universidade de aveiro

deti universidade de aveiro departamento de eletrónica, telecomunicações e informática

Relatório de Análise

Projeto 2 – Application Security Verification Standard (ASVS)

Segurança Informática e nas Organizações (SIO)

Departamento de Eletrónica, Telecomunicações e Informática

Ano Letivo 2023/2024

Guilherme Santos, 107961

João Gaspar, 107708

Miguel Ferreira, 93419

Pedro Coutinho, 93278

Índice

Introdução

No último projeto realizado, foi criada uma aplicação online de uma loja de recordações do DETI da Universidade de Aveiro, possuindo algumas soluções de proteções de possíveis ataques feitos por terceiros e com soluções de proteção do utilizador.

Nesta segunda fase, temos como objetivo fazer a auditoria da aplicação Web desenvolvida anteriormente de acordo com os requisitos para o nível 1 do *Application Security Verification Standard* (ASVS). Além disso, temos como objetivo melhorar a aplicação focando principalmente nos seguintes pontos: *Multi-factor Authentication* (MFA) e *Encrypted database storage.*

ASVS Level 1

Levar em conta que o a numeração dos problemas corresponde respetivamente à numeração da solução (e.g. problema 1 corresponde à solução 1).

Problemas

Problema 1 (#2.1.1)

A aplicação original permitia que os utilizadores definissem passwords com menos de 12 caracteres, o que apresentava uma vulnerabilidade significativa à segurança. O requisito C6 do *Proactive Controls* destaca a importância de senhas fortes e que é crucial garantir que as mesmas tenham um comprimento mínimo adequado.

Problema 2 (#2.1.4)

A aplicação anteriormente limitava a inclusão de caracteres Unicode, incluindo caracteres imprimíveis, espaços e Emojis, nas passwords dos utilizadores. Esta restrição comprometia a diversidade de caracteres disponíveis, prejudicando a robustez da segurança das passwords.

Problema 3 (#2.1.7)

Antes a aplicação não realizava a verificação das passwords submetidas durante o registo da conta, login e alteração de password em relação a uma lista de passwords comprometidas. Essa verificação é essencial para garantir que as passwords não estejam entre as mais comuns ou comprometidas, o que é fundamental para aumentar a segurança do sistema.

Problema 4 (#2.1.9)

Antes das melhorias, a aplicação impunha regras de composição de password, limitando o tipo de caracteres permitidos. Essas regras incluíam exigências quanto à presença de maiúsculas, minúsculas, números ou caracteres especiais. Isso não apenas restringia a escolha dos utilizadores, mas também podia tornar as passwords mais previsíveis e, consequentemente, menos seguras.

Problema 5 (#2.5.1)

Antes a aplicação enviava em texto simples para o utilizador o segredo de ativação ou recuperação inicial gerado pelo sistema. Isso representava um risco significativo de segurança, pois a transmissão de segredos sensíveis em texto simples pode ser intercetada e comprometer a integridade do sistema.

Problema 6 (#3.4.1)

Os cookies da aplicação original não possuíam o atributo “Secure” configurado, fazendo com que as informações presentes nas cookies possam ser transmitidas em conexões HTTP não seguras, provocando com que essa informação seja vulnerável a ataques de interceção.

Problema 7 (#3.4.3)

Os cookies de sessão baseados em cookies na aplicação não utilizavam o atributo “SameSite”. Essa omissão deixava a aplicação mais vulnerável a ataques de falsificação de solicitação entre sites (CSRF), pois não limitava adequadamente a exposição do cookie a solicitações de outros sites.

Problema 8 (#8.3.4)

A aplicação não possuía um processo de identificação e tratamento de dados sensíveis e esta ausência representava um risco de exposição e manipulação inadequada de informações sensíveis. Os dados eram processados sem aplicar medidas de cifragem o que exponha essas informações a ameaças de segurança.

