NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Computer Science

Software Security and Data Privacy

TDT4237 - Exercise 2

Identifying Vulnerabilities in a Web Application

Group 19

Author(s):
Andrea Ritossa
Oskar Hetey
Nerea Francés Pérez

Table of Contents

1 Introduction

The following paper has been made as an assignment by three students enrolled in the course "Software Security and Data Privacy" at NTNU. The primary objective of this assessment was to identify any potential security vulnerabilities that may exist within the SecureHelp application, which operates on port 21019 on the server molde.ini.ntnu.no. This report will outline the vulnerabilities that were discovered during the testing process and will reference both black-box and white-box verification of the identified vulnerabilities.

The vulnerabilities that will be discussed in this report are classified based on the OWASP top-10 vulnerabilities and the Web Security Testing Guide 4.2 (WSTG). The team utilized the guidelines and instructions provided in the penetration testing manual while performing the testing.

The testing phase followed a period of information gathering on the SecureHelp application. The knowledge gathered during the initial phase was utilized by the students to conduct the penetration testing. The vulnerabilities identified during the testing process will be presented in a structured format with a unique code, in accordance with the OWASP guidelines, represented as (WSTG-XXXX-XX).

2 WSTG-IDNT-02 Test User Registration Process

One email-address can be used by several users. This may be helpful for our testing purposes but one user can create infinite (not really, but very many) accounts which should not be possible.

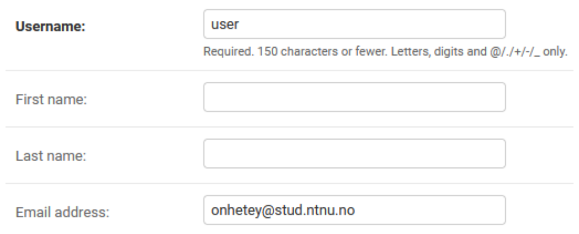
2.1 White-Box

This code shows where some limits are defined, like a 150 char limit on usernames. But validation for existing emails is missing.

```
// SECUREHELP/backend/apps/users/migrations/0001_initial.py
    operations = [
        migrations.CreateModel(
            name='User',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
                    serialize=False, verbose_name='ID')),
                ('password', models.CharField(max_length=128, verbose_name='password')),
                ('last_login', models.DateTimeField(blank=True, null=True,
                    verbose_name='last login')),
                ('is_superuser', models.BooleanField(default=False, help_text='Designates
                    that this user has all permissions without explicitly assigning
                    them.', verbose_name='superuser status')),
                ('username', models.CharField(error_messages={'unique': 'A user with that
                    username already exists.'}, help_text='Required. 150 characters or
                    fewer. Letters, digits and @/./+/-/_ only.', max_length=150,
                    unique=True,
                    validators=[django.contrib.auth.validators.UnicodeUsernameValidator()],
                    verbose_name='username')),
                ('first_name', models.CharField(blank=True, max_length=150,
                    verbose_name='first name')),
                ('last_name', models.CharField(blank=True, max_length=150,
                    verbose_name='last name')),
                ('email', models.EmailField(blank=True, max_length=254,
                    verbose_name='email address')),
                ('is_staff', models.BooleanField(default=False, help_text='Designates
                    whether the user can log into this admin site.', verbose_name='staff
                    status')),
                ('is_active', models.BooleanField(default=True, help_text='Designates
                    whether this user should be treated as active. Unselect this instead
                    of deleting accounts.', verbose_name='active')),
                ('date_joined', models.DateTimeField(default=django.utils.timezone.now,
                    verbose_name='date joined')),
                ('is_volunteer', models.BooleanField(default=False)),
                ('groups', models.ManyToManyField(blank=True, help_text='The groups this
                    user belongs to. A user will get all permissions granted to each of
                    their groups.', related_name='user_set', related_query_name='user',
                    to='auth.Group', verbose_name='groups')),
                ('user_permissions', models.ManyToManyField(blank=True,
                    help_text='Specific permissions for this user.',
                    related_name='user_set', related_query_name='user',
                    to='auth.Permission', verbose_name='user permissions')),
            ],
            options={
                'verbose_name': 'user',
                'verbose_name_plural': 'users',
                'abstract': False,
            },
            managers=[
                ('objects', django.contrib.auth.models.UserManager()),
            ],
        ),
    ]
```

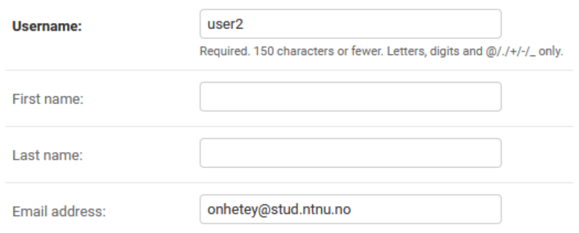
2.2 Black-Box

This shows a the user view page as an admin, showing two different users with the same email-address.



Username:	user Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
First name:	
Last name:	
Email address:	onhetey@stud.ntnu.no

Figure 1: user 1 with a specific email-address



Username:	user2 Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
First name:	
Last name:	
Email address:	onhetey@stud.ntnu.no

Figure 2: user 2 with the same email-address

3 WSTG-IDNT-05 Testing for Weak or Unenforced Username Policy

User account names are often highly structured (e.g. Joe Bloggs account name is jbloggs and Fred Nurks account name is fnurks) and valid account names can easily be guessed.

3.1 White-Box

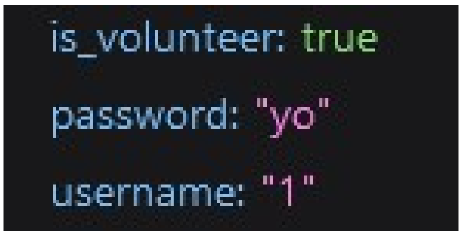
There are no restrictions for the user name apart from the minimum and maximum length. Therefore, the user can be easily guessable.

```
//SECUREHELP/backend/app/users/serializers.py

class RegisterSerializer(UserSerializer):
    """Serializer for user registration"""
    password = serializers.CharField(
        max_length=128, min_length=1, write_only=True, required=True)
    email = serializers.CharField(
        max_length=128, min_length=1, required=True)
```

3.2 Black-Box

This vulnerability can be exploited by registering using as username the number "1" and as a password "yo" which is very short and easy to crack after a few tries.



```
is_volunteer: true
password: "yo"
username: "1"
```

Figure 3: Screenshot of common user registered.

4 WSTG-ATHN-01 Testing for Credentials Transported over an Encrypted Channel

This test category refers to testing if sniffing account details. Since the connection is not secured but using HTTP as a protocol, it is possible to sniff login data on the network. If other people login, the username and password could be easily captured. Even if the website is not transporting data securely, login data could be encrypted.

