Хмельницький національний університет
Факультет інформаційних технологій
Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

Бабаєвського Віталія Михайловича

на здобуття ступеня вищої освіти Бакалавра

Система виявлення атак
за допомогою автоматичного моніторингу вебсервера

Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітня програма Кібербезпека

Шифр КРБКБ.200101.20.01.01 ПЗ

Виконав студент 4 курсу група КБ-20-1 Віталій БАБАЄВСЬКИЙ
Керівник канд. техн. наук, ст. викладач Ігор МУЛЯР
Нормоконтролер старший викладач Сергій МОСТОВИЙ

До захисту допускаю:
Завідувач кафедри кібербезпеки Юрій КЛЬОЦ

19 06 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Кібербезпеки

Рівень вищої освіти Бакалавр

Галузь знань 12 – Інформаційні технології

Спеціальність 125 – Кібербезпека

Освітня програма Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри кібербезпеки

Юрій КЛЬОЦ

15 лютого 2024 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бабаєвському Віталію Михайловичу

1 Тема роботи Система виявлення атак за допомогою автоматичного моніторингу вебсервера

Керівник роботи канд. техн. наук, ст. викладач Муляр І. В.

Затверджено наказом ректора університету від 15 лютого 2024 № 8

2 Строк подання студентом кваліфікаційної роботи на кафедру 12.06.2024

3 Вихідні дані до роботи Розробити систему виявлення атак за допомогою автоматичного моніторингу вебсервера. Дослідити предметну область що стосується загроз інформаційної безпеки та сканерів вразливостей. Дослідити наявні методи виявлення атак на вебресурси. На основі отриманих результатів досліджень спроєктувати та розробити програмний модуль для виявлення атак. Протестувати розроблену систему виявлення атак на вебдодатки.

4 Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі. Проєктування програмного модуля для виявлення атак. Програмна реалізація модуля виявлення атак на вебдодатки.

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
«Загальний алгоритм системи виявлення атак», «Алгоритм захисту від атак перебору паролей», «Діаграма бази даних модуля виявлення атак».

6 Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Мостовий С.В., старший викладач кафедри кібербезпеки		

7 Дата видачі завдання 16 лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
Вибір і затвердження теми кваліфікаційної роботи	Січень-Лютий	виконано
Ознайомлення з предметною областю	Лютий	виконано
Дослідження існуючих рішень	Лютий	виконано
Постановка задачі	Березень	виконано
Визначення загальних принципів рішення задачі	Березень	виконано
Деталізація принципів рішення задачі	Квітень	виконано
Розробка проєктних рішень	Квітень	виконано
Апробація проєктних рішень	Травень	виконано
Оформлення пояснювальної записки згідно вимог	Травень	виконано
Оформлення графічної частини	Червень	виконано
Захист КР	Червень	виконано

Студент _____ Віталій БАБАЄВСЬКИЙ

Керівник кваліфікаційної роботи _____ Ігор МУЛЯР

АНОТАЦІЯ

Тема кваліфікаційної роботи: Система виявлення атак за допомогою автоматичного моніторингу вебсервера.

Автор роботи: Бабаєвський Віталій Михайлович.

Керівник роботи: Муляр Ігор Володимирович.

Пояснювальна записка: 82 с., 2 додатки, 27 рис., 45 джерел.

Графічна частина: 3 плакати, 10 презентаційних слайдів.

СИСТЕМА ВИЯВЛЕННЯ АТАК ЗА ДОПОМОГОЮ АВТОМАТИЧНОГО МОНІТОРИНГУ ВЕБСЕРВЕРА.

Метою даної роботи є розробка системи виявлення атак на вебдодатки. Головним завданням було підвищення рівня інформаційної безпеки шляхом виявлення та запобігання можливим атакам на вебсервери.

Для досягнення цих цілей було проведено дослідження предметної області, зокрема аналіз загроз інформаційної безпеки, типів атак на вебдодатки та існуючих сканерів вразливостей. Також розглянуто потенційні наслідки атак на вебсервери. На основі проведеного аналізу була поставлена задача створення ефективного програмного рішення для виявлення атак.

У роботі спроектовано структуру вебдодатку, розроблено методи виявлення атак на вебресурси та розглянуто можливі варіанти обходу систем виявлення вторгнень (IDS). Для реалізації програмного модуля було вибрано відповідні інструменти, налаштовано проект та спроектовано базу даних.

Проведено тестування розробленої системи, що підтвердило її ефективність у виявленні атак. У результаті було досягнуто основної мети роботи – забезпечення безпеки вебдодатків шляхом своєчасного виявлення та запобігання атакам.

Розроблений модуль може бути впроваджений у різноманітні вебресурси для підвищення їхньої інформаційної безпеки.

12.06.2024 _____

ANNOTATION

Course project: An attack detection system using automatic web server monitoring.

Author of the work: Babaievskyi Vitalii Mykhailovych.

Supervisor: Mulyar Ihor Volodymyrovych.

Amount: 82 p., 2 appendix, 27 figures, 45 sources.

Graphic part: 3 schemes, 10 presentation slides.

AN ATTACK DETECTION SYSTEM USING AUTOMATIC WEB SERVER MONITORING.

The purpose of this work is to develop a system for detecting attacks on web applications. The main task was to increase the level of information security by detecting and preventing possible attacks on web servers.

To achieve these goals, we conducted a study of the subject area, including an analysis of information security threats, types of attacks on web applications, and existing vulnerability scanners. The potential consequences of attacks on web servers were also considered. Based on the analysis, the task of creating an effective software solution for detecting attacks was set.

The paper designs the structure of the web application, develops methods for detecting attacks on web resources, and considers possible options for bypassing intrusion detection systems (IDS). To implement the program module, the appropriate tools were selected, the project was configured and the database was designed. The developed system was tested, which confirmed its effectiveness in detecting attacks. As a result, the main goal of the work was achieved - to ensure the security of web applications by timely detection and prevention of attacks. The developed module can be implemented in various web resources to improve their information security.

The developed module can be implemented in various web resources to improve their information security.

12.06.2024 

ЗМІСТ

ВСТУП

Швидкий розвиток інформаційних систем (ІС) та технологій має значний вплив на всі сфери суспільства. Багато сучасних державних та приватних компаній використовують ІС для управління виробничими процесами, прийняття рішень, пошуку необхідної інформації та іншого. Проте, цей розвиток призводить до збільшення вразливостей ІС і загроз для них. Для забезпечення їх нормального функціонування і запобігання вторгненням необхідні спеціалізовані засоби безпеки. Один із актуальних напрямів розвитку сфери інформаційної безпеки – виявлення кібератак та запобігання вторгненням в ІС з боку неавторизованих сторін.

Кібератаки на державні підприємства та приватні компанії призвели до того, що стали актуальними розробка спеціальних технічних рішень та систем протидії. Для виявлення мережевих вторгнень застосовуються сучасні методи, моделі, програмне забезпечення та комплексні технічні рішення. Вони мають бути адаптивними до нових або модифікованих видів кіберзагроз. Проте, коли з'являються нові загрози або аномалії, які викликані атаками з неясними характеристиками, ці заходи можуть виявитися недостатньо ефективними і вимагати значних зусиль для адаптації. Тому системи виявлення вторгнень повинні постійно оновлюватися і удосконалюватися.

Серед цих систем можна виокремити спеціалізовані програмні рішення, які спрямовані на виявлення незвичайної активності або втручання в інформаційні системи та вживання відповідних заходів для протидії кібератакам. Ці інструменти включають міжмережеві файрволи, антивірусні програми, системи виявлення та запобігання вторгнень. Оскільки сучасні вебдодатки найбільш широко використовуються, виявлення атак на них є надзвичайно актуальним завданням. Тому було очевидним обрати відповідну тему для дипломної роботи.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд загроз інформаційної безпеки

У сучасному цифровому світі, де вебдодатки відіграють ключову роль у різноманітних аспектах життя, забезпечення безпеки цих додатків стає критично важливим завданням. Інформаційні технології відкривають безліч можливостей для зручного та ефективного обміну даними, але разом із цим існує загроза для конфіденційності, цілісності та доступності цих даних [1, 2].

В інформаційних технологіях, дані представляють собою важливу інформацію, яка може бути записана, збережена, оброблена та передана за допомогою комп'ютерних систем. Вони можуть мати різні формати, такі як текст, числа, зображення, аудіо або відео, або бути структурованими в базах даних.

Значення та рівень конфіденційності даних можуть варіюватися. Наприклад, особисті дані користувачів, такі як імена, адреси або фінансові відомості, потребують особливого захисту через їх чутливість. У той же час інші дані, які стосуються, наприклад, стану системи або результатів операцій, можуть бути менш чутливими і потребувати меншого рівня захисту.

Обробка та аналіз даних відіграють ключову роль у багатьох сферах інформаційних технологій, таких як аналітика даних, машинне навчання та штучний інтелект. З цього приводу важливо забезпечити належний захист даних від несанкціонованого доступу, змін або втрати [3].

Тріада цілісності, доступності та конфіденційності – це основні принципи безпеки даних. Ці принципи становлять основу для захисту інформації від різних загроз [4].

Конфіденційність – це принцип та стан інформації, при якому доступ до неї обмежений лише певним особам або групам осіб, яким ця інформація довірена, та які мають право використовувати її відповідно до встановлених правил та політик.

Конфіденційність включає:

– інформація повинна бути доступна лише особам або групам осіб, яким це необхідно для виконання їхніх обов’язків або функцій;

– інформація повинна бути захищена від несанкціонованого доступу, такого як атаки хакерів, витоку даних або крадіжки;

– шифрування часто використовується для захисту конфіденційної інформації під час передачі або зберігання, щоб запобігти її читання для несанкціонованих осіб;

– керування правами доступу дозволяє визначати, які користувачі або групи користувачів мають доступ до якоїсь інформації, та визначати рівень цього доступу (наприклад, читання, запис, виконання);

– навчання персоналу правилам та процедурам захисту конфіденційної інформації, щоб запобігти випадковим витокам або порушенням;

– інформація може бути захищена фізичними заходами безпеки, такими як замки, ключі, картки доступу та контрольовані доступні приміщення.

