Projeto nº 2 - Secure Shop

Universidade de Aveiro

Gonçalo Lopes, Gabriel Couto, Tiago Cruz,
Rodrigo Graça, Vasco Faria

Projeto nº 2 - Secure Shop

Segurança Informática e nas Organizações Universidade de Aveiro

Gonçalo Lopes, Gabriel Couto,

Tiago Cruz, Rodrigo Graça, Vasco Faria

(107572) goncalorcml@ua.pt, (103270) gabrielcouto@ua.pt,

(108615) tiagofcruz78@ua.pt, (107634) rodrigomgraca@ua.pt, (107323) vascomfaria@ua.pt

3 de Janeiro 2024

Índice

Capítulo 1

Introdução

Como segundo projeto foi-nos proposto aprimorar a loja online do DETI para atender aos requisitos de Nível 1 do Application Security Verification Standard (ASVS).

Isso envolve uma auditoria completa de conformidade do aplicativo Web, seguida pela implementação de melhorias de segurança identificadas na auditoria.

O objetivo é manter a funcionalidade original da loja, que visa vender itens como canecas, copos, camisetas e moletons, enquanto garantimos um ambiente seguro.

1.1 Project Setup

Frontend:

* React JS - Utilizado para a criação da interface do utilizador (UI) para aprimorara experiência geral do utilizador.

Backend:

* Node JS - Utilizado para a criação do servidor backend, que trata das solicitações do frontend e comunica com a base de dados.

Database:

* SQL - Utilizado para a criação da base de dados, que armazena todas as informações relacionadas à loja online.

Capítulo 2

Auditoria da aplicação Web desenvolvida (versão segura) de acordo com os requisitos do nível 1 do Application Security Verification Standard (ASVS)

A auditoria foi realizada utilizando a checklist do OWASP ASVS, com uma combinação de métodos automatizados e análises manuais. Realizamos testes manuais para validar questões específicas de segurança, como a gestão de sessões e autenticação.

A auditoria revelou várias vulnerabilidades significativas na aplicação. É vital que medidas corretivas sejam implementadas nas aplicações para proteger os dados dos utilizadores e manter a integridade da aplicação. A adesão contínua aos padrões de segurança do OWASP ASVS é essencial para o desenvolvimento seguro de aplicações web.

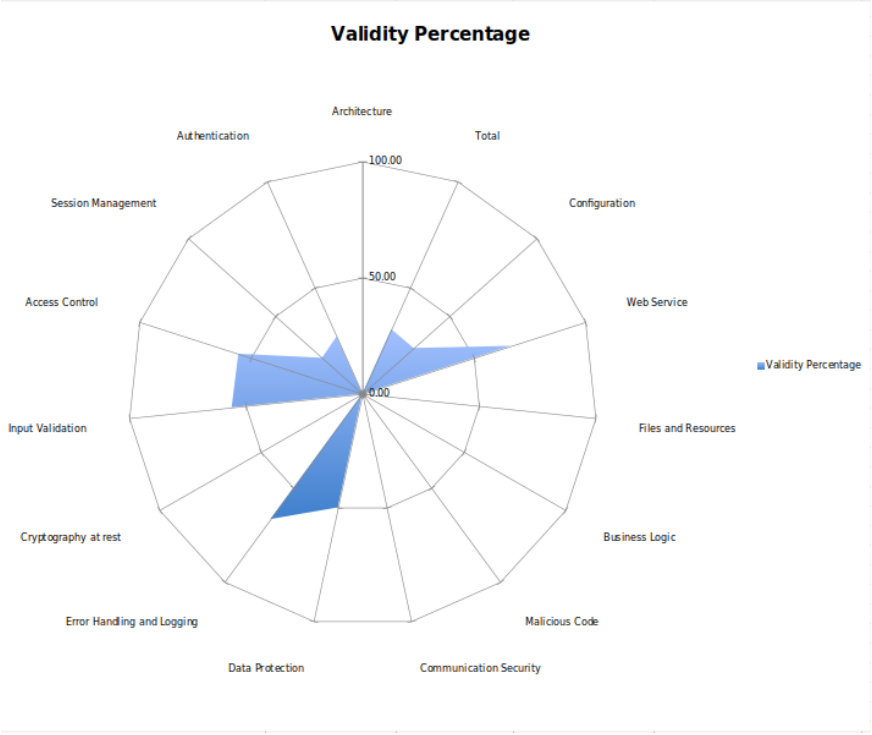

Fig. 1 - Percentagem de Validez na Auditoria

Capítulo 3
Key Issues

2. Authentication

2.1 Password Security Credencials

 # 2.1.7 (ASVS Level 1)

Podemos identificar esse key issue nos processos de login, registo e alteração de senha em nosso site. Este key issue ressalta a necessidade de impor verificações rigorosas nas senhas dos utilizadores para garantir que não sejam facilmente comprometidas. Tais verificações incluem a análise para determinar se a senha é comum ou se atende aos critérios de segurança estabelecidos por uma API.

