Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра математики
Факультету інформатики

Курсова робота
за спеціальністю 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»

Автоматизована система аналізу захищеності Web застосунків

Керівник курсової роботи
ст. викл. Вознюк Я. І.

_____
(підпис)

"____"_____2024 р.

Виконав студент
3-го року навчання спеціальності
122 "Комп'ютерні науки"
Галиць Владислав
Олександрович
"____"_____2024 р

Київ 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри інформатики,
доц., канд.ф.-м.н.
_____С. С. Гороховський

„____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту 3-го курсу факультету інформатики курсу
Галицю Владиславу Олександровичу

ТЕМА Автоматизована система аналізу захищеності Web застосунків

Вихідні дані:

- Досліджено найпоширеніші вразливості Web застосунків

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Календарний план
Анотація
Вступ
1.Огляд основних типів вразливостей
2. Огляд методів захисту від вразливостей
3. Проектування системи аналізу вразливостей
4. Створення системи вразливостей
5. Створення застосунку для користувача
Висновки
Список літератури

Дата видачі „___”_____2024 р. Керівник _____
(підпис)

Завдання отримав_____
(підпис)

Тема: Автоматизована система аналізу захищеності Web-застосунків

Календарний план виконання роботи:

Номер	Назва етапу курсової	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	жовтень	
2.	Огляд технічної літератури за темою роботи.	листопад	
3.	Аналіз методів для виявлення вразливостей.	грудень-січень	
4.	Аналіз типів вразливостей.	лютий	
5.	Вибір інструментів для побудови застосунку.	березень	
6.	Створення модулів для тестування різних типів вразливостей на сервері.	кінець березня	
7.	Створення веб застосунку для взаємодії з користувачем.	квітень	
8.	Текстове оформлення результатів роботи.	квітень	
9.	Попередній аналіз курсової. Виправлення помилок.	травень	
10.	Захист курсової роботи.	травень	

Студент Галиць В. О.

Керівник Вознюк Я. І.

“_____” _____

Зміст

Анотація 5

Вступ 6

1. *Огляд основних веб-вразливостей* 7

   1.1 SQL-ін'єкції 7

   1.2 Приклади SQL-ін'єкції 8

   1.3 XSS атаки 9

   1.3 SSRF вразливість 10

   1.3 HTTP заголовки 11

   1.3 Атрибути Cookie 12

2. *Методи захисту від SQL-ін'єкцій* 13

3. *Методи захисту від XSS-атак* 16

4. *Проектування системи аналізу захищеності* 17

5. *Реалізація системи* 20

Висновки 27

Список літератур 28

Анотація

Метою цієї курсової є дослідження найпоширеніших вразливостей веб-застосунків та розробка системи аналізу захищеності. Курсова робота базується на дослідженні поведінки веб-застосунків при різних типах атак, вивченні інструментів обробки веб-сторінок для подальшого аналізу захищеності шляхом автоматизованого проходження сторінками застосунку та випробування різних типів атак для викривання можливих вразливостей.

Вступ

У сучасному світі ,особливо у нестабільні, кризові періоди, де більшість бізнес-операцій відбуваються онлайн, зростає нагальна потреба на захищеність від кібернетичних атак. Враховуючи , що веб-застосунки грають ключову роль у досягненні комерційного успіху , їх складність та функціонал постійно зростає. Це безпосередньо впливає на постійне збільшення загроз та вразливостей. Виявлення та усунення цих вразливостей є пріоритетною задачею для усіх розробників не тільки для збереження конфіденційності особистої інформації користувачів , а й для уникнення репутаційних ризиків компаній.

Ручне тестування додатків на вразливості є дуже довготривалим та не завжди охоплює усі потенційні загрози. Автоматизована система аналізу захищеності дозволить полегшити розробку додатку та на ранніх етапах розробки виявить усі недоліки з високою точністю та у короткий проміжок часу.

Система виявлення вразливостей відкриває нові можливості у розробці веб-застосунків. Вона не тільки покращує рівень захищеності та сприяє розширенню знань розробників сфері вразливостей , але й дає поради по методам їх усунення та сприянню захищеності свого застосунку. Таким чином, автоматизована система аналізу стає незамінним інструментом у сфері веб-безпеки, забезпечуючи більш стійкі та надійні веб-середовища для всіх користувачів.

1 Огляд основних веб-вразливостей

У сучасному цифровому світі веб-програми стикаються з різноманітними вразливими місцями, які можуть серйозно поставити під загрозу їх безпеку та цілісність. , ін’єкції SQL дозволяють зловмисникам маніпулювати запитами до бази даних, що може призвести до несанкціонованого доступу та втрати конфіденційної інформації. Міжсайтовий виклик (XSS) дозволяє вставляти шкідливі вирізки коду у вміст, який переглядають інші користувачі, з можливістю викрадення облікових даних автентифікації. Проблеми з керуванням файлами «cookie» можуть призвести до втрати важливих даних сеансу, якщо вони не зашифровані чи іншим належним чином захищені. Неправильно налаштовані заголовки безпеки у веб-серверах і програмах можуть відкрити двері для атак типу "людина посередині". Уразливості в механізмах автентифікації та управлінні сеансами можуть дозволити зловмисникам перехоплювати або повторно використовувати сеанси користувачів. Також, атаки DoS або DDoS відбуваються, коли сервери або програми перевантажені зловмисним трафіком, що робить їх недоступними

