NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4237 Software Security and Data Privacy

Exercise 3. Finding Vulnerabilities

Group 65

Author(s):
Adria Torija Ruiz
Sander Strand Engen
Sebastian Vildalen Ekpete

Table of Contents

1 Introduction

This report identifies vulnerabilities in the web application "TrustedSitters". Most of the vulnerabilities are identified by performing white-box testing, which shows where in the source code the vulnerabilities exist. Then, for each vulnerability, a demonstration of the exploit is provided through black-box testing. The white-box and black-box testing methods are all described in The Open Web Application Security Project's (OWASP's) Web Security Testing Guide (WSTG 4.2) [1].

2 WSTG-CONF-02 - Vulnerability 1

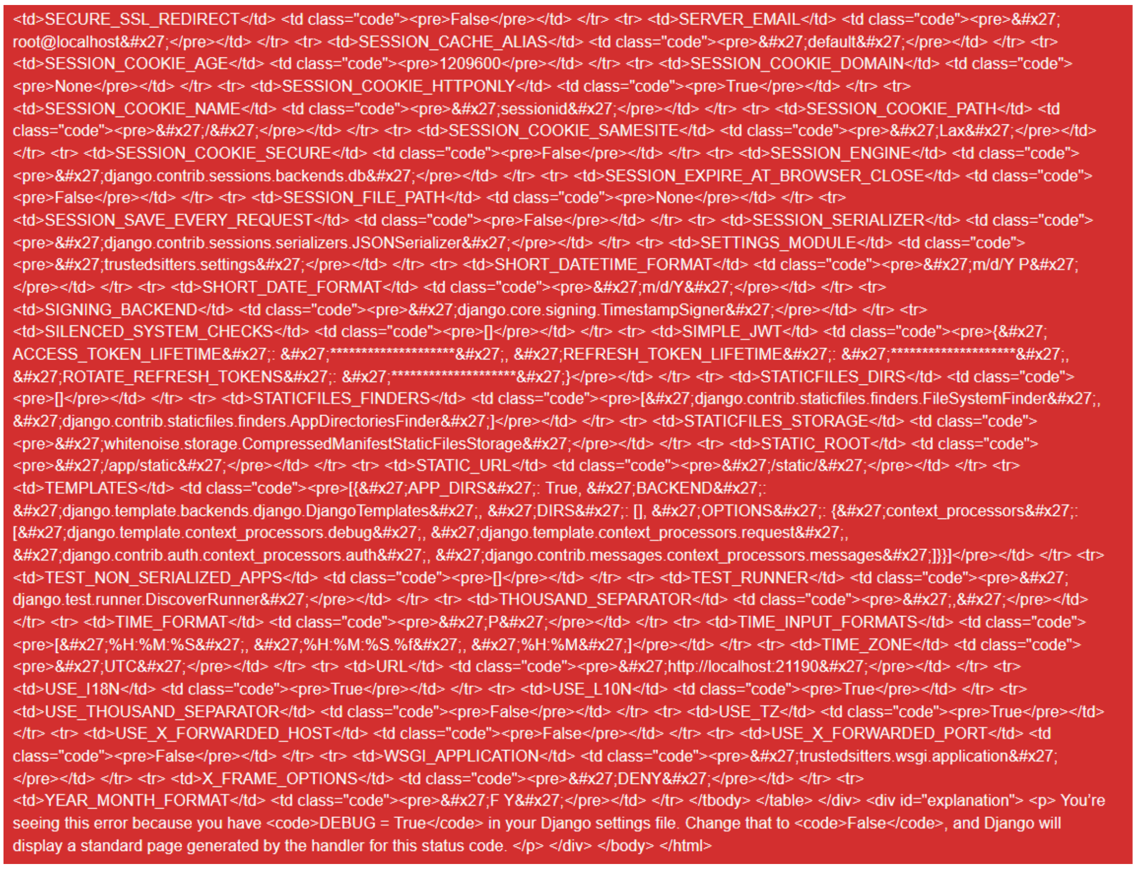
Debug option is turned on.

2.1 White-Box

```
DEBUG = True
```

2.2 Black-Box

If we register with a password no valid, for example just a letter, we find the following debug



```
<td>SECURE_SSL_REDIRECT</td> <td class="code"><pre>False</pre></td> </tr> <tr> <td>SERVER_EMAIL</td> <td class="code"><pre>&#x27;
root@localhost&#x27;</pre></td> </tr> <tr> <td>SESSION_CACHE_ALIAS</td> <td class="code"><pre>&#x27;default&#x27;</pre></td> </tr> <tr>
<td>SESSION_COOKIE_AGE</td> <td class="code"><pre>1209600</pre></td> </tr> <tr> <td>SESSION_COOKIE_DOMAIN</td> <td class="code">
<pre>None</pre></td> </tr> <tr> <td>SESSION_COOKIE_HTTPONLY</td> <td class="code"><pre>True</pre></td> </tr> <tr>
<td>SESSION_COOKIE_NAME</td> <td class="code"><pre>&#x27;sessionid&#x27;</pre></td> </tr> <tr> <td>SESSION_COOKIE_PATH</td> <td
class="code"><pre>&#x27;/&#x27;</pre></td> </tr> <tr> <td>SESSION_COOKIE_SAMESITE</td> <td class="code"><pre>&#x27;Lax&#x27;</pre></td>
</tr> <tr> <td>SESSION_COOKIE_SECURE</td> <td class="code"><pre>False</pre></td> </tr> <tr> <td>SESSION_ENGINE</td> <td class="code">
<pre>&#x27;django.contrib.sessions.backends.db&#x27;</pre></td> </tr> <tr> <td>SESSION_EXPIRE_AT_BROWSER_CLOSE</td> <td class="code">
<pre>False</pre></td> </tr> <tr> <td>SESSION_FILE_PATH</td> <td class="code"><pre>None</pre></td> </tr> <tr>
<td>SESSION_SAVE_EVERY_REQUEST</td> <td class="code"><pre>False</pre></td> </tr> <tr> <td>SESSION_SERIALIZER</td> <td class="code">
<pre>&#x27;django.contrib.sessions.serializers.JSONSerializer&#x27;</pre></td> </tr> <tr> <td>SETTINGS_MODULE</td> <td class="code">
<pre>&#x27;trustedsitters.settings&#x27;</pre></td> </tr> <tr> <td>SHORT_DATETIME_FORMAT</td> <td class="code"><pre>&#x27;m/d/Y P&#x27;
</pre></td> </tr> <tr> <td>SHORT_DATE_FORMAT</td> <td class="code"><pre>&#x27;m/d/Y&#x27;</pre></td> </tr> <tr>
<td>SIGNING_BACKEND</td> <td class="code"><pre>&#x27;django.core.signing.TimestampSigner&#x27;</pre></td> </tr> <tr>
<td>SILENCED_SYSTEM_CHECKS</td> <td class="code"><pre>[]</pre></td> </tr> <tr> <td>SIMPLE_JWT</td> <td class="code"><pre>{&#x27;
ACCESS_TOKEN_LIFETIME&#x27;: &#x27;********************&#x27;, &#x27;REFRESH_TOKEN_LIFETIME&#x27;: &#x27;********************&#x27;,
&#x27;ROTATE_REFRESH_TOKENS&#x27;: &#x27;********************&#x27;}</pre></td> </tr> <tr> <td>STATICFILES_DIRS</td> <td class="code">
<pre>[]</pre></td> </tr> <tr> <td>STATICFILES_FINDERS</td> <td class="code"><pre>[&#x27;django.contrib.staticfiles.finders.FileSystemFinder&#x27;,
&#x27;django.contrib.staticfiles.finders.AppDirectoriesFinder&#x27;]</pre></td> </tr> <tr> <td>STATICFILES_STORAGE</td> <td class="code">
<pre>&#x27;whitenoise.storage.CompressedManifestStaticFilesStorage&#x27;</pre></td> </tr> <tr> <td>STATIC_ROOT</td> <td class="code">
<pre>&#x27;/app/static&#x27;</pre></td> </tr> <tr> <td>STATIC_URL</td> <td class="code"><pre>&#x27;/static/&#x27;</pre></td> </tr> <tr>
<td>TEMPLATES</td> <td class="code"><pre>[{&#x27;APP_DIRS&#x27;: True, &#x27;BACKEND&#x27;:
&#x27;django.template.backends.django.DjangoTemplates&#x27;, &#x27;DIRS&#x27;: [], &#x27;OPTIONS&#x27;: {&#x27;context_processors&#x27;:
[&#x27;django.template.context_processors.debug&#x27;, &#x27;django.template.context_processors.request&#x27;,
&#x27;django.contrib.auth.context_processors.auth&#x27;, &#x27;django.contrib.messages.context_processors.messages&#x27;]}}]</pre></td> </tr> <tr>
<td>TEST_NON_SERIALIZED_APPS</td> <td class="code"><pre>[]</pre></td> </tr> <tr> <td>TEST_RUNNER</td> <td class="code"><pre>&#x27;
django.test.runner.DiscoverRunner&#x27;</pre></td> </tr> <tr> <td>THOUSAND_SEPARATOR</td> <td class="code"><pre>&#x27;,&#x27;</pre></td>
</tr> <tr> <td>TIME_FORMAT</td> <td class="code"><pre>&#x27;P&#x27;</pre></td> </tr> <tr> <td>TIME_INPUT_FORMATS</td> <td class="code">
<pre>[&#x27;%H:%M:%S&#x27;, &#x27;%H:%M:%S.%f&#x27;, &#x27;%H:%M&#x27;]</pre></td> </tr> <tr> <td>TIME_ZONE</td> <td class="code">
<pre>&#x27;UTC&#x27;</pre></td> </tr> <tr> <td>URL</td> <td class="code"><pre>&#x27;http://localhost:21190&#x27;</pre></td> </tr> <tr>
<td>USE_I18N</td> <td class="code"><pre>True</pre></td> </tr> <tr> <td>USE_L10N</td> <td class="code"><pre>True</pre></td> </tr> <tr>
<td>USE_THOUSAND_SEPARATOR</td> <td class="code"><pre>False</pre></td> </tr> <tr> <td>USE_TZ</td> <td class="code"><pre>True</pre></td>
</tr> <tr> <td>USE_X_FORWARDED_HOST</td> <td class="code"><pre>False</pre></td> </tr> <tr> <td>USE_X_FORWARDED_PORT</td> <td
class="code"><pre>False</pre></td> </tr> <tr> <td>WSGI_APPLICATION</td> <td class="code"><pre>&#x27;trustedsitters.wsgi.application&#x27;
</pre></td> </tr> <tr> <td>X_FRAME_OPTIONS</td> <td class="code"><pre>&#x27;DENY&#x27;</pre></td> </tr> <tr>
<td>YEAR_MONTH_FORMAT</td> <td class="code"><pre>&#x27;F Y&#x27;</pre></td> </tr> </tbody> </table> </div> <div id="explanation"> <p> You're
seeing this error because you have <code>DEBUG = True</code> in your Django settings file. Change that to <code>False</code>, and Django will
display a standard page generated by the handler for this status code. </p> </div> </body> </html>
```

