ZAP Security Remediation - Implementation Guide

Estimated Time: 90 minutes

Difficulty: Medium

Impact: Fixes 8/8 security issues (5 Medium + 3 Low)

📋 Pre-Implementation Checklist

- [ ] Backup current codebase
- [ ] Note current ZAP scan results
- [ ] Have test environment ready
- [ ] Django server can be restarted
- [ ] Access to browser DevTools for verification

🚀 Step-by-Step Implementation

Step 1: Update CSP Configuration (20 minutes)

File: `backend/config/addon/csp.py` (or create if doesn't exist)

1. **Backup existing file**:

```bash
cp backend/config/addon/csp.py backend/config/addon/csp.py.backup
```

2. **Replace with production CSP**:
   - Copy content from `csp_production.py` (provided)
   - Update `backend/config/addon/csp.py`

3. **Verify import in base settings**:

```python
# backend/config/settings/base.py
# Should have this line:
from config.addon.csp import *
```

4. **Test CSP is loaded**:

```bash
python manage.py shell
>>> from django.conf import settings
>>> print(settings.CSP_SCRIPT_SRC)
# Should show: ["'self'"] (production) or ["'self'", "'unsafe-inline'", "'unsafe-eval'"] (dev)
```

Verification:

```bash
# Start server
python manage.py runserver

# Check CSP header
curl -I http://localhost:8000/admin/login/ | grep Content-Security-Policy

# Expected: Should NOT contain 'https:' or 'unsafe-inline' (if DEBUG=False)
```

Step 2: Add Enhanced Security Middleware (25 minutes)

File: `backend/config/middleware/security.py`

1. **Backup existing middleware**:

```bash
cp backend/config/middleware/security.py backend/config/middleware/security.py.backup
```

2. **Update with enhanced middleware**:
   - Copy classes from `security_middleware_enhanced.py`
   - Add to `backend/config/middleware/security.py`

3. **Update MIDDLEWARE in settings**:

```python
# backend/config/settings/base.py
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'whitenoise.middleware.WhiteNoiseMiddleware',
    'config.middleware.security.SecurityHeadersMiddleware',  # ADD
    'config.middleware.security.SecureStaticFilesMiddleware',  # ADD
    'csp.middleware.CSPMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'config.middleware.security.CookieSecurityMiddleware',  # ADD (after Auth)
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

4. **Restart Django**:

```bash
python manage.py runserver
```

Verification:

```bash
# Check Server header is removed
curl -I http://localhost:8000/admin/login/ | grep Server
# Expected: Should NOT show Server header

# Check X-Content-Type-Options
curl -I http://localhost:8000/admin/login/ | grep X-Content-Type-Options
# Expected: X-Content-Type-Options: nosniff
```

Step 3: Update Security Settings (15 minutes)

File: `backend/config/settings/security.py`

1. **Backup existing security settings**:

```bash
cp backend/config/settings/security.py backend/config/settings/security.py.backup
```

2. **Update cookie security**:

```python
```

```
# Add/update these settings in config/settings/security.py

# CRITICAL FIX: HttpOnly on CSRF cookie
CSRF_COOKIE_HTTPONLY = True
CSRF_USE_SESSIONS = True

# Ensure session cookie flags
SESSION_COOKIE_HTTPONLY = True
SESSION_COOKIE_SAMESITE = 'Strict'
```

3. **Verify settings loaded**:

```bash
python manage.py shell
>>> from django.conf import settings
>>> print(settings.CSRF_COOKIE_HTTPONLY)
# Expected: True
>>> print(settings.SESSION_COOKIE_HTTPONLY)
# Expected: True
```

Verification:

```bash
# Login to admin and check cookies in browser DevTools
# 1. Open http://localhost:8000/admin/login/
# 2. F12 → Application → Cookies → localhost
# 3. Check sessionid cookie → HttpOnly should be ✓
# 4. Check csrftoken cookie → HttpOnly should be ✓ (if present)
```

Step 4: Add Security Tests (15 minutes)

File: `backend/config/tests/test_security_remediation.py`

1. **Create test file**:

```bash
# Copy provided test file
cp test_security_remediation.py backend/config/tests/test_security_remediation.py
```

2. **Run tests**:

```bash
pytest backend/config/tests/test_security_remediation.py -v
```

3. **Expected results**:

```
test_csp_header_present_on_admin PASSED
test_no_server_header PASSED
test_x_content_type_options_present PASSED
test_session_cookie_httponly PASSED
test_csrf_cookie_httponly PASSED

============ 13 passed in 2.45s ============
```

Step 5: Run ZAP Scan (15 minutes)

1. **Start Django server**:

