<과제 4 > Cross-Site Request Forgery (CSRF): 사이트 간 요청 위조 생성 프로젝트

CSRF(Cross-Site Request Forgery, 사이트 간 요청 위조)는 사용자가 자신의 의지와 무관하게 웹 애플리케이션에 요청을 보내게 만드는 공격입니다. 이를 방지하기 위해 웹 애플리케이션에서는 **CSRF 토큰**을 사용하여 요청의 유효성을 검증합니다.

아래는 CSRF 방지 기능을 **Django**와 **Flask**를 예시로 설명하고, 어떻게 구현하는지를 보여줌.

✅ 1. CSRF란?

공격 시나리오 예시:

1. 사용자가 로그인된 상태에서,
2. 공격자가 조작된 HTML을 통해 사용자의 인증 정보를 사용해 요청을 보냄
3. 서버는 사용자가 보낸 정상 요청으로 판단하고 처리

이걸 막으려면 **서버가 임의의 토큰을 생성해서, 폼 요청 시 같이 전송**하도록 만들어야 함.

✅ 2. Django에서 CSRF 구현

Django는 기본적으로 CSRF 보호 기능이 내장되어 있습니다.

🔷 HTML 템플릿에서 폼 사용 시

```
<form method="post">
  {% csrf_token %}
  <!-- form fields -->
  <input type="submit" value="Submit">
</form>
```

`{% csrf_token %}`을 반드시 `form` 안에 넣어야 합니다. 이게 CSRF 토큰을 숨겨진 input으로 만들어줍니다.

🔷 views.py에서 데코레이터 사용

- 일반적으로는 설정할 필요 없음.
- API View 등에서는 명시적으로 데코레이터를 붙이기도 함

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def my_view(request):
    # CSRF 체크 하지 않음 (주의!)
```

- 혹은 `@csrf_protect`를 사용해서 명시적으로 보호할 수도 있습니다.

✅ 3. Flask에서 CSRF 구현

Flask는 기본적으로 CSRF 보호 기능이 없기 때문에, `Flask-WTF` 확장 모듈을 사용해서 구현합니다.

🔷 설치

```
pip install Flask-WTF
```

🔷 설정

```
from flask import Flask, render_template
from flask_wtf.csrf import CSRFProtect

app = Flask(__name__)
app.secret_key = 'your-secret-key'  # 반드시 설정
csrf = CSRFProtect(app)
```

🔷 HTML 템플릿

```
<form method="post">
    {{ form.hidden_tag() }}
    {{ form.username.label }} {{ form.username() }}
```

```
    <input type="submit" value="Submit">
</form>
```

- Flask-WTF의 `Form` 클래스에서 자동으로 CSRF 토큰을 넣어줍니다.

✅ 4. API에서 CSRF 처리 (SPA, AJAX)

만약 Vue/React 등의 SPA에서 CSRF를 구현한다면:

- Django: CSRF 토큰을 쿠키에 담아 보내고, JS가 이를 `X-CSRFToken` 헤더로 설정해서 전송
- 예시:

```
fetch('/api/data', {
  method: 'POST',
  headers: {
    'X-CSRFToken': getCookie('csrftoken'),
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ key: 'value' })
});
```

아래는 **Django 기반으로 CSRF 공격이 어떻게 발생하는지**, 그리고 그것을 **실습을 통해 직접 체험**할 수 있도록 구성한 예제입니다.

> 💡 실습 목적: CSRF 공격의 원리를 이해하고, Django의 기본 보안 설정이 이를 어떻게 막는지 확인

🧪 1. 실습 구성

1.1. 준비 환경

- Django 프로젝트 (예: `csrf_demo`)
- 앱 (예: `main`)
- 간단한 로그인 시스템과 게시판(글쓰기) 구현

1.2. 주요 구성

기능	설명
`/login/`	로그인 기능 (세션 기반)
`/write/`	게시글 작성 폼 (POST 요청 필요)
`/attack/`	공격자가 만든 악성 사이트

🛠️ 2. 코드 작성

2.1. 로그인 뷰 (예: `views.py`)

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

def user_login(request):
    if request.method == 'POST':
        user = authenticate(username=request.POST['username'], password=
request.POST['password'])
        if user:
            login(request, user)
            return redirect('/write/')
    return render(request, 'login.html')
```