Щоб забезпечити конфіденційність даних, організації встановлюють механізми контролю доступу, шифрування і персоналізовані права доступу. Наприклад, фінансові організації забезпечують конфіденційність даних за допомогою систем аутентифікації, двофакторної перевірки і захисту від несанкціонованого доступу. Банки та інші фінансові установи мають строгі політики конфіденційності, які регулюють, хто має доступ до клієнтських даних і як ці дані можуть бути використані [5].

Цілісність даних відноситься до їх збереження та недоторканності. У контексті інформаційної безпеки, цілісність означає, що дані залишаються незмінними та незіпсованими під час їх передачі, зберігання чи обробки. Іншими словами, дані повинні залишатися точними, повними та незмінними щодо того, як вони були створені чи зафіксовані.

Основна мета забезпечення цілісності даних полягає в тому, щоб гарантувати, що жодних несанкціонованих змін не внесено до інформації. Це

важливо для запобігання спотворенням, помилкам або зловмисним маніпуляціям, які можуть призвести до недостовірності або втрати цінної інформації.

Приклади методів забезпечення цілісності даних включають використання хеш-функцій для перевірки цілісності файлів, цифрові підписи для перевірки автентичності повідомлень, контрольні суми для виявлення помилок у передачі даних, а також методи контролю версій і аудиту для відстеження змін у даних та їх історії.

Доступність даних – це здатність отримати доступ до інформації та ресурсів у потрібний час без ненормативного затримання чи переривання. В інформаційній безпеці доступність означає, що дані та сервіси мають бути доступні користувачам у моменти, коли їм це потрібно, та мають функціонувати без збоїв чи переривань.

Цей аспект інформаційної безпеки особливо важливий для забезпечення безперервної роботи бізнес-процесів та задоволення потреб користувачів. Відмова у доступі до даних або сервісів може призвести до серйозних негативних наслідків, таких як втрата доходів, шкода репутації бренду, втрата клієнтів та порушення довіри.

Для забезпечення доступності даних застосовуються різні методи та технології, такі як:

– створення резервних копій даних та систем, щоб у разі збою або аварії можна було швидко відновити працездатність;

– використання високодоступних архітектур та резервування ресурсів для забезпечення безперервної роботи систем навіть у разі відмови частини інфраструктури;

– розподіл навантаження між різними ресурсами для запобігання перевантаженням та забезпечення рівномірної доступності;

– постійний моніторинг стану систем та ресурсів для швидкого виявлення та реагування на загрози для доступності;

– застосування механізмів для запобігання та пом’якшення наслідків DDoS-атак, які можуть призвести до відмови у доступі до даних та сервісів.

Однак існують різні загрози, які можуть порушити ці принципи безпеки даних. Загроза (англ. threat) – це будь-який потенційний фактор події або дії, який може завдати

шкоди системі, даним або ресурсам. У контексті інформаційної безпеки, загроза є можливість виникнення інциденту безпеки або порушення безпеки інформації [6, 7].

Загрози можуть виникати як з внутрішнього, так і зовнішнього середовища. Вони можуть бути пов'язані з діями зловмисників, помилками персоналу, технічними несправностями, природними лихами та іншими факторами.

В інформаційній безпеці загрози класифікуються на різні типи в залежності від їх характеристик, таких як:

– загрози конфіденційності направлені на неправомірне отримання або розкриття конфіденційної інформації;

– загрози цілісності направлені на несанкціоновані зміни чи модифікації даних;

– загрози доступності спрямовані на зниження доступності системи або даних для легальних користувачів;

– загрози автентичності направлені на заміну або фальшиве подання особи або даних;

– загрози авторизації направлені на несанкціоноване отримання доступу до системи чи ресурсів.

Захист даних від різних загроз вимагає комплексного підходу для забезпечення безпеки, збереження та цілісності чогось від загроз та ризиків. У контексті інформаційних технологій та комп'ютерної безпеки захист включає широкий спектр заходів, спрямованих на забезпечення безпеки інформації, систем та ресурсів [8, 9].

Основні аспекти захисту включають:

– запобігання загрозам безпеці та атакам. Вони включають реалізацію політик безпеки, застосування найкращих практик і стандартів, навчання персоналу та використання технологій, здатних виявити та блокувати потенційні загрози;

– виявлення можливих загроз та інцидентів безпеки. Це включає моніторинг та аналіз діяльності системи, виявлення аномальної поведінки та сигналів безпеки, а також регулярний аудит системи;

– виявлені загрози та інциденти безпеки. Вони включають швидке реагування на інциденти, ізоляцію та ліквідацію вразливостей, відновлення системи після атаки та запобігання повторному виникненню загрози;

– відновлення нормальної системи після інциденту безпеки. Це може включати відновлення даних з резервних копій, оновлення програмного забезпечення та систем, а також аналіз причин інциденту для запобігання його повторному виникненню.

Backend (серверна частина) вебдодатку є частиною програмного забезпечення, яка обробляє дані, взаємодіє з базами даних, виконує бізнес-логіку програми та забезпечує зв'язок з клієнтською) частиною програми.

Основні характеристики та функції серверної частини включають:

– приймання HTTP-запитів від клієнтської частини програми та їх обробка;

– взаємодія з базами даних для збереження, отримання та обробки даних, необхідних для роботи програми;

– реалізація бізнес-логіки програми, яка визначає, як додаток повинен обробляти дані та виконувати різні операції;

– забезпечення механізмів аутентифікації користувачів та авторизації;

– відповідальність за захист даних та забезпечення безпеки програми, включаючи захист від вразливостей та атак;

– реалізація API для взаємодії з іншими зовнішніми сервісами та додатками.

Надання безпечного середовища для обробки та зберігання даних – це не лише відповідальність перед користувачами, а й необхідність довгострокового успіху бізнесу. Користувачі очікують, що їх дані зберігатимуться та оброблятимуться надійно, без ризику витоку або зміни без їхньої згоди. Крім того, порушення роботи вебдодатків може призвести до негативних наслідків як для користувачів, так і для бізнесу. Втрата доступності сервісу може призвести до втрати клієнтів, довіри та доходів.

Поняття захисту даних та функціоналу вебдодатків охоплює широкий спектр заходів та технологій, спрямованих на запобігання, виявлення та реагування на

загрози безпеці. Це включає не лише технічні аспекти, такі як використання шифрування та механізмів аутентифікації, але й організаційні заходи, такі як навчання персоналу, розробка політик безпеки та регулярний аудит системи [10-12].

1.2 Типи атак на вебдодатки

SQL injection – це вид атаки на вебдодатки, яка використовується для зловмисного виконання SQL-запитів на сервері баз даних через недоліки в обробці введених даних користувачем [13-15]. Суть атаки полягає в тому, що зловмисник вставляє SQL-код в поля вводу на вебсторінці, змушуючи сервер баз даних виконати небезпечні операції, такі як видалення, модифікація або отримання конфіденційної інформації.

Розглянемо просту систему автентифікації з використанням таблиці бази даних з іменами користувачів і паролями. Запит користувача POST передає змінні user та pass, які вставляються в оператор SQL:

```
sql = "SELECT id FROM users WHERE username='" + user + "' AND password='" + pass + "'"
```

Проблема полягає в тому, що оператор SQL використовує конкатенацію для об’єднання даних. Зловмисник може надати такий рядок замість pass змінної:

```
SELECT id FROM users WHERE username='user' AND password='pass' OR 1=1
```

Оскільки 1=1 це умова, яка оцінюється як істинна, оператор WHERE поверне перший ідентифікатор із таблиці користувачів, яким зазвичай є адміністратор. Це

означає, що зловмисник може отримати доступ до програми без автентифікації, а також має права адміністратора.

Міжсайтові скриптові атаки, також звані XSS-атаками, є типом ін'єкційних атак, які впроваджують шкідливий код на безпечні вебсайти [16, 17]. Зловмисник використає недолік цільової вебпрограми, щоб надіслати кінцевому користувачеві якийсь шкідливий код, найчастіше клієнтський JavaScript. Замість того, щоб націлюватися на сам хост програми, XSS-атаки зазвичай спрямовані безпосередньо на користувачів програми.

XSS-атаки загалом можна розділити на три категорії:

– Stored XSS зазвичай виникає, коли введені користувачем дані зберігаються на цільовому сервері, наприклад у базі даних, у форумі повідомлень, журналі відвідувачів, полі коментарів тощо. Тоді жертва може отримати збережені дані з вебпрограми без цього безпечне відображення даних у браузері. З появою HTML5 та інших технологій браузера ми можемо уявити, що корисне навантаження атаки постійно зберігається у браузері жертви, наприклад у базі даних HTML5, і взагалі ніколи не надсилається на сервер;

– Reflected XSS виникає, коли вебдодаток негайно повертає введені користувачем дані у повідомленні про помилку, результатах пошуку або будь-якій іншій відповіді, яка включає частину або всі введення, надані користувачем як частину запиту, без захисту цих даних. візуалізувати в браузері без постійного збереження наданих користувачем даних. У деяких випадках надані користувачем дані можуть навіть не залишити веббраузер;

– DOM-based XSS – це XSS-атака, у якій корисне навантаження атаки виконується в результаті модифікації «оточення» DOM у браузері жертви, що використовується оригінальною стороною клієнта. сценарій, щоб код на стороні клієнта запускався «несподіваним» способом. Тобто сама сторінка (тобто HTTP-відповідь) не змінюється, але код на стороні клієнта, що міститься на сторінці, виконується по-різному через зловмисні зміни, які відбулися в середовищі DOM.

