УДК 004.056.54 **DOI:** 10.31673/2412-9070.2024.021417

О. В. БРАТКОВСЬКИЙ, студент;
А. М. ТУШИЧ, доктор філософії (PhD), доцент,
Державний університет інформаційно-комунікаційних технологій, Київ

УДОСКОНАЛЕННЯ ЗАХИСТУ ДАНИХ У ВЕБЗАСТОСУНКАХ ІЗ ВИКОРИСТАННЯМ DJANGO

Розглянуто тему вдосконалення захисту даних у вебзастосунках із використанням фреймворку Django. Визначено основні вразливості вебзастосунків і проаналізовано властивості Django для захисту від атак, зокрема SQL-ін'єкцій, XSS, CSRF, клікджекінгу, перехоплення даних.

Підкреслено ефективність вбудованих механізмів захисту Django проти більшості поширених атак та запропоновано способи поліпшення і посилення заходів безпеки. Наведено детальний аналіз модуля автентифікації Django з огляду на його ключові особливості щодо забезпечення високого рівня надійності у сфері керування доступом користувачів. Описано етапи хешування пароля, а також логіку його оброблення.

Висвітлено важливість використання протоколів HTTPS, SSL та TLS для підвищення рівня безпеки вебзастосунків. Окремо приділено увагу налаштуванню HTTPS та використанню заголовків безпеки, таких як HTTP Strict Transport Security (HSTS), котрі забезпечують захищене з'єднання та запобігають атакам, спрямованим на перехоплення даних.

У процесі аналізу основних видів атак, зокрема SQL-ін'єкцій, XSS, CSRF, увагу зосереджено на вбудованих функціях захисту вебсистеми, можливих уразливостях та запропоновано рекомендації щодо вдосконалення безпеки даних.

Також розглянуто заходи стосовно захисту від клікджекінгу та надано пропозиції з використання проміжного програмного забезпечення для ефективної протидії такому типу атак. Наголошено на потребі уважно контролювати можливість укладання сторінок та обмеженні цього функціонала для гарантування безпеки вебсистеми.

Ключові слова: Django; безпека даних; вебзастосунки; SQL-ін'єкції; XSS; CSRF; автентифікація; HTTPS; SSL; TLS; ORM; клікджекінг.

ВСТУП

Постановка проблеми. У сучасному світі майже все працює завдяки інформаційним системам та технологіям: від онлайн-банкінгу і безготівкових розрахунків до віддаленої роботи, електронної комерції тощо. Не дивно, що через це програми стають основною ціллю для зловмисників, які намагаються виявити потенційно слабкі місця в API, вихідному коді з відкритим доступом, віджетах сторонніх розробників і системах контролю доступу. Така цифрова активність потребує від розробників надійних заходів щодо забезпечення високого рівня безпеки вебзастосунків, збереження довіри користувачів та уникнення серйозних наслідків від втрати чутливої інформації.

Безпека даних — це процес захисту цифрової інформації від несанкціонованого доступу, пошкодження чи спроб крадіжки. Політики безпеки вебзастосунків спрямовано на захист даних за допомогою таких заходів, як брендмауери вебпрограм (WAF), багатофакторна автентифікація (MFA) для користувачів, використання, захист і перевірка файлів cookie для підтримання статусу конфіденційності, а також різноманітні методи перевірки безпеки даних, перш ніж їх обробить програма [1].

Дослідження проблем безпеки даних у вебзастосунках є ключовим завданням для розробників та академічних учених. Результати досліджень дають змогу поліпшувати наявні методи та створювати нові для забезпечення безпеки, сприяючи посиленню можливостей кібербезпеки в мережі. Практичні завдання захисту даних у вебзастосунках потребують реалізації та впровадження сучасних технологій, методів, практик та протоколів для ефективного захисту інформації.

Роль фреймворку Django у створенні безпечних та надійних вебсистем дедалі зростає. Серед поширених атак на вебсистеми використовуються зокрема методи грубої сили, sql- і formjacking-ін'єкції, міжсайтовий скриптинг, перехоплення сесій. На основі останніх досліджень та публікацій у галузі захисту даних вебсистем можна виокремити тенденції, що ґрунтуються на вбудованих можливостях Django та дають змогу забезпечити надійний захист даних навіть у самих простих застосунках. Цей фреймворк уже містить підтримання захисту від основних типів атак, а деякі функції захисту треба лише налаштувати відповідно до потреб.

