ОНЛАЙН-СЕРВІС ДЛЯ ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ БІБЛІОТЕЧНОЇ СИСТЕМИ

Боклач М. А., Негоденко О.В.
Київський столичний університет імені Бориса Грінченка, м. Київ

ВСТУП

Відкритий веб-інтерфейс бібліотеки без нормального захисту характеризується уразливістю, через яку автоматичний сканер витягне список користувачів раніше, ніж адміністратор встигне це помітити. Факт наявності потенційних загроз підтверджено під час практичної експлуатації тестової версії системи. Під час аналізу серверних логів встановлено значну кількість автоматизованих запитів від сторонніх сканерів і ботів. Саме тому необхідно проєктувати архітектуру з урахуванням вимог інформаційної безпеки, оскільки система обробляє персональні дані користувачів.

Актуальність і постановка проблеми. Більшість бібліотечних ILMS-систем мають спільну проблему, яка характеризується тим, що безпека в них додавалась постфактум, а не закладалась в основу, а саме паролі за алгоритму MD5, жодної ролевої моделі та пошукові форми без валідації.

Мета роботи — підвищення рівня безпеки системи управління бібліотекою за допомогою сучасних механізмів захисту. Основними впровадженням передбачено заміна хешування паролів на Argon2id, закриття SQL-векторів через ORM, побудува RBAC із п’ятьма рівнями доступу і захист мережевого периметру через HSTS та CSP.

Актуальність проблеми підтверджується методологією OWASP Top 10:2021 [1], де injection-уразливості, збої автентифікації та зламаний контроль доступу стабільно очолюють рейтинг критичних ризиків. Для бібліотечних систем університетів, де обробляються персональні дані тисяч осіб, це не теоретична статистика, а реальна площина атаки.

Аналіз останніх досліджень і публікацій. У сучасних дослідженнях інформаційної безпеки веб-застосунків значна увага приділяється захисту від типових уразливостей, безпечному зберіганню облікових даних користувачів та реалізації механізмів контролю доступу [2; 3]. Рекомендації організації OWASP, зокрема стандарт OWASP Top 10:2021 [1], визначають ін’єкційні атаки, порушення автентифікації та некоректний контроль доступу як ключові ризики для інформаційних систем. Сучасні рекомендації щодо зберігання облікових даних користувачів передбачають використання спеціалізованих алгоритмів хешування паролів, зокрема Argon2, які забезпечують стійкість до атак перебору та використання попередньо обчислених таблиць. Використання застарілих алгоритмів, таких як MD5, розглядається як критичний недолік систем безпеки [2].

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Django-додаток для управління бібліотекою включає облік книг, реєстрацію читачів, видача та повернення, базова звітність. Функціонально достатній, але з погляду безпеки має недоліки, а саме паролі зберігались через стандартний Django PBKDF2-SHA256, але при GPU-атаці на дамп бази hashcat на RTX 4090 видає до 800 МГеш/с для PBKDF2. При ревізії views.py (рядок 47) виявилось, що search_view формував SQL-запит через f-рядок: f“SELECT * FROM books WHERE title LIKE ‘%{query}%’”. Навіть базовий payload ’ OR 1=1-- спрацьовував у тестах. Спочатку спробували додати regex-фільтрацію поверх цього запиту, але виявилось, що payload типу ’ UNION SELECT username, password FROM auth_user-- обходить будь-який regex. Правильне рішення використати ORM-фільтр Book.objects.filter(title__icontains=query), який виконує параметризацію psycopg2-драйвер на рівні протоколу. В даному додатку відсутністня рольова модель, тому будь-який авторизований користувач міг підставити чужий user_id в URL і отримати чужу читацьку картку. Запит /orders/42/ від іншого акаунту повертав дані без помилки.

Практика в «Сервіо» підтвердила, що реальні інциденти трапляються не через одну уразливість, а через ланцюжок дрібниць, кожна з яких окремо нешкідлива. Єдиним рішенням є зміна підходу цілком за принципом Security by Design, тобто безпека не патч зверху, а частина кожного архітектурного рішення.

Технічна реалізація модулів безпеки. Весь шар роботи з БД дійснено виключно через Django ORM. Щоб унеможливити обхід цього правила, впроваджено заборону на використання raw() SQL через pre-commit hook із плагіном flake8-sql. Так при спробі коміту коду з raw()-запитом CI автоматично завершується помилкою до merge в main. Таким чином Django ORM примусово реалізує Prepared Statements на рівні psycopg2-драйвера. Тому дані ніколи не інтерпретуються як частина SQL-виразу.

