NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4237 Software Security and Data Privacy

Exercise 2. Finding Vulnerabilities

Group 97

Author(s):
Iannello Stefano
Mandana Eleni

Table of Contents

List of Figures

1 Introduction

The purpose of this report is to show the identified vulnerabilities and potential exploits (if possible with tools, with an explanation otherwise) of the SecureHelp application, using the OWASP Web Security Testing Guide as a guideline. The authors are Computer Science students who are learning and experimenting with software security for the first time.
Our tactic for filling this report was first to browse through the WSTG list in order to know how and what we are looking for. After that, we tried to identify vulnerabilities in the code files. When a vulnerability arose we tried to think of ways to exploit it. When we had a complete overview of the vulnerability, it was added on the report.
At this point, it is important to mention that we could not perform any DoS or SQL injection attacks for the black box section. We were testing the website from two different machines on Fedora 37 and Windows 10. We used both http://localhost:21097/ and molde.idi.ntnu.no:21097/ to test most of our vulnerabilities. All the tools that were used in this assignment can be found in our references.
The following section contains a brief introduction to the authors and after that, you can find the vulnerabilities we discovered. We start from the WSTG-complied vulnerabilities, ordered by category code. For example, Authentication Testing (4.4) is before Input Validation Testing (4.7). We followed the same tactic for vulnerabilities between the same category. After WSTG complied vulnerabilities, OWASP-complied vulnerabilities can be found with arithmetical order.

1.1 About the authors

Stefano Iannello: I am currently a first-year Master's student in Computer Science. During my bachelor studies, my academic focus was on Object-Oriented Programming (OOP) using the Java language. I am excited to share that this is my first experience in the realm of software security, which I find to be both engaging and challenging.

Eleni Mandana: I am a final-year computer science student at Aristotle University of Thessaloniki (AUTh) with a specialization in Data and Web Management. My primary interests lie in machine learning, information retrieval, and database technology. Throughout my academic career, I have been highly motivated to explore and develop my expertise in these areas.

Konstantinos Giotakos: I am a Computer Science Student at AUTh, with no specialization yet. I am interested in Artificial Intelligence and Machine Learning.

2 WSTG-ATHN-07

Vulnerabilities in this field include Weak Password Policies [1]. One of the bad practices of the given website is that allows short and common passwords. For example, you are allowed to have a password such as '1a'. The only prerequisite is that it contains at least a number and a letter.

2.1 White-Box

The problem is that there are not enough checks when a user sets a password. This can be seen in the following line of code.

```
# backend/apps/users/serializers.py
# settings.py
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
]
```

2.2 Black-Box

In order to prove that a password that short is really easy to crack we had the scenario that the attacker knows the user's username. We generated passwords with a python script, the password length could be a combination of numbers and/or caps and/or lower letters. The assumption that the user has a weak password has been made. Then with the assistance of Burp Suite Community Edition [2] and the FoxyProxy plugin (Mozilla) [3], we activated the burp proxy through FoxyProxy and in the Burp application we activated Intercept. We made a failed attempt to gain access to the request. We send this line to the intruder section of Burp. We set only the password field as a variable and we uploaded as payload the contents of the file that contained the generated passwords. We wanted to try passwords when we were getting the response "No active account found with the given credentials", which was retrieved from the Network traffic, in the response field in the inspection section of the website. This was set in the Burp in the setting¿ GREP match section. Everything else was cleared from this field. At this point, we were ready to perform the brute force attack to retrieve the password of the user "testest".

```
POST /api/login/ HTTP/1.1
Host: molde.idi.ntnu.no:21097
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
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 37
Origin: http://molde.idi.ntnu.no:21097
Connection: close
Referer: http://molde.idi.ntnu.no:21097/login
Cookie: nmstat=b1918690-8de0-541a-3ced-58e8964b1515

{"username":"testest","password":"test"}
```

The results when a password was incorrect were the following. As we can see in the details there is no user with the password in the payload value (the orange line). We can notice that the section

"No active account found with the given credentials" is True (1).

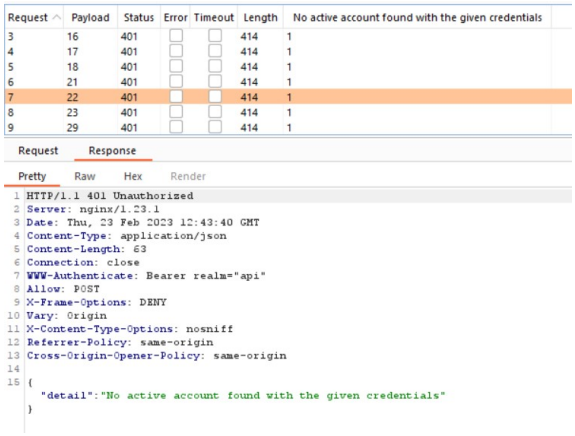

Figure 1: Incorrect Password attempt

But while observing this window we noticed one line with the "No active account found with the given credentials" value to be empty. By clicking this line, it was confirmed that we had managed to gain access with the value in the payload section.

