
- System tests that were made all involved using access levels that a basic user or attacker has realistically.

3.4 TIMELINE

The timeline below outlines the major phases of our initial penetration testing project, carried out between April 7, 2024, and April 24, 2024.

Penetration testing timeline:

Phase	Date
Project Initiation and Setup	07/04/2025
Reconnaissance Phase	10/04/2025
Vulnerability Identification	11/04/2025 - 12/04/2025
Exploitation and Testing	13/04/2025 - 14/04/2025
Report Writing	15/04/2025 - 24/04/2025

3.5 SUMMARY OF FINDINGS

DASH Web Application Risk Level Table

Risk Level	Number of Vulnerabilities
Low	0
Medium	0
High	2
Critical	2

DASH Web Application Risk Distribution

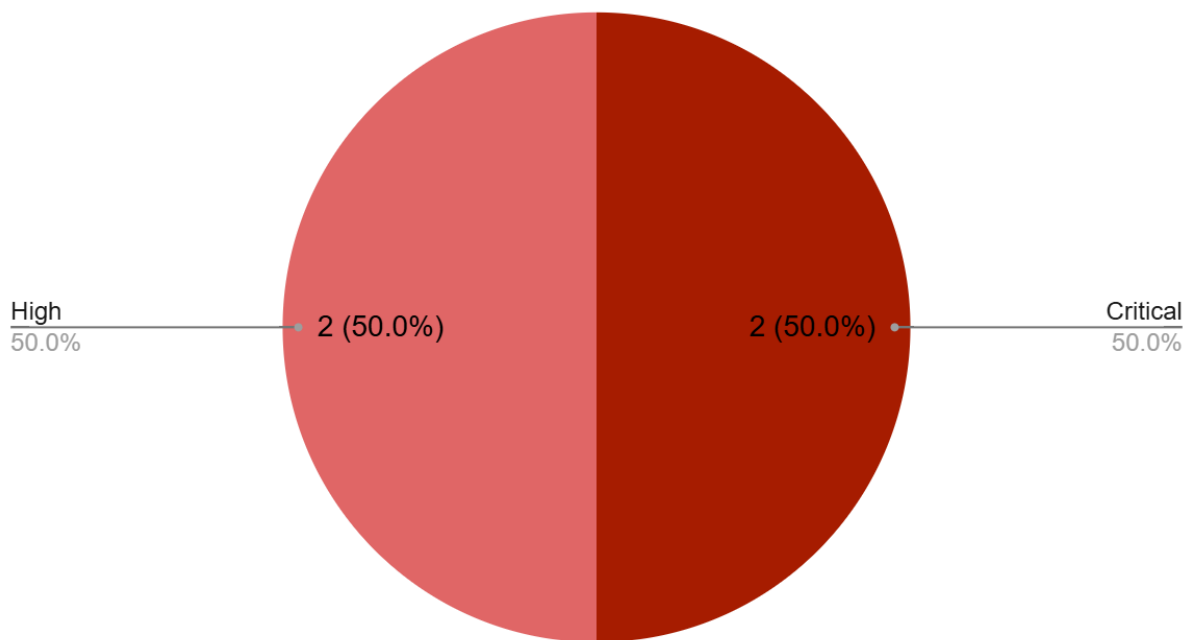


Figure 1.0
Dash Web Application Risk Distribution

1. SQL injection:

a. We were able to manipulate user input in a login form by inserting special characters in the field 'username', and this allowed us to log in without actually knowing the real password. This means that the website did not check the user input properly which is extremely critical since it gives full access to the attacker.

b. Risk Level: Critical

2. Broken Access Control

a. Since there were no proper authorization checks through role based access checks, users were able to access admin areas and private and sensitive files that were supposed to be restricted.

b. Risk Level: Critical

3. Insecure Direct Object Reference (IDOR):

a. The object level access control was clearly not enforced properly or misconfigured since users could access other's data and their profiles just by changing the numerical values that are in the URL.

b. Risk Level: High

4. Cross-Site Request Forgery (CSRF):

- a. A malicious HTML form was created so that when it is triggered (clicked on by a user) it is then used to submit requests on behalf of the user that is logged in. This allowed us to make some unauthorized changes to the data of the user without their knowledge and permission.
- b. Risk Level: High

3.6 SUMMARY OF RECOMMENDATIONS

It is recommended that a defense-in-depth strategy be followed to fortify the overall security posture by mitigating threats at web application levels. Priority should be accorded to the following steps:

1. Sanitize User Input and Parameterize Queries: Prevent SQL injection by keeping the inputs clean and use prepared statements instead of dynamic SQL queries. A Web Application Firewall (WAF) installed will provide extra protection.
2. Fix IDOR and BAC Access Control Flaws: Restrict users to view only their data by tightening the access controls. Enforce role-based access control (RBAC) and maintain distinct, non-sequential identities at all the sensitive endpoints.
3. Apply CSRF Protection: Employ CSRF tokens, validate each incoming request source, and set SameSite and HttpOnly properties to the cookies to avoid unwanted actions by attackers.

4. To keep DASH secure, use **Role-Based Access Control (RBAC)** to check user roles on the backend before giving access to sensitive data (OWASP, n.d.). Always **validate access on the server**, not just the frontend (Security and Privacy Controls, 2020).
5. Add **middleware** to block unauthorized users based on session and role (Security and Privacy Controls, 2020). **Log and monitor** all access to sensitive data and flag anything unusual (OWASP, 2021).
6. Make sure users can **only access their own data**. Use random or hashed IDs to prevent guessing (OWASP, 2013), and test for **IDOR** by changing IDs in requests (OWASP, n.d.).
7. Protect against **CSRF** by using **POST requests** for actions, and don't let data change just from clicking a link (OWASP, 2012). Add **origin/referrer checks** to block fake requests.

4. METHODOLOGY

Methodology Phases:

We followed an approach that consists of the following:

Phase	Description
Reconnaissance	Using tools like Nmap to gather information about the DASH web application such as its pages and inputs.
Vulnerability Assessment	Testing how the web application called DASH can handle input, access, and sessions. Through the testing process, vulnerabilities such as SQL injection, IDOR, broken access control, and CSRF were found.
Exploitation	We were able to bypass login by using SQL injection, also accessing other users data with IDOR, changed data using cross site request forgery (CSRF) and finally access unauthorized areas because of broken access controls.
Reporting	Documentation of all our findings was performed such as including screenshots, analyzing the severity of each vulnerability that was present, and then providing recommendations for all vulnerabilities to mitigate them.

