Missão Prática RPG0035 - Software sem Segurança Não Serve!

Seu N0ome: Felipe de Souza Komatsu

Data: 9 de Junho de 2025

1. Introdução

Este documento detalha a resolução da Missão Prática RPG0035 - "Software sem Segurança Não Serve!", que envolve a análise e refatoração de uma aplicação web legada. Atuando como desenvolvedor especialista, o objetivo principal foi identificar vulnerabilidades de segurança críticas, como acesso não autorizado e falhas de injeção, e implementar medidas corretivas para assegurar o funcionamento seguro e correto da aplicação. A refatoração focou em modernizar e fortalecer os mecanismos de autenticação, controle de acesso e tratamento de dados, alinhando-se com as melhores práticas de segurança de software.

2. Análise das Vulnerabilidades Identificadas na Aplicação Legada

A investigação da aplicação web legada revelou diversas falhas de segurança significativas que, se exploradas, poderiam levar a vazamento de dados e outros problemas sérios. As principais vulnerabilidades identificadas são descritas a seguir:

- **Vulnerabilidade na Geração e Tráfego do `session-id`:**
- **Mecanismo Falho:** O `session-id` era gerado através da criptografia do ID do usuário logado. O grande problema era que a chave de criptografia utilizada era o nome da empresa (`'Software House '`), um segredo facilmente adivinhável. Isso tornava o `session-id` altamente suscetível a ataques de força bruta e engenharia reversa.

- **Tráfego Inseguro:** O `session-id` era trafegado diretamente na URI (ex: `http://dominio.com/nome-do-recurso/{session-id}`). Essa prática

expunha o identificador da sessão, facilitando sua captura e manipulação por atacantes.

- **Consequência:** A previsibilidade do `session-id` permitia que invasores, ao forjarem valores aleatórios e válidos, acessassem recursos protegidos, simplesmente manipulando o ID do usuário.

    - **Controle de Acesso Insuficiente e Acesso Não Autorizado:**
    - O endpoint `/api/users/:sessionid` permitia a recuperação dos dados de todos os usuários. Embora houvesse uma tentativa de controle de acesso baseado em perfil (`admin`), a vulnerabilidade do `session-id` permitia que um atacante, ao forjar um `session-id` de administrador, burlasse essa proteção.
    - O endpoint `/api/contracts/:empresa/:inicio/:sessionid` não possuía **nenhum controle de acesso** baseado em perfil, permitindo que qualquer usuário autenticado (mesmo com perfis de menor privilégio) acessasse dados sensíveis de contratos.

    - **Vulnerabilidades de "Injection" (SQL Injection):**
    - O método `getContracts` no backend construía a consulta SQL concatenando diretamente os parâmetros (`empresa` e `data_inicio`) recebidos na requisição. Isso abria uma porta para **SQL Injection**, onde um atacante poderia injetar comandos SQL maliciosos, como `' OR '1'='1--`, para retornar todos os registros do banco de dados ou até mesmo manipular/excluir informações.

    - **Exposição de Endpoint de Decriptografia:**
    - A existência do endpoint `/api/auth/decrypt/:sessionid` era uma falha gravíssima. Ele permitia que qualquer pessoa com um `session-id` válido testasse o processo de decriptografia, facilitando a engenharia reversa da chave secreta e, consequentemente, a falsificação de `session-ids`.

3. Soluções de Segurança Implementadas (Refatoração)

Para remediar as vulnerabilidades identificadas, a aplicação foi refatorada com as seguintes medidas de segurança:

- **Autenticação Robusta com JWT (JSON Web Tokens):**
- **Substituição do `session-id`:** O mecanismo de `session-id` criptografado e vulnerável foi completamente substituído pela utilização de **JSON Web Tokens (JWTs)**.

- **Geração do Token:** O token JWT é gerado no endpoint de login (`/api/auth/login`). Seu **payload** agora inclui informações essenciais do usuário (ID, username e perfil). Crucialmente, foi adicionado o parâmetro `exp` (expiration time), garantindo que o token tenha um tempo de vida limitado (`1h`), aumentando a segurança contra reuso de tokens antigos. A **chave secreta** utilizada para assinar o token (`JWT_SECRET`) foi definida para ser carregada de variáveis de ambiente, reforçando a segurança em produção.

- **Tráfego do Token:** O token agora é trafegado de forma segura no **cabeçalho `Authorization`**, utilizando o padrão `Bearer` (ex: `Authorization: Bearer <seu_token>`), e não mais na URI, protegendo-o de loggers e proxies.

- **Middleware `authenticateToken`:** Um middleware global foi implementado para centralizar a **validação de cada token JWT** recebido. Ele verifica a assinatura do token e se ele não está expirado. Em caso de token ausente (401) ou inválido/expirado (403), respostas claras e genéricas são retornadas ao cliente, sem expor detalhes internos.

- **Controle de Acesso Baseado em Papéis (RBAC - Role-Based Access Control):**
- **Middleware `authorizeAdmin`:** Criou-se um middleware específico (`authorizeAdmin`) que verifica o `perfil` do usuário (extraído do payload do token JWT) para autorizar ou negar o acesso a recursos.