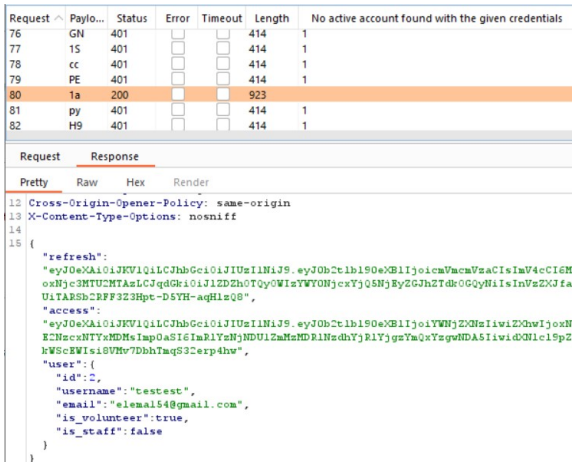

Figure 2: Correct Password attempt

3 WSTG-ATHN-07

A vulnerability in this field arose because the application does not enforce a password policy that requires users to choose a new password when their old password expires or has been used for a certain period [1].

3.1 White-Box

The application is allowing the user to set a new password that is the same as the previous one.

```
# backend\apps\users\serializers.py

class SetNewPasswordSerializer(serializers.Serializer):
    # code
    def validate(self, attrs):
    password = attrs.get('password')
    password1 = attrs.get('password1')
    token = attrs.get('token')
    uid = attrs.get('uid')

    id = force_str(urlsafe_base64_decode(uid))
    user = get_user_model().objects.get(id=id)
    errorMessage = dict()

    try:
        # validate password using django's validate_password
        validate_password(password)
    except exceptions.ValidationError as error:
        errorMessage['message'] = list(error.messages)

    if errorMessage: # if there is an error, raise it
        raise serializers.ValidationError(errorMessage)

    if password != password1: # check if passwords match
        raise serializers.ValidationError("Passwords must match!")

    user.set_password(password) # set new password
    user.save()

    return user
```

3.2 Black-Box

Allowing users to set their new password as the same as their old password is a security risk. If an attacker has obtained a user's password through a data breach or other means, and the user changes their password to the same password as before, the attacker will still be able to access the user's account. This makes it easier for attackers to gain access to sensitive information or perform malicious actions using the compromised account.

4 WSTG-ATHN-07

Another vulnerability of this category came up because the application does not implement proper controls to limit the number of password reset requests [1] that can be made within a certain time period, which can allow an attacker to perform a brute-force attack on a user's account.

4.1 White-Box

There is no rate limit on how many times a user can ask for a password reset email.

```
# backend/apps/users/views.py

class PasswordResetEmailView(generics.GenericAPIView):
    """View for sending password reset email"""
    serializer_class = ResetPasswordSerializer

    def post(self, request):
        # Check if email and username are provided
        if request.data.get("email") and request.data.get("username"):
            email = request.data["email"]
            username = request.data["username"]

            if get_user_model().objects.filter(email=email, username=username).exists():
                user = get_user_model().objects.get(email=email, username=username)

                uid = urlsafe_base64_encode(force_bytes(user.pk))
                domain = get_current_site(request).domain
                token = PasswordResetTokenGenerator().make_token(user) # Generate token
                link = reverse(
                    'password-reset', kwargs={"uidb64": uid, "token": token})

                url = f"{settings.PROTOCOL}://{domain}{link}"
                email_subject = "Password reset"
                mail = EmailMessage(
                    email_subject,
                    url,
                    None,
                    [email],
                )
                mail.send(fail_silently=False)
        return Response({'success': "If the user exists, you will shortly receive a link
            to reset your password."}, status=status.HTTP_200_OK)
```

4.2 Black-Box

If the password reset functionality has no rate limiting, an attacker could potentially use it to launch a denial-of-service (DoS) attack on the system by sending a large number of password reset requests to the server, overwhelming its resources and making it unavailable to legitimate users. We are not allowed to perform a DoS attack.

5 WSTG-ATHZ-01

This is a type of vulnerability that occurs when an application allows a user to specify a file path that is not properly sanitized, resulting in unauthorized access to files on the system [4]. Attackers can exploit this vulnerability to access sensitive files, including configuration files, credential files, and source code files, which can lead to a compromise of the system or application. Additionally, this vulnerability can be used to include arbitrary files, such as web shells or malicious code, into the application, leading to remote code execution.