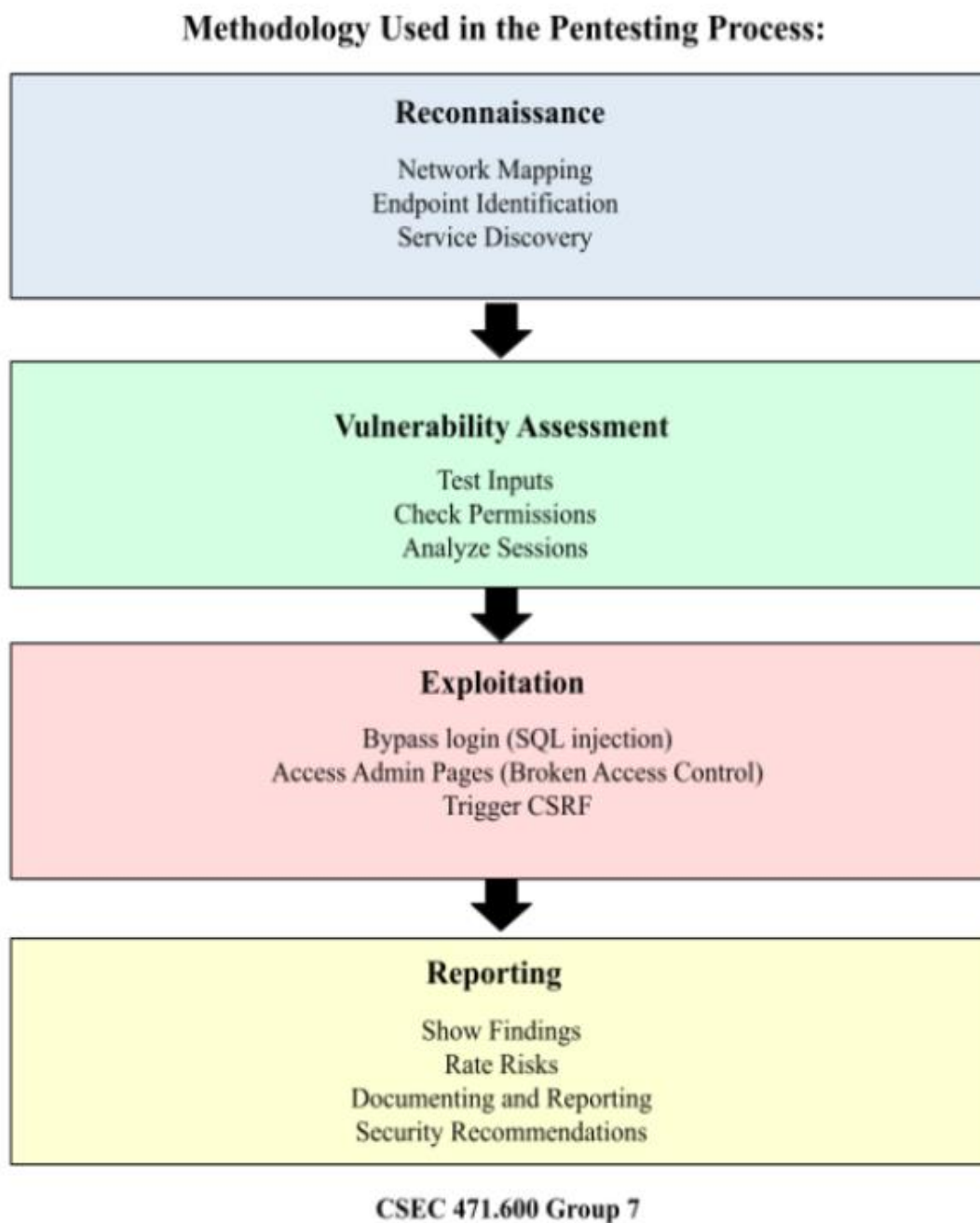


Figure 2.0
Methodology Used in this PT

5. RECONNAISSANCE

5.1 Objective

The objective of the reconnaissance phase is mainly to collect and gather information about the DASH web application as much as possible. This information gathering will help with being able to identify the potential attack surface. The attack surface included finding pages that were exposed, input fields, some hidden directories and request parameters, in which all of this can be used later on to identify vulnerabilities and then use them for exploitation.

5.2 Details Provided

We were given the following details about the testing environment:

- Target IP Address: 192.168.192.0/24.
- Access Credentials for Kali Linux VM:
 - o Username: kali
 - o Password: kali
- Environment Setup:
 1. Kali Linux VM: This was the machine we used for testing.
 2. Windows VM (Database): Running database services.
 3. Windows VM (Code): Hosting application code

5.3 Steps Taken

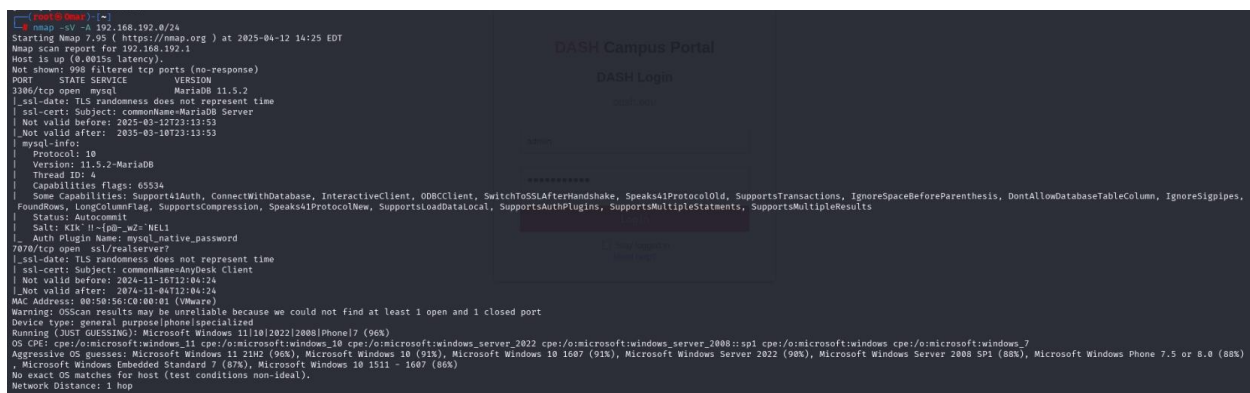
1. Preparation of the testing environment:

We started by first setting up Kali Linux virtual machine and then accessing it. This virtual machine worked as our main system in order to conduct reconnaissance and all needed scanning against the target which is the DASH web application.

2. Scanning the target network:

We ran the command ‘nmap -sV -A 192.168.192.0/24’ to perform an nmap scan on the target system in order to identify what ports and services are open.