Figure 1: Debug

3 WSTG-CONF-02 - Vulnerability 2

The logs from the frontend contain sensitive information.

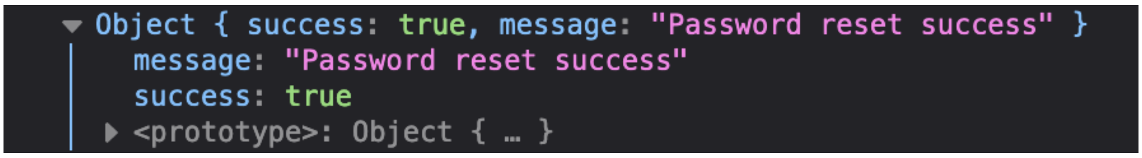
3.1 White-Box

The console log outputs the whole object which may contain sensitive information, instead of just the message.

```
AuthService.newPassword(request)
      .then((response) => {
        console.log(response);
        if (response.success) {
          setSnackbarOpen(true);
        }
      })
```

3.2 Black-Box

By checking the console in the browser, you can see the logs 2.



```
▾ Object { success: true, message: "Password reset success" }
    message: "Password reset success"
    success: true
  ▸ <prototype>: Object { … }
```

Figure 2: Console logs

4 WSTG-CONF-04 - Vulnerability 1

In the .env we can see the django user and password of a superuser

4.1 White-Box

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

DJANGO_SUPERUSER_PASSWORD=password123
DJANGO_SUPERUSER_USERNAME=admin123
DJANGO_SUPERUSER_EMAIL=admin@mail.com

FIELD_ENCRYPTION_KEY=7hfCjr16z4q4q5HKeSK8ZyTBS2o0hKKpU3qSSqp-cEI=
```

4.2 Black-Box

Here we can see how we can log in as admin with the credentials of the file

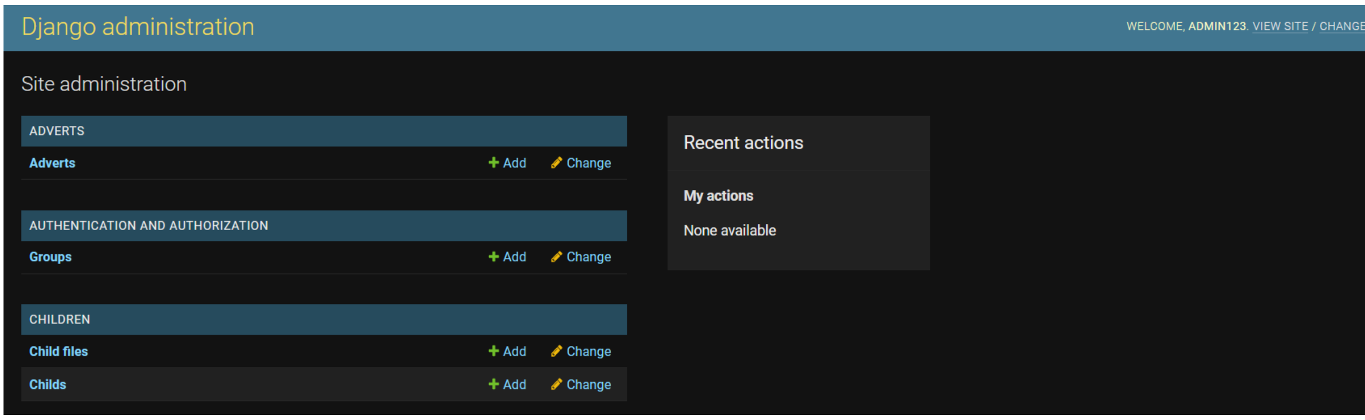


Figure 3: Admin

5 WSTG-CONF-04 - Vulnerability 2

In the file settings.py in the back-end we can see the secret key and the field encryption key.

5.1 White-Box

```
SECRET_KEY = 'aslkjwojdw09wdlg6u=986qz+fh2t!dj-i%)s*vebg@w&r92p2ci(ixc_25cm5!t'
FIELD_ENCRYPTION_KEY = os.environ.get('FIELD_ENCRYPTION_KEY',
    'yXhjluyPu6bYlul9wyAaHW2CT25ky9W7TKC7h8y166E=')
```

5.2 Black-Box

Upon further inspection, we were unable to find anywhere in the code where these keys are used to initialize randomness. As the password reset view is modified to be vulnerable and not base their link on the SECRET_KEY for randomness, and the MFA uses pyotp for it's random key generation, we couldn't find a way to properly exploit this key.

6 WST-CONF-04 - Vulnerability 3

The version of Django is old as the actual one its 4.0.3.

6.1 White-Box

```
Django==3.2.4
```

6.2 Black-Box

In the version 3.2.4 there was the vulnerability: CVE-2021-33203: Potential directory traversal via admindocs That vulnerability was about the Staff membersthat could use the admindocs TemplateDetailView view to check the existence of arbitrary files

7 WSTG-IDNT-02

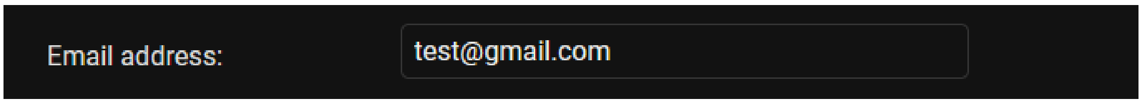
It is possible to repeat username if you are creating the admin user.

7.1 White-Box