2.2. 게시글 작성 뷰

```
@login_required
def write_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')
        print(f"글 작성됨: {title}, {content}")
        return HttpResponse("게시글 작성 완료")
    return render(request, 'write.html')
```

2.3. 게시글 작성 폼 (`templates/write.html`)

```
<h2>게시글 작성</h2>
<form method="post">
 {% csrf_token %}
 제목: <input type="text" name="title"><br>
 내용: <textarea name="content"></textarea><br>
 <input type="submit" value="작성">
</form>
```

💥 3. CSRF 공격 시도

3.1. 공격자 사이트 (`templates/attack.html`)

> 이 페이지는 CSRF 토큰 없이 글을 자동 전송함

```
<h2>이 페이지는 공격자 사이트입니다</h2>
<form action="http://127.0.0.1:8000/write/" method="post">
 <input type="hidden" name="title" value="공격제목">
 <input type="hidden" name="content" value="이건 공격자가 작성한 글입니다.">
 <input type="submit" value="클릭하지 마세요">
</form>

<script>
 // 자동 전송 (사용자 인증 정보가 있으면 글이 작성되는지 테스트)
 document.forms[0].submit();
</script>
```

3.2. `views.py` 에 공격 페이지 추가

```
def csrf_attack(request):
    return render(request, 'attack.html')
```

3.3. `urls.py`

```
urlpatterns = [
    path('login/', views.user_login),
    path('write/', views.write_post),
    path('attack/', views.csrf_attack),
]
```

🔐 4. 실습 진행 순서

1. Django 서버 실행: `python manage.py runserver`
2. 브라우저에서 `http://127.0.0.1:8000/login/` 접속, 로그인
3. `http://127.0.0.1:8000/attack/` 방문
4. **결과 확인**
    - CSRF 토큰 없이 요청 → Django가 403 오류 반환
    - 에러 메시지: `CSRF verification failed. Request aborted.`

✅ 5. 결론

- Django는 **기본적으로 CSRF 방어가 활성화**되어 있으므로, 토큰 없는 요청은 거절됩니다.
- 이 실습을 통해:
    - CSRF 공격의 원리 이해
    - Django가 어떻게 보호하는지 확인
    - `@csrf_exempt`를 잘못 사용하면 위험하다는 점도 알 수 있음

Django 기반 **CSRF 공격 실습 예제**의 전체 흐름을 **단계별로 상세하게** 설명함. 각 단계는 **설정 → 로그인 시스템 구현 → 게시글 작성 → 공격 페이지 구성 → 실습 실행 → 결과 확인** 흐름으로 구성되어 있습니다.

🧪 CSRF 공격 실습 예제 (Django)

🔧 1단계: 프로젝트 및 앱 설정

1.1. 프로젝트 생성

```
django-admin startproject csrf_demo
cd csrf_demo
python manage.py startapp main
```

1.2. `settings.py` 에 앱 등록

```
INSTALLED_APPS = [
    ...
    'main',
]
```

1.3. 템플릿 디렉토리 설정

`settings.py` 에 `TEMPLATES` 옵션에서 `'DIRS': [BASE_DIR / 'templates']` 로 수정하고, `templates` 폴더 생성

🔐 2단계: 로그인 기능 구현

2.1. 기본 유저 생성 (관리자)

```
python manage.py createsuperuser
```

2.2. 로그인 폼 템플릿 (`templates/login.html`)

```
<h2>로그인</h2>
<form method="post">
  {% csrf_token %}
  <label>아이디: <input type="text" name="username"></label><br>
  <label>비밀번호: <input type="password" name="password"></label><br
>
  <input type="submit" value="로그인">
</form>
```

2.3. 로그인 뷰 (`main/views.py`)

```
from django.shortcuts import render, redirect
from django.contrib.auth import authenticate, login

def user_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            return redirect('/write/')
    return render(request, 'login.html')
```

📝 3단계: 게시글 작성 기능 구현

3.1. 게시글 작성 폼 템플릿 (`templates/write.html`)

```
<h2>게시글 작성</h2>
<form method="post">
  {% csrf_token %}
  제목: <input type="text" name="title"><br>
  내용: <textarea name="content"></textarea><br>
  <input type="submit" value="작성">
</form>
```

3.2. 게시글 작성 뷰 (`main/views.py`)

```
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

@login_required
def write_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')
        print(f"글 작성됨: 제목={title}, 내용={content}")
```

```
        return HttpResponse("게시글 작성 완료")
    return render(request, 'write.html')
```

3.3. URL 설정 (csrf_demo/urls.py)

```
from django.contrib import admin
from django.urls import path
from main import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('login/', views.user_login),
    path('write/', views.write_post),
]
```

💣 4단계: 공격자 페이지 만들기

> CSRF 공격 페이지를 만들어 Django의 방어 메커니즘을 실습합니다.

