Testrapport

Project: https://mheruer.pythonanywhere.com
Versie: 1.0
Testdatum: 20 Maart 2025
Tester: Matthias Heruer

1. Inleiding

Dit document beschrijft de teststrategie en resultaten van de beveiligingsaudit uitgevoerd op de website. Het doel van deze audit is om kwetsbaarheden te identificeren en de beveiliging van de applicatie te verbeteren.

2. Testomgeving

Framework: Django (versie 5.1.4)
Database: MySQL
Server: uWSGI 2.0.20 op Linux-6.5.0-1022-aws, x86_64 architectuur, nodenaam: blue-liveweb27
Testtools: Pytest, observatory.mozilla.org, Django's tool check –deploy, djcheckup.com

3. Testmethodologie
De testmethodologie omvat zowel handmatige als geautomatiseerde tests.

De volgende gebieden worden onderzocht:

4. Testgebieden en Resultaten

4.1. Debug = False
Test: Controleer of de DEBUG-parameter is ingesteld op False in de live omgeving.
Resultaat: De DEBUG-parameter is correct ingesteld op False, wat bevestigt dat de website in de live omgeving niet onnodige informatie lekt die door kwaadwillenden kan worden misbruikt.

4.2. Beveiligde verbinding (HTTPS)
Test: Controleer of de website een beveiligde HTTPS-verbinding gebruikt.
Resultaat: De website maakt gebruik van een HTTPS-verbinding, wat zorgt voor een versleutelde communicatie tussen de server en de client. Dit draagt bij aan de bescherming van gevoelige gegevens.

4.3. Wijziging van de standaard Django admin URL
Test: Controleer of de standaard Django admin URL (/admin) is gewijzigd naar een unieke URL om ongeautoriseerde toegang te voorkomen.
Resultaat: De standaard Django admin URL is succesvol gewijzigd naar een unieke URL, waardoor het moeilijker wordt voor kwaadwillenden om de admin-pagina te vinden en te benaderen.

4.4. Inlogbeveiliging
Test: Controleer of sterke wachtwoorden vereist zijn voor gebruikersaccounts.
Resultaat: Het systeem vereist sterke wachtwoorden, wat helpt om ongeautoriseerde toegang te voorkomen en de beveiliging van gebruikersaccounts te waarborgen.

4.5. Cross-site Scripting (XSS)

Test: Controleer of de applicatie beschermd is tegen Cross-site Scripting (XSS) aanvallen door gebruikersinvoer te filteren en te ontsmetten.
Resultaat: De applicatie heeft effectieve maatregelen geïmplementeerd om XSS-aanvallen te voorkomen, zoals het correct ontsmetten van gebruikersinvoer en het gebruik van Content Security Policy (CSP). Dit helpt bij het beschermen van gebruikers tegen kwaadaardige scripts en verhoogt de algehele veiligheid van de applicatie.

4.6. SSL Redirect
Test: Controleer of de website automatisch HTTP-verzoeken omleidt naar HTTPS om een veilige verbinding af te dwingen.
Resultaat: De website omleidt succesvol alle HTTP-verzoeken naar HTTPS, wat zorgt voor een versleutelde en veilige communicatie tussen de server en de client. Dit voorkomt dat gevoelige gegevens onversleuteld worden verzonden.

4.7. HTTP Strict Transport Security (HSTS)
Test: Controleer of HSTS (HTTP Strict Transport Security) is ingeschakeld om ervoor te zorgen dat browsers altijd via HTTPS met de server verbinden en om Man-in-the-Middle aanvallen te voorkomen.
Resultaat: HSTS is succesvol ingeschakeld, waardoor browsers gedwongen worden om altijd via HTTPS verbinding te maken met de server. Dit voorkomt dat gevoelige gegevens onversleuteld worden verzonden en biedt extra bescherming tegen Man-in-the-Middle aanvallen.

4.8. Cross-site request forgery (CSRF) bescherming
Test: Controleer of CSRF-bescherming is ingeschakeld om te voorkomen dat onbevoegde verzoeken namens een gebruiker worden uitgevoerd.
Resultaat: CSRF-bescherming is succesvol ingeschakeld, waardoor de applicatie effectief beschermd is tegen CSRF-aanvallen. Dit helpt bij het waarborgen van de integriteit van gebruikerssessies.

4.9. Gebruik van python-decouple
Test: Controleer of de applicatie de python-decouple bibliotheek gebruikt om gevoelige configuratie-informatie, zoals API-sleutels en databasegegevens, uit de broncode te halen en deze in een aparte configuratiebestand (.env) op te slaan.
Resultaat: De applicatie maakt succesvol gebruik van python-decouple om gevoelige configuratie-informatie extern op te slaan in een .env-bestand. Dit vermindert het risico op onbedoelde blootstelling van gevoelige gegevens in de broncode.

5. Aanbevelingen
Op basis van de testresultaten worden de volgende aanbevelingen gedaan om de beveiliging van de Django-website te verbeteren:

Content Security Policy (CSP)
Implementeer een Content Security Policy (CSP) om te voorkomen dat niet-goedgekeurde bronnen worden geladen en uitgevoerd op de website. CSP helpt bij het beschermen tegen clickjacking, cross-site scripting (XSS) en andere code-injectie aanvallen.

Leesbaar e-mailadres op /contact/ pagina
Verberg e-mailadressen op de /contact/ pagina om te voorkomen dat spambots deze kunnen oogsten voor het versturen van ongewenste e-mails. Gebruik JavaScript om e-mailadressen

dynamisch weer te geven of encodeer ze zodat ze niet direct leesbaar zijn in de HTML-broncode. Dit helpt bij het verminderen van de kans op spam en beschermt de privacy van gebruikers.

Django Admin Honeypot
Gebruik de django-admin-honeypot bibliotheek om een honeypot voor de Django admin-pagina in te stellen. Dit zal valse login-pagina's tonen om ongeautoriseerde toegangspogingen te detecteren en te waarschuwen, wat bijdraagt aan de algehele beveiliging van de admin-pagina.

Django Security Middleware
Controleer en configureer de ingebouwde Django Security Middleware om extra beveiligingsheaders zoals X-Content-Type-Options, X-Frame-Options en X-XSS-Protection toe te voegen. Deze headers helpen bij het voorkomen van diverse aanvallen.

6. Conclusie
De beveiligingsaudit van de website heeft enkele kwetsbaarheden geïdentificeerd. Door de implementatie van de hierboven genoemde aanbevelingen kan de algehele beveiliging van de applicatie worden verbeterd.