5.1 White-Box

The validators argument is used to apply certain validation rules to the uploaded files. In this case, the allowed_mimetypes and allowed_extensions arguments are empty strings, which means that any file type can be uploaded.

```
# backend\apps\users\models.py

document = models.FileField(upload_to=document_directory_path,
    validators=[FileValidator(
    allowed_mimetypes='', allowed_extensions='', max_size=1024*1024*5)], blank=False,
        null=False)
```

5.2 Black-Box

An attacker could upload a malicious file, such as a PHP file, and use it to execute arbitrary code on the server. To prevent this, the code should validate that only allowed file types are uploaded, and reject any other file types. This could be used to view, modify, or delete sensitive files.

6 WSTG-ATHZ-02

The vulnerability identified in the web application falls under the WSTG-ATHZ-02 category, which relates to Authorization Testing. Authorization is the process of determining whether a user or entity has the necessary permissions to access a particular resource or perform a specific action within a system [5].

6.1 White-Box

Unauthenticated users can access and approve/decline certifications.

```
# backend/apps/certifications/views.py - line 37

class AnswerCertificationRequest(generics.GenericAPIView):
    """View to answer certification requests"""
    permission_classes = [permissions.IsAuthenticated]

    def post(self, request):

        # check if id and status is provided
        if not(request.data.get('id') and request.data.get('status')):
            return Response({'error': 'id and status is required'},
                status=status.HTTP_400_BAD_REQUEST)

        certification_request = CertificationRequest.objects.get(
            id=request.data['id'])

        if certification_request is None: # check if certification request exists
            return Response({'error': 'Certification Request does not exist'},
                status=status.HTTP_400_BAD_REQUEST)

        if certification_request.status != 'P': # check if certification request is
             pending
            return Response({'error': 'Certification Request is not pending'},
                status=status.HTTP_400_BAD_REQUEST)

        state = request.data['status']

        if state == 'A': # if accepted
            certification_request.status = Status.ACCEPTED
        elif state == 'D': # if declined
            certification_request.status = Status.DECLINED
        else:
            return Response({'error': 'Status must be A or D'},
                status=status.HTTP_400_BAD_REQUEST)
        certification_request.save()
        return Response(status=status.HTTP_200_OK)
```

6.2 Black-Box

It can be exploited by interacting directly with the API, the payload needs the certifications' id (incremental) and "A" for accepting it, "D" for declining it.

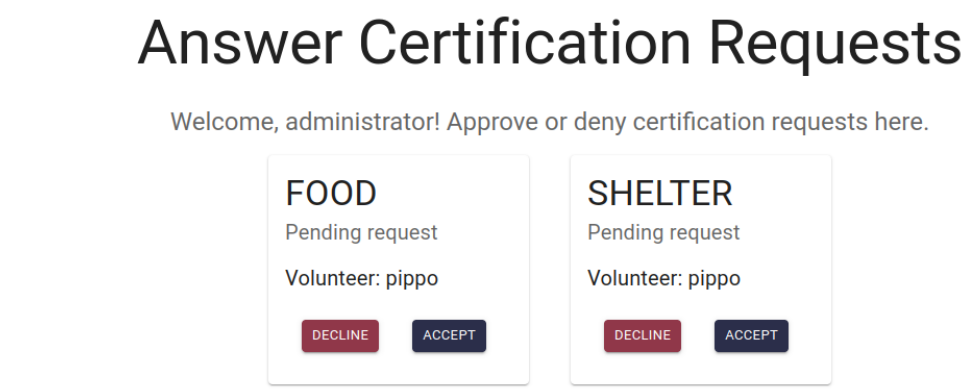

Figure 3: Two Cerfications need to be approved/declined.

```
curl 'http://localhost:21097/api/certifications/answer/' -X POST -H 'User-Agent:
    Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0' -H
    'Accept: application/json, text/plain, */*' -H 'Accept-Language:
    it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3' -H 'Accept-Encoding: gzip, deflate, br' -H
    'Content-Type: application/json' -H 'Authorization: Bearer
    eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ0b2tlbl90eXBlIjoiYWNj
ZXNzIiwiZXhwIjoxNjgxMDQwNTIxLCJpYXQiOjE2NzcONDA1M
jEsImp0aSI6ImFhNTRhMjBlYTIxMjQ1MWU4NTQyOGNkNmJlYzdmNTQ5IiwidXNlc
    l9pZCI6Mn0.fMXYFoUtIXEGEHJrEH3XcX0lJ7Q1-JMzo5B6zaaBqgE' -H 'Origin:
    http://localhost:21097' -H 'Connection: keep-alive' -H 'Referer:
    http://localhost:21097/approve-certifications' -H 'Sec-Fetch-Dest: empty' -H
    'Sec-Fetch-Mode: cors' -H 'Sec-Fetch-Site: same-origin' --data-raw
    '{"id":7,"status":"A"}'
```