Метою дослідження є огляд рівня безпеки вебзастосунків на основі Django та способи вдосконалення захисту.

ОСНОВНА ЧАСТИНА

Безпека автентифікації в Django

Django має багатий набір інструментів для забезпечення надійної автентифікації та авторизації на основі фреймворку сеансів, котрий дає змогу перевіряти облікові дані користувача та визнача-

ти, які дії дозволено виконувати кожному користувачеві системи.

Сеанси — це механізм, яким послуговується Django для відстеження стану між сайтом і певним браузером. Сеанси уможливлюють зберігання даних для кожного браузера та отримання цих даних на сайті щоразу, коли браузер здійснює підмкнення. Окремі елементи даних, пов'язаних із сеансом, потім посилаються на ключ, який застосовується для зберігання і отримання даних.

Фреймворк Django використовує файли cookie, які містять спеціальні ідентифікатори сеансів, що дає змогу ідентифікувати кожен браузер і його сеанс, пов'язаний із вебсистемою. Усталене налаштування фактичних даних сеансу зберігаються в базі даних застосунку, що є безпечнішим, ніж збереження даних у файлі cookie, де вони більш уразливі для зловмисників. Зберігання даних сеансів також можна налаштувати в інших місцях (кеш, файли, захищені файли cookie з використанням різних розширень), але усталене розміщення є більш прийнятним і надійнішим варіантом захисту даних.

Система автентифікації користувачів django-allauth уже містить вбудовані моделі для Users та Groups, дозволи/прапорці, які вказують, чи може користувач виконувати завдання, захищені форми та подання для входу користувачів, а також інструменти перегляду для обмеженого вмісту [2].

Фреймворк Django в автоматичному режимі використовує механізм хешування паролів із застосуванням усталеного одноразового ключа PBKDF2 і хешем SHA256, рекомендованим NIST. Це підсистема безпеки даних, котра зберігає паролі користувачів як хеші, що робить їх складними для відновлення і читання, навіть потрапляючи до бази даних. Для переважної більшості користувачів цього достатньо: це безпечно, а для зламування такого пароля знадобиться значний час.

Принцип роботи фреймворку передбачає наведені далі етапи.

1. Створення солі (salt): сіль — це випадковий рядок, що генерується для кожного пароля перед застосуванням хеш-функції.

2. Змішування пароля та солі: звичайний пароль змішується із сіллю. Це гарантує, що однакові паролі користувачів матимуть вигляд випадкових рядків через унікальні солі.

3. Застосування PBKDF2: змішаний пароль та сіль передаються через PBKDF2, алгоритм з ітераціями та внутрішньою криптографічною хеш-функцією, що створює кінцевий хеш. Кількість ітерацій зроблено так, щоб у разі спроби атаки збільшити вартість обчислень.

4. Збереження в базі даних: здобутий хеш, а не оригінальний пароль, зберігається в базі даних.

Наведений підхід до хешування паролів робить їх складними для атак, оскільки навіть знання хешу не дає змоги легко відновити оригінальні паролі чи спробувати авторизуватись у системі.

Під час уведення стандартного пароля для автентифікації хешування виконується знову, отриманий хеш порівнюється зі збереженим у базі даних, якщо вони збігаються, автентифікація вважається успішною.

Захист від типових атак та розширення функцій захисту Django

1. ***Вбудований захист від SQL-ін'єкцій.*** SQL-ін'єкції — це тип атак, коли злочинці намагаються виконати код SQL у базі даних, що в подальшому може призвести до втрати даних або їх витоку.

Django має вбудований захист від такого типу атак завдяки використанню ORM (*Object-Relational Mapping*) для взаємодії з базою даних. ORM здійснює автоматичну генерацію SQL-запитів на основі об'єктів Python із використанням параметризації запитів, що унеможливлює проведення даного типу атаки, адже код SQL визначається окремо від параметрів запиту. Оскільки користувач може ввести небезпечні параметри, вони екрануються базовим драйвером БД, даючи змогу уникати шкідливих ін'єкцій даних та забезпечуючи безпечне оброблення інформації в БД.