```
#!/bin/sh

# Code to run after building the image

python manage.py migrate

python manage.py collectstatic --noinput

DJANGO_SUPERUSER_USERNAME=${DJANGO_SUPERUSER_USERNAME} \
DJANGO_SUPERUSER_PASSWORD=${DJANGO_SUPERUSER_PASSWORD} \
DJANGO_SUPERUSER_EMAIL=${DJANGO_SUPERUSER_EMAIL} \
python manage.py createsuperuser --noinput || \
echo "WARNING: This error is ignored as it most likely is 'That username is already
    taken.'" \
&& echo "If you wish to alter the user credentials, then delete the user first."
python manage.py loaddata initial.json # Add some initial data

gunicorn trustedsitters.wsgi --log-file - -b 0.0.0.0:8000
```

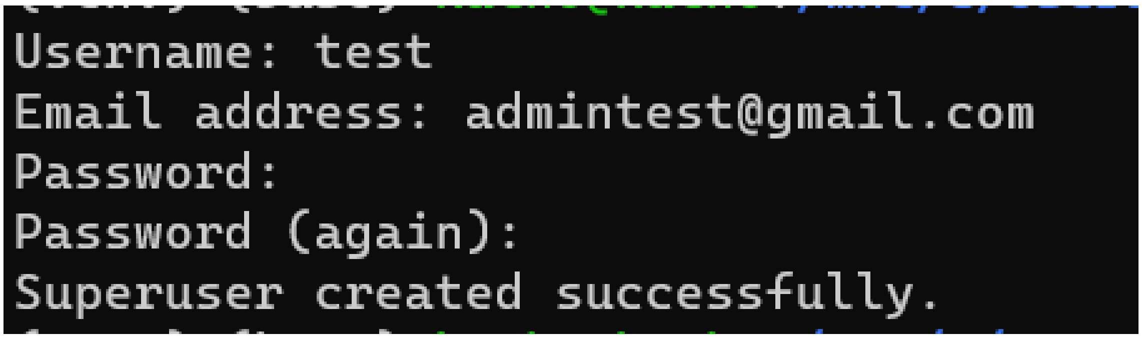
7.2 Black-Box

First of all we created an user that had: username:test password:test email: test Then we created a superuser with the same username and we could see that the password an email from the normal user were overwritten so that he couldn't sign in again. And we could sign in as admin with the new password



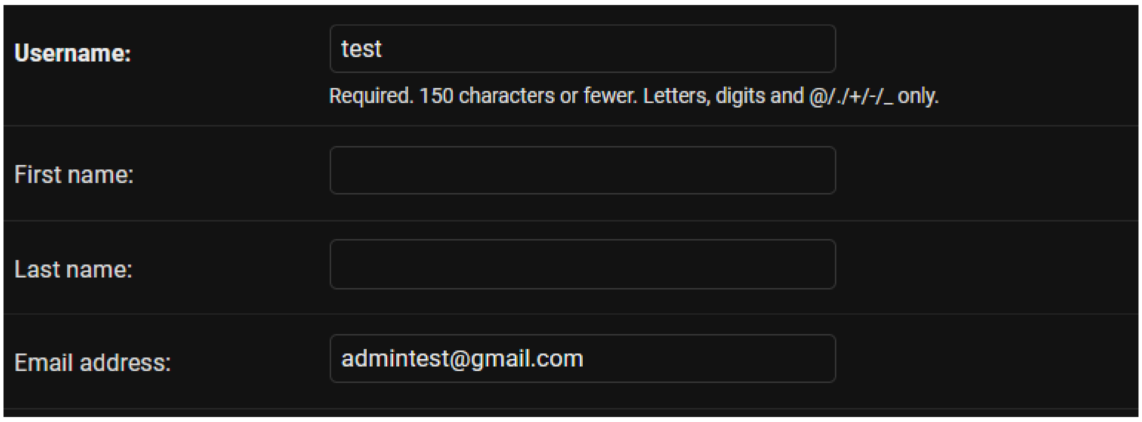
Email address: test@gmail.com

Figure 4: Before



```
Username: test
Email address: admintest@gmail.com
Password:
Password (again):
Superuser created successfully.
```

Figure 5: Changing



Username:	test Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
First name:	
Last name:	
Email address:	admintest@gmail.com

Figure 6: After

8 WSTG-ATHN-01

Credentials are transported over an unencrypted channel.

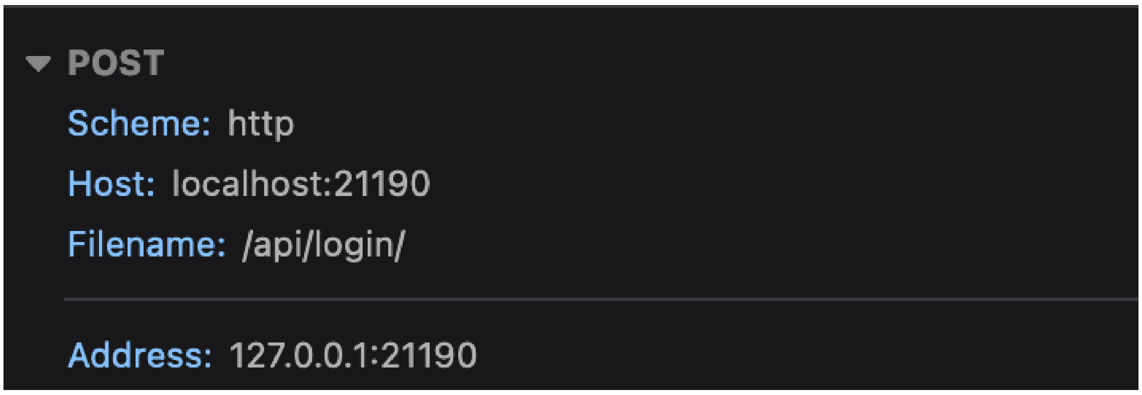
8.1 White-Box

The protocol specified in the settings file is "http" and not "https". One way to solve this is also to set SECURE_SSL_REDIRECT to True such that http will be redirected to https.

```
GROUP_ID = os.environ.get("GROUP_ID", "3000")
PORT_PREFIX = os.environ.get("PORT_PREFIX", "")
DOMAIN = os.environ.get("DOMAIN", "localhost")
PROTOCOL = os.environ.get("PROTOCOL", "http")
```

8.2 Black-Box

The test was done by creating an account and logging in. Then, the post request was inspected, and it showed "http" instead of "https" (figure 7). Thus, the test failed. This can be exploited by an attacker using a tool such as Wireshark [2] and can allow them to view credentials and possibly use them to steal a user's account.



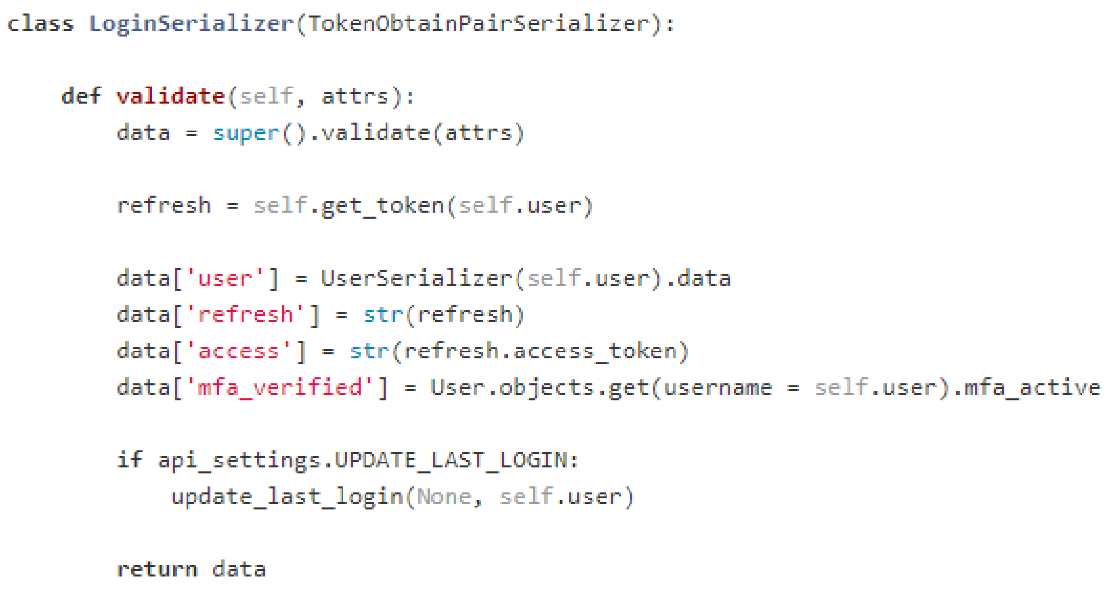
```
▼ POST
  Scheme: http
  Host: localhost:21190
  Filename: /api/login/

