universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Segurança Informática e nas Organizações

Assignment 2

VULNERABILITIES IN SOFTWARE PRODUCTS

Nicole Rakov - 96661

Alberto Oliveira - 98445

Eduarda Aires - 98447

David Ribeiro - 98586

Assignment 2

Vulnerabilities in Software Products

Introduction

The previous assignment consisted of the development of a web application (DETI Shop), featuring an online shop that sells DETI memorabilia. This assignment is a progression of that project, which aims to further enhance the web application's security aspects through **ASVS** (Application Security Verification Standard). This task follows a security audit designed to evaluate the current security of our application and identify areas that need improvement.

Implemented Improvements

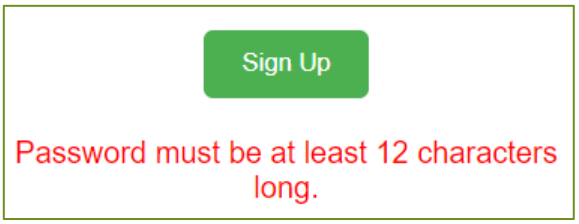
ASVS 2.1 - Password Security Credentials

ASVS 2.1.1 - Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined)

Code implementation

```python
# Check if the password meets the minimum length requirement
if len(password) < 12:
    error_message = "Password must be at least 12 characters long."
    return render_template("signup.html", error=error_message)
```

```
{% if error %}
    <div class="error-message">
        {{ error }}
    </div>
{% endif %}
```



The first code snippet shows that the user now must choose a password that has a length of 12 or more. In case the chosen password is too short, the user will get an error message.

ASVS 2.1.2 - Verify that passwords 64 characters or longer are permitted but may be no longer than 128 characters.

Code implementation

```
if len(password) > 128:
    error_message = "Password must be at most 128 characters long."
    return render_template("signup.html", error=error_message)
```

Similar to ASVS 2.1.1, this aims to further enhance the password's security, by not allowing the user to choose a password that is more than 128 characters long.

ASVS 2.1.3 - Verify that password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.

Code implementation

```
password = password.replace("  ", " ")
```

This implementation assures that while password truncation is not performed, consecutive spaces in the password are now normalized. As in, if a user inserts more than one space, it is replaced with a single space. This should improve usability without compromising the safety of the password.

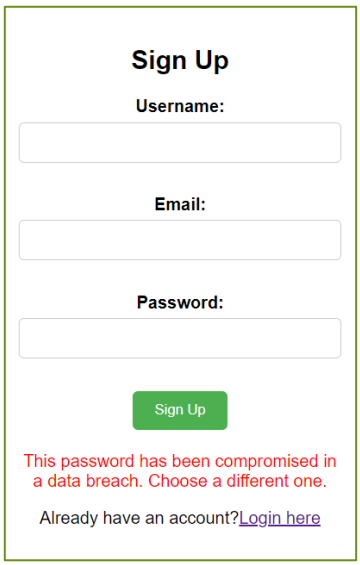
ASVS 2.1.7 - Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API, 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.

This ASVS is part of the mandatory set of requirements for assignment 2: "Password strength evaluation: requiring a minimum of strength for passwords according to V2.1, with breach verification using an external service".

Code implementation

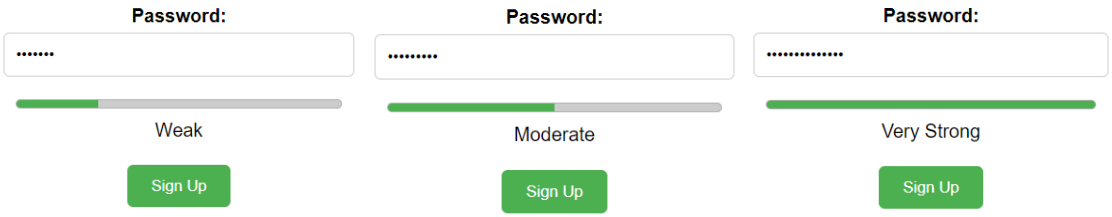
```
hash_password_api = hashlib.sha1(password.encode()).hexdigest().upper()
hash_prefix_api = hash_password_api[:5]
hash_suffix_api = hash_password_api[5:]

# Check if the password has been breached
if is_password_breached(hash_prefix_api, hash_suffix_api):
    error_message = "This password has been compromised in a data breach. Choose a different one."
    return render_template("signup.html", error=error_message)
```

```
def is_password_breached(hash_prefix, hash_suffix_api):
    # Have I Been Pwned API
    api_url = f'https://api.pwnedpasswords.com/range/{hash_prefix}'
    response = requests.get(api_url)
    if response.status_code == 200:
        # Check if the password's hash sufix exists in the response
        hash_suffixes = [line.split(':')[0] for line in response.text.splitlines()]
        return hash_suffix_api.upper() in hash_suffixes
    else:
        print(f"API request failed with status code {response.status_code}")
        print(response.text)
        print("returning false")
        return False
```



This code uses the “Have I been Pwned” API to check if the user’s chosen password has been previously compromised in any known data breaches. It starts by generating a hash of the user’s password, turning it to uppercase, and extracting the first five and last characters of the hash. The function “is_password_breached” then queries the API using the hashed prefix and suffix. The API then responds with a list of hash suffixes from breached passwords that share the same prefix. The function checks if the hash suffix of the user's password is present in this response, indicating that the password has been compromised in a known data breach. If this is the case, the user is asked to choose a different password.

ASVS 2.1.8 - Verify that a password strength meter is provided to help users set a stronger password.



Code Implementation

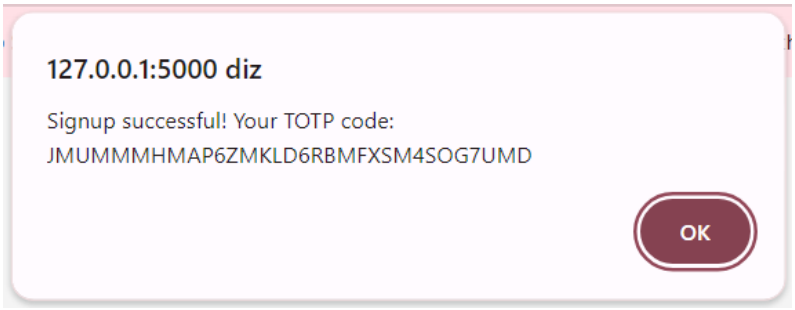
```
<script src="https://cdnjs.cloudflare.com/ajax/libs/zxcvbn/4.4.2/zxcvbn.js"></script>
<script>
    function checkPasswordStrength() {
        var password = document.getElementById('password').value;
        var meter = document.getElementById('password-strength-meter');
        var text = document.getElementById('password-strength-text');

        var result = zxcvbn(password);
        var score = result.score;

        // Update meter
        meter.value = score;

        // Update text based on the score
        switch (score) {
            case 0:
                text.innerHTML = "Very Weak";
                break;
            case 1:
                text.innerHTML = "Weak";
                break;
            case 2:
                text.innerHTML = "Moderate";
                break;
            case 3:
                text.innerHTML = "Strong";
                break;
            case 4:
                text.innerHTML = "Very Strong";
                break;
            default:
                text.innerHTML = "";
        }
    }
</script>
```

This script provides real-time feedback on the strength of the password the user is choosing. It uses the *zxcvbn* library, which analyzes passwords and provides a strength score. This function retrieves the password, calculates its strength using *zxcvbn*, and updates the password strength meter. The strength score is categorized from 0 to 4, with corresponding labels, from "Very Weak" to "Very Strong." This visual indicator helps users create stronger passwords by encouraging the use of complex and secure password patterns.