Розглянемо XSS ін'єкцію на прикладі сервісу авторизації який не фільтрує дані:

```
<!DOCTYPE html>
<html>
<head>
</head>
 <title>Авторизачія</title>
<body>
 <h1>Вітаємо!</h1>
 <p>Введіть дані для авторизації:</p>
 <form method="post">
       <input type="text" name="username">
  <input type="password" name="password">
  <input type="submit" value="Надіслати">
 </form>
</body>
</html>
```

При обробці введеного імені username на сервері передбачається, що його буде безпечно відображено в особистому кабінеті. Однак, якщо введене ім'я не фільтрується належним чином, зловмисник може використовувати це для впровадження шкідливого JavaScript коду. Наприклад, зловмисник може ввести наступне ім'я:

```
<script>alert('XSS atack');</script>
```

Якщо сторінка авторизації не екранує це введення шкідливого JavaScript коду, цей код буде впроваджено в HTML-код сторінки і виконається у браузері іншого користувача, який перегляне сторінку привітання.

Даний вид атаки небезпечний через те, що можуть бути викрадені кукі, які згодом можуть використовувати при авторизації.

Підробка міжсайтового запиту (CSRF) – це вразливість, через яку зловмисник виконує дії, видаючи себе за іншого користувача [18-20]. Наприклад, перерахувати кошти на рахунок зловмисника, змінити адресу електронної пошти жертви або навіть просто перенаправити піцу на адресу зловмисника! Схема атаки (рис. 1.1) полягає в тому, що зловмисник змушує жертву виконати шкідливий запит, під виглядом легітимної дії, часто через підроблену форму або шкідливий лінк, розміщений на іншому вебсайті.

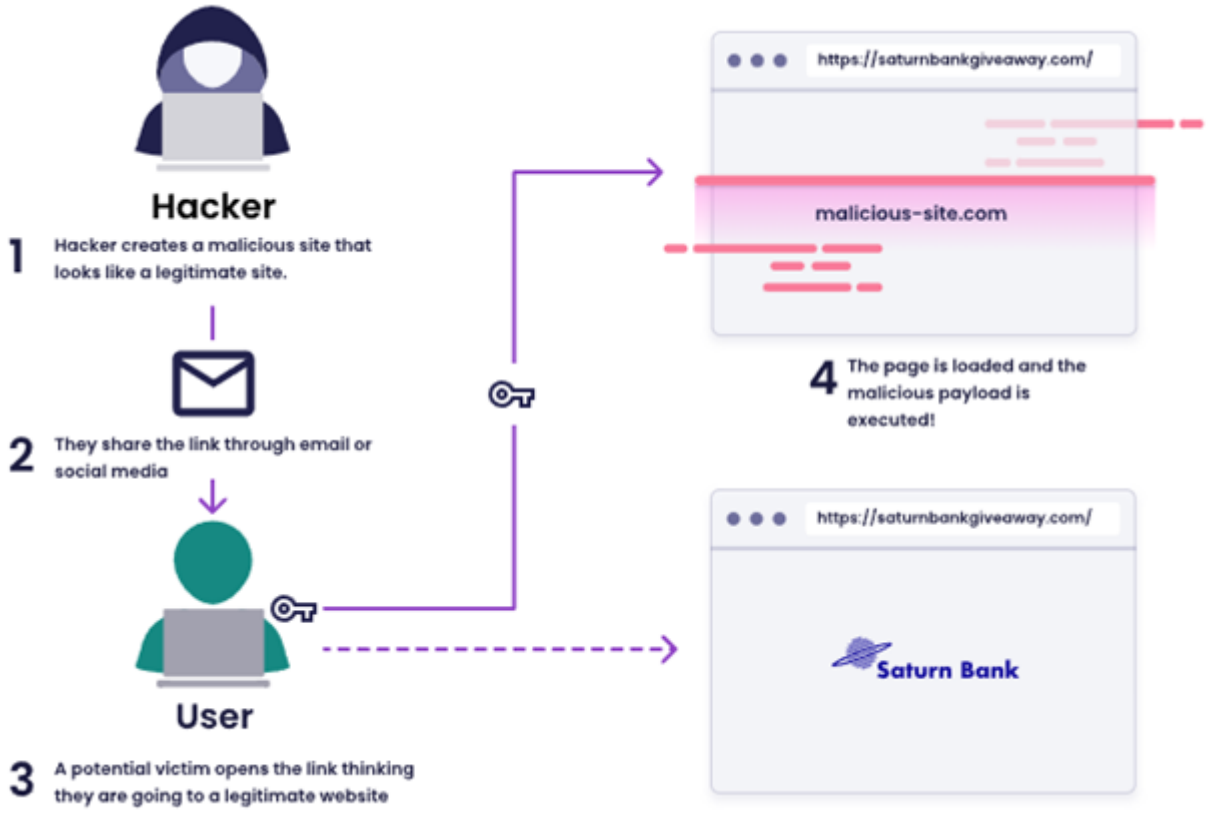


Рисунок 1.1 - Схема CSRF атаки [21]

Щоб така атака була успішною, зазвичай потрібна певна форма соціальної інженерії, як-от фішинг або спуфінг. Для здійснення атаки зловмиснику зазвичай потрібно змусити користувача відвідати шкідливий вебсайт. Потім цей шкідливий вебсайт містив би запит до цільового вебсайту. Якщо користувач автентифікований цільовим вебсайтом, запит виконується. Ця атака працює, оскільки файли cookie користувача автоматично включаються в модифікований запит до законної програми.

Уразливості CSRF виникають, коли вразливі вебпрограми просто довіряють файлам cookie, надісланим веббраузерами, без подальшої перевірки.

Вебдодаток уразливий до CSRF, якщо він покладається на файли cookie сеансу для ідентифікації користувачів і не має іншого механізму перевірки запитів. Крім того, запит, який ми хочемо використовувати, повинен мати передбачувані параметри запиту, щоб ми могли створити власний запит, як у цьому прикладі.

На щастя для нас, ми отримали інформацію про вразливу банківську програму під назвою «Saturn Bank». Ця програма просто використовує сеансові файли cookie для перевірки запитів. Крім того, файли cookie сеансу не мають атрибута SameSite. Пізніше ми розглянемо, що таке цей атрибут. Коротше кажучи, вони контролюють, як файли cookie надсилаються в міжсайтових запитах.

За лаштунками, коли наші жертви відвідали наш шкідливий сайт “saturnbankgiveaway.com”, запит POST був ініційований і надісланий до законної програми Saturn Bank. JavaScript, який міститься в тегах «script», гарантує, що форма надсилається, щойно користувач завантажує сторінку, без будь-якої взаємодії з користувачем, або користувач навіть не помічає, що відбувається. Форма нище створює наступний запит до програми Saturn Bank:

```
<html>
  <body>
    <form action="https://saturnbank.com/transfer" method="POST">
      <input type="hidden" bsb="421314" accountNo="1736123125" amount="100" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

Запит містить сеансовий файл cookie (рис. 1.2) законного користувача, але містить номер нашого банківського рахунку!

