ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ ІНФОРМАЦІЇ У WEB-ЗАСТОСУНКУ КРАУДСОРСИНГУ СОЦІАЛЬНОЇ ПІДТРИМКИ MICROVOLUNTEER

Поліщук Я.О.
Державний університет інформаційно-комунікаційних технологій, Київ

ВСТУП

Актуальність і постановка проблеми. У сучасному світі волонтерства все починається з простого: людина просить про допомогу, переходить до чат-боту чи на web-сервіс і залишає в системі адресу, телефон, координати та опис ситуації. Звичайний web-сервіс розглядає це як персональні дані, а у випадку платформи для соціальної підтримки, це питання безпеки цільового отримувача послуги та допомоги. Це може бути людина з вадами, людина похилого віку, внутрішньо переміщені особи (ВПО), багатодітна сім'я, батько чи мати, який/яка самостійно виховує дитину. У OWASP Top 10:2021 [1] серед найнебезпечніших ризиків стоять Broken Access Control і Cryptographic Failures. Застосунок краудсорсингу соціальної підтримки не може нехтувати безпекою користувачів, саме тому важливо визначити, як зберігаються паролі, контактні дані, адреси і як запобігти їхнє перехоплення.

Мета роботи. Метою MicroVolunteer є обґрунтування та демонстрація на прикладі MicroVolunteer багаторівневої моделі захисту, базуючись на вимогах трьох ключових міжнародних організацій та спільнот: OWASP, NIST та IETF.

Аналіз останніх досліджень і публікацій. Під час створення захисних шарів для сервісу було використано OWASP Top 10:2021 [1] як базову карту ризиків для web-додатків. З корисних стандартів та практик було взято насамперед контроль доступу, захист від ін'єкцій, новітні практики проєктування та криптографічну систему. OWASP ASVS 4.0.3 [2] використовується як більш детальне джерело з планом, порадами та прикладами реалізації: сесій, автентифікації, прав доступу, криптографічним захистом та роботою з вхідними даними користувача. Для роботи з паролями проаналізовано публікацію від NIST: SP 800-63B [3] та технічний довідник OWASP Password Storage Cheat Sheet [4]. Ці джерела виявилися дуже корисними при імплементації безпечного зберігання паролів у розроблений web-застосунок. Для транспортного рівня використано RFC 6797 [5], який є технічною специфікацією протоколу HTTP Strict Transport Security. У цьому документі описано механізм HSTS, що змушує браузер взаємодіяти з web-сервісом лише через захищене HTTPS-з'єднання. Принцип багаторівневого захисту базується на основі публікації NIST SP 800-160 [6] про кіберстійкі системи. У ній викладено підхід, згідно з яким збій одного з рівнів захисту не повинен завдати шкоди системі в цілому. Без такого підходу до безпеки

користувача соціальний сервіс подібного типу не може вважатися життєздатним.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Об'єктом дослідження є захист інформації у web-застосунку краудсорсингу соціальної підтримки. У межах об'єкта розглядаються способи та процеси збереження та обробки даних користувача, зокрема його персональної інформації, місця знаходження, даних облікового запису та іншої чутливої інформації.

Предметом дослідження є методи та засоби побудови подібного захисту на стеку Python + Django + PostgreSQL. Також важливо відмітити схему, що типова, але не менш важлива у продакшені: де запит спочатку проходить через Nginx, потім через Gunicorn і лише після цього потрапляє в Django. Для перевірки написано 125 pytest-тестів, що швидко проганялись на in-memory SQLite при кожному ком'юніті.

За результатами аналізу для MicroVolunteer застосунку залишаються типові ризики OWASP. Серед них не тільки ін'єкції, XSS, CSRF і контроль доступу, але й більш гострий ризик для соціального сервісу, а ніж звичайній CRUD-системі - деанонімізація через метадані запиту. Якщо на карті видно точну точку місця знаходження, або відома локація де вразлива людина часто проводить час, зловмиснику не треба буде ломати базу даних, бо він вже отримає точні координати підопічного. Для платформи окремо враховано credential stuffing, коли необізнаний користувач повторює дані від входу з іншого сервісу, а атакувальнику залишається перевірити лише вже скомпрометовані пари логін-пароль. У MicroVolunteer це реалізовано за допомогою практики defense-in-depth: якщо один контроль не спрацює, наступний має зупинити чи пом'якшити шкоду. Для успішної атаки, зловмисник має пройти 5 рівнів, а саме: транспортний, рівень HTTP-сесій, автентифікацію, прикладний рівень і рівень даних (рис. 1).

