



universidade
de aveiro

Relatório de SIO Projecto Nº 2 - Application Security Verification Standard

Universidade de Aveiro

João Pedro Nunes Vieira

José Miguel Guardado Silva

Henrique Miguel Escudeiro Cruz

Luís Manuel Trindade Diogo

Relatório de SIO

Projecto Nº 2 - Application Security

Verification Standard

Departamento de Electrónica Telecomunicações e Informática

Universidade de Aveiro

João Pedro Nunes Vieira, Nº Mec.: 50458

joaopvieira@ua.pt

José Miguel Guardado Silva, Nº Mec.: 103248

jm.silva@ua.pt

Henrique Miguel Escudeiro Cruz, Nº Mec.: 103442

henriquecruz@ua.pt

Luís Manuel Trindade Diogo, Nº Mec.: 108668

luismtd@ua.pt

Resumo

Este relatório aborda a auditoria e implementação de correções de software para uma loja online desenvolvida como parte do Projeto Nº1 (https://github.com/detiuaveiro/1st-project-group_26) da Unidade Curricular de Segurança Informática e nas Organizações (SIO), no decorrer do ano letivo 2023/24.

O website é uma loja online destinada à venda de merchandise do Departamento de Eletrónica, Telecomunicações e Informática (DETI) da Universidade de Aveiro (UA), cujo objetivo principal é melhorar a funcionalidade e a segurança da loja original, seguindo os procedimentos definidos no **Application Security Verification Standard (ASVS)** Nível 1.

A abordagem do projeto inclui:

1. **Auditoria do Website:** Realização de uma auditoria de conformidade com os requisitos do Nível 1 do ASVS.
2. **Implementação de Melhorias de Segurança:** Abordagem das questões identificadas durante a auditoria para cumprir os requisitos do Nível 1 do ASVS.
3. **Documentação Detalhada:** Elaboração de documentação detalhada sobre o impacto dos problemas identificados e das correções implementadas.
4. **Novas Funcionalidades Implementadas:**
 - **Password strength evaluation:** exigindo um mínimo de força para passwords, de acordo com V2.1 e com verificação de violação recorrendo a um serviço externo.
 - **Autenticação de Dois Fatores (MFA):** utilizando Google Identity baseado em OAuth 2.0 com OpenID Connect (OIDC).

Este projeto alcançou com sucesso os objetivos propostos, destacando-se a realização de uma auditoria abrangente, a implementação eficaz das melhorias de segurança identificadas e a introdução bem-sucedida de novas funcionalidades. O repositório https://github.com/detiuaveiro/2nd-project-group_26 contém todos os elementos necessários para uma compreensão completa do processo.

Índice

1	Introdução	1
2	Auditoria	1
2.1	Recursos Utilizados na Auditoria	1
2.2	Vulnerabilidades Identificadas	1
2.3	Seleção de Vulnerabilidades	3
3	Correcção: Mitigação de Vulnerabilidades Detectadas	1
3.1	Correcções Prioritárias:	1
3.2	Correcções Secundárias:	6
4	Software Features	9
4.1	Password Strength Evaluation	9
4.2	Multi-factor Authentication (MFA)	10

Lista de Figuras

1	Resultado da Auditoria: Validity Percentage	2
2	Resultado da Auditoria: Results Tabel	2
3	Detecção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	1
4	Correcção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	2
5	Detecção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	2
6	Correcção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	2
7	Detecção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	3
8	Correcção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)	3
9	Exemplo de Detecção - Requisito: 4.0.2-8.2.1 - (app_sec.py, rota signup)	3
10	Correcção - Requisito: 4.0.2-8.2.1 - (app_sec.py, rota _signup)	4
11	Correcção do Requisito: 4.0.2-14.3.2 - (app_sec.py)	5
12	Exemplo da correcção do requisito: 4.0.2-14.4.3 - (app_sec.py)	6
13	Correcção - Requisito: 4.0.2-2.1.1 e 2.1.2 - (app_sec.py)	6
14	Correcção - Requisito: 4.0.2-2.1.1 e 2.1.2 - (app_sec.py)	7
15	Correcção - Requisito: 4.0.2-2.1.7 - (app_sec.py)	7
16	Correcção - Requisito: 4.0.2-2.1.8 - (RegisterPage.js)	7
17	Correcção - Requisito: 4.0.2-2.1.8 - (RegisterPage.js)	7
18	Correcção - Requisito: 4.0.2-8.3.3	8
19	Função de verificação da password - (app_sec.py)	9
20	Função lookup_pwned_api_password - (app_sec.py)	10
21	Função responsável pela obtenção do JSON Web Token (JWT) Token	11
22	Função responsável pelo envio do JWT Token para o backend	11
23	Rota responsável pela verificação do JWT Token e o login do utilizador	12
24	Função verify_google_token	12

Acrónimos

UA Universidade de Aveiro

DETI Departamento de Eletrónica, Telecomunicações e Informática

SIO Segurança Informática e Nas Organizações

ASVS Application Security Verification Standard

SIO Segurança Informática e nas Organizações

MFA Multi-factor Authentication

JWT JSON Web Token

OIDC OpenID Connect

Capítulo 1

Introdução

O ASVS representa um guia essencial de requisitos e testes de segurança para aplicações, destacando-se como uma ferramenta crucial para profissionais que se dedicam ao desenvolvimento de soluções informáticas seguras. Este projeto tem como objetivo principal aperfeiçoar e melhorar a loja online original do DETI por forma a cumprir os requisitos estabelecidos pelo ASVS de nível 1.