1.1 SQL-ін’єкції

Ін’єкція SQL відбувається, коли зловмисник вводить зловмисний запит SQL через поле виклику функцій або запитів. Експлойт цього типу може надати доступ до конфіденційних даних, змінювати, додавати або видаляти записи бази даних, виконувати адміністративні операції, такі як зупинка бази даних, а також відновлювати вміст файлів, що зберігаються в системі керування базами даних, а в деяких випадках навіть виконувати команди операційної системи. SQL-ін’єкції особливо поширені в програмах, написаних на PHP і ASP, через широке використання старих інтерфейсів програмування. У той же час додатки на основі ASP.NET менш сприйнятливі до SQL-ін'єкцій через особливості їх програмних

інтерфейсів. Рівень загрози, яку становлять SQL-ін’єкції, залежить від навичок і креативності зловмисника, а також від наявності заходів безпеки, таких як використання обмежених прав доступу

1.2 Приклади SQL-ін’єкції

Для здійснення атаки через SQL-ін'єкцію, нападник спочатку має знайти вразливі точки введення запитів на веб-сторінці. Нападник може створити вміст для введення який є ключовою частиною атаки.Є різні типи SQL атак, в залежності від типу формується запит , який буде передано Цей запит передається від веб-застосунку до сервера та виконується у базі даних.

Розглянемо основні види SQL-ін'єкцій. In-band SQL-ін'єкція є найпоширенішим типом, коли зловмисник використовує той самий канал для посилення зловмисних запитів і отримання результатів. Підтипи In-band SQL-ін'єкція включають SQL-ін’єкції на основі помилок і SQL-ін’єкції на основі об’єднання (на основі команд UNION). SQL-ін’єкція на основі помилок дозволяє зловмиснику отримати інформацію про структуру бази даних за допомогою повідомлень про помилки, створених сервером бази даних. SQL-ін’єкції на основі об’єднання використовує інструкцію UNION для об’єднання результатів двох різних запитів, що дозволяє зберігати дані з іншої бази даних.

Blind або сліпа SQL-ін’єкція виконується, коли сервер бази даних не повертає жодних помилок або результатів запиту, але виконує всі шкідливі команди. Метод використання включає в себе булеві логічні які намагається вивести результат , коли вираз успішний.

Розглянемо практичні приклади з кодом ін’єкції.

SELECT name, age FROM users WHERE id = 1 OR 1=1; --

В цьому прикладі зловмисник може повернути ім’я та вік користувача, за допомогою булевого запиту, оскільки «**1=1**» завжди істина , а «--» коментує решту

коду. Зловмисна вставка у цьому коді «1 OR 1=1; --» , яку ми вставляємо напряму у поле для запитів на веб-сайті для того щоб викликати помилку та у відповіді подивитись на структуру бази даних для подальших атак.

SELECT name, age FROM users WHERE id = 1 UNION SELECT credit_card_number, pin FROM credit_cards;

Вище ми можемо спостерігати за атакою методом об'єднання. Вставка «**1 UNION SELECT credit_card_number, pin FROM credit_cards;**» є спробою повернути користувача по його ідентифікатору , але при цьому об'єднати інші таблиці з запитом , та отримати відповідь.

Ці типи SQL-ін'єкцій демонструють різноманіття методів, які зловмисники можуть використовувати для компрометації баз даних, і підкреслюють важливість надійного захисту веб-додатків від таких атак.

1.3 XSS атаки

Міжсайтові запити (XSS) – це тип ін'єкцій, під час яких шкідливі сценарії впроваджуються на безпечні та надійні веб-сайти. XSS-атаки відбуваються, коли зловмисник використовує веб-програму для надсилання шкідливого коду, як правило, на стороні браузера. Вразливості, які дозволяють цим атакам бути успішними, досить широко поширені та виникають у будь-якому місці, де веб-додаток використовує вхідні дані від користувача в межах вихідних даних, які він генерує, без їх перевірки чи кодування. Браузер не може дізнатися, що коду не можна довіряти, і виконає його. Оскільки він вважає, що сценарій надійшов із надійного джерела, зловмисний запит може отримати доступ до будь-яких файлів cookie, маркерів сеансу чи іншої конфіденційної інформації, яка зберігається браузером і використовується на цьому сайті. Далі розглянемо декілька типів атак.

Відображені атаки – це такі атаки, коли ін'єктований код відображається на веб-сервері, наприклад, у повідомленні про помилку, результатах пошуку або будь-якій іншій відповіді, яка включає частину або всі вхідні дані, надіслані на

сервер як частину запиту. Відображені атаки доставляються жертвам іншим шляхом, наприклад, у повідомленні електронної пошти або на іншому веб-сайті. Коли користувача обманом змушують натиснути зловмисне посилання, надіслати спеціально створену форму або навіть просто перейти на зловмисний сайт, ін'єктований код переходить на вразливий веб-сайт, який відображає атаку назад у браузер користувача. Потім браузер виконує код, оскільки він надійшов із «надійного» джерела. Відображений XSS також іноді називають непостійним

або XSS типу 1 (атака здійснюється через один цикл запиту/відповіді).