Перший рівень захисту спрацьовує ще до того як запит потрапить до Django. TLS завершується на Nginx, який працює як reverse proxy перед Gunicorn. У Strict-Transport-Security задано max-age=31536000 та includeSubDomains. Це дає вказівку браузеру звертатися до сервісу тільки через HTTPS і лише протягом певного часу [5]. Сесійні cookies і CSRF cookies передаються тільки через захищене з’єднання, із прапорцями Secure та HttpOnly. Також був заданий параметр SECURE_HSTS_SECONDS, щоб застосунок навіть при зміні конфігурації проксі, продовжував віддавати HSTS, захист працює через токен у POST-формі та перевірку заголовка Origin [7]. Вихід із системи можливий лише через POST запит. Тому, якщо спробувати підставити GET запит (наприклад через <img src="/logout/">), завершити сесію користувача не вийде, сайт поверне помилку 405. На цьому ж рівні безпеки додатково посилений захист проти clickjacking через заголовок X-Frame-Options:

DENY. Це блокує вид кібершахрайства, за якого користувача змушують натиснути на прихований елемент на сайті.

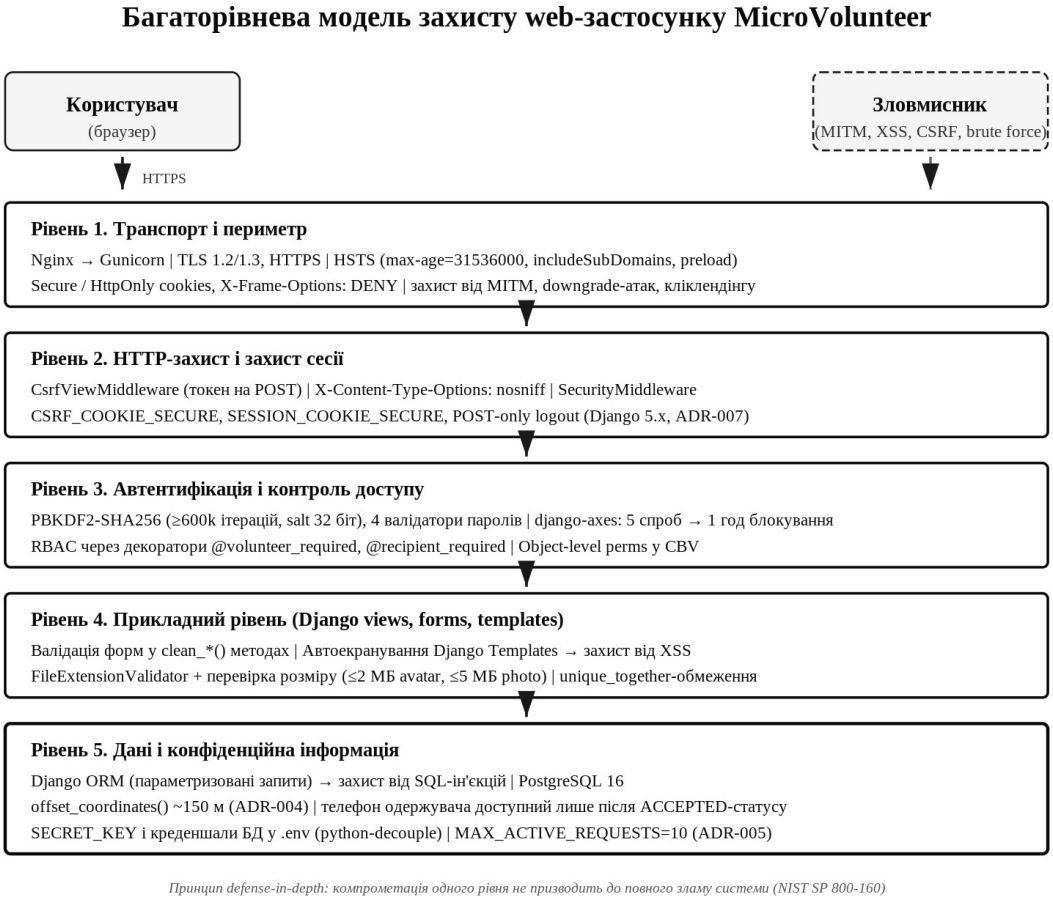

Рис. 1. Багаторівнева модель захисту web-застосунку краудсорсингу соціальної підтримки MicroVolunteer