Assim, será conduzida uma auditoria de segurança abrangente, resultando na implementação de melhorias e na introdução de novas funcionalidades. A análise de segurança será orientada recorrendo ao ASVS usando os seguintes recursos:

- **ASVS Checklist:** Compatível com ASVS version 4.0.2, em <https://github.com/shenril/owasp-asvs-checklist>.
- **Revisão e Análise Manual de código fonte.**
- **Ferramentas de pesquisa automática (*automatic penetration tests*):**
 - **OWASP ZAP (*Zed Attack Proxy*):** Ferramenta de segurança open source para testar a segurança de aplicações web, para identificar vulnerabilidades no decorrer do desenvolvimento.
 - **Bandit:** Ferramenta de segurança open source focada em identificar vulnerabilidades em código Python, analisando o código fonte relativamente a boas práticas e potenciais vulnerabilidades.
 - **Snyk:** Ferramenta de segurança que identifica vulnerabilidades em bibliotecas de software open-source, com suporte para diversas linguagens de programação, oferecendo ainda soluções para mitigar os riscos inerentes a detecções.
 - **Nikto:** Ferramenta de código aberto que testa a segurança de servidores web, identificando potenciais vulnerabilidades e/ou configurações incorretas.
 - **Safety:** Ferramenta para Python developers que verifica e alerta sobre possíveis vulnerabilidades em bibliotecas *3rd party*, melhorando a segurança informática em projetos desenvolvidos com Python.

Os problemas e vulnerabilidades identificadas/detectadas pelos recursos anteriormente mencionados serão corrigidos, mitigando o seu impacto na aplicação web, tendo em consideração e prioridade as 10 mais importantes/urgentes (*High Priority*), de acordo com *OWASP Top Ten*. Além disso, serão incorporadas duas novas funcionalidades de software que pretendem melhorar a segurança da loja online:

1. ***Multi-factor Authentication (MFA):*** Utilizando OAuth2.0 em conjunto com a autenticação fornecida pelos serviços *Google Identity*
2. ***Password strength evaluation:*** Exigindo um nível mínimo de força e o uso de um serviço externo para verificação da integridade da password.

Capítulo 2

Auditoria

No seguimento do projecto, foi conduzida uma auditoria abrangente com o objetivo de avaliar a segurança da loja online do DETI com os requisitos estabelecidos pelo ASVS de nível 1. A auditoria consistiu numa série de testes automáticos e uma verificação manual do código fonte, permitido assim identificar potenciais vulnerabilidades e problemas de segurança.

2.1 Recursos Utilizados na Auditoria

A análise de segurança foi orientada recorrendo ao ASVS e utilizando a seguinte lista de recursos mencionados na Introdução 1:

- **ASVS Checklist**
- **Revisão e Análise Manual de Código Fonte**
- **Ferramentas de Pesquisa Automática:**
 - *OWASP ZAP (Zed Attack Proxy)*
 - *Bandit*
 - *Snyk*
 - *Nikto*
 - *Safety*

2.2 Vulnerabilidades Identificadas

As vulnerabilidades identificadas pelos recursos anteriormente mencionados, foram catalogadas na **ASVS Checklist**, documento ASVS-CHECKLIST-EN.ODS, dando assim uma visão intuitiva e detalhada do estado de segurança da aplicação e um conjunto de resultados obtidos, como podemos verificar nas figuras 1 e 2:

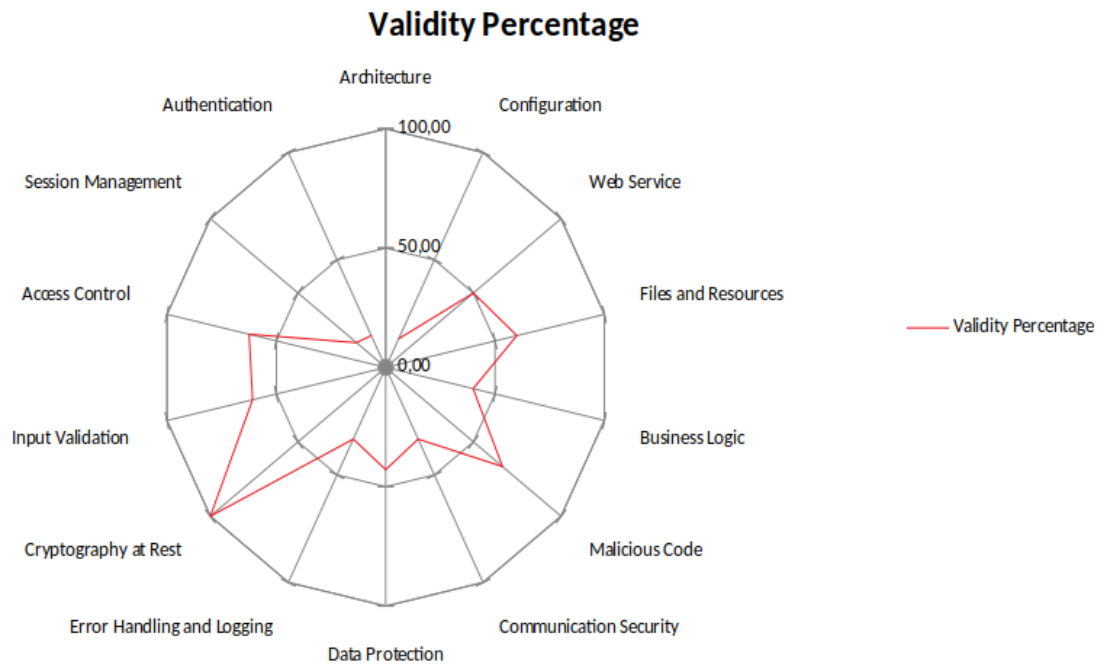


Figura 1: Resultado da Auditoria: Validity Percentage

Security Category	Valid criteria	Total criteria	Validity Percentage	ASVS Level Acquired
Architecture	0	0	#DIV/0!	
Authentication	4	27	14,81	
Session Management	2	12	16,67	
Access Control	5	8	62,50	
Input Validation	14	23	60,87	
Cryptography at Rest	1	1	100,00	Level 1
Error Handling and Logging	1	3	33,33	
Data Protection	3	7	42,86	
Communication Security	1	3	33,33	
Malicious Code	2	3	66,67	
Business Logic	2	5	40,00	
Files and Resources	3	5	60,00	
Web Service	3	6	50,00	
Configuration	2	15	13,33	
Total	43	118	36,44	

Figura 2: Resultado da Auditoria: Results Tabel

Como se pode analisar pelas imagens anteriores, o projecto original (presente na pasta **app_org** contém diversos problemas de segurança e na sua correção será dada prioridade às 10 questões mais importantes e urgentes, conforme definido pelo *OWASP Top Ten*. Também serão corrigidas as vulnerabilidades consideradas importantes pelos elementos responsáveis do grupo do projeto. No auxílio a essas correções serão implementadas duas novas funcionalidades (referidas em 1):

1. *Multi-factor Authentication (MFA)*
2. *Password Strength Evaluation*

2.3 Selecção de Vulnerabilidades

Após análise dos resultados demonstrados em reunião geral com os elementos do grupo do projeto, foram seleccionadas diversas vulnerabilidades que exigem uma correção prioritária, visto que constituem uma maior ameaça:

Nota: o símbolo * indica que vulnerabilidades (problema) está correlacionado de certa forma.

- 4.0.2-2.1.5
- 4.0.2-5.2.5 *
- 4.0.2-5.2.8 *
- 4.0.2-5.5.1 *
- 4.0.2-8.2.1
- 4.0.2-8.3.4
- 4.0.2-12.1.1
- 4.0.2-13.2.2
- 4.0.2-14.2.2
- 4.0.2-14.3.2 *
- 4.0.2-14.4.3

Para além das vulnerabilidades que exigem uma correção prioritária, foram seleccionadas algumas vulnerabilidades extra que irão também contribuir para a mitigação dos problemas de segurança detetados. "secundária":

- 4.0.2-2.1.1
- 4.0.2-2.1.2
- 4.0.2-2.1.7
- 4.0.2-2.1.8
- 4.0.2-2.1.12
- 4.0.2-3.2.3 *
- 4.0.2-3.3.1 *
- 4.0.2-3.4.1
- 4.0.2-3.4.2
- 4.0.2-3.4.3
- 4.0.2-3.4.4
- 4.0.2-5.1.5
- 4.0.2-8.3.3

- 4.0.2-14.3.3
- 4.0.2-14.4.2
- 4.0.2-14.4.5
- 4.0.2-14.4.6

Capítulo 3

Correcção: Mitigação de Vulnerabilidades Detectadas

3.1 Correcções Prioritárias:

Requisito: 4.0.2-2.1.5

O 5º requisito ASVS do capítulo 2 ("*Authentication*"), secção 1 ("*Password Security Credentials*") foi detetado através da análise manual do código. Na deteção foram identificadas falhas no sistema de alteração de passwords para utilizadores verificados. Estas falhas não permitiam a realização desta ação e foram corrigidas.

Requisito: 4.0.2-5.2.5

O 5º requisito ASVS do capítulo 5 ("*Input Validation*"), secção 2 ("*Sanitization and Sandboxing Requirements*") foi detectado através da ferramenta automática, Bandit (Python), no ficheiro **app_sec.py**, como se pode visualizar na Figura 3:

```
420 if __name__ == "__main__":
421     app.run(debug=True)
422
423 |
```

Figura 3: Deteção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

De forma a proteger ainda mais a aplicação contra "*template injection attacks*" e por forma a solucionar a CWE-94, "*CWE-94: Improper Control of Generation of Code ('Code Injection')*", foi efectuada a seguinte correcção de código Backend (Figura 4):

```
533
534 if __name__ == "__main__":
535     app.run(debug=False, ssl_context="adhoc")
536
```

Figura 4: Correção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

Neste caso, debug deve estar desactivado para evitar injeção de código arbitrário de um potencial atacante.

Requisito: 4.0.2-5.2.8

O 8º requisito ASVS do capítulo 5 (*"Input Validation"*), secção 2 (*"Sanitization and Sandboxing Requirements"*) foi detectado através da ferramenta automática, Bandit (Python), no ficheiro **app_sec.py**.

```
420 if __name__ == "__main__":
421     app.run(debug=True)
422
423
```

Figura 5: Detecção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

De forma a proteger ainda mais a aplicação contra por forma a que os conteúdos fornecidos aos utilizadores seja restringidos (ou desactivados), e por forma a solucionar a CWE-94, *"CWE-94: Improper Control of Generation of Code ('Code Injection')"*, foi efectuada a seguinte correção de código Backend (Figura 6):

```
533
534 if __name__ == "__main__":
535     app.run(debug=False, ssl_context="adhoc")
536
```

Figura 6: Correção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

Neste caso, debug deve estar desactivado para evitar injeção de código arbitrário de um potencial atacante.

Requisito: 4.0.2-5.5.1

O 8º requisito ASVS do capítulo 5 (*"Input Validation"*), secção 2 (*"Sanitization and Sandboxing Requirements"*) foi detectado através da análise manual do código Backend presente no ficheiro **app_sec.py**:

```
420 if __name__ == "__main__":  
421     app.run(debug=True)  
422  
423
```

Figura 7: Detecção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

Neste, por uma questão de segurança, deve ser ativado HTTPS com SSL, tal como demonstrado na Figura 8:

```
533  
534 if __name__ == "__main__":  
535     app.run(debug=False, ssl_context="adhoc")  
536
```

Figura 8: Correção Bandit - Requisito: 4.0.2-5.2.5 - (app_sec.py)

No entanto, esta alteração gera um certificado auto-assinado para fins de desenvolvimento, pelo que é importante no *deployment* alterar por forma a conter um certificado de uma Autoridade Certificadora (CA) de confiança, permitindo assim que seja reconhecido pelos browsers modernos mantendo confiança e segurança.

Requisito: 4.0.2-8.2.1

O 1º requisito ASVS do capítulo 8 ("*Data Protection*"), secção 2 ("*Client-side Data Protection*"), foi detectado através da análise manual do código Backend presente no ficheiro **app_sec.py**. Neste, não são definidos cabeçalhos "*anti-caching*" por forma a evitar que dados sensíveis sejam armazenados em cache nos browsers modernos, como se pode verificar na Figura 9:

```
@app.route('/signup', methods=['GET', 'POST'])  
def signup():  
    if request.method == 'POST':  
        fname = bleach.clean(request.form['fname'], strip=True)  
        lname = bleach.clean(request.form['lname'], strip=True)  
        email = bleach.clean(request.form['email'], strip=True)  
        nick = bleach.clean(request.form['nick'], strip=True)  
        password = bleach.clean(request.form['password'], strip=True)  
        confirm = bleach.clean(request.form['confirm'], strip=True)  
        nib = bleach.clean(request.form['nib'], strip=True)  
  
        if password != confirm:  
            flash("Passwords do not match.", 'error')  
        else:  
            existing_user = db_session.query(User).filter_by(email=email).first()  
            if existing_user:  
                flash("Email or nickname is already in use.", 'error')  
            else:  
                # Hash de passwords!!! :)  
                hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())  
                user = User(fname=fname, lname=lname, email=email, nick=nick, passwd=hashed_password, nib=nib, role="Customer")  
                db_session.add(user)  
                db_session.commit()  
                flash("Registration complete!", 'SUCESS')  
                return redirect('/login')  
  
    return render_template('signup.html')
```

Figura 9: Exemplo de Detecção - Requisito: 4.0.2-8.2.1 - (app_sec.py, rota signup)

Assim, por forma a corrigir este problema, foram implementadas respostas (responses) encapsuladas em cabeçalhos com instruções de controlo de cache, como podemos analisar na Figura 10:

```
30 # Route: Signup
31 @app.route('/signup', methods=['GET', 'POST'])
32 def signup():
33     if request.method == 'POST':
34
35         data = request.json
36         fname = bleach.clean(data['FName'], strip=True)
37         lname = bleach.clean(data['LName'], strip=True)
38         email = bleach.clean(data['Email'], strip=True)
39         nick = bleach.clean(data['Email'], strip=True)
40         password = bleach.clean(data['Password'], strip=True)
41         nib = data['Nif']
42
43         existing_user = db_session.query(User).filter_by(email=email).first()
44         if existing_user:
45             return jsonify(False)
46         if not password_requirements(password):
47             return jsonify(False), "Password does not match length requirements. Careful with consecutive spaces."
48         else:
49             hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt(rounds=16)) |
50             user = User(fname=fname, lname=lname, email=email, nick=nick, passwd=hashed_password, nib=nib, role="User")
51             db_session.add(user)
52             db_session.commit()
53
54             response = jsonify(True)
55             response.headers['Cache-Control'] = 'no-store, no-cache, must-revalidate, max-age=0'
56             response.headers['Pragma'] = 'no-cache'
57             response.headers['Expires'] = '0'
58
59             return response
60
```

Figura 10: Correção - Requisito: 4.0.2-8.2.1 - (app_sec.py, rota _signup)

A principal alteração foi a adição de cabeçalhos de controlo de cache (Cache-Control, Pragma e Expires) na resposta (response) encapsulada nestes. que indicam explicitamente que o conteúdo não deve ser armazenado, evitando assim persistência de dados sensíveis em caches não controladas de browsers, anulando a possibilidade das informações sejam "recuperadas" por utilizadores não autorizados (atacantes).

Requisito: 4.0.2-8.3.4

O 8º requisito ASVS do capítulo 14 ("*Data Protection*"), secção 3 ("*Sensitive Private Data*") foi detectado através da ferramenta automática OWASP ZAP, no ficheiro **app_inserts.py**. O não cumprimento deste requisito pode resultar na exposição de dados sensíveis.

Para resolver essa questão, foram eliminados todos os comentários e possíveis mensagens de erro que poderiam guiar utilizadores maliciosos a descobrir vulnerabilidades no website e aplicação.