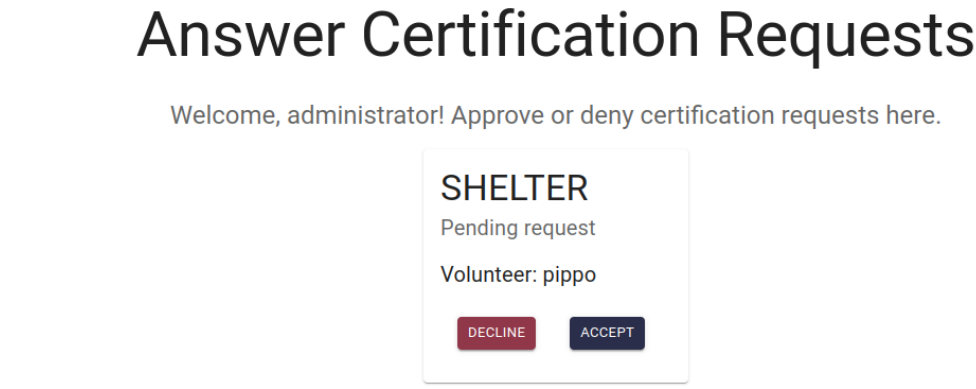

Figure 4: Only one certification left to be approved.

7 WSTG-ATHZ-02

This category focuses on testing the authorization mechanism of the web application, which is responsible for verifying if a user or entity has the required privileges to access a particular resource or execute a specific action within the system [5].

7.1 White-Box

The issue is that the password of a user can be changed by interacting directly with the API, since the reset password method is based on the base64 encoded user id (incremental), and the token.

```
#backend/apps/users/views.py - line 150

class ResetPasswordView(generics.GenericAPIView):
    """View for password reset redirect"""

    def get(self, request, uidb64, token):

        new_password_url = settings.URL + "/new_password"
        invalid_url = settings.URL + "/invalid"
        try:
            id = force_str(urlsafe_base64_decode(uidb64))
            user = get_user_model().objects.get(pk=id)

            if not PasswordResetTokenGenerator().check_token(user, token): # Verify
                that the token is valid for the user
                return redirect(invalid_url)

            return redirect(f'{new_password_url}?uid={uidb64}&token={token}')

        except Exception as ex:
            pass

        return redirect(invalid_url)
```

7.2 Black-Box

Since the user id is incremental, it can be easily found, moreover, usually the id number 1 would be the admin's id, therefore the admin's account can be exploited:

1. Encode in base 64 the id (1) = MQ
2. Interact directly with the API

```
curl -X POST -F "uid=MQ" -F "password=fancynewpsw" -F "password1="fancynewpsw" -F
    "token=1" 'http://localhost:21097/api/reset-password-validate/'
```

As a result, the admin's password (id 1) is changed without authorization.

8 WSTG-SESS-01

This vulnerability arises due to bad management in session tokens [6].

8.1 White-Box

Access token lifetime and refresh token lifetime are set to extremely high values.

```
# backend/secure_help/settings.py

SIMPLE_JWT = {
'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60000),
'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
'ROTATE_REFRESH_TOKENS': True,
}
```

8.2 Black-Box

Access tokens are used to authorize access to protected resources, and if they are leaked, an attacker could potentially use a stolen token for a longer period of time, also brute-force methods to try and guess a valid token can be used because of the longer time to live.

9 WSTG-SESS-07

In Secure Help, there is no hard session timeout to log out a user after being idle for a specified timeframe [7].

9.1 White-Box

The application does not automatically log out a user when that user has been idle for a certain amount of time, ensuring that it is not possible to “reuse” the same session and that no sensitive data remains stored in the browser cache.

9.2 Black-Box

If an application does not automatically log out a user after a certain amount of time of inactivity, it can leave the user’s session vulnerable. An attacker could steal the user’s session ID (stored in a cookie), use it to impersonate the user and potentially escalate their privileges to gain access to sensitive parts of the application that they would not otherwise have access to. This can be done with the assistance of tools such as Burp Suite or Wireshark.

10 WSTG-INPV-01

This section deals with testing for reflected cross-site scripting (XSS) vulnerabilities in web applications. Reflected XSS attacks occur when an attacker injects malicious code into a web application that is then reflected back to the user, often via an input field [8].

10.1 White-Box

Input from the "send help request" form is not sanitized, therefore the application it's vulnerable to XSS attack.

```
# backend/apps/help_requests/models.py

class HelpRequest(models.Model):
"""Model for help requests"""
service_type = models.CharField(
    max_length=20, choices=Competence.choices) # Type of service requested. Use
        Competence enum as choices.

description = models.TextField() # Description of the request

refugee = models.ForeignKey( # User who created the request
    get_user_model(), on_delete=models.CASCADE, related_name='refugee_help_request')
volunteer = models.ForeignKey( # User who accepted the request
    get_user_model(), on_delete=models.DO_NOTHING,
        related_name='volunteer_help_request', null=True, blank=True)

finished = models.BooleanField(default=False) # If the request is finished

def __str__(self):
    return self.title
```

10.2 Black-Box

This vulnerability can be exploited by injecting malicious code in the request textbox. In the following figures, you can see the process of injecting and executing the code in the App.