Збережені атаки XSS – це ті, коли ін'єктований код постійно зберігається на цільових серверах, наприклад у базі даних, у форумі повідомлень, у журналі відвідувачів, у полі коментарів тощо. Потім жертва отримує шкідливий код із сервера. Збережений XSS також іноді називають постійним XSS або XSS типу II.

Сліпий XSS – є формою постійного XSS. Зазвичай це відбувається, коли корисне навантаження зловмисника зберігається на сервері та повертається жертві з серверної програми. Наприклад, у формах зворотного зв'язку зловмисник може надіслати зловмисне корисне навантаження за допомогою форми, і коли серверний користувач/адміністратор програми відкриє надіслану форму зловмисника через серверну програму, надісланий код зловмисника буде виконано.

1.4 Приклади XSS атаки

Розглянемо декілька прикладів застосування XSS атак у веб застосунках. Перший приклад – проста атака шляхом впровадження коду у поля , які відображаються на сайті , наприклад коментарі. У фрагменті:

«<script>alert('1');</script>»,

бачимо , що при відображенні коментаря з цим наповненням , браузер запустить команду «alert», яка відкриє вікно сповіщення. Така атака називається «збереженою». У випадку , якщо ми створимо посилання:

«**http://example.com/search?query=<script>alert('1');</script>»,**

результат буде аналогічним , але так, як це зловмисне посилання зазвичай може бути отримано від третіх осіб, цей тип атаки називається «відображеною».

1.5 SSRF вразливість

SSRF (Server-Side Request Forgery) є типом кібератаки, де зловмисник може змусити сервер виконувати запити до інших систем або файлів . Це часто призводить до викриття конфіденційної інформацію користувачів , або налаштувань серверу. Зазвичай , цей тип атаки також є ін'єкційним , тобто посилання на сховані ресурси вкладається у запит до серверу. Успішна атака може мати серйозні наслідки , так як часто йде звернення до файлів які містять ключі доступу до серверу чи бази даних. Одним з прикладів є звернення до «**127.0.0.1**» , це зарезервована адреса яка вказує на адаптер петлі , або більш відоме як «localhost». Особливо вразливими є застосунки в яких є багато посилань на внутрішні ресурси, наприклад онлайн магазин, де кожен товар у списку є посиланням на цей товар. При вставці посилання на «**http::/localhost/admin**» вразливий застосунок перейде за цим посиланням. У деяких випадках такі випадки можливі на ресурси з статичною IP адресою, наприклад «**http://192.168.0.68/admin**» має шанс повернути файл з ключами доступу до захищених ресурсів.

1.6 HTTP заголовки

Для забезпечення захищеності застосунку , один з найважливіших факторів це правильне налаштування заголовків при здійсненні запитів. Було розглянуто наступні заголовки:

«Content-security-policy» – Цей заголовок обмежує скрипти які браузер буде завантажувати та виконувати. Запобігає SQL ін'єкціям та XSS атакам. Приклад вірного використання:

«**Content-Security-Policy: default-src 'self'; script-src 'self' https://trustedscripts.example.com; object-src 'none'; frame-ancestors 'none';**

base-uri 'self';»

«X-Frame-Options» – цей заголовок запобігає доступу до веб-застосунку у фреймах інших сайтів , що блокує атаки типу clickjacking. Приклад вірного застосування:

«**X-Frame-Options: DENY**»

«X-Content-Type-Options» – запобігає MIME-sniffing, атакам які намагаються вгадати MIME тип відповіді з сервера. Вірне використання:

«**X-Content-Type-Options: nosniff**»

«Strict-Transport-Security» – Цей заголовок змушує браузери використовувати захищене HTTPS з'єднання на певний термін для усіх під-доменів. Приклад використання:

«**Strict-Transport-Security: max-age=31536000; include subdomains; preload**»

«Referrer-Policy» – цей заголовок контролює яка інформація відправляється між посиланнями , нижче наведено приклад , в якому заборонено робити запити, у випадку якщо зменшена безпека з'єднання з HTTPS до HTTP:

«**Referrer-Policy: no-referrer-when-downgrade**»

Набір цих заголовків покращують захищеність веб-застосунку , але не у всіх випадках є сенс їх використовувати.

1.7 Атрибути Cookie

Атрибути Cookie відіграють значну роль у захищеності веб-застосунків. Cookie – це дані , які браузер зберігає на стороні клієнта. Найпоширеніше їх використання це ідентифікація користувача , або сеансу. У cookie є атрибути , які надають вказівки по їх використанню, вони були розроблені тому, що cookie є одним з найбажаніших речей для зловмисників , адже вони часто зберігають інформацію про користувача, наприклад сесію , яку можна використати для входу на ресурс та отримати конфіденційні дані. Розглянемо основні :

«Secure» - атрибут , який вказує , що cookie має передаватись тільки по захищеному

з’єднанню HTTPS, що запобігає перехопленню по незахищеним протоколам.

«HttpOnly» – атрибут , який робить cookie недоступним для внутрішніх скриптів. Цей атрибут є особливо корисним при запобіганні XSS атак.

«SameSite» – атрибут , який вирішує чи можливо надсилати cookie міжсайтовими запитами. У цього атрибута є три значення : «Strict», «Lax», «None», розглянемо їх.