É importante destacar que uma política de processamento de dados sensíveis já estava em vigor na versão anterior do projeto. Essa política consistia na aplicação de técnicas de criptografia para proteger os dados sensíveis durante o armazenamento, garantindo uma camada adicional de segurança.

Requisito: 4.0.2-12.1.1

O 1º requisito ASVS do capítulo 12 ("*Files and Resources*"), secção 1 ("*File Upload Requirements*", foi detectado através da ferramenta automática Snyk.

Esta vulnerabilidade afeta um pacote utilizado (@adobe/css-tools) afectando selectors de CSS inválidos

e manipulando os mesmos. Ao aceitar ficheiros grandes pode provocar uma exposição o website a ataques de Negação de Serviço (REDoS/DoS).

Para corrigir esta vulnerabilidade foram atualizadas as bibliotecas `testing-library/jest-dom` e `adobe/css-tools`, utilizadas no frontend, para as versões 5.17.0 e 4.3.2, respetivamente.

Requisito: 4.0.2-13.2.2

O 2º requisito ASVS do capítulo 13 ("*Web Services*"), secção 2 ("*RESTful Web Service Verification Requirements*") foi detectado através da ferramenta automática Snyk.

Esta vulnerabilidade foi corrigida pela atualização da biblioteca `"postcss"` para uma versão superior ou igual a 8.4.31, permitindo que os dados transmitidos em JSON estivessem no formato correto.

Requisito: 4.0.2-14.2.2

O 2º requisito ASVS do capítulo 14 ("*Configuration*"), secção 2 ("*Dependency*") foi detectado através da análise manual do código. O não cumprimento deste requisito pode levar a falhas de segurança.

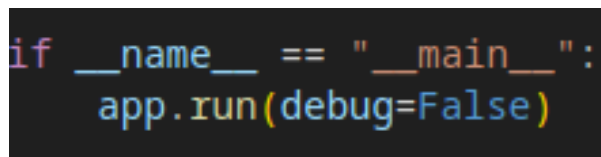
Para resolver essa questão, todos os utilizadores de teste e administradores foram removidos no processo de criação da base de dados. Essa medida visa garantir que informações sensíveis não estejam presentes em ambientes de produção.

A remoção dos administradores, em específico, do código também contribuiu para reforçar a segurança, uma vez que a presença de contas de administrador poderia representar um risco significativo. Se tais contas permanecessem no ficheiro `db_inserts`, podiam ser exploradas por atacantes que tentam obter privilégios não autorizados. Além disso, a existência de administradores não necessários aumenta o risco de ataques e pode resultar em ações não autorizadas que requerem privilégios elevados.

Requisito: 4.0.2-14.3.2

O 2º requisito ASVS do capítulo 14 ("*Configuration*"), secção 3 ("*Unintended Security Disclosure Requirements*") foi detectado através da análise manual do código, no ficheiro `app_sec.py`. O não cumprimento deste requisito pode resultar na exposição de dados sensíveis dos utilizadores, assim como vulnerabilidades presentes no website e na aplicação, devido à possível divulgação de mensagens de erro."

Para corrigir este problema, quando se pretende executar a aplicação Flask modo de debug encontra-se desativado, ou seja, com a opção definida como `False`.



```
if __name__ == "__main__":  
    app.run(debug=False)
```

Figura 11: Correção do Requisito: 4.0.2-14.3.2 - (`app_sec.py`)

Requisito: 4.0.2-14.4.3

O 3º requisito ASVS do capítulo 14 ("*Configuration*"), secção 4 ("*HTTP Security Headers Requirements*") foi detectado através da ferramenta automática OWASP ZAP. O não cumprimento deste requisito

deixa o website mais vulnerável a ameaças, como ataques JavaScript injection, Cross-Site Scripting, entre outros.

Para mitigar esta vulnerabilidade foram adicionados cabeçalho CSP a todas às respostas HTTP feitas pelo backend. Os cabeçalhos adicionados têm como por objetivo evitar problemas de segurança relacionados ao armazenamento em cache, garantido assim, que as respostas não serão armazenadas localmente nos dispositivos dos utilizadores.

A figura 12 é um exemplo de um cabeçalho adicionado a uma resposta de modo a obter a segurança pretendida.

```
def password_requirements(password):  
    if not password_requirements(password):  
        response = jsonify("Password does not match length requirements. Careful with consecutive spaces.")  
        response.headers['Cache-Control'] = 'no-store, no-cache, must-revalidate, max-age=0'  
        response.headers['Pragma'] = 'no-cache'  
        response.headers['Expires'] = '0'  
        return response
```

Figura 12: Exemplo da correção do requisito: 4.0.2-14.4.3 - (app_sec.py)

3.2 Correções Secundárias:

Requisito: 4.0.2-2.1.1 e 4.0.2-2.1.2

O 1º e 2º requisito ASVS do capítulo 2 ("*Authentication*"), secção 1 ("*Password Security Credentials*") foi detetado através da análise manual do código.

Para corrigir esta vulnerabilidade, no backend foi criada uma função que garanta que as passwords cumpram um conjunto de requisitos. Esses requisitos são:

1. Passwords têm obrigatoriamente de ter um tamanho superior a 12 caracteres. Múltiplos espaços consecutivos são contados como apenas um carácter
2. Passwords não podem ter um tamanho superior a 128 caracteres.