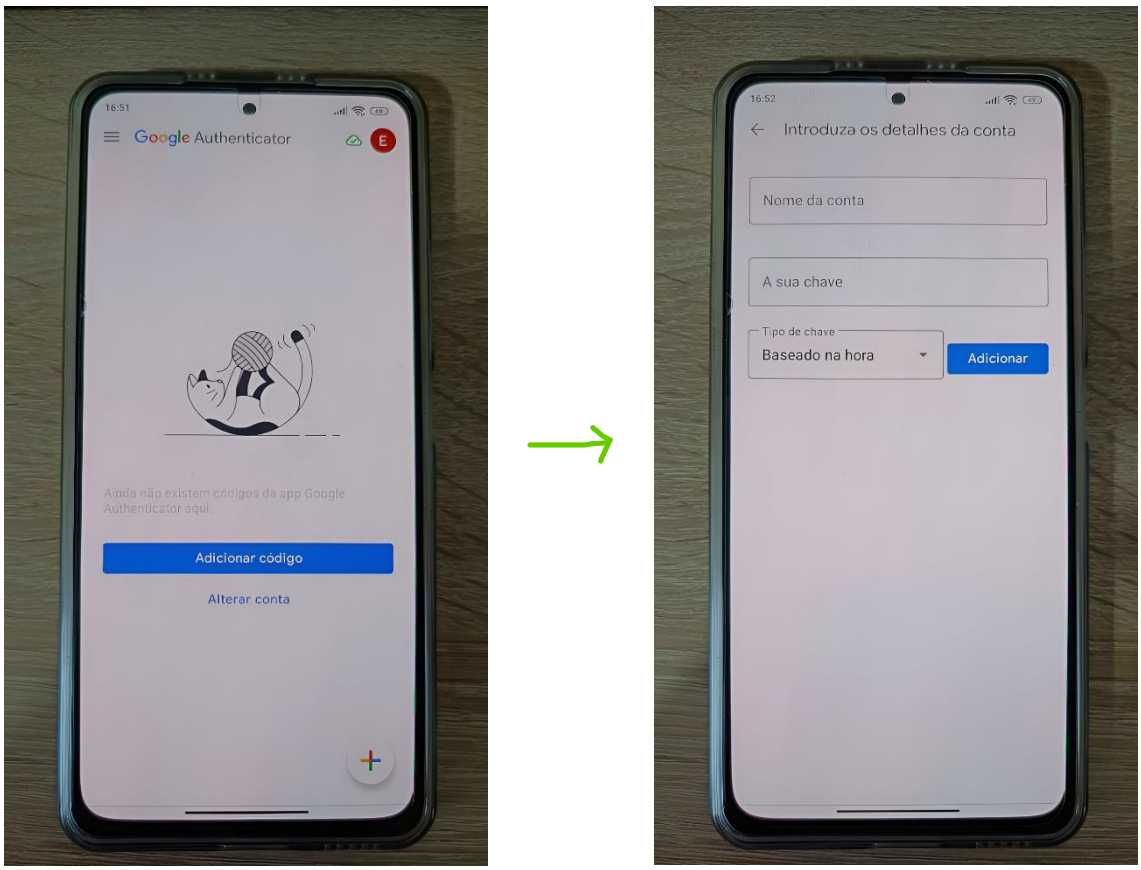
ASVS 2.8 - Single or Multifactor One Time Verifier Requirements

To implement MFA in our web application, we chose to use TOTP – time-based one-time passwords. Here's how the process works:

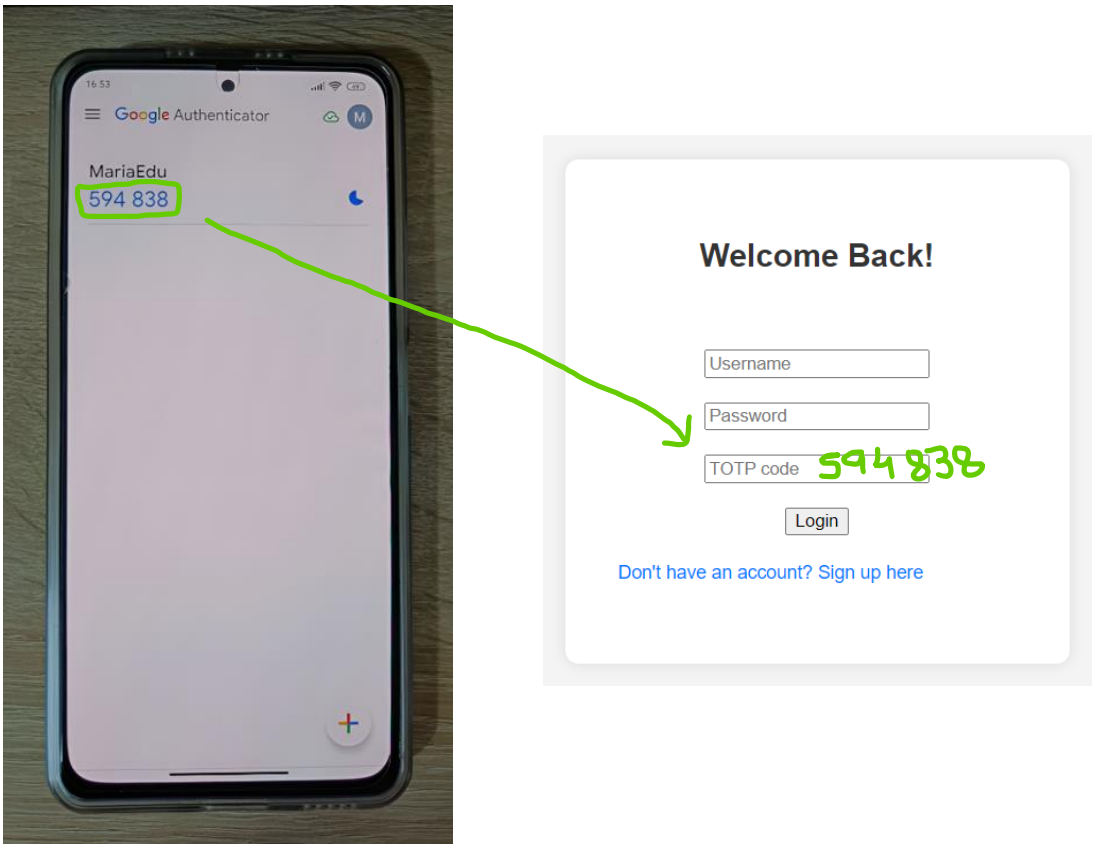
- When the user registers on the website, they receive a TOTP code. This code will be used on an authenticator application, which the user must download from the app store. The one we are going to use on this example is **Google Authenticator.**



- Once the user installs the authenticator application, they should click the "add code" button and choose the "Introduce configuration key" option. Then, the TOTP code given on sign-up should be introduced on the "your key" box.



- After the authenticator is set up, the code to be introduced during log-in should appear on the application and refresh every few seconds.



ASVS 2.8.1 - Verify that time-based OTPs have a defined lifetime before expiring.

This MFA method (TOTP) verifies this ASVS, as the codes generated based on a shared secret do have a defined lifetime before they expire.

Code Implementation

```
import pyotp
```

```
# Generate a TOTP secret for the user
totp_secret = pyotp.random_base32()

query = "INSERT INTO users (username, email, password, totp_secret) VALUES (?, ?, ?, ?)"
try:
    cursor.execute(query, (username, email, hashed_password, totp_secret))
    connection.commit()
    # Provide the TOTP secret to the user
    alert_message = f"Signup successful! Your TOTP code: {totp_secret}"
    #return render_template("login.html", totp_alert=totp_alert)
    return render_template("signup.html", alert_message=alert_message)
```

sign up()

In our web application, we use the *pyotp* library to generate and validate the OTP. In the sign-up function, a **totp_secret** is generated: this is the code that the user will introduce on an external authenticator application. Then, the log-in function verifies if the provided TOTP code **(totp.now())** matches the expected value based on the secret. If it doesn't match, an error message is shown, and the user must enter the code again.

```
query = "SELECT username, password, login_attempts, totp_secret FROM users WHERE username = ?"
cursor.execute(query, (username,))
user = cursor.fetchone()
if user:
    # Check if the user has totp enabled
    if user[3]:
        totp = pyotp.TOTP(user[3])
        if not totp.verify(totp.now()):
            error_message = "Incorrect TOTP code. Please try again."
            return render_template("login.html", error=error_message)
```

log-in()

ASVS 5.1 - Input Validation Requirements

ASVS 5.1.3 - Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (allow lists).

Our code did have input validation and sanitization measures (such as auto-escaping), but it was not done through positive validation, which we decided to implement during this assignment, in order to improve input validation security.

Code Implementation

```
# Allow lists:

# Allowed query values
allowed_q_values = {"index", "login", "signup", "shop", "checkout", "reviews", ""}
# Username must start with a letter and be at least 3 characters long
username_pattern = re.compile(r'^[a-zA-ZÀ-ÖØ-öø-ÿ0-9_ \'-]+$')
# List of reserved usernames
reserved_usernames = ["admin", "root", "system"]
# List of valid product names
product_names = ["black deti cup", "black deti mug", "black deti shirt", "deti cup", "deti mug", "deti shirt"]
```

```
@app.route('/submit_review', methods=['POST'])
def submit_review():
    if request.method == "POST":
        name = request.form.get("user_name")
        item = request.form.get("product_name")
        review_text = request.form.get("review")

        # Check if the name is valid
        if not username_pattern.match(name):
            return "Invalid username. Usernames must start with a letter and be at least 3 characters long."

        # Check if the name is in the list of reserved usernames
        if name.lower() in reserved_usernames:
            return "Invalid username. Please choose a different username."

        # Check if the product name is valid
        if item.lower() not in product_names:
            return "Invalid product name. Please choose a product sold in our shop."
```

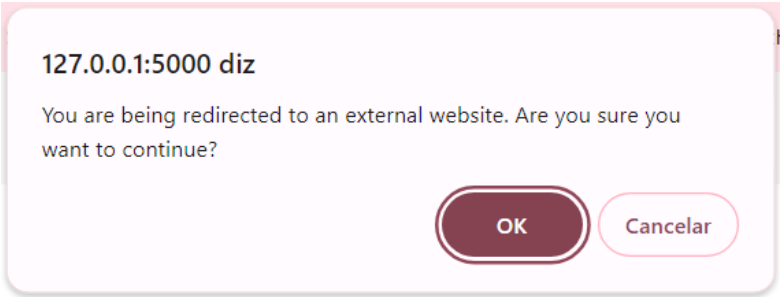
```python
@app.route('/')
def index():
    # Get the 'q' query parameter
    q = request.args.get('q', '')

    # Check if the 'q' parameter is in the allowed list of values
    if q not in allowed_q_values:
        return "This page does not exist."

    # Sanitize and escape the 'q' parameter
    q = escape(q)
    return render_template('index.html', q=q)

@app.route('/<path:invalid_path>')
def handle_invalid_path(invalid_path):
    q = request.args.get('q', '')
    if q not in allowed_q_values:
        return "Invalid query. This page does not exist."
    return f"This page does not exist: {invalid_path}"
```

In this code, we established some allow lists for the input fields in our “submit a review” page: username and product name. We don’t want the user to insert a username with characters outside of the alpha-numerical established ones, or to choose a product that our shop does not sell. Since we are only currently selling three products (Deti shirt, mug, and cup) those are the present in the allow list. There is also an allow list for possible queries, as well as a catch-all route, so incase the user tries to go to a page that is not present on that list by changing the URL, an error will be shown.

ASVS 5.1.5 - Verify that URL redirects and forwards only allow destinations which appear on an allow list or show a warning when redirecting to potentially untrusted content.

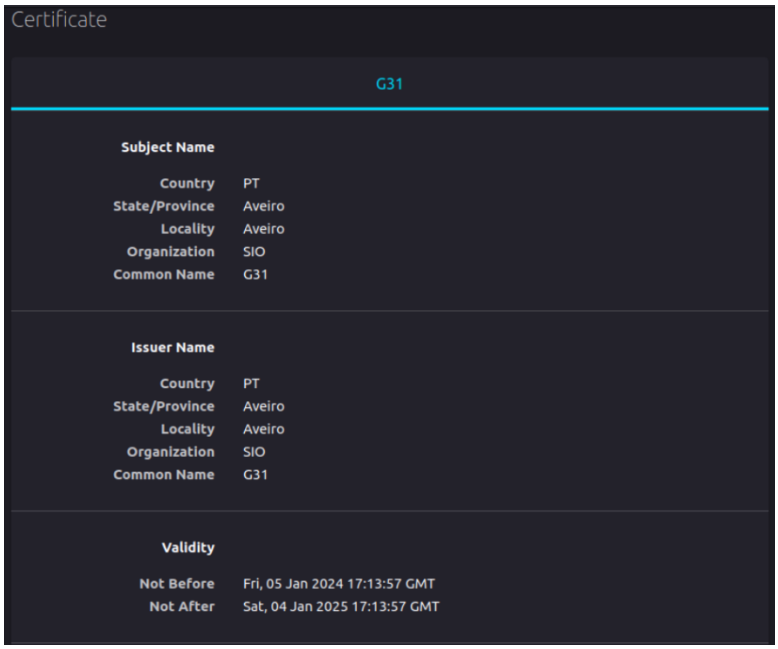
Code Implementation

```html
<script>
    document.addEventListener('DOMContentLoaded', function () {
        // Get all links on the page
        var redirectionLink = document.querySelectorAll('a');

        // Add a click event listener to each link
        redirectionLink.forEach(function (link) {
            link.addEventListener('click', function (event) {
                // Prevent the default behavior
                event.preventDefault();

                // Display a confirmation prompt
                var userConfirmed = confirm("You are being redirected to an external website. Are you sure you want to continue?");

                // If the user confirmed, navigate to the link
                if (userConfirmed) {
                    window.location.href = link.href;
                }
            });
        });
    });
    </script>
```



If the user clicks a link that redirects them to any external website, a warning will be shown, asking the user if they wish to proceed. If they do, they will be redirected. If they don't, they will remain on our website.

ASVS 9.1.1 - Verify that secured TLS is used for all client connectivity and does not fall back to insecure or unencrypted protocols.



We previously had no kind of certification and client connectivity did not use TLS. As such, we've created our own certificates to ensure that TLS is used for client connectivity and have a HTTPS url.

Code Implementation

```
351
352
353 if __name__ == '__main__':
354     app.run(debug=False, ssl_context=('cert.pem','key.pem'))
355
```

```
^Cdavid@david-laptop:~/2nd-project-group_31/app_sec$ openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
.....+...+....+..+....+......+.....................+...+........+.+......+...+..+...+.......+..+.+.............+..............+.+..+.......+...+...+......+..
...+....+.............+.+...+..+....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*...+.+.....+...+...+....+...+.....+.+..++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++*..+....+.....................+......+..............+.+........+............+.....+....+.....+...+.+
.............+..............+...+.+......+...........+......+...+..+.+....+.....................+..+...........+......+...+....+......+..............+......
..+.....+.....................+...+.+...........+.....+....+.....+...+.........+.........+...+.....+.....+...........+...+.........+......+.+......+......
......+...+.+......+.....................+..............+..........+...+.+......+........+.+......+...+..+......+.....................+......+...+......
...+.+..+.............+.+......+...+.....................+.+.....+...+.....................+..........+.........+.++++++++++++++++++++++++++++++++++++++++++++++++++
++++
...+......+....+.........+......+........+.+......+......+...+......+.....+.++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*......+......+.+...+
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*.+............+...+......+...+.....................+........+.+........+.....................+.+...+......
.+...........+.........+.+..+.........+....+......+...+.....+......+......+....++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:PT
State or Province Name (full name) [Some-State]:Aveiro
Locality Name (eg, city) []:Aveiro
Organization Name (eg, company) [Internet Widgits Pty Ltd]:SIO
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:G31
Email Address []:
```

ASVS 9.1.2 - Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.

We've used the OPENSSL packet to connect to localhost and analyze the certificate, which specifies protocols and ciphers. The following shows the command and result:

```
openssl s_client -showcerts -connect 127.0.0.1:5000
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
verify error:num=18:self-signed certificate
verify return:1
depth=0 C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
verify return:1
---
Certificate chain
0 s:C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
i:C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
a:PKEY: rsaEncryption, 4096 (bit); sigalg: RSA-SHA256
v:NotBefore: Jan  5 17:13:57 2024 GMT; NotAfter: Jan  4 17:13:57 2025 GMT
-----BEGIN CERTIFICATE-----
MIIFdzCCA1+gAwIBAgIUJ/ah394a10kKYwwHerE505zNFGIwDQYJKoZIhvcNAQEL
BQAwSzELMAkGA1UEBhMCUFQxDzANBgNVBAgMBkF2ZWlybzEPMA0GA1UEBwwGQXZl
aXJvMQwwCgYDVQQKDANTSU8xDDAKBgNVBAMMA0czMTAeFw0yNDAxMDUxNzEzNTda
Fw0yNTAxMDQxNzEzNTdaMEsxCzAJBgNVBAYTAlBUMQ8wDQYDVQQIDAZBdmVpcm8x
DzANBgNVBAcMBkF2ZWlybzEMMAoGA1UECgwDU0lPMQwwCgYDVQQDDANHMzEwggIi
MA0GCSqGSIb3DQEBAQUAA4ICDwAwggIKAoICAQDK13Mt97gWMyGwt9zk+EVzbe5V
GoKJ3iNwdxmJjK4bAd/pfkY9HvdX4SEq9dDKwufShVWSpt5EnZFiwk6v8wpcY6Uk
qs8MBX/+L5nAOk0BJX3dhlrZJQPFW/+s/zJsjBkPKDyzzKlBpczxig7R7XLbdS1t
MwNrgpHfwttIt7k9HUKXtTLoFX8UciIAm3dQUknLCbOXu7HQRty/Xoq2oIomMYbx
bNW7GDteSdFUqRCIlnq6grGfgjrhOAjbaldRVhoFYwVqEFAJsqYRBxVAQ7Q3edCG
kQH8//PyVzZVCgMaMqj2ebb9VfXwcpVrg2UP8lQ2nSreFduwR2JHbfxcS4lht13H
ZaBeiWOkyPEg55sxW9CaDfQa00QtXM73XlwoWAibEGvH/DRcBYmvanwiPos0G5Di
CmC+PEvdUw0fGEw9J7eccK2n3RFBTaLuq4obzFgk0Y04yWVay3VL3kQ6lURjaGSA
JCDQVGbsWyPm6VBcUpJAG2c5emWMpxH6AMrYP/JyYiwpJUsvnamdQzNJoUUKF+z3
+34jcJjMByx5OOJ8++tasNjmliQ7WNfPQL9zXysXBZKBQArf3IG/e2y0o6EvvkMx
EQWTmIW/O2XKHgnCJjvQnqgnWcj5ue9yLHtdqUEsQ4rvaY8ac1/y4Y574K+mc7IS
Z0VPky9xKxp17HZXuwIDAQABo1MwUTAdBgNVHQ4EFgQUbn8FYjWjcquBTgO81oUF
```

```
u2ti6jcwHwYDVR0jBBgwFoAUbn8FYjWjcquBTgO81oUFu2ti6jcwDwYDVR0TAQH/
BAUwAwEB/zANBgkqhkiG9w0BAQsFAAOCAgEAJYRAeSiLtEP6+9/PFjyJxrEaZ/sb
ctQDKnhMt060aevY/b3q66po0BAf7g7YEn0KJCZ6S+KJu3YUhVfmjfGOUQqe+NIR
/skdudN6af0svIwIuXmoqitfASotiMOIxhLV7NRe/uvSkK4KxLg1GSdtXTK/0ebB
YcnWRMIdq92JtCOd+FI67d9s/tNKMOM8lND0WHJHlqGuXbItjCw4pzWES4x7OxFU
1+wBj9YGEeXGd+RavRciWsoVT6CAd0nJqGyzP19aMa14Ih/o4ugfQuLa02taumZa
KlSmBv4mbq3qO/S3zaAGYg85MjpErOb5AsEJr7KJLjN87C1VpQpssSQRCzgNxziN
e53Yw+fQ++WDx+sb09kMyc2LFQEf84mmW8rJi+laemqk4CJLg2RLkGQ8hFaO6tBs
kZJE9QqQDRs3lXQjVhqwB3YneQ9E2ycg1bf2GJZPQg/IBHv1a6cr5/6eN83EvlHY
yZlWoDNg7U5vbVAJWfRNWra91MhHebkckGvfTSC89ZekOuXKp4XmtCUH0+7GlWDF
5IEJoY7rj04bSJsLQNmcAtWtQkichlIP/gWabFcvRrSD721lu9EFhVgWSqw0XaN5
zVsGEgfdJE68NHLVkVX+8mPUK4Un1Sfu7lFEO/9n7Oh2Pamt7Zt6a1K2weOq2upl
brJdutGO610hFEk=
-----END CERTIFICATE-----
---
Server certificate
subject=C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
issuer=C = PT, ST = Aveiro, L = Aveiro, O = SIO, CN = G31
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: RSA-PSS
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 2215 bytes and written 373 bytes
Verification error: self-signed certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 4096 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self-signed certificate)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
Protocol  : TLSv1.3
Cipher    : TLS_AES_256_GCM_SHA384
Session-ID: 5AB385D30EC60782FB290E8C551EDF11AC332B71A878EFBD914762B94B399B7F
Session-ID-ctx:
Resumption PSK: 43DD9FC139D72D68AAFABCB234E419F586A65CE1EBBAA1C701BA93AD9CDF9710A2F7B7211E6C5FD6F91C873D4AB8F224
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 7200 (seconds)
TLS session ticket:
0000 - 9a cb 70 41 9f 00 a0 77-2a 8b 06 81 f6 4b 8b cd   ..pA...w*....K..
0010 - 3f 31 fa d0 db e1 68 aa-5d a9 7c 35 95 a9 7a 37   ?1....h.].|5..z7
0020 - 63 a7 72 ad b0 47 bd 0a-78 5d b2 a8 b5 3b 95 7f   c.r..G..x]...;..
0030 - d0 83 df e6 ad 1f 8c af-4d e4 c7 83 c7 67 14 a6   ........M....g..
0040 - d0 ed b3 ef 74 8c 92 2f-94 66 4b 8b 7d 4c 84 2b   ....t../.fK.}L.+
0050 - 4b a3 12 e2 6d 00 b4 c0-e1 fb de 07 6a f3 a4 d2   K...m.......j...
0060 - 77 d8 90 40 80 d2 7c 80-02 60 89 cc 03 27 bb ba   w..@..|..`...'..
0070 - fa d6 20 8f ff 13 69 d3-95 86 78 e2 03 d4 73 5a   .. ...i...x...sZ
0080 - a3 a3 52 ab 22 43 a8 42-e7 38 a1 e4 7a f4 a6 aa   ..R."C.B.8..z...
0090 - 9f 7a 38 f1 95 f8 d8 e1-3c 21 56 ce 78 19 f1 ae   .z8.....<!V.x...
00a0 - 51 2b 19 97 10 e6 d0 6c-e5 eb 3f 7d 50 7a c8 b8   Q+.....l..?}Pz..
00b0 - 3b 6f 71 f9 2a 2a 7f c3-6a bc 37 46 3a ab 17 4f   ;oq.**..j.7F:...O
00c0 - 0f f3 6f 59 63 a2 38 13-eb 17 3f df 0e 48 02 bd   ..oYc.8...?..H..

Start Time: 1704491155
Timeout   : 7200 (sec)
Verify return code: 18 (self-signed certificate)
```

```
Extended master secret: no
Max Early Data: 0
---
read R BLOCK
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
Protocol : TLSv1.3
Cipher    : TLS_AES_256_GCM_SHA384
Session-ID: 476CE83FDF07435E4AA87B3A1E5F4FF0C9C799958F3E8F632B3F829E78D01ACA
Session-ID-ctx:
Resumption PSK: 406F52D3EAE81F28A2AF499E6BA1E72788D62B83E87D8B4F7234AE12087D5C60BCA376EBB596F795FBB7BB40B3B49A40
PSK identity: None
PSK identity hint: None
SRP username: None
TLS session ticket lifetime hint: 7200 (seconds)
TLS session ticket:
0000 - 9a cb 70 41 9f 00 a0 77-2a 8b 06 81 f6 4b 8b cd   ..pA...w*....K..
0010 - bb 5a 05 ac bc c4 d4 e3-cd 1b 36 15 96 e7 f2 66   .Z........6....f
0020 - 70 5c fb 97 3d 1b 89 9f-48 6d 3b a3 4a 6d 2d c6   p\..=...Hm;.Jm-.
0030 - b3 a2 9c c6 89 18 25 c0-5a c0 98 f1 eb a8 e7 0e   ......%.Z.......
0040 - b1 19 9f a8 d9 b0 83 5c-54 8e c9 35 91 81 3d a5   .......\T..5..=.
0050 - ac f8 0d 78 23 80 e2 ba-84 17 f1 fc 5b b6 b2 0e   ...x#.......[...
0060 - 26 54 44 e6 a8 10 39 c8-7b ed ac 63 81 6d 7c 44   &TD...9.{..c.m|D
0070 - 6a ab 1b 50 8e 4a 4f 21-33 78 5c b2 24 87 20 c2   j..P.JO!3x\.$. .
0080 - bf 84 96 73 94 6e fc 5a-d3 45 a4 97 5a 8a d3 9e   ...s.n.Z.E..Z...
0090 - 13 1e af b9 d3 00 4a 8d-08 06 3b c7 a9 52 11 f7   ......J...;..R..
00a0 - 0c 19 a4 5d 03 6e 10 f1-74 52 f3 ac 29 23 c1 69   ...].n..tR..)#.i
00b0 - 4a 15 c5 87 1e 11 56 92-34 ce a0 98 9f 2a cb 3f   J.....V.4....*.?
00c0 - 3f bf 92 f0 8e fc 65 f0-0d ae 59 94 65 83 6f 02   ?.....e...Y.e.o.

Start Time: 1704491155
Timeout   : 7200 (sec)
Verify return code: 18 (self-signed certificate)
Extended master secret: no
Max Early Data: 0
---
```

ASVS 9.1.3 - Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.

Through the data provided in the analysis above, we can verify that the TLS version used is TLS 1.3, the latest version available.

ASVS 10.3.2 – Verify that the application employs integrity protections, such as code signing or subresource integrity. The application must not load or execute code from untrusted sources, such as loading includes modules, plugins, code, or libraries from untrusted sources or the Internet.

This was previously non-valid, since in the html files the scripts didn't have the integrity attribute.
Now, the integrity attribute allows the browser to check the fetched script to ensure that the code is never loaded if the source has been manipulated.
A code was generated:
c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4
And added to html files which contained a script tag.

Code Implementation

```
<!-- Js Plugins -->
<script src="../static/js/jquery-3.3.1.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/bootstrap.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/jquery.nice-select.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/jquery-ui.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/jquery.slicknav.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/mixitup.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/owl.carousel.min.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
<script src="../static/js/main.js" integrity = "c5be56967fe0886b36fe05695aec6198047dd3ea7f208bc5968894c9d6d5dfe4"></script>
```

ASVS 14.4.4 – Verify that all responses contain a X-Content-Type-Options: nosniff header.

Code Implementation

```
+ # Add X-Content-Type-Options header to all responses
+ @app.after_request
+ def add_content_type_options_header(response):
+     response.headers['X-Content-Type-Options'] = 'nosniff'
+     return response
```

Through this code, we add a security header ('X-Content-Type-Options') with the value 'nosniff' to the HTTP response after each request, which helps enhance security by preventing MIME type sniffing in browsers.

ASVS 4.2.2 - Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.

Code Implementation

```
10 + from flask_wtf.csrf import CSRFProtect
11
12
13   app = Flask(__name__, static_folder="./static", template_folder="./templates")
14   bcrypt = Bcrypt(app)
15 + csrf = CSRFProtect(app_sec)
16 +
17 + csrf.init_app(app)
18 +
```

Flask has built in CSFR protection, but it needed to be implemented, which we've done through this code.

ASVS 14.2.1 - Verify that all components are up to date, preferably using a dependency checker during build or compile time.

Code Implementation

It was not implemented, but now a requirement.txt was added and it contains the vertions of the framework and libraries.

requirements.txt ASVS 14.2.1

```
1   bcrypt==4.0.1
2   blinker==1.6.3
3   certifi==2023.11.17
4   cffi==1.16.0
5   charset-normalizer==3.3.2
6   click==8.1.7
7   colorama==0.4.6
8   cryptography==41.0.5
9   Flask==3.0.0
10  Flask-Bcrypt==1.0.1
11  idna==3.6
12  itsdangerous==2.1.2
13  Jinja2==3.1.2
14  MarkupSafe==2.1.3
15  pycparser==2.21
16  pyotp==2.9.0
17  requests==2.31.0
18  urllib3==2.1.0
19  Werkzeug==3.0.1
```

When running locally, it can be easily installed as:
Install dependencies: `pip install -r requirements.txt

ASVS 14.2.1 - Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.