4.1. 공격용 HTML (templates/attack.html)

```
<h2>공격자 페이지입니다</h2>
<form action="http://127.0.0.1:8000/write/" method="post">
  <input type="hidden" name="title" value="공격 제목">
  <input type="hidden" name="content" value="공격자가 강제로 작성한 글입니다.">
  <input type="submit" value="이 버튼을 누르지 마세요">
</form>

<script>
  document.forms[0].submit();  // 자동으로 전송 (사용자 몰래)
</script>
```

4.2. 뷰 추가 (main/views.py)

```
def csrf_attack(request):
    return render(request, 'attack.html')
```

4.3. URL 추가 (`csrf_demo/urls.py`)

```
urlpatterns += [
    path('attack/', views.csrf_attack),
]
```

🧪 5단계: 실습 실행

1. Django 서버 실행

   ```
   python manage.py runserver
   ```

2. 브라우저에서 접속
   - 로그인: `http://127.0.0.1:8000/login/`
   - 정상 글 작성: `http://127.0.0.1:8000/write/`
   - 공격 페이지 접속: `http://127.0.0.1:8000/attack/`

🛡️ 6단계: 결과 확인 및 보안

✅ 기대 결과

- 정상 작성 시: 글 작성 성공
- 공격 페이지 접근 시:
  - **403 오류 발생**
  - 메시지: `CSRF verification failed. Request aborted.`

🔍 이유

- Django는 POST 요청에 대해 CSRF 토큰을 확인
- `attack.html`에서 전송된 요청에는 CSRF 토큰이 없음
- 따라서 Django가 요청을 **차단**

⚠️ 7단계: 실험 변형 (고급)

👉 보안을 비활성화해보기 (`@csrf_exempt`)

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
@login_required
def write_post(request):
    ...
```

이렇게 하면 공격 페이지에서 글 작성이 **성공**하게 됩니다.

=> 절대로 실 서비스에서는 사용하면 안 됩니다!

✅ 요약

항목	설명
목적	CSRF 공격 이해 및 방어 실습
방법	Django의 로그인-글쓰기 흐름과, 공격용 페이지 구성
결과	Django는 기본적으로 CSRF 공격을 방어
교훈	`@csrf_exempt`의 위험성, CSRF 토큰의 필요성

`CSRF verification failed. Request aborted.` 오류는 **Django가 CSRF 토큰 검증에 실패했을 때 발생하는 보안 에러**입니다. 이 에러는 의도된 보안 기능이 작동 중일 수도 있고, 코드 작성 중 실수일 수도 있습니다.

❗ 에러 원인 요약

원인	설명
🔐 CSRF 토큰 누락	POST 요청에 CSRF 토큰이 포함되지 않음
⛔ 잘못된 토큰	토큰이 만료되었거나 올바르지 않음

⚙️ 브라우저 쿠키 차단	CSRF 토큰이 세션 쿠키 기반인데 쿠키가 차단됨
🔄 클라이언트와 서버가 도메인 불일치	AJAX 요청 시 토큰 전송 방식이 잘못됨

🧪 가장 흔한 상황: HTML form에서 토큰을 빼먹음

✅ 잘된 예시

```
<form method="post">
  {% csrf_token %}
  <input type="text" name="title">
  <input type="submit">
</form>
```

❌ 잘못된 예시

```
<form method="post">
  <!-- {% csrf_token %} 누락됨 -->
  <input type="text" name="title">
  <input type="submit">
</form>
```

→ 이 경우 Django는 **403 Forbidden**과 함께 `CSRF verification failed` 에러를 반환합니다.

🛠️ 해결 방법: 상황별 체크리스트

✅ [1] 템플릿에 `{% csrf_token %}` 추가

- `POST` 메서드를 사용하는 모든 폼에 포함해야 합니다.