  Address: 127.0.0.1:21190
```

Figure 7: Inspect element shows http instead of https

9 WSTG-ATHN-03

There is no mechanism in place to prevent brute-forcing of login credentials.

9.1 White-Box

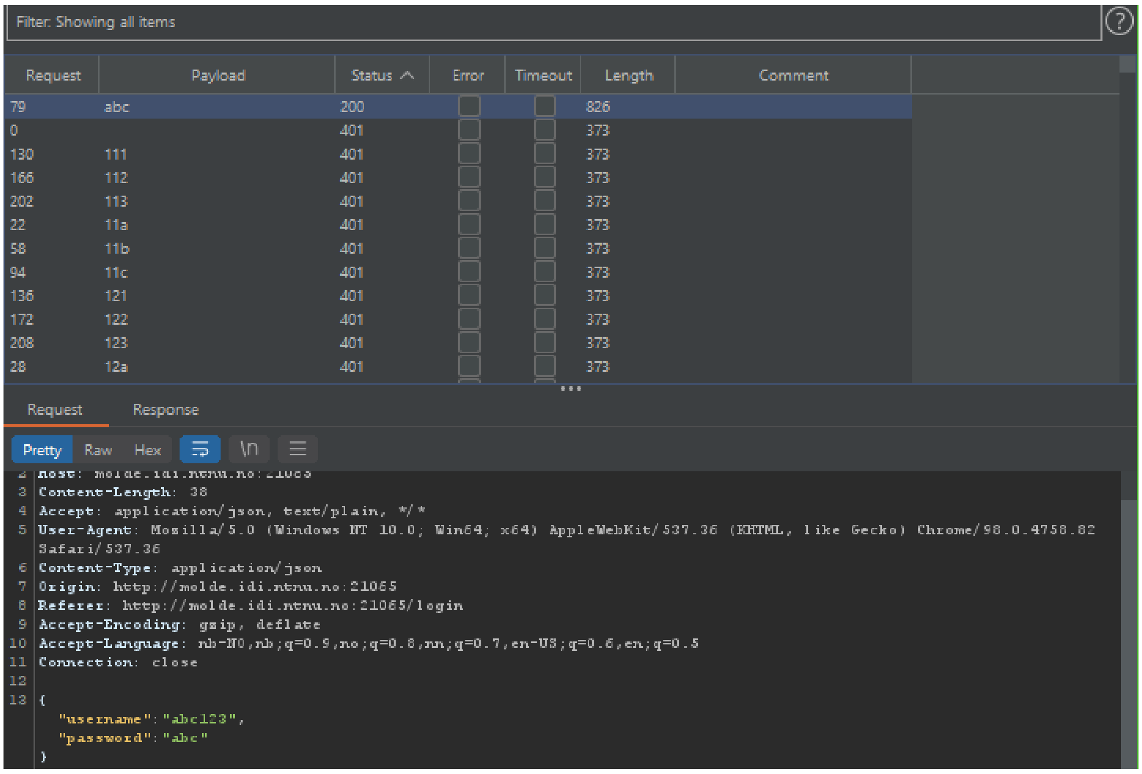


```
class LoginSerializer(TokenObtainPairSerializer):

    def validate(self, attrs):
        data = super().validate(attrs)

        refresh = self.get_token(self.user)

        data['user'] = UserSerializer(self.user).data
        data['refresh'] = str(refresh)
        data['access'] = str(refresh.access_token)
        data['mfa_verified'] = User.objects.get(username = self.user).mfa_active

        if api_settings.UPDATE_LAST_LOGIN:
            update_last_login(None, self.user)

        return data
```

Figure 8: Code showing no lock out mechanism in place

9.2 Black-Box

Creating a account with an easy password and trying to brute-force the password resulted in no lock-outs and a successful result.


Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
79	abc	200	☐	☐	826	
0		401	☐	☐	373	
130	111	401	☐	☐	373	
166	112	401	☐	☐	373	
202	113	401	☐	☐	373	
22	11a	401	☐	☐	373	
58	11b	401	☐	☐	373	
94	11c	401	☐	☐	373	
136	121	401	☐	☐	373	
172	122	401	☐	☐	373	
208	123	401	☐	☐	373	
28	12a	401	☐	☐	373	

```
Host: molde.idi.ntnu.no:21065
Content-Length: 38
Accept: application/json, text/plain, */*
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82
Safari/537.36
Content-Type: application/json
Origin: http://molde.idi.ntnu.no:21065
Referer: http://molde.idi.ntnu.no:21065/login
Accept-Encoding: gzip, deflate
Accept-Language: nb-NO,nb;q=0.9,no;q=0.8,nn;q=0.7,en-US;q=0.6,en;q=0.5
Connection: close

{
  "username":"abc123",
  "password":"abc"
}
```

Figure 9: Successful online brute-force attack on an account.

10 WSTG-ATHN-04

We can access the adverts service without authentification

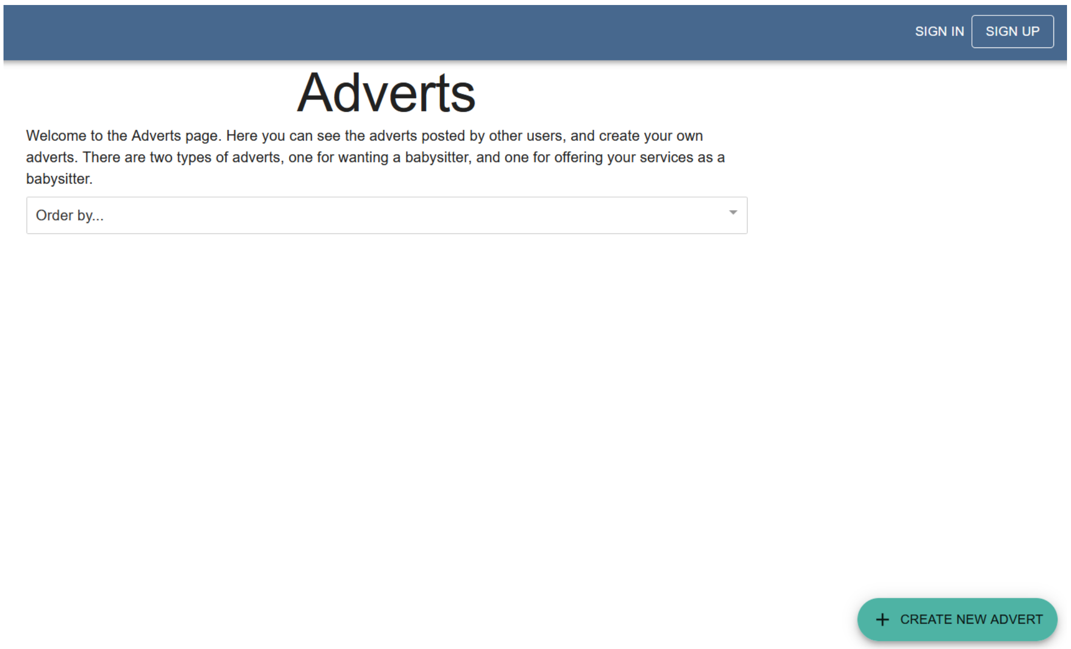
10.1 White-Box

Here we can see how it's automatically redirected to adverts when logging in

```
AuthService.login(request)
      .then((response) => {
        if(response.mfa_verified){
          setVerified(true)
          setEmail(response.user['email'])
          setId(response.user['id'])
        }
        else {
          console.log("Signed in successfully");
          setUsername("");
          setPassword("");
          setUser(response.user);
          history.push("/adverts");
          setAppSnackbarText("Signed in successfully");
          setAppSnackbarOpen(true);
        }
      })
```

10.2 Black-Box

Going to adverts without being logged.



Adverts

Welcome to the Adverts page. Here you can see the adverts posted by other users, and create your own adverts. There are two types of adverts, one for wanting a babysitter, and one for offering your services as a babysitter.

Figure 10: Going to /adverts without login in

11 WSTG-ATHN-07

For validating the password it uses the django password validation. The problem is that it just configured the Numeric Password validator,which checks whether the password isn't entirely numeric. So that is a weak Password Policy

11.1 White-Box

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
def validate(self, data):

        # get the password from the data
        password = data.get('password')

        errors = dict()
        try:
            # validate the password and catch the exception
            validate_password(password=password)

        # the exception raised here is different than serializers.ValidationError
        except exceptions.ValidationError as e:
            errors['password'] = list(e.messages)

        if errors:
            raise serializers.ValidationError(errors)
        return super(RegisterSerializer, self).validate(data)
```

11.2 Black-Box

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?