Бібліотека django-argon2==23.1.0 інтегрована безпосередньо в модель користувача через перевизначення методу set_password(), а саме при кожному збереженні пароля автоматично викликається хешування Argon2id, а не PBKDF2. Жодних змін у бізнес-логіці не потрібно, оскільки Django сам делегує хешування через PASSWORD_HASHERS у settings.py.

Санітизацію реалізовано через bleach==6.1.0, метод bleach.clean() з порожнім allowed_tags=[]. Початково використовували html.escape() зі стандартної бібліотеки, але виявилось, що він не обробляє конструкцію javascript: в href-атрибутах. Тестування на наборі з 500 payload репозиторію PayloadsAllTheThings показало, що bleach заблокував усі вектори, а html.escape() пропустив три. Django-шаблони екранують змінні автоматично, але для полів із дозволеною розміткою (анотації книг)

застосовано явний whitelist тегів. Використання mark_safe() без обґрунтування заборонено code review checklist.

Аналіз джерел показав, що MD5 зламано ще наприкінці 90-х, тоді як SHA-256 визнано надійною функцією цілісності, але не хешування паролів, оскільки на RTX 4090 вона дає до 800 МГеш/с. Argon2id став переможцем Password Hashing Competition 2015, оскільки має memory-hard і є повільний [4].

Інтеграція django-argon2 до 23.1.0 реалізовано через підключення PASSWORD_HASHERS у settings.py. Алгоритм Argon2id є гібрид, оскільки Argon2i захищає від side-channel атак, але вразливий до GPU, тоді як Argon2d діє навпаки. Версія id закриває обидва вектори одночасно.

Параметри memory_cost заклали 65536 КБ (64 МБ) так, щоб при менших значеннях ASIC-атака залишалася рентабельною, тоді як time_cost=2 отримали стабільні 190–200 мс та при parallelism=2 відповідає кількості vCPU.

Використання солі в 16 байт від os.urandom() перед кожним хешуванням дало зберігання разом із хешем. Два однакових паролі дають різні хеші в БД, що повністю унеможливлює rainbow-table атаки.

Рольову модель реалізовано через Django Groups із кастомними Permission-об’єктами, яка включає п’ять рівнів: гість — анонімний пошук без персональних даних у відповіді; читач — бронювання, особиста історія (@permission_required(’library.view_own_orders’) на кожній в’юсі); бібліотекар - видача, повернення, метадані книг; адміністратор - AuditLog і керування акаунтами читачів та бібліотекарів та superadmin - повний доступ до Security Dashboard, який реалізує кастомний інтерфейс замість стандартного /admin/. Форма створення користувача: логін, email, тимчасовий пароль і вибір ролі через radio-buttons. Всередині закладено Groups.add() із відповідними Permission-об’єктами. Кожна дія (створення, редагування, деактивація) логується в AuditLog із timestamp і IP-адресою Superadmin.

Потрібно відмітити, що у settings.py налаштовано мережеві параметри, а саме SECURE_HSTS_SECONDS=31536000, SECURE_HSTS_INCLUDE_SUBDOMAINS=True,CSP_DEFAULT_SRC=(’self’,), CSP_SCRIPT_SRC=(’self’, ’nonce-{random}’). Nonce генерується middleware на кожен запит, що блокує вбудовані XSS-скрипти навіть при вразливому полі. HSTS після першого відвідування забороняє браузеру підключатися по HTTP упродовж року навіть при підміні DNS.

На рівні Nginx: rate limiting 10 req/s на /api/ та /login/ (відповідь 429 без виклику Django), Fail2Ban банить IP після 20 помилок за 10 хвилин, X-Frame-Options: DENY запобігає Clickjacking. Комплекс заходів відповідає підходу до багаторівневого захисту периметру.

Систему аудиту забезпечує модуль AuditLog, який проєктувався з урахуванням типових сценаріїв інцидентів в освітніх мережах, а саме

несвоєчасне виявлення через відсутність моніторингу та привілейований доступ без аудит-сліду. Фіксувалося два типи подій: штатні (вхід, видача книги, зміна пароля) та аномалії (5+ невдалих спроб за 2 хвилини, звернення до маршруту поза роллю, нетипова кількість API-запитів). При спрацюванні порогу django-axes==6.1.1 автоматично блокує IP.

