Assignment 2: Application Security Verification Standard (ASVS)

Universidade de Aveiro

ASVS, Loja Online DETI, Audit, Issues, OWASP, Software Features

David Palricas, Eduardo Alves , Inês Silva, João Alcatrão, Mariana Silva

universidade de aveiro

Assignment 2: Application Security Verification Standard (ASVS)

Universidade de Aveiro

(108780) David Palricas
davidpalricas@ua.pt

(104179) Eduardo Alves
eduardoalves@ua.pt

(104322) Inês Silva
inesasilva@ua.pt

(76763) João Alcatrão
jalcatrao@ua.pt

(98392) Mariana Silva
marianabarbara@ua.pt

2 de janeiro de 2024

Índice

Capítulo 1

Introdução

No âmbito da disciplina de Segurança Informática e nas Organizações, concluímos o segundo projeto: "Assignment 2: Application Security Verification Standard (ASVS)", que é baseado na implementação de requisitos de nível 1 do ASVS (Application Security Verification Standard (ASVS) no nosso website (que iremos listar mais à frente), bem como de algumas features críticas para a segurança do mesmo. Quanto a estas últimas, decidimos implementar: (1) um sistema de autenticação com 2 fatores (2FA) recorrendo a códigos TOTP para permitir o login dos utilizadores; (2) um sistema de avaliação das passwords que os utilizadores pretendem submeter aquando a criação de uma nova conta, bem como o uso de um serviço externo (usámos a API HIBP (HaveIbeenPwned) para determinar se determinada password já foi exposta anteriormente (de acordo com os dados da API) ou não.

Para isto, começamos por realizar uma análise á versão segura do website de merchandising do DETI que realizámos no primeiro projeto, de acordo com os requisitos do nível 1 do ASVS. Conduzimos posteriormente uma análise de segurança de modo a identificar 10 problemas-chave de elevada relevância na nossa aplicação. Após a identificação destes, melhorámos a nossa aplicação através da resolução desses problemas e implementámos das duas funcionalidades de software críticas mencionadas anteriormente

Neste último passo, eram-nos dadas as seguintes opções:

- Password strength evaluation and breach verification;
- Multi-factor Authentication (MFA) (via TOTP no nosso caso);
- Encrypted database storage.

Para o nosso projeto, escolhemos implementar a Multi-factor Authentication (MFA) recorrendo a autenticação com códigos TOTP, e a Password strength evaluation e breach verification.

Capítulo 2

Introdução aos issues

Para este projeto, escolhemos os 10 issues mais relevantes no processo de segurança no nosso website de Memorabilia do Deti. No entanto, antes de escolhermos os issues mais relevantes, precisámos da definição de issue de forma a melhor perceber o problema.

Issue: "An issue corresponds to a specific requirement not being met.", ou seja, trata-se de um requisito especifico que não está a ser corretamente implementado de acordo com as politicas do ASVS.

Agora que sabemos do que se trata um issue, podemos escolher os mais adequados para a nossa aplicação. Neste caso, escolhemos os issues com maior relevância na resolução de problemas relacionados com a segurança na autenticação e com a gestão de sessões. Escolhemos colocar maior foco neste tipo de issues, dado que o nosso website inicial trabalha muito com este tipo de medidas e achamos que são essenciais no aumento do grau de segurança na pãgina de merchandising do DETI.

Capítulo 3

Issues

De seguida apresentamos os 10 issues mais relevantes que escolhemos, identificamos e analisamos o problema e apresentamos a nossa resolução do mesmo.

Os 3 primeiros issues são relacionados com a gestão de sessões, enquanto os restantes têm um maior foco no processo de autenticação.

3.0.1 Issue 1

Verify that cookie-based session tokens have the 'Secure' attribute set.

Este issue está relacionado com o CWE 614 e tem o NIST: 7.1.1 #3.4.1
No nosso primeiro projeto, na nossa aplicação web não tivemos em atenção a configuração dos cookies da sessão para que eles tivessem o atributo 'secure' definido. Sendo isto crucial uma vez que, ao garantirmos que os cookies da sessão só são transmitidos através de conexões seguras (https) ajuda a previnir ataques de interceptação de dados como ,por exemplo, ataques man in the middle (MitM). Desta forma ao seguirmos este requisito, estamos a melhorar a proteção do nosso site contra possíveis ameaças à integridade e confidencialidade das sessões dos utilizadores.

De forma a implementar a verificação que os tokens dos cookies têm o atributo seguro:

```
1 cherrypy.tools.response_headers = cherrypy.Tool('before_finalize',
      cherrypy.tools.response_headers, priority=30)
2 cherrypy.config.update({'tools.response_headers.on': True})
3 cherrypy.config.update({'tools.response_headers.headers': [('Set-
      Cookie', 'Secure; HttpOnly; SameSite=Strict;')]})
```

Desta forma, as definições de como o cherrypy lida com as cookies são atualizadas, sendo que no nosso webserver só serão transmitidos cookies através de ligações seguras e encriptadas, o que mitiga os riscos de haver algum atacante a fazer eavedropping na comunicação e conseguir captar quaisqueres dados sensíveis (ou não sensíveis) relevantes.