The only passwords that are forbidden are the numerical ones. Everything else its allowed

How often can a user change their password? How quickly can a user change their password after a previous change?

There is no limit so the users can change it as much as they want and without waiting.

How often can a user reuse a password? Does the application maintain a history of the user's previous used 8 passwords?

As there is no history a user can reuse a password.

What are the minimum and maximum password lengths that can be set, and are they appropriate for the sensitivity of the account and application?

The minimum length it's 1 and the max is 128. I think that 128 is quite excessive but it's fine but the minimum length should be around 6 and 8 probably.

```
class RegisterSerializer(UserSerializer):
    password = serializers.CharField(
        max_length=128, min_length=1, write_only=True, required=True)
```

12 WSTG-INPV-01

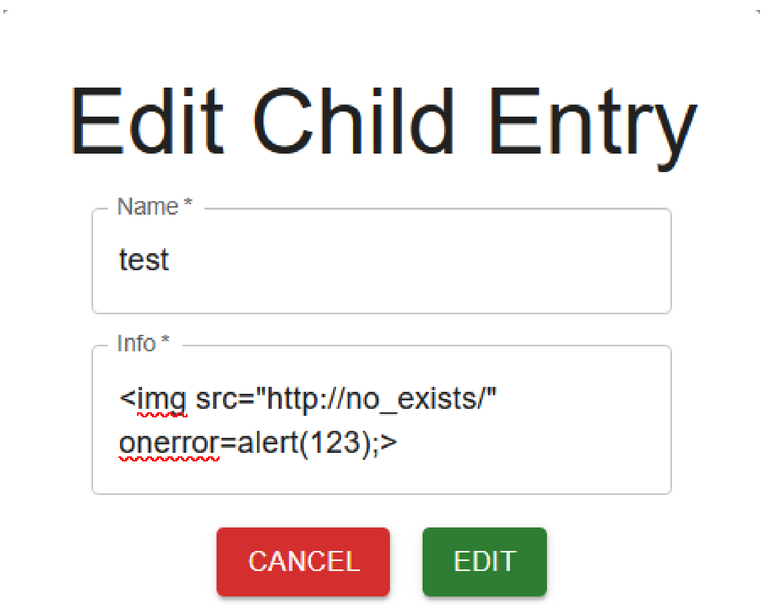
There is no control of the input so it's possible to insert a java-script inside.

12.1 White-Box

```
 <form onSubmit={onSubmit}>
        <Stack spacing={2} padding={2}>
          <TextField
            onInput={(e) => setName(e.target.value)}
            value={name}
            label={"Name"}
            required
          />
```

12.2 Black-Box

We will change a child information. In the Text field we are going to insert our code as in the example bellow:



Edit Child Entry

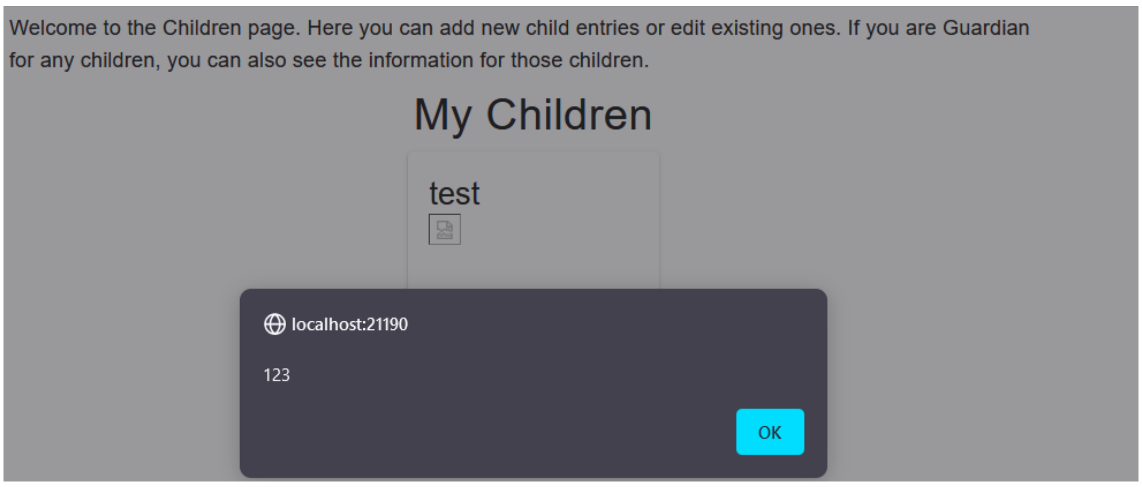
Name *

test

Info *

<img src="http://no_exists/" onerror=alert(123);>

Figure 11: Test



Welcome to the Children page. Here you can add new child entries or edit existing ones. If you are Guardian for any children, you can also see the information for those children.

My Children

test

localhost:21190

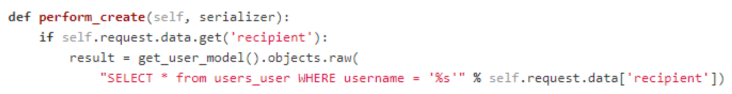
123

Figure 12: Result

13 WSTG-INPV-05

Form parameters when sending a guardian offer at /offers are not cleaned, providing a possibility for a SQL-injection.

13.1 White-Box

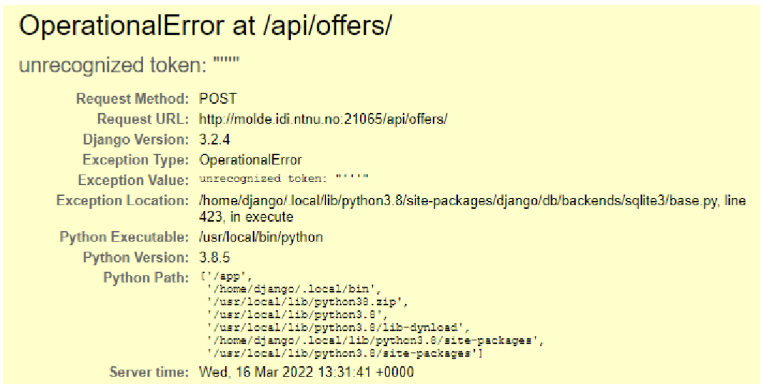


```
def perform_create(self, serializer):
    if self.request.data.get('recipient'):
        result = get_user_model().objects.raw(
            "SELECT * from users_user WHERE username = '%s'" % self.request.data['recipient'])
```

Figure 13: Vulnerable code for SQL injection - Parameters not cleaned

13.2 Black-box

Sending a POST request with the Guardian Offer field, an attacker may edit the payload of the recipient parameter to raise errors or potentially leaking database information and content. Sqlmap was not able to sufficiently find an entry point due to server timeouts and potentially a Website Application Firewall, but the code indicates that the vulnerability is there.