Aqui podemos ver como estes requisitos são verificados.

```
def password_requirements(password):  
    password = " ".join(password.split())  
    if len(password) < 12 or len(password) > 128:  
        return False  
    return True
```

Figura 13: Correção - Requisito: 4.0.2-2.1.1 e 2.1.2 - (app_sec.py)

Aqui podemos ver como a função é utilizada.

```
if not password_requirements(password):  
    response = jsonify("Password does not match length requirements. Careful with consecutive spaces.")
```

Figura 14: Correção - Requisito: 4.0.2-2.1.1 e 2.1.2 - (app_sec.py)

Requisito: 4.0.2-2.1.7

O 7º requisito ASVS do capítulo 2 ("*Authentication*"), secção 1 ("*Password Security Credentials*") foi detetado através da análise manual do código.

Esta vulnerabilidade foi corrigida aplicando a função `lookup_pwned_api_password` no backend que retorna se a password foi encontrada em data breaches ou não. Caso tenha sido, um popup irá alertar o utilizador desta vulnerabilidade e encaminhá-lo para a página de alteração de password.

```
def lookup_pwned_api_password(pwd):  
    shalpwd = hashlib.sha1(pwd.encode('utf-8')).hexdigest().upper()  
    head, tail = shalpwd[:5], shalpwd[5:]  
    url = 'https://api.pwnedpasswords.com/range/' + head  
    res = pyrequests.get(url)  
    if res.ok:  
        hashes = (line.split(':') for line in res.text.splitlines())  
        count = next((int(count) for t, count in hashes if t == tail), 0)  
        if count:  
            return True  
        else:  
            return False  
    else:  
        return False
```

Figura 15: Correção - Requisito: 4.0.2-2.1.7 - (app_sec.py)

Requisito: 4.0.2-2.1.8

O 8º requisito ASVS do capítulo 2 ("*Authentication*"), secção 1 ("*Password Security Credentials*") foi detetado através da análise manual do código.

```
<PasswordStrengthBar style={{width : "80%", margin: "10px 0 0 0"}} minLength={12} password={password} />
```

Figura 16: Correção - Requisito: 4.0.2-2.1.8 - (RegisterPage.js)

A correção desta vulnerabilidade baseou-se na adição de uma barra indicadora, que demonstra de forma visual e intuitiva a força da password que o utilizador deseja utilizar. O resultado desta implementação pode ser verificado pela figura 17.



Figura 17: Correção - Requisito: 4.0.2-2.1.8 - (RegisterPage.js)

Requisito: 4.0.2-2.1.12

O 12º requisito ASVS do capítulo 2 ("*Authentication*"), secção 1 ("*Password Security Credentials*") foi detetado através da análise manual do código.

Esta vulnerabilidade apenas ocorria na página de alteração de password. Portanto, foi mitigada nesta página pela aplicação de mecanismos anteriormente aplicados noutras páginas que envolvem passwords.

Requisito: 4.0.2-3.4.1

O 3º requisito ASVS do capítulo 3 ("*Session_Management*"), secção 4 ("*Cookie-based Session Management*") foi detetado através da análise manual do código.

Esta vulnerabilidade apenas ocorria na criação do cookie de sessão. Portanto, foi corrigida fornecendo ao cookie de sessão o parametro necessário para obter o paramtro fornecido.

Requisito: 4.0.2-3.4.2

O 3º requisito ASVS do capítulo 3 ("*Session_Management*"), secção 4 ("*Cookie-based Session Management*") foi detetado através da análise manual do código.

Esta vulnerabilidade apenas ocorria na criação do cookie de sessão. Portanto, foi corrigida fornecendo ao cookie de sessão o parametro necessário para obter o paramtro fornecido.

Requisito: 4.0.2-3.4.3

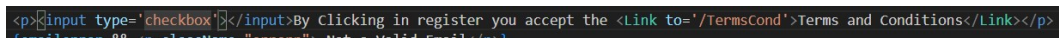
O 3º requisito ASVS do capítulo 3 ("*Session_Management*"), secção 4 ("*Cookie-based Session Management*") foi detetado através da análise manual do código.

Esta vulnerabilidade apenas ocorria na criação do cookie de sessão. Portanto, foi corrigida fornecendo ao cookie de sessão o parametro necessário para obter o paramtro fornecido.

Requisito: 4.0.2-8.3.3

O 3º requisito ASVS do capítulo 8 ("*Data Protection*"), secção 1 ("*Sensitive Private Data*") foi detetado através da análise manual do código. O não cumprimento deste requisito pode resultar em uma quebra de confiança entre o website e os clientes, pois estes não têm uma compreensão clara de como seus dados sensíveis podem ser utilizados.

Para lidar com esse problema, foram implementados "*Termos e Condições*". Todos os utillidores são obrigados a aceitar ou rejeitar esses termos. Dependendo da resposta do utilizador, o acesso ao website ou a outras funcionalidades pode ser concedido ou negado.



```
<p><input type='checkbox'></input>By clicking in register you accept the <Link to='/TermsCond'>Terms and Conditions</Link></p>
```

Figura 18: Correção - Requisito: 4.0.2-8.3.3

Ao criar "*Termos e Condições*" promovemos transparência com os utilizadores, uma vez que eles têm conhecimento de como os seus dados podem ser usados. Essa abordagem reflete um compromisso com a comunicação clara e a proteção da privacidade dos utilizadores.