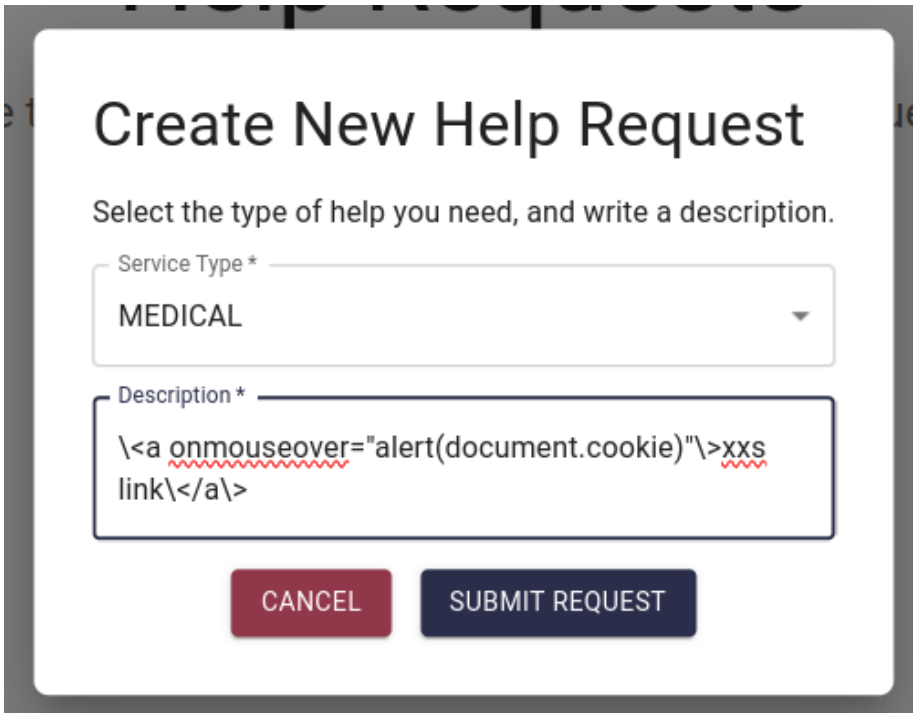

Figure 5: Malicious input on the Request description.

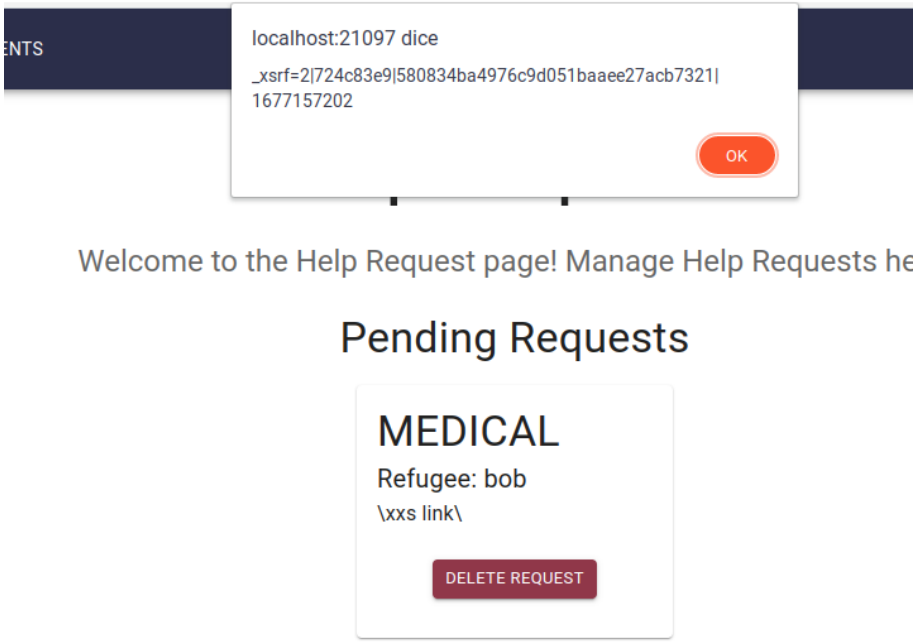

Figure 6: Execution of the malicious code.

Every user visualizing the "injected" request will execute the malicious code.

11 WSTG-INPV-05

When handling users' requests to accept a help request there is no proper input validation for Secure Help [9].