Para abordar essa questão, implementamos verificações utilizando a API https://api.pwnedpasswords.com. Esta abordagem permite avaliar a robustez das senhas dos utilizadores, verificando-as em relação a uma base de dados de senhas comprometidas anteriormente. Dessa forma, fortalecemos a segurança do sistema ao evitar o uso de senhas vulneráveis, contribuindo para a proteção efetiva das contas dos utilizadores.

```
const checkPasswordAgainstBreaches = async (password) => {
  const sha1Password = crypto.createHash('sha1').update(password).digest('hex');
  const prefix = sha1Password.slice(0, 5);
  const suffix = sha1Password.slice(5).toUpperCase();

  const response = await fetch(`https://api.pwnedpasswords.com/range/${prefix}`);
  const text = await response.text();
  const hashes = text.split('\r\n').map(line => line.split(':')[0]);

  return hashes.includes(suffix);
};
```

Fig. 2 - Password Security Credencials

2.5 Credencial Recovery Requirements

 # 2.5.4 (ASVS Level 1)

Podemos identificar este key issue no arquivo register.js. Este key issue destaca a necessidade de evitar o uso de contas compartilhadas ou padrão, como aquelas com nomes como 'root', 'admin' ou 'sa', por parte dos utilizadores.

Para resolver isso, implementamos uma verificação no register.js que impede a criação de contas caso o nome de utilizador seja um dos mencionados acima ou alguns outros especificados como inválidos. Abaixo está a melhoria do código:

```
let errorMessage = '';
if (sanitizedUsername !== username || sanitizedEmail !== email) {
  errorMessage = 'Invalid username or email input.';
} else if (!validatePassword(sanitizedPassword)) {
  errorMessage = 'Invalid password. Please ensure it meets the requirements.';
} else if (username === '' || email === '' || username === 'sa' || username === 'root' || username === 'admin'
  errorMessage = 'Invalid username or email input.';
}
```

Fig. 3 - Credencial Recovery Requirements

Agora, o código verifica se o nome de utilizador está presente na lista de nomes de utilizador inválidos. Isso proporciona uma maneira mais escalável e legível de garantir que contas compartilhadas ou padrão não sejam utilizadas no momento do registo.

2.7 Out of Band Verifier Requirements

 # 2.7.2 (ASVS Level 1)

Podemos identificar esta key issue na criação do token de login no nosso website. A preocupação aqui é que o verificador não deve expirar solicitações, códigos ou tokens de autenticação fora de banda após 10 minutos.

Para resolver essa questão, implementamos a utilização da API 'Jwt.sign(...ExpiresIn)' no código abaixo. Agora, o token gerado expira após 10 minutos, proporcionando uma abordagem mais segura e alinhada às boas práticas de segurança.

```
const token = jwt.sign(
  { userId: user.id },
  JWT_SECRET,
  { expiresIn: 10*60*1000 }
);
```

Fig. 4 - Out of Band Verifier Requirements

3. Session Management

3.2 Session Binding Requirements

 # 3.2.2 (ASVS Level 1)

Este key issue foi identificada no processo de verificação de tokens de sessão no backend do nosso website, especificamente no módulo jwt_verify. A preocupação é garantir que os tokens de sessão possuam pelo menos 64 bits de entropia (C6).