Capítulo 4

Software Features

Foram implementadas duas funcionalidades de software com o objetivo de melhorar a segurança do website. As funcionalidades escolhidas para implementar foram:

- Password Strength Evaluation
- MFA com OAuth2.0 e Google Identity

4.1 Password Strength Evaluation

A necessidade de passwords fortes e seguras é essencial para a segurança do website e do utilizador. Para cumprir este requisito, foram aplicadas medidas imperativas para a criação ou alteração de uma password, tais como, a verificação de tamanho e variedade de caracteres usados, a avaliação da força da password e a verificação de correspondências com data breaches.

Para garantir uma password forte, na sua criação ou alteração é verificado se cumpre os critérios impostos. Esta operação pode-se observar pela figura 19.

```
def password_requirements(password):  
    password = " ".join(password.split())  
    if len(password) < 12 or len(password) > 128:  
        return False  
    return True
```

Figura 19: Função de verificação da password - (app_sec.py)

Finalmente, para a verificação de correspondências com data breaches, a função `lookup_pwned_api_password` foi implementada, fazendo uso da API do Have I Been Pwned. Esta função verifica se a password já foi descoberta em data breaches e consequentemente insegura. Também foi verificado que esta api é segura devido ao uso da propriedade k-anonymity, onde apenas é fornecido os primeiros 5 caracteres do hash da password para a api retornar uma lista de hashes de passwords. A partir dessa lista é procurado o hash completo igual ao hash da password a ser avaliada para então tirar conclusões. Esta função pode ser analisada pela figura20.

```
def lookup_pwned_api_password(pwd):
    shalpwd = hashlib.sha1(pwd.encode('utf-8')).hexdigest().upper()
    head, tail = shalpwd[:5], shalpwd[5:]
    url = 'https://api.pwnedpasswords.com/range/' + head
    res = pyrequests.get(url)
    if res.ok:
        hashes = (line.split(':') for line in res.text.splitlines())
        count = next((int(count) for t, count in hashes if t == tail), 0)
        if count:
            return True
        else:
            return False
    else:
        return False
```

Figura 20: Função lookup_pwned_api_password - (app_sec.py)

4.2 MFA

A autenticação de múltiplos fatores (MFA) é uma medida crucial para reforçar a segurança do acesso ao website, por parte de utilizadores. Optámos por implementar o OAuth2.0 com o Google Identity.

Google Identity é um serviço de gestão de identidade fornecido pelo Google. Permite que os utilizadores se autentiquem em diversas aplicações e serviços usando as suas credenciais do Google. Ao integrar o Google Identity no nosso website, beneficiamos da segurança robusta proporcionada pelo Google, incluindo verificações de autenticação em dois fatores e deteção de atividades suspeitas.

O OAuth2.0 é um protocolo de autorização utilizado por aplicações de modo a permitir que aplicações obtenham acesso a recursos em nome de utilizadores sem relevarem as suas credencias.

O processo de autenticação com MFA, usando o Google Identity e OAuth 2.0, segue os seguintes passos:

1. O utilizador seleciona a opção de login com Google no nosso website.
2. O website redireciona o utilizador para o serviço de autenticação do Google usando OAuth 2.0.
3. O utilizador introduz as suas credenciais da sua conta do Google.
4. Após a autenticação bem-sucedida da conta do utilizador, o Google Identity gera um JWT Token de acesso.
5. O frontend envia o JWT Token para o backend.
6. O backend verifica o token de acesso com o Google Identity para garantir a sua autenticidade. Caso essa autenticidade seja verificada o backend sinaliza o frontend criando assim a sessão do utilizador.

Esta abordagem proporciona uma camada adicional de segurança, garantindo que apenas utilizadores autorizados, que passaram pela autenticação em dois fatores, possam aceder ao website.

Envio do JWT Token do frontend para o backend.

```
const [ user, setUser ] = useState({});

function handleCallbackResponse(response) {
  var userObject = jwtDecode(response.credential);
  setUser(userObject);
  document.getElementById("signInDiv").hidden = true;
  logGoogle(response.credential).then(function (result) {
    if(result === 1) {
      navigatetoHome();
    }
  });
}
```

Figura 21: Função responsável pela obtenção do JWT Token

```
export async function logGoogle(JWT) {
  const res = await( await fetch('/login_google', {
    method: "POST",
    body: JSON.stringify(JWT),
    headers: {
      "content-type": "application/json"
    }
  })).json()

  return res
}
```

Figura 22: Função responsável pelo envio do JWT Token para o backend

De modo a verificar a autenticidade da JWT Token enviada pelo frontend recorreremos à função `verify_google_token`. Após essa validação realizamos o login do utilizador

```
@app.route('/login_google', methods=['GET', 'POST'])

# Limit Route Calls: 5 requests per hour, 20 requests per day
@limiter.limit("5/hour")
@limiter.limit("20/day")
def login_google():
    if request.method == 'POST':
        try:
            # Get Data from Frontend
            data = request.json # Token in Json format --> 'google_token': token
            google_token = data
            print(google_token)

            # Verify the token using Google's token verification endpoint
            id_info = verify_google_token(google_token)
            if not id_info: return jsonify(0) # Login Failed --> Token is Invalid

            # Extract user information from the token
            email = id_info['email']
            name = id_info['name']

            # Check if user exists
            user = db_session.query(User).filter_by(email=email).first()

            # Register in case user does not exist yet
            if not user:
                # Create a new user account if the user doesn't exist
                user = User(fname=name, lname="", email=email, nick=name, passwd="", nib=None, role="User")
                db_session.add(user)
                db_session.commit()

            return jsonify(1) # Login and/or Register Successful

        except ValueError as e:
            return False, f'Invalid token: {str(e)}'
```