Наступний рівень безпеки стосується автентифікації та контролю доступу. У базі не зберігаються не зашифровані, відкриті паролі. Саме тому Django зберігає не сам пароль, а результат роботи алгоритму хешування PBKDF2-SHA-256 [8]. А коли користувач повертається та вводить пароль, він знову проганяється через той самий алгоритм і порівнюються з тим хешом, що лежить у базі даних. Для PBKDF2 у MicroVolunteer використано 600 000 ітерацій. Це робиться для того, щоб обчислення хеша відбувалося достатньо повільно для масового підбору, але у той самий час придатним для звичайного користувача. Також для кожного пароля окремо додається сіль перед хешуванням. Вона дозволяє однаковим паролям, мати різний хеш у базі даних. Крім того сіль робить rainbow-таблиці майже непридатними, бо заздалегідь підготовлені таблиці вже не можна напряму застосувати до конкретного запису [3]. Важливо зазначити, що число 600 000 є рекомендацією від OWASP Password Storage Cheat Sheet для алгоритму хешування PBKDF2-SHA-256 [4].

Під час реєстрації кожен пароль проходить через чотири перевірки: несхожість на логін чи email, мінімальна довжина має становити 8 символів, не має міститися у словнику поширених паролів і не є числовою комбінацією без інших типів символів. Цього усе ще мало проти credential stuffing, тому додано django-axes [9]. Після п'яти невдалих спроб входу обліковий запис блокується на годину без можливості входу чи підбору даних для нього. Для користувача це виглядає як невелике обмеження, але для автоматичного перебору воно різко зменшує темп атаки.

У моделі користувача закладено дві ролі - Volunteer та Recipient. Роль визначає не тільки різницю в дизайні меню, а й доступу до дій. Тому для сторінок, що призначені конкретній ролі, використано декоратори @volunteer_required і @recipient_required. У class-based views перевизначено метод get_object(). Це зроблено для контролю доступу на рівні самого об’єкта. Метод не тільки фільтрує запити, а й перевіряє чи належить цей об’єкт до поточного користувача, або чи має він право його бачити. Це важливо для IDOR-сценаріїв, коли первинний ключ запису можна підставити прямо в URL.

XSS ризики зменшуються завдяки автоекрануванням шаблонів Django і security headers [7]. Усі форми проходять перевірку на сервері через метод clean_* і сервіс не покладається на JavaScript валідацію, бо з точки зору безпеки є не достатньою і може бути легко вимкнена на стороні браузера. Тому є лише допоміжною. Для файлів використовується інструмент перевірки FileExtensionValidator і ліміти розміру. Це не тільки слугуватиме рівнем захисту, а й для підтримки стабільності сервісу, через зниження фактичного навантаження на сервер. Дублювання логічно-унікальних запитів закрито на рівні моделі. Наприклад, для пари "волонтер - запит" у відгуках використовується unique_together, тому один і той самий волонтер не може створити декілька однакових відгуків на один і той самий запит.

Окремо варто згадати наступні способи захисту даних. SQL-ін’єкції, блокує сам Django ORM, бо звернення до БД виконуються через параметризовані запити [7], а сирий SQL у проєкті не використовується. Це означає, що дані користувача не підставляються напряму в SQL-рядок як частина команди. Вони передаються окремо як параметри, тому введений користувачем текст не може перетворитися на SQL-команду. SECRET_KEY і облікові дані PostgreSQL, винесені у змінні середовища через python-decouple, тому їх немає в історії git. Це важливо, бо разом опубліковані з репозиторієм секретні ключі, можуть бути використанні зловмисником для отримання доступа до production-конфігурації або підробити частину механізмів Django. Для зменшення спаму встановлено ліміт у 10 активних запитів від одного отримувача. Це обмеження не дає одному користувачу створити необмежену кількість активних заявок і засмітити систему. У соціальному сервісі це важливо, бо волонтери мають

бачити лише актуальні запити, а не дублікати чи навмисно створений шум. Окремо реалізовано захист від географічної деанонімізації. Якщо на публічній карті показати точне місце знаходження отримувача, зловмиснику навіть не потрібно буде ламати базу даних. Саме тому на публічну карту сервіс віддає не точні координати, а лише точку з випадковим офсетом до 150 метрів. Точна адреса відкривається тільки після того, як відгук волонтера переходить у статус ACCEPTED. Тобто це означає, що спочатку волонтер бачить приблизну локацію, щоб оцінити, чи може він допомогти, і після підтвердження відгука отримує точні дані. Телефон отримувача також не показується публічно. Він доступний лише в деталях прийнятої заявки.