4.1 White-Box

The code snipped shows the method submitting the login data

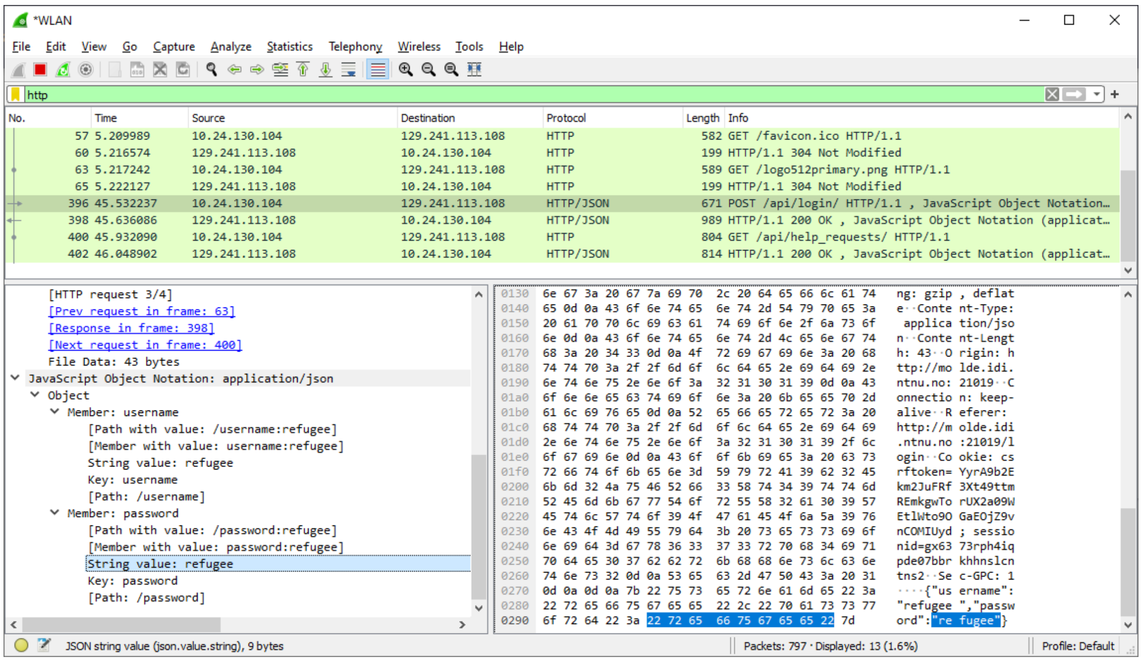
```
// SECUREHELP/frontend/src/components/LoginForm.jsx
const onSubmit = async (e) => {
    e.preventDefault();
    if (!username) {
      setUsernameErrorText("Please enter username");
    } else {
      setUsernameErrorText("");
    }
    if (!password) {
      setPasswordErrorText("Please enter password");
    } else {
      setPasswordErrorText("");
    }

    if (!username || !password) {
      return;
    }

    const request = { username: username, password: password };

    AuthService.login(request)
      .then((response) => {
        console.log("Signed in successfully");
        setUsername("");
        setPassword("");
        setUser(response.user);
        if (response.user.is_staff) {
          navigate("/approve-certifications");
        } else {
          navigate("/help-requests");
        }
        setAppSnackbarText("Signed in successfully");
        setAppSnackbarOpen(true);
      })
      .catch((err) => {
        console.log(err);
        setSnackbarOpen(true);
        setUsernameErrorText("Wrong username or password");
        setPasswordErrorText("Wrong username or password");
      });
  };
```

4.2 Black-Box


*WLAN
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.     Time       Source            Destination        Protocol    Length  Info
   57 5.209989     10.24.130.104     129.241.113.108    HTTP          582 GET /favicon.ico HTTP/1.1
   60 5.216574     129.241.113.108   10.24.130.104      HTTP          199 HTTP/1.1 304 Not Modified
   63 5.217242     10.24.130.104     129.241.113.108    HTTP          589 GET /logo512primary.png HTTP/1.1
   65 5.222127     129.241.113.108   10.24.130.104      HTTP          199 HTTP/1.1 304 Not Modified
  396 45.532237    10.24.130.104     129.241.113.108    HTTP/JSON     671 POST /api/login/ HTTP/1.1 , JavaScript Object Notation…
  398 45.636086    129.241.113.108   10.24.130.104      HTTP/JSON     989 HTTP/1.1 200 OK , JavaScript Object Notation (applicat…
  400 45.932090    10.24.130.104     129.241.113.108    HTTP          804 GET /api/help_requests/ HTTP/1.1
  402 46.048902    129.241.113.108   10.24.130.104      HTTP/JSON     814 HTTP/1.1 200 OK , JavaScript Object Notation (applicat…
```

```
    [HTTP request 3/4]
    [Prev request in frame: 63]
    [Response in frame: 398]
    [Next request in frame: 400]
    File Data: 43 bytes
v JavaScript Object Notation: application/json
  v Object
    v Member: username
        [Path with value: /username:refugee]
        [Member with value: username:refugee]
        String value: refugee
        Key: username
        [Path: /username]
    v Member: password
        [Path with value: /password:refugee]
        [Member with value: password:refugee]
        String value: refugee
        Key: password
        [Path: /password]
```

```
0130  6e 67 3a 20 67 7a 69 70  2c 20 64 65 66 6c 61 74   ng: gzip , deflat
0140  65 0d 0a 43 6f 6e 74 65  6e 74 2d 54 79 70 65 3a   e··Conte nt-Type:
0150  20 61 70 70 6c 69 63 61  74 69 6f 6e 2f 6a 73 6f    applica tion/jso
0160  6e 0d 0a 43 6f 6e 74 65  6e 74 2d 4c 65 6e 67 74   n··Conte nt-Lengt
0170  68 3a 20 34 33 0d 0a 4f  72 69 67 69 6e 3a 20 68   h: 43··O rigin: h
0180  74 74 70 3a 2f 2f 6d 6f  6c 64 65 2e 69 64 69 2e   ttp://mo lde.idi.
0190  6e 74 6e 75 2e 6e 6f 3a  32 31 30 31 39 0d 0a 43   ntnu.no: 21019··C
01a0  6f 6e 6e 65 63 74 69 6f  6e 3a 20 6b 65 65 70 2d   onnectio n: keep-
01b0  61 6c 69 76 65 0d 0a 52  65 66 65 72 65 72 3a 20   alive··R eferer: 
01c0  68 74 74 70 3a 2f 2f 6d  6f 6c 64 65 2e 69 64 69   http://m olde.idi
01d0  2e 6e 74 6e 75 2e 6e 6f  3a 32 31 30 31 39 2f 6c   .ntnu.no :21019/l
01e0  6f 67 69 6e 0d 0a 43 6f  6f 6b 69 65 3a 20 63 73   ogin··Co okie: cs
01f0  72 66 74 6f 6b 65 6e 3d  59 79 72 41 39 62 32 45   rftoken= YyrA9b2E
0200  6b 6d 32 4a 75 46 52 66  33 58 74 34 39 74 74 6d   km2JuFRf 3Xt49ttm
0210  52 45 6d 6b 67 77 54 6f  72 55 58 32 61 30 39 57   REmkgwTo rUX2a09W
0220  45 74 6c 57 74 6f 39 4f  47 61 45 4f 6a 5a 39 76   EtlWto9O GaEOjZ9v
0230  6e 43 4f 4d 49 55 79 64  3b 20 73 65 73 73 69 6f   nCOMIUyd ; sessio
0240  6e 69 64 3d 67 78 36 33  37 33 72 70 68 34 69 71   nid=gx63 73rph4iq
0250  70 64 65 30 37 62 62 72  6b 68 68 6e 73 6c 63 6e   pde07bbr khhnslcn
0260  74 6e 73 32 0d 0a 53 65  63 2d 47 50 43 3a 20 31   tns2··Se c-GPC: 1
0270  0d 0a 0d 0a 7b 22 75 73  65 72 6e 61 6d 65 22 3a   ····{"us ername":
0280  22 72 65 66 75 67 65 65  22 2c 22 70 61 73 73 77   "refugee ","passw
0290  6f 72 64 22 3a 22 72 65  66 75 67 65 65 22 7d      ord":"re fugee"}
```

```
JSON string value (json.value.string), 9 bytes        Packets: 797 · Displayed: 13 (1.6%)        Profile: Default
```

Figure 4: captured package using wireshark showing login credentials in plaintext

5 WSTG-ATHN-02 Testing for Default Credentials

The default password is too generic and guessable.

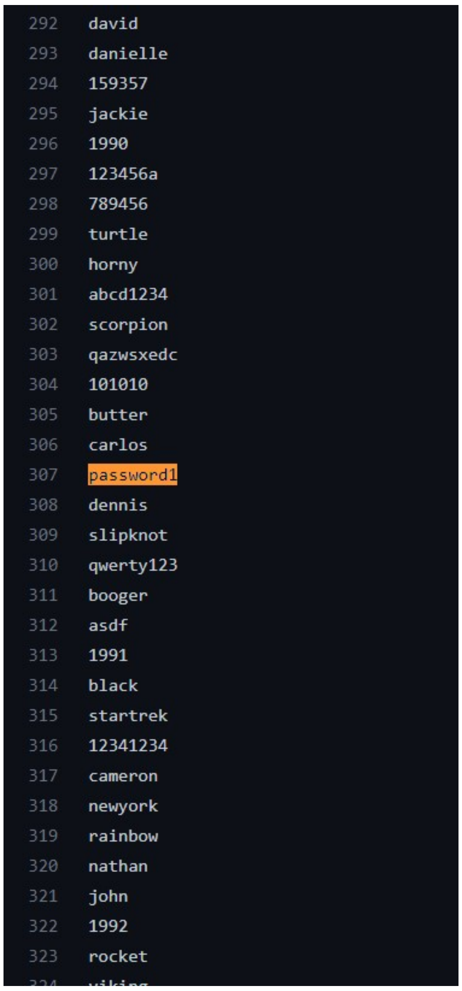
5.1 White-Box

```
//SECUREHELP/.env

DJANGO_SUPERUSER_PASSWORD=password1
DJANGO_SUPERUSER_USERNAME=admin1
DJANGO_SUPERUSER_EMAIL=admin@mail.com
```

5.2 Black-Box

Any user can gain access by trying different common user credentials. *password1* appears in the most common password databases which makes it more insecure.



```
292   david
293   danielle
294   159357
295   jackie
296   1990
297   123456a
298   789456
299   turtle
300   horny
301   abcd1234
302   scorpion
303   qazwsxedc
304   101010
305   butter
306   carlos
307   password1
308   dennis
309   slipknot
310   qwerty123
311   booger
312   asdf
313   1991
314   black
315   startrek
316   12341234
317   cameron
318   newyork
319   rainbow
320   nathan
321   john
322   1992
323   rocket
```

Figure 5: Screenshot of common password database.

6 WSTG-ATHN-07 Testing for Weak Password Policy

This Category is about Testing for Weak Password Policy and states 8 questions of which most turn out to show a weak password policy.

1. What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?
   - There are the following restrictions:
     (a) This password is too common
     (b) This password is entirely numeric
   - This means pure number-passwords are forbidden, as well as too common passwords
   - No limits on characters. Many special characters like äöü or 章内容 are allowed
   - The user is not required to use any special characters, numbers, or capitalized letters
2. How often can a user change their password? How quickly can a user change their password after a previous change? Users may bypass password history requirements by changing their password 5 times in a row so that after the last password change they have configured their initial password again.
   - A password can be changed unlimited times, even with the same reset code
   - A reset link is becoming invalid after some time, but as long as it is active it can be used several times to reset
   - Several links can be active at the same time
   - There is no delay on how quickly a password can be reset
3. When must a user change their password? Both NIST and NCSC recommend against forcing regular password expiry, although it may be required by standards such as PCI DSS.
   - Only when forgetting it, no other policy
   - This might actually suit the NIST and NSCS recommendations.
4. How often can a user reuse a password? Does the application maintain a history of the user's previous used 8 passwords?
   - Due to no password history, a password can be reused arbitrarily, even several times in a row
5. How different must the next password be from the last password?
   - Due to no password history, old passwords can be retaken
6. Is the user prevented from using his username or other account information (such as first or last name) in the password?
   - No, using the same string for username and password is allowed, there are no regulations.
7. What are the minimum and maximum password lengths that can be set, and are they appropriate for the sensitivity of the account and application?
   - We tested for the length of 1 to 1 Million chars, which are allowed, so there is no length regulation of relevance.
8. Is it possible set common passwords such as Password1 or 123456?
   - Some Passwords like 123456, password, Password1 ... are not allowed, due to the common password validator. Using the letter y would be not allowed, the two letters yo are not filtered out, so the limitation is low.

6.1 White-Box

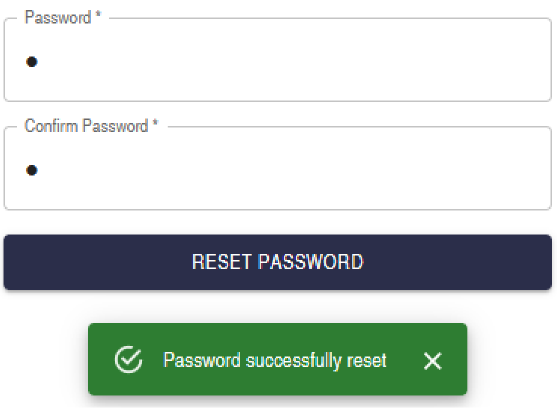
Most of the listed password regulations where tried out using a black box test, so not all points will be proven by code. The following snipped is showing the definition of the only two password validators being used.

```
// SECUREHELP/backend/securehelp/settings.py

# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settings/#auth-password-validators


AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
]
```

6.2 Black-Box



Password successfully reset

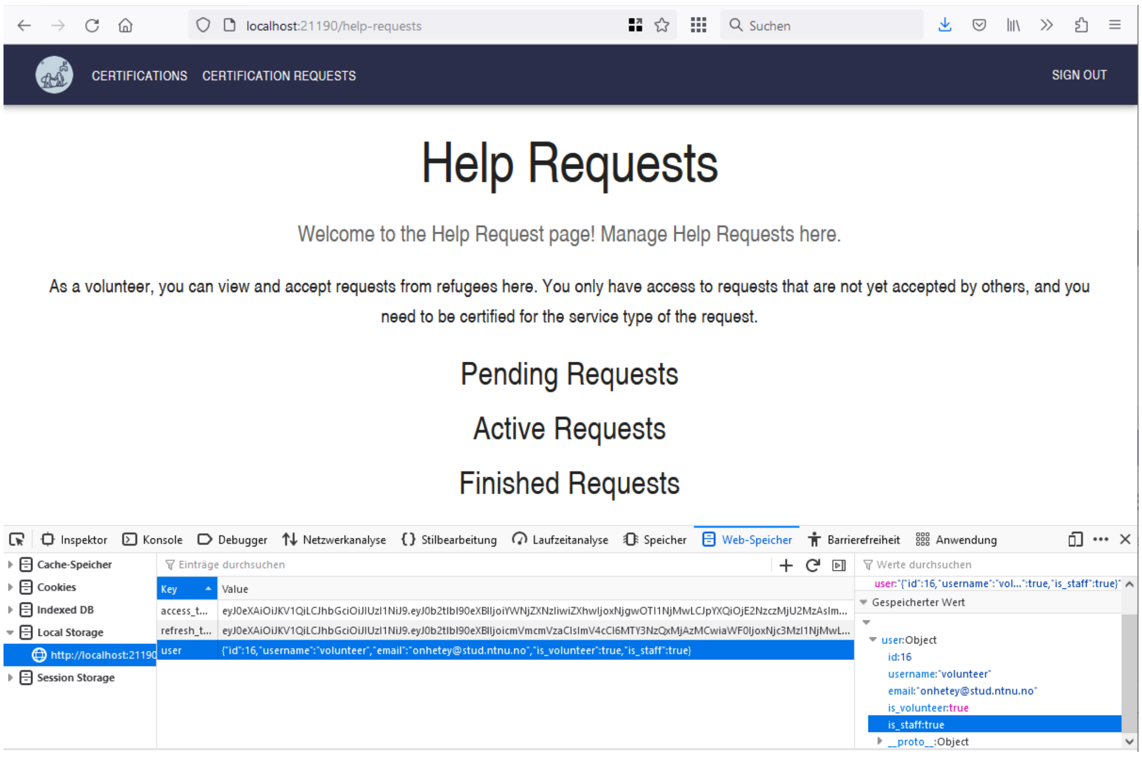
Figure 6: Screenshot of a one-char-password being successfully changed ("ö")

7 WSTG-ATHZ-03 Testing for Privilege Escalation

Privilege escalation can be achieved by changing the passing parameters and reloading the page. To do this, it is sufficient to use the developer tools available in Firefox. Changing the value of is_staff from False to True will load the menu, which should only be visible to administrators. Graphics in the Black-Box test show that the Certification Requests button becomes visible. This vulnerability can be seen as a vertical escalation, as the user can now accept or deny Certification Requests, but only for themselves and not other users. It is also possible to do so without the staff status, the menu is not visible but one can access the url anyway, as shown in the Black-Box test.

7.1 Black-Box

After using the browser tools to change the stored property is_staff to true, as shown in the image, the staff menu, which is different to the volunteer menu, shows up.



localhost:21190/help-requests

CERTIFICATIONS CERTIFICATION REQUESTS SIGN OUT

Help Requests

Welcome to the Help Request page! Manage Help Requests here.

As a volunteer, you can view and accept requests from refugees here. You only have access to requests that are not yet accepted by others, and you need to be certified for the service type of the request.

Pending Requests

Active Requests

Finished Requests

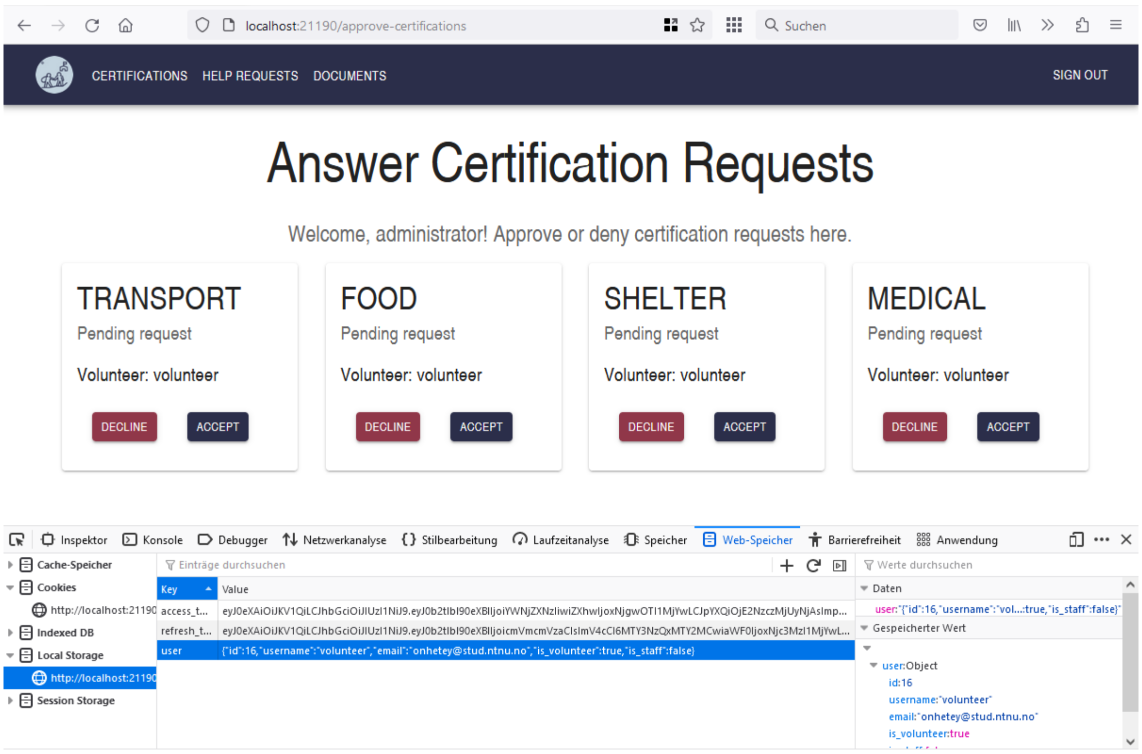
Local Storage – http://localhost:21190

Key	Value
access_t...	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzliwiZXhwljoxNjgwOTI1NjMwLCJpYXQiOjE2NzczMjU2MzAslm...
refresh_t...	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaClslmV4cCI6MTY3NzQxMjAzMCwiaWF0IjoxNjc3MzI1NjMwL...
user	{"id":16,"username":"volunteer","email":"onhetey@stud.ntnu.no","is_volunteer":true,"is_staff":true}

```
user:"{"id":16,"username":"vol...":true,"is_staff":true}"
Gespeicherter Wert
  user:Object
    id:16
    username:"volunteer"
    email:"onhetey@stud.ntnu.no"
    is_volunteer:true
    is_staff:true
    __proto__:Object
```

Figure 7: Left side: logged in as volunteer, Right side: logged in as admin

This Screenshot shows the regular menu for volunteers. The developer tools reveal that the variable is_staff is False (as per default). Nonetheless the URL /approve-certifications/ can be invoked, resulting in the availability of viewing and approving (or denying) the own requests of certification.


localhost:21190/approve-certifications

CERTIFICATIONS HELP REQUESTS DOCUMENTS SIGN OUT

Answer Certification Requests

Welcome, administrator! Approve or deny certification requests here.

TRANSPORT	FOOD	SHELTER	MEDICAL
Pending request	Pending request	Pending request	Pending request
Volunteer: volunteer	Volunteer: volunteer	Volunteer: volunteer	Volunteer: volunteer
DECLINE ACCEPT	DECLINE ACCEPT	DECLINE ACCEPT	DECLINE ACCEPT

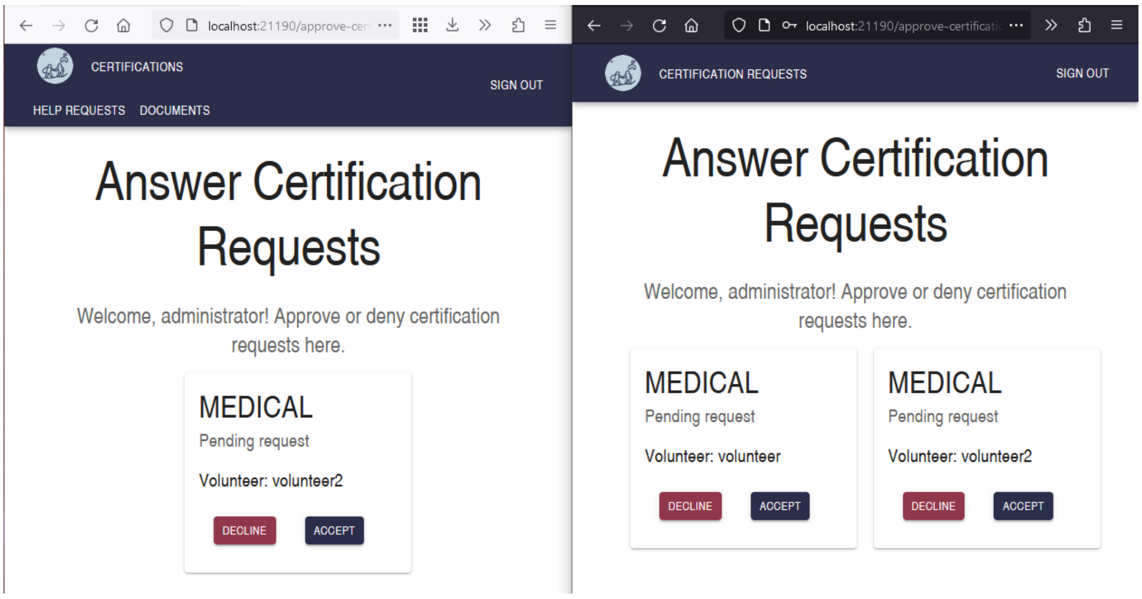
Inspektor Konsole Debugger Netzwerkanalyse Stilbearbeitung Laufzeitanalyse Speicher Web-Speicher Barrierefreiheit Anwendung

Key	Value
access_t...	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNjZXNzIiwiZXhwIjoxNjgwOTI1MjYwLCJpYXQiOjE2NzczMjUyNjAsImp...
refresh_t...	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoicmVmcmVzaCIsImV4cCI6MTY3NzQxMTY2MCwiaWF0IjoxNjc3MzI1MjYwL...
user	{"id":16,"username":"volunteer","email":"onhetey@stud.ntnu.no","is_volunteer":true,"is_staff":false}

```
Daten
  user:"{"id":16,"username":"vol...:true,"is_staff":false}"
Gespeicherter Wert
  user:Object
    id:16
    username:"volunteer"
    email:"onhetey@stud.ntnu.no"
    is_volunteer:true
```

Figure 8: Left side: logged in as volunteer, Right side: logged in as admin

The screenshot shows that it is possible to see (and approve) own certificate requests. The session on the right is showing the interface when being logged in as admin, having the rights to view and modify all outstanding requests.


localhost:21190/approve-cer

CERTIFICATIONS HELP REQUESTS DOCUMENTS SIGN OUT

Answer Certification Requests

Welcome, administrator! Approve or deny certification requests here.

MEDICAL
Pending request
Volunteer: volunteer2
DECLINE ACCEPT

localhost:21190/approve-certificati

CERTIFICATION REQUESTS SIGN OUT

Answer Certification Requests

Welcome, administrator! Approve or deny certification requests here.

MEDICAL	MEDICAL
Pending request	Pending request
Volunteer: volunteer	Volunteer: volunteer2
DECLINE ACCEPT	DECLINE ACCEPT

Figure 9: Left side: logged in as volunteer, Right side: logged in as admin

8 WSTG-SESS-05 Testing for Cross Site Request Forgery

8.1 White-Box

The input field for loading the documents is missing the usage of **CSRF tokens**.

```
//Missing of input validation both here and in DocumentService
const uploadFile = (e) => {
  e.preventDefault();

  let formData = new FormData();

  formData.append("document", selectedFile);
  DocumentService.UploadDocument(formData)
    .then((response) => {
      console.log("File uploaded");
      OpenSnackbar("File uploaded");
      DocumentService.GetDocumentInfos().then((c) => setDocuments(c));
    })
    .catch((error) => console.error(error));
};
```

8.2 Black-Box

This can be exploited with a CSRF attack:
A cross-site request forgery attack involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. [OWASP ZAP]

9 WSTG-SESS-07 Testing For Session Timeout

9.1 White-Box

The test objective is to verify the existence of a hard session timeout in the SecureHelp website. This is crucial to prevent unauthorized access to sensitive information and protect against attacks such as session hijacking and cookie replay. A hard session timeout automatically logs out the user after a specified period of inactivity. However, the SecureHelp website currently does not have any session form for session timeout, making it possible to reuse the session. This increases the risk of stealing and reusing a valid sessionID token, which could lead to unauthorized access to the website.

Missing line in settings.py:

```
#...
SESSION_EXPIRE_AT_BROWSER_CLOSE
#...
```

9.2 Black-Box

In its current state, the SecureHelp website only logs out the user when they manually log out from the website, and not when the session times out due to inactivity. This means that if a user forgets to log out or leaves the website without logging out, their session may remain active for an indefinite period of time. This increases the risk of session hijacking, where an attacker can gain access to the user's session and perform malicious actions.

A common scenario where session timeout management is important is when a public computer is used to access private information such as web mail or online bank accounts. If the user forgets to log out and the application does not enforce a session timeout, an attacker could potentially access the same account by using the "back" button on the browser. [**OWASP**]

10 WSTG-CONF-02 Test Application Platform Configuration

10.1 White-Box

The django server does not have a basic logging configuration in the "settings.py" file because the "LOGGING" variable that defines the logging settings is undefined. An example of the right code to implement:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'formatters': {
        # This is the default formatter
        'default': {
            'format': '%(asctime)s %(name)-12s %(levelname)-8s %(message)s'
        },
    },
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
            'formatter': 'default',
        },
        'file': {
            'level': 'WARNING',
            'class': 'logging.StreamHandler',
            'filename': 'os.path.join(BASE_DIR, 'debug.log')',
        },
        'mail_admins': {
            'level': 'ERROR',
            'class': 'django.utils.log.AdminEmailHandler',
        },
    },
    'loggers': {
        '': {
            'level': LOGLEVEL,
            'handlers': ['console', 'file', 'mail_admins'],
        },
    }
```

10.2 Black-Box

A feasible attack is to brute force the login to the site: repeatedly sending requests with different credentials until we get inside. By not logging these requests, the administrator can not notice the attack or be able to block it.

To prevent this, the failed login attempts should be logged and some methods to throttle or stop the attacker, such as blocking their IP address, should be implemented.

11 WSTG-CONF-07 Test HTTP Strict Transport Security

The application is missing the enforcement of the HTTP Strict Transport Security (HSTS). HSTS informs the browser through, the use of a special response header, that it should never establish a connection to the specified domain servers using un-encrypted HTTP.

11.1 White-Box

In settings.py the appropriate middleware is declared:

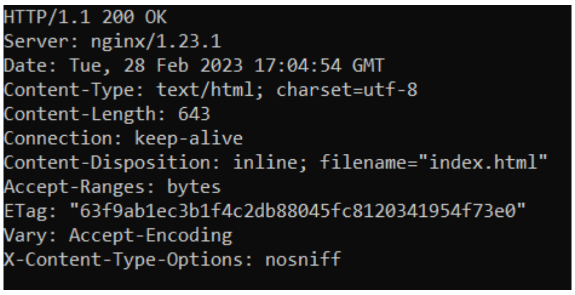
```
# settings.py
MIDDLEWARE = [
    # ...
    'django.middleware.security.SecurityMiddleware', # already present
    # ...
]
```

but the HSTS options are missing:

```
# Set HSTS options ex > MISSING
SECURE_HSTS_SECONDS = 31536000 # one year
SECURE_HSTS_INCLUDE_SUBDOMAINS = True
SECURE_HSTS_PRELOAD = True
```

11.2 Black-Box

An attacker could identify this vulnerability by executing:
curl -s -D- http://molde.idi.ntnu.no:21019



```
HTTP/1.1 200 OK
Server: nginx/1.23.1
Date: Tue, 28 Feb 2023 17:04:54 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 643
Connection: keep-alive
Content-Disposition: inline; filename="index.html"
Accept-Ranges: bytes
ETag: "63f9ab1ec3b1f4c2db88045fc8120341954f73e0"
Vary: Accept-Encoding
X-Content-Type-Options: nosniff
```

Figure 10: Missing ”Strict-Transport-Security”

We can see that the Strict-Transport-Security header is missing. This results in a vulnerability as an attacker could perform a man in the middle attack [MDN], that is based on directing visitors to a malicious site instead of the secure version of the original site.

An example scenario:
Step 1: The attacker sets up a fake WiFi hotspot near a location where people often connect to the WiFi network.
Step 2: The user connect to the fake WiFi and tries to access the page.
Step 3: As the website does not have HSTS enabled the attacker is able to intercept the HTTP request and prevent it from being redirected to HTTPS by the website.
Step 4: The attacker sends back a fake HTTP response with malicious content or links to the user.
Step 5: The user interacts with the fake HTTP response and exposes their sensitive data.

12 WSTG-ERRH-01 Testing for Improper Error Handling

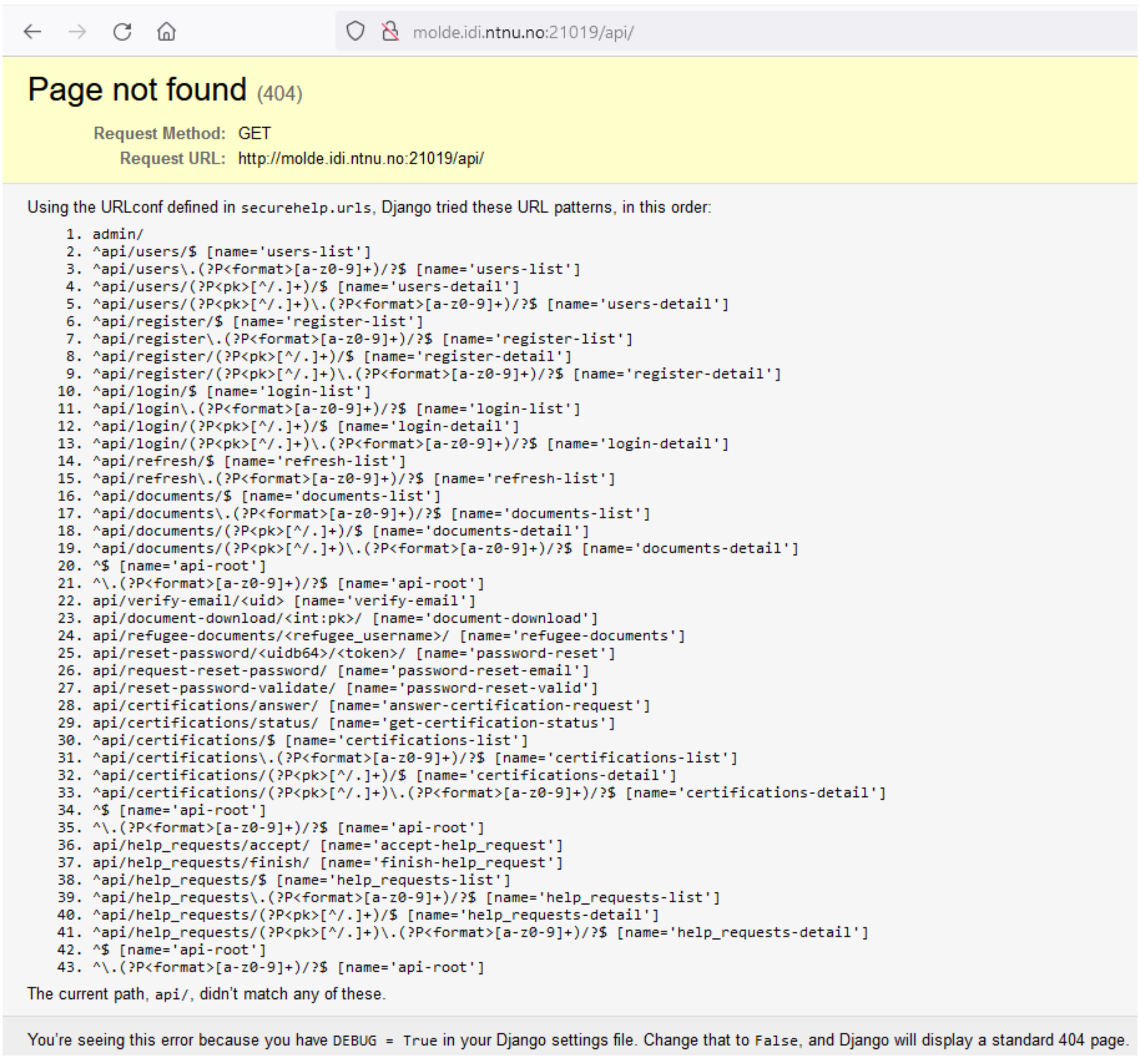
By trying out different file paths, one can find /api/, which reveils a lot of information because the DEBUG-Mode is turned on. This is not restricted to logged in users but accessible for everyone. An attacker can gather information about the structure of the API being used internally which should only be available to admins.

12.1 White-Box

The Debug property can be toggled in line 51

```
// SECUREHELP/backend/securehelp/settings.py
DEBUG = True
```

12.2 Black-Box



molde.idi.ntnu.no:21019/api/

Page not found (404)

Request Method: GET
Request URL: http://molde.idi.ntnu.no:21019/api/

Using the URLconf defined in `securehelp.urls`, Django tried these URL patterns, in this order:

```
 1. admin/
 2. ^api/users/$ [name='users-list']
 3. ^api/users\.(?P<format>[a-z0-9]+)/?$ [name='users-list']
 4. ^api/users/(?P<pk>[^/.]+)/$ [name='users-detail']
 5. ^api/users/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='users-detail']
 6. ^api/register/$ [name='register-list']
 7. ^api/register\.(?P<format>[a-z0-9]+)/?$ [name='register-list']
 8. ^api/register/(?P<pk>[^/.]+)/$ [name='register-detail']
 9. ^api/register/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='register-detail']
10. ^api/login/$ [name='login-list']
11. ^api/login\.(?P<format>[a-z0-9]+)/?$ [name='login-list']
12. ^api/login/(?P<pk>[^/.]+)/$ [name='login-detail']
13. ^api/login/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='login-detail']
14. ^api/refresh/$ [name='refresh-list']
15. ^api/refresh\.(?P<format>[a-z0-9]+)/?$ [name='refresh-list']
16. ^api/documents/$ [name='documents-list']
17. ^api/documents\.(?P<format>[a-z0-9]+)/?$ [name='documents-list']
18. ^api/documents/(?P<pk>[^/.]+)/$ [name='documents-detail']
19. ^api/documents/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='documents-detail']
20. ^$ [name='api-root']
21. ^\.(?P<format>[a-z0-9]+)/?$ [name='api-root']
22. api/verify-email/<uid> [name='verify-email']
23. api/document-download/<int:pk>/ [name='document-download']
24. api/refugee-documents/<refugee_username>/ [name='refugee-documents']
25. api/reset-password/<uidb64>/<token>/ [name='password-reset']
26. api/request-reset-password/ [name='password-reset-email']
27. api/reset-password-validate/ [name='password-reset-valid']
28. api/certifications/answer/ [name='answer-certification-request']
29. api/certifications/status/ [name='get-certification-status']
30. ^api/certifications/$ [name='certifications-list']
31. ^api/certifications\.(?P<format>[a-z0-9]+)/?$ [name='certifications-list']
32. ^api/certifications/(?P<pk>[^/.]+)/$ [name='certifications-detail']
33. ^api/certifications/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='certifications-detail']
34. ^$ [name='api-root']
35. ^\.(?P<format>[a-z0-9]+)/?$ [name='api-root']
36. api/help_requests/accept/ [name='accept-help_request']
37. api/help_requests/finish/ [name='finish-help_request']
38. ^api/help_requests/$ [name='help_requests-list']
39. ^api/help_requests\.(?P<format>[a-z0-9]+)/?$ [name='help_requests-list']
40. ^api/help_requests/(?P<pk>[^/.]+)/$ [name='help_requests-detail']
41. ^api/help_requests/(?P<pk>[^/.]+)\.(?P<format>[a-z0-9]+)/?$ [name='help_requests-detail']
42. ^$ [name='api-root']
43. ^\.(?P<format>[a-z0-9]+)/?$ [name='api-root']
```

The current path, `api/`, didn't match any of these.

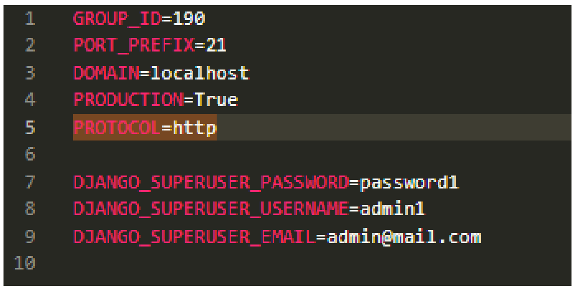
You're seeing this error because you have `DEBUG = True` in your Django settings file. Change that to `False`, and Django will display a standard 404 page.

Figure 11: Screenshot of path /api/ as seen in Firefox

13 WSTG-CRYP-03 Testing for Sensitive Information Sent via Unencrypted Channel

13.1 White-Box

SecureHelp utilizes the unencrypted http protocol instead of the https, a well known vulnerability.



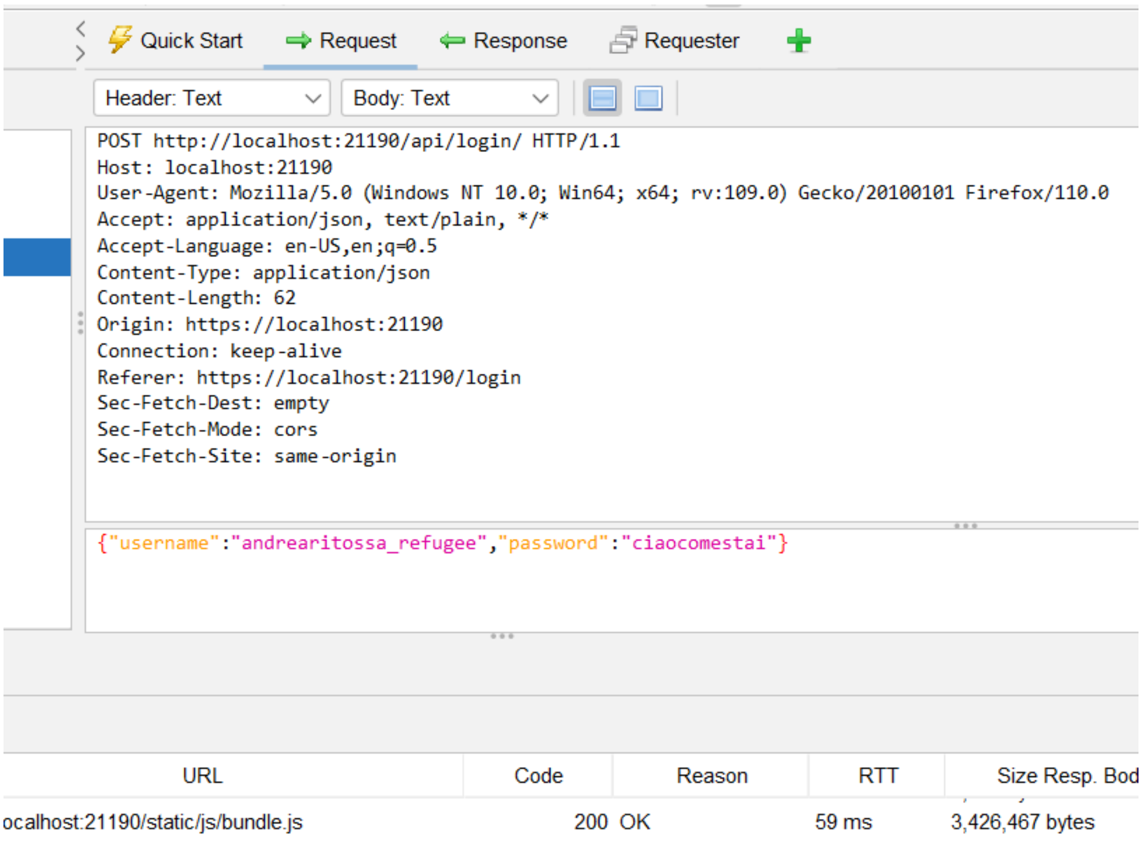
```
GROUP_ID=190
PORT_PREFIX=21
DOMAIN=localhost
PRODUCTION=True
PROTOCOL=http

DJANGO_SUPERUSER_PASSWORD=password1
DJANGO_SUPERUSER_USERNAME=admin1
DJANGO_SUPERUSER_EMAIL=admin@mail.com
```

Figure 12: Variable PROTOCOL in .env setted to HTTP

13.2 Black-Box

Furthermore, combining this with [SESS-04] in the login method the application sends username and password in plain text, and this combination of vulnerabilities make exposes to simple interception:



```
POST http://localhost:21190/api/login/ HTTP/1.1
Host: localhost:21190
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/110.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Content-Type: application/json
Content-Length: 62
Origin: https://localhost:21190
Connection: keep-alive
Referer: https://localhost:21190/login
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{"username":"andrearitossa_refugee","password":"ciaocomestai"}
```

URL	Code	Reason	RTT	Size Resp. Bod
ocalhost:21190/static/js/bundle.js	200	OK	59 ms	3,426,467 bytes

Figure 13: Plain username and password

14 WSTG-CRYP-04 Testing for Weak Encryption

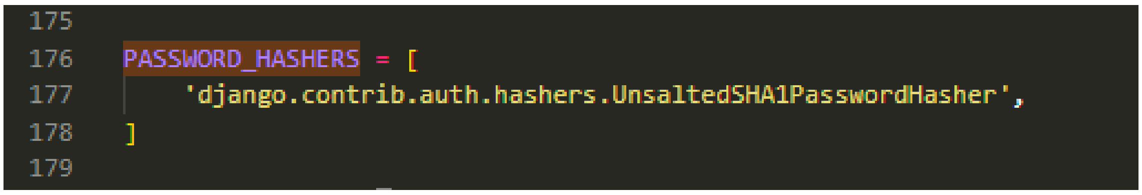
14.1 White-Box

When registering a user, one of the attributes is its user ID: *uid*, which is encoded using Base64 and can be easily decoded externaly.

```
//SECUREHELP/backend/app/users/serializers.py

uid = urlsafe_base64_encode(user.username.encode())
```

Also line 176 in settings.py shows that password are hashed using the method $UnsaltedSHA1PasswordHasher$:

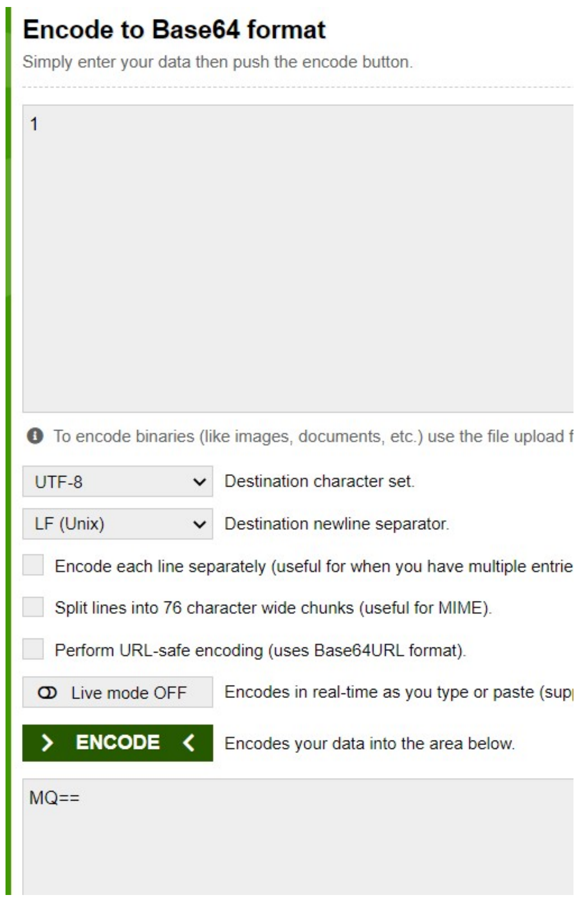


```
175
176    PASSWORD_HASHERS = [
177        'django.contrib.auth.hashers.UnsaltedSHA1PasswordHasher',
178    ]
179
```

Figure 14: PASSWORD HASHERS

14.2 Black-Box

When *uid* is decoded, it can be seen that it's a number and users' ids go from 1 and on. So we can know that the admin user has as *uid* the number 1 as it's the first user. This can be easily guessed and encoded with Base64 and we will find that the *uid* of the admin is MQ.



Encode to Base64 format

Simply enter your data then push the encode button.

1

To encode binaries (like images, documents, etc.) use the file upload f

UTF-8 Destination character set.

LF (Unix) Destination newline separator.

Encode each line separately (useful for when you have multiple entrie

Split lines into 76 character wide chunks (useful for MIME).

Perform URL-safe encoding (uses Base64URL format).

Live mode OFF Encodes in real-time as you type or paste (sup

ENCODE Encodes your data into the area below.

MQ==

Figure 15: Screenshot of encoding ”1” with Base64.

15 WSTG-CLNT-5 CSS Injection

15.1 White-Box

In HelpRequest.jsx (frontend), the *description* value is set through html on the website.

```
// SECUREHELP/frontend/src/components/HelpRequest.jsx

<div dangerouslySetInnerHTML={{ __html: helpRequest.description }}></div>
```

dangerouslySetInnerHTML allows the injection of raw HTML content into a component. It's considered dangerous because it can allow an attacker to inject malicious HTML or scripts into the application.

15.2 Black-Box

In the *description* input of a new request, it is possible to inject css code and change the style.

For example, writing:

```
<style>
    body {
      background-color: red;
    }
</style>
```

We managed to change the background color to red:

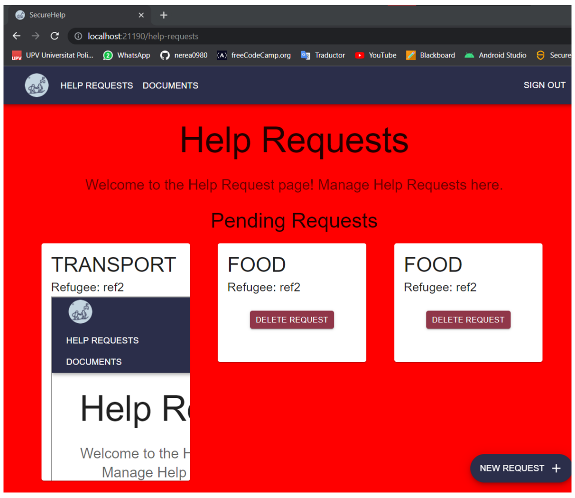


Figure 16: Screenshot of the SecureHelp request section after injecting CSS.

16 WSTG-CLNT-09 Testing for Clickjacking

16.1 White-Box

X-Frame-Options HTTP header is set in the backend but not in the frontend, making it vulnerable to Clickjacking attacks.

```
// SECUREHELP/backend/securehelp/settings.py

# Application definition

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    "corsheaders.middleware.CorsMiddleware",
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware'
]
```

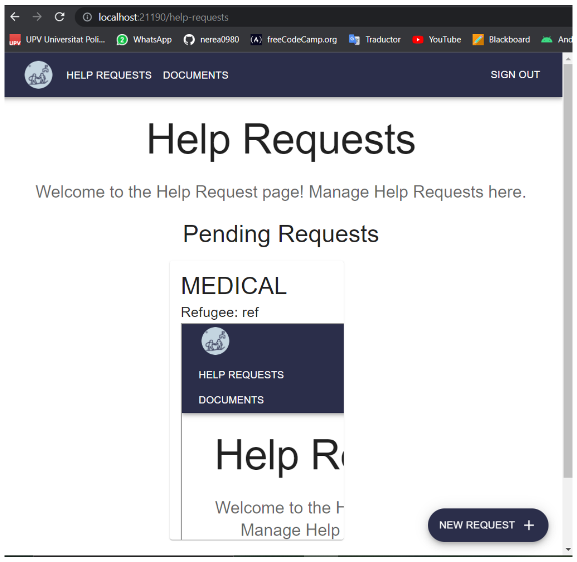
The X-Frame-Options HTTP header is a security feature that helps to protect web applications from clickjacking attacks.

16.2 Black-Box

In the *description* input of a New Request, code injection can be performed. In this case, writing

```
<iframe src=" http://localhost:21190/help-requests" width="500" height="500"></iframe>
```

, we could set a second window.



Help Requests

Welcome to the Help Request page! Manage Help Requests here.

Pending Requests

MEDICAL

Refugee: ref

Help R

Welcome to the H
Manage Help

Figure 17: Screenshot of the SecureHelp request feature after injecting Clickjacking.

17 WSTG-CLNT-12 Browser Storage

17.1 White-Box

Use of **localStorage** property. Entries to localStorage persist even when the browser window closes.

```
// SECUREHELP/frontend/src/components/Home.jsx

  useEffect(() => {
    const loggedUserJSON = window.localStorage.getItem("user");
    if (loggedUserJSON) {
      const user = JSON.parse(loggedUserJSON);
      setUser(user);
      if (user.is_staff) {
        navigate("/approve-certifications");
      } else {
        navigate("/help-requests");
      }
    }
  }, [navigate, setUser]);
```

The code retrieves the *user* key from the browser's local storage. Client-side authentication can be easily bypassed, as the user can modify the local storage data or intercept and manipulate the requests and responses using tools like browser extensions, proxy servers, or intercepting proxies.

Sensitive user data, such as passwords or tokens, should be stored in a more secure storage mechanism, such as cookies with the *HttpOnly* and Secure flags or server-side storage with proper encryption.

Applications should be storing sensitive data on the server-side, and not on the client-side, in a secured manner following best practices.

18 Conclusion

In conclusion, the code vulnerability report has identified several potential security issues that could compromise the integrity and confidentiality of the system. The vulnerabilities include improper input validation, lack of authentication, and weak encryption, among others. These vulnerabilities could be exploited by attackers to gain unauthorized access to sensitive data or execute arbitrary code.

There are more vulnerabilities to find, but due to little to no previous knowledge on some of the tools and frameworks we are happy to present these results.

References

[1] OWASP Foundation, *https://owasp.org/www-project-web-security-testing-guide/*.

[2] OWASP ZAP, *https://www.zaproxy.org/docs/*.

[3] Mozilla Developer Network Web Docs, *https://developer.mozilla.org/en-US/*

[4] Django documentation, *https://docs.djangoproject.com/en/4.1/*

[5] Postaman learning, *https://learning.postman.com/docs/sending-requests/capturing-request-data/capturing-http-requests/*

[6] GitHub common passwords, *https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-10000.txt*

[7] Base64 encode, *https://www.base64encode.org/*