



Projeto 2 – DetiShop (ASVS)

SEGURANÇA INFORMÁTICA E NAS ORGANIZAÇÕES



Alexandre Martins, 103552

Diogo Pires, 97889

Diogo Silva, 104341

Tomás Rodrigues, 104090

3/01/2024

Índice

Índice	2
1. Introdução	3
2. Resultados ASVS-Checklist	3
2.1. Authentication.....	3
2.2. Session Managment	3
2.3. Access Control	3
2.4. Input validation	4
2.5. Cryptography at rest	4
2.6. Error handling and Logging	4
2.7. Data protection	4
2.8. Communication Security	4
2.9. Malicious Code	4
2.10. Business Logic.....	5
2.11. Files and Resources	5
2.12. Web Services	5
2.13. Configuration.....	5
2.14. Results	6
3. Correção das vulnerabilidades	6
3.1. ASVS 4.2.2: Cross-Site Request Forgery (CSRF)	6
3.2. Software Feature No 1: Password Strength & Breach Verification	7
3.3. ASVS 2.5.4: No default Accounts.....	11
3.4. ASVS 9.1.1: Secure TLS	12
3.5. ASVS 3.2.2: Tokens with 64 bits of entropy	12
3.6. ASVS 3.4.1-3: Protecting session Cookies	12
3.7. Multi-factor Authentication (MFA)	13
3.7.1. TOTP authentication login.....	13
3.7.2. OAUTH 2.0	14
4. Conclusão	16

1. Introdução

Este projeto é a continuação do projeto *Vulnerabilities in software products* que consistia no desenvolvimento de um website para uma loja de produtos comercializáveis relacionados com o Departamento de Engenharia Computadores e Informática (DETI). Nesta segunda versão da loja começámos por preencher uma ASVS-Checklist para identificar os problemas de nível um presente na nossa aplicação. Foi nos pedido que escolhêssemos oito dos problemas presentes na ASVS-Checklist e os resolvêssemos.

Adicionalmente tivemos de escolher dois problemas-chave de uma lista fornecida pelos professores totalizando um total de dez problemas resolvidos.

2. Resultados ASVS-Checklist

2.1. Authentication

"Password Security Credentials" possui 5 non valid e 7 valid. Em "General Authenticator Requirements", encontramos 3 non valid e nenhuma valid. Já em "Authenticator Lifecycle Requirements", há 1 non valid e 2 valid. "Credential Recovery Requirements" apresenta 2 non valid e 3 valid. "Out of Band Verifier Requirements" possui 4 non valid e nenhuma valid. Para "Single or Multi Factor One Time Verifier Requirements", são 2 non valid e 5 valid.

Em "Credentials Storage Credentials", "Look-up Secret Verifier Requirements", "Service Authentication Requirements" e "Cryptographic Software and Devices Verifier Requirements" são todos not applicable.

2.2. Session Managment

Marcamos todas as alíneas de Session Managment como non-valid/Not Applicable pois não tínhamos usado session-tokens nem verificado nenhuma das suas funcionalidades

2.3. Access Control

Na parte de *acess control* apercebemos-mos que tínhamos seis parâmetros que podemos marcar como *valid*, no entanto, existiam três *non-valid* e um *not*

applicable por ser de nível 2. Dos três parâmetros *non-valid* do *access control* decidimos escolher um deles que vamos abordar na secção 3.1.

2.4. Input validation

Muitas das vulnerabilidades referentes a esta unidade já estão seguras a partir do momento que usamos o flask, pois este já vem com várias medidas de segurança incorporadas. Ou seja, boa parte das vulnerabilidades já estão resolvidas à priori. Contudo vulnerabilidades e recomendações da lista que já requisitavam cuidados mais específicos mantiveram-se.

2.5. Cryptography at rest

Somos vulneráveis face à única vulnerabilidade de nível 1.

2.6. Error handling and Logging

"Log Content Requirements" possui 1 non valid e 1 valid. Em "Error Handling", encontramos 1 valid e nenhum non valid.

Em "Log Processing Requirements " e "Log Protection Requirements" são todos not applicable.

2.7. Data protection

Não tivemos nenhum ASVS requirement de nível 1 desta secção a ser cumprido

2.8. Communication Security

Não verificamos TLS quando fizemos a primeira versão do website por isso consideramos os requirements desta secção como Non-valid.