```
POST /transfer HTTP/1.1
Host: saturnbank.com
Content-Length: 42
Content-Type: application/x-www-form-urlencoded
Cookie: session=OM19vamvikL4yvPQfTqrcjW2ItpDAkDm
bsb=421314&accountNo=1736123125&amount=100
```

Рисунок 1.2 - Пост запит до банку

Однією зі стратегій пом’якшення є використання випадкового та унікального маркера для використання в запитах HTTP; вони називаються CSRF, маркери захисту від підробки або маркери перевірки запиту. Вони є спільним секретом між клієнтською та серверною частиною програми та включені до будь-яких запитів, які клієнт робить до сервера. Сервер перевіряє маркер під час кожного запиту, щоб переконатися, що це авторизований користувач, який робить запит. Маркер зазвичай міститься в прихованому полі форми HTML. Оскільки маркер є випадковим і унікальним, зловмисник не може передбачити значення для використання у своєму зловмисному запиті.

Ще один спосіб захиститися від CSRF – застосувати атрибут SameSite до файлів cookie. Цей атрибут додається до заголовка відповіді Set-Cookie і може мати значення «Strict» або «Lax». Наприклад:

Set-Cookie: SessionId=sYMnfCUrAlmqVVZn9dqevxyFpKZt30NN; SameSite=Strict;

Значення «Strict» гарантує, що браузер не включає файли cookie в будь-які запити, які надходять з іншого сайту. В свою чергу значення «Lax» змушує браузер включати файли cookie, лише якщо запит є методом GET і запит ініціював користувач, а не сценарії.

Атака проходження каталогу (рис. 1.3) спрямована на доступ до файлів і каталогів, які зберігаються за межами цільової папки [21]. Маніпулюючи файлами за допомогою послідовностей «точка-крапка-слеш (../)» та її варіацій або використовуючи абсолютні шляхи до файлів, можна отримати доступ до довільних файлів і каталогів, що зберігаються у файловій системі; включаючи вихідний код програми, конфігурацію та інші важливі системні файли.

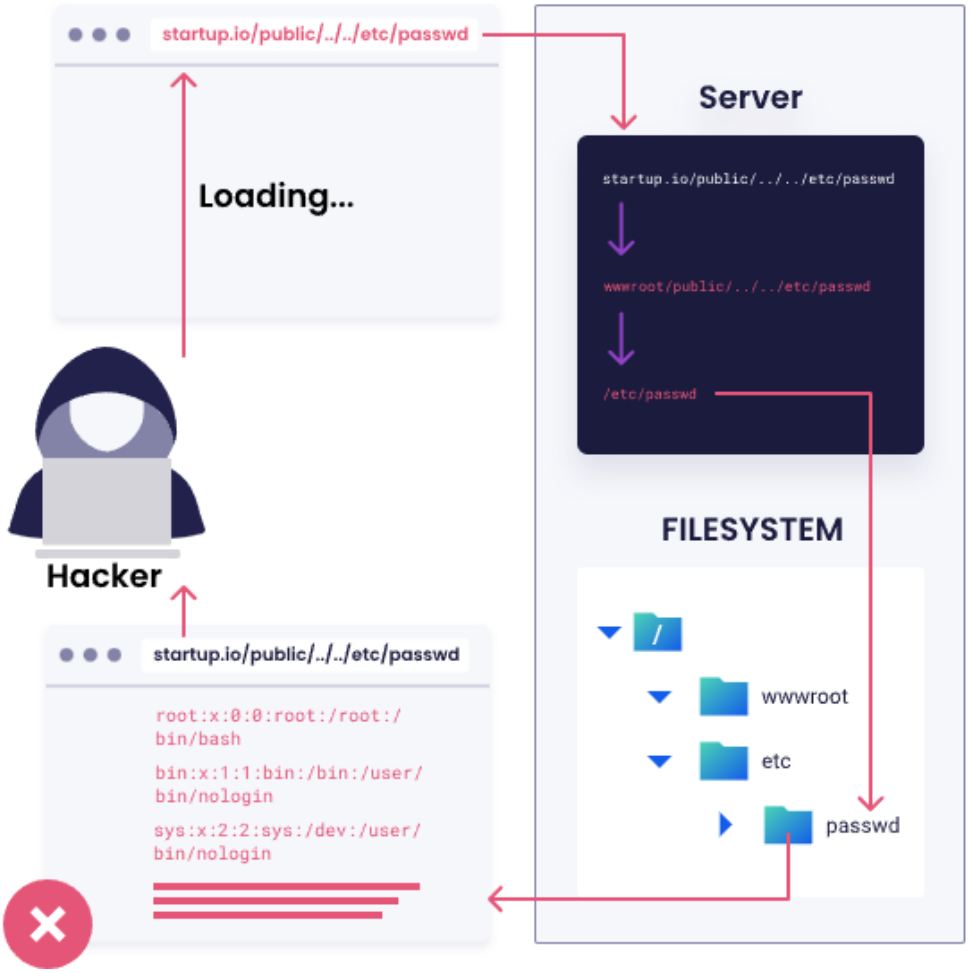


Рисунок 1.3 - Схема атаки обходження каталогу [22]

По суті, атака здійснюється шляхом додавання символів, наприклад, ../до URL-адреси, яка подає вміст зі структури каталогу. Вміст зазвичай подається з базового каталогу, такого як /public. Зловмисник може надати імена файлів, які містять ../або еквівалент у кодуванні URL-адреси %2e%2e%2f. Ці URL-адреси

дозволяють зловмиснику вийти з основного каталогу та переглянути файли, що зберігаються в інших папках у файловій системі.

Атака проходження каталогу, яка показує, як хакер надсилає зловмисне навантаження на сервер і отримує доступ до файлів, які не повинні бути публічними.

Найнадійніший спосіб запобігти атакам обходу каталогу – уникати покладатися на дані користувача під час роботи з API файлової системи.. Більш реалістичним механізмом пом’якшення є запобігання тому, щоб наданий користувачем каталог був вище у файловій системі, ніж каталог, який використовується для обслуговування статичного вмісту.

У наведеному нижче прикладі коду показано, як нормалізувати наданий користувачем шлях і перевірити, чи він починається в очікуваному каталозі.

1.3 Огляд сканерів вразливостей

Сканер вразливостей – це автоматизований інструмент тестування вразливостей, який відстежує неправильні конфігурації або недоліки програмного коду, що становлять загрозу безпеці вебсайту. Сканери вразливостей або покладаються на базу даних відомих вразливостей, або досліджують поширені типи помилок, щоб виявити невідомі вразливості [24].

Сканування вразливостей допомагає компаніям визначити можливі способи використання зловмисником вразливостей, які можуть спричинити збій системи, дозволити несанкціонований доступ до мережі або отримати привілейовану інформацію. Застарілі програмні продукти, невиправлені операційні системи та неправильно налаштоване апаратне забезпечення часто призводять до вразливостей [25].

Сканування використовує багато різних методів, щоб змусити програми реагувати або зчитувати інструкції в несподіваний спосіб. Зловмисники можуть використовувати ці недоліки для виконання шкідливого коду, викрадення інформації з пам'яті та встановлення зловмисного програмного забезпечення [26].

Як малі, так і великі організації можуть отримати вигоду від періодичного сканування вразливостей, щоб гарантувати, що їхня IT-інфраструктура не буде вразливою до атак. Для зловмисників зараз простіше, ніж будь-коли, використовувати спеціалізовані інструменти для пошуку компаній з конкретними вразливостями, які можна використати.

З огляду на вартість кібератаки, сканування вразливостей є економічно ефективним способом активного захисту мережі шляхом виявлення та усунення вразливостей до того, як зловмисники зможуть їх знайти.

Сканери додатків далі поділяються на ті, які націлені на вебдодатки, і ті, які націлені на власні програми. Сканери також існують для ряду спеціалізованих підкатегорій, таких як хмарна інфраструктура, мобільні програми або вебдодатки, створені з використанням певної платформи чи технології.

Сканери вебвразливостей сканують код вебсайту, щоб знайти вразливості, які загрожують самому вебсайту або його серверним службам. Вони є важливим компонентом тестування безпеки програми .

