**Second-Order IDOR (Whitebox)**

Now that we have covered some background information about second-order vulnerabilities, we will explore the methods for identifying, exploiting, and mitigating them within a web application, taking a whitebox approach.

**Code Review - Identifying the Vulnerability**

Before analyzing the web application's source code, let us poke at the application to see what functionality awaits us. The application seems to be a file storage application. After logging in with our test user htb-stdnt, we can see the files stored for this user:

A screenshot of a computer

Description automatically generated

Clicking the first stored file results in a request to /get\_data.php?id=2, which redirects us to a page displaying the file:

A screenshot of a computer

Description automatically generated

Since the parameter id is an obvious IDOR injection point, let us try to change the parameter to see if we can access other users' files. If we access the link /get\_data.php?id=1 in our browser, we are logged out, and an error message is displayed:

A screen shot of a login form

Description automatically generated

Thus, the web application is not vulnerable to a classical first-order IDOR vulnerability. Let us move on to analyze the source code to see if we can spot a second-order IDOR vulnerability.

When requesting a file, the backend checks our access and redirects us to either display\_data.php or error.php, depending on whether we have access to the requested file, as we can see in get\_data.php:

Code: php

<?php

session\_start();

require\_once ('db.php');

if(!$\_SESSION['user']){

header("Location: index.php");

exit;

}

$\_SESSION['id'] = $\_GET['id'];

if(check\_access($\_SESSION['id'], $\_SESSION['user'])){

header("Location: display\_data.php");

exit;

} else {

header("Location: error.php");

exit;

}

?>

If the check is successful, the file is fetched and displayed in display\_data.php:

Code: php

<?php

session\_start();

require\_once ('db.php');

if(!$\_SESSION['user']){

header("Location: index.php");

exit;

}

$user\_data = fetch\_user\_data($\_SESSION['user']);

$data = fetch\_data($\_SESSION['id']);

?>

<SNIP>

// HTML content displaying the file

Otherwise, we are logged out and redirected to index.php:

Code: php

<?php

session\_start();

session\_unset();

$\_SESSION['msg'] = 'Something went wrong. You have been logged out for security reasons.';

header("Location: index.php");

exit;

?>

As we can see, the session variable id is set to the file ID we provide in the GET request to get\_data.php. If the access check succeeds, the file is retrieved based on the session variable id. However, if the access check fails, the session variable is only cleared after redirecting to error.php via the call to the session\_unset function. Thus, the session variable id remains set to the file ID we provide to the get\_data.php endpoint until we access the error.php endpoint. This enables us to access any file ID we want by not following the redirect to error.php and instead accessing display\_data.php directly after setting any file ID via the get\_data.php endpoint.

**Running the Application Locally**

As always, in a whitebox penetration test, we will verify our exploit plan by confirming it on a locally running version of the web application. The code contains a sql file which contains the required tables as well as some seed data:

Code: sql

CREATE TABLE `data` (

`id` int(11) NOT NULL,

`owner` varchar(256) NOT NULL,

`data` varchar(10000) NOT NULL,

`name` varchar(256) NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `users` (

`id` int(11) NOT NULL,

`username` varchar(256) NOT NULL,

`description` varchar(256) NOT NULL,

`password` longtext NOT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

# htb-stdnt:Academy\_student!

INSERT INTO `users` (`id`, `username`, `description`, `password`) VALUES

(2, 'htb-stdnt', 'This is the user for HackTheBox Academy students.', '$2a$12$f4QYLeB2WH/H1GA/v3M0I.MkOqaDAkCj8vK4oHCvI3xxu7jNhjlJ.');

INSERT INTO `data` (`id`, `owner`, `data`, `name`) VALUES

(1, 'admin', "<SNIP>", 'Secret Apple Pie Recipe');

INSERT INTO `data` (`id`, `owner`, `data`, `name`) VALUES

(2, 'htb-stdnt', '<SNIP>', 'Lorem Ipsum');

We can then start a MySQL docker container that initializes the database from the provided db.sql file:

yovecio@htb[/htb]$ docker run -p 3306:3306 -e MYSQL\_USER='db' -e MYSQL\_PASSWORD='db-password' -e MYSQL\_DATABASE='db' -e MYSQL\_ROOT\_PASSWORD='db' --mount type=bind,source="$(pwd)/db.sql",target=/docker-entrypoint-initdb.d/db.sql mysql

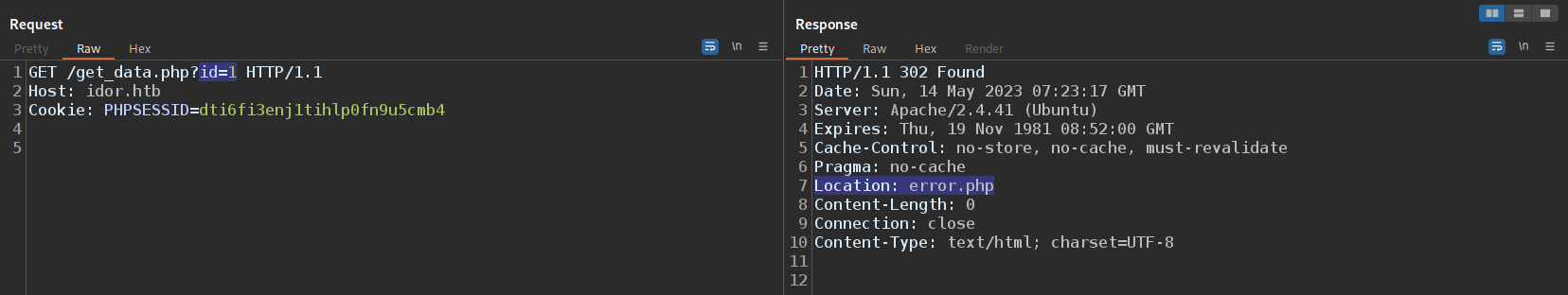
Afterward, we can use PHP's built-in web server to run the application:

yovecio@htb[/htb]$ php -S 127.0.0.1:8000

[Sun May 14 11:48:02 2023] PHP 7.4.33 Development Server (http://127.0.0.1:8000) started

**Exploitation**

To exploit the second-order IDOR vulnerability, we need to force the web application to set the session variable id to a different user's file such that we can display it. We can do so by supplying an arbitrary ID to the get\_data.php endpoint:



As long as we do not follow the redirect to error.php, the session variable id remains set. Thus, we can now navigate to /display\_data.php in our web browser to display the file, which is the admin user's secret apple pie recipe:

A screenshot of a recipe

Description automatically generated

This proof-of-concept allows us to write a small script that exfiltrates all existing files on the web application.

**Patching**

To patch the vulnerability, we need to ensure the access is checked before the file can be accessed. In this web application, the file can be accessed as soon as the session variable id is set. Thus, we must ensure that this session variable is only set after the access has been checked. Thus, we can fix the vulnerability by changing the code in get\_data.php:

Code: php

<?php

session\_start();

require\_once ('db.php');

if(!$\_SESSION['user']){

header("Location: index.php");

exit;

}

if(check\_access($\_GET['id'], $\_SESSION['user'])){

$\_SESSION['id'] = $\_GET['id'];

header("Location: display\_data.php");

exit;

} else {

header("Location: error.php");

exit;

}

?>