UNIVERSIDADE ESTÁCIO DE SÁ

DESENVOLVIMENTO FULLSTACK

Mundo 05 - Nível 05

RPG0035 - SOFTWARE SEM SEGURANÇA NÃO SERVE!

Ruan Hernandes Finamor Correia Matrícula 202208175252

Gravataí – RS 2024

Objetivo 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.

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

Procedimentos

```
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 decriptação 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 reserva 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 tratativa 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.

- 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.