3.0.2 Issue 2

Verify that cookie-based session tokens have the 'HttpOnly' attribute set.

Este issue está relacionado com o CWE 1004 e tem o NIST: 7.1.1 #3.4.2
E tal como o anterior, está relacionado com a segurança na gestão de cookies em aplicações web. Ao definirmos o atributo 'HttpOnly', significa que o cookie não pode ser acedido através de scripts do lado do cliente, ou seja, reforça a segurança do website ao probir o javascript de aceder a cookies. Só será enviado ao servidor em soliticação HTTP, qualquer tentativa de acesso por meio de scripts do cliente é bloqueada. Assim, é possível mitigar alguns tipos de ataques como referido em cima através de scripts, podendo roubar informação de cookies incluindo tokens de sessão.

De forma a implementar a verificação que os tokens dos cookies têm o atributo HttpOnly:

```
1 cherrypy.tools.response_headers = cherrypy.Tool('before_finalize',
      cherrypy.tools.response_headers, priority=30)
2 cherrypy.config.update({'tools.response_headers.on': True})
3 cherrypy.config.update({'tools.response_headers.headers': [('Set-
      Cookie', 'Secure; HttpOnly; SameSite=Strict;')]})
```

Desta forma, os dados contidos nos cookies (como, por exemplo, o ID da sessão) ficam isolados, não podendo ser acedidos ou lidos ou editados pelo javascript no client-side. Um claro benefício disto, é que são muito mitigados os potenciais danos de um ataque de XSS, por exemplo.

3.0.3 Issue 3

Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks [C6]

Este issue está relacionado com o CWE 16 e tem o NIST: 7.1.1 #3.4.3
O facto de não termos a configuração apropriada do atributo 'SameSite' nas cookies da sessão, deixa o nosso site exposto a sérios riscos de segurança. Está falha pode residir na vulnerabilidade a um ataque CSRF, por exemplo. Assim, os cookies da sessão tornam-se suscetíveis a serem utilizados em soliticações de terceiros, abrindo a porta para manipulações por parte de atacantes, que pode acabar na execução não autorizada de ações em nome do utilizador. Desta forma, ao implementarmos o requisito 'SameSite' permite-nos controlar quando é que as cookies são enviadas com um request, estamos assim a fortalecer as nossas defesas contra este tipo de ataques, propocionando uma camada de segurança esencial no nosso site.

De forma a implementar a verificação que os tokens dos cookies têm o atributo sameSite:

```
1 cherrypy.tools.response_headers = cherrypy.Tool('before_finalize',
      cherrypy.tools.response_headers, priority=30)
2 cherrypy.config.update({'tools.response_headers.on': True})
3 cherrypy.config.update({'tools.response_headers.headers': [('Set-
      Cookie', 'Secure; HttpOnly; SameSite=Strict;')]})
```

No contexto da proteção contra CSRF, com uma política estrita (SameSite=Strict), cookies apenas são enviadas quando a request origina do mesmo site onde a cookie foi criada. Isto impede a ocorrência de ataques CSRF, porque enviar cookies apenas com requests no mesmo site é o oposto de as enviar com cross-site requests (same site requests).

3.0.4 Issue 4

Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password.

Este issue está relacionado com o CWE 521, #2.1.8.

As palavras-passe no nosso site de memorabilia do DETI são avaliadas por um processo seletivo antes de poderem ser escolhidas (que faz parte de uma das funcionalidades de software críticas que escolhemosm, e de que falaremos mais à frente). Para além deste processo, estas são analisadas por um serviço externo (API HIBP) para garantir a proteção contra um conjunto de falhas "breach verification". Este processo está ligado diretamente à funcionalidade de software "Password strength evaluation"que implementamos neste projeto, e a nossa solução partilha código entre ambas as features:

```
1  def password_breach_verification(self,password):
2          #criar hash da password usando SHA-1 (tal como a API Have I
       Been Pwned exige (...Each password is stored as both a SHA-1
       and...))
3          hash = hashlib.sha1(password.encode('utf-8')).hexdigest().
       upper()
4
5          #enviar apenas os 5 primeiros caract res da hash para a
       API e depois verificar se os restantes caract res se encontram
       em alguma das hashs que a API devolver (modelo K-anonymity, em
       que envia-se apenas por  es da hash e nunca ela inteira,
       para n o levar esta leak a meio da verifica  o e fazer mais
       mal que bem)
6          prefix = hash[:5]
7          suffix = hash[5:]
8
9          #fazer o request para a API e verificar se o sufixo aparece
       nos resultados obtidos (que t m todos o prefixo enviado). Se
       aparecer,    porque a palavra passe levou breach
10         response = requests.get(f'https://api.pwnedpasswords.com/
       range/{prefix}')
11         return suffix in response.text
```

Esta função cria uma hash da password (em SHA-1, tal como a API HIBP exige), e envia um prefixo da mesma para a API através de um request de modo a obter de volta uma lista de todas as hashes de passwords expostas presentes na API que contenham esse mesmo prefixo. Se o sufixo também se encontrar em algumas dessas hashes, é porque a password é vulnerável.

