

**Rochester Institute of Technology**



**CSEC 472 Authentication and Security Models**

**BS in Cybersecurity**

**Department of Cybersecurity**

**Lab 2: OAuth**

# Virtual Environment:

|  |  |  |  |
| --- | --- | --- | --- |
| **Hostname** | **Image** | **IP Address** | **RustDeskID** |
| ClientApp | UbuntuJammy2204-Desktop | 10.0.0.5 | 507956689 |
| AuthServer | UbuntuJammy2204-Desktop | 10.0.0.10 | 519784601 |
| OAuthProvider | UbuntuJammy2204-Desktop | 10.0.0.15 | 509006859 |
| AppServer | UbuntuJammy2204-Desktop | 10.0.0.20 | 507497424 |

**RustDesk Password:** Password1!

The OpenStack virtual environment is comprised of four VMs:

* UbuntuJammy2204 [10.0.0.5] acts as the client application (ClientApp).
  + sudo apt-get install net-tools curl
  + Apache2 v2.4.52: [httpd:80] sudo apt-get install apache2
* HTML Files: index.html file added to server root. (ref:/Assets/ClientApp/index.html)
* PHP Files: proccess.php added to server root. (ref:/Assets/ClientApp/proccess.php)
  + PHP v8.1.2: [php8.1:9000] sudo apt-get install php libapache2-mod-php
* UbuntuJammy2204 [10.0.0.10] is the authentication server (AuthServer).
  + sudo apt-get install net-tools curl
  + Apache2 v2.4.52: [httpd:80] sudo apt-get install apache2
* PHP Files: index.php added to server root. (ref:/Assets/AppServer/index.php)
  + PHP v8.1.2: [php8.1:9000] sudo apt-get install php libapache2-mod-php

UbuntuJammy2204 [10.0.0.15] is the open authentication provider (OAuthProvider).

* + sudo apt-get install net-tools, curl
  + Apache2 v2.4.52: [httpd:80] sudo apt-get install apache2
* PHP Files: auth.php added to server root. (ref:/Assets/AuthServer/auth.php)
  + PHP v8.1.2: [php8.1:9000] sudo apt-get install php libapache2-mod-php
* UbuntuJammy2204 [10.0.0.20] is the application server (AppServer).
  + sudo apt-get install net-tools curl git
  + NginX v1.18.0: [nginx:80] sudo apt-get install nginx
* git clone https:github.com/bshaffer/oauth2-server-php
* default server /etc/nginx/site-settings/default configuration file changed:

SSL port 443 enabled

PHP UNIX socket enabled

Index files changed

* PHP Files: info.php, server.php, token.php, authorize.php, resource.php added to server root; (ref:/Assets/OAuthProvider/\*) created per: <http://bshaffer.github.io/oauth2-server-php-docs/cookbook/>
* HTTPS self-signed certificates were created per: <https://www.techrepublic.com/article/how-to-enable-ssl-on-nginx/>
* The required PEM file used for communication was extracted using:

openssl x509 -in /etc/ssl/certs/nginx-selfsigned.crt -out /var/www/html/certificate.pem -outform PEM

* + MySQL v8.0.36: [mysqlserver:3306] sudo apt-get install mysql-server
* User ‘Root’ changed to password authentication ‘student’.

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY '[password]';

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION;

* Database ‘oauth’ created.
* ‘database.sql’ script run on ‘oauth’ database.
* NOTE: THE FOLLOWING COMMAND HAD TO BE RUN IN MYSQL FOR PDO.PHP TO NOT CRASH:

ALTER TABLE oauth\_authorization\_codes ADD COLUMN code\_challenge VARCHAR(1000), ADD COLUMN code\_challenge\_method VARCHAR(20);

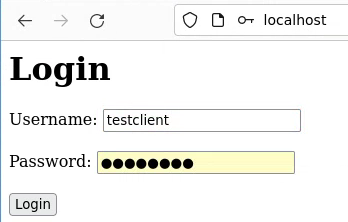
* + PHP v8.1.2: [php8.1-fpm:9000] sudo apt-get install php8.1-fpm php8.1-cli php8.1-curl php8.1-mysql
* Composer installed to PATH. <https://getcomposer.org/download/>
* Oauth2 library PHP Composer packages installed and updated.
* ‘curl’ extension enabled.

# User Credential Entry:

To be able to have a ‘testclient’ user, the following SQL command was issued on the open authentication provider MySQL Server:

INSERT INTO oauth\_clients (client\_id, client\_secret, redirect\_uri) VALUES ("testclient", "testpass", "http://fake/");

On the ClientApp index.html page those credentials (‘username’ = ‘testclient’, ‘password’ = ‘testpass’) are entered into the form and when ‘Login’ is clicked, the information is sent to the process.php script on the ClientApp.



# Client Browser → Authentication Server:

The ClientApp (process.php) encrypts the credentials with the AuthServer’s public key and sends it (POST request) to the AuthServer’s auth.php script.

The AuthServer’s auth.php receives the encrypted credentials and decrypts them with its private key.