Code Implementation

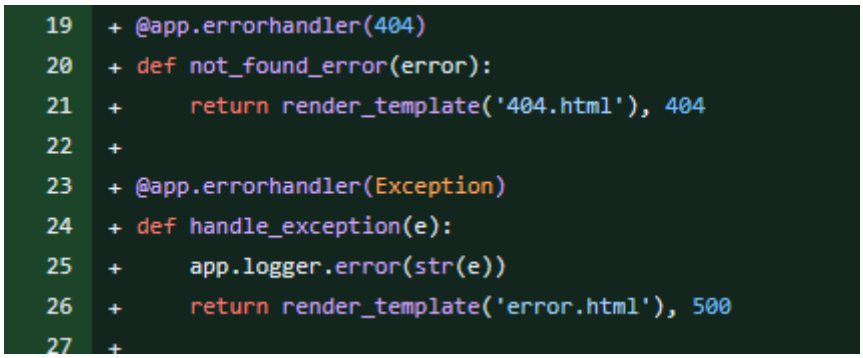
Error html pages are now present at app_sec.py. And routes have being applied:

```
19  + @app.errorhandler(404)
20  + def not_found_error(error):
21  +     return render_template('404.html'), 404
22  +
23  + @app.errorhandler(Exception)
24  + def handle_exception(e):
25  +     app.logger.error(str(e))
26  +     return render_template('error.html'), 500
27  +
```

ASVS 14.3.1 – Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.

Code Implementation

Error html pages are now present at app_sec.py. And routes have being applied:



ASVS 14.3.2 - Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.

Code Implementation

```
16  + app.debug = False
```

ASVS 14.4.5 - Verify that a Strict-Transport-Security header is included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.

Code Implementation

```
def add_strict_transport_security_header(response):
    max_age_seconds = 15724800
    # Set the header with includeSubdomains directive
    response.headers['Strict-Transport-Security'] = f'max-age={max_age_seconds}; includeSubdomains'
    return response
```

ASVS 14.4.6 - Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".

Code Implementation

```
+ # Referrer-Policy header to all responses
+ @app.after_request
+ def add_referrer_policy_header(response):
+     response.headers['Referrer-Policy'] = 'no-referrer'
+     return response
```

ASVS 14.4.6 - Verify that the content of a web application cannot be embedded in a third-party site by default and that embedding of the exact resources is only allowed where necessary by using suitable Content-Security-Policy: frame-ancestors and X-Frame-Options response headers.

Code Implementation

```
55      -     csp_policy = "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'"
55      +     csp_policy = "default-src 'self'; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline';
        frame-ancestors 'self'"
```

ASVS 14.5.1 - Verify that the application server only accepts the HTTP methods in use by the application/API, including pre-flight OPTIONS, and logs/alerts on any requests that are not valid for the application context.

Code Implementation

A function to verify the most common methods was implemented to add more security:

```
55 + # Validate the request method
56 + @app.before_request
57 + def validate_request_method():
58 +     valid_methods = {'GET', 'POST'}
59 +     if request.method not in valid_methods:
60 +         # Log or alert about the invalid request method
61 +         app.logger.warning(f"Invalid HTTP method: {request.method} for URL: {request.url}")
62 +         return "Invalid request method", 405
```

Justifying our choices

We decided to choose these security improvements for a variety of reasons, such as:

- User authentication is one of they key factors in a web application like ours. As such, we decided that improving security regarding passwords was very important and something we'd like to work on, therefore we chose to implement the following ASVS: 2.1.1, 2.1.2, 2.1.3, 2.1.5. Working with an external API such as "Have I been Pwned" to verify possible password breaches on user sign-up was also something we wanted to implement, to further enhance password security (2.1.7).
- MFA was also an important feature to implement on our web application. Though we already had user authentication via username/password enabled, we decided to add another layer of protection, through TOTP (time-based one-time passwords) – a method that is widely used in various applications that we are familiar with.
- Input validation is an important matter to avoid possible malicious activity, so we made sure that our website verified both ASVS 5.1.3 and 5.1.5, implementing allow lists for input validation (we previously relied on auto-escaping alone) and warning users whenever they click a link to an external website, which is something that our applications that we use tend to do (such as Discord, for example, that shows a warning whenever you're leaving the app through a link).
- Our website did not have any kind of certification, which had our URL show as "HTTP" instead of "HTTPS". We've added certificates and ensured that client connectivity is done through TLS, verifying 9.1.1, 9.1.2 and 9.1.3.
- By employing integrity protections, such as code signing or sub resource integrity (10.3.2), we enhance the code authenticity verifying that he comes from trusted sources, we prevent code injections and even protects against supply attack chain verifying that the code did not change during traffic.
- By implementing a strong anti-CSRF mechanism we protect against one of the most used attacks, the Cross-Site Request Forgery, making the prevention of unintended actions and the user data integrity stronger (4.2.2).

- Addressing the issue of keeping all components up to date is essential for the app keep the latest security patches and fixes. (14.2.1)
- With the rest of the implementation of ASVS 14.X, we tries to address various best practices, as well as robust code and Proactive Security Header Implementation

Audit Log

Authentication

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
2.1.1	Verify that user set passwords are at least 12 characters in length (after multiple spaces are combined).	Non-valid		The code does not check the length of the password leading to weaker passwords.
2.1.2	Verify that passwords 64 characters or longer are permitted but may be no longer than 128 characters.	Non-valid		The code does not check the length of the password leading to vulnerabilities associated with excessively long passwords
2.1.3	Verify that password truncation is not performed. However, consecutive multiple spaces may be replaced by a single space.	Non-valid		The code does not check multiples spaces or truncation. This may lead unauthorized access due to incomplete passwords.
2.1.4	Verify that any printable Unicode character, including language neutral characters such as spaces and Emojis are permitted in passwords.	Valid		By default, it's already working, enchanting the complexity of the password and making it more secure.
2.1.5	Verify users can change their password.	Non-valid		Not implemented in the code, this lead to more compromised or forgotten passwords since users can't change the old one. This can cause more unauthorized logins.
2.1.6	Verify that password change functionality requires the user's current and new password.	Not Applicable		Not implemented since there is no way to change password.

				But it could lead to unauthorized individuals changing passwords without proper authentication.
2.1.7	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.	Non-valid		Not implemented in the code. This can lead to unauthorized logins since the user password is breached.
2.1.8	Verify that a password strength meter is provided to help users set a stronger password.	Non-valid		Not implemented in the code. This may lead to user not knowing if the password is strong enough and might choose guessable passwords.
2.1.9	Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters.	Valid		There are no requirements implemented which lead to stronger passwords because there are less predictable patterns.
2.1.10	Verify that there are no periodic credential rotation or password history requirements.	Valid		There is no periodic credential rotation what is good because may lead to users choosing weaker passwords to remember.
2.1.11	Verify that "paste" functionality, browser password helpers, and external password managers are permitted.	Valid		Paste functionality is working leading to user convenience writing the password.
2.1.12	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	Non-valid		Not implemented. This may lead to frustration, errors during password entry and forgotten credentials.

	do not have this as built-in functionality.			
2.2.1	Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.	Non-valid	login_attempts = user[2] + 1 if login_attempts >= MAX_LOGIN_ATTEMPTS: error_message = "Account locked due to excessive login attempts. Please contact support." cursor.execute("UPDATE users SET login_attempts = ? WHERE username = ?", (login_attempts, username))	Even though there are a little protection against several trys to enter an account, it is very little compared to the required since it is needed more, for example: block breached passwords, check IP address, etc.. This can lead to more unauthorized logins since it is easier to attack.
2.2.2	Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.	Not Applicable		There is no use of authenticators in the app. Leading to less layers of protection therefore weaker security.
2.2.3	Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.	Non-valid		There is no notifications being sent to users that can result in unaware of potentially unauthorized activities on their accounts.
2.3.1	Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6	Not Applicable		There is no code or passwords being generated in the code. This way an attacker can't exploit.