3.0.5 Issue 5

Verify that time-based OTPs have a defined lifetime before expiring.

Este issue está relacionado com o CWE 613, #2.8.1.
A realização deste issue esteve diretamente ligado à implementação da funcionalidade de software "Multi-factor Authentication", uma vez que o timed-based OTPs (TOTPs) que usámos serviu para produzir um código através da aplicação do Google Authenticator temporário que permite aceder à plataforma no processo de Login no site. Considerámos este issue como muito relevante devido ao impacto positivo na melhoria de segurança que causa em comparação com o processo previamente trabalhado no primeiro projeto (apenas username e password).

O uso dos TOTPs permite garantir que um atacante não consiga aceder diretamente ao site com a conta de um outro utilizador, mesmo conseguindo obter, de forma maliciosa, acesso ao username e palavra-passe desse utilizador, pois precisa do código TOTP correto para se autenticar. Sem este código temporário, o atacante não deverá conseguir comprometer o utilizador e entrar no site.

Em grande parte das apps de geração de códigos TOTP, como o Google Authenticator, é gerado um novo código a cada 30s. Na nossa aplicação, geramos o código sempre que o utilizador se tenta autenticar, mas o código dado varia em função do segredo gerado para esse utilizador aquando o registo (que é aleatório e desconhecido) e em função do tempo, sendo que a cada 30s, o código é atualizado, estando em sintonia com o que aparece no Google Authenticator (isto é, se os dispositivos onde se encontra o cliente do webserver e o Google Authenticator estiverem com o horário atualizado).

Segue-se o excerto de código que nos dá os códigos TOTP:

```
1 def generate_totp(secret):
2     totp = pyotp.TOTP(secret)
3     return totp.now()
```

Por muito simples que este excerto de código pareça, a biblioteca pyotp.TOTP usa o algoritmo TOTP, tal como o Google Authenticator (e daí a total conformância entre os valores obtidos por ambos), que é uma extensão do algoritmo HMAC-based One-Time Password (HOTP), e ambos estão específicados no standard RFC 6238. Todos os parâmetros do algoritmo, tal como o intervalo (30s), o número de dígitos (6), modo de operação (HMAC-SHA-1) são iguais entre os valores por defeito do pyotp.TOTP e o Google Authenticator, sendo que se também o segredo for igual, os resultados obtidos são iguais. E mudam a cada 30s.

É de alta importância que estes códigos de acesso sejam temporários de forma a evitar riscos de segurança. Como o código secreto está em constante mudança, torna-se impossível para um atacante utilizar ataques de força bruta (entre outros ataques), pois nunca é possível descartar opções de possíveis códigos com o formato de tentativa e erro. Caso os códigos não fossem temporários, estes

códigos secretos, agiriam apenas como uma segunda palavra-passe, perdendo uma parte importante das suas capacidades de manter a segurança.

3.0.6 Issue 6

Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.

Este issue está relacionado com o CWE 330, #2.3.1.

Como explicado no issue anterior, o Multi-factor Authentication recorrendo a TOTP garante a disponibilidade de códigos únicos temporários, gerados de forma aleatória através, e usando como segredo um valor gerado aleatoriamente com 20 bytes (caractéres) aquando o utilizador cria a conta. Estes códigos são constituidos por 6 digitos que expiram ao fim de 30 segundos, garantindo assim, a sua imprevisibilidade e como efeito, aumentando a sua segurança. Estas códigos de acesso nunca se podem tornar na password de longo prazo, pois não cumprem os requisitos mínimos que nós definimos acima.

É importante que estes códigos não atuem como uma palavra-passe de longo prazo, para este propósito, o utilizador já tem a sua própria palavra-passe à sua escolha como parte das suas credenciais de acesso. A função destes códigos é agirem como uma camada extra protetora de uma qualidade diferente, diminuindo a chance de sucesso de possiveis ataques maliciosos, fornecendo um método extra de autenticação ao utilizador (para além do genérico username e password).

3.0.7 Issue 7

Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as built-in functionality.

Este issue está relacionado com o CWE 521, #2.1.12.
No site do projeto 1, não existia a capacidade de visualizar temporariamente a senha, o que podia levar o utilizador a cometer erros durante a inserção das credenciais no processo de login, tendo como consequência uma experiência mais frustante. Para combater isto, nesta nova versão, permitimos ao utilizador a visualização da sua senha através de um botão ("Mostrar") na página de login no site. Este botão, revela a palavra-passe escrita pelo utilizador durante 3 segundos, ao fim deste tempo, a palavra-passe volta a ficar oculta. Ao fazermos esta implementação estamos a melhorar a usabilidade, fornecendo um feedback visual durante a entrada da senha.
Além disso, o tempo definido pelo temporizador (3 segundos neste caso) é uma escolha que tem em consideração a experiência do utilizador e as preocupações de segurança.
Função em JavaScript:

```
function show_password(){
    var passwordContent = document.getElementById('password');
    passwordContent.type = passwordContent.type === 'password' ? '
    text' : 'password';
    setTimeout(function(){
        passwordContent.type = 'password';

    }, 3000); // Ao fim de 3 segundos, a palavra-passe    camuflada
}
```

Acrescentámos também em HTML:

```
<button type="button" class="button text-success" id ='
    revealPassword' onclick = "show_password()">Mostrar</button>
```

3.0.8 Issue 8

Verify that a password strength meter is provided to help users set a stronger password.

Este issue está relacionado com o CWE 521 e tem o NIST: 5.1.1.2. #2.1.8, tal como está diretamente ligado à feature de software crítica de Password Strength Evaluation que escolhemos implementar.

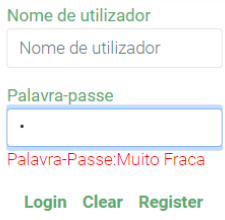
Um medidor de força de palavras chaves é uma ferramenta extremamente útil para um utilizador novo no website saber se a palavra-passe que está a criar é segura (e combinado com a feature de Password Strength Evaluation, não só permite o utilizador saber se está uma palavra-passe mais forte se ele assim o desejar, mas também o obrigado a criar uma que seja Muito Forte, guiando-o para esse objetivo).

Para isto, a nova versão do nosso site, implementa um medidor de força de palavras-passe que consoante a palavra-passe inserida pelo utilizador retorna um feedback através do uso de várias cores demonstrativas e alertando com o respetivo nivel de força, desde muito fraco a muito forte, o utilizador.

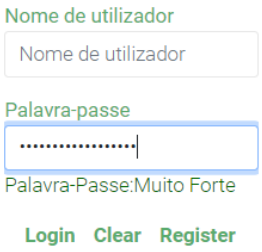
As várias cores do medidor são:

- Vermelho - Muito fraca: O nível mais baixo de segurança.
- Laranja - Fraca
- Dourado - Média
- Verde - Forte
- Verde escuro - Muito forte: O nivel mais alto de segurança.

Caso a palavra-passe não seja apresentada como muito forte (verde escuro), o nosso website não permite que a palavra-passe seja utilizada. Garantindo assim um alto nivel de segurança no processo de autenticação do utilizador neste parâmetro. Caso a password não tenha um valor inserido, não aparece uma cor demonstrativa.



(a) Exemplo: Muito Fraca



(b) Exemplo: Muito Forte

Para implementar isto, criámos uma função em javascript:

```
function password_Strength() {
    let password = document.getElementById("password").value;
    let password_strength = document.getElementById("
    password_strength");
    var strength = -1;
    const strength_values = ["Muito Fraca", "Fraca", "M dia", "
    Forte", "Muito Forte"];
    const colors = ["red", "orange", "gold", "green", "darkgreen"];

    if (password.match(/[a-z]/)) {
        strength++;
    }
    if (password.match(/[A-Z]/)) {
        strength++;
    }
    if (password.match(/[0-9]/)) {
        strength++;
    }
    if (password.match(/[#*]/)) {
        strength++;
    }
    if (password.length >= 12 && password.length <= 128) {
        strength++;
    } else if (password.length > 128) {
        strength = -1;
    }
    if (strength > colors.length - 1) {
        strength = colors.length - 1;
    }
    if (strength == -1) { //caso a password    apagada ou tenha
    mais de 128 caracteres
        password_strength.textContent = "";

    } else {
        password_strength.style.color = colors[strength];

        password_strength.textContent = "Palavra-Passe:" +
    strength_values[strength];
    }
}
```

Foi também necessário fazer as seguintes alterações em html:

```
<div class="form-group">
                    <label class="label text-success" for="
    exampleInputPassword1"><b>Palavra-passe</b></label>
                    <input type="password" class="form-control"
    placeholder="Palavra-passe" id="password" onkeyup="
    password_Strength()">
                </div>

                <div class="form-group">
                    <a id="password_strength" style="color: black;"
    ></a>
                </div>
```

3.0.9 Issue 9

Verify that passwords 64 characters or longer are permitted but may be no longer than 128 characters.

Este issue está relacionado com o CWE 521 e tem o NIST: 5.1.1.2. #2.1.2.
Na realização do primeiro projeto, colocámos um limite máximo de 500 caracters, no entanto, este valor ao ser utilizado por um elevado número de utilizadores do website pode causar um aumento desnecessário de consumo de espaço na base de dados, que por si, pode levar a outros problemas.

Tendo em consideração que as passwords dos utilizadores já requerem um processo rigoroso de criação (todas as palavras-passe devem ter pelo menos uma letra maiúscula, uma minúscula, um número e um caracter especial, para além do número de caracters minimos que no caso do nosso website é 12), estas já seguem as normas de requesitos minimos e são por si fortes.

Sendo uma quantidade superior a 128 desnecessária e deteriorante para os recursos de backend do website.