11.1 White-Box

The following line of code constructs a raw SQL query using string concatenation, which can be manipulated by an attacker to execute arbitrary SQL commands.

```
# \backend\apps\help_requests\views.py - line 70

class AcceptHelpRequest(generics.GenericAPIView):
    # other lines of code
    help_request = HelpRequest.objects.raw(
                "SELECT * FROM help_requests_helprequest WHERE id = %s", [rId])[0]
```

11.2 Black-Box

The code is vulnerable to SQL injections. An attacker could construct a request_id parameter like the following to execute a SQL injection attack.

```
import base64
import pickle

# Injected SQL code
malicious_code = "1; DROP ALL TABLES; --"

# Base64-encode and pickle the malicious code
payload = base64.b64encode(pickle.dumps(malicious_code)).decode()

# Send the HTTP POST request with the malicious payload
response = requests.post("http://example.com/api/help_requests/accept/",
    data={"request_id": payload})
```

In this case, the injected SQL code would cause all the tables to be dropped from the database, potentially causing data loss and other problems. We would have to verify the type of the DataBase [10] which we did not do because we were not confident enough with an SQL injection, we were unsure of what is allowed and what is not. This means that the previous code snippet is not tested and we just assume it would be something close to this.

11.3 Notes

The same problem is in the same file in a different line.

```
#  \backend\apps\help_requests\views.py - line 102

class FinishHelpRequest(generics.GenericAPIView):
    # other lines of code
    help_requests = HelpRequest.objects.raw(
                "SELECT * FROM help_requests_helprequest WHERE id = '%s'" % rId)
```

12 WSTG-ERRH-01

Errors are an inevitable part of any application's lifecycle, and they can arise from a variety of sources. These errors can take the form of stack traces, network timeouts, input mismatches, or memory dumps, among others. This section of the WSTG covers error handling and reporting testing. It emphasizes the importance of proper error handling, which is critical for ensuring the stability and security of web applications [11].

12.1 White-Box

The application does not log errors. Settings for logging errors should be in the settings.py file but as we can see the code snippet for this use is missing. We are not aware of other ways of logging errors but we could not find anything.

12.2 Black-Box

This could lead to many exploits such as difficulty in identifying and responding to security incidents. If important security events are not being logged, it may be difficult for system administrators and security teams to identify and respond to security incidents in a timely manner. This could result in a long time to detect and remediate attacks, which could allow attackers to cause more damage or exfiltrate sensitive data. Another problem arises in tracking the user activity, because Without proper logging, it may be difficult to track user activity and identify suspicious or malicious behaviour. Attackers may be able to move laterally within a network undetected, escalate privileges, or carry out other attacks without being detected. Another difficulty arises in forensic analysis because it will make it harder to determine the root cause of an incident, identify affected systems or data, and develop effective remediation plans. Lastly, insider threats are a problem. Insiders with access to sensitive data or systems may be able to carry out attacks without being detected if their actions are not logged properly. This could include data theft, sabotage, or other malicious activity.

13 WSTG-CRYP-04

The objective of WSTG-CRYP-04 is to evaluate the strength of the cryptographic implementation used by a web application. The problem with the following vulnerability is that it uses an algorithm known to be avoided [12].

13.1 White-Box

The process of generating an email verification link involves base64 encoding of the user id, without encryption. It's important to note that Base64 is not a cryptographic algorithm and does not provide encryption or data integrity. Base64 encoding simply converts binary data into ASCII text format that can be transmitted over channels that do not support binary data. Anyone who intercepts the Base64-encoded data can easily decode it back into its original binary format. This means that individuals with fake email addresses can potentially be created because the verification link can be circumvented by someone who knows the user id. The fact that the user id is included in the email verification link means that someone who knows the user id can potentially bypass the verification process by constructing the verification link themselves, without needing access to the user's email address. This is a security concern for the application because it relies on email verification as a way of confirming a user's identity since fake accounts could be created without needing a valid email address.

```
# backend/apps/users/serializers.py - line 70.

uid = urlsafe_base64_encode(user.username.encode())
domain = get_current_site(self.context["request"])
link = reverse('verify-email', kwargs={"uid": uid})

url = f"{settings.PROTOCOL}://{domain}{link}"
```

13.2 Black-Box

Intercepting and decoding the Base64-encoded data during transmission, we can easily decode it back into its original binary format. This reveals sensitive information that was being transmitted, in our case the user name of the verification link. The tool we used for this is a simple online decoder/encoder [13].

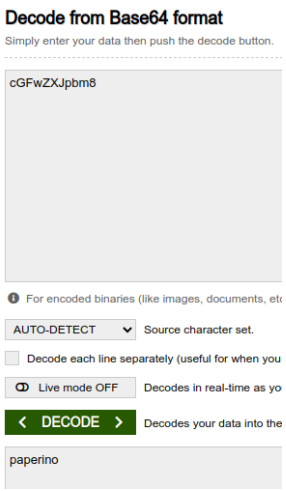

Figure 7: Verification link tail for the username "paperino"

14 WSTG-CRYP-04

Again, the application uses another algorithm that should also be avoided due to security issues.