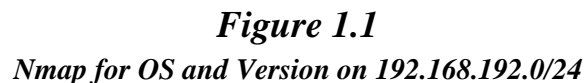
- -sV option was used to be able to detect the versions of the services.
- -O Enables aggressive scan options, which includes: OS detection ,Version detection , Script scanning and Traceroute.



```

root@kali:~# nmap -sV -A 192.168.192.0/24
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-12 14:25 EDT
Nmap scan report for 192.168.192.1
Host is up (0.0015s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
3306/tcp   open  mysql        MariaDB 11.5.2
|_ ssl-date: TLS randomness does not represent time
|_ ssl-cert: Subject: commonName=MariaDB Server
|_ Not valid before: 2025-03-12T23:13:53
|_ Not valid after: 2035-03-10T23:13:53
|_ mysql-info:
|   | Protocol: 10
|   | Version: 11.5.2-MariaDB
|   | Thread ID: 4
|   | Capabilities flags: 65534
|   | Some Capabilities: Supports4Auth, ConnectWithDatabase, InteractiveClient, ODBCClient, SwitchToSSLAfterHandshake, Speaks4ProtocolOld, SupportsTransactions, IgnoreSpaceBeforeParenthesis, DontAllowDatabaseTableColumn, IgnoreSigpipes, FoundRows, LongColumnFlag, SupportsCompression, Speaks4ProtocolNew, SupportsLoadDataLocal, SupportsAuthPlugins, SupportsMultipleStatements, SupportsMultipleResults
|   | Status: Autocommit
|   | Salt: KXk H-[p]-w2-NELl
|   | Auth Plugin Name: mysql_native_password
7070/tcp   open  ssl/realserver?
|_ ssl-date: TLS randomness does not represent time
|_ ssl-cert: Subject: commonName=AnyDesk Client
|_ Not valid before: 2024-11-16T12:04:24
|_ Not valid after: 2074-11-04T12:04:24
MAC Address: 08:50:56:C8:00:01 (VMware)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose/phone/specialized
Running (JUST GUESSING): Microsoft Windows 11/10/2022/2008/Phone/7 (96%)
OS CPE: cpe:/o:microsoft:windows_11 cpe:/o:microsoft:windows_10 cpe:/o:microsoft:windows_server_2022 cpe:/o:microsoft:windows_server_2008:sp1 cpe:/o:microsoft:windows cpe:/o:microsoft:windows_7
Aggressive OS guesses: Microsoft Windows 11 21H2 (98%), Microsoft Windows 10 (93%), Microsoft Windows Server 2022 (90%), Microsoft Windows Server 2008 SP1 (88%), Microsoft Windows Phone 7.5 or 8.0 (88%), Microsoft Windows Embedded Standard 7 (87%), Microsoft Windows 10 1511 - 1607 (86%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop
  
```

Figure 1.0
Nmap for OS and Version on 192.168.192.0/24



```
Nmap scan report for 192.168.192.131
Host is up (0.0011s latency).
Not shown: 996 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
135/tcp    open  msrpc            Microsoft Windows RPC
139/tcp    open  netbios-ssn      Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds?
3000/tcp   open  http             Node.js (Express middleware)
|_http-title: DASH Login
MAC Address: 00:0C:29:D3:8A:90 (VMware)
Device type: general purpose
Running: Microsoft Windows 10
OS CPE: cpe:/o:microsoft:windows_10
OS details: Microsoft Windows 10 1709 - 21H2
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
| smb2-security-mode:
|   3:1:1:
|_   Message signing enabled but not required
|_nbstat: NetBIOS name: DESKTOP-DK2Q0IV, NetBIOS user: <unknown>, NetBIOS MAC: 00:0c:29:d3:8a:90 (VMware)
| smb2-time:
|   date: 2025-04-12T18:26:31
|_  start_date: N/A

TRACEROUTE
HOP RTT      ADDRESS
1   1.11 ms  192.168.192.131
```

Figure 1.3
Nmap Scan for host “ 192.168.192.131 “

3. Findings and observations of nmap results:

- **Status of the host:** Host was identified to be active on the network.
- Many hosts were found to be running on the network and each host was running on a different operating system.
- We found that on host '192.168.192.131' port 445 (tcp) was open and which is used for SMB and also port 3000 (tcp) was open which was running using a web application called Node.js using the Express framework.
- On port 3000 the title of the HTTP showed 'DASH Login' which means that the host was in fact running the target web application 'DASH'.
- '00:0C:29:D3:8A:90' which is the MAC address of the host with IP address '192.168.192.131' showed that the host was a virtual machine which was running on VMware.

5.4 Key Findings

- The Nmap scan showed us that there were multiple hosts that were active on the network. Host '192.168.192.131' was running the DASH web application (which is the target), port 3000 (tcp) was open which showed that the web application was live and accessible, and the mac address of the host confirmed that the virtual machine was running on VMware.
- In addition, when we identified the web application which was running on port 3000, it helped us focus our testing process more on the possible issues that might be present in the DASH web application.