Para abordar essa questão, implementamos a utilização de um JWT_SECRET com aproximadamente 190 bits. No entanto, é crucial destacar que o JWT_SECRET não deve ser um valor hardcoded, conforme exemplificado no código a seguir. Em um ambiente de produção, o segredo JWT deve ser gerado de forma dinâmica e não deve ser armazenado explicitamente no código, aumentando assim a segurança do sistema.

```
const JWT_SECRET = process.env.JWT_SECRET || 'a3Zb2Cf7gH1jK41M9oQ0rS5tU8xY1wD3'; //nao deve ser assim, nao deve ser guardado em lado nenhum mas e para experimentar
```

Fig. 5 - Session Binding Requirements

 # 3.2.3 (ASVS Level 1)

Podemos identificar este key issue no tratamento do jwt_token. A preocupação aqui é verificar se a aplicação armazena apenas tokens de sessão no navegador usando métodos seguros, como cookies devidamente protegidos ou armazenamento de sessão HTML5 (consulte a seção 3.4).

Para resolver essa questão, implementamos o uso de cookies no código abaixo. Agora, o token de sessão é armazenado de forma segura no navegador, proporcionando maior segurança no tratamento das credenciais do utilizador.

```
res.cookie('sessionToken', token, {
  httpOnly: true,
  secure: true,
  maxAge: 10 * 60 * 1000,
  sameSite: 'Lax'
});
```

Fig. 6 - Session Binding Requirements (1)

Agora, o token de sessão é armazenado de maneira segura usando cookies, proporcionando uma solução alinhada às práticas recomendadas de segurança.

3.4 Cookie-Based Session Requirements

 # 3.4.1 (ASVS Level 1)

Este key issue, referente ao requisito 3.4.1 do ASVS Level 1, destaca a importância de verificar se os tokens de sessão baseados em cookies possuem o atributo 'Secure' configurado (C6).

Para solucionar este problema, implementamos a configuração secure: true no momento da definição do cookie, como exemplificado no trecho de código abaixo:

```
res.cookie('sessionToken', token, {
  httpOnly: true,
  secure: true,
  maxAge: 10 * 60 * 1000,
  sameSite: 'Lax'
});
```

Fig. 7 - Cookie-based Session Requirements

Ao adicionar o parâmetro secure: true, garantimos que os tokens de sessão são transmitidos apenas por meio de conexões seguras (HTTPS), contribuindo para a proteção dos dados sensíveis durante a autenticação do utilizador. Essa medida está em conformidade com as boas práticas de segurança estabelecidas pelo ASVS Level 1.

 # 3.4.2(ASVS Level 1)

Este key issue (C6) destaca a importância de verificar se os tokens de sessão baseados em cookies possuem o atributo 'HttpOnly' configurado. Para abordar essa preocupação, implementamos a configuração httpOnly: true no código abaixo:

```
res.cookie('sessionToken', token, {
  httpOnly: true,
  secure: true,
  maxAge: 10 * 60 * 1000,
  sameSite: 'Lax'
});
```

Fig. 8 - Cookie-based Session Requirements (1)

A inclusão do atributo httpOnly: true assegura que o token de sessão seja acessível apenas pelo servidor, impedindo que seja manipulado pelo JavaScript do lado do cliente. Em conjunto com a configuração secure: true, que garante a transmissão segura do cookie apenas em conexões HTTPS, estamos fortalecendo a segurança do sistema. Essas medidas visam mitigar riscos associados à exposição indevida do token de sessão.

 # 3.4.3(ASVS Level 1)

Esta questão-chave aborda a necessidade de verificar se os tokens de sessão, baseados em cookies, utilizam o atributo 'SameSite' para mitigar a exposição a ataques de falsificação de solicitação entre sites (CSRF) (C6).

Para resolver essa questão, implementamos a utilização do atributo 'SameSite' no código, configurando-o como 'Lax'. Além disso, já tínhamos a opção secure: true para garantir que os cookies sejam transmitidos apenas através de conexões seguras.

```
res.cookie('sessionToken', token, {
  httpOnly: true,
  secure: true,
  maxAge: 10 * 60 * 1000,
  sameSite: 'Lax'
});
```

Fig. 9 - Cookie-based Session Requirements (2)

Essa abordagem assegura que os tokens de sessão, armazenados em cookies, sejam protegidos contra potenciais ataques CSRF, proporcionando uma camada adicional de segurança ao restringir sua exposição a solicitações entre sites não autorizadas.

8. Data Protection

8.1 General Data Protection

 # 8.1.4 (ASVS Level 2)

Este key issue está relacionado à capacidade da aplicação de detetar e alertar sobre números anormais de solicitações, seja por IP, utilizador, total por hora, dia, ou qualquer métrica relevante para a aplicação. Para abordar essa preocupação, implementamos o uso do middleware express-rate-limit no arquivo Server.js.