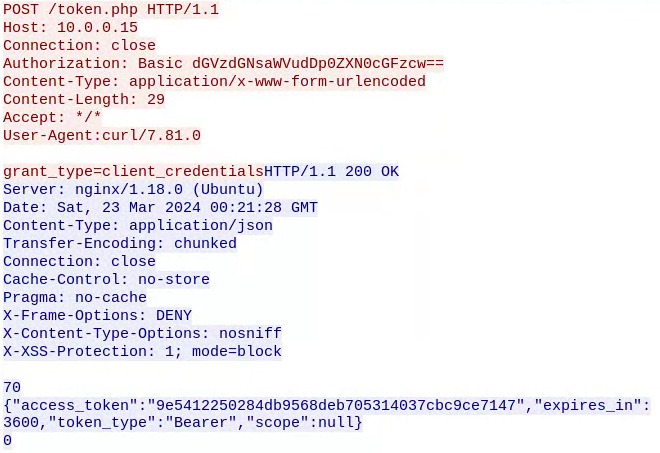
# Authentication Server → OAuth Provider:

The AuthServer’s auth.php script wraps the credentials in a JSON HTTPS POST request to the OAuthProvider’s token.php script.

The OAuthProvider receives these credentials in the format of:

curl -u testclient:testpass http://localhost/token.php -d 'grant\_type=client\_credentials'

The OAuthProvider token.php script calls functions from the server.php script to checks the received credentials against the information in its MySQL database and responds to the HTTPS POST with an appropriate JSON response.



# Invalid Credentials:

If the credentials are invalid the OAuthProvider responds with an error:

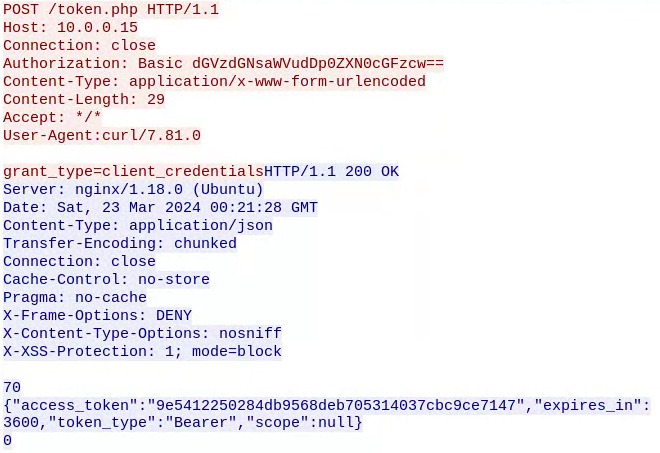
{"error":"invalid\_client","error\_description":"The client credentials are invalid"}



# Valid Credentials:

If the credentials are valid the OAuthProvider responds with a token:

{"access\_token":"03807cb390319329bdf6c777d4dfae9c0d3b3c35","expires\_in":3600,"token\_type":"bearer","scope":null}



# JSON Token Response Symmetric Encryption:

The JSON response is parsed in the auth.php script on the AuthServer. If the token is present, the token is AES256 encrypted with a SHA256 hash of the symmetric key between the AuthServer and the AppServer.

# JSON Construction:

This encrypted token is present, it is then wrapped into the JSON response: {“auth”: “success”, “token”: “<encrypted JSON token>”}. If the token is not present, the AuthServer creates the JSON response: {“auth”:”fail”, “token”:””}.

# JSON Response Hashed Encryption → Client:

The JSON response is then AES256 encrypted with the SHA256 hash of the ‘password’ credential sent by the user and sent back to the ClientApp.



# Client Browser → Application Server:

The ClientApp process.php script then takes this response and AES256 decrypts it with the ‘password’ credential. This decrypted JSON is then parsed and if the ‘auth’ tag is ‘fail’ then the user is sent back to the index.html page for credential re-entry. If the ‘auth’ tag is ‘success’ the ‘token’ tag is then extracted. This token is then base64\_encoded and inserted into the URL:

http://10.0.0.20/index.php?token=encryptedJSONtoken

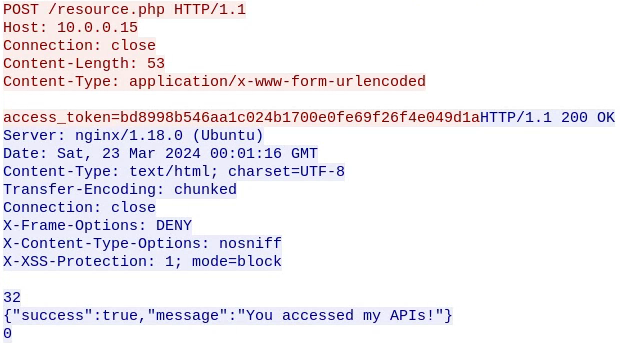
and sent to the AppServer index.php for parsing.



# Application Server Login:

The AppServer’s index.php script parses the URL and extracts the ‘token’. This token is then base64\_decoded and then AES256 decrypted with the symmetric key used between the AppServer and the AuthServer. This final token form is then sent back to the OAuthProvider’s resource.php script via HTTPS POST request for verification in the form of:

curl http://localhost/resource.php -d 'access\_token=YOUR\_TOKEN'

If the token is valid the OAuthProvider responds with the JSON: {"success":true,"message":"You accessed my APIs!"}. If the OAuthProvider responds with the JSON: {"success":false}. 

The AppServer’s index.php parses this JSON response and if the ‘success’ tag is false it errors out. If the ‘success’ tag is true then it redirects the user to the login.html page.