2. ***Захист від атак типу Cross-Site Scripting (XSS).*** XSS-атаки дають можливість зловмисникам вбудовувати клієнтські скрипти в браузери інших користувачів. Зазвичай це досягається завдяки зберіганню шкідливих сценаріїв у БД, звідки вони будуть відображені для інших користувачів, або ж через спонукання користувачів перейти за небезпечним посиланням, яке викликає шкідливий JavaScript злочинця в браузері. Django автоматично екранує виведення даних у HTML-шаблонах. Це означає, що будь-які введені користувачем дані до шаблону, автоматично обробляються в такий спосіб, щоб вони не викликали сторонніх скриптів у браузері. Така функція запобігає виконанню зловмисних скриптів, що можуть бути вбудовані в уведені дані у формі. Цей вбудований захист береже користувачів від більшості атак, але не від усіх, тому в процесі розроблення потрібно детальніше вивчати документацію. Наприклад, наведений далі запис немає змоги захистити:

<style class={{ var }}>...</style>. (1)

Якщо var установлено якесь значення, то це може призвести до несанкціонованого виконання зловмисного скрипту, а отже, такі випадки потрібно брати до уваги. Цитування значення атрибута виправить цю проблему [4]:

classl onmouseover=javascript:func(). (2)

3. ***Захист від атак типу CSRF*** (захист від підробки міжсайтових запитів CSRF). Такі атаки в разі успіху дають зловмисникам змогу виконувати дії, використовуючи облікові дані користувача без його відома або згоди. Для атаки спочатку підроблюють html-форму, що відображається на сайті, наприклад форму додавання нового товару, яка містить файл, котрий надсилається всім адміністраторам системи, і пропонує відкрити його. Якщо файл відкриє будь-який адміністратор системи, не важливо з якими привілеями, тоді форму буде надіслано з його обліковими даними і, відповідно, додано до сайту.

Django має вбудований захист від такого типу атак, але для цього його треба активувати, скориставшись тегом шаблону у формі

{% csrf_token %}. (3)

Застосування POST також потрібно для запитів, які можуть змінювати дані чи поповнювати ними базу даних. У разі активованого захисту генерується спеціальний ключ користувача, що відхиляє форми, які не мають поля або містять неправильне його значення. Це додасть злочинцю проблем, адже потрібно буде знайти та ввімкнути ключ CSRF для конкретного користувача. Також не вдасться надіслати шкідливий файл усім користувачам системи, оскільки ключ залежить від браузера.

4. ***Використання протоколу HTTPS***. Для підвищення рівня безпеки вебсистеми завжди рекомендується використовувати протокол HTTPS (*Hypertext Transfer Protocol Secure*), який здатен унеможливити перехоплення облікових даних під час автентифікації користувачем або будь-якої інформації, що передається між клієнтом та сервером завдяки шифруванню трафіку. Протоколи SSL (*Secure Sockets Layer*) або його вдосконалена версія TLS (*Transport Layer Security*) працюють разом із HTTPS і відповідають за забезпечення шифрування, конфіденційність та цілісність даних.

Після ввімкнення захисту HTTPS на своєму сервері потрібно дотримуватись деяких рекомендацій щодо правильного налаштування безпеки:

- установіть SECURE_SSL_REDIRECT на значення True, щоб запити через HTTP перенаправлялись на HTTPS;
- використовуйте HTTP Strict Transport Security (HSTS) — HTTP-заголовок, який повідомляє браузер, що всі майбутні з'єднання із сайтом мають послуговуватися HTTPS. Окрім перенаправлення запитів це гарантує, що з'єднання завжди застосовуватимуть додатковий захист SSL. Для правильного налаштування HSTS потрібно використовувати такі параметри: SECURE_HSTS_SECONDS, SECURE_HSTS_INCLUDE_SUBDOMAINS і SECURE_HSTS_PRELOAD;
- щоб уникнути витоку файлів cookie під час з'єднання через HTTP, потрібно використовувати параметри SESSION_COOKIE_SECURE і CSRF_COOKIE_SECURE зі значенням True. Це дає змогу браузеру відправляти файли cookie тільки через захищене з'єднання HTTPS.

5. ***Захист від клікджекінгу***. Клікджекінг — це тип атаки, коли шкідливий сайт вкладає інший сайт у фрейм, намагаючись обманом змусити користувача виконати ненавмисну дію на цільовому сайті.