Antes da implementação da nossa solução, tinhamos:

```
@cherrypy.expose
    def credentials_valid(self, credential, type='password'):
        if type == 'password':
            if len(credential) >= 12 and len(credential) <= 500 and
     re.search("[a-z]", credential) and re.search("[A-Z]",
    credential) and re.search("[0-9]", credential) and re.search("
    [#*]", credential):
                return True
            return False
```

Após a implementação da nossa solução, temos:

```
@cherrypy.expose
    def credentials_valid(self, credential, type='password'):
        if type == 'password':
            if len(credential) >= 12 and len(credential) <= 128 and
     re.search("[a-z]", credential) and re.search("[A-Z]",
    credential) and re.search("[0-9]", credential) and re.search("
    [#*]", credential):
                return True
            return False
```

Desta forma, a password do nosso website permite um cumprimento de 12 a 128 caracters.

3.0.10 Issue 10:

Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).

Este issue está relacionado com o CWE 521 e tem o NIST: 5.1.1.2. #2.1.1. Na realização do primeiro projeto, o nosso website garantia um limite minimo de caracteres para a palavra-passe de 12 caracters. No entanto, não tinha em consideração que um utilizador pudesse colocar todos os caracters como espaços e colocar de seguida apenas os requistos minimos para formar a palavra-passe.

Se o utilizador fizer isto, vai reduzir o grau de segurança da sua palavra-passe de forma significativa. Para evitar isto, nesta nova versão melhorada, todos os espaços seguidos que um utilizador colocar na criação da sua palavra-passe contaráo apenas como um caracter. Desta forma, é melhorada a consistencia de dados, prevenção de erros e é promovida a segurança da conta do utilizador.

Antes da implementação da nossa solução, tinhamos:

```
@cherrypy.expose
    def credentials_valid(self, credential, type='password'):
        if type == 'password':
            if len(credential) >= 12 and len(credential) <= 12 and
    re.search("[a-z]", credential) and re.search("[A-Z]",
    credential) and re.search("[0-9]", credential) and re.search("
    [#*]", credential):
                return True
            return False
```

Após a implementação da nossa solução, temos:

```
 @cherrypy.expose
    def credentials_valid(self, credential, type='password'):
        def correct_multiple_spaces(credential):
            return ' '.join(credential.split())

        if type == 'password':
            credential = correct_multiple_spaces(credential)
            if len(credential) >= 12 and len(credential) <= 128 and
     re.search("[a-z]", credential) and re.search("[A-Z]",
    credential) and re.search("[0-9]", credential) and re.search("
    [#*]", credential):
                return True
            return False
        elif type == 'username':
            if len(credential) >= 10 and len(credential) <= 30:
                return True
            return False
```

Desta forma, a utilização de espaços consecutivos passa a contar como um caracter no processo de registo na loja do DETI.

Capítulo 4

Software features

Neste segundo projeto, como já foi mencionado na introdução, apresentamos duas funcionalidades de software que transmitem importância para a segurança do nosso website.

As funcionalidades de software que escolhemos implementar no nosso projeto foram o Multi-factor Authentication (MFA), com recurso a códigos gerados por TOTP como segundo fator de autenticação, e também a feature de Password Strength Evaluation, com recurso a uma API externa (HIBP) para testar a vulnerabilidade passada de alguma password.

A nossa escolha relativamente a estas funcionalidades deve-se à sua relevância perante o tema do nosso projeto e do trabalho previamente desenvolvido, uma vez que apresentam melhorias na segurança ao conteúdo já desenvolvido e trabalhado no primeiro projeto, em que a autenticação dá acesso a basicamente todas as features do nosso website.

4.0.1 Software feature 1

A primeira funcionalidade de software implementada foi a Multi-factor Authentication.

Nesta funcionalidade, apresentamos um passo extra no processo de login do nosso site. Assumindo que é a primeira vez que um utilizador entra no website, este deve registar-se na conta colocando um username, password, confirmação de password e email.

Após concluir o processo de inserção de credenciais de registo no website, e caso as credenciais sejam aceites, o utilizador será apresentados, uma só vez apenas, com um código QR. Se já não o tiver feito, o utilizador deverá instalar uma aplicação baseada em geração de código TOTP, tal como a aplicação "Google Authenticator", pois será um passo fundamental no processo de login no site, sem o qual não será possível entrar no mesmo. Após a instalação, o utilizador deverá scannear o código QR presente no website do DETI, de forma a obter os códigos de acesso secreto temporários. Após isto, deve selecionar no website o botão "login instead". Aqui, poderá entrar no site através das suas

credenciais (como na versão inicial), no entanto deve usar o código secreto, que é temporário, sendo alterado a cada 30 segundos (que é gerado em função do tempo e de um segredo (uma string de 20 bytes) que é criado de forma aleatória no momento em que as credenciais do utilizador são aceites) de forma a permitir a entrada no site.