O código a seguir demonstra a utilização do express-rate-limit para mitigar o risco de ataques de negação de serviço (DoS) e garantir que a aplicação seja capaz de lidar com um volume razoável de solicitações:

```
const rateLimit = require('express-rate-limit');


const limiter = rateLimit({
  windowMs: 15*60*1000, //15min
  max: 100,  //100 max
  standardHeaders: true,
  legacyHeaders: false,
});

app.use(limiter);
```

Fig. 10 - General Data Protection

Configuramos o express-rate-limit para permitir no máximo 100 solicitações a cada 15 minutos por IP. Isso ajuda a prevenir potenciais abusos ou ataques de sobrecarga ao impor limites e fornecer uma camada adicional de segurança à aplicação.

Essa abordagem contribui para a deteção e mitigação proativa de atividades anormais, protegendo a aplicação contra possíveis explorações maliciosas.

14. Configuration

14.4 Http Header Security Requirements

 # 14.4.5 (ASVS Level 1)

Esta key issue envolve a verificação da presença do cabeçalho Strict-Transport-Security em todas as respostas, garantindo que esteja configurado para todos os subdomínios, como exemplificado por Strict-Transport-Security: max-age=15724800; includeSubdomains.

Para solucionar essa questão, implementamos o uso do middleware helmet() no arquivo Server.js. O código abaixo demonstra a configuração específica para o cabeçalho Strict-Transport-Security, atendendo às recomendações de segurança:

```
const sixtyDaysInSeconds = 15724800;
app.use(helmet.hsts({
  maxAge: sixtyDaysInSeconds,
  includeSubDomains: true
}));

app.use(helmet());
```

Fig. 11 - Http Header Security Requirements

Ao utilizar o helmet() com a configuração específica para Strict-Transport-Security, garantimos que todas as respostas incluam o cabeçalho apropriado, contribuindo para a segurança da aplicação. Essa abordagem ajuda a proteger contra ataques como downgrade de protocolo e reforça a política de segurança no transporte (HSTS) para todos os subdomínios.

14.5 Validate Http Request Header

 # 14.5.3 (ASVS Level 1)

Esta key issue específica reside no arquivo server.js e destaca a importância de verificar se o cabeçalho Cross-Origin Resource Sharing (CORS), especificamente o Access-Control-Allow-Origin, utiliza uma lista rigorosa de permissões para domínios confiáveis. Além disso, é crucial incluir subdomínios para corresponder e evitar o suporte à origem 'null'.

Para abordar essa questão, implementamos a solução utilizando a biblioteca cors. No trecho de código a seguir, configuramos o corsOptions para especificar um único domínio confiável e permitir o uso de credenciais:

```
const corsOptions = {
  origin: 'http://localhost:3000',
  credentials: true,
};

app.use(cors(corsOptions));
```

Fig. 12 - Validate Http Request Header

Essa abordagem reforça a segurança ao restringir explicitamente as origens permitidas, incluindo subdomínios para corresponder, e habilitando a transferência de credenciais somente para domínios confiáveis. Isso contribui para um controle mais robusto sobre as solicitações CORS, evitando a origem 'null' e mitigando possíveis vulnerabilidades relacionadas à segurança.

Capítulo 4
Features

Password strength evaluation

Em seguida serão detalhadas as funcionalidades implementadas no nosso projeto para assegurar que as senhas dos utilizadores atendem a critérios específicos de segurança. Essas implementações visam estar em conformidade com as diretrizes da OWASP ASVS V2.1, contribuindo para a robustez e a integridade da segurança das credenciais dos utilizadores.

Comprimento Mínimo da Senha:

A primeira linha de defesa na segurança de senhas é garantir que todas as senhas atinjam um comprimento mínimo requerido. Na nossa aplicação, esse comprimento é de 12 caracteres. Esta medida aumenta a dificuldade para ataques de força bruta, sendo um passo inicial vital para a segurança da conta do utilizador. Esta medida foi aplicada da seguinte forma:

```
if (normalizedPassword.length < 12) {
  return res.status(400).json({ error: 'Password must be at least 12 characters long.' });
}
```