Strict – визначає , що інформація cookie буде передаватись тільки за запитами поточного сайту. Це значення особливо рекомендується для дій, які виконуються в одному домені. Однак це може мати деякі обмеження, оскільки деякі системи керування сеансами негативно впливають на навігацію користувача. Оскільки веб-переглядач не буде надсилати файли cookie на будь-які запити, згенеровані стороннім доменом або електронною поштою, користувачеві потрібно буде повторно ввійти, навіть якщо він уже має автентифікований сеанс.

Lax – атрибут схожий на попередній , файл буде надісланий в середені домену, але якщо запит надійшов з стороннього ресурсу.

None – cookie не захищена та буде надсилатись за будь якою адресою.

«Path» – цей атрибут вказує на шлях , який має бути у посиланні за яким надсилається cookie.

«Expires» – визначає дату закінчення дії cookie. По закінченню терміну , cookie буде видалено . У випадку , якщо атрибут не встановлений , cookie буде вважатись тимчасовим , та буде видалено при закритті браузера.

2 Методи захисту від SQL-ін’єкцій

Захист від SQL-ін’єкцій є одним з найважливіших аспектів кібербезпеки. Розглянемо випадок , коли запит до бази даних виконується на основі даних, які надішли від користувача. Цей метод ще називають методом «підготовлених запитів»

Коли розробників навчають, як писати запити до бази даних, їм слід сказати, що вони повинні використовувати підготовлені оператори зі зв’язуванням змінних (так звані параметризовані запити). Підготовлені оператори легко написати та легше зрозуміти, ніж динамічні запити та параметризовані запити, які змушують розробника спочатку визначати весь код SQL, а потім передавати кожен параметр у запит. Якщо в запитах до бази даних використовується цей стиль кодування, база даних завжди розрізнятиме код і дані, незалежно від того, який введений користувачем надано. Крім того, підготовлені оператори гарантують, що зловмисник не зможе змінити мету запиту, навіть якщо зловмисник вставляє команди SQL.

Наступний спосіб захисту – «збережені процедури». Хоча збережені процедури не завжди захищені від впровадження SQL, розробники можуть використовувати певні стандартні конструкції програмування збережених процедур. Цей підхід має той самий ефект, що й використання параметризованих запитів, якщо збережені процедури реалізовані безпечно (що є нормою для більшості мов збережених процедур). Безпечний підхід до збережених процедур Якщо потрібні збережені процедури, найбезпечніший підхід до їх використання вимагає від розробника створювати оператори SQL з параметрами, які параметризуються автоматично, якщо розробник не робить щось, що значно виходить за межі норми. Різниця між підготовленими операторами та збереженими процедурами полягає в тому, що код SQL для збереженої процедури визначається та зберігається в самій базі даних, а потім викликається з програми. Оскільки підготовлені оператори та безпечні збережені процедури однаково ефективні для запобігання впровадження SQL, ваша організація повинна вибрати підхід, який є найбільш доцільним для вас.

Вище розглянуті методи захисту від SQL-ін’єкцій можуть бути застосовані

лише на серверній частині застосунку та є основними та обов’язковими. На стороні клієнта теж є можливості для захисту , але вони менш надійні, так як їх легко обійти. Вони можуть бути корисні у випадках , коли працює автоматизована система знаходження вразливостей , яка працює з полями вводу. Далі розглянемо ці методи захисту.

Валідація вхідних даних на клієнті – хоча цей метод не можна вважати надійним захистом від SQL-ін'єкцій, але він допомагає зменшити кількість неправильних або шкідливих даних, які надсилаються на сервер. Використовуйте регулярні вирази для перевірки введення даних, видаляючи символи, які часто використовуються в SQL-ін'єкціях, таких як «' " `», «;» та інші.

Використання сучасних фреймворків – сучасні JavaScript фреймворки та бібліотеки, такі як React, Angular або Vue.js, мають вбудовані засоби для захисту від різних веб-вразливостей, включаючи XSS, які індиректно допомагають запобігти SQL-ін'єкціям, оскільки зменшують ризик шкідливого введення.

Наведені вище методи особливо корисні , коли веб-сайт сканується автоматизованою системою пошуку елементів веб-сторінок (crawler).

3 Методи захисту від XSS атак

Перший та більш сучасний підхід для захисту від XSS атак – використання сучасних інструментів для розробки веб-застосунків. На щастя, додатки, створені за допомогою сучасних веб-фреймворків, мають менше помилок XSS, оскільки ці фреймворки спрямовують розробників до ефективних практик безпеки та допомагають пом’якшити XSS за допомогою шаблонів, автоматичного екранування тощо.

Щоб XSS-атака була успішною, зловмисник повинен мати можливість вставити та запустити шкідливий скрипт на веб-сторінці. Таким чином, усі змінні у веб-додатку повинні бути захищені. Гарантія того, що всі змінні проходять валідацію, а потім відхиляються або дезінфікуються, відома як ідеальна стійкість до ін’єкції. Будь-яка змінна, яка не проходить цей процес, є потенційною слабкістю, особливо змінні , які далі будуть відправлені на сервер.