Експериментальна оцінка ефективності. Параметри Argon2id підбиралися через бенчмарки на університетському VPS, Intel Core i5-10400, 16 ГБ DDR4, SSD при умові, що 10 паралельних логінів одночасно забезпечують пік навантаження на початку занять (таблиця 1).

Конфігурація 64 МБ / 2 ітерації дає оптимальний компроміс. Затримка 194 мс непомітна для людини, але для brute-force-скрипта це 5 спроб/с на один потік. У поєднанні з django-axes (блок після 5 невдалих за 2 хвилини) та Nginx rate limiting (10 req/s) атака стає практично нереалізовною. Конфігурація 128 МБ / 4 ітерації давала 28% завантаження CPU при 10 паралельних логінах створює умови, що сервер не справлявся з іншими запитами.

Таблиця 1

Бенчмарки Argon2id при 10 паралельних запитах автентифікації

Конфігурація	**Час (мс)**	**GPU-стійкість**	**CPU (%)**	**Сфера**
16 МБ / 1 іт.	48	Низька	2	Мобільні сервіси
32 МБ / 2 іт.	108	Середня	5	Загальні ресурси
64 МБ / 2 іт. ✓	**194**	**Висока**	**12**	**Освітні / корп.**
128 МБ / 4 іт.	442	Критична	28	Фінансові

За тиждень тестового моніторингу виявили, що Fail2Ban заблокував 847 запитів на рівні Nginx. До шару автентифікації Django дійшло лише 12, що менше 2% від загальної кількості спроб. Ешелонована оборона підтвердила свою ефективність, оскільки більшість атак відсікається ще до того, як вони торкнуться криптографічного шару.

Тестування валідатора на наборі з 500 payload (PayloadsAllTheThings) показало 100% блокування без хибно-позитивних спрацювань на легітимних даних. Перевірка User Enumeration встановила, що після переходу на єдине повідомлення про помилку автоматизований сканер не зміг скласти список валідних логінів за 1000 спроб.

ВИСНОВКИ

Проведений моніторинг протягом 7 днів виявив, що Fail2Ban заблокував 847 запитів на рівні Nginx, до логіки автентифікації Django дійшло лише 12. Також ешелонована оборона відсікає левову частку атак задовго до того, як вони торкнуться Argon2id. Заборона raw()-запитів через

flake8-sql на рівні CI унеможливлює SQL-ін’єкцію навіть при недбалості розробника. Argon2id із параметрами memory_cost=65536, time_cost=2, parallelism=2 забезпечує баланс між захистом і реальними ресурсами університетського VPS. Встановлено, що Nginx забезпечує першу лінію оборони, тоді як Argon2id надає кінцеву, якщо вже трапився витік бази. Рішення охоплює обидва кінці. Перспективним рішенням надалі є використання TOTP через RFC 6238 (django-otp вже у requirements.txt), та WebAuthn для автентифікації без паролів, оскільки останній усуває вектор credential stuffing, так що приватний ключ не покидає пристрій, а Touch ID є у більшості сучасних ноутбуків студентів.

ДЖЕРЕЛА

1. OWASP Top 10:2021. The Ten Most Critical Web Application Security Risks. URL: https://owasp.org/www-project-top-ten/

2. Пірог О.В. Безпека вебдодатків : навч. посібн. / О.В. Пірог. – Електронні дані. – Житомир : Житомирська політехніка, 2025. – 290 с. http://eztuir.ztu.edu.ua/123456789/8813

3. Боскін О.О., Корніловська Н.В., Поліщук В.М., Сарафаннікова Н.В. (2023). Безпека веб-додатків та хакерські атаки. Вісник Херсонського національного технічного університету, № 3(86), 83-92. https://journals.kntu.kherson.ua/index.php/visnyk_kntu/article/view/452

4. Biryukov A., Dinu D., Khovratovich D. Argon2: the memory-hard function for password hashing and other applications. 2015. URL: https://www.password-hashing.net/argon2-specs.pdf

5. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., Hlushak, O., Zhyltsov, O., Ilich, L., Kobets, N., Kovaliuk, T., Kuchakovska, H., Lytvyn, O., Lytvyn, P., Mashkina, I., Morze, N., Nosenko, T., Proshkin, V., Radchenko, S., & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science. https://doi.org/10.28925/9720213284km