Figura 23: Rota responsável pela verificação do JWT Token e o login do utilizador

Contactamos o Google Token verification endpoint para garantir a autenticidade do Token.

```
def verify_google_token(google_token):
    CLIENT_ID = "933478365143-rff11v0iq4vls8ouobltdou7b7mlbdfb.apps.googleusercontent.com"
    try:

        # Verify the token using Google's token verification endpoint
        id_info = id_token.verify_oauth2_token(google_token, requests.Request(), CLIENT_ID)

        return id_info

    except ValueError as e:
        # Token is invalid
        #f'Invalid token: {str(e)}'
        return False
```

Figura 24: Função verify_google_token

Conclusão:

Em conclusão, este projeto desempenhou um papel fundamental na elevação dos padrões de segurança da loja online de memorabilia do DETI. A auditoria de segurança de nível 1, baseada no ASVS proporcionou uma visão abrangente das vulnerabilidades presentes na aplicação, possibilitando a implementação de correções das mesmas.

A identificação e resolução das principais questões de segurança, delineadas pelo ASVS, não fortaleceram apenas a resistência da aplicação a potenciais ameaças, mas também contribuíram para a melhoria geral da integridade e confidencialidade dos dados dos utilizadores. A explicação detalhada do impacto das correções evidencia o compromisso com a construção de uma aplicação robusta e a sua segurança.

Além da correção das vulnerabilidades detetadas, provenientes da realização da auditoria, a adição de duas novas funcionalidades, Autenticação Multi-Fator (MFA) com OAuth 2.0 e Google Identity, juntamente com a avaliação da força de senha por meio de um serviço externo, aumentou significativamente a segurança da loja online, Detitalismo.

Este projecto ilustra claramente as consequências significativas no mundo real das práticas seguras de programação para aplicações e/ou soluções Web. A negligência em relação às boas práticas e segurança pode resultar em violações de dados e privacidade, perdas financeiras, furto de identidade e danos à reputação de uma organização ou utilizador comum.

Contudo, é fundamental referir que, apesar de todo o esforço e empenho disponibilizado neste projecto, ainda existem vulnerabilidades presentes, as quais podem ser usadas por um potencial atacante. Esta situação deve-se ao facto de, na opinião dos elementos dos grupo de trabalho, serem necessárias posteriores auditorias para corrigir todos os erros e vulnerabilidades presentes.

Assim, o grupo de trabalho gostaria de reforçar a ideia de que auditorias periódicas e uma constante vigilância são fundamentais para proporcionar aplicações web mais seguras e robustas contra ataques maliciosos, protegendo utilizadores e dados tendo em conta Confidencialidade, Integridade e Disponibilidade de serviços.

Contribuições dos Autores

Os autores responsáveis pela pesquisa e desenvolvimento do presente projecto, contribuíram de forma coletiva, refletindo-se diversidade de experiências, opiniões e competências que enriqueceram a abordagem ao mesmo. Este projecto pode ser visualizado na sua totalidade na página: https://github.com/detiuaveiro/2nd-project-group_26.

Autores:

- João Pedro Nunes Vieira, N^oMec.: 50458
- José Miguel Guardado Silva, N^o Mec.: 103248
- Henrique Miguel Escudeiro Cruz, N^o Mec.: 103442
- Luís Manuel Trindade Diogo, N^o Mec.: 108668

Tabela 1: Contribuições dos Autores.

Contribuição	João Vieira	José Silva	Henrique Cruz	Luís Diogo
Produção de Relatório (LaTex)	25 %	25 %	25 %	25 %
Correcções de Projecto iniciais	50 %	50 %	0 %	0 %
Identificação ASVS, Testes e Auditoria	80 %	0 %	20 %	0 %
Correcção de Vulnerabilidades	25 %	25 %	25 %	25 %
Implementação de 2 Novas Funcionalidades	0 %	10 %	40 %	40 %

Bibliografia

- [1] Allen Downey *Think Python - How to Think Like a Computer Scientist*. | Green Tea Press, 2nd Edition, Version 2.4.0, 2015
- [2] André Zúquete *Segurança em Redes Informáticas*. | FCA - Editora de Informática LDA, 5th Edition, 2018
- [3] Wikipédia: Enciclopédia livre. | pt.wikipedia.org, acedido 04/12/2023.
- [4] SQLite Website. | <https://www.sqlite.org/docs.html>, acedido 09/12/2021.
- [5] SQLAlchemy Website. | <https://docs.sqlalchemy.org/en/20/>, acedido 09/12/2021.
- [6] MITRE: A Community-Developed List of Software and Hardware Weakness Types. | <https://cwe.mitre.org/index.html>, acedido 04/10/2021.
- [7] ASVS Checklist - Compatível com ASVS version 4.0.2 <https://github.com/shenril/owasp-asvs-checklist>, acedido a 21/12/2023
- [8] OWASP: Open Web Application Security Project <https://owasp.org/www-project-application-security-verification-standards/>
- [9] OWASP: Open Web Application Security Project - TOP 10 <https://owasp.org/www-project-top-ten/>, acedido a 27/12/2023
- [10] OWASP: Open Web Application Security Project - Zed Attack Proxy. | <https://owasp.org/www-project-devsecops-guideline/latest/01b-Linting-Code>, acedido a 28/12/2023