Отже , одним з найбільш надійних способів захисту є валідація та перетворення вхідних даних або вихідного кодування для запобігання вразливостей. Метою вихідного кодування (що стосується міжсайтового сценарію) є перетворення ненадійних вхідних даних у безпечну форму, де вхідні дані відображаються користувачеві як дані без виконання як код у браузері. Далі наведемо декілька прикладів такого кодування:

- Перетворення **&** на **&**, перетворення **<** на **<**, , перетворити **'** на **'**
- Кодуйте всі символи за допомогою формату кодування Unicode \uXXXX, де XXXX представляє шістнадцяткову кодову точку Unicode. Наприклад, A стає **\u0041**. Усі буквено-цифрові символи (літери від A до Z, a до z і цифри від 0 до 9) залишаються незакодованими.

4 Проектування системи аналізу захищеності

У цьому розділі ми розглянемо інструменти , які були використані при розробці системи аналізу захищеності веб-застосунків та обгрунтуємо їх вибір.

Основними критеріями вибору стали: здатність до глибокої інтеграції з різними веб-технологіями, гнучкість конфігурації, висока швидкість обробки даних та відкритий вихідний код. Мову програмування було обрано Python , адже вона є дуже гнучкою у виборі бібліотек та інших інструментів. Нижче наведено перелік основних інструментів для створення REST API та внутрішньої логіки застосунку та обгрунтування їх вибору.

```
1   fastapi~=0.111.0
2   uvicorn~=0.29.0
3   requests~=2.31.0
4   beautifulsoup4~=4.12.3
5   webdriver-manager
6   selenium~=4.20.0
7   pydantic~=2.7.1
8   cryptography~=42.0.7
9   PySocks
10  MarkupSafe
11  pyyaml
12  aiohttp
13  aiosignal
14  pyopenssl
15  httpx
```

Рисунок 1.1

З наведеного зображення , бачимо що Pydantic використовується для валідації даних та налаштувань з використанням Python type hints, FastAPI - для швидкого розгортання REST API з автоматичною генерацією документації (Swagger UI). Ці

інструменти забезпечують швидке розгортання, високу продуктивність та легкість інтеграції.

Для взаємодії з веб-застосунок був використаний Selenium Web Driver , так як він є одним з найпопулярніших інструментів для автоматизації веб-браузерів. Це компонент у наборі інструментів Selenium, який дозволяє розробникам створювати скрипти, що імітують взаємодію користувача з веб-сторінкою на рівні оперативної системи, що забезпечує можливість взаємодії з різними веб-елементами, такими як кнопки, поля вводу, переключення між сторінками та інше. Selenium WebDriver взаємодіє прямо з веб-браузером на більш низькому рівні, що дозволяє більш точно імітувати поведінку користувача. Він використовує спеціальні драйвери для кожного типу браузерів, таких як ChromeDriver для Google Chrome, GeckoDriver для Firefox та інші, що спілкуються з браузерами через відповідний API. Цей вибір був критичним , так як для реалізації точного автоматичного збору інформації з різних веб-сторінок, включаючи посилання, форми та інші елементи, які можуть бути потенційно уразливими для атак він підходить краще за будь-які інші рішення. Також , Selenium дуже добре себе показав при тестуванні SQL-ін’єкцій. Отже , Використання Selenium WebDriver для цих цілей забезпечує глибокий та детальний аналіз захищеності веб-додатків, дозволяючи розробникам ідентифікувати та усувати вразливості на ранній стадії розробки.

Requests – це бібліотека Python, яка дозволяє легко виконувати HTTP-запити. Вона забезпечує інтуїтивно зрозумілий інтерфейс для надсилання запитів та обробки відповідей від веб-серверів. Це одна з найпопулярніших бібліотек для роботи з HTTP завдяки своїй простоті використання і гнучкості. Requests працює, надсилаючи HTTP-запити через високорівневі функції, такі як **requests.get**, **requests.post**, та інші. Ці функції повертають об'єкт відповіді, який заголовки відповіді та тіло відповіді. Бібліотека також підтримує сесії, що дозволяє підтримувати стан між запитами, використовуючи ті ж cookies та заголовки.

BeautifulSoup — це бібліотека Python, яка дозволяє парсити HTML або XML, надаючи прості інтерфейси для доступу та зміни даних. Бібліотека широко

використовується для веб-скрапінгу, оскільки дозволяє легко витягувати потрібну інформацію з веб-сторінок. BeautifulSoup працює шляхом конвертації комплексного HTML документа в дерево об'єктів Python, з якими можна легко взаємодіяти. Користувачі можуть шукати теги за назвою, атрибутами, навігаційними зв'язками та іншими параметрами. Парсери, такі як **html.parser**, **lxml**, чи **html5lib**, можуть бути використані з BeautifulSoup для різних потреб та оптимізації продуктивності. Використання BeautifulSoup разом з іншими інструментами, такими як Requests і Selenium, створює потужний набір інструментів для аналізу та посилення безпеки веб-додатків, що дозволяє розробникам проводити глибоке тестування вразливостей та оптимізувати захист своїх ресурсів.