14.1 White-Box

The UnsaltedSHA1PasswordHasher uses the SHA-1 hash algorithm to hash user passwords, without adding salt. This means that the password hashes are not unique, and attackers can easily use precomputed hash tables, such as rainbow tables, to crack passwords and gain unauthorized access to user accounts. Additionally, the SHA-1 hash algorithm is considered insecure and has been deprecated in many applications due to known security vulnerabilities.

```
# backend/secure_help/settings.py

PASSWORD_HASHERS = [
'django.contrib.auth.hashers.UnsaltedSHA1PasswordHasher',
]
```

14.2 Black-Box

An attacker could try to obtain access to the application's database or obtain the hash values of the user's passwords by intercepting network traffic. Once the attacker has the hash values, he can generate a list of commonly used passwords or dictionary words. For each password in the list, compute the SHA-1 hash value and compare it to the hash values obtained from the database. If a match is found, the attacker has successfully cracked the password and can use it to gain unauthorized access to the user's account [14]. There are also already pre-calculated hashes of different passwords on the web that an attacker can use in order to make the process faster [15].

15 A02:2021

As we can see all the data in secure help are transmitted over text with the HTTP protocol. Passwords, which are sensitive information [16] require specific handling.

15.1 White-Box

Data is transmitted over HTTP and not HTTPS.

```
 [stefano@fedora ~]$ curl -v http://localhost:21097/login
*   Trying 127.0.0.1:21097...
* Connected to localhost (127.0.0.1) port 21097 (#0)
> GET /login HTTP/1.1
> Host: localhost:21097
> User-Agent: curl/7.85.0
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.23.3
< Date: Mon, 27 Feb 2023 01:02:34 GMT
< Content-Type: text/html; charset=utf-8
< Content-Length: 1770
< Connection: keep-alive
< X-Powered-By: Express
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: *
< Access-Control-Allow-Headers: *
< Accept-Ranges: bytes
< ETag: W/"6ea-C45jOO6DgDruPi5eaGHsWGYSWpo"
< Vary: Accept-Encoding
< X-Content-Type-Options: nosnif
```

15.2 Black-Box

Without HTTPS encryption, data is sent in plain text, which means that attackers can intercept the communication between two parties and act as a proxy. This allows them to read, modify, or even inject new data into the communication, giving them access to sensitive information.

As we can see the help of the tool Wireshark [17] we can easily sniff a user's password.