Ці сканери працюють над пошуком поширених експлойтів, який підтримує OWASP та інші. Ці експлойти використовують різноманітні методи ін’єкцій та обходу механізмів захисту інформацію, щоб отримати несанкціонований доступ до вебсайту, щоб викрасти дані, змусити користувачів або системи надати конфіденційну інформацію або порушити роботу програми. Деякі з найбільш відомих експлойтів – це впровадження SQL, міжсайтовий сценарій (XSS), атака типу «людина посередині» (MITM) та ін’єкція шкідливого програмного коду.

Відомі сканери вразливостей:

Zed Attack Proxy (ZAP), спочатку створений OWASP, є сканером уразливостей з відкритим кодом, спеціально розробленим для вебдодатків. Він функціонує як проксі сервер для перехоплення запитів, що дозволяє користувачам перевіряти та змінювати трафік між своїм браузером і вебсервером.

ZAP (рис. 1.4) пропонує автоматизовані сканери, а також набір інструментів для ручного тестування безпеки. Ключовою особливістю є його павук, який автоматично сканує нові програми, щоб ідентифікувати URL-адреси, які потім виявляють уразливості.

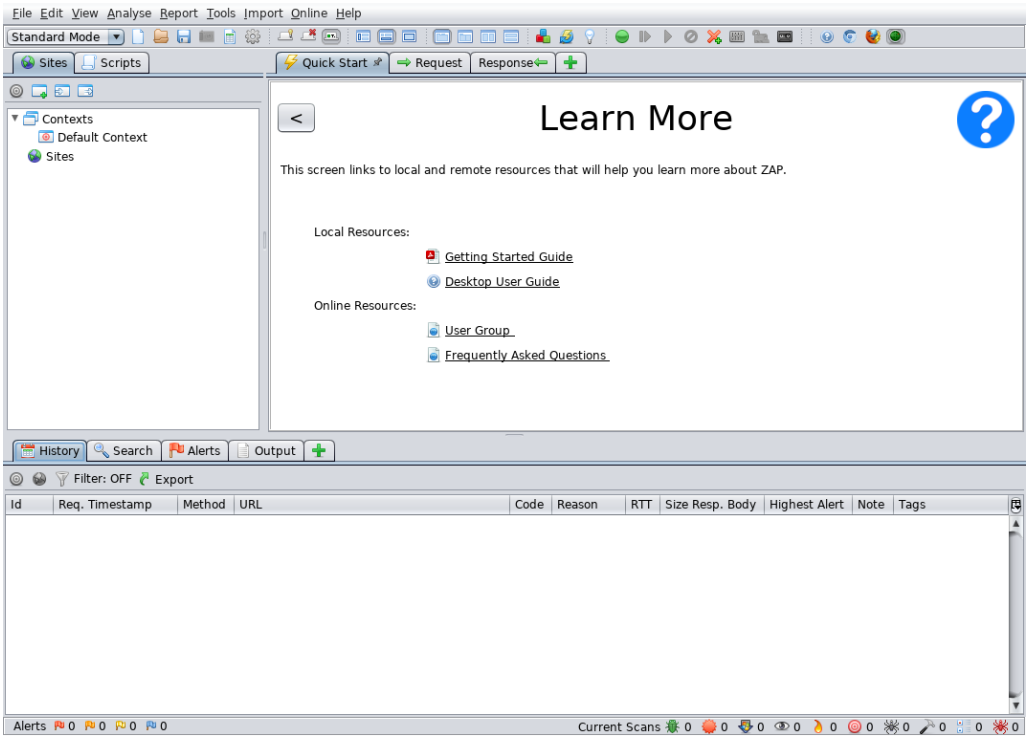


Рисунок 1.4 - Інтерфейс сканера ZAP

ZAP також надає функцію пасивного сканування, яка виявляє проблеми безпеки у фоновому режимі під час перегляду, що робить його ефективним інструментом для поточної оцінки безпеки [27, 28]. Його активний сканер перевіряє ідентифіковані URL-адреси, щоб знайти потенційні вразливості. Оскільки він керується спільнотою, він регулярно оновлюється останніми даними про загрози.

Nessus, розроблений Tenable Network Security, є комерційним інструментом керування вразливістю. Він шукає широкий спектр вразливостей, включаючи недоліки програмного забезпечення, відсутні виправлення, зловмисне програмне забезпечення та неправильні конфігурації в різних системах.

Nessus (рис. 1.5) славиться своїм швидким виявленням, аудитом конфігурації, виявленням конфіденційних даних та аналізом патчів безпеки для

уразливостей [29, 30]. Його архітектура плагінів гарантує, що він завжди оновлюється з огляду на найсвіжіші уразливості.

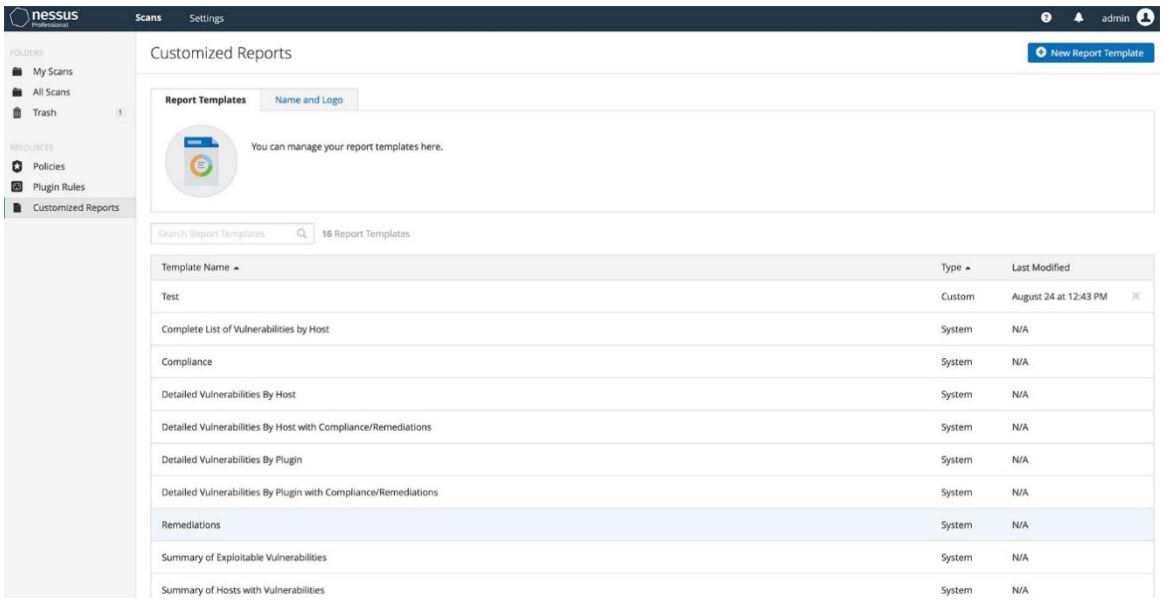


Рисунок 1.5 - Графічний інтерфейс Nessus

Ключовою особливістю Nessus є його здатність зберігати політики та конфігурації, що дозволяє виконувати послідовне сканування в різних середовищах. Він також містить докладні кроки для усунення кожної виявленої вразливості, що може допомогти IT-фахівцям в усуненні прогалин у безпеці.

Burp Suite Community Edition – безкоштовна платформа для тестування безпеки вебдодатків [31]. Версія для спільноти обмежена, ніж професійна версія у ній відсутній автоматичний сканер, але вона все ще корисна для ручного тестування. Загальна робота Burp є модульною. Деякі його модулі встановлені в програмному забезпеченні за замовчуванням. Це основні модулі для проведення аудиту. Інші додаткові модулі, які називаються розширеннями, доступні для завантаження через розширювач. Сильна сторона Burp Suite, окрім його модульності та зручності для користувача, полягає в його дуже активній спільноті, яка розробляє нові розширення та створює детальну документацію щодо модулів.

Головною особливістю Burp Suite є проксі (рис. 1.6). Проксі-сервер дозволяє Burp діяти як посередник між клієнтом і сервером, на якому розміщено вебдодаток.

Розміщуючи себе між цими двома компонентами, Burp зможе перехоплювати всі обміни та запити, зроблені між веббраузером і сервером. Таким чином пентестер зможе детально проаналізувати запити та, якщо забажає, змінити їх. Щоб змінити запити, проксі-сервер перехоплює запити один за одним і дозволяє пентестеру вибирати, пропускати їх чи відхиляти. Якщо він пропускає їх, він може змінити їх перед передачею на сервер. Проксі також дозволяє переглядати історію запитів у реальному часі без необхідності передавати їх на сервер вручну. Фактично, це найбільш часто використовуваний режим проксі.

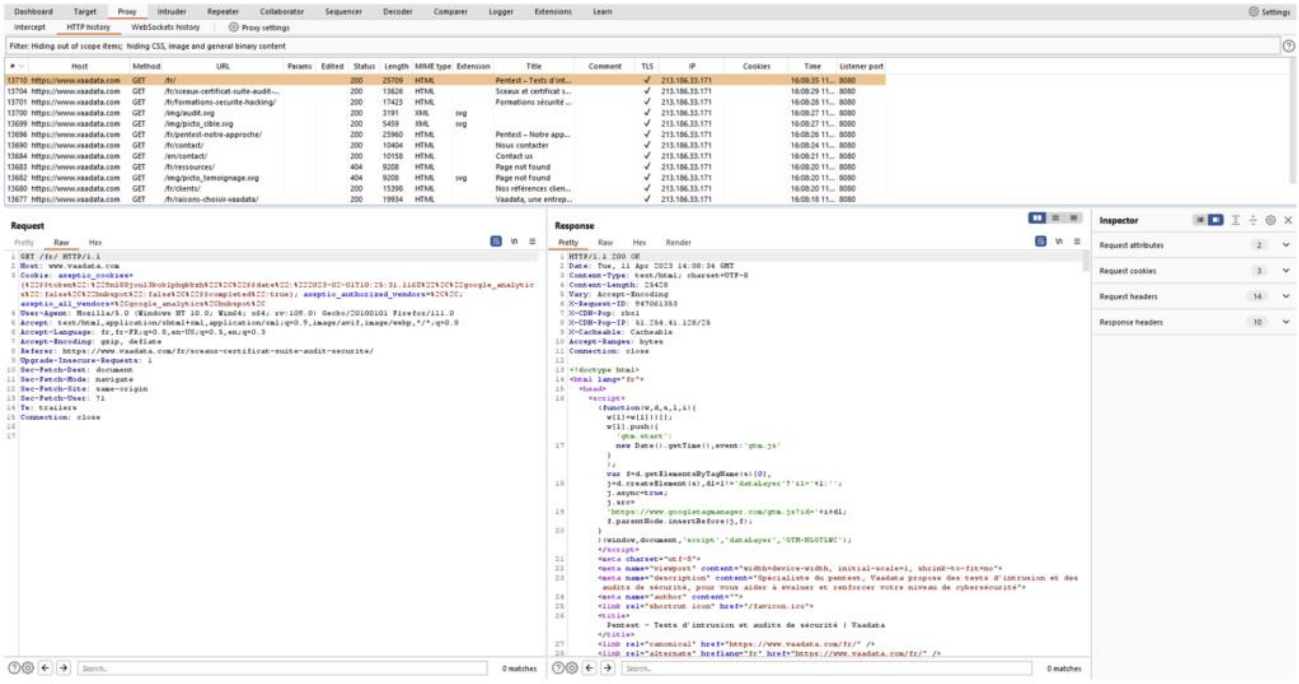


Рисунок 1.6 - Проксі-інтерфейс Burp Suite

Nikto2 – це сканер вебсерверів із відкритим кодом, який може тестувати вебсервери на наявність понад 6700 потенційно небезпечних файлів і програм, перевіряти наявність застарілих версій понад 1250 серверів і виявляти проблеми, пов’язані з версією, на понад 270 серверах [32]. Він призначений для виявлення потенційно шкідливих файлів і програм на вебсервері, пропонуючи простий підхід до тестування вебсервера.

Nikto2 може сканувати на наявність неправильних конфігурацій сервера та програмного забезпечення, файлів і програм за замовчуванням, незахищених

файлів і сценаріїв CGI. Його архітектура на основі плагінів дозволяє користувачам розширювати його можливості. Хоча це не прихований інструмент і призначений для швидкого сканування, його ретельність робить його відповідним вибором для планових перевірок і раннього виявлення поширених вразливостей вебсервера.

1.4 Потенційні наслідки атак на вебсервери

Якщо компанія не захистить свій вебсайт від хакерів і вірусів, існує ризик того, що конфіденційні дані клієнтів можуть з’явитися в темній мережі [33]. У цьому випадку компанія, ймовірно, назавжди втратить клієнтів, які постраждали від порушення.

Кіберзлочинці, як правило, мотивовані фінансовою вигодою, вбачаючи у фінансовому секторі можливість позбавити клієнтів і власників фінансового бізнесу їхні гроші. Вони можуть зосереджуватися на атаках, таких як фішинг або програми-вимагачі, щоб отримати облікові дані доступу та використовувати їх для здійснення несанкціонованих транзакцій.