```
<form method="post">
  {% csrf_token %}
  ...
</form>
```

✅ [2] CSRF Middleware 켜져 있는지 확인 (`settings.py`)

```
MIDDLEWARE = [
    'django.middleware.csrf.CsrfViewMiddleware',  # 기본값이므로 보통 이미 있음
    ...
]
```

✅ [3] `@csrf_exempt` 사용하지 않았는지 확인

개발 중이라도 CSRF 보호는 꺼두면 안 됩니다. 정말 필요한 경우에만 아래처럼 명시합니다:

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def my_view(request):
    ...
```

✅ [4] AJAX 또는 SPA에서 요청하는 경우

React/Vue에서 Django API를 호출할 때는 토큰을 `X-CSRFToken` 헤더로 보내야 합니다.

예시: JS Fetch 요청

```
fetch('/api/submit/', {
  method: 'POST',
  headers: {
    'X-CSRFToken': getCookie('csrftoken'),
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(data)
});
```

✅ [5] 세션/쿠키 확인

- CSRF 토큰은 세션 쿠키 기반입니다.
- 브라우저에서 쿠키가 비활성화되어 있다면 검증에 실패합니다.

🔍 디버깅 팁

1. 개발자 도구(F12) → Network → 요청 확인
    - Request Headers에 `X-CSRFToken` 또는 폼 안에 hidden input이 있는지 확인
2. Django 로그 확인
    - `manage.py runserver` 창에 디버깅 메시지가 뜰 수 있음

🧪 테스트용으로 임시로 비활성화 (주의)

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def test_post(request):
    return HttpResponse("임시로 CSRF 검증 생략됨")
```

> ⚠️ 실제 서비스에선 절대로 쓰지 마세요!

✨ 마무리

> "CSRF verification failed"는 Django가 보안을 잘 지키고 있다는 뜻입니다.
>
> 다만, 실수로 보호 기능을 우회하거나 토큰을 누락하면 의도치 않게 막힐 수 있어요.

`http://127.0.0.1:8000/admin/`(Django Admin 페이지)에서 `write`된 글을 확인하려면, 글 데이터를 저장할 **모델(Post)**을 만들어서 **admin 페이지에 등록**해야 해요. 지금까지는 단순히 `print()`로만 출력하고 있어서 DB에 저장되지 않았습니다.

✅ 목표

- 글 모델(`Post`) 생성
- `write_post`에서 글을 DB에 저장

- Django Admin에 모델 등록하여 관리 가능하게 함

🛠️ 1단계: 모델 생성

`main/models.py`에 글(Post) 모델 추가:

```
from django.db import models
from django.contrib.auth.models import User

class Post(models.Model):
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"[{self.author.username}] {self.title}"
```

🛠️ 2단계: 마이그레이션 적용

```
python manage.py makemigrations
python manage.py migrate
```

🛠️ 3단계: admin 페이지에 Post 등록

`main/admin.py` 파일을 수정:

```
from django.contrib import admin
from .models import Post

@admin.register(Post)
class PostAdmin(admin.ModelAdmin):
    list_display = ('id', 'title', 'author', 'created_at')
    search_fields = ('title', 'content', 'author__username')
```

🛠️ 4단계: `write_post` 뷰 수정하여 DB에 저장

`main/views.py`:

```
from .models import Post
from django.contrib.auth.decorators import login_required
from django.http import HttpResponse

@login_required
def write_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')

        Post.objects.create(
            author=request.user,
            title=title,
            content=content
        )