Відповідність основних загроз OWASP Top 10:2021 і реалізованих у MicroVolunteer контрзаходів подано у табл. 1.

Таблиця 1

Відповідність загроз OWASP Top 10:2021 і контрзаходів MicroVolunteer

Категорія загрози (OWASP)	**Контрзахід у MicroVolunteer**
A01 Broken Access Control	Декоратори ролей @volunteer_required, @recipient_required; метод get_object(); статус ACCEPTED.
A02 Cryptographic Failures	PBKDF2-SHA-256 (600 000 ітерацій, 256-бітна сіль); HTTPS і HSTS на Nginx; Secure/HttpOnly cookies.
A03 Injection	Django ORM без сирого SQL; серверна валідація форм; FileExtensionValidator.
A04 Insecure Design	Defense-in-depth на 5 рівнях; офсет координат до 150 м; відкладена видача телефону; ліміт 10 активних запитів.
A05 Security Misconfiguration	Поділ settings (base/dev/prod/testing); SECRET_KEY у .env; DEBUG=False у production; X-Frame-Options: DENY; X-Content-Type-Options: nosniff.
A07 Identification and Authentication Failures	Чотири password validators; django-axes (5 спроб і блокування на 1 годину); autocomplete-атрибути у формах.

ВИСНОВКИ

У дослідженні проаналізовані способи захисту інформації і реалізований багаторівневий захист у web-застосунку краудсорсингу соціальної підтримки MicroVolunteer. Модель складається з п'яти рівнів: транспортного, HTTP-сесій, автентифікації та доступу, прикладного рівня і рівня даних. Для кожного рівня підібрано не загальну рекомендацію, а конкретний контроль: HSTS і Secure cookies, CSRF middleware, password

validators, django-axes, рольові декоратори, ORM-параметризацію та обмеження видимості контактних даних. Подальший розвиток доцільно зосередити на тих місцях, де ризик залежить від довіри до користувача. Для верифікованих волонтерів варто додати 2FA. Для адміністративних і контактних дій потрібен журнал аудиту: вхід, зміна профілю, перегляд телефону та відкриття точної адреси.

ДЖЕРЕЛА

1. OWASP Foundation. OWASP Top 10:2021. URL: https://owasp.org/Top10/ (дата звернення: 03.05.2026).

2. OWASP Foundation. Application Security Verification Standard 4.0.3. 2021. URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 03.05.2026).

3. Grassi P. A., Garcia M. E., Fenton J. L. NIST Special Publication 800-63B: Digital Identity Guidelines: Authentication and Lifecycle Management. National Institute of Standards and Technology, 2017. DOI: 10.6028/NIST.SP.800-63b.

4. OWASP Foundation. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 03.05.2026).

5. Hodges J., Jackson C., Barth A. RFC 6797: HTTP Strict Transport Security (HSTS). Internet Engineering Task Force, 2012. URL: https://datatracker.ietf.org/doc/html/rfc6797.

6. Ross R., Pillitteri V., Graubart R., Bodeau D., McQuaid R. NIST Special Publication 800-160 Vol. 2 Rev. 1: Developing Cyber-Resilient Systems: A Systems Security Engineering Approach. National Institute of Standards and Technology, 2021. DOI: 10.6028/NIST.SP.800-160v2r1.

7. Django Software Foundation. Security in Django: official documentation. URL: https://docs.djangoproject.com/en/5.2/topics/security/ (дата звернення: 03.05.2026).

8. Turan M. S., Barker E., Burr W., Chen L. NIST Special Publication 800-132: Recommendation for Password-Based Key Derivation. National Institute of Standards and Technology, 2010. DOI: 10.6028/NIST.SP.800-132.

9. Django Axes: documentation. URL: https://django-axes.readthedocs.io/en/latest/ (дата звернення: 03.05.2026).