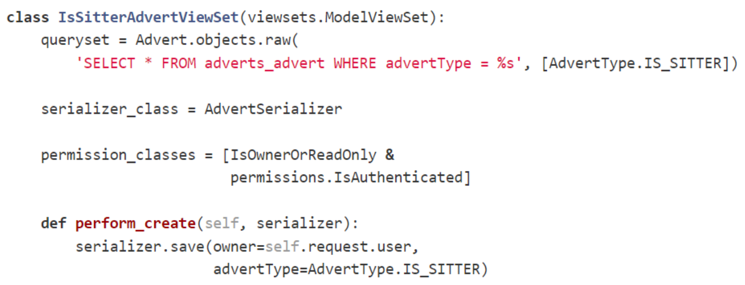
```
OperationalError at /api/offers/
unrecognized token: "'''"

     Request Method:  POST
        Request URL:  http://molde.idi.ntnu.no:21065/api/offers/
     Django Version:  3.2.4
     Exception Type:  OperationalError
    Exception Value:  unrecognized token: "'''"
 Exception Location:  /home/django/.local/lib/python3.8/site-packages/django/db/backends/sqlite3/base.py, line
                      423, in execute
  Python Executable:  /usr/local/bin/python
     Python Version:  3.8.5
        Python Path:  ['/app',
                       '/home/django/.local/bin',
                       '/usr/local/lib/python38.zip',
                       '/usr/local/lib/python3.8',
                       '/usr/local/lib/python3.8/lib-dynload',
                       '/home/django/.local/lib/python3.8/site-packages',
                       '/usr/local/lib/python3.8/site-packages']
        Server time:  Wed, 16 Mar 2022 13:31:41 +0000
```

Figure 14: Error raised when sending POST request with a single quote as payload

14 WSTG-INPV-09

At the page /adverts, the input parameter orderby appears XPath-injectable, as the input is not cleaned.

14.1 White-Box

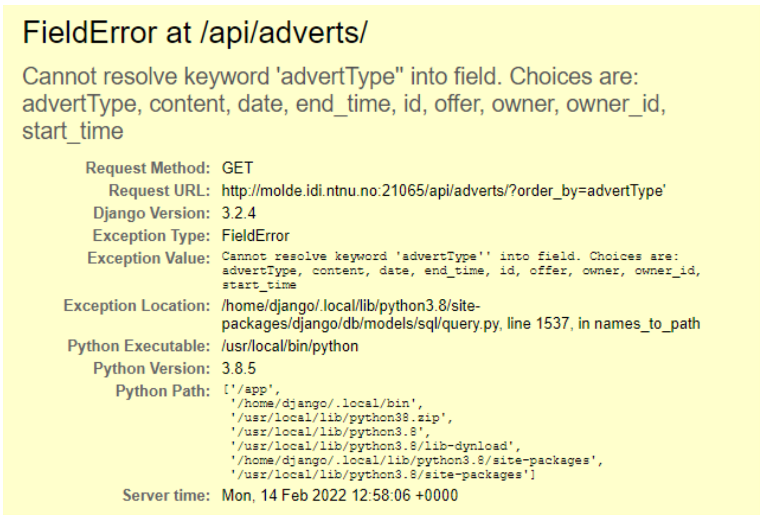


```
class IsSitterAdvertViewSet(viewsets.ModelViewSet):
    queryset = Advert.objects.raw(
        'SELECT * FROM adverts_advert WHERE advertType = %s', [AdvertType.IS_SITTER])

    serializer_class = AdvertSerializer

    permission_classes = [IsOwnerOrReadOnly &
                          permissions.IsAuthenticated]

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user,
                        advertType=AdvertType.IS_SITTER)
```

Figure 15: Vulnerable code for XPath injection - Parameters not cleaned

14.2 Black-Box

Similar to SQL-injection, passing a single quote as parameter value raises an error, indicating that the application may be vulnerable to XPath injections.



```
FieldError at /api/adverts/
Cannot resolve keyword 'advertType'' into field. Choices are:
advertType, content, date, end_time, id, offer, owner, owner_id,
start_time

      Request Method: GET
         Request URL: http://molde.idi.ntnu.no:21065/api/adverts/?order_by=advertType'
      Django Version: 3.2.4
      Exception Type: FieldError
     Exception Value: Cannot resolve keyword 'advertType'' into field. Choices are:
                      advertType, content, date, end_time, id, offer, owner, owner_id,
                      start_time
  Exception Location: /home/django/.local/lib/python3.8/site-
                      packages/django/db/models/sql/query.py, line 1537, in names_to_path
   Python Executable: /usr/local/bin/python
      Python Version: 3.8.5
         Python Path: ['/app',
                       '/home/django/.local/bin',
                       '/usr/local/lib/python38.zip',
                       '/usr/local/lib/python3.8',
                       '/usr/local/lib/python3.8/lib-dynload',
                       '/home/django/.local/lib/python3.8/site-packages',
                       '/usr/local/lib/python3.8/site-packages']
         Server time: Mon, 14 Feb 2022 12:58:06 +0000
```

Figure 16: Triggered error when single quote is passed as parameter

15 WSTG-CRYP-04 - Vulnerability 1

In the script they used base64 to encode and decode. It is in backend apps users in the file views.py

15.1 White-Box

```python
class PasswordResetEmailView(generics.GenericAPIView):
    serializer_class = ResetPasswordSerializer

    def post(self, request):
        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 = urlsafe_base64_encode(force_bytes(user.username))
                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)
```

15.2 Black-Box

We could construct a password reset link for any user by encoding the userid and corresponding username with base64. Assuming the admin123 account had userid 1, we constructed the link `/new_password?uid=MQ==&token=YWRtaW4xMjM=` and was able to reset the password for the admin account and access it.

16 WSTG-CRYP-04 - Vulnerability 2

When the app stores the password in django it is encrypted using UnsaltedMD5 hash, this hash is easy to decrypt so it makes it insecure