Django має захист від такого типу атак у формі і запобігає відображенню сайтів у підтримуваних браузерах. Цей захист можна вимкнути для окремих переглядів або налаштувати відповідно до потреб за допомогою точного значення заголовка X-Frame_Options middleware.

Також наполегливо рекомендується використовувати проміжне програмне забезпечення для будь-якого сайту, якому не потрібно вкладати сторінки у фрейм стороннім сайтам або потрібно дозволити це лише для невеликої частини сайту.

ВИСНОВКИ

У статті було розглянуто основні функції захисту в Django та способи їх удосконалення. Звичайно, можливості для забезпечення безпеки даних підтримуються великою кількістю різних фреймворків, але Django вирізняється вбудованим захистом, що працює на високому рівні з самого початку його використання у вебзастосунку. Завдяки вбудованому захисту та можливості вибору чималої кількості готових бібліотек, налаштування та підтримання функцій безпеки є менш часозатратними порівняно з іншими фреймворками.

Проведений аналіз показує, що Django має високий рівень захисту від SQL-ін'єкцій, XSS, CSRF та інших типів атак. Завдяки використанню ORM, контролю за параметрами запитів та автоматичному екрануванню введених даних у формах є можливість уникати багатьох потенційних загроз безпеці системи.

Використовуючи протокол HTTPS у парі з належними конфігураціями параметрів безпеки, зокрема SECURE_SSL_REDIRECT, захищеним файлом cookie і HTTP Strict Transport Security (HSTS), Django забезпечує надійний захист від потенційних атаках, таких як перехоплення повідомлень, клікджекінг та атака на сесії.

У процесі розроблення важливо брати до уваги індивідуальні деталі та характеристики вебсистеми для ефективного використання, модифікації функцій захисту даних і підвищення рівня безпеки додатковими модулями. Фреймворк Django є хорошим вибором для тих, хто хоче розробити захищений та надійний вебзастосунок.

Список використаної літератури

1. ***Що таке*** *безпека веб-додатків? [Електронний ресурс]. URL:*

https://www.f5.com/glossary/web-application-security.

2. ***Автентифікація*** *користувача та дозволи [Електронний ресурс]. URL:*

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication.

3. ***Керування*** *паролями в Django [Електронний ресурс]. URL:*

https://docs.djangoproject.com/en/4.2/topics/auth/passwords/.

4. ***Безпека*** *в Django [Електронний ресурс]. URL:*

https://docs.djangoproject.com/en/4.2/topics/security/.

5. ***Що таке*** *безпека даних? [Електронний ресурс]. URL:*

https://www.ibm.com/topics/data-security.

6. ***Безпека*** *веб-додатків Django [Електронний ресурс]. URL:*

https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/web_application_security.

O. Bratkovskyi, A. Tushych

ENHANCING DATA PROTECTION IN A WEB APPLICATION USING DJANGO

This article explores the theme of enhancing data protection in web applications using the Django framework. It identifies the primary vulnerabilities of web applications and analyzes Django's features for defense against attacks such as SQL injections, XSS, CSRF, clickjacking, and data interception. Examining Django's built-in protection mechanisms, the article not only emphasizes their effectiveness against most common attacks but also proposes methods to improve and strengthen security measures. A detailed analysis of Django's authentication module is provided, revealing its key features in ensuring a high level of reliability in user access management. The article describes the stages of password hashing and the logic of its processing.

The importance of utilizing HTTPS, SSL, and TLS protocols to enhance the security level of web applications is highlighted. Special attention is given to configuring HTTPS and using security headers, such as HTTP Strict Transport Security (HSTS), which secure connections and prevent data interception attacks.

In the analysis of major attack types, including SQL injections, XSS, CSRF, focus is placed on the built-in protection functions of the web system, potential vulnerabilities, and recommendations for improving data security are presented. Additionally, protective measures against clickjacking are discussed, and recommendations for employing intermediary software to effectively counteract such attacks are provided. Emphasis is placed on the necessity of closely monitoring the possibility of page embedding and restricting this functionality to ensure the security of the web system.

Keywords: Django; data security; web applications; SQL injections; XSS; CSRF; authentication; HTTPS; SSL; TLS; ORM; clickjacking.