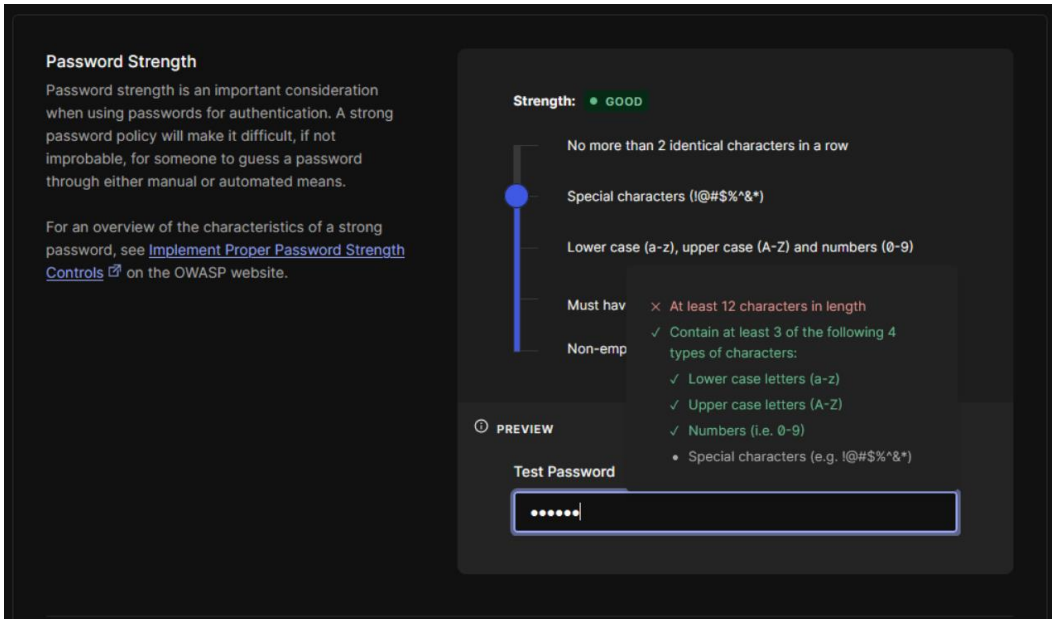
Problema 9 (#12.1.1)

A aplicação não deve permitir o upload de ficheiros demasiado grandes para o servidor de modo que o armazenamento do servidor não encha e cause DoS (Denial of Service).

Soluções

Solução 1 (#2.1.1)

Após a implementação da autenticação Auth0, o sistema foi automaticamente configurado para impor requisitos mínimos de password, garantindo assim que todas as passwords definidas pelos utilizadores tenham pelo menos 12 caracteres. Esta medida reforçou a segurança da aplicação, eliminando a vulnerabilidade anterior a ataques de força bruta.



Solução 2 (#2.1.4)

Com as atualizações feitas, a aplicação foi ajustada para permitir qualquer caráter Unicode, incluindo caracteres imprimíveis, espaços e Emojis, nas passwords dos utilizadores. Essa medida resolveu o problema inicial, proporcionando uma maior diversidade de caracteres e fortalecendo a segurança global das passwords.

Solução 3 (#2.1.7)

Agora a aplicação foi configurada para verificar as passwords submetidas. Esta verificação é realizada através de uma API externa. Se a password estiver comprometida, a aplicação agora exige que o utilizador defina uma nova password que não esteja comprometida. Este procedimento aumenta significativamente a segurança, garantindo que as passwords utilizadas não estejam entre as mais comuns ou comprometidas.

Solução 4 (#2.1.9)

Após implementar a autenticação Auth0, a aplicação foi ajustada para não impor regras de composição de password que limitem o tipo de caracteres permitidos. Não há mais exigências quanto à presença de maiúsculas, minúsculas, números ou caracteres especiais.

Solução 5 (#2.5.1)

De modo a resolver este problema, a aplicação, após a aplicação da autenticação Auth0, foi ajustada para garantir que o segredo de ativação ou recuperação inicial gerado pelo sistema não seja enviado em texto simples para o utilizador. Agora, mecanismos seguros, como cifragem ou hash, são utilizados para proteger a transmissão dessas informações sensíveis.