A geração dos segredos, bem como dos códigos correspondentes, é feita utilizando as funções oferecidas na biblioteca pyotp, que permitem a geração de códigos baseados no algoritmo TOTP:

```
def generate_totp_secret():
    #gerar segredo aleat rio de 20 bytes
    random_bytes = secrets.token_bytes(20)

    #codificar o segredo gerado com base32
    base32_encoded_secret = base64.b32encode(random_bytes).decode('
    utf-8')

    return base32_encoded_secret




def generate_totp(secret):
    #gerar c digo TOTP com base no segredo e no tempo atual
    totp = pyotp.TOTP(secret)
    return totp.now()
```

Estas funções são usadas para criar o segredo associado a um determinado utilizador no memento em que ele cria a conta e para obter o código TOTP correspondente a esse segredo e ao tempo atual quando o utilizador tenta fazer login

No entanto, todos estes cálculos são feitos no servidor e não saem de lá; o segredo é guardado na base de dados, na entrada correspondente à conta do utilizador quando este a cria, e só é acedido/lido quando o utilizador tenta autenticar-se, sendo chamada a função que cria o código TOTP com segredo como único parâmetro. O resultado será usado para verificar se o código resultante iguala o código que o utilizador inseriu para se verificar, que leu do Google Authenticator. Mas como é que o utilizador consegue ver este código no Google Authenticator?

```
def generate_totp_qr_code(account_name, secret):
    # Create a TOTP object
    totp = pyotp.TOTP(secret)

    # Generate the TOTP URI for the QR code
    totp_uri = totp.provisioning_uri(name=account_name, issuer_name
    ='LojaDoDeti')

    # Create a QR code for the TOTP URI
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_L,
        box_size=10,
        border=4,
    )
    qr.add_data(totp_uri)
    qr.make(fit=True)

    # Create an image from the QR code
    img = qr.make_image(fill_color="black", back_color="white")

    img_bytes = io.BytesIO()
    img.save(img_bytes, format='PNG')
    img_bytes = img_bytes.getvalue()

    return totp.now(), img_bytes
```

Esta função consegue criar, a partir do segredo, um código QR que o representa de forma visual. Este código QR terá que ser lido pelo dispositivo do utilizador que contém a app do Google Authenticator para poder gerar os mesmos códigos, em função do tempo, que o servidor gera quando o utilizador quer autenticar-se; tanto o servidor como o Google Authenticator possuem uma representação do segredo, sendo que a representação por extenso nunca sai do servidor, e a representação visual do mesmo é apresentada uma única vez ao cliente, na página web, para esta scannar e/ou guardar o código QR para gerar os códigos TOTP.

O segredo e o QR code são criados aquando o registo, e o segredo é guardado na base de dados, numa entrada referente ao utilizador que acabou de criar conta, e o QR code correspondente é enviado para a webpage do cliente:

```
            ...
            #criar segredo
            totp_secret = totp.generate_totp_secret()
            totp_code, totp_qr_code = totp.generate_totp_qr_code(username, totp_secret)

            #criar conta na base de dados
            db = sql.connect('base_dados.db')
            db.execute("INSERT INTO accounts(username, password, email, secret) VALUES (?, ?, ?, ?)", (username, password, email, totp_secret))
            db.commit()
            db.close()
            return json.dumps({"result" : "Conta criada. Leia o código QR que vai aparecer de seguida com o Google Authenticator:", "totp_qr_code": base64.b64encode(totp_qr_code).decode()}).encode("utf-8")
            ...
```

Após o registo, segue-se o login, em que se calcula o código TOTP a partir do segredo e do tempo, e que deve ser igual ao código TOTP dado pelo Google Authenticator após a leitra do código QR correspondente:

```
        ...
        result = db.execute("SELECT * FROM accounts WHERE username = ?", (username,))
        linha = result.fetchone()
        db.close()

        if linha == None:
            return json.dumps({"result" : "Nome de utilizador ou palavra-passe incorretos."}).encode("utf-8")

        account_info = dict()
        account_info={"id": linha[0], "username": linha[1], "password":linha[2], "mail": linha[3], "secret": linha[4]}

        #gerar totp a partir do segredo guardado na base de dados relativo ao username providenciado
        totp_code_db = totp.generate_totp(account_info['secret'])

        if account_info["username"].lower() == username.lower() and account_info["password"]==password and totp_code==totp_code_db:
            #adicionar sessão/cookies
            session = cherrypy.session
            ...
```

De modo a concluir a implementação desta feature, segue-se o excerto de código responsável por mostrar de forma visual o QR code na webpage do cliente, usando javascript:

```
        ...
        xhr.onreadystatechange = () => {
```

```
3             if (xhr.readyState === XMLHttpRequest.DONE && xhr.
    status === 200) {
4                 // Request finished. Do processing here.
5                 response = JSON.parse( xhr.response );
6                 alert(response.result)
7                 var image = new Image();
8                 image.src = ' '+response.
    totp_qr_code;
9                 document.getElementById("QR_image").innerHTML = '';
10                document.getElementById("QR_image").append(image)
11            }
12        }
13        ...
```

O funcionamento desta feature por completo, incluindo a criação de conta e o login (processos que foram alterados desde a última entrega, e que já não partilham os mesmos campos e são diferenciados visualmente por butões diferentes, e têm agora os seus campos próprios, embora a transição dê a ilusão que partilham alguns campos) poderão ser vistos no vídeo que enviámos como anexo do projeto.