6 DETAILED STEPS FOR EXPLOITATION

6.1 AUTHENTICATION BYPASS – SQLi

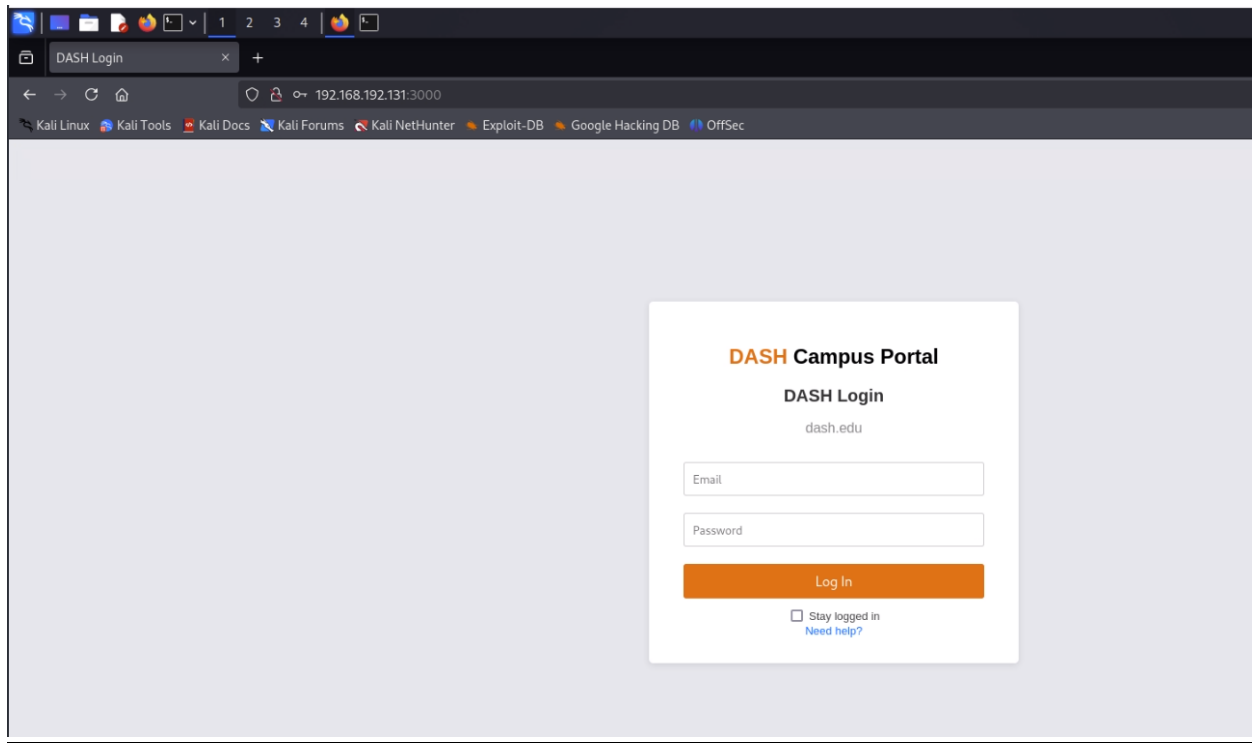


Figure 2.0

Accessing http website for 192.168.192.131 on port 3000

6.1.1 Way of Exploitation

We exploited the vulnerability by opening a web browser and going to the target IP and port found during reconnaissance (<http://192.168.192.131:3000>). When the login page appeared, we tried to enter SQL payloads into the login form where the username and password fields are by using the ' OR 1=1 --' injection, which worked since we were able to bypass login without the

actual credentials. This shows us that the user input handling was not properly done which allowed u to manipulate the SQL query which is for authentication.

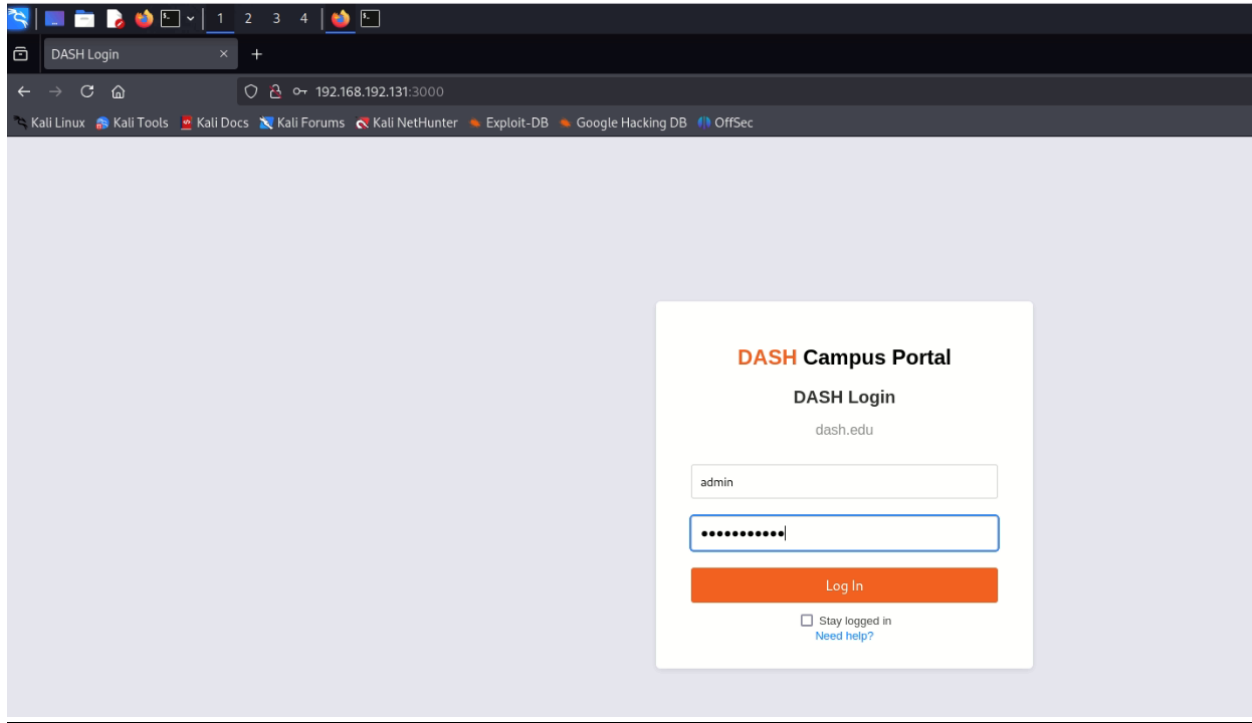


Figure 3.0
SQLi on DASH login portal

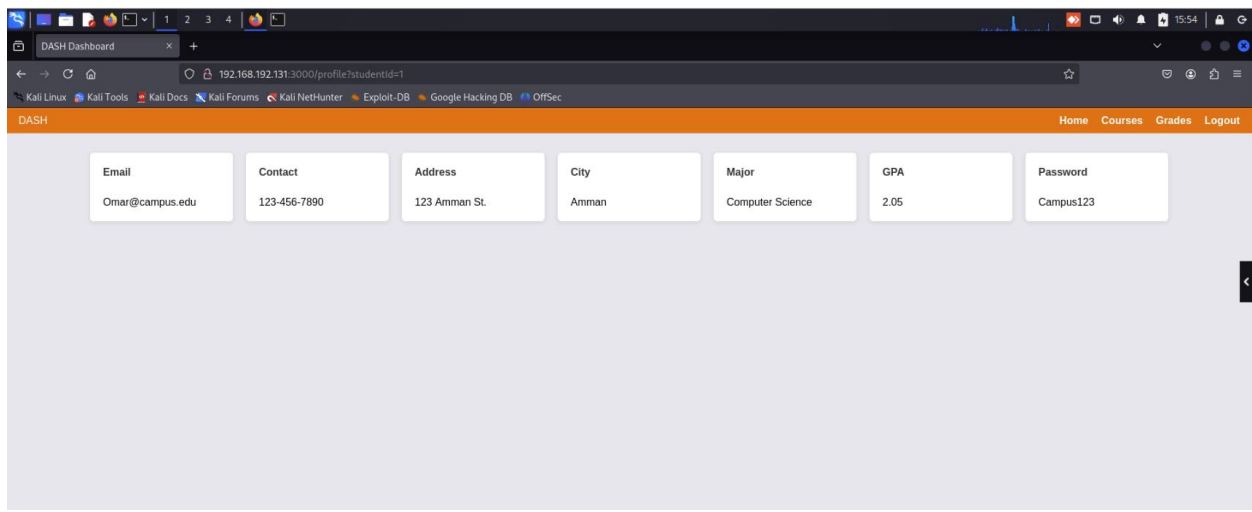


Figure 4.0
In band SQL injection worked on DASH login page.

6.1.2 The reason for the vulnerability

The reason that this SQL injection vulnerability exists is because user input handling in the login form was not properly done. For example, when a user sends an input and then it is passed directly into the backend SQL query without being checked or verified, it would be very easy for attackers to just insert malicious code in order to change the behavior of the SQL query. According to (OWASP 2023), web applications that do not implement any input sanitization or checking user input before it is validated, are very likely to be vulnerable to SQL injection. Improper user input handling will allow attackers to bypass login authentication without real credentials and that puts the application at risk of unauthorized access.