У липні 2020 року Cashaa, платформа онлайн-банкінгу, яка керує фіатними валютами та криптовалютами, зазнала кібератаки з крадіжки грошей . Хакери використовували різні методи, включаючи фішинг, віруси, щоб отримати контроль над комп’ютером співробітника з активними сесіями, відкритими в браузері. Хакерам вдалося проникнути в уразливу машину співробітника і протягом мікросекунд всі кошти рахунку були вкрадені.

Також однією з причин атаки на сайт є незаконне отримання доступу до інформації. Часто хакер продає вкрадену інформацію або використовує її для отримання викупу. Як у випадку з Європейським центральним банком у 2014 році, коли його вебсайт було зламано і викрадено дані, а потім вимагали викуп. Крадіжка особистих

даних може призвести до прямих витрат, якщо дані є цінними або їх можна продати, і це може мати непрямі витрати через підрив довіри.

Щоб викрасти інформацію, хакери можуть зламати систему, щоб стежити за організацією. Цей тип шпигунства може бути важко ідентифікувати. Може пройти кілька років, перш ніж жертва дізнається, що її безпеку було скомпрометовано та що за нею шпигували. Шпигуни можуть шукати комерційні таємниці, такі як рецепти чи патенти. Або вони можуть шукати конфіденційну та секретну інформацію.

Це може бути державна шпигунська місія або промисловий конкурент, який неетично намагається вирватися на ринку.

Останнім часом спостерігається збільшення кількості атак на кібербезпеку на державному рівні. Наприклад, крадіжка секретних кодів військово-морських сил США, яку приписують державним спецслужбам Китаю.

У вересні 2020 року газета The Sydney Morning Herald повідомила, що зловмисні кібератаки на австралійські підприємства та державні установи з боку державного актора зросли за останні два місяці, оскільки національний центр кібербезпеки отримує звіти про атаку кожні 10 хвилин.

Є також активісти, які зламують сайти, щоб шпигувати за те, що вони вважають вищим благом. Наприклад, у серпні 2020 року Sky News повідомила, що група активістів виявила, що комуністична партія Китаю використовувала розпізнавання облич, щоб націлюватися на групи меншин і пригнічувати їх, використовуючи високотехнологічне спостереження, яке потім мобілізували для ув’язнення дисидентів у табори для ідеології.

Додатки для мобільних телефонів і планшетів постійно шпигують за людьми. Нещодавно соціальний додаток Tik Tok було розкритиковано за те, що він використовував його для збору даних, а потім для створення профілів даних про американців для вербування в розвідку. Завжди доцільно знати, якими даними ви погоджуєтеся ділитися та з ким.

1.5 Постановка задачі

Для вирішення проблем захисту вебсерверів потрібно створити систему для ефективного виявлення та реагування на різноманітних атак на вебсервер. Основним завданням є розробка алгоритмів та методів моніторингу, які забезпечать постійний аналіз вхідних запитів до вебсайту з метою виявлення шкідливих запитів та аномальної активності з боку користувача. Отже, на основі теоретичних знань, ми визначили перелік вимог до системи виявлення атак:

– система повинна виявляти XSS та SQL ін’єкції за допомогою сигнатурного методу виявлення атак на вебресурси;

– система повинна протидіяти спробам брутфорсу паролів;

– система повинна блокувати користувач при перевищені ліміту запитів до сервера;

– система не повинна впливати на продуктивність інформаційного ресурсу та залишатись надійною.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ДЛЯ ВИЯВЛЕННЯ АТАК

2.1 Структура вебдодатку

Вебдодаток – це програмне забезпечення, яке запускається на сервері і доступне для користувачів через Інтернет за допомогою веббраузера. Вебдодаток складається з клієнтської частини, яка відповідає за взаємодію з користувачем та відображення інформації, і серверної частини, яка обробляє запити від клієнтів, взаємодіє з базою даних та забезпечує основну логіку додатку. Такий підхід дозволяє створювати потужні та гнучкі програми, які можуть бути доступні для використання з будь-якого пристрою з підключенням до Інтернету.

Архітектура вебдодатків визначає спосіб взаємодії між додатками, проміжним програмним забезпеченням та базами даних для спільної роботи кількох додатків. Коли користувач вводить URL-адресу, браузер звертається до вебсайту за необхідною інформацією. Сервер відповідає, надсилаючи файли, які браузер відображає. Архітектура враховує ефективність, масштабованість, надійність та безпеку, оскільки вебдодатки відіграють ключову роль у глобальному мережевому трафіку.

У традиційній 2-рівневій архітектурі є два компоненти: клієнтська система або інтерфейс користувача та серверна система, якою зазвичай є сервер бази даних. Недоліком 2-рівневої архітектури є зниження продуктивності зі збільшенням кількості користувачів. Крім того, безпосередня взаємодія бази даних і пристрою користувача викликає проблеми безпеки. Системи залізничного бронювання та системи керування вмістом – це кілька додатків, які зазвичай будуються з використанням цієї архітектури.

Трирівнева архітектура вебдодатку (рис. 2.1) поділяє всю систему на три взаємопов’язані, але незалежні модулі:

– клієнтський рівень представляє інтерфейс користувача програми та відповідає за представлення даних користувачеві та отримання введених даних. Він

включає компоненти графічного інтерфейсу користувача (GUI), такі як вебсторінки, форми та віджети;