2.9. Malicious Code

"Deployed Application Integrity Controls " possui 3 non valid e nenhum valid.

Em "Code Integrity Controls" e "Malicious Code Search " são todos not applicable.

2.10. Business Logic

Nesta unidade foi concluído que a aplicação demonstrava vulnerabilidade a 2 dos 5 elementos.

2.11. Files and Resources

Toda esta secção é not applicable visto que não é possível um utilizador inserir qualquer tipo de ficheiro no nosso website.

2.12. Web Services

Na parte de *web services* existem oito parâmetros de nível dois, ou seja, foram marcados como *non applicable*. Quanto aos parâmetros de nível 1 existem 5 *valid* e dois *non-valid*.

2.13. Configuration

Na *configuration* existem nove parâmetros *non applicable* por serem de nível dois ou três. No entanto existem dezasseis parâmetros de nível um dos quais sete marcamos como *non-valid* e nove como *valid*.

2.14. Results

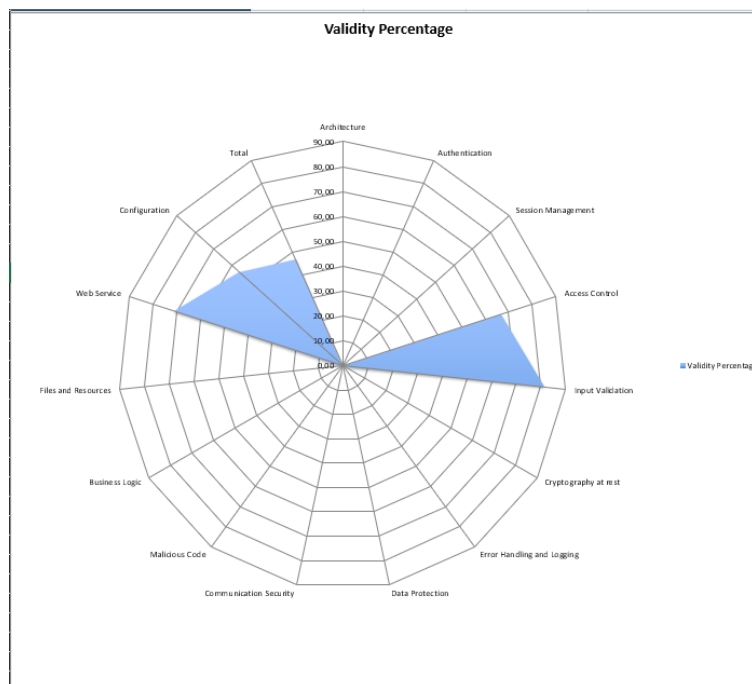


Figura 1 - Resultados

3. Correção das vulnerabilidades

3.1. ASVS 4.2.2: Cross-Site Request Forgery (CSRF)

Um dos problemas que nos deparamos foi com a falta de proteção contra ataques de CSRF na nossa aplicação. Esta vulnerabilidade ocorre quando um utilizador, em quem a aplicação confia, é induzido a executar ações não intencionadas sem o seu conhecimento ou consentimento.

O *CSRF* (Cross-Site Request Forgery) é uma ameaça de segurança na qual um atacante consegue realizar ações em nome de um utilizador autenticado, explorando a confiança que a aplicação deposita no navegador do utilizador. Isso pode levar a operações indesejadas, como alterar configurações, efetuar transações financeiras ou modificar dados pessoais.

Para mitigar esse problema, é crucial implementar medidas de proteção, como a utilização de tokens *CSRF*.

Então para resolver este problema resolvemos implementar tokens *CSRF* cada vez que o utilizador é autorizado a enviar dados para o site.

Para isso foi utilizada a biblioteca do Flask (***FLASK-WTF***) que fornece algumas ferramentas para lidar com tokens *CSRF*.

Exemplos de funções onde foi integrado o token *CSRF* são todas aquelas que usam métodos HTTP que permitem ao utilizador mexer de alguma forma com a base de dados (**Erro! A origem da referência não foi encontrada.**)

```
app.config['WTF_CSRF_METHODS'] = ['POST', 'PUT', 'DELETE']
csrf = CSRFProtect(app)
```

Figura 2 - Implementação de *CSRF*

Assim caso o token não esteja presente no formulário destas *routes* a aplicação dá o seguinte erro (**Erro! A origem da referência não foi encontrada.**)

Bad Request

The CSRF token is missing.