5 Реалізація системи аналізу захищеності веб-застосунків

Реалізація системи аналізу захищеності веб-застосунків націлена на забезпечення глибокого аналізу веб-додатків щодо потенційних вразливостей та слабких місць, які можуть бути використані зловмисниками. Основна мета системи полягає у виявленні, оцінці та звітуванні про такі вразливості як SQL ін'єкції, міжсайтовий скриптинг (XSS), ненадійне управління сесіями та некоректні налаштування безпеки заголовків, що мають критичне значення для забезпечення захисту інформації та даних користувачів. Одна з переваг створеної системи це простота використання тому ,що все що потрібно надати користувачеві це посилання на веб-застосунок, а далі система його просканує та надасть результат. Така система не тільки збільшує загальний рівень безпеки організації, але й сприяє оптимізації процесу розробки та впровадження веб-додатків, роблячи їх стійкішими до потенційних кібератак.

Розглянемо побудову застосунку. На рисунку 1.2 можемо побачити , що серверна частина представляє собою простий REST API з однією кінцевою точкою.

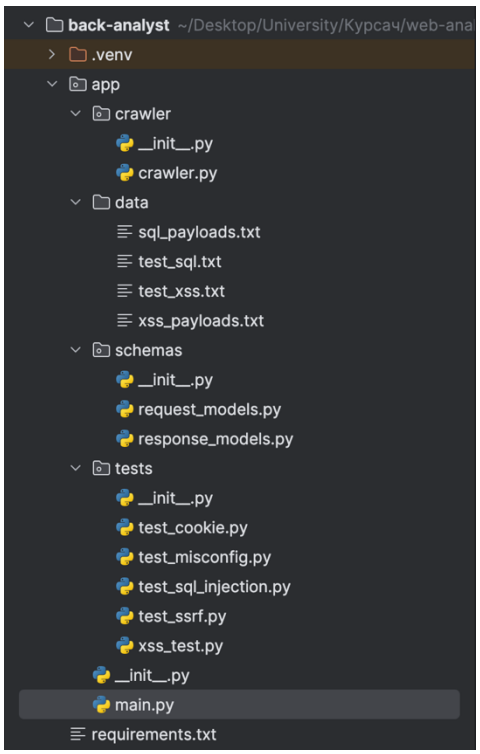

Рисунок 2.1

Вибір такої простої структури зумовлений тим , що операції , які виконує система є суттєво довготривалими , тому що загалом потрібно протестувати по шістсот п’ятдесят різних ін’єкцій на кожне посилання, яких може бути більше двадцяти. У директорії «schemas» в нас схеми запитів , щоб не виникло ситуацію , коли на вхід отримуємо невірні дані.

Основний механізм пошуку пов’язаних посилань реалізований у директорії «crawler» (рисунок 2.2).

```
def crawl_page(driver, url: str, depth: int, visited: Set[str], base_domain: str = "") -> Set[str]:
    """Recursively crawl pages starting from the URL up to the given depth."""
    if depth < 0 or url in visited:
        return set()

    visited.add(url)
    driver.get(url)
    try:
        WebDriverWait(driver, timeout: 10).until(EC.presence_of_element_located((By.TAG_NAME, 'a')))
        elements = driver.find_elements(By.TAG_NAME, 'a')
        links = {element.get_attribute('href') for element in elements if element.get_attribute('href') is not None}

        if not base_domain:
            base_domain = get_domain(url)

        same_domain_links = {link for link in links if link and get_domain(link) == base_domain}

        all_links = set(same_domain_links)
        if depth > 0:
            for link in list(all_links):
                if link not in visited:
                    all_links.update(crawl_page(driver, link, depth - 1, visited, base_domain))
    except Exception as e:
        logging.error(f"Error while processing {url}: {e}")
        return set()

    return all_links
```

Рисунок 2.2

Компонент краулера розроблений для автоматизованого обходу веб-сторінок з метою збору посилань на них. Він використовує бібліотеку Selenium, яка дозволяє керувати браузером Google Chrome в безголовому режимі (headless), тобто без графічного інтерфейсу, для взаємодії з веб-сторінками. Краулер ініціалізується з визначеними параметрами браузера, які запобігають завантаженню файлів і вимикають деякі необхідні налаштування для підвищення безпеки та продуктивності, такі як відключення 'sandbox' і 'shm-usage'. Це забезпечує оптимізацію швидкості і надійності краулера.

Функція crawl_page рекурсивно обходить кожен URL починаючи з заданої точки входу і до заданої глибини. Вона використовує метод **WebDriver.get()** для завантаження сторінки, а потім збирає всі URL, які є у тегах **<a>** на сторінці. Щоб зосередити збір данних лише в межах початкового домену, використовується функція get_domain, яка витягує домен із URL. Такий підхід дозволяє уникнути збору зовнішніх посилань і зосередити зусилля на аналізі внутрішніх посилань веб-сайту.

Під час обходу сторінок краулер також обробляє виключення, що дозволяє йому продовжувати роботу навіть у разі виникнення помилок на певних сторінках. Це забезпечує стабільність та надійність процесу збору даних.