	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.			
2.5.1	Verify that a system generated initial activation or recovery secret is not sent in clear text to the user.	Not Applicable		There is no recovery secret or initial security being sent to the user
2.5.2	Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.	Valid		There is no password hints or secret questions in the app reducing the risk of attackers guessing the questions and unauthorized logins.
2.5.3	Verify password credential recovery does not reveal the current password in any way.	Not Applicable		There is no way to change an authentication factor enhancing security.
2.5.4	Verify that if an authentication factor is changed or replaced, that the user is notified of this event.	Not Applicable		There is no way to recovery the password.
2.7.1	Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.	Not Applicable		There is no authenticator like that being used.
2.7.2	Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.	Not Applicable		There is no authenticator like that being used.
2.7.3	Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.	Not Applicable		There is no authenticator like that being used.
2.7.4	Verify that the out of band authenticator and verifier communicates over a secure independent channel.	Not Applicable		There is no authenticator like that being used.
2.8.1	Verify that time-based OTPs have a defined lifetime before expiring.	Not Applicable		There is no time-based OTPs being used.

Session Management

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
3.1.1	Verify the application never reveals session tokens in URL parameters or error messages.	Valid		This website does not reveal any session tokens in URL or any error code.
3.2.1	Verify the application generates a new session token on user authentication.	Valid		The website does not generate any session tokens or store any data on Local Storage.
3.2.2	Verify that session tokens possess at least 64 bits of entropy.	Not Applicable		
3.2.3	Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.	Not Applicable		
3.3.1	Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties.	Not Applicable		
3.3.2	If authenticators permit users to remain logged in, verify that re-authentication occurs periodically both when actively used or after an idle period.	Valid		As session tokens are not used, authentication must be made every separate visit to the website.
3.4.1	Verify that cookie-based session tokens have the 'Secure' attribute set.	Not Applicable		Cookies are not used.
3.4.2	Verify that cookie-based session tokens have the 'HttpOnly' attribute set.	Not Applicable		
3.4.3	Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks.	Not Applicable		
3.4.4	Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.	Not Applicable		
3.4.5	Verify that if the application is published under a domain name	Not Applicable		

	with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible.			
3.7.1	Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.	Valid		The website checks if authentication credentials are valid through a username and password. Although, this step is not required before engaging in a transaction.

Access Control

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
4.1.1	Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.	Valid	role TEXT NOT NULL CHECK(Role IN ('admin', 'customer')) DEFAULT 'customer'	Users have roles.
4.1.2	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	Not Applicable		

4.1.3	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection against spoofing and elevation of privilege.	Valid	if bcrypt.check_password_hash(user[1], password):
			# Reset login attempts on successful login
			cursor.execute("UPDATE users SET login_attempts = 0 WHERE username = ?", (username,))
			connection.commit()
			return render_template("logged_in.html")
				if user is authenticated, the app determines what they are authorized to access.

4.1.4	Verify that the principle of deny by default exists whereby new users/roles start with minimal, or no permissions and users/roles do not receive access to new	Valid	role TEXT NOT NULL CHECK(Role IN ('admin', 'customer')) DEFAULT 'customer'	by default, a new user is assigned as customer.

	features until access is explicitly assigned			

4.1.5	Verify that access controls fail securely including when an exception occurs.	Valid	if user:
			if bcrypt.check_password_hash(user[1], password):
			# Reset login attempts on successful login
			cursor.execute("UPDATE users SET login_attempts = 0 WHERE username = ?", (username,))
			connection.commit()
			return render_template("logged_in.html")
			else:
			# Increment login attempts and lock account if necessary
			login_attempts = user[2] + 1
			if login_attempts >= MAX_LOGIN_ATTEMPTS:
			error_message = "Account locked due to excessive login attempts. Please contact support."
			cursor.execute("UPDATE users SET login_attempts = ? WHERE username = ?", (login_attempts, username))
			connection.commit()
			else:
			error_message = "Incorrect credentials. Please check your username and password and try again."
			cursor.execute("UPDATE users SET login_attempts = ? WHERE username = ?", (login_attempts, username))
			connection.commit()
			return render_template("login.html", error=error_message)
			
			else:
			error_message = "Incorrect credentials. Please check your username and password and try again."
			return render_template("login.html", error=error_message)	

4.2.1	Verify that sensitive data and APIs are protected against Insecure Direct Object Reference (IDOR) attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.	Not Applicable		
4.2.2	Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.	Non-valid		framework enforces a strong anti-CSRF mechanism. But need some steps: To enable CSRF protection globally for a Flask app, register the CSRFProtect extension.

				from flask_wtf.csrf import CSRFProtect csrf = CSRFProtect(app)
4.3.1	Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.	Not Applicable		Admin role was implemented but it does not have a distinct access control requirement or mechanism for multi-factor authentication.
4.3.2	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	Not Applicable		No web server is being used.

Input Validation

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	Valid	request.form.get() methods=['GET','POST']	The flask app makes distiction from where the informations comes, if it is GET or POST and explicitly gets the data from the form.
5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar.	Non-valid		Flask does not provide any built-in feature to protect against mass parameter assignment attacks, and none were added.
5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds,	Non-valid		The input sanitizion present uses auto-escaping instead of positive validation.

	etc) is validated using positive validation (allow lists).			
5.1.4	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match)	Not Applicable		There is no form present in the website that would take this kind of data.
5.1.5	Verify that URL redirects and forwards only allow destinations which appear on an allow list or show a warning when redirecting to potentially untrusted content.	Non-valid		While the website itself has no URL redirects to exterior destinations, a user could possibly place a link to one in a review, and no warning would be shown.
5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature.	Valid		Flask provides built-in auto-escaping, this way is protected against Cross-Site Scripting (XSS) attacks
5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	Valid		Flask provides built-in auto-escaping, which escapes special characters.
5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	Not Applicable		There are no mail systems present in the website, so it does not include functionalities related to sending or receiving e-mails.
5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.	Valid		There are no dynamic code execution features in the website.
5.2.5	Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.	Valid		Flask's auto-escaping feature is enabled which sanitizes user input.
5.2.6	Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such	Not Applicable		There are no file or URL input fields in the website.

	as filenames and URL input fields, and uses allow lists of protocols, domains, paths and ports.			
5.2.7	Verify that the application sanitizes, disables, or sandboxes user-supplied Scalable Vector Graphics (SVG) scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.	Not Applicable		The user is not allowed to upload any kind of file.
5.2.8	Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.	Valid		Flask provides built-in auto-escaping, disabling expression template language content.
5.3.1	Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara).	Valid		Flask provides built-in auto-escaping, which ensures that characters that have special meanings in HTML are replaced with their HTML entity equivalents.
5.3.2	Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled.	Valid		The HTML templates include the correct character set declaration (UTF-8).
5.3.3	Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS	Valid		Flask provides built-in output auto-escaping, which protects against XSS attacks.
5.3.4	Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks.	Valid		Database queries use parameterized queries.
5.3.5	Verify that where parameterized or safer mechanisms are not present, context-specific output	Not Applicable		Parameterized queries are present in every situation.

	encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection.			
5.3.6	Verify that the application protects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, Content Security Policy (CSP) bypasses, DOM XSS, and JavaScript expression evaluation	Non-valid		While Flask's built-in auto-escaping feature prevents JS injections when rendering HTML content, it doesn't protect against JSON injections.
5.3.7	Verify that the application protects against LDAP injection vulnerabilities, or that specific security controls to prevent LDAP injection have been implemented.	Not Applicable		The application does not use LDAP.
5.3.8	Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding.	Not Applicable		No OS commands are used.
5.3.9	Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.	Not Applicable		There is no file upload or download feature, or dynamic file inclusion functions/paths.
5.3.10	Verify that the application protects against XPath injection or XML injection attacks	Valid		Special characters are escaped due to Flask's built-in auto-escape, and there are no dynamic XPath queries.
5.5.1	Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering.	Non-valid		We could store a calculated hash and then compare it after deserialization.
5.5.2	Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XML eXternal Entity (XXE) attacks.	Not Applicable		
5.5.3	Verify that deserialization of untrusted data is avoided or is protected in both custom code	Not Applicable		

	and third-party libraries (such as JSON, XML and YAML parsers).			
5.5.4	Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).	Not Applicable		

Cryptography at Rest

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
6.2.1	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.	Valid	bcrypt.check_password_hash() bcrypt.generate_password_hash()	We do not directly use decryption and encryption because we use bcrypt, an imported function designed to be resistent. The error handling is done in a generic way for protection.

Error Handling and Logging

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
7.1.1	Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form.	Not Applicable		There are no logs in the application. The error handling is done in a generic way for protection.
7.1.2	Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy.	Not Applicable		There are no logs in the application. The error handling is done in a generic way for protection.
7.4.1	Verify that a generic message is shown when an unexpected or	Non-valid	error_message = "Incorrect credentials. Please check	Even though a good generic answer is given to each

	security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate.		your username and password and try again." error_message = "Account locked due to excessive login attempts. Please contact support."	problem it does not give a unique ID to a problem for further investigation.

Data Protection

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
8.1.1	Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.	Non-Valid		The code does not have anti-caching headers. Here is an example of code that could be used: .headers['Cache-Control'] = 'no-store, no-cache, must-revalidate, max-age=0' .headers['Expires'] = '0' Without it there is a security risk as it could be accessed by unauthorized users, leading to unauthorized access to sensitive information
8.2.3	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.	Non-valid		There is no clearing from client storage in the app. Code that could fix the problem: session.pop('ID') Without it the data may persist in the client storage after the log out and other users in the same machine could acess the previous user's information, posing a security risk.
8.3.1	Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.	Valid	username = request.form.get("username", False) email = request.form.get("email", False)	The app send the sensitive information correctly avoiding getting logged, cached or other ways that are possible to retrieve the data.

			password = request.form.get("password", False)	
8.3.2	Verify that users have a method to remove or export their data on demand.	Non-valid		There is no way to the user remove or export data. This is a problem because do not align with the privacy and user rights principles.
8.3.3	Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.	Non-valid		There is no privacy policy neither opt-in consent leading to privacy concerns.
8.3.4	Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data	Non-valid		There is no listing of policies to identify them and deal with the sensitive data in the code. This may lead to mishandling sensitive data.

Communication Security

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
9.1.1	Verify that secured TLS is used for all client connectivity and does not fall back to insecure or unencrypted protocols.	Non-valid		TLS is not used for any connectivity. Instead, Flask is using WSGI for its development.
10.3.2	Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.	Not Applicable		
10.3.3	Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.	Not Applicable		

Malicious Code

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
10.3.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	Not Applicable		Application does not have a client or server auto-update feature
10.3.2	Verify that the application employs integrity protections, such as code signing or sub resource integrity. The application must not load or execute code from untrusted sources, such as loading includes modules, plugins, code, or libraries from untrusted sources or the Internet.	Non-valid	<script src="../static/js/jquery-ui.min.js"></script>	External scripts are used in some HTML files but no integrity attribute was added.
10.3.3	Verify that the application has protection from subdomain takeovers if the application relies upon DNS entries or DNS subdomains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.	Not Applicable		

Business Logic

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
11.1.1	Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.	Non-Valid		Since session tokens aren't maintained, it's possible for a user to start a transaction without logging in previously. The payment step is also skipped, simply reloading the page as if the order was successfully processed.
11.1.2	Verify the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.	Non-valid		The payment process skips straight back to a new checkout page.
11.1.3	Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.	Non-valid		The application uses and generates the appropriate amount of resources/transactions. Although being able to place an order without prior authentication does facilitate certain attacks, and unreasonable amounts of requests could be made.
11.1.4	Verify the application has sufficient anti-automation controls to detect and protect against data exfiltration, excessive business logic requests, excessive file uploads or denial of service attacks.	Non-valid		The application does not limit or employ any defense against such attacks.
11.1.5	Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modeling or similar methodologies.	Non-valid		It is possible to leave a review before logging in or purchasing a product, which could be used against the service provider.

Files and Resources

ASVS	Verification Requirement	Valid	Source Code Reference	Comment

12.1.1	Verify that the application will not accept large files that could fill up storage or cause a denial of service.	Not Applicable		No file handling was implemented

Since no file handling was implemented, the remaining ASVS of “Files and Resources” are also not applicable.

Web Services

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
13.1.1	Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.	Valid	hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')	Aside from the source code reference, there are no more instances of encryption or decryption
13.1.2	Verify that access to administration and management functions is limited to authorized administrators.	Not Applicable		
13.1.3	Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.	Valid	URL's do not show any sensitive information	

13.2.1	Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.	Not Applicable		
13.2.2	Verify that JSON schema validation is in place and verified before accepting input.	Not Applicable		
13.2.3	Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: double submit cookie pattern, CSRF nonces, or Origin request header checks.	Not Applicable		
13.3.1	Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.	Not Applicable		

Configuration

ASVS	Verification Requirement	Valid	Source Code Reference	Comment
14.2.1	Verify that all components are up to date, preferably using a dependency checker during build or compile time.	Non-valid		
14.2.2	Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.			
14.2.3	Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.	Not Applicable		
14.3.1	Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.	Valid		
14.3.2	Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.	Valid		
14.3.3	Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.	Non-valid		
14.4.1	Verify that every HTTP response contains a Content-Type header. text/*, */*+xml and application/xml content types should also specify a safe character set (e.g., UTF-8, ISO-8859-1).	Non-valid		

14.4.2	Verify that all API responses contain Content-Disposition: attachment; filename="api.json" header (or other appropriate filename for the content type).	Not applicable		
14.4.3	Verify that a Content Security Policy (CSP) response header is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.	Non-valid		
14.4.4	Verify that all responses contain a X-Content-Type-Options: nosniff header.	Non-valid		
14.4.5	Verify that a Strict-Transport-Security header is included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.	Non-valid		
14.4.6	Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".	Non-valid		
14.4.7	Verify that the content of a web application cannot be embedded in a third-party site by default and that embedding of the exact resources is only allowed where necessary by using suitable Content-Security-Policy: frame-ancestors and X-Frame-Options response headers.	Non-valid		
14.5.1	Verify that the application server only accepts the HTTP methods in use by the application/API, including pre-flight OPTIONS, and logs/alerts on any requests that are not valid for the application context.	Non-valid		

14.5.2	Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.	Not applicable		
14.5.3	Verify that the Cross-Origin Resource Sharing (CORS) Access-Control-Allow-Origin header uses a strict allow list of trusted domains and subdomains to match against and does not support the "null" origin.	Not Applicable		