Figura 3 - Token não foi passado no formulário

Para que este erro não aconteça é necessário introduzir um token (**Erro! A origem da referência não foi encontrada.**), assim a aplicação fica protegida contra ataques de *CSRF*.

```
<input type="hidden" name="csrf_token" value="{ csrf_token() }">
```

Figura 4 - Token passado corretamente no formulário

3.2. Software Feature No 1: Password Strength & Breach Verification

Foram corrigidos todos os requerimentos ASVS que faltavam relacionados com Password Security Credentials (2.1)

- 2.1.1 & 2.1.2: Passwords entre 12 e 128 caracteres são permitidas.

```
password = PasswordField('Password', validators=[validators.DataRequired(), validators.Length(min=12, max=128)])
```

Figura 5 - Largura máxima e mínima para a palavra passe

Sign Up

Email

Name

Password


 Please lengthen this text to 12 characters or more (you are currently using 8 characters).

Figura 6-Exemplo de uma palavra sem largura mínima

- 2.1.7: Verificação de Breach de password usando uma API

Para a verificação de uma breached password usamos o [Have I Been Pwned API](#) e em caso de estar breached mandamos uma mensagem de erro ao utilizador para colocar outra palavra-passe.

```
# Check if password has been breached
sha1_password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
first5_chars, tail = sha1_password[:5], sha1_password[5:]
response = requests.get('https://api.pwnedpasswords.com/range/' + first5_chars)
hashes = (line.split(':') for line in response.text.splitlines())
count = next((int(count) for t, count in hashes if t == tail), 0)
if count:
    flash(f"Password has been breached {count} times, please use a different password", category='error')
    return render_template('sign_up.html', form=form)
```

Figura 7-Código para usar Pwned API

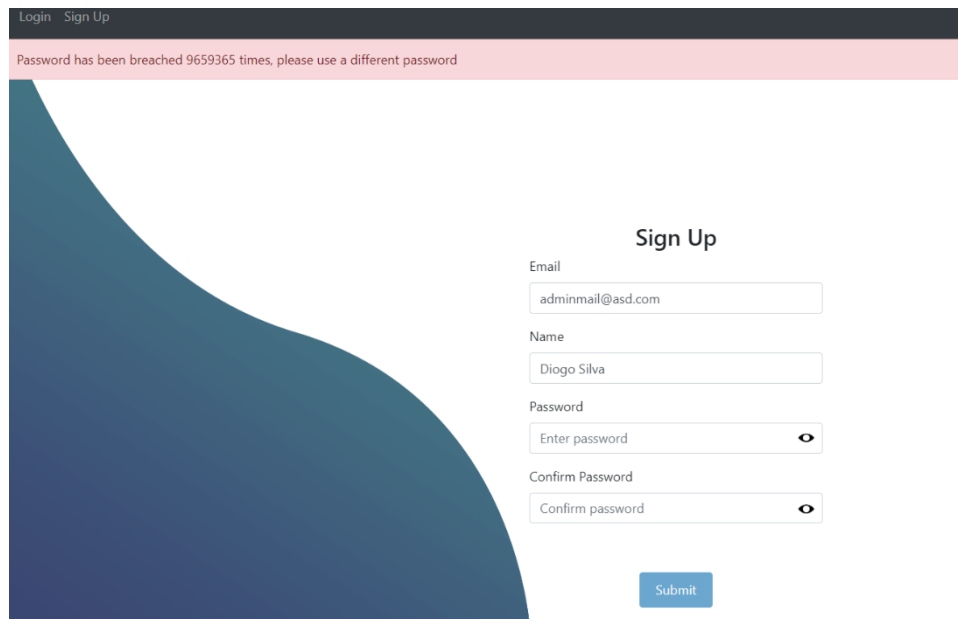


Figura 8-Resultado obtido após tentar criar uma conta com a password, "password"

- 2.1.8: Password strength meter

Uma barra a dizer quão forte é a password do utilizador, os fatores que influenciam são:

