NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4237 Software Security and Data Privacy

# Exercise 4. Mitigating Vulnerabilities

Group 65

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

# Table of Contents

# 1 Introduction

In this assignment we are going to work on the mitigation strategies of different errors, and fix them in the code available in gitlab. Every commit has the name of the Error fixed.

# 2 WSTG-ATHN-01

Sensitive Information Sent via Unencrypted Channels

## 2.1 Mitigation strategy

We will change so that website sends requests over the encrypted https using ssl. We will create a certificate and private key with openssl, and make changes in the code from http to https, as well as changing the port.

## 2.2 Code change

In nginx.conf the following code is added, along with two new files "certificate.crt" and "private.key". The port in docker-compose.yaml is changed. These changes can be seen in git commit with message "ATHN-01".

```
server {
    listen 443 ssl;
    ssl_certificate nginx/certificate.crt
    ssl_certificate_key nginx/private.key
    server_name localhost;
```

The protocol in .env file is changed from http to https, which can be seen in commit "update .env".

```
PROTOCOL=https
```

# 3 WSTG-ATHN-03

Unlimited login attempts, no lockout

## 3.1 Mitigation strategy

As there isn't a limit we can set a timer for blocking the login if it has failed 3 times the password in a row. Setting that variable to 0 if some time happened.

## 3.2 Code change

We will use a package called django-axes for doing the lockout when there are 3 failed logins. First we will add at the file settings some information for getting our tool for lockout.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
```

```
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'corsheaders',
    'rest_framework',
    'apps.adverts.apps.AdvertsConfig',
    'apps.users.apps.UsersConfig',
    'apps.children.apps.ChildrenConfig',
    'apps.offers.apps.OffersConfig',
    'encrypted_model_fields',
    'axes',
]
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',
     axes .middleware.AxesMiddleware,
]
```

Then we add the setting of the lockout

```
 AXES_FAILURE_LIMIT =3
 AXES_COOLOFF_TIME  = timedelta(minutes=60)
AXES_LOCK_OUT_BY_COMBINATION_USER_AND_IP =  True
```

These changes are in the commit named "ATHN-03" and "update ATHN-03".

# 4 WSTG-ATHN-04

Vulnerable MFA

## 4.1 Mitigation strategy

We need to make sure that the one time password is correctly sent with the rest of the login info. Then, we have to check if the one time password is valid.

# 5 WSTG-ATHN-07

Weak password validation

## 5.1 Mitigation strategy

For validating the password it's just set the NumericPasswordValidator which checks if the password isn't entirely numeric. So that we are adding more validators such as minimum length, commom passwords...

### 5.2 Code change

```
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
        'OPTIONS': {
            'min_length': 8,
        }
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

These changes are in the commit "ATHN-07".

## 6 WSTG-ATHZ-02

Children API endpoint open

### 6.1 Mitigation strategy

We will change the code such that only a parent or a guardian of a child can view the child. We will also make the children API only be available when you are authenticated.

## 7 WSTG-ATHZ-02

Accept offers on behalf of others

### 7.1 Mitigation strategy

We need to change so that the offer ID doesn't just increment by 1. We can change it to a large random number, such that it isn't guessable anymore.

## 8 WSTG-SESS-01

Guessable email verification link

### 8.1 Mitigation strategy

The verification link uses the uid and the status, we will make a hash of them so the link is not guessable anymore

### 8.2 Code change

```
import random
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*random.randint(0,1000000), "status": 1})
```

This change is in the second commit named "SESS-01".

## 9 WSTG-SESS-01

Authentication tokens lifetimes

### 9.1 Mitigation strategy

The current access token life expires in 6000 minutes so we will decrease it to just and hour.

### 9.2 Code change

```
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=15),
    'ROTATE_REFRESH_TOKENS': True,
}
```

This change is in the first commit named "SESS-01".

## 10 WSTG-SESS-06

Access token only deleted client side

### 10.1 Mitigation strategy

We will need to make the server invalidate the session when a user logs out. This can be done by either deleting the token in the server side or blacklisting it. We will blacklist the refresh token with the JWT blacklist module.

### 10.2 Code change

We add this to users/views.py

```
from rest_framework_simplejwt.tokens import RefreshToken
```

```
class BlacklistRefreshView(generics.GenericAPIView):
    def post(self, request)
        token = RefreshToken(request.data.get('refresh'))
        token.blacklist()
        return Response("Success")
```

As well as this to users/urls.py

```
router.register('api/logout', views.BlacklistRefreshView, basename="logout"),
```

We also add this to settings.py

```
'rest_framework_simplejwt.token_blacklist',
```

The changes can be seen in the commit named SESS-06.

# 11 WSTG-INPV-02

Unsanitized html field allowing injection

## 11.1 Mitigation strategy

We could sanitize the input, however React takes care of this if we just don't use dangerouslySetInnerHtml, but rather display the text with another method. Therefore we will display the child info the same way as the child title. The change is made in the commit titled "INPV-02".

## 11.2 Code change

We remove this line in Child.jsx:

```
<div dangerouslySetInnerHTML={{ __html: child.info }}></div>
```

and add this:

```
<Typography variant='h7' component='div'>
    {child.info}
</Typography>
```

# 12 WSTG-INPV-05

SQL injection when sending guardian offers

## 12.1 Mitigation strategy

We will prevent the SQL injection by passing the recipient as a named parameter.

## 12.2 Code change

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

The changes can be seen in the second commit named INPV-05.

# 13 WSTG-INPV-05

Vulnerable Django version

## 13.1 Mitigation strategy

We will change the version to the current one.

## 13.2 Code change

```
Django==4.0.3
```

The changes can be seen in the first commit named INPV-05.

# 14 WSTG-CRYP-04

Insecure password hasher

## 14.1 Mitigation strategy

The current password hash used is UnsaltedMD5. This one is not secure anymore so we will change it to a more secure one.

## 14.2 Code change

We will use the default password hasher of django that uses PBKDF2 for storing all passwords but will support checking passwords with PBKDF2SHA1, argon2, and bcrypt.

```
PASSWORD_HASHERS = [
    'django.contrib.auth.hashers.PBKDF2PasswordHasher',
    'django.contrib.auth.hashers.PBKDF2SHA1PasswordHasher',
    'django.contrib.auth.hashers.Argon2PasswordHasher',
    'django.contrib.auth.hashers.BCryptSHA256PasswordHasher',
    'django.contrib.auth.hashers.ScryptPasswordHasher',
]
```

The changes can be seen in the commit named CRYP-04.

# 15 WSTG-BUSL-08

Unexpected file types

## 15.1 Mitigation strategy

We will add the allowed extensions when uploading the file in children/models.py

## 15.2 Code change

```
file = models.FileField(upload_to=child_directory_path,
                        blank=False, validators=[FileValidator(allowed_mimetypes='',
                            allowed_extensions=('.jpg','png'), max_size=math.inf)])
```

The changes can be seen in the commit named BUSL-08.

# 16 WSTG-CLNT-09

Clickjacking

## 16.1 Mitigation strategy

We will be using the X-Frame-Options header and use the option ”same origin” which tells the browser that only pages of the same website are allowed to include that page within an iframe.

## 16.2 Code change

Changing the nginx conf file we add this option that fix the problem.

```
add_header X-Frame-Options SAMEORIGIN;
```

The changes can be seen in the commit named CLNT-09.