Solução 6 (#3.4.1)

Com o atributo configurado como "*secure*", o cookie só será enviado de volta ao servidor se a conexão estiver sendo feita através de conexões HTTPS (HTTP seguro). Ou seja, as informações dos cookies só serão enviadas em conexões seguras e criptografadas.

```
# Atributo "Secure" dos Tokens
app.config['SESSION_COOKIE_SECURE'] = True
```

Solução 7 (#3.4.3)

```
28
29    # Atributo "SameSite" dos Tokens (Strict)
30    app.config['SESSION_COOKIE_SAMESITE'] = 'Lax'
31
```

A aplicação foi ajustada para garantir que os cookies utilizem o atributo "*SameSite*" com o valor "*Lax*". Assim o cookie será enviado em solicitações de origem cruzada, mas apenas para solicitações GET geradas por um site de terceiros. Isso ajuda a proteger contra ataques de CSRF.

Solução 8 (#8.3.4)

De modo a resolver este problema tivemos de identificar todas as informações sensíveis criadas e processadas pela aplicação. Posteriormente, implementamos o algoritmo de cifragem *Fernet* para proteger dados sensíveis durante o processamento.

Solução 9 (#12.1.1)

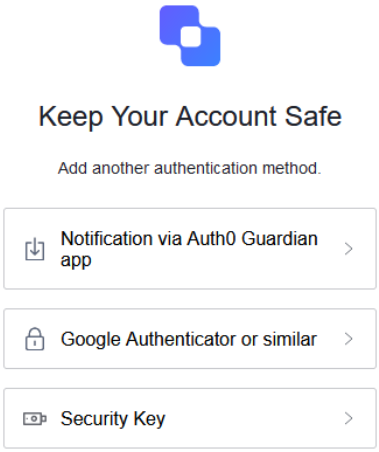
```
26
27    app.config['MAX_CONTENT_LENGTH'] = 500 * 1000 * 1000
28
```

Com o parâmetro de configuração “MAX_CONTENT_LENGTH” disponibilizado pelo Flask, garante-se que não será possível dar upload a ficheiros demasiado grandes.

Software Features

Multi-Factor Authentication (MFA)

Com a implementação do Auth0 para ser o sistema de autenticação usado pela aplicação, foi possível integrar autenticação por dois fatores, visto que este serviço externo lida com todo o processo de autenticação, facilmente pode-se oferecer vários métodos de autenticação. Isto aumenta significativamente a segurança do sistema, pois requer autenticação adicional além da password.



Encrypted Database Storage

A funcionalidade de armazenar dados encriptados na base de dados também foi adicionada, garantindo que dados críticos são cifrados. Isto contribui para a proteção de informações sensíveis armazenadas no sistema. Os dados que achamos relevantes encriptar são os dados de faturação do utilizador. A encriptação é feita usando uma *Fernet key*, que é uma *key* composta por duas *keys* mais pequenas, uma *encryption key* 128-bit AES e uma *signing key* 128-bit SHA256 HMAC.

As seguintes duas imagens apresentam o código usado para encriptar os dados:

```
key = Fernet.generate_key()
self.cipher_suite = Fernet(key)
```

```
billing_country = self.cipher_suite.encrypt(billing_country.encode())
billing_fname = self.cipher_suite.encrypt(billing_fname.encode())
billing_lname = self.cipher_suite.encrypt(billing_lname.encode())
billing_companyname = self.cipher_suite.encrypt(billing_companyname.encode())
billing_address = self.cipher_suite.encrypt(billing_address.encode())
billing_state_country = self.cipher_suite.encrypt(billing_state_country.encode())
billing_postal_zip = self.cipher_suite.encrypt(billing_postal_zip.encode())
billing_email_address = self.cipher_suite.encrypt(billing_email_address.encode())
billing_phone = self.cipher_suite.encrypt(billing_phone.encode())
```

A seguinte imagem apresenta como os dados ficam guardados na base de dados: