

Missão Prática | Nível 5 | Mundo 5

Material de **orientações** para desenvolvimento da **missão prática** do **2º nível de conhecimento**.

📌 Conforme a metodologia gamificada propõe, a dificuldade das missões aumenta a cada nível, então o Nível 5 é o mais complexo do Mundo atual. Para melhorar a experiência do aluno, orientamos a desenvolver esse nível envolvendo empresas parceiras, colegas ou interações externas, para desenvolver o contato externo e praticar a presencialidade em suas atividades.

RPG0035 - 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;

Entrega e Progresso

- As microatividades irão dar suporte para o desenvolvimento da Missão Prática. Elas têm apoio/gabarito para resolução no próprio documento;
- A entrega esperada é a Missão Prática, descrita neste documento após as Microatividades;
- A missão prática progride 5% na entrega e até 5% dependendo da nota atribuída pelo tutor em sua correção.

👉 Atividades práticas

Software sem segurança não serve

Contextualização

Ao longo das microatividades a seguir, você atuará como o desenvolvedor responsável pelo tratamento de segurança e proteção de vulnerabilidades ao ecossistema de softwares, escritos em diferentes linguagens de programação, de uma Software House. Para isso, deverá analisar fragmentos de código, entender as vulnerabilidades neles contidas e realizar os devidos tratamentos, garantindo, dessa forma, a saúde de todo o ecossistema em questão.

- Material necessário para a prática

- Editor ou IDE para escrita de códigos-fonte.

- Procedimentos

Analise o fragmento de código abaixo:

```
app.get('/confidential-data', (req, res) => {  
  
  //executa um serviço fictício para obter os dados a serem retornados  
  
  jsonData = service.call(req)  
  
  
  //retorna os dados  
  
  res.json(jsonData)  
  
});
```

1. No código é apresentado um exemplo, não totalmente funcional, do endpoint de uma API Rest, onde temos o retorno a uma chamada realizada via método HTTP GET. Tal código está escrito em Javascript, com o framework express;
2. Repare que a requisição é processada e os dados são retornados sem nenhum tipo de autenticação ou controle de acesso;
3. A partir do fragmento de código fornecido e da contextualização acima, você deverá:
 - a. Copiar o código em uma IDE ou editor – ou então escrever um novo código ou adaptar um existente, na linguagem de sua preferência ou com pseudocódigo, que processe uma requisição HTTP, como no exemplo fornecido;
 - b. Incluir um tratamento de segurança, baseado em autenticação, para validar o acesso ao recurso em questão antes de o executar (antes de realizar a chamada ao “service”);
 - c. Em caso de acesso não autorizado, você deverá retornar uma mensagem genérica e alterar o status da resposta HTTP para 401 (Não autorizado);
 - d. Caso a autenticação seja realizada com sucesso, você deverá enviar a resposta da requisição.

- Resultados esperados ✨

O resultado esperado dessa microatividade é verificar se o aluno possui conhecimentos relacionados ao controle de acesso a recursos de software – no caso em questão, aos endpoints de uma API Rest - sendo capaz de aplicar a proteção necessária a fim de garantir o acesso seguro ao recurso em questão.

Microatividade 2: Descrever o tratamento de erros com foco em segurança

- Material necessário para a prática

- Editor ou IDE para escrita de códigos-fonte.

- Procedimentos:

Copie e cole, no Editor ou IDE, o fragmento de código abaixo:

```
password = new_password  
username = new_username
```

```
#Verifica se o nome de usuário já está em uso  
IF USER_EXISTS(username) THEN  
RETURN Error("Já existe usuário com esse nome.")  
ENDIF
```

```
#Limita a senha a apenas caracteres numéricos  
IF NOT IS_NUMERIC(password) THEN  
RETURN Error("Senha inválida")  
ENDIF
```

```
#Verifica as credenciais, sem limite de tentativas
```

#Verifica a senha e o usuário, informando qual deles, quando for o caso, não é válido

```
IS_VALID_PASSWORD=LOOKUP_CREDENTIALS_IN_DATABASE(username, password)
```

```
IF NOT IS_VALID_PASSWORD THEN
```

```
RETURN Error("Senha Inválida!")
```

```
ENDIF
```

```
IS_VALID_USERNAME=LOOKUP_CREDENTIALS_IN_DATABASE(username, password)
```

```
IF NOT IS_VALID_USERNAME THEN
```

```
RETURN Error("Usuário Inválido!")
```

```
ENDIF
```

1. Reescreva o fragmento de código acima, aplicando as seguintes tratativas de segurança:
 - a. Defina uma quantidade mínima de caracteres para o campo “password”;
 - b. Verifique se a quantidade mínima de caracteres foi respeitada para o campo “password”;
 - c. Permita que o usuário utilize qualquer caractere como senha e não apenas caracteres numéricos;
 - d. Limite a quantidade de tentativas inválidas de login;
 - e. Crie uma mensagem de erro genérica para o caso de credenciais inválidas. Ex: “Usuário ou senha incorretos”;

- Resultados esperados ✨

O resultado esperado dessa microatividade é fornecer ao aluno o conhecimento básico sobre algumas boas práticas de segurança, conforme literatura relacionada, pertinente ao tratamento de dados sensíveis no processo de login em uma aplicação.

Microatividade 3: Descrever a prevenção de acesso não autorizado com base em tokens desprotegidos

- Material necessário para a prática

- Editor ou IDE para escrita de códigos-fonte.

- Procedimentos

Copie e cole, no Editor ou IDE, o fragmento de código abaixo:

```
//Backend (JS + pseudocódigo)

function do_Login(){

    jwt_token = BASE64URL(UTF8(JWS Protected Header)) + '.' +
        BASE64URL(JWS Payload) + '.' +
        BASE64URL(JWS Signature)

    return jwt_token

}

function do_SomeAction(jwt){

    //Executar alguma ação

}

//Frontend (JS + pseudocódigo)

function login(){

    let _data = {

        username: username,

        password: password

    }

    fetch('https://dominio.com/auth', {

        method: "POST",

        body: JSON.stringify(_data),

        headers: {"Content-type": "application/json; charset=UTF-8"}

    })

    .then(response => response.json())

    .then(json => {

        localStorage.setItem("token", json.jwt_token)

    });

}
```

```

        .catch(err => console.log(err));

    }

    function doAction(){

    const token = localStorage.getItem("token")

    fetch('https://dominio.com/do_SomeAction', {

        method: "POST",

        body: JSON.stringify(null),

        headers: {

            "Content-type": "application/json; charset=UTF-8",

            "Authorization": `Bearer ${token}`,

        }

    })

    .then(response => response.json())

    .then(json => {

        console.log(`response: ${json}`)

    });

    .catch(err => console.log(err));

    }

```

1. O Código acima é formado por fragmentos referentes ao backend e ao frontend. No backend há uma função responsável pelo login de um usuário e pela geração e retorno do token de acesso. Tal token é recebido pelo frontend e armazenado no localStorage, devendo ser recuperado e utilizado posteriormente a cada nova requisição ao backend;
2. A partir da IDE ou editor, reescreva o código acima (utilizando pseudocódigo, JS ou uma outra linguagem de sua preferência) incluindo os seguintes tratamentos no backend:
 - a. Na geração do token JWT, inclua no seu payload o parâmetro (claim) “exp” (expiration time) e armazene nele o date/time de expiração do token;
 - b. Na função que recebe o jwt como parâmetro, inclua uma validação do token, incluindo seu date/time de expiração. Tal tarefa poderá ser feita “manualmente” ou com auxílio de alguma biblioteca de terceiros;
 - c. Caso a validação acima resulte num token inválido, interrompa a execução da função e retorne um erro genérico (que não forneça dados da implementação ou causa raiz) para o usuário.
3. Ainda na IDE ou editor, reescreva o frontend e:
 - a. No recebimento do token, confirme que será armazenado (no localStorage) o JWT e também a data/hora de expiração do mesmo;
 - b. Na utilização do token, no momento anterior ao seu envio em requisição para o

- b. Na utilização do token, no momento anterior ao seu envio em requisição para o backend, valide se a sua data/hora de expiração ainda é válida. Caso contrário, interrompa a execução do código, não permitindo que a requisição para o backend seja realizada;
- c. Caso, no passo anterior, o token esteja expirado, você poderá redirecionar o usuário para a página de login, para que um novo token seja obtido, ou então você poderá implementar um método no backend para atualização do token e o chamar a partir do front, recebendo o token atualizado e o armazenando.

- Resultados esperados ✨

O resultado esperado dessa microatividade é verificar se o aluno possui o conhecimento básico sobre problemas comuns relacionados ao uso de tokens para controle de acesso em aplicações e se é capaz de aplicar medidas corretivas em algumas situações, conforme descrito nos procedimentos.

Microatividade 4: Descrever o tratamento dos dados e o código-fonte

- Material necessário para a prática

- Editor ou IDE para escrita de códigos-fonte.

- Procedimientos

1. Abra, na IDE ou Editor, o fragmento de código abaixo:

```
function doDBAction(id){  
  
    var query="SELECT * FROM users WHERE userID='"+ id + "'";  
  
}
```

2. A função acima é chamada através de uma requisição HTTP, onde o parâmetro “id” é passado via query, conforme exemplo a seguir:

<https://dominio.com/app/usuario?id=10>

3. Considere que um invasor realizou, para o endereço citado, a seguinte requisição:

<http://example.com/app/userView?id=' or '1'='1-->

4. A partir da requisição acima, e do fragmento de código apresentado, serão retornados todos os usuários existentes, o que representa uma falha grave de segurança chamada SQL Injection;

5. Com base no cenário exposto, você deverá refatorar o fragmento de código para:

- Tratar, usando a técnica (e também a linguagem de programação) de sua escolha, o recebimento do parâmetro a fim de impedir que uma query arbitrária seja executada;
- Caso queira, você poderá reescrever todo o fragmento de código, modificando sua lógica, mas com o cuidado de evitar que seja possível realizar um ataque de SQL Injection.

- Resultados Esperados ✨

O resultado esperado dessa microatividade é apresentar ao aluno um cenário onde é possível ser aplicado um ataque de SQL Injection, além de verificar se ele possui conhecimento sobre tal falha e se é capaz de escrever um código que trate e evite a ocorrência da vulnerabilidade em questão.

Microatividade 5: Descrever o tratamento de códigos-fonte

- Material necessário para a prática

- Editor ou IDE para escrita de códigos-fonte.

- Procedimentos

Veja o fragmento de código abaixo (escrito na linguagem PHP):


```
<?php  
  
$redirectUrl = $_GET['url'];  
  
header('Location: ' . $redirectUrl);  
  
?>
```

1. Nesse fragmento temos:

- a. Na segunda linha, a variável “redirectUrl” recebe o valor da variável “url”, proveniente de uma requisição HTTP (submetida com o método Get);
- b. Na terceira linha, com o comando “header”, é feito o redirecionamento do usuário para uma nova página, cujo endereço foi recebido na etapa anterior.

2. Agora, veja esse exemplo de requisição (que será tratado pelo fragmento de código acima):

```
http://dominio.com/index.php?url=http://sitefake-malicioso.com/%0D%0AContent-  
Length:%200%0D%0A%0D%0AHTTP/1.1%20200%20OK%0D%0A%0D%0Aevilcontent
```

3. Repare o que está sendo passado para a variável ‘url’: trata-se de um endereço para um domínio externo ao da aplicação. Além disso, repare nos caracteres CRLF (Carriage Return e Line Feed) presentes na requisição, através dos quais um novo cabeçalho HTTP é também enviado;

4. A partir da descrição do problema apresentado, você deverá:

- a. Criar um fragmento de código utilizando a linguagem de sua preferência;
- b. Implementar uma função semelhante à apresentada, onde um redirecionamento é feito a partir do valor recebido de uma variável via método GET;
- c. Utilizar um método de sua escolha (bibliotecas de terceiros, expressões regulares, funções internas da linguagem, etc.) e tratar a possível injeção de caracteres CRLF;
- d. Também pode ser interessante impedir o redirecionamento para domínios diferentes do próprio domínio da aplicação.

- Resultados esperados 🌟

O resultado esperado dessa microatividade é apresentar ao aluno um cenário onde pode ser visto um código-fonte suscetível a ataques do tipo CRLF Injection, assim como verificar se ele é capaz de escrever um código seguro, onde a possível ocorrência de tal ataque é tratada.

Através dessa atividade o aluno analisará uma falha de segurança, em uma aplicação web, e aplicará as medidas corretivas necessárias para garantir o seu correto e seguro funcionamento.

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.

Frente ao exposto, seu trabalho consistirá em refatorar a aplicação, conforme procedimentos descritos a seguir.

```
const express = require('express')
```

```
const bodyParser = require('body-parser')
```

```
const crypto = require('crypto')
```

```
const app = express()
```

```
app.use(bodyParser.json())
```

```
const port = process.env.PORT || 3000

app.listen(port, () => {

  console.log(`Server is running on port ${port}`)

})

//Endpoint para login do usuário

// Dados do body da requisição: {"username": "user", "password": "123456"}

// Verifique mais abaixo, no array users, os dados dos usuários existentes na app

app.post('/api/auth/login', (req, res) => {

  const credentials = req.body

  let userData;

  userData = doLogin(credentials)

  if(userData){

    //cria o token que será usado como session id, a partir do id do usuário

    const dataToEncrypt = `{"usuario_id":${userData.id}}`;

    const bufferToEncrypt = Buffer.from(dataToEncrypt, "utf8");

    hashString = encrypt(bufferToEncrypt)

  }

  res.json({ sessionid: hashString })

})

//Endpoint para demonstração do processo de quebra da criptografia da session-id
gerada no login

// Esse endpoint, e consequente processo, não deve estar presente em uma API
oficial,

//  aparecendo aqui apenas para finalidade de estudos.

app.post('/api/auth/decrypt:sessionid', (req, res) => {

  const sessionid = req.params.sessionid;

  //const decryptedSessionid = decryptData(sessionid);

  const decryptedSessionid = decrypt(sessionid);

  res.json({ decryptedSessionid: decryptedSessionid })

})
```

//Endpoint para recuperação dos dados de todos os usuários cadastrados

```
app.get('/api/users/:sessionid', (req, res) => {  
  const sessionid = req.params.sessionid;  
  const perfil = getPerfil(sessionid);  
  
  if (perfil !== 'admin' ) {  
    res.status(403).json({ message: 'Forbidden' });  
  }else{  
    res.status(200).json({ data: users })  
  }  
  
})
```

//Endpoint para recuperação dos contratos existentes

```
app.get('/api/contracts/:empresa/:inicio/:sessionid', (req, res) => {  
  const empresa = req.params.empresa;  
  const dtInicio = req.params.inicio;  
  const sessionid = req.params.sessionid;  
  
  const result = getContracts(empresa, dtInicio);  
  if(result)  
    res.status(200).json({ data: result })  
  else  
    res.status(404).json({data: 'Dados Não encontrados'})  
})
```

//Outros endpoints da API

// ...

```
////////////////////////////////////  
///
```

//Mock de dados

```
const users = [  
  {"username": "user", "password": "123456", "id": 123, "email": "user@dominio.com",  
  "perfil": "user"},
```

```
{ "username": "admin", "password": "123456789", "id": 124, "email": "admin@dominio.com", "perfil": "admin" },  
  
{ "username": "colab", "password": "123", "id": 125, "email": "colab@dominio.com", "perfil": "user" },  
  
]
```

```
//APP SERVICES
```

```
function doLogin(credentials){  
  
    let userData  
  
    userData = users.find(item => {  
  
        if(credentials?.username === item.username && credentials?.password === item.password)  
  
            return item;  
  
    });  
  
    return userData;  
  
}
```

```
// Gerando as chaves necessárias para criptografia do id do usuário
```

```
// Nesse caso, a palavra-chave usada para encriptação é o nome da empresa detentora do software em questão.
```

```
const secretKey = 'nomedaempresa';  
  
function encrypt(text) {  
  
    const cipher = crypto.createCipher('aes-256-cbc', secretKey);  
  
    let encrypted = cipher.update(text, 'utf8', 'hex');  
  
    encrypted += cipher.final('hex');  
  
    return encrypted;  
  
}
```

```
// Função de exemplo para demonstrar como é possível realizar a quebra da chave gerada (e usada como session id),
```

```
// tendo acesso ao algoritmo e à palavra-chave usadas na encriptação.
```

```
function decrypt(encryptedText) {  
  
    const decipher = crypto.createDecipher('aes-256-cbc', secretKey);  
  
    let decrypted = decipher.update(encryptedText, 'hex', 'utf8');  
  
    decrypted += decipher.final('utf8');  
  
    return decrypted;  
  
}
```

//Recupera o perfil do usuário através da session-id

```
function getPerfil(sessionId){  
  
    const user = JSON.parse(decrypt(sessionId));  
  
    //varre o array de usuarios para encontrar o usuário correspondente ao id obtido da  
    sessionId  
  
    const userData = users.find(item => {  
  
        if(parseInt(user.usuario_id) === parseInt(item.id))  
  
            return item;  
  
    });  
  
    return userData.perfil;  
}
```

//Classe fake emulando um script externo, responsável pela execução de queries no banco de dados

```
class Repository{  
  
    execute(query){  
  
        return [];  
  
    }  
}
```

//Recupera, no banco de dados, os dados dos contratos

// Metodo não funcional, servindo apenas para fins de estudo

```
function getContracts(empresa, inicio){  
  
    const repository = new Repository();  
  
    const query = `Select * from contracts Where empresa = '${empresa}' And  
data_inicio = '${inicio}'`;  
  
    const result = repository.execute(query);  
  
    return result;  
}
```

- Explicação do código-fonte

- Endpoint '/api/auth/login'

Usado para realização do login do usuário. Nesse processo, um dos dados dos usuários fake constantes ao final do script poderá ser utilizado.

Além de validação do nome de usuário e senha, nesse ponto é gerada a “session-id”, usando o método “encrypt”, também disponível no código-fonte acima.

- Endpoint '/api/auth/decrypt/:sessionid'

Esse endpoint foi incluído no código apenas para que você possa testar, no Insomnia ou Postman, o processo de decifração da senha. Para isso, basta realizar o login e passar, para esse endpoint, a “session-id” obtida.

- Endpoint '/api/users/:sessionid'

Através desse endpoint é possível recuperar os dados de todos os usuários existentes na aplicação.

Nesse endpoint há um controle de acesso baseado em perfil, onde apenas usuários com o perfil ‘admin’ podem ter acesso aos dados.

Como mencionado anteriormente, através do uso de brute force, é possível realizar o processo de engenharia reversa a partir da “session-id”. Para isso, basta o invasor possuir um usuário normal da aplicação e analisar o padrão de URL (URI) da mesma. De posse de uma “session-id” válida é possível testar diferentes algoritmos de quebra de criptografia – processo esse facilitado, uma vez que a chave usada na criptografia da aplicação é uma chave simples e até mesmo óbvia: o nome da própria empresa. De posse do valor obtido após a quebra da chave, o invasor perceberá que a “session-id” é formada por uma string JSON:

```
{"usuario_id":124}
```

No exemplo acima, o ID obtido é o 124. De posse dessa informação, o invasor pode gerar outras “sessions ids”, testando diferentes valores para o “usuario_id”, até encontrar um que seja válido e conceda a ele acesso a endpoints protegidos da aplicação.