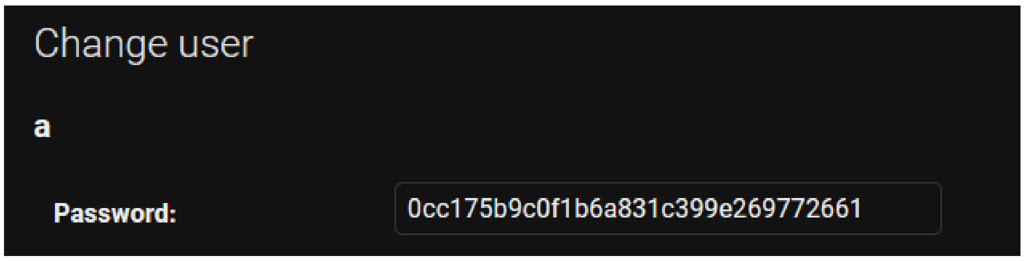
16.1 White-Box

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.UnsaltedMD5PasswordHasher',
]
```

16.2 Black-Box

If we got all the passwords and users from a file it would be encrypted with the MD5 hash so that we could decode it.

We can check the hash as admin. We created an user with the name ”a” and the password ”a” We are going to decrypt the password that we could have stolen in an attack.

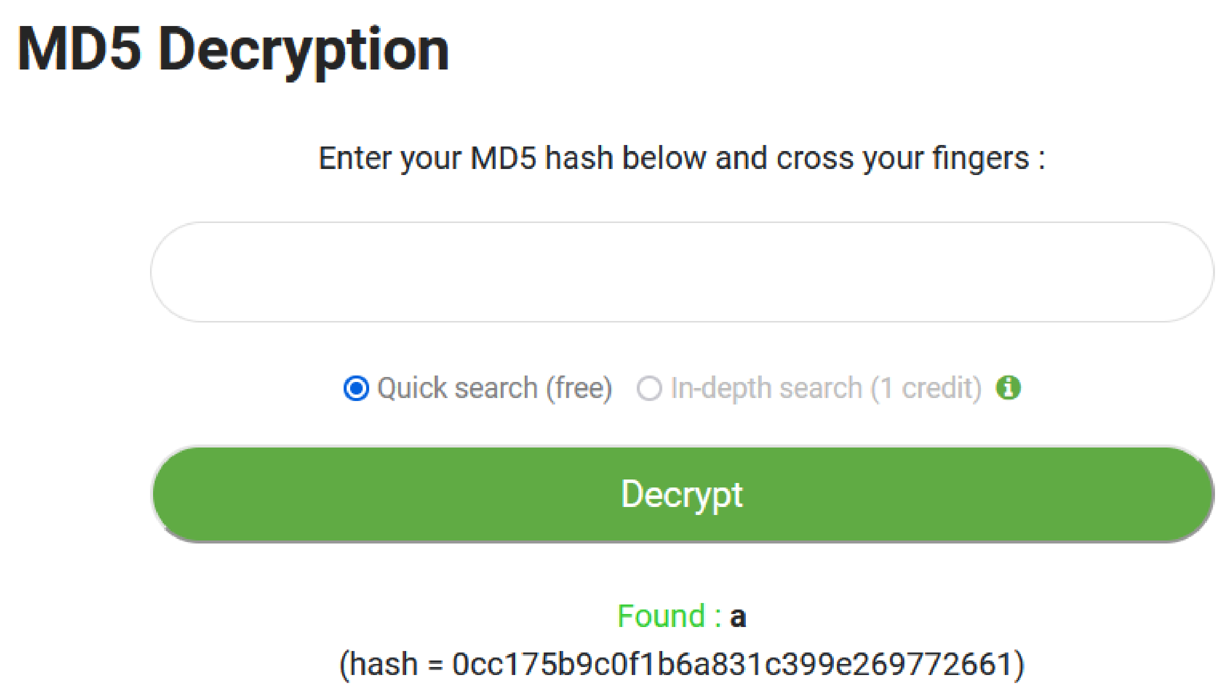


Change user

a

Password: 0cc175b9c0f1b6a831c399e269772661

Figure 17: Hash


MD5 Decryption
Enter your MD5 hash below and cross your fingers :
Quick search (free)
In-depth search (1 credit)
Decrypt

Found : a
(hash = 0cc175b9c0f1b6a831c399e269772661)

Figure 18: Decoding

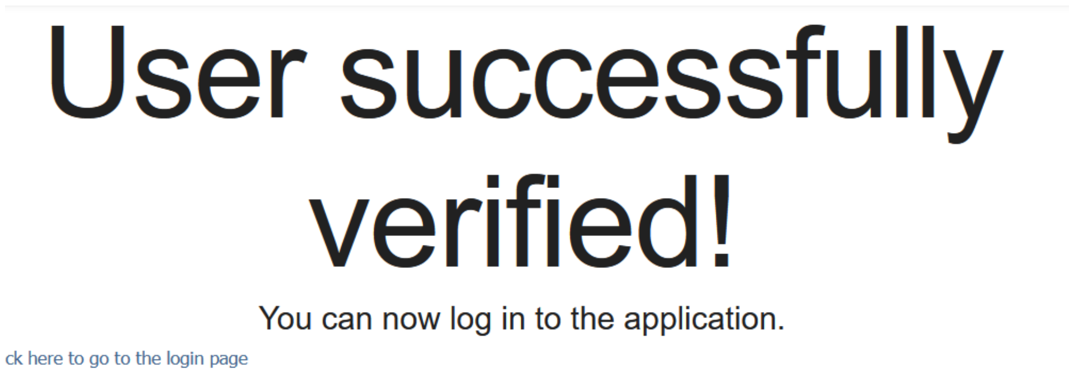
17 WSTG-BUSL-01

We can validate our new account without visiting the email. Number 3 is the number of the user and we can keep trying in order so that at the end we have an error and we get into http://localhost:21190/invalid link localhost:21190/api/verify-email/3/1

17.1 White-Box

```
def create(self, validated_data):
        user = get_user_model().objects.create_user(**validated_data)
        user.is_active = False
        user.save()
        email = validated_data["email"]
        email_subject = "Activate your account"
        uid = user.pk
        domain = get_current_site(self.context["request"])
        link = reverse('verify-email', kwargs={"uid": uid, "status": 1})
```

17.2 Black-Box


User successfully verified!

You can now log in to the application.

ck here to go to the login page

Figure 19: Verified

18 WSTG-BUSL-08

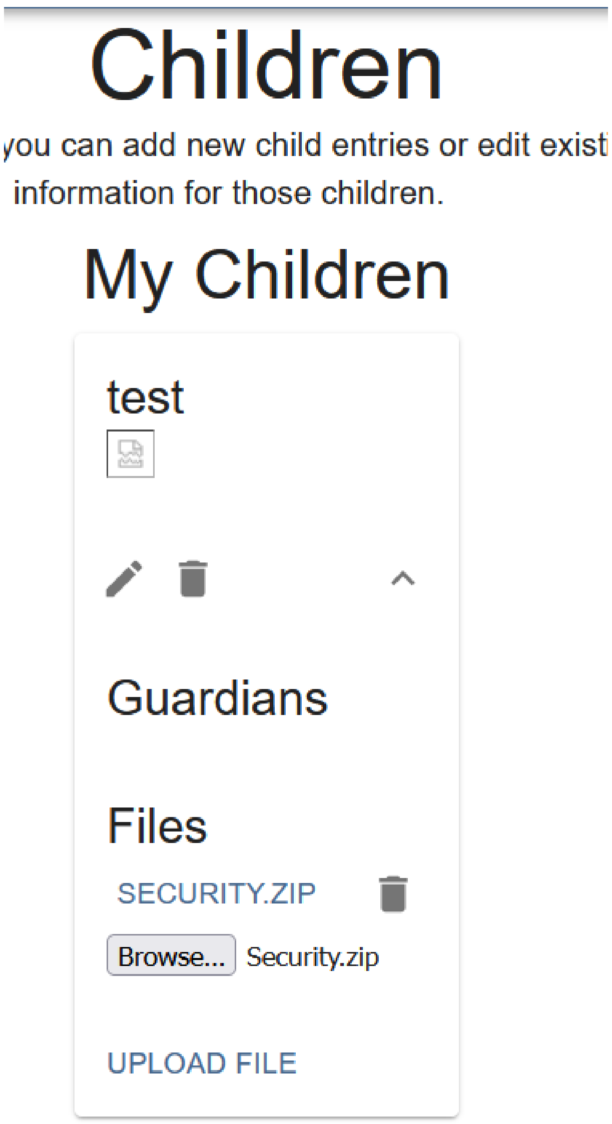
There is no control of the uploading files extension as allowed extensions is not initialized. It is in the backend at children in the file validators.py

18.1 White-Box

```
        # Check the extension
        ext = splitext(value.name)[1][1:].lower()
        if self.allowed_extensions and not ext in self.allowed_extensions:
            code = 'extension_not_allowed'
            message = self.messages[code]
            params = {
                'extension': ext,
                'allowed_extensions': ', '.join(self.allowed_extensions)
            }
            raise ValidationError(message=message, code=code, params=params)

        # Check extension and file format consistency
        if ext == 'jpg':
            ext = 'jpeg'
        if format != ext:
            code = 'format_mismatch'
            message = self.messages[code]
            params = {
                'extension': ext,
                'format': format
            }
            raise ValidationError(message=message, code=code, params=params)
```

18.2 Black-Box

Here we can see that we could insert a possible zip with malicious content.


Children

you can add new child entries or edit exist information for those children.

My Children

test

Guardians

Files

SECURITY.ZIP

Browse... Security.zip

UPLOAD FILE

Figure 20: File

19 WSTG-BUSL-09

When uploading the child file, the content is not checked

19.1 White-Box