Наступний модуль застосунку – тестування SQL-ін’єкціями.

```
1 usage
def sql_injection_scan(self, url):
    forms = self.get_forms(url)
    logging.info(f"[+] Detected {len(forms)} forms on {url}.")

    for form in forms:
        details = self.form_details(form)
        for payload in self.payloads:
            data = {input_tag['name']: input_tag['value'] + payload for input_tag in details['inputs'] if
                    input_tag['type'] != 'submit'}
            try:
                if details['method'] == "post":
                    res = self.s.post(urljoin(url, details['action']), data=data)
                else:
                    res = self.s.get(urljoin(url, details['action']), params=data)
                if self.vulnerable(res):
                    logging.info(
                        f"SQL injection vulnerability detected in form at {urljoin(url, details['action'])}")
                    return [urljoin(url, details['action'])]
            except Exception as e:
                logging.error(f"Failed to test payload on {urljoin(url, details['action'])}: {e}")

    return []
```

Рисунок 2.3

Модуль SQL-ін'єкцій призначений для автоматизованого аналізу веб-форм на вразливості до SQL-ін'єкцій. Цей модуль використовує бібліотеку requests для надсилання HTTP-запитів та BeautifulSoup з бібліотеки bs4 для парсингу HTML, що дозволяє використовувати дані з форм.

Основна функція **sql_injection_scan** перебирає всі форми, знайдені на сторінці, і для кожного поля вводу вставляє рядок, який може виконати SQL-ін'єкцію, використовуючи заздалегідь визначені вирази, завантажені з текстового файлу. За кожним надсиланням форми система перевіряє відповідь сервера на наявність повідомлень про помилки баз даних, що можуть вказувати на успішну SQL-ін'єкцію.

Модуль **XSSScanner** (рис. 2.4) розроблено для автоматизованого виявлення уразливостей типу міжсайтового скриптингу (XSS) у веб-формах. Він використовує бібліотеки requests для відправлення HTTP-запитів та BeautifulSoup для аналізу HTML-вмісту сторінок. Метод **check_xss_form** для кожної форми на сторінці збирає інформація про дію, метод відправки. Модуль формує дані, вставляючи заготовлені вирази в поля вводу і виконує запит до сервера. Відповіді сервера аналізуються на предмет наявності виразу у відповіді, що свідчить про успішну XSS-атаку.

```
1 usage
def check_xss_form(self, url, form_details):
    """Check if a form is vulnerable to XSS using loaded payloads."""
    logging.info(f"Starting to check payloads on url: {url} for form action {form_details['action']}")
    for payload in self.payloads:
        data = {input_tag['name']: payload if input_tag['type'] == 'text' else input_tag['value']
                for input_tag in form_details['inputs']}
        full_url = urljoin(url, form_details['action'])
        if not full_url.startswith(('http://', 'https://')):
            logging.warning(f"Skipping non-http URL: {full_url}")
            continue
        try:
            response = requests.post(full_url, data=data) if form_details['method'] == 'post' else requests.get(
                full_url, params=data)
            if payload in response.text:
                logging.info(f"XSS vulnerability found with payload {payload} in form action {full_url}")
                return [payload]
        except requests.exceptions.RequestException as e:
            logging.error(f"Failed to send request to {full_url}: {e}")
    logging.info(f"No XSS vulnerabilities found for form action {full_url}")
    return []
```

Рисунок 2.4

Клас **SSRFTester** (рис. 2.5)призначений для автоматизованого виявлення вразливостей типу Server-Side Request Forgery (SSRF) на веб-ресурсах. SSRF ризиком , що дозволяє зловмиснику відправляти запити від сервера до внутрішніх систем, що може призвести до незапланованого доступу до даних або сервісів в межах внутрішньої мережі.

Метод **test_ssrf** проводить ітерацію по всіх переданих посилань та додає до кожного з них тестові вирази. Для кожної комбінації посилання і виразу відправляється HTTP-запит. Якщо сервер відповідає з статусом, відмінним від 404, вважається, що можлива SSRF-вразливість.

```
3 usages
class SSRFTester:
    def __init__(self, urls):
        self.urls = urls
        self.test_payloads = [
            "http://127.0.0.1/admin",
            "http://localhost/",
            "file:///etc/passwd",
        ]
        logger.info( msg: "Initialized SSRFTester with URLs: %s",  *args: urls)

    1 usage
    def test_ssrf(self):
        vulnerable_urls = []
        for url in self.urls:
            for payload in self.test_payloads:
                test_url = f"{url}{payload}"
                try:
                    response = requests.get(test_url, timeout=3)
                    if response.status_code != 404:
                        vulnerable_urls.append(test_url)
                        logger.info( msg: "Potential SSRF vulnerability found: %s",  *args: test_url)
                except requests.exceptions.RequestException as e:
                    logger.error( msg: "Error testing URL %s: %s",  *args: test_url, str(e))
        return vulnerable_urls
```

Рисунок 2.5

Клас **SecurityMisconfigTester** (рис. 2.6) розроблено для аналізу веб-запитів на наявність ключових заголовків безпеки, які допомагають захистити веб-додатки від різних атак, таких як міжсайтовий скриптинг та clickjacking. Цей клас використовує **requests** для відправлення HTTP-запитів та аналізу відповідей. Його основна задача — перевірка HTTP-відповідей на наявність важливих заголовків безпеки, таких як **Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, Strict-**