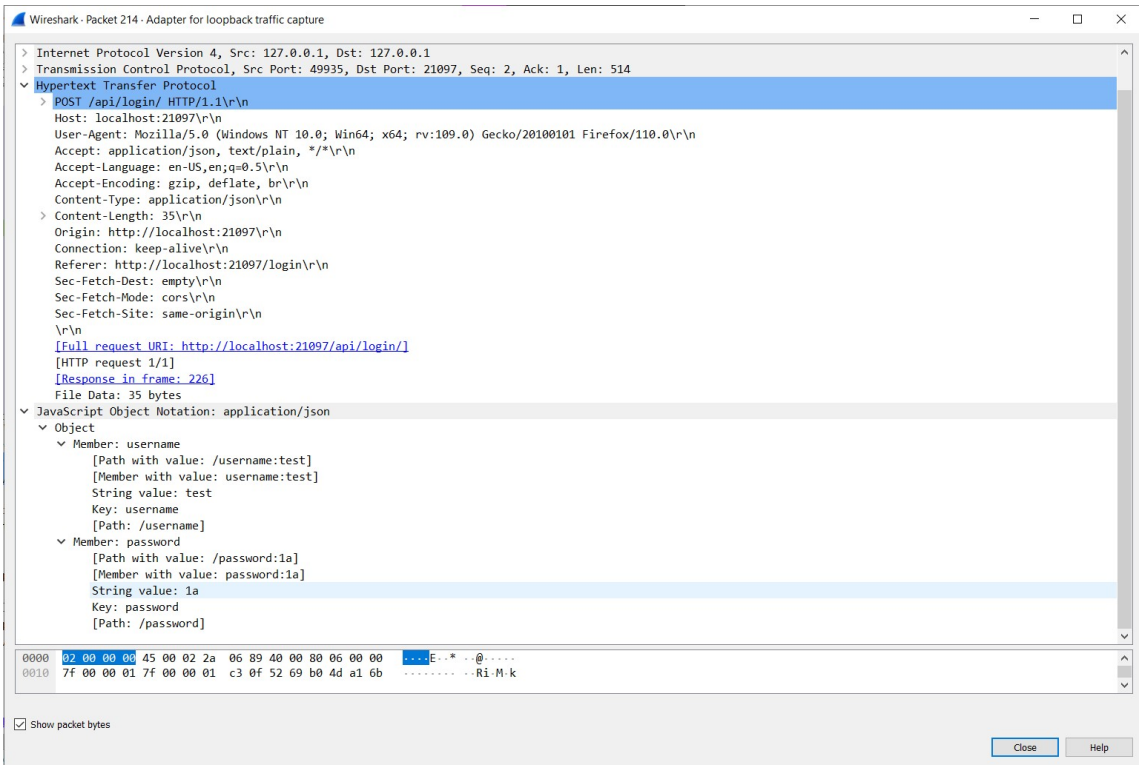

Figure 8: Package captured from Wireshark with a user's password

16 A06:2021

This vulnerability refers to the use of known vulnerable components such as an older version of software, libraries etc. [18]

16.1 White-Box

We noticed that the DJANGO version that is used in the application is an older version with a known vulnerability of DoS attack [19].

```
# requirements.txt

# code
Django==4.0.8
# code
```

16.2 Black-Box

Obviously, this means that somebody can exploit the Django DoS attack vulnerability.

17 Conclusion

In conclusion, our investigation of Secure Help has identified several vulnerabilities that could potentially compromise the security and privacy of its users. Many of the vulnerabilities we identified are a result of bad practices for input handling. We were told through the course that we should never trust users and always sanitise user input, thus a vulnerability related to this practice was easy to find. These types of vulnerabilities are relatively easy to solve.

We are proud of our work since we had to cooperate for a demanding assignment with people we did not know, communicate in a language that for none of us is his primary, on an unknown field for us, but still managed to get this far, identifying fifteen out of thirty vulnerabilities. In the following chart a summary of our work is shown. Thirteen of the fifteen vulnerabilities were complied with WSTG and two were complied with OWASP.

WSTG CATEGORY	# vulnerabilities	# discovered vulnerabilties	percentage
WSTG-IDNT	3	0	0%
WSTG-ATHN	6	3	50%
WSTG-ATHZ	2	3	100%
WSTG-SESS	4	2	50%
WSTG-INPV	3	2	66%
WSTG-ERRH	2	1	50%
WSTG-CRYP	2	2	100%
WSTG-CLNT	4	0	0%
WSTG-BUSL	2	0	0%
WSTG-CONF	2	0	0%
Total	30	13	

Figure 9: Progress of our work

References

[1] *Authentication Testing - Testing for Weak Password Policy.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/04-Authentication_Testing/07-Testing_for_Weak_Password_Policy.

[2] *Burp Suite.* URL: https://portswigger.net/burp/communitydownload.

[3] *FoxyProxy.* URL: https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/?utm_source=addons.mozilla.org&utm_medium=referral&utm_content=search.

[4] *Authorization Testing - Testing Directory Traversal File Include.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/05-Authorization_Testing/01-Testing_Directory_Traversal_File_Include.

[5] *Authorization Testing - Testing for Bypassing Authorization Schema.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/05-Authorization_Testing/02-Testing_for_Bypassing_Authorization_Schema.

[6] *Session Management Testing - Testing for Session Management Schema.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/06-Session_Management_Testing/01-Testing_for_Session_Management_Schema.

[7] *Session Management Testing - Testing Session Timeout.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/06-Session_Management_Testing/07-Testing_Session_Timeout.

[8] *Input Validation Testing - Testing for Reflected Cross Site Scripting.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/01-Testing_for_Reflected_Cross_Site_Scripting.

[9] *Input Validation Testing - Testing for SQL Injection.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection.

[10] Kamal B. *Dumping a complete database using SQL injection [updated 2021].* 2021. URL: https://resources.infosecinstitute.com/topic/dumping-a-database-using-sql-injection/.

[11] *Testing for Error Handling - Testing for Improper Error Handling.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/01-Testing_For_Improper_Error_Handling.

[12] *Testing for Weak Cryptography - Testing for Weak Encryption.* URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/04-Testing_for_Weak_Encryption.

[13] *Base64 Online Encoding/Decoding Tool.* URL: https://www.base64decode.org/.

[14] David Bryan. *How Not to Store Passwords: SHA-1 Fails Again.* 2017. URL: https://securityintelligence.com/how-not-to-store-passwords-sha-1-fails-again/.

[15] *Really Bad Passwords (with Unsalted Hashes).* 2012. URL: https://rietta.com/blog/really-bad-passwords-with-unsalted-hashes/.

[16] *Cryptographic Failures - A02:2021.* URL: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/.

[17] *Wireshark.* URL: https://www.wireshark.org/.

[18] *Vulnerable and Outdated Components - A06:2021.* URL: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/.

[19] *Django 4.0.8 DoS attack Vulnerability.* URL: https://security.snyk.io/package/pip/django/4.0.8.