        return HttpResponse("게시글 작성 완료")
    return render(request, 'write.html')
```

✅ 5단계: 관리자 계정으로 확인

1. 서버 실행

   ```
   python manage.py runserver
   ```

2. 브라우저에서 접속: http://127.0.0.1:8000/admin/
3. 관리자 로그인 후 `Post` 모델 확인
4. 작성된 글 목록이 보이면 성공 🎉

✅ 마무리

이제 `write_post()`에서 작성된 글이 **데이터베이스에 저장**되고, **admin 페이지에서 확인**할 수 있음.

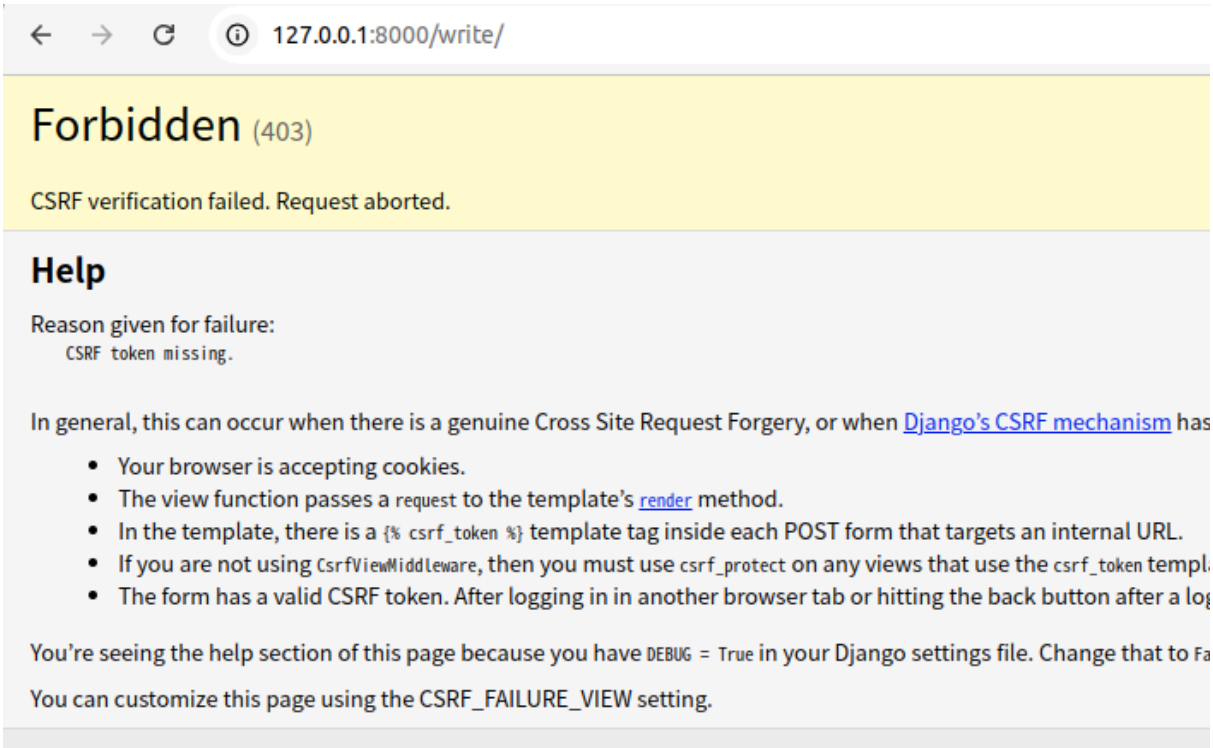
< 과제 4 내용>

- 제출 내용
  - @csrf_exempt 를 사용하지 않을 때의 웹사이트 결과 화면 캡쳐 (3점)
  - @csrf_exempt 를 사용할 때 웹사이트 결과 화면 캡쳐 (3점)
  - 127.0.0.8000/admin에서 공격자가 작성한 리스트 결과 화면 캡쳐 (2점)
  - 구현 코드 깃허브 링크 (2점)
- **제출일시: 2025년 4월 7일 (월) 13:30**
  - 가상강의실에 제출
  - 늦게 제출은 인정하지 않습니다

  < 예시 >
  - @csrf_exempt 를 사용하지 않을 때

```python
#@csrf_exempt
/write/
@login_required  1개의 사용 위치
def write_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')

