

 Estácio	<p align="center"> UNIVERSIDADE ESTÁCIO DE SÁ POLO DALPLAZA CENTER – SÃO LUÍS/MA DESENVOLVIMENTO FULL STACK - 22.3 Relatório da Missão Prática Nível 5 Mundo 5 </p>
Aluno:	Lucas Silva Costa
Professor:	Robson Lorbieski
Repositório:	https://github.com/LutchasDev/M5-nivel5-main

Título da Prática: Software sem segurança não serve

Objetivos da Prática:

- Descrever o controle básico de acesso a uma API Rest;
- Descrever o tratamento de dados sensíveis e log de erros com foco em segurança;
- Descrever a prevenção de ataques de acesso não autorizado com base em tokens desprotegidos/desatualizados;
- Descrever o tratamento de SQL Injection em códigos-fonte; Descrever o tratamento de CRLF Injection em códigos-fonte;
- Descrever a prevenção a ataques do tipo CSRF em sistemas web;

Contextualização

O time de segurança da Software House, onde você atua como Especialista em Desenvolvimento de Software, identificou uma falha de segurança, explorada por ataques que geraram o vazamento de dados, além de outros problemas, em uma das aplicações legadas, desenvolvida há alguns anos atrás. Tal falha consiste na concessão de acesso não autorizado de recursos a usuários. O cenário completo é descrito a seguir:

A aplicação web possui um frontend e um backend, sendo esse último uma API Rest. O padrão geral da estrutura de URLs (e URI) da aplicação é:

- `http://dominio.com/nome-do-recurso/{session-id}`
- `http://dominio.com/nome-do-recurso/{id}/{session-id}`

O padrão acima é usado tanto no frontend, no navegador, como no backend, nos endpoints.

Após uma simples análise, foi identificado que o valor do parâmetro "session-id" é obtido com a encriptação do id do usuário logado no sistema, usando um processo suscetível a falhas, uma vez que um dos principais dados necessários no processo de criptografia é o próprio nome da empresa detentora do software.

Logo, tal falha é passível de ser explorada via ataques de força bruta para descoberta do padrão usado na geração da "session-id" e consequente geração de valores aleatórios que serão usados para a realização de requisições – como solicitações de dados e também criação e atualização – na aplicação, até a obtenção do acesso indevido.

Além do problema já relatado, o time de segurança descobriu que, atualmente, não é realizado nenhum tratamento no processamento dos parâmetros trafegados na aplicação. Logo, também é possível explorar outras falhas, como as de "Injection" de códigos maliciosos.

Resultados:

```
Arquivo  Editar  Seleção  Ver  ...  NV5MD5

JS nv5md5.js 8 X
Missão Prática > JS nv5md5.js > ...
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const jwt = require('jsonwebtoken');
4  const app = express();
5
6
7  app.use(bodyParser.json());
8
9
10 const port = process.env.PORT || 3000;
11 const secretKey = 'chave-secretEmpresa';
12
13
14 const user = [
15   {username: 'user', password: '123456', id: '635', email: 'user@dominio.com', perfil: 'user'},
16   {username: 'colab', password: '12345678', id: '001', email: 'colab@dominio.com', perfil: 'user'},
17   {username: 'admin', password: '0123456789', id: '435', email: 'admin@dominio.com', perfil: 'admin'},
18 ]
19
```

```
Arquivo  Editar  Seleção  Ver  ...  NV5MD5

JS nv5md5.js 8 X
Missão Prática > JS nv5md5.js > ...
20 function authenticateToken(req, res, next){
21   const token = req.header("Autorização")?.split(' ')[1];
22   if (!token) return res.sendStatus(401);
23
24   jwt.verify(token, secretKey, (err, user) => {
25     if (err) return res.status(403);
26     req.user = user;
27     next();
28   });
29 }
30
31 Tabnine | Edit | Test | Explain | Document | Ask
32 function authorizeAdmin(req, res, next) => {
33   if (req.user.perfil !== 'admin') return res.status(403).json({message: 'Acesso Proibido!'});
34   next();
35 }
36
37 app.post('/api/auth/login', (req, res) => {
38   const {username, password} = req.body;
39   const user = users.find(u => u.username === username && u.password === password);
40
41   if (!user) return res.status(400).json({message: 'Credencial inválido, tente novamente'});
42
43   const token = jwt.sign({id: user.id, perfil: user.perfil}, secretKey, {expiresIn: 'Expira em 2 horas'});
44   res.json({id: user.id, email: user.email, perfil: user.perfil});
45 }
```

```
Arquivo  Editar  Seleção  Ver  ...  NV5MDS

JS nv5md5.js 8 x
Missão Prática > JS nv5md5.js > authenticateToken

45
46   Tabnine | Edit | Test | Explain | Document | Ask
47   app.get('/api/users', authenticateToken, authorizeAdmin, (req,res)=>{
48     res.json(users);
49   });

50   Tabnine | Edit | Test | Explain | Document | Ask
51   app.get('/api/contracts/:empresa:inicio', authenticateToken, authorizeAdmin, (req,res)=> {
52     const empresa = req.params.empresa;
53     const inicio = req.params.inicio;
54     if(!/^[\w\s]+$/.test(empresa) || !/^d{4}-d{2}-d{2}$/.test(inicio)){
55       return res.status(400).json({message: 'Parâmetros inválidos'});
56     }
57     const contracts = getContrats(empresa, inicio);
58     res.json(contracts.length ? contracts : {message: 'Informação não encontrada.'});
59   });
60
61   Tabnine | Edit | Test | Fix | Explain | Document | Ask
62   function getContrats(empresa, inicio);
63   return[
64     {empresa, inicio, contratoId: "0001", descricao: "Contrato De Empresa I"},
65     {empresa, inicio, contratoId: "0002", descricao: "Contrato de Empresa II"},
66   ];
67
68
```