Letras maiúsculas

Letras minúsculas

Números

Caracteres especiais

A utilização destes fatores em conjunto indicará ao utilizador que a sua password é forte, a não utilização de um deles ou de vários indicará que será mais fraca.

```
{ { form.password(id="password", class="form-control", placeholder="Enter password", onkeyup="isGood(this.value); adjustIconPosition()", style="flex-grow: 1;") }}  
  
</div>  
<small class="help-block" id="password-text"></small>
```

Figura 9-Caixa da password chama o script isGood para avaliar quão forte é a password

```
// Function to show the password strength
function isGood(password) {
    var password_strength = document.getElementById("password-text");

    //TextBox left blank.
    if (password.length == 0) {
        password_strength.innerHTML = "";
        return;
    }

    //Regular Expressions.
    var regex = new Array();
    regex.push("[A-Z]"); //Uppercase Alphabet.
    regex.push("[a-z]"); //Lowercase Alphabet.
    regex.push("[0-9]"); //Digit.
    regex.push("[!@#$%^&*?&]"); //Special Character.

    var passed = 0;

    //Validate for each Regular Expression.
    for (var i = 0; i < regex.length; i++) {
        if (new RegExp(regex[i]).test(password)) {
            passed++;
        }
    }

    //Display status.
    var strength = "";
    switch (passed) {
        case 0:
        case 1:
        case 2:
            strength = "<small class='progress-bar bg-danger' style='width: 40%'>Weak</small>";
            break;
        case 3:
            strength = "<small class='progress-bar bg-warning' style='width: 60%'>Medium</small>";
            break;
        case 4:
            strength = "<small class='progress-bar bg-success' style='width: 100%'>Strong</small>";
            break;
    }
    password_strength.innerHTML = strength;
}
```

Figura 10-Script isGood para mostrar quão forte é a password

Password

👁

Strong

Figura 11-Exemplo de uma password forte

Password

👁

Medium

Figura 12-Exemplo de uma password média

Password



Weak

Figura 13-Exemplo de uma password fraca


- 2.1.8: Show and hide passwords

Foi adicionada a opção para os utilizadores mostrarem ou esconderem a password que estão a escrever

```
function togglePasswordVisibility(passwordId, iconId) {  
    var passwordInput = document.getElementById(passwordId);  
    var showPassIcon = document.getElementById(iconId);  
    if (passwordInput.type === "password") {  
        passwordInput.type = "text";  
        showPassIcon.src = "../static/img/hidePassIcon.png"; // change to hide icon  
    } else {  
        passwordInput.type = "password";  
        showPassIcon.src = "../static/img/showPassIcon.png"; // change back to show icon  
    }  
}
```

Figura 14-Javascript para esconder e mostrar a password, mudando o icon também

Password



Medium

Confirm Password




Figura 15-A mesma password escondida e a mostrar

3.3. ASVS 2.5.4: No default Accounts

Retiramos as contas admin@gmail.com , alex@gmail.com e eve@gmail.com que tínhamos usado na 1ª iteração do projeto:

```
cursor.execute("""INSERT OR IGNORE INTO users VALUES (NULL,'admin@gmail.com', 'admin', 'admin')""")
cursor.execute("""INSERT OR IGNORE INTO users VALUES (NULL,'alex@gmail.com', 'Alex', '1' )""")
cursor.execute("""INSERT OR IGNORE INTO users VALUES (NULL,'eve@gmail.com', 'Eve', '1' )""")
```

Figura 16-Contas default removidas

3.4. ASVS 9.1.1: Secure TLS

Também configuramos as session cookies para usarem apenas HTTPS que é um protocolo encriptado, verificando assim que usamos Transport Layer Security segura para o cliente

```
app.config.update(
    SESSION_COOKIE_SECURE=True,
    SESSION_COOKIE_HTTPONLY=True,
    SESSION_COOKIE_SAMESITE='Lax',
)
```

Figura 17-SECURE Policy

3.5. ASVS 3.2.2: Tokens with 64 bits of entropy

A configuração necessária para corrigir esta vulnerabilidade foi a simples adição da seguinte linha de código:

```
app.secret_key = os.urandom(64 // 8)
```

Figura 18-Mínimo de entropia

Esta linha contém a função `os.urandom` que como o nome já sugere é uma função que gera uma string de bytes aleatórios baseados na fonte de entropia do sistema operacional, geralmente é utilizada para gerar chaves criptográficas seguras.