        Post.objects.create(
            author=request.user,
            title=title,
            content=content
        )

        return HttpResponse("게시글 작성 완료")
    return render(request, template_name: 'write.html')

```

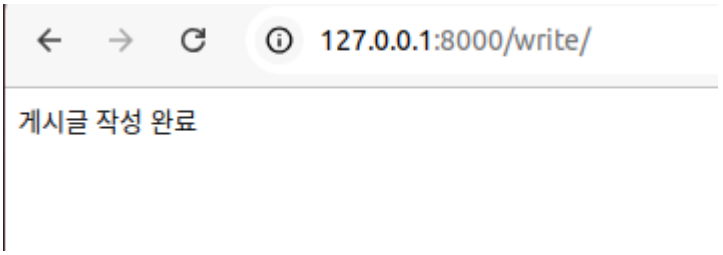
- 127.0.0.1:8000/attack으로 접근하면, 서버가 CSRF verification이 인증되지 않았음을 나타냄



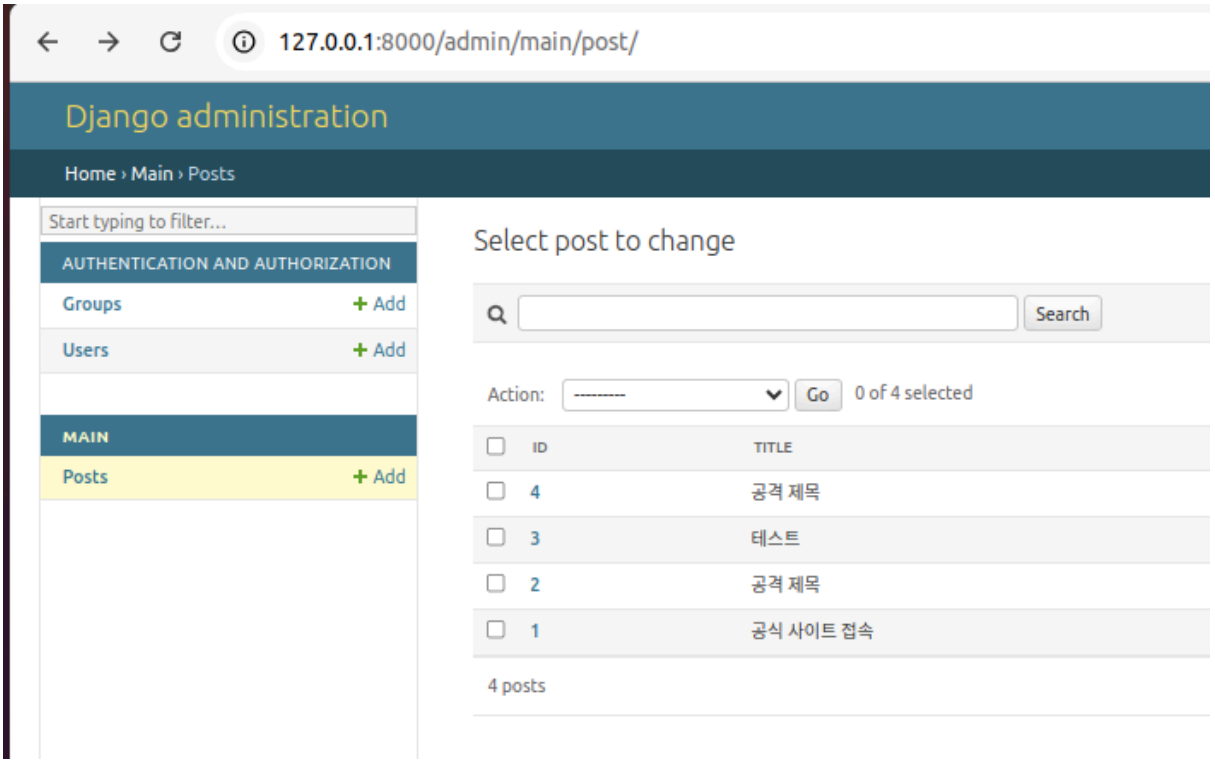
- @csrf_exempt 를 사용할때

```python
@csrf_exempt  1개의 사용 위치
@login_required
def write_post(request):
    if request.method == 'POST':
        title = request.POST.get('title')
        content = request.POST.get('content')