```bash
```

```
python manage.py runserver
```

2. **Run OWASP ZAP**:
   - Open ZAP
   - Set target: http://localhost:8000
   - Run Active Scan
   - Wait for completion (~10 minutes)
3. **Check results**:
   - Medium issues: Should be 0 (was 5)
   - Low issues: Should be 0 (was 3)
4. **Generate report**:
   - Report → Generate HTML Report
   - Compare with previous scan

✅ Verification Checklist

Browser DevTools Verification

1. **Open Admin Login**: http://localhost:8000/admin/login/
2. **Check Network Headers** (F12 → Network → admin/login/):
   - [ ] Content-Security-Policy present
   - [ ] CSP does NOT contain `https:` wildcard
   - [ ] CSP does NOT contain `unsafe-inline` (production)
   - [ ] CSP does NOT contain `unsafe-eval` (production)
   - [ ] X-Content-Type-Options: nosniff

   - [ ] X-Frame-Options: DENY
   - [ ] Referrer-Policy present
   - [ ] Server header NOT present

3. **Check Console** (F12 → Console):
   - [ ] No CSP violation errors
   - [ ] Admin interface loads correctly

4. **Check Cookies** (F12 → Application → Cookies):
   - [ ] sessionid → HttpOnly ✓
   - [ ] sessionid → SameSite: Strict
   - [ ] csrftoken → HttpOnly ✓ (if cookie-based)

Automated Test Verification

```bash
# All security tests should pass
pytest backend/config/tests/test_security_remediation.py -v

# Expected: 13/13 tests passing
```

ZAP Scan Verification

Before Remediation:

- High: 0
- Medium: 5 ❌
- Low: 3 ❌

After Remediation:

- High: 0 ✅

- Medium: 0 ✅
- Low: 0 ✅

🐛 Troubleshooting

Issue: Admin won't load (CSP blocking scripts)

Symptom: Console shows CSP violation errors

Solution:

1. Check if you're in production mode (`DEBUG=False`)
2. For development, ensure CSP allows unsafe-inline:

```python
# config/addon/csp.py
if IS_DEBUG:
    CSP_SCRIPT_SRC = ["'self'", "'unsafe-inline'", "'unsafe-eval'"]
```

Issue: Tests failing

Symptom: `test_csp_no_unsafe_inline_in_script_src` fails

Cause: You're in DEBUG mode (expected behavior)

Solution: Tests that check for strict CSP will fail in DEBUG=True mode. This is normal. Run in production mode:

```bash
DJANGO_DEBUG=False pytest config/tests/test_security_remediation.py::CSPSecurityTests -v
```

Issue: Cookies still showing without HttpOnly

Symptom: DevTools shows HttpOnly unchecked

Cause: Settings not loaded or browser cache

Solution:

1. Clear browser cookies completely
2. Restart Django server
3. Verify settings:

```python
python manage.py shell
>>> from django.conf import settings
>>> print(settings.CSRF_COOKIE_HTTPONLY)
```

Issue: Static files blocked by CSP

Symptom: CSS/JS not loading, console shows CSP errors

Solution: Add static file domain to CSP:

```python
# If using S3 or CDN
CSP_SCRIPT_SRC = ["'self'", "https://your-cdn.com"]
CSP_STYLE_SRC = ["'self'", "https://your-cdn.com"]
```

📊 Success Metrics

Before Implementation

- ZAP Medium Issues: 5

- ZAP Low Issues: 3
- Security Score: ~85%

After Implementation

- ZAP Medium Issues: 0 ✅
- ZAP Low Issues: 0 ✅
- Security Score: ~95% ✅

Expected Improvements

1. ✅ XSS protection strengthened (no unsafe-inline)
2. ✅ Clickjacking prevented (frame-ancestors 'none')
3. ✅ Information disclosure prevented (no Server header)
4. ✅ Cookie theft prevented (HttpOnly flags)
5. ✅ MIME-sniffing attacks blocked (nosniff on all responses)

🎯 Next Steps

1. **Deploy to staging** and re-run ZAP scan
2. **Monitor production** for CSP violations
3. **Document any CSP exceptions** needed for third-party integrations
4. **Set up automated security scanning** in CI/CD
5. **Schedule quarterly security audits**

📞 Support

Issues? Check:

1. Django logs: `backend/logs/`
2. Browser console: F12 → Console tab
3. Test output: `pytest -v`

Still stuck? Review:

- `V2_SECURITY_ASSESSMENT.md` - Full security analysis
- `BACKEND_DOCUMENTATION.md` - System documentation
- Django security docs: https://docs.djangoproject.com/en/stable/topics/security/

Document Version: 1.0
Created: November 14, 2025
Estimated Completion: 90 minutes
Status: Ready for implementation