'(64//8)' é uma operação matemática cujo resultado é 8, isso é feito para converter o tamanho da chave de 64 bits para a quantidade de bytes correspondente, ou seja, 8 bytes.

3.6. ASVS 3.4.1-3: Protecting session Cookies

Na seguinte sequência de código:

```
app.config.update(
    SESSION_COOKIE_SECURE=True,
    SESSION_COOKIE_HTTPONLY=True,
    SESSION_COOKIE_SAMESITE='Lax',
)
```

Figura 19- Configurações

A configuração “SESSION_COOKIE_SECURE = True” garante que o cookie é enviado apenas sobre HTTPS, isso evita que este seja interceptado.

Por recorrermos à configuração “SESSION_COOKIE_HTTPONLY = True” nós prevenimos a cookie da sessão de ser acesada via client-side scripts. Devido a isso podemos garantir que a aplicação nunca revela tokens da sessão nos parâmetros URL ou nas mensagens de erro.

(Também corrige ASVS 3.1.1)

Em “SESSION_COOKIE_SAMESITE=Lax” usamos a policy “Lax” no atributo SAMESITE para prevenir ataques CSRF.

Se fosse usada a policy “None” continuávamos suscetíveis a ataques CSRF.

Se fosse usada a policy “Strict” seria impossível usar a funcionalidade de login usando as contas google.

3.7. Multi-factor Authentication (MFA)

3.7.1. TOTP authentication login

A rota /mfaTOTP é definida para os métodos POST e GET. Esta rota é usada para lidar com o processo de verificação TOTP.

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

Uma key é definida para gerar o TOTP. A função pyotp.TOTP é usada para criar um objeto TOTP com esta chave e um intervalo de 180 segundos. Estes 180 segundos servem para compensar a demora do envio do código para o email, assim como dar tempo suficiente ao utilizador de copiar e colar o código.

```
key = "JBSWY3DPEHPK3PXP"
totp = pyotp.TOTP(key, interval=180)
```

Se o método da solicitação for POST, o código TOTP do utilizador é recuperado dos dados do formulário e posteriormente verificado se é composto por apenas dígitos de forma a sanitizar o conteúdo, caso não seja o utilizador será redirecionado para a página de login.

```
if request.method == 'POST':
    user_tfa_code = request.form.get('tfa')
    if not user_tfa_code.isdigit():
        return redirect('/login')
```

Este código é então verificado usando o método totp.verify. Se a verificação for bem-sucedida, o utilizador é redirecionado para a página inicial. Se não, eles são redirecionados para a página de login.

```

if totp.verify(user_tfa_code):
    session['user_tfa_code'] = user_tfa_code
    return redirect('/')
else:
    return redirect('/login')

```

Se o método da solicitação não for POST, um código TOTP é gerado usando o método `totp.now`. Este código é então enviado para o endereço de email do utilizador usando a função `send_email`. A função `send_email` usa as bibliotecas `smtplib` e `email.message.EmailMessage` para enviar um email com o código TOTP para o utilizador.

```

else:
    # Send the email when the page is loaded
    send_email(email_receiver, 'DetiShop: Temporary verification code', totp.now())

```

Em várias outras rotas responsáveis pela renderização da página web (como `/`, `/profile`, `/userManager`, etc.), há verificações se a variável de sessão `user_tfa_code` existe. Se não existir, o utilizador é redirecionado para a página de login. Isto garante que apenas os utilizadores que completaram com sucesso a verificação TOTP possam aceder a estas rotas.

```

if not session.get('user_tfa_code'):
    return redirect('/login')

```

Em resumo, este código implementa TOTP gerando um código único, enviando-o para o email do utilizador e, em seguida, verificando o código quando o utilizador o submete. Isto adiciona uma camada extra de segurança à aplicação, exigindo que os utilizadores tenham acesso à sua conta de email além de suas credenciais de login.