Fig. 13 - Check password lenght

Permissão de Senhas Longas

Além do comprimento mínimo, é importante que o sistema aceite senhas longas para permitir que os utilizadores criem senhas complexas e seguras. As senhas podem ter 64 caracteres ou mais, até um máximo de 128 caracteres. Esta regra é implementada para prevenir restrições desnecessárias no comprimento da senha, permitindo assim que os utilizadores coloquem frases como senha ou outras técnicas de criação de senhas seguras.

```
if (normalizedPassword.length >128){
  return res.status(400).json({error: 'Password must not be longer than 128 characters'})
}
```

Fig. 14 - Check password lengh

Não Truncamento de Senhas

Para assegurar a integridade das senhas dos utilizadores, o nosso sistema não realiza truncamento de senhas. Em vez disso, qualquer sequência com vários espaços é reduzida a um único espaço. Isso evita a perda de informação da senha devido a truncamento e garante que a senha utilizada pelo utilizador seja processada e armazenada de forma exata.

```
try {

  const normalizedPassword = password.replace(/\s+/g, ' ').trim();
```

Fig. 15 - Check password

Verificação de Senha Contra Violações Anteriores:

Para evitar que os nossos utilizadores reutilizem senhas que já foram comprometidas em violações anteriores, implementamos uma verificação que utiliza um serviço externo. Esta verificação é feita através da API "Have I Been Pwned", que permite identificar se uma senha já foi exposta em qualquer violação de dados conhecida sem comprometer a senha do utilizador.

```
const checkPasswordAgainstBreaches = async (password) => {
  const sha1Password = crypto.createHash('sha1').update(password).digest('hex');
  const prefix = sha1Password.slice(0, 5);
  const suffix = sha1Password.slice(5).toUpperCase();

  const response = await fetch(`https://api.pwnedpasswords.com/range/${prefix}`);
  const text = await response.text();
  const hashes = text.split('\r\n').map(line => line.split(':')[0]);

  return hashes.includes(suffix);
};
```

Fig. 16 - API implementation

Implementação do OAuth 2.0 + OIDC Login

Aqui é implementado OAuth 2.0 na aplicação Web, focando na autenticação dos utilizadores através do GitHub.

OAuth 2.0 é um padrão de autorização que permite a um utilizador conceder a uma aplicação terceira um acesso limitado aos seus recursos armazenados noutro serviço, sem expor as suas credenciais. É amplamente utilizado para permitir logins sociais e integrações entre diferentes plataformas online, oferecendo um método seguro e eficiente para gerenciar permissões de acesso.

A aplicação utiliza o OAuth 2.0 para permitir que os utilizadores se autentiquem usando suas contas do GitHub. Esta implementação oferece uma experiência de login mais simples para o utilizador, pois estes podem usar uma conta existente em vez de criar uma nova.

O nosso projeto inclui uma funcionalidade que direciona o utilizador para a página de autenticação do GitHub, que é um exemplo clássico do uso do OAuth 2.0:

Fig. 17 - OAuth 2.0 implementation

Este botão inicia o processo de autenticação OAuth 2.0. Quando um utilizador clica no botão, é redirecionado para a página de login do GitHub, onde pode autorizar a aplicação a aceder às suas informações de perfil sob a meta especificada (read:user). Após a autorização, o GitHub redireciona o utilizador de volta para a aplicação com um código de autorização.

Capítulo 5
Conclusão

Este projeto proporcionou-nos uma valiosa oportunidade para aprofundar os nossos conhecimentos e explorar diversos temas essenciais abordados durante o curso. Em particular, dedicamo-nos ao estudo dos intricados processos de uma PKI, bem como aos princípios fundamentais de confidencialidade, integridade e autenticidade dos dados transmitidos.

Ao desenvolver protocolos e mecanismos de segurança robustos, adotamos uma abordagem proativa ao assumir o papel de potenciais atacantes. Este exercício constante nos permitiu identificar e corrigir potenciais pontos vulneráveis em nossa implementação. Antecipamos várias formas em que as mensagens poderiam ser manipuladas por agentes externos, considerando cuidadosamente os possíveis efeitos e impactos resultantes.

Diante desse desafio, estamos satisfeitos com os resultados obtidos e reconhecemos a importância fundamental do nosso trabalho. A consideração diligente das ameaças potenciais e a implementação de medidas preventivas consolidaram não apenas a segurança do projeto, mas também enriqueceram significativamente nossa compreensão prática dos conceitos fundamentais de segurança da informação. Este trabalho representa uma etapa crucial em nossa jornada de aprendizado, fortalecendo nossa capacidade de projetar e implementar soluções seguras em contextos práticos.