6.1.3 Tools used

Web Browser	We used a web browser to access the login page hosted at http://192.168.192.131:3000 .
Manual Payloads	We entered some simple SQL payloads into the login field such as ' OR 1=1 --' while performing the testing process.

6.1.4 Risk Analysis

Threat level:

High

SQL injection is a vulnerability that creates a very serious threat since it lets attackers be able to bypass login forms without using real credentials. The attackers are able to input malicious code into the SQL query of the application which then allows the attacker to be able to manipulate the query and also gain unauthorized access to the web application.

Vulnerability:

Critical

The vulnerability is critical because when the login form accepts what the user has input into the login field without checking or validating the input, the attacker will easily be able to insert malicious SQL code and gain unauthorized access. Without validating user input, the web application will be very highly vulnerable to an SQL injection attack.

Impact:

High

If the attacker was able to successfully exploit this vulnerability, they could gain access to sensitive data such as accounts of users and even admin areas, which all can lead to compromising the security of the web application and data breaches, leading to lots of damage for both users and the organization.

Risk Rating:

Critical

This SQL injection vulnerability is seen as a very critical vulnerability that can be very easily exploited by attackers. This is why it should be mitigated because it can cause a lot of damage to the organization.

6.1.5 Recommendation

- **Using parameterized queries** will help to separate the user input from the SQL query. This works because even if an attacker injects malicious code into the login field, it will just be treated as data and not actually part of the query (OWASP, 2025).
- **Setting up a Web Application Firewall** so that when an attempt of an SQL injection attack is made, the firewall will help with blocking it in real time. It adds an extra layer of protection because it checks for attack patterns that are already known (OWASP, 2025).
- **Applying input filtering** to filter out inputs that users may inject such as ', ", --, =, etc. These characters are almost always used in codes that are malicious which use to inject into login fields. By filtering these characters, it will be less likely that an attacker can bypass the login (OWASP, 2025).
- **Validating and sanitizing user input** which will make sure that any data that is entered in the login fields do not contain any malicious codes that can compromise the security of the application (OWASP, 2025).

6.2 BROKEN ACCESS CONTROL

6.2.1 Way of Exploitation

The successful sql injection allowed us to bypass the login page so that we can identify the access control vulnerabilities within the webs structure. we gained unauthorized access although we didn't have admin privileges , and was carried out through direct URL manipulation. After successfully logging in via an SQL Injection vulnerability, I was redirected to a standard user profile page located at `/profile?studentid=2`. Observing the structure of this URL, I hypothesized that access to administrative resources might be available under a similarly structured endpoint. By substituting the “profile” path with “admin”, I navigated to `/admin?studentid=2` . Surprisingly, the server granted access to the administrative interface without verifying whether the logged-in user had sufficient privileges. , we were able to see the admin dashboard and access sensitive files and resources. The system had no authorization feature which checked the user rights before showing the administrator content. The result allowed all users to have administrator content just by accessing the url.

A proper authorization logic deficiency existed in the backend system alongside inadequate enforcement of rules at critical routes.

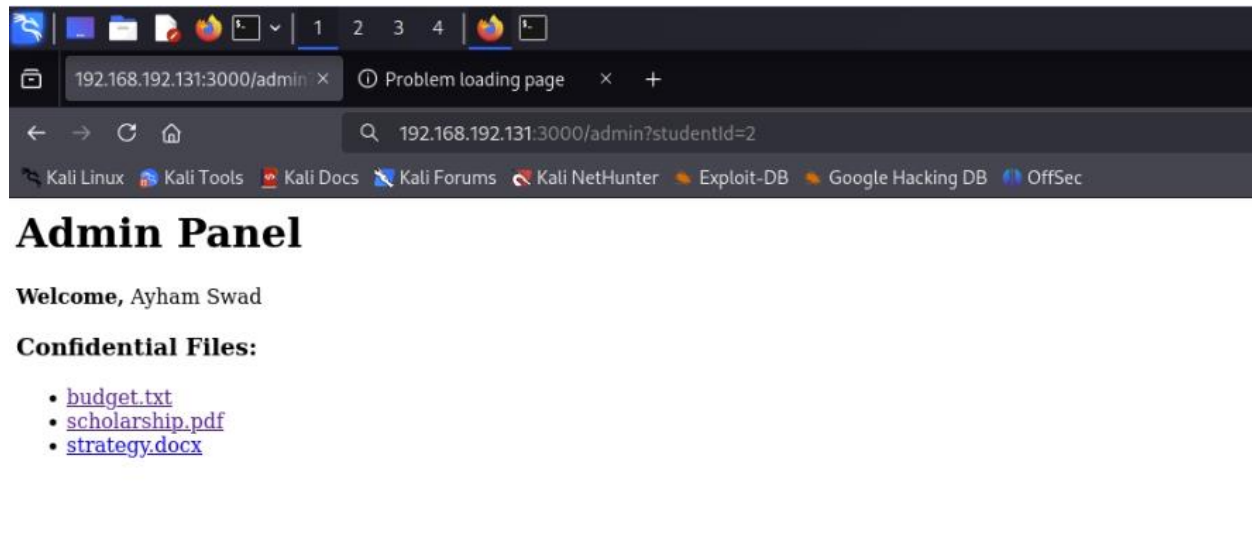


Figure 6.1

BAC worked on DASH login page.

6.2.2 The reason for the vulnerability

The system vulnerability emerged because critical routes do not contain proper role-based access control (RBAC) systems. The backend of the application failed to verify user permissions despite the frontend interface concealing particular buttons which only allowed administrative access or sensitive functionality.

Unpredictable paths which users can find manually result in privilege escalation because these users are able to access data meant for elevated user roles through guessing methods. The (OWASP, 2023) rates Broken Access Control as a leading vulnerability because attackers will seize the opportunity to misuse their access permissions.

6.2.3 Tools used

1. Manual URL entry of admin-specific paths like /admin permitted us to access those specific pages directly through the browser.
2. The Browser Developer Tools system enables users to track requests and responses and check individual web elements for undetectable or unreachable admin paths.
3. Manual testing required no automated tools because logical analysis and trial-run web navigation procedures were used for completion.

6.2.4 Risk Analysis

Threat Level:

High

This shows how likely the attacker will exploit the vulnerability , since as mentioned before any user could have access without admin privilege by just modifying the url, the chance of sensitive resources being discovered is very high.

Vulnerability:

Critical

The vulnerability itself is considered to be critical , and thats because it completely fails to enforce access control , so if anyone logged in , they can gain full access to sensitive data just by navigating to hidden routes

Impact:**High**

This could lead to sensitive data being stolen , and also could lead to full compromise of the application . the attack can access admin dashboards and view or manipulate sensitive files , which can lead to damaged data integrity and even worse .