        Post.objects.create(
            author=request.user,
            title=title,
            content=content
        )

        return HttpResponse("게시글 작성 완료")
    return render(request, template_name: 'write.html')
```

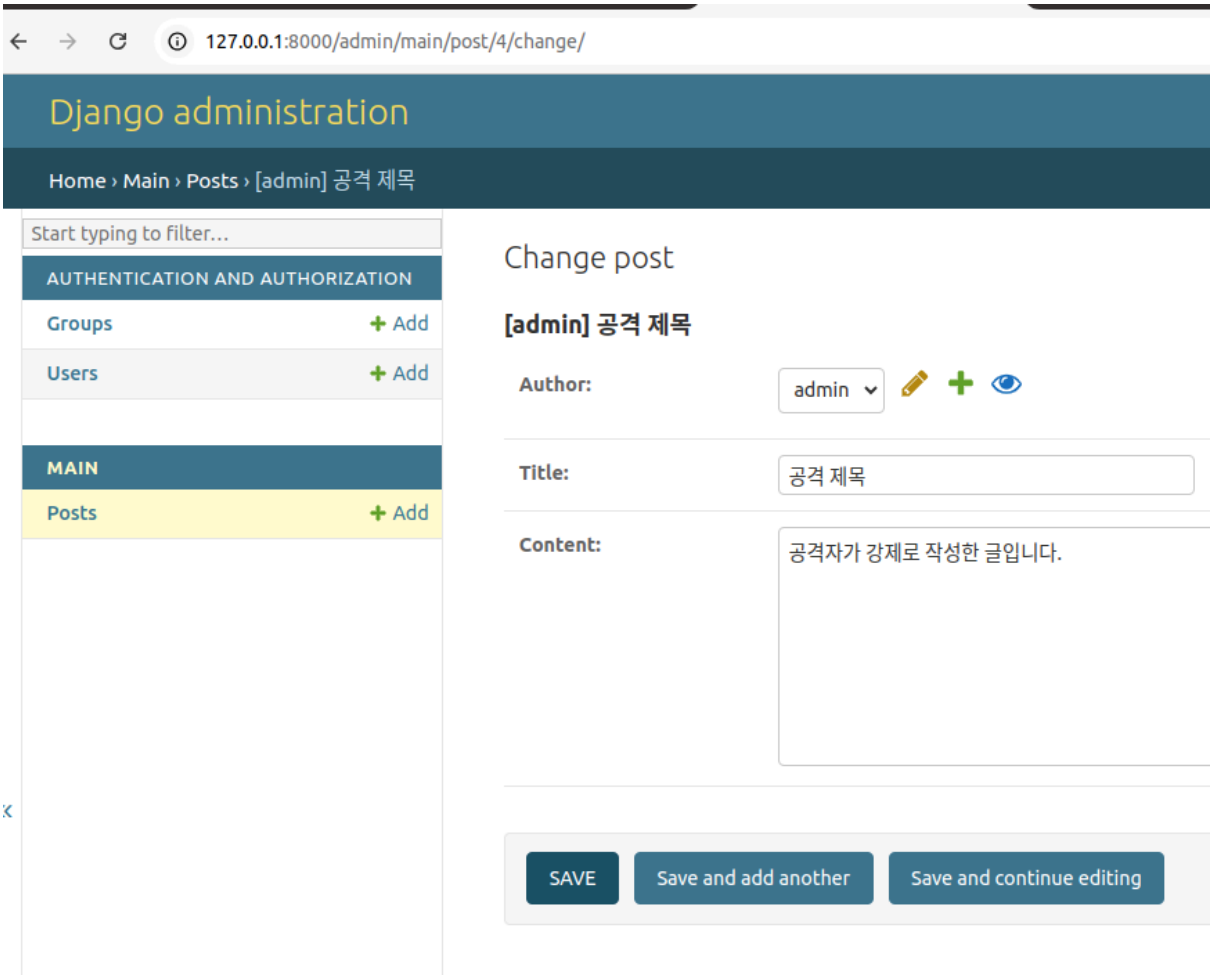
- 127.0.0.1:8000/attack으로 접근하면, 서버의 CSRF verification이 면제되어, attacker의 공격이 인증 프로세스없이 일반글 게시가 되는 상황이 됨



- 127.0.0.8000/admin에서 공격자가 작성한 리스트를 볼 수 있다.



- 2번, 4번이 공격자가 작성한 글이다.


127.0.0.1:8000/admin/main/post/4/change/
Django administration
Home › Main › Posts › [admin] 공격 제목
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
MAIN
Posts + Add
Change post
[admin] 공격 제목
Author:
admin
Title:
공격 제목
Content:
공격자가 강제로 작성한 글입니다.
SAVE
Save and add another
Save and continue editing