– серверний рівень містить бізнес-логіку програми та виконує основну обробку даних. Він відповідає за обробку запитів користувачів, отримання та обробку даних, а також виконання складних операцій. Він включає такі модулі, як контролери, служби та API;

– рівень даних відповідає за керування зберіганням і отриманням даних у системі. Він включає базу даних або файлову систему, де зберігаються дані, а також рівень доступу до даних, який взаємодіє з базою даних для читання та запису даних.

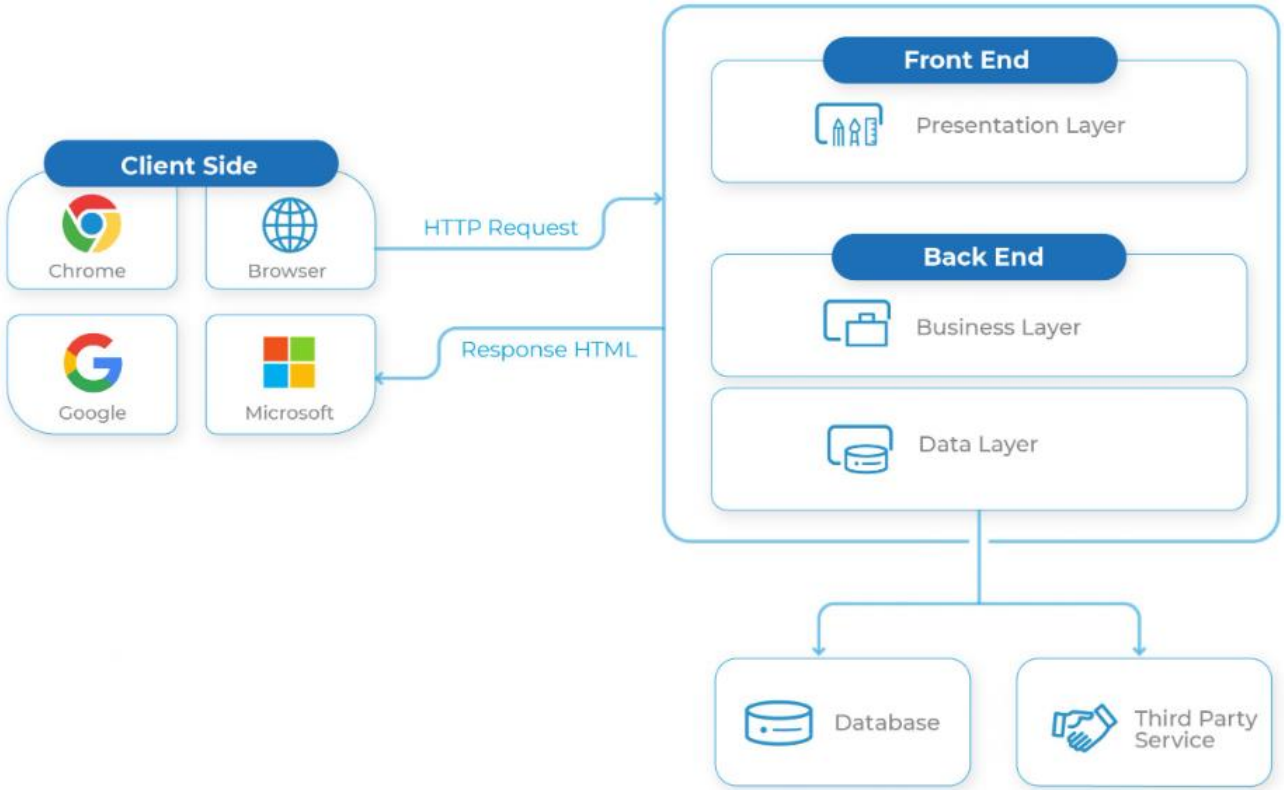


Рисунок 2.1 - Схема трирівневої архітектури [34]

Три рівні з’єднані через чітко визначені інтерфейси, що дозволяє кожному рівню функціонувати незалежно та дозволяє легко модифікувати та підтримувати програму. Трирівнева архітектура є широко використовуваним шаблоном проектування для створення програмних програм, які можна масштабувати та підтримувати, і вона зазвичай використовується у вебдодатках і корпоративних програмних системах.

2.2 Методи виявлення атак на вебресурси

Під час створення систем виявлення атак важливо знайти оптимальний метод для виявлення потенційно шкідливих дій [35]. У галузі вебсистем існують два основних підходи: статичний і динамічний.

Статичний підхід базується на аналізі журналів подій програмного забезпечення та операційної системи. Цей метод привертає увагу своєю простотою впровадження і не потребує додаткових ресурсів. Проте він має свої обмеження, такі як складність аналізу через великий обсяг інформації в журналах, можливість аналізувати події лише після їх виникнення та ризик видалення записів зловмисником.

Використання статичного методу для захисту вебдодатків має свої відмінності. Наприклад, він не може ефективно аналізувати дані, що передаються методом POST, а також не може проводити повний аналіз HTTP-заголовків. Таким чином, при використанні цього методу для захисту вебдодатків потрібно враховувати певні обмеження та брати їх до уваги під час його впровадження.

Системи виявлення загроз, що використовують сигнатурний підхід, працюють подібно антивірусному ПЗ, сповіщаючи про потенційні небезпеки, коли вони впізнають відповідність з уже відомими атаками у своїй базі сигнатур [36, 37]. Однак, як і у випадках, коли антивірус не може впізнати нові віруси через відсутність або застарілість бази даних, сигнатурні системи виявлення загроз також не можуть впізнати нові небезпеки. Це пояснюється тим, що сигнатури створюються спеціально для відомих атак і не можуть розпізнати навіть невеликі модифікації. Хоча такий підхід зазвичай має малу кількість помилкових сповіщень, він не ефективний проти атак "нульового дня" або поліморфних атак, які залишаються невиявленими, поки відповідні сигнатури не додаються до баз даних.

Для уникнення обмежень, що притаманні традиційним системам виявлення загроз, застосовують методи виявлення відхилень. Цей підхід ґрунтується на аналізі статистичних аномалій, а не на підписах. Процес починається зі створення статистичної моделі, яка описує типові шаблони поведінки системи. Під час

навчання модель вивчає звичайні характеристики ресурсів. Потім вона порівнює кожен новий запит з цією моделлю і виявляє будь-які відмінності, які можуть бути аномаліями. Оскільки атаки зазвичай відрізняються статистично, цей метод може виявити їх, навіть якщо вони не були відомі раніше.

Замість реагування на конкретних користувачів, виявлення аномалій шукає відхилення у поведінці, які можуть вказувати на потенційну атаку. Це полягає в порівнянні звичайної поведінки зі виявленою активністю, яка суттєво відрізняється і може бути ознакою небезпечної діяльності користувачів, що загрожує безпеці вебсайту.

Перевага цих систем полягає в їх здатності раніше виявляти нові види атак, навіть ті, які не були відомі раніше. Оскільки вони базуються на виявленні аномалій, вони можуть впіймати нові атаки, які обмежуються системами, що працюють за сигнатурами. Цю інформацію про атаку можна використовувати для покращення сигнатурних систем.

Проте такі системи мають свої недоліки. Вони часто надто чутливі і можуть спрацьовувати помилково через велику кількість помилок. Особливо це стосується ситуацій, де моніторинг ресурсів непостійний і часто змінюється, наприклад, у вебдодатках. Якщо статистична модель занадто строга і не ураховує зміни, то можливо, що такі системи будуть надто часто генерувати неправдиві сповіщення. Однак на ринку вже доступні деякі продукти, які використовують цей метод виявлення аномалій.

Загальний алгоритм системи виявлення атак (рис. 2.1, 2.2) на вебдодатки включає ряд послідовних етапів, що взаємодіють між собою для забезпечення безпеки введених даних і запобігання потенційним атакам:

– передача параметрів визначає початковий процес, де вебдодаток очікує введення даних від користувача або отримання їх з внутрішньої логіки системи. Вхідні параметри можуть передаватися через різноманітні канали, такі як вебформи, API-запити тощо;

– всі вхідні дані збираються з різних джерел, таких як глобальні масиви, що відповідають методам передачі HTTP, наприклад POST і GET. Ці дані консолідуються в один загальний масив для подальшої обробки;

– кожен параметр з отриманого масиву перевіряється на відповідність певним сигнатурам. Якщо параметр відповідає визначеним сигнатурам, відповідна подія реєструється для подальшого аналізу;