- Endpoint '/api/contracts/:empresa/:inicio/:sessionid'

Esse endpoint permite a recuperação dos dados de contratos cadastrados na aplicação. No mesmo são recebidos os seguintes parâmetros: “empresa”, “inicio” e “sessionid”.

Repare que nesse endpoint não é realizada nenhuma tentativa de controle de acesso baseado em perfil. Além disso, no método responsável por montar e executar a consulta no banco de dados, nenhuma sanitização é realizada nos parâmetros de filtro recebidos. Esses dois pontos consistem em uma séria ameaça de segurança.

- Demais métodos da API

Além dos endpoints explicados acima, a API possui alguns métodos. Tais métodos, para fins de simplificação, foram incluídos no mesmo script onde as rotas dos endpoints se encontram. Numa aplicação real, é importantíssimo separar o código em diferentes scripts, de acordo com sua responsabilidade. Ainda em relação aos métodos, há comentários no código disponibilizado explicando a função de cada um deles.

Roteiro de prática

- Material necessário para a prática

• Editor ou IDE para escrita do código-fonte

- Procedimentos

1. Abra o código-fonte fornecido acima na IDE ou editor;
2. Refatore o método de criptografia utilizado atualmente, substituindo a geração do “session-id” por um outro mecanismo de segurança, como tokens JWT;
3. Refatore a arquitetura da API, para que o token (atualmente representado pelo “session-id”) não seja trafegado via URI, mas através do header da requisição;
4. A cada requisição recebida pela API, valide o token de segurança, incluindo a identidade do usuário, data/hora de expiração do mesmo, etc.;
5. Inclua, em todos os endpoints, controle de acesso a recursos baseado no perfil do usuário. Garante que, à exceção do endpoint de login, todos os demais sejam acessados apenas por usuários com perfil ‘admin’;
6. Para testar a implementação do item anterior, crie um novo endpoint que permita a recuperação dos dados do usuário logado. Tal método não deverá conter o controle de acesso limitado ao perfil ‘admin’;
7. Refatore o método que realiza a busca de contratos no banco, tratando os parâmetros recebidos contra vulnerabilidades do tipo “Injection”. Para isso você poderá utilizar bibliotecas de terceiros, expressões regulares ou outro mecanismo que garanta o sucesso do processo em questão;
8. Salve o código e coloque a API para ser executada;
9. Utilizando um cliente (Insomnia, Postman ou outro de sua preferência), realize testes na API, garantindo que todos os pontos acima foram tratados.

- Resultados esperados 🌟

O resultado esperado dessa microatividade é demonstrar ao aluno uma situação real de software vulnerável, permitindo ao mesmo obter e/ou aumentar seu conhecimento sobre o tema de forma teórica e também prática – através da refatoração da aplicação fornecida, aplicando medidas e boas práticas recomendadas na literatura relacionada.

📌 Referências

OWASP CheatSheetSeries. Disponível em: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Authentication_Cheat_Sheet.md#authentication-and-error-messages

Entrega da prática

Chegou a hora, gamer!

👉 Armazene o projeto em um repositório no GIT.

👉 Anexar a documentação do projeto (PDF) no GIT.

👉 Compartilhe o link do repositório do GIT com o seu tutor para correção da prática, por meio da **Sala de Aula Virtual**, na aba "**Trabalhos**" do respectivo nível de conhecimento.

👉 **Ei, não se esqueça de entregar este trabalho na data estipulada no calendário acadêmico!**

Feito com o Microsoft Sway

Crie e compartilhe apresentações, histórias e relatórios interativos e muito mais.

Introdução