```
const uploadFile = (e) => {
    e.preventDefault();

    let formData = new FormData();

    formData.append("file", selectedFile);
    formData.append("child", child.id);
    ChildrenService.UploadChildFile(formData)
      .then((response) => {
        console.log("File uploaded");
        ChildrenService.GetChildFileInfos().then((c) => setChildFiles(c));
      })
      .catch((error) => console.error(error));
  };

const UploadChildFile = (data) => {
  const request = api.post("/child-file/", data, {
    headers: {
      "Content-Type": "multipart/form-data",
    },
  });
  return request.then((response) => response.data);
};
```

19.2 Black-Box

For this example we are going to use a zipBomb that is an archive file that contains a large volume of data. So that as they don't secure the content we can exhaust the disk space if we unzip it.

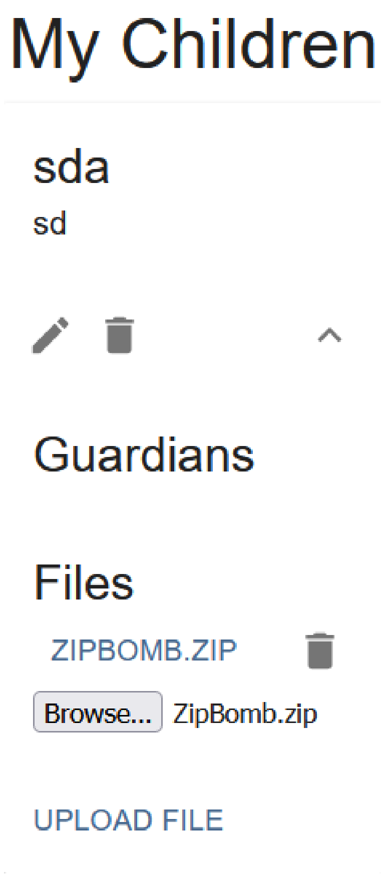


Figure 21: Zip Bomb

20 WSTG-CLNT-03

HTML- and CSS-injection. The frontend code uses "dangerouslySetInnerHTML", where one can easily do injections, for example HTML or CSS.

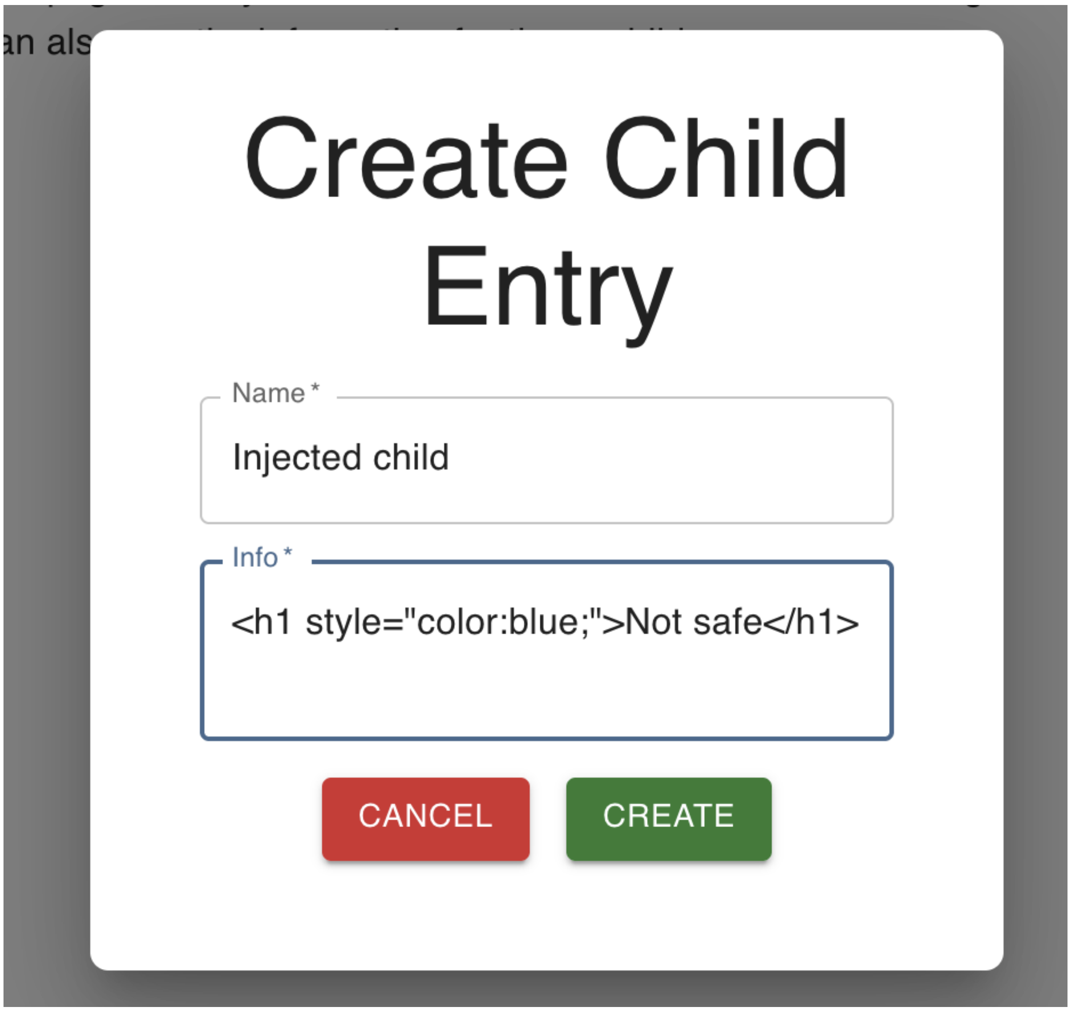
20.1 White-Box

The code provided is from Child.jsx

```
<Card>
        <CardContent>
          <Typography variant='h5' component='div'>
            {child.name}
          </Typography>
          <div dangerouslySetInnerHTML={{ __html: child.info }}></div>
        </CardContent>
```

20.2 Black-Box

The test was done by writing HTML code in the info field of the new child entry (figure 22). The result can be seen in figure 23. Note that this is similar to the test done for WTSG-INPV-01, however the reasons for the exploits are in different places in the code.



Create Child Entry

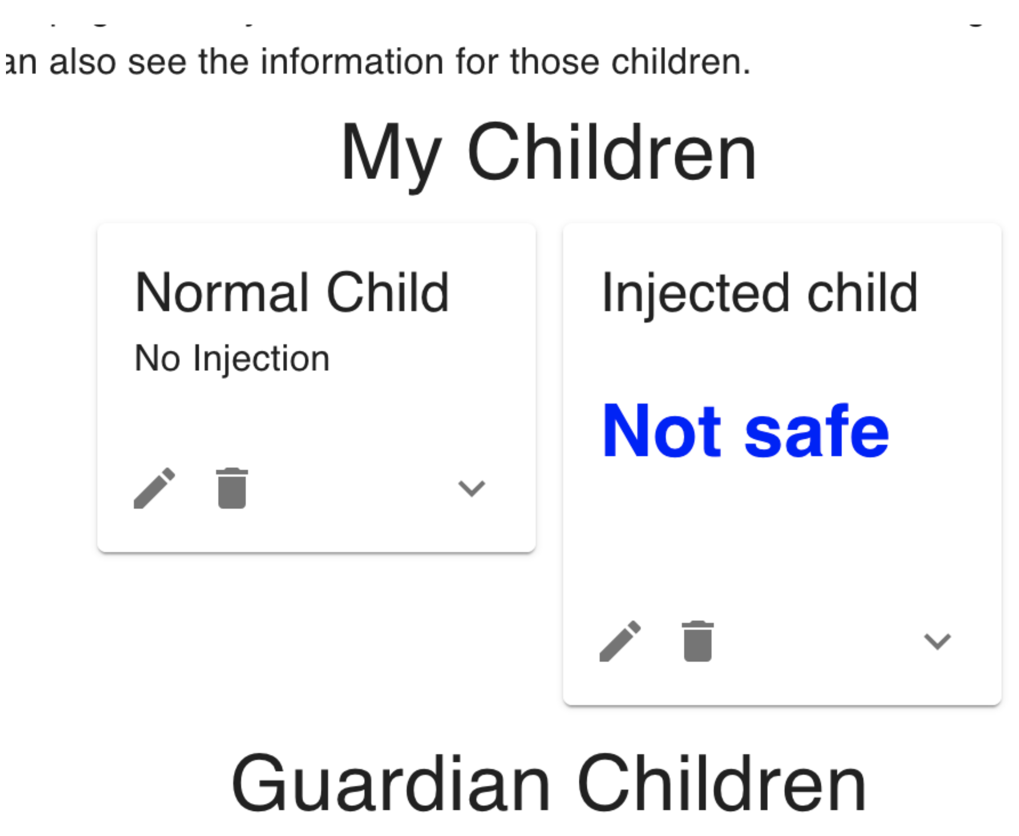
Name *

Injected child

Info *

<h1 style="color:blue;">Not safe</h1>

Figure 22: Doing the HTMl/CSS injection



ın also see the information for those children.

My Children

Normal Child

No Injection

Injected child

Not safe

Guardian Children

Figure 23: Result of the HTML/CSS injection

References

[1] OWASP. *WSTG*. 2022. URL: https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/.

[2] Wireshark. *Wireshark*. 2022. URL: https://www.wireshark.org.