Risk Rating:**Critical**

The risk rating is critical, as the system is shown to be highly exposed to attackers stealing/modifying sensitive files.

6.2.5 Recommendation

1. Apply role-based access control (RBAC): it checks user roles on the backend before giving access to any sensitive routes like data (OWASP, n.d.).
2. Secure backend routes: be more careful on what you trust , never rely on frontend control alone , best option is to validate access server-side (*Security and Privacy Controls for Information Systems and Organizations*, 2020)
3. Use middleware: it restricts unauthorized access based on users roles and session data (*Security and Privacy Controls for Information Systems and Organizations*, 2020)
4. Log and monitor: track all access to sensitive data or resources that may be in the risk of being compromised and flag unusual patterns (OWASP, 2021).

6.3 INSECURE DIRECT OBJECT REFERENCE (IDOR)

6.3.1 Way of Exploitation

We were able to access profiles of other users in addition to their information and data without being restricted since we manually changed the numerical value that existed in the URL to another value. This shows us that the web application did not perform any checks to check if the user that was logged in was actually authorized or not to log in and access the data that is linked with the user ID in the URL. The application just simply trusted that that the numerical value in the URL was correct without any verification which allowed us to access data that should have been restricted.

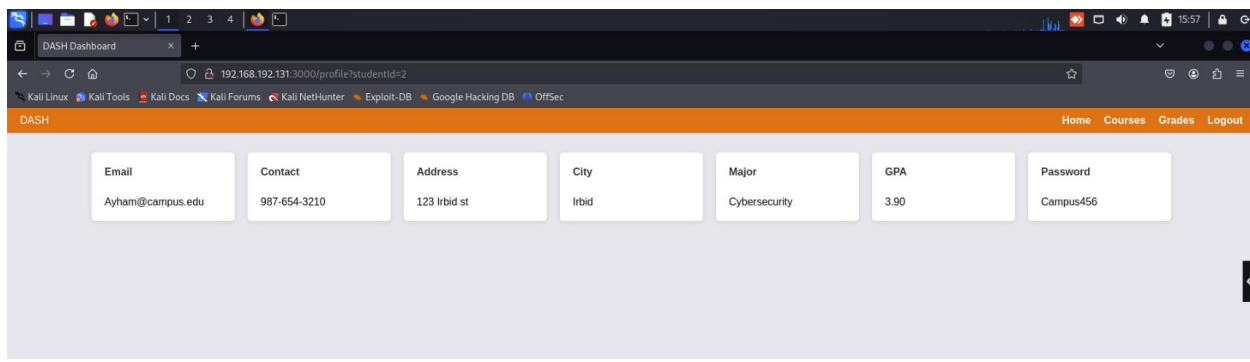


Figure 5.0

IDOR worked on DASH login page.

6.3.2 The reason for the vulnerability

The reason for this vulnerability is because the application had decided to blindly trust the numerical value that was in the URL without any kind of verification to see if the user actually had permission to access the data. There were no access control checks present that make sure that

only the user is allowed to view their own information, therefore, attackers can very easily manipulate those numerical values in URL since the application relies on them in terms of access.

6.3.3 Tools used

Web Browser	We used this to be able to directly manipulate the URL to check if we could access data with being authorized to.
--------------------	-------------------------------------------------------------------------------------------------------------------

6.3.4 Risk Analysis

Threat level:

High

This vulnerability is considered a high threat level vulnerability because it is very easy to exploit.

An attacker only has to change a numerical value in the URL and then they will be able to access some other users data. Since the application does not check if the user is actually allowed to see that data, attackers are very easily able to expose sensitive information which is that of users.

Vulnerability:

Critical

This vulnerability is critical because the application just trusts that the number given in the URL is correct and does not check if the user actually has permission to access the data which is connected to the id in the URL, all of this creates a problem where the sensitive data of users can be exposed

Impact:

High

An attacker will be able to access all sensitive information related to users if they exploit this vulnerability, which can lead to issues in terms of privacy and users will no longer trust the application since their personal information can be accessed and seen by others easily.

Risk Rating:

High

This has a high risk rating since the attack is very simple to execute and it can still expose very sensitive information especially that of users.

6.3.5 Recommendation

- **Adding access control checks** will help with making sure that users are only able to access data that is their own, so before returning any data to the user, the application must check if they are authorized to view it (OWASP, n.d).
- **IDs that can easily be guessed are recommended to be avoided.** For example, using random ids or hashed values will make it way harder for an attacker to be able to guess (OWASP, 2013).
- Checking if data access is actually being restricted properly by doing **regular IDOR tests**. This can be done by changing identifiers that are in the URL or the request bodies (*WSTG - Latest / OWASP, n.d.*).
- **Access control should only happen on the server's side**, so what is in the URL or the client's side should never be relied on since these can always be tampered with (OWASP, 2021a).

6.4 CROSS-SITE REQUEST FORGERY (CSRF)

6.4.1 Way of Exploitation

In order to exploit this vulnerability we used the HTML hint to create a fake HTML form that would send a request automatically, on behalf of the user that's logged in already, to the DASH web application. We set this form up to change the user's GPA but without them knowing that and then once they open the page or even click a link, that form will automatically be submitted. Because the application did not have any protections like for example a CSRF token or an origin check, it just accepted the request with no verifications as if the request actually came from the user. Therefore, we were able to make changes to the data of the user without their permission or knowledge all because the application did not check where the request was coming from exactly.

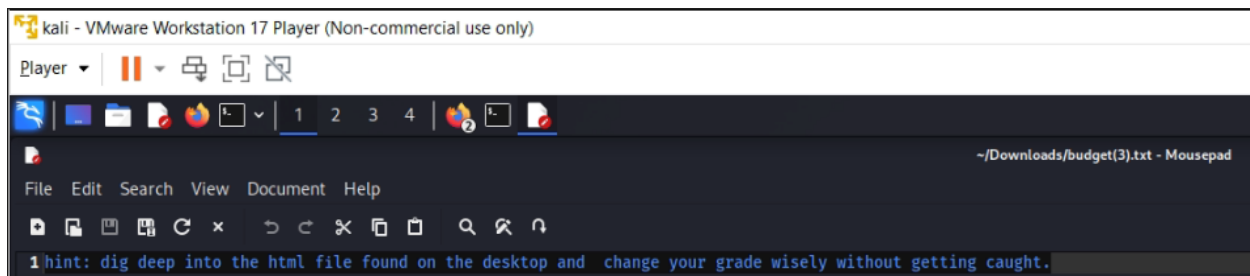


Figure 7.0

Budget.txt file contains *Hint on HTML file*.