4.0.2 Software feature 2

A segunda funcionalidade de software implementada foi a Password strength evaluation.

Esta funcionalidade, garante um aumento de segurança na criação de uma palavra-passe no nosso website. A sua função é garantir que a palavra-passe escolhida pelo utilizador é muito forte e prevenir que sejam escolhidas passwords vulneráveis através de verificação de falhas "breach verification"usando um serviço externo, sendo usado neste caso a API HIBP (Have I been Pwned), dificultando o processo de possiveis atacantes obterem a password de um dos utilizador da plataforma de merchandising do DETI.

A verificação da força da password é feita tanto server-side como client-side, sendo que apenas no client-side é usado um medidor de força, que vai de Muito Fraco a Muito Forte.

No servidor, apenas se verifica se a passeword é Muito Forte, pois apenas estas passwords devem ser admitidas. Eis a função do servidor que faz esta verificação:

```
    #verificar se as cred ncias s o v lidas (password e username
    )
    @cherrypy.expose
    def credentials_valid(self, credential, type='password'):
        # verificar se a password cumpre os nossos requisitos
        if type == 'password':
            if len(credential) >= 12 and len(credential) <= 128 and
     re.search("[a-z]", credential) and re.search("[A-Z]",
    credential) and re.search("[0-9]", credential) and re.search("
    [#*]", credential):
                return True
            return False

        # verificar se o username cumpre os nossos requisitos
        elif type == 'username':
            if len(credential) >= 10 and len(credential) <= 30:
                return True
            return False
```

Este função é usada tanto na criação de nova conta, como na mudança de password, uma feature que implementámos na fase anterior deste projeto:

```
        #verificar se a password cumpre os requisitos m nimos
        if not self.credentials_valid(new_pw, 'password'):
            return json.dumps({"result" : "Erro: a password n o
    cumpre os requisitos m nimos.", "status": False}).encode("utf
    -8")
```

Do lado do cliente, as funções são mais complexas, pois queremos que seja analisada a força da password sempre que esta é modificada aquando a sua escrita, apresentando a par do texto que declara a força atual da password uma cor para o próprio de acordo com um código de cores:

```
function password_Strength() {
    let password = document.getElementById("password1").value;
    let password_strength = document.getElementById("
    password_strength");


    var strength = -1;
    const strength_values = ["Muito Fraca", "Fraca", "M dia", "
    Forte", "Muito Forte"];
    const colors = ["red", "orange", "gold", "green", "darkgreen"];


    if (password.match(/[a-z]/)) {
        strength++;
    }

    if (password.match(/[A-Z]/)) {
        strength++;
    }

    if (password.match(/[0-9]/)) {
        strength++;
    }

    if (password.match(/[#*]/)) {
        strength++;
    }


    if (password.length >= 12 && password.length <= 128) {
        strength++;
    } else if (password.length > 128) {
        strength = -1;

    }

    if (strength > colors.length - 1) {
        strength = colors.length - 1;
    }


    if (strength == -1) { //caso a password    apagada ou tenha
    mais de 128 caracteres
        password_strength.textContent = "";

    } else {
        password_strength.style.color = colors[strength];

        password_strength.textContent = "Palavra-Passe:" +
    strength_values[strength];
    }
}
```

Esta função javascript mostra e altera a cor do texto que informa o cliente da força atual da password que está a criar. A função é chamada sempre que a password é alterada, tal como mostra esta linha de código html:

```
<input type="password" class="form-control" placeholder="Palavra-passe" id="password1" onkeyup="password_Strength()">
```

As cores também ajudam a perceber o nível de força da palavra-passe sem ser preciso sequer ler o texto. Cada nível de força tem associado uma cor, e abaixo segue-se o esquema cor-força correspondente:

- Vermelho - Muito fraca: O nível mais baixo de segurança.
- Laranja - Fraca
- Dourado - Média
- Verde - Forte
- Verde escuro - Muito forte: O nivel mais alto de segurança.

Por fim, caso a palavra-passe não seja forte o suficiente, e o utilizador a tentar submeter, será apresentada a seguinte mensagem de ajuda de criação de password:

```
function restricoes() {
    alert("A palvavra-passe dever  ter pelo menos 12 carateres, com as seguintes regras:" +
        " Pelo menos uma letra min scula," +
        " uma letra mai scula," +
        " um n mero" +
        " e pelo menos um  destes carat res especiais: # *");
}

function Utilizador() {
    alert("O nome do utilizador, dever  ter 10 a 30 carateres.");
}
```

O medidor de força de palavras chaves é extremamente útil para um utilizador saber se a palavra-passe que está a criar é segura, sendo que apenas palavras-passes Muito Fortes são admitidas para passarem à próxima etapa: a verificação da sua presença ou ausência na API HIBP.