- **Aplicação em Endpoints:** Os endpoints sensíveis, como `/api/users` (para listar todos os usuários) e `/api/contracts` (para listar contratos), foram protegidos por este middleware, permitindo acesso apenas a usuários com perfil `admin`.

- **Novo Endpoint `/api/me`:** Para atender à necessidade de acesso a dados do próprio usuário, um novo endpoint (`GET /api/me`) foi criado. Este endpoint é protegido apenas por `authenticateToken`, permitindo que **qualquer usuário autenticado** (independentemente do perfil) acesse seus próprios dados (sem a senha).

- **Respostas de Autorização:** Em caso de acesso negado por falta de privilégio, uma mensagem genérica de "Acesso negado. Requer privilégios de administrador." é retornada com status 403 Forbidden.

- **Prevenção de SQL Injection:**
- **`getContracts` e `Repository`:** A função `getContracts`, responsável por buscar contratos, foi refatorada. A construção da query SQL foi alterada para utilizar **consultas parametrizadas** (também conhecidas como *prepared statements*).

- **Mecanismo:** Em vez de concatenar diretamente os parâmetros (`empresa` e `inicio`) na string da query, são usados **placeholders** (ex: `$1`, `$2`). Os valores dos parâmetros são passados separadamente para o driver do banco de dados (simulado pela classe `Repository`), que garante que esses valores sejam tratados como dados e não como parte do código SQL. Isso neutraliza qualquer tentativa de injeção de comandos SQL maliciosos.

- **Hashing Seguro de Senhas:**
- As senhas no mock de dados (`users` array) foram substituídas por seus **hashes gerados com `bcryptjs`** durante a inicialização da aplicação.

- O processo de login agora utiliza `bcrypt.compare` para verificar a senha fornecida pelo usuário contra o hash armazenado. Isso garante que as senhas nunca sejam armazenadas ou comparadas em texto claro, protegendo-as contra vazamentos e ataques de força bruta/dicionário.

- **Validação de Entrada (`express-validator`):**
- Foi implementada uma validação robusta para os dados de entrada em endpoints como `/api/auth/login` e `/api/contracts`. Utilizando a biblioteca `express-validator`, é possível verificar se campos obrigatórios estão preenchidos e se os dados estão no formato esperado. Isso ajuda a prevenir uma gama de ataques baseados em dados malformados ou ausentes.

- **Remoção de Endpoints Vulneráveis:**
- O endpoint `/api/auth/decrypt/:sessionid`, que permitia a quebra da criptografia do `session-id` original, foi **removido completamente** do código, eliminando essa grave falha de exposição de informações.

4. Testes e Validação das Medidas de Segurança

Os testes foram realizados utilizando ferramentas como **Postman** (ou `curl` no terminal) para simular as requisições HTTP e verificar o comportamento da API diante das novas medidas de segurança.

- **Ferramentas de Teste Recomendadas:** Postman, Insomnia ou `curl`.
- **Cenários de Teste e Resultados Esperados:**

4.1. Login de Usuário (`POST /api/auth/login`)

Objetivo: Obter um token JWT para acesso posterior.

Requisição de Sucesso (Usuário `user`):

- **URL:** `http://localhost:3000/api/auth/login`
- **Método:** POST
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```
{
    "username": "user",
    "password": "123456"
}
```

- **Resultado Esperado (Status 200 OK):** Um token JWT será retornado.

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

- **Requisição de Sucesso (Usuário `admin`):**
- **URL:** `http://localhost:3000/api/auth/login`
- **Método:** POST
- **Headers:** `Content-Type: application/json`
- **Body (raw JSON):**

```
{
    "username": "admin",
    "password": "123456789"
}
```

- **Resultado Esperado (Status 200 OK):** Um token JWT de administrador.

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
```

```
}
```

- **Requisição de Falha (Credenciais Inválidas):**
- **URL:** http://localhost:3000/api/auth/login
- **Método:** POST
- **Headers:** Content-Type: application/json
- **Body (raw JSON):**

```
{
    "username": "usuarioinexistente",
    "password": "senhaerrada"
}
```

- **Resultado Esperado (Status 401 Unauthorized):** Mensagem de erro genérica.
  - ```
    {
        "message": "Credenciais inválidas"
    }
    ```
    Requisição de Falha (Validação de Entrada - Campos Vazios):
  - **URL:** http://localhost:3000/api/auth/login
  - **Método:** POST
  - **Headers:** Content-Type: application/json
  - **Body (raw JSON):**

```
{
    "username": "",
    "password": ""
}
```

- **Resultado Esperado (Status 400 Bad Request):** Erros de validação.

```
{
    "errors": [
        { "msg": "Username é obrigatório", "param": "username",
"location": "body" },
        { "msg": "Password é obrigatório", "param": "password",
"location": "body" }
    ]
}
```

4.2. Dados do Usuário Logado (GET /api/me)

- **Objetivo:** Acessar os próprios dados do usuário autenticado.
```

- **Requisição de Sucesso (Com Token Válido):**
- **URL:** http://localhost:3000/api/me
- **Método:** GET
- **Headers:** Authorization: Bearer <TOKEN_DE_USUARIO_VALIDO>
- **Resultado Esperado (Status 200 OK):** Dados do usuário (sem a senha).

```
{
    "username": "user",
    "id": 123,
    "email": "user@dominio.com",
    "perfil": "user"
}
```

- Requisição Sem Token:
- URL: http://localhost:3000/api/me
- Método: GET
- Headers: (Nenhum cabeçalho Authorization)

- **- Resultado Esperado (Status 401 Unauthorized):**

```
{
    "message": "Token de autenticação ausente."
}
```

- **Requisição com Token Inválido/Expirado:**
- **URL:** http://localhost:3000/api/me
- **Método:** GET
- **Headers:** Authorization: Bearer <TOKEN_EXPIRADO_OU_INVALIDO>
- **Resultado Esperado (Status 403 Forbidden):**

```
{
    "message": "Token inválido ou expirado."
}
```

4.3. Listar Todos os Usuários (GET /api/users)

- Objetivo: Acessar a lista de todos os usuários (privilégio de administrador).

- Requisição de Sucesso (Com Token de admin Válido):

- **URL:** http://localhost:3000/api/users
- **Método:** GET
- **Headers:** Authorization: Bearer <TOKEN_DE_ADMIN_VALIDO>

- **Resultado Esperado (Status 200 OK):** Lista de usuários (sem senhas).

```
[
    {"username": "user", "id": 123, "email": "user@dominio.com",
"perfil": "user"},
    {"username": "admin", "id": 124, "email": "admin@dominio.com",
"perfil": "admin"},
    {"username": "colab", "id": 125, "email": "colab@dominio.com",
"perfil": "user"}
]
```

- **Requisição com Token de Perfil user ou colab:**
  - **URL:** http://localhost:3000/api/users
  - **Método:** GET
  - **Headers:** Authorization: Bearer <TOKEN_DE_USER_OU_COLAB>
  - **Resultado Esperado (Status 403 Forbidden):**

```
{
    "message": "Acesso negado. Requer privilégios de administrador."
}
```

- **Requisição Sem Token / Com Token Inválido:** (Mesmos resultados de 401/403 do /api/me).

4.4. Listar Contratos (GET /api/contracts/:empresa/:inicio)

- **Objetivo:** Buscar contratos filtrando por empresa e data de início.

- **Requisição de Sucesso (Com Token de admin e Parâmetros Válidos):**

  - **URL:** http://localhost:3000/api/contracts/Empresa%20A/2023-01-01
  - **Método:** GET**Headers:** Authorization: Bearer <TOKEN_DE_ADMIN_VALIDO>
  - **Resultado Esperado (Status 200 OK):** Lista de contratos correspondentes.

```
[
    { "id": 1, "empresa": "Empresa A", "data_inicio": "2023-01-01",
```

```
"valor": 10000 }
]
```

- **Requisição de Falha (Contratos Não Encontrados):**
- **URL:** http://localhost:3000/api/contracts/Empresa%20X/2025-01-01
- **Método:** GET
- **Headers:** Authorization: Bearer <TOKEN_DE_ADMIN_VALIDO>
- **Resultado Esperado (Status 404 Not Found):**

```
{
    "message": "Nenhum contrato encontrado para os critérios
fornecidos."
}
```

- **Requisição com Tentativa de SQL Injection (Com Token de admin):**
- **URL:** http://localhost:3000/api/contracts/Empresa%20A/'%20OR%20'1'='1--/2023-01-01
- **Método:** GET
- **Headers:** Authorization: Bearer <TOKEN_DE_ADMIN_VALIDO>
- **Resultado Esperado (Status 404 Not Found):** A injeção é **neutralizada**. O servidor interpreta `' OR '1'='1--` como parte do valor do parâmetro `empresa` e não como código SQL, por isso, nenhum contrato será encontrado ou a requisição falhará de forma segura.

```
{
    "message": "Nenhum contrato encontrado para os critérios
fornecidos."
}
```

- **Requisição com Token de Perfil user ou colab:** (Mesmos resultados de 403 Forbidden do `/api/users`).
- **Requisição Sem Token / Com Token Inválido:** (Mesmos resultados de 401/403 do `/api/me`).

5. Conclusão

A refatoração da aplicação legada demonstrou a importância e a eficácia das medidas de segurança implementadas. Através da substituição de mecanismos vulneráveis por soluções modernas e robustas como JWT para autenticação, hashing de senhas com bcrypt, controle de acesso baseado em perfis e o uso de consultas parametrizadas para prevenção de SQL Injection, a aplicação agora possui uma camada de segurança significativamente mais resiliente.

Este trabalho não apenas corrigiu falhas críticas, mas também reforçou o conhecimento sobre as melhores práticas de desenvolvimento seguro, essenciais para a proteção de dados e a integridade de sistemas em um cenário de ameaças cibernéticas em constante evolução.