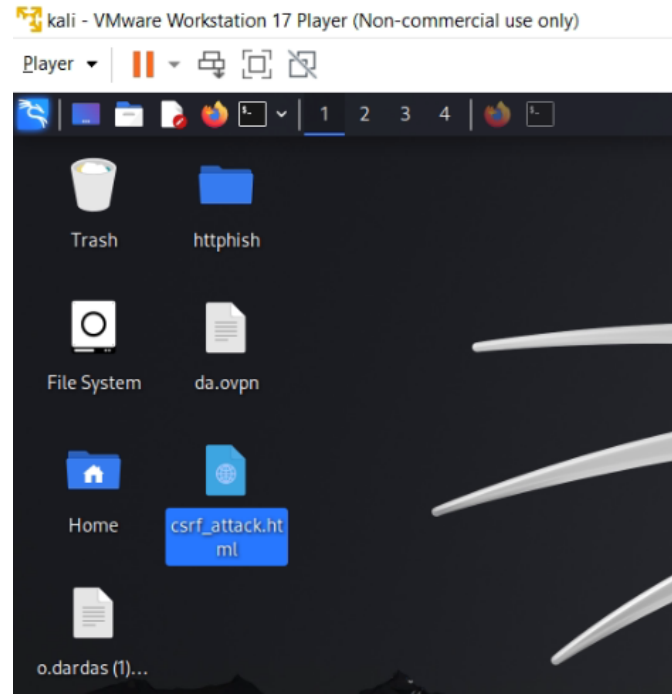


Figure 8.0

HTML file Found

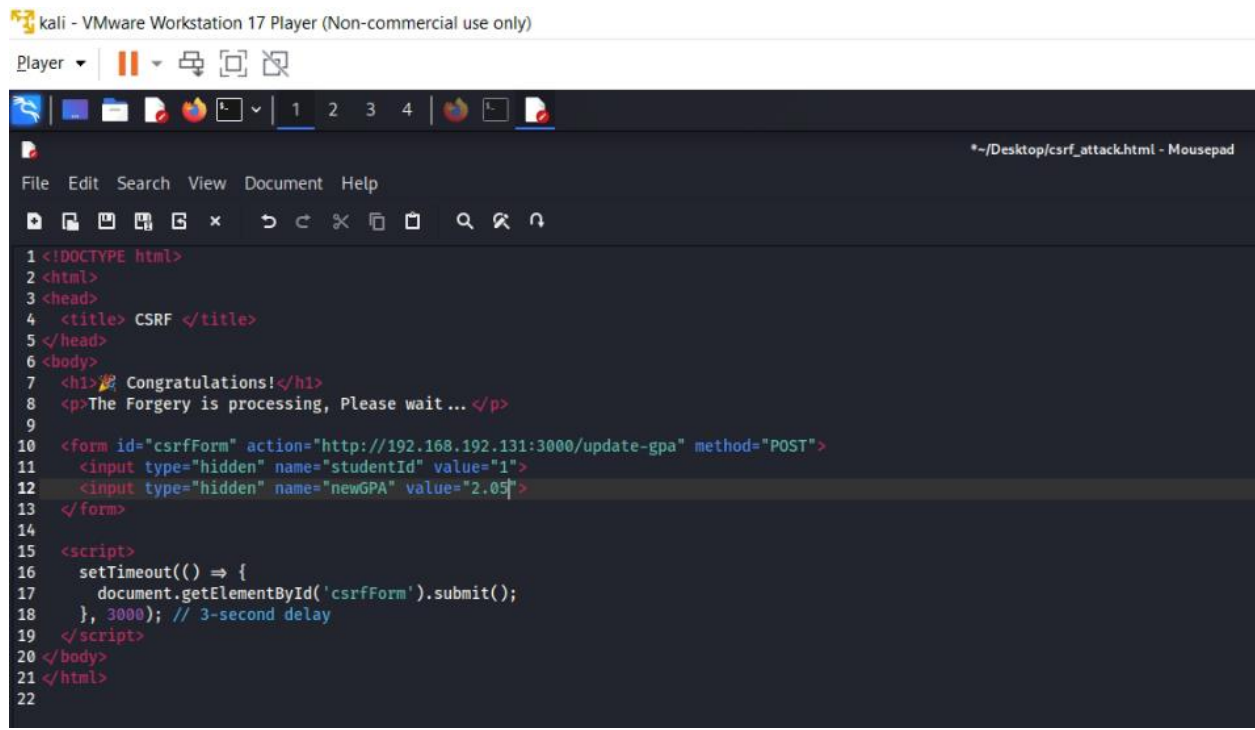


Figure 8.1

HTML Code For CSRF Vulnerability

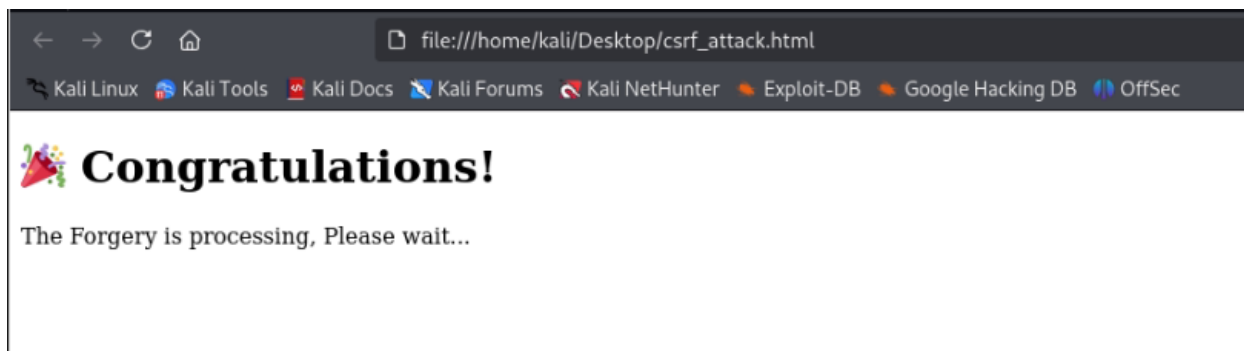


Figure 8.3

Executed the HTML code and it's loading

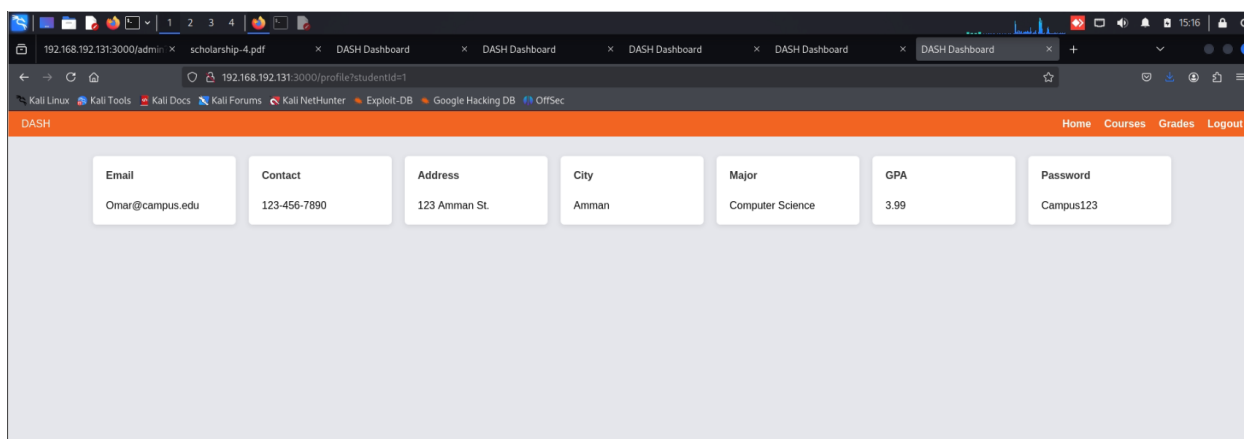


Figure 8.4

The HTML code nailed the CSRF vulnerability

6.4.2 The reason for the vulnerability

The reason that this vulnerability exists is because the DASH web application did not implement any kind of protections against unauthorized requests that were sent on behalf of the user without their knowledge as if the request actually came from the user. The web application did not use a CSRF token which is a unique session based value which would confirm that the request that is being made is actually made by the user, neither did it check where the request came from and it just accepted the request. This shows that attackers can easily take advantage of this vulnerability and submit requests without the users permission or knowledge since the application does not perform any checks to see if the request was actually made by the user or not.

6.4.3 Tools used

Hint From Admin Files	Once navigated to Admin Dashboard, found sensitive files and found a hint about an HTML file on the Desktop
Mousepad Text Editor	Used Mousepad to edit the HTML file

Web Browser	We used the browser to enable the HTML code that executes the CSRF.
--------------------	---------------------------------------------------------------------

6.4.4 Risk Analysis

Threat level:

High

This vulnerability is considered to be a high threat because attackers can easily modify the data of a user without their knowledge as long as the user is logged in. Although it needs the user to click a link or visit the page themselves, it still is a high threat because the request is accepted by the server without any checks or verification that the request actually came from the user.

Vulnerability:

Critical

Since the DASH web application did not perform an origin check to see if the request was actually coming from the user or no, and also did not use CSRF tokens, it was possible for attackers to create a fake form in order to change the data of a user if they were logged in (such as the user's GPA) and without them knowing.

Impact:**High**

The CSRF vulnerability has a high impact because it allows attackers to change the data of the user without them knowing. In our case, we were able to change the user's GPA just by having them open a page that had the fake HTML form that we created, which can really affect the integrity of the data of users.

Risk Rating:**High**

The risk is rated to be high because although it does need some kind of user interaction, in this case clicking a link or visiting a page, it still acts as high risk since the server does not have any kind of security checks to check if the requests have been submitted by the actual user, making the user's data prone to being changed by an attacker.

6.4.5 Recommendation

- **Using CSRF tokens** (unique token that is session based to ensure that the user is the one making the request) in addition to them being verified by the server before the data is shown so the server ensures that the ones making the requests are the users themselves (OWASP, 2012).
- Make sure that the important things are done **using POST request** and not GET, and also do not allow any changes to be made to data when just a link is clicked or a page is loaded (OWASP, 2012).

- Developers must know how the CSRF vulnerability works and they should **test the application regularly** to make sure that protections they have implemented are working properly. They can do that using tools like BurpSuite.
- **The addition of origin checks and referrer checks** are recommended in order to confirm that the requests being submitted are coming from the user and not from a fake site or form (OWASP, 2012).