Várias palavras-passes têm sido expostas ao longo dos anos, o que aumenta a sua probabilidade voltarem a ser expostas e de comprometerem as contas de utilizador na web, pois costumam ser utilizados em ataques de força bruta. Em 2021, um utilizador num fórum de hackers criou um post titulado RockYou2021 (em referência ao massivo data breach que ocorreu em 2009 denominado por RockYou), em que o anexo apresentava um documento de texto de quase 100GBs, que continha... 84 mil milhões de palavras-passe expostas ao longo dos anos. Como tal, é imperativo nos dias de hoje, durante o processo de criação de conta, verificar se a palavra-passe que estamos a inserir, por muito segura que pareça, já foi exposta alguma vez, o que abre caminho para que seja novamente exposta (na verdade não abre caminho, pois este já foi tragicamente aberto).

A nossa aplicação faz esta verificação aquando a criação de uma conta e aquando a mudança de password, sendo apenas aceitas passwords que nunca foram expostas. Eis a função que desenvolvemos para fazer esta essencial verificação:

```
1    #ver se a password submetida pelo user j  levou leak
     anteriormente (de acordo com a API do Have I Been Pwned)
2    def password_breach_verification(self,password):
3        #criar hash da password usando SHA-1 (tal como a API Have I
      Been Pwned exige (...Each password is stored as both a SHA-1
     and...))
4        hash = hashlib.sha1(password.encode('utf-8')).hexdigest().
     upper()
5
6        #enviar apenas os 5 primeiros caract res da hash para a
     API e depois verificar se os restantes caract res se encontram
      em alguma das hashs que a API devolver (modelo K-anonymity, em
      que envia-se apenas por  es da hash e nunca ela inteira,
     para n o levar esta leak a meio da verifica  o e fazer mais
     mal que bem)
7        prefix = hash[:5]
8        suffix = hash[5:]
9
10       #fazer o request para a API e verificar se o sufixo aparece
      nos resultados obtidos (que t m todos o prefixo enviado). Se
     aparecer,    porque a palavra passe levou breach
11       response = requests.get(f'https://api.pwnedpasswords.com/
     range/{prefix}')
12       return suffix in response.text
```

A password que o utilizador submeter é igualmente submetida a uma processo de hashing, sendo feita uma hash SHA-1 da password (tal como a API HIBP exige). Decidimos optar por boas práticas e não enviar a hash por inteiro para a API, pois tal poderia expor na transição a hash de uma password perfeitamente segura até então, fazendo mais mal que bem. Optámos por seguir o modelo K-anonymity, e enviamos apenas uma pequena porção da hash, um prefixo de 5 caractéres apenas, para que no caso de levar leak, se trate dum dissabor relativamente inofensivo.
Usando a libraria requests do python, o nosso servidor cherrypy envia um GET request (usando o url indicado na página https://haveibeenpwned.com/API/v3PwnedPasswords)

para a API HIBP, sendo obtida uma lista de hashs que partilham o mesmo prefixo que a nossa hash. Basta agora verificar se o sufixo, -os restantes caractéres que não foram enviados-, também fazem parte de alguma destas hashes dadas pela API. Se o sufixo fizer parte de pelo menos uma destas hashes, é porque infelizmente a nossa password levou leak no passado.

Uso da verificação de breach de uma potencial nova password no processo de mudança de password:

```
1         ...
2         #verificar se a nova password j   levou leak
3         if self.password_breach_verification(new_pw):
4             return json.dumps({"result" : "Erro: a nova password
      insegura, por favor submeta outra.", "status": False}).encode(
      "utf-8")
5         ...
```

Capítulo 5

Conclusões

Com este projeto aprendemos a utilizar o ASVS como uma ferramenta de implementação de requisitos importantes no processo de segurança de websites. Aprendemos como identificar, reconhecer e implementar melhorias nas falhas de segurança da nossa aplicação através da análise de requisitos pré-definidos. Aprendemos a implementar um processo de autenticação com 2 fatores recorrendo a códigos TOTP, tendo sido gratificante conseguirmos efetuar com sucesso um login usando o código que obtivémos no Google Authenticator após ler um código QR cuja aparição na nossa página web também foi muito satisfatória. Aprendemos a criar um medidor de força de passwords, que é uma ferramenta útil para as criar, e também verificámos se as nossas passwords usuais foram, em outrora, expostas ou não (e sim, foram). Este trabalho promoveu também o trabalho em equipa. Desde a análise de Issues, divisão de tarefas de forma eficiente, implementação dos critérios de segurança, e implementação das funcionalidades de software.

Contribuições dos autores

Percentagem	das	contribuições	do	Projeto
David Palricas	Eduardo Alves	Inês Silva	João Alcatrão	Mariana Silva
20%	20%	20%	20%	20%

Todos os membros do grupo contribuiram de forma igual para a Avaliação.

Bibliografia

OWASP ASVS checklist for audits:
https://github.com/shenril/owasp-asvs-checklist

OWASP Application Security Verification Standard:
https://owasp.org/www-project-application-security-verification-standard/

Have I Been Pwned API:
https://haveibeenpwned.com/API/v3