3.7.1.1. OAUTH 2.0

Configuração do OAuth: Usando a biblioteca `authlib.integrations.flask_client`, o Google é registado como um cliente OAuth com o ID do cliente, o segredo do cliente e a URL de metadados do servidor fornecidos pelo Google. Além disso, é definido o `'scope'` para `'openid email profile'`, informações que procuramos aceder do utilizador do Google.

```

oauth = OAuth(app)
google = oauth.register(
    name='google',
    client_id='391924228564-elhiuobuab4083rb17s4erotdju4t8ts.apps.googleusercontent.com',
    client_secret='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',
    server_metadata_url='https://accounts.google.com/.well-known/openid-configuration',
    client_kwargs={'scope': 'openid email profile'},
)

```

Rota de signup do Google: É definida uma rota `/google_signup` que redireciona o utilizador para a página de autorização do Google.

```

@app.route('/google_signup')
def google_signup():
    redirect_uri = url_for('authorize_signup', _external=True)
    return google.authorize_redirect(redirect_uri, prompt='select_account')

```

Rota de Autorização de signup: Após o utilizador autorizar o acesso, o Google redireciona-o de volta para a rota '/authorize_signup'. Aqui, é obtido o token de acesso e as informações do utilizador do Google.

```
@app.route('/authorize_signup')
def authorize_signup():
    token = google.authorize_access_token()
    resp = google.get('https://www.googleapis.com/oauth2/v3/userinfo')
    user_info = resp.json()
    user_email = user_info.get('email')

    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()
    cursor.execute("SELECT * FROM users WHERE email=?", (user_email,))
    user = cursor.fetchone()
```

Se o utilizador já existir na base de dados, é retornada uma mensagem informando que a conta já existe. Caso contrário, é inserido o novo utilizador na base de dados.

```
if user:
    connection.close()
    return "An account with this email already exists. Please log in."
else:
    # Generate a dummy password
    dummy_password = generate_password_hash(user_email, method='sha256')

    # Insert the new user into the database with the dummy password
    cursor.execute("INSERT INTO users (email, name, password, is_google_account) VALUES (?, ?, ?, ?)",
                  (user_email, user_info.get('name'), dummy_password, 1))
    connection.commit()
    connection.close()

    session['user_email'] = user_email
    session['user_name'] = user_info.get('name')
    session['user_tfa_code'] = 1
    return redirect('/')
```

Rota de Login do Google: Similar à rota de signup, é definida uma rota '/google_login' que redireciona o utilizador para a página de autorização do Google.

```
@app.route('/google_login')
def google_login():
    google = oauth.create_client('google')
    redirect_uri = url_for('authorize', _external=True)
    return google.authorize_redirect(redirect_uri, prompt='select_account')
```

Rota de Autorização: Após o utilizador autorizar o acesso, o Google redireciona o utilizador de volta para a rota '/authorize'. Aqui, é obtido o token de acesso e as informações do utilizador do Google.

```
@app.route('/authorize')
def authorize():
    google = oauth.create_client('google')
    token = google.authorize_access_token()
    resp = google.get('https://www.googleapis.com/oauth2/v3/userinfo')
    user_info = resp.json()
    user_email = user_info.get('email')

    connection = sqlite3.connect('data.db', check_same_thread=False)
    cursor = connection.cursor()
    cursor.execute("SELECT *, is_google_account FROM users WHERE email=?", (user_email,))
    user = cursor.fetchone()
    connection.close()
```

Se o utilizador existir na base de dados, a sessão do utilizador é iniciada. Caso contrário, é retornada uma mensagem informando que a conta não existe.

```
if user and user[-1]:
    session['user_email'] = user_email
    session['user_name'] = user_info.get('name')
    session['user_tfa_code'] = 1
    return redirect('/')
else:
    return "No google account with this email exists. Please sign up."
```

Em resumo, o OAuth 2.0 permite que os utilizadores façam signup e façam login usando as suas contas do Google. Quando um utilizador tenta dar signup ou fazer login, será redirecionado para a página de autorização do Google. Após a autorização, o Google redireciona o utilizador de volta para o nosso site com um código de autorização, que o nosso site troca por um token de acesso. O token de acesso é então usado para obter as informações do utilizador do Google.

4. Conclusão

Após resolvermos estes 10 problemas que achamos mais pertinentes para a segurança do nosso website, reparamos que por consequência resolvemos também muitos dos outros requerimentos ASVS de nível 1 que não tínhamos escolhido resolver, assim achamos que obtivemos os resultados desejados para esta segunda iteração do projeto.

Todos os membros trabalharam de igual forma para o trabalho contribuindo todos para a realização da ASVS checklist, resolução de problemas e composição do relatório.