- Don't leave any HTML files that is sensitive and could harm the system such as the one we found.

7 INNOVATION JOURNEY

Our penetration testing project involved a lot of innovative and technical problem solving skills. The penetration testing process that we conducted was divided into two phases, phase 1 focusing on the virtual network that we have created to simulate a real-world small office environment, and phase 2 which involved exploiting another group's DASH web application. Phase 1 setup for our own network included setting up certain services like Telnet, SMB, and SNMP, all with purposely added misconfigurations to test how easy it would be for an attacker to exploit our network. In this phase, we found that we had a total of six vulnerabilities which included using default or weak credentials, using protocols that are insecure, and also access control between both VLANS (VLAN 10 and VLAN 20) was misconfigured.

In phase 2, our focus shifted to the DASH application, and since we did not build it ourselves, it had to be treated as if it were a real world target and with almost little to no visibility on how the application worked and what misconfigurations may be present. Through that, we were pushed to think more innovatively and focused on using some creative techniques in order to find

the vulnerabilities in the web application. We found a total of four vulnerabilities in DASH which included SQL injection, IDOR, CSRF, and also broken access control.

This project made our approach to be less basic and more creative through the use of different tools and also manual testing. We did not just follow standard steps, in fact we had constantly kept adapting to the behavior of both the network and application. In some other cases, we also wrote our own SQL payloads and created our own fake HTML form when we were performing the exploitation for the CSRF.

This project has taught us that when it comes to cybersecurity, innovation does not have to be creating something new from scratch, it is more about learning how to think like an attacker, and using many different tools in addition to the knowledge that you have in a more innovative and creative way. Through this project we also learned that the security of networks and systems are essential, since otherwise important information can be compromised for both users and organizations if left unmonitored.

References

kingthorin, & zbraiterman. (2024). *SQL Injection*. OWASP. https://owasp.org/www-community/attacks/SQL_Injection

Security and privacy controls for information systems and organizations. (2020).
<https://doi.org/10.6028/nist.sp.800-53r5>

OWASP. (2025). *SQL Injection Prevention · OWASP Cheat Sheet Series*. Owasp.org;
Owasp.

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

OWASP. (n.d.). *Authorization - OWASP Cheat Sheet Series*. Cheatsheetseries.owasp.org.
https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html

OWASP. (2021). *A09 Security Logging and Monitoring Failures - OWASP Top 10:2021*.
Owasp.org. https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/

OWASP. (2013). *Insecure Direct Object Reference Prevention · OWASP Cheat Sheet Series*. Owasp.org.
https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

OWASP. (2021a). *A01 Broken Access Control - OWASP Top 10:2021*. Owasp.org;

OWASP. https://owasp.org/Top10/A01_2021-Broken_Access_Control/

WSTG - Latest / OWASP. (n.d.). Owasp.org. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/04-Testing_for_Insecure_Direct_Object_References

OWASP. (2012). *Cross-Site Request Forgery Prevention · OWASP Cheat Sheet Series*.

Owasp.org. <https://cheatsheetseries.owasp.org/cheatsheets/Cross->

[Site Request Forgery Prevention Cheat Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site Request Forgery Prevention Cheat Sheet.html)