UniSENAI

Módulo 6: Autenticação e Segurança

Módulo 6: Autenticação e Segurança

Objetivo:

Garantir a segurança nas aplicações web, protegendo os dados dos usuários por meio de boas práticas de desenvolvimento, autenticação segura e prevenção contra vulnerabilidades comuns.

Conteúdo Detalhado:

Aula 6.1: Introdução à autenticação com JWT (JSON Web Token)

O que é JWT?

- **Definição:** Um padrão aberto para transmissão segura de informações entre partes como um objeto JSON compactado.
- **Composição:**
    1. **Header:** Define o tipo de token e o algoritmo de assinatura.
    2. **Payload:** Contém as informações (claims) codificadas.
    3. **Signature:** Garante a integridade dos dados.

Como funciona?

1. O cliente envia credenciais (ex.: login e senha).
2. O servidor valida e emite um token JWT.
3. O cliente armazena o token e o envia em cada requisição subsequente.

Exemplo de Uso:

```
const jwt = require('jsonwebtoken');

const token = jwt.sign({ id: 1, role: 'admin' }, 'secret_key', { expiresIn: '1h' });
console.log(token);

const decoded = jwt.verify(token, 'secret_key');
console.log(decoded);
```

Aula 6.2: Gerenciamento de Sessões e Cookies

Sessões

- Armazenam dados temporários relacionados ao usuário no servidor.

- Usado para persistir estados entre requisições HTTP.

Cookies

- Pequenos arquivos armazenados no navegador.
- Contêm informações como tokens de autenticação.

Exemplo de Uso no Express:

```
const session = require('express-session');
const cookieParser = require('cookie-parser');

app.use(cookieParser());
app.use(session({
  secret: 'chave_secreta',
  resave: false,
  saveUninitialized: true
}));
```

Dicas de Segurança:

- Configure o atributo `HttpOnly` para evitar acessos via JavaScript.
- Use `Secure` para transmitir cookies apenas via HTTPS.

Aula 6.3: Proteção contra vulnerabilidades comuns (CORS, CSRF, SQL Injection)

CORS (Cross-Origin Resource Sharing)

- **Definição:** Um mecanismo que controla como recursos de um servidor podem ser acessados a partir de domínios diferentes.
- **Problema Resolvido:** Bloqueia requisições não autorizadas de origens externas.
- **Configuração no Express:**

```
const cors = require('cors');

app.use(cors({
  origin: 'https://dominio-autorizado.com',
  methods: ['GET', 'POST']
}));
```

CSRF (Cross-Site Request Forgery)

- **Definição:** Ataque onde comandos maliciosos são enviados de um usuário autenticado para um servidor.
- **Solução:**
  - Utilize tokens CSRF que são validados pelo servidor.
  - Exemplo no Express:

```
const csrf = require('csurf');

app.use(csrf());

app.get('/form', (req, res) => {
  res.render('form', { csrfToken: req.csrfToken() });
});
```

SQL Injection

- **Definição:** Ataque que explora falhas nas consultas SQL, permitindo a injeção de comandos maliciosos.
- **Exemplo de Vulnerabilidade:**

```
SELECT * FROM usuarios WHERE nome = 'admin' OR '1'='1';
```

- **Solução:**
  - Use prepared statements:

```
connection.query('SELECT * FROM usuarios WHERE nome = ?', [nome],
(err, results) => {
  if (err) throw err;
  console.log(results);
});
```

`

Aula 6.4: Implementando autenticação baseada em roles com MySQL

O que é Autenticação Baseada em Roles?

- Permite restringir ações com base nos papéis (roles) dos usuários, como `admin`, `editor` e `usuário comum`.

Exemplo de Estrutura de Tabelas:

```sql
CREATE TABLE usuarios (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nome VARCHAR(255),
  email VARCHAR(255),
  senha VARCHAR(255),
  role VARCHAR(50)
);
```

Middleware de Autorização:

```javascript
function authorize(role) {
  return (req, res, next) => {
    if (req.user.role !== role) {
      return res.status(403).send('Acesso negado');
    }
    next();
  };
}

app.get('/admin', authorize('admin'), (req, res) => {
  res.send('Bem-vindo, Admin!');
});
```

Lista de Exercícios

Questões Teóricas

1. Explique o que é JWT e quais são suas vantagens.
2. Qual a diferença entre sessões e cookies?
3. Defina CORS e por que é importante configurá-lo corretamente.
4. O que é CSRF e como prevenir este tipo de ataque?
5. Explique o conceito de SQL Injection e seus riscos.
6. Como o atributo `HttpOnly` ajuda a proteger cookies?
7. Quais são as principais partes de um token JWT?
8. Liste três boas práticas para evitar SQL Injection em aplicações Node.js.
9. O que é autenticação baseada em roles e quais são seus benefícios?
10. Como o middleware `csrf` auxilia na proteção de aplicações web?

Questões Práticas

1. Configure um servidor Express para gerar e validar tokens JWT.
2. Implemente um sistema de sessão que armazene informações temporárias de um usuário.
3. Configure o CORS para permitir apenas requisições de um domínio específico.
4. Crie uma rota protegida que utilize tokens CSRF para validação.
5. Desenvolva uma função que previna SQL Injection utilizando prepared statements.
6. Configure cookies seguros com os atributos `HttpOnly` e `Secure`.
7. Crie uma tabela de usuários com roles e insira dados fictícios.
8. Implemente middleware para restringir acesso com base em roles.
9. Simule um ataque CSRF e implemente a solução para preveni-lo.
10. Crie um sistema de login simples que emita um JWT e proteja rotas baseadas no token.

Instruções para a Entrega das Atividades

1. **Elaboração e Envio do Arquivo**
   - Responda todas as questões de forma clara e objetiva.
   - Gere um arquivo no formato **.PDF** contendo as respostas de cada questão.
   - Envie o arquivo para os e-mails dos professores responsáveis.
2. **Validação da Atividade**
   - Após o envio do arquivo, procure o(s) professor(es) para realizar a validação da atividade.
   - **Não inicie a próxima atividade sem antes validar a anterior com o professor.**
3. **Forma de Validação**
   - **Explicação Verbal**: Explique cada resposta verbalmente ao(s) professor(es).
   - **Perguntas e Respostas**: Esteja preparado para responder aos questionamentos do(s) professor(es) sobre o conteúdo das respostas.
   - **Orientação**: Receba orientações sobre a apresentação do(s) tema(s).