Transport-Security, X-XSS-Protection та Referrer-Policy. Якщо ці заголовки відсутні, це може вказувати на конфігураційні проблеми, які потенційно збільшують ризик безпеки додатків.

```
1 usage
def check_security_headers(self):
    """Check for the presence of important security headers."""
    headers = self.fetch_headers()
    if isinstance(headers, dict) and 'error' in headers:
        return headers  # Return error if failed to fetch headers

    required_headers = {
        'Content-Security-Policy': 'Helps prevent XSS attacks and data injection.',
        'X-Frame-Options': 'Protects against clickjacking.',
        'X-Content-Type-Options': 'Prevents MIME-sniffing.',
        'Strict-Transport-Security': 'Enforces secure (HTTP over SSL/TLS) connections to the server.',
        'X-XSS-Protection': 'Enables XSS filtering (better to use CSP).',
        'Referrer-Policy': 'Controls information that is set in the Referer header.'
    }

    missing_headers = {header: desc for header, desc in required_headers.items() if header not in headers}
    return {
        "missing_headers": missing_headers,
        "headers_found": {header: headers.get(header) for header in required_headers if header in headers}
    }
```

Рисунок 2.6

Клас **TestCookie** (рис. 2.7) служить для оцінки безпеки куків, що встановлюються веб-додатками. Використовуючи бібліотеку requests, клас отримує cookie з веб-сторінки і аналізує їх налаштування безпеки, такі як атрибути Secure, HttpOnly та SameSite. Ці атрибути важливі для захисту конфіденційної інформації від різних типів атак, включаючи XSS атаки та викрадення сесії. Зокрема, наявність атрибута Secure вказує на те, що cookie можуть передаватися тільки через безпечні HTTPS з'єднання, тоді як HttpOnly робить їх недоступними для скриптів на стороні клієнта, зменшуючи ризики XSS-атак.

```python
class TestCookie:
    def __init__(self, url):
        self.response = requests.get(url)
        self.url = url

    2 usages (1 dynamic)
    def check_cookie_security(self):
        """ Check the security settings of cookies from a given URL. """
        try:
            cookies = self.response.cookies
            report = {}
            for cookie in cookies:
                cookie_details = {
                    "Secure": cookie.secure,
                    "HttpOnly": cookie.has_nonstandard_attr('HttpOnly'),
                    "SameSite": cookie.get_nonstandard_attr('SameSite', 'Not set'),
                    "Path": cookie.path,
                    "Domain": cookie.domain,
                    "Expires": cookie.expires
                }
                report[cookie.name] = cookie_details

            return report
        except requests.RequestException as e:
            print(f"Failed to fetch cookies from {self.url}: {e}")
            return None
```

Рисунок 2.7

Клієнтська частина застосунку (рис. 3) була створена за допомогою бібліотеки React та Axios для виконання запитів. Це примітивний веб застосунок , який виконує всього дві функції : надсилання запиту , та відображення відповіді з сервера. Ця простота була зумовлена первинною ціллю зробити простий у використанні застосунок , який полегшить життя розробникам веб-застосунків.

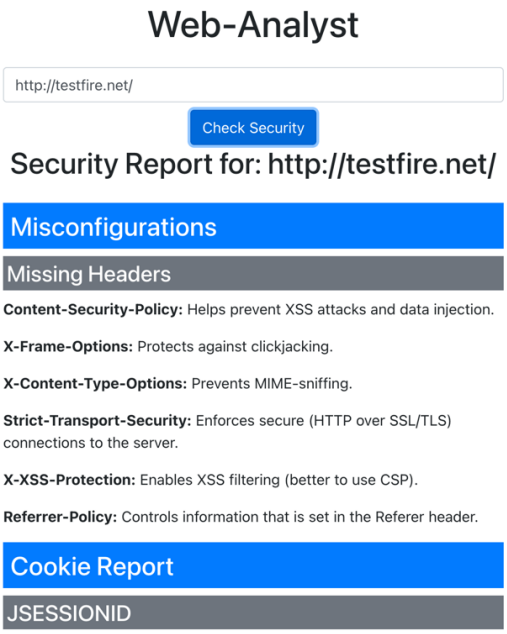

Рисунок 3

Висновки

У рамках даної курсової роботи було досліджено важливе питання захищеності веб-застосунків, можемо з впевненістю сказати , що зараз це одне з найважливіших питань в індустрії, адже забезпечення цілісності та безпеки особистих даних є критичним , особливо у часи війни. Завдяки комплексному підходу була розроблена система автоматизації пошуку вразливостей у веб-застосунках , яка може бути використана при розробці додатків , щоб упереджено захистити дані ще до того як їх намагатимуться здобути зловмисними діями.

Отже , захист від SQL-ін’єкцій , XSS атак , безпека cookie та інші покращення безпеки є невід’ємною частиною сучасної розробки будь-яких систем та застосунків у сучасному світі не тільки для збереження конфіденційності , але й для підвищення довіри користувачів.

Список використаної літератури

1. https://github.com/BekBrace/sqli-scanner/blob/main/scan.py
2. https://denizhalil.com/2024/03/06/detecting-xss-vulnerabilities-with-python/
3. https://www.geeksforgeeks.org/how-to-build-a-sql-injection-scanner-in-python/
4. https://owasp.org/www-community/attacks/xss/
5. https://owasp.org/www-community/attacks/SQL_Injection
6. https://owasp.org/www-project-top-ten/
7. https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/06-Session_Management_Testing/02-Testing_for_Cookies_Attributes
8. https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
9. https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html
10. https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master
11. https://www.reflectiz.com/blog/owasp-top-ten-2023/
12. https://github.com/commixproject/commix/wiki/Usage-Examples