– якщо виявлено збіг з сигнатурою, значення параметра піддається фільтрації для забезпечення безпеки. Це може включати видалення потенційно небезпечних символів або екранування даних для запобігання SQL-ін'єкціям та іншим атакам;

– після завершення всіх перевірок та фільтрації вхідні параметри передаються у вебдодаток для подальшої обробки. Вебдодаток перебуває у стані очікування до завершення аналізу, а після цього виконує необхідні операції з даними.

Система виявлення атак вебдодатків складається з таких частин:

– модуль реєстрації подій для моніторингу активності вебдодатків;

– модуль управління системою виявлення атак;

– модуль формування звіту про виявлені атаки;

– модуль виявлення атак.

Механізм розпізнавання атак фільтрує вхідні дані на проміжному рівні програмного забезпечення, перевіряючи GET та POST параметри запитів через аналіз з використанням регулярних виразів. Регулярні вирази – це спеціальна мова для пошуку та обробки фрагментів тексту, що базується на символах та метасимволах. Вони слугують правилами пошуку певних шаблонів у тексті. Якщо знайдено потенційно небезпечні зразки, middleware-компонент активує відповідні заходи безпеки.

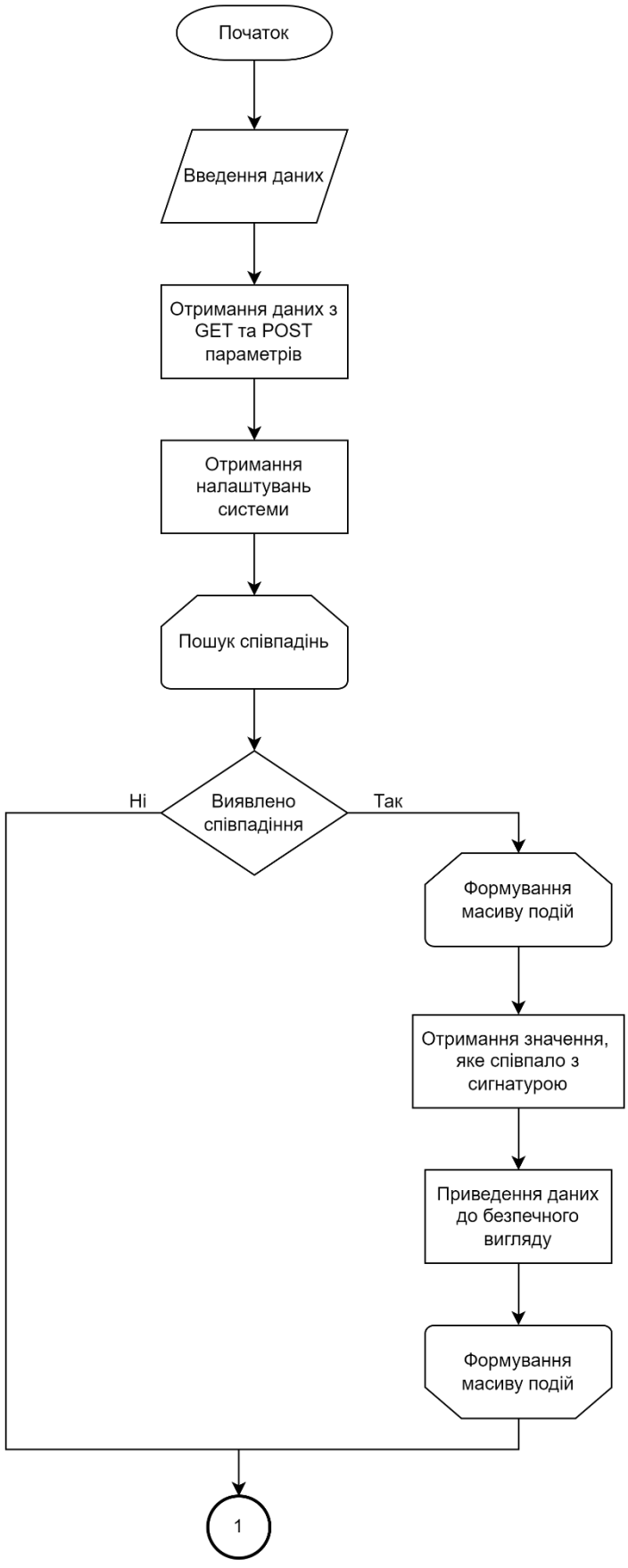


Рисунок 2.2 - Загальний алгоритм системи виявлення атак

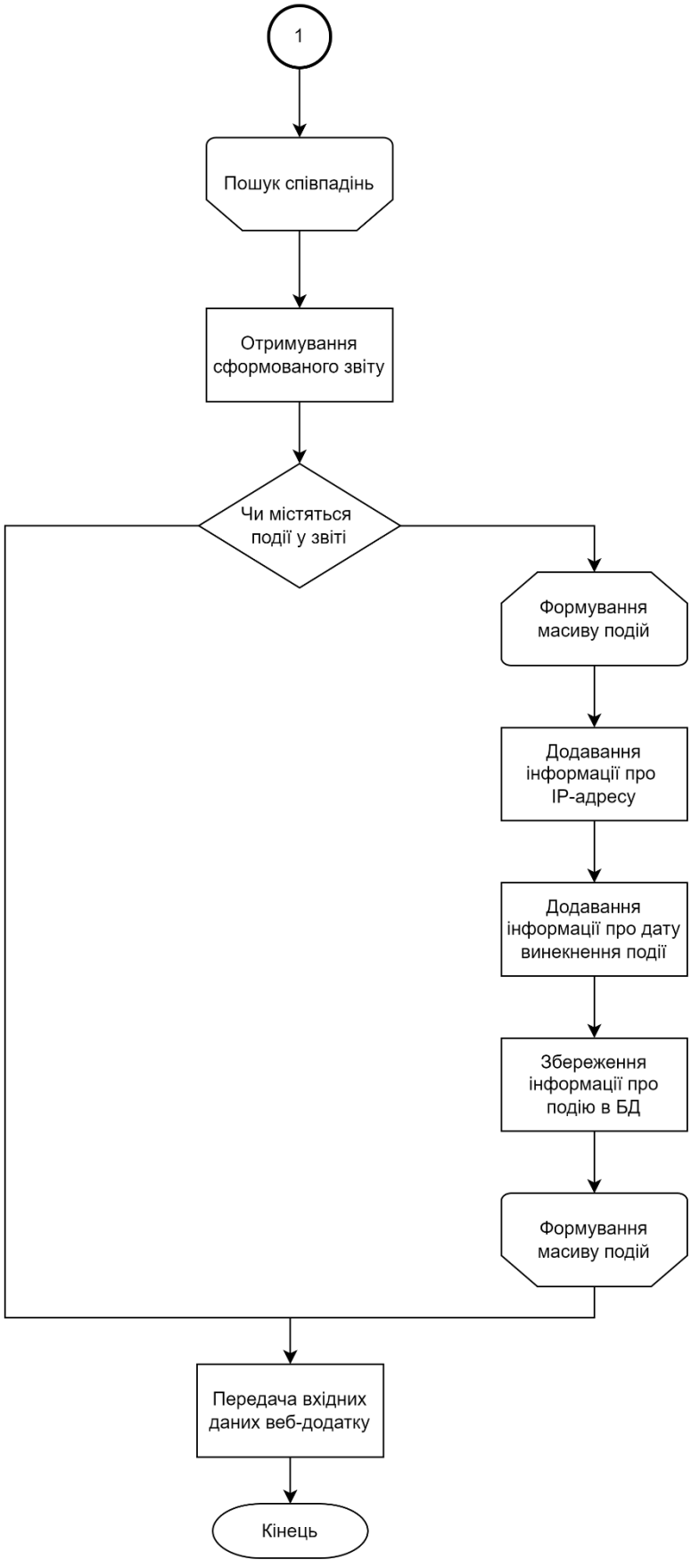


Рисунок 2.3 - Загальний алгоритм системи виявлення атак (продовження)

Метод аналізу клієнтських запитів, відомий як сигнатурний метод, оперує політикою безпеки "чорного списку". Це означає, що лише запити з сигнатурами, які заборонені програмним модулем, будуть заблоковані. Сховище сигнатур містить набір правил (рис. 2.4) для перевірки отриманих від користувача даних у зашифрованому вигляді, щоб запобігти розповсюдженим видам атак, таким як XSS, SQL injection і т.д. Якщо система виявляє нові види атак, їх сигнатури можуть бути додані до сховища для подальшої виявлення подібних запитів. Таким чином, список сигнатур може бути розширений, щоб збільшити ступінь перевірки вхідних даних. Проводиться аналіз таких символів, як апостроф, рядки 'select', 'union all' і 'javascript', знаки ">" та "<", послідовність "../", а також інших виразів, які можуть бути потенційно небезпечними у запитах.

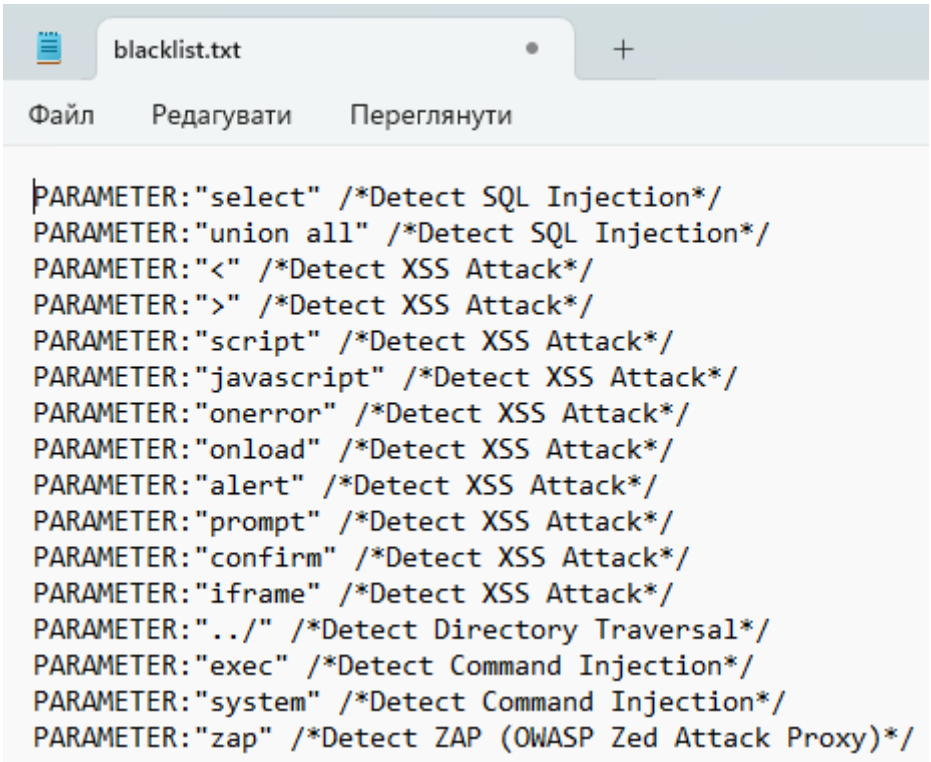


Рисунок 2.4 - Приклади сигнатур

Для захисту від атаки перерахування з метою ускладнення потенційним зловмисникам доступу до конфіденційної інформації про користувачів та структуру каталогів вебсайту, необхідно правильно налаштувати права доступу до

сторінок сайту, а також впровадити механізм блокування надмірних запитів до сервера. Для реалізації такого механізму можна використати алгоритм, який базується на виявленні аномально високої активності з певної IP-адреси користувача. Для цього потрібно встановити ліміт за певний проміжок часу на рівні, наприклад, 1000 запитів за годину для кожного користувача. Далі програмний модуль аналізує кожен запит, підраховуючи кількість запитів, якщо кількість запитів перевищує встановлений поріг, це вважається аномалією після чого, конкретний користувач або IP-адреса тимчасово блокуються на певний проміжок часу. Блокування дозволяє нейтралізувати потенційну загрозу та зменшити ризик успішної атаки на систему. Після закінчення встановленого часу блокування, система автоматично розблокує користувачеві до ресурсів сайту.

У вебдодатках захист від атак перебору паролів вимагає комплексного підходу. Поєднання кількох методів може суттєво зменшити ризик успішної атаки. Першим кроком є вимога до користувачів зберігати сильні паролі, які складаються з різних символів, цифр, регістрів та мають довжину не менше 8 символів. Додавання більш надійних методів аутентифікації, таких як двохфакторна аутентифікація, може значно підвищити рівень безпеки. При цьому важливо обмежити кількість спроб входу до системи (рис. 2.5).

Після виявлення шкідливого запиту, модуль формування звіту приймає управління. В цей момент створюється спеціальний об'єкт події, що описує цей збіг. При створенні об'єкта події передається інформація про параметр, його значення та сигнатуру, з якою відбувся збіг. Крім того, у модулі існує сховище, в якому зберігаються всі події, що виникають при обробці вхідних даних. Таким чином, цей масив містить звіт про всі виниклі події та збіги.

Механізм генерації звітів передає управління до блоку реєстрації подій, який приймає всі накопичені події зі звіту й фіксує їх в системі. Реєстрація подій залучає базу даних для зберігання інформації про всі відбуті події. У цьому блоку відбувається запис інформації про кожну подію у спеціальну таблицю. Цю таблицю використовує блок управління для надання адміністратору системи звіту про стан системи.

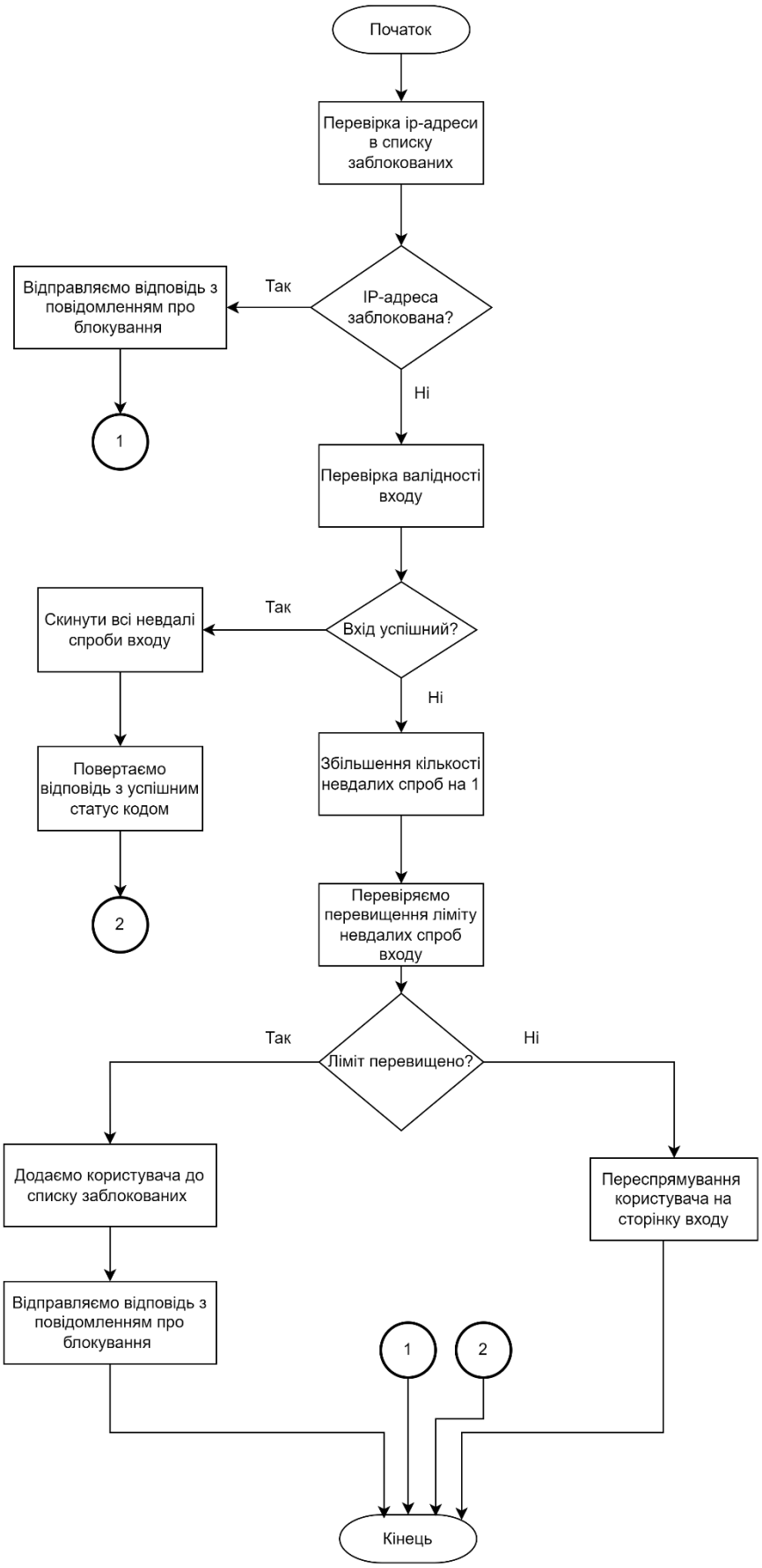


Рисунок 2.5 - Алгоритм захисту від атак перебору паролів

Модуль керування системою виявлення атак представляє собою панель адміністратора, доступну через вебінтерфейс. Для входу до цього модулю потрібно автентифікуватись, після чого адміністратор може керувати системою виявлення атак. Усі дані про зареєстровані події зберігаються в базі даних, з якої модуль управління отримує інформацію для відображення у панелі. Цей модуль забезпечує адміністратора актуальною інформацією про всі зареєстровані події у вигляді таблиці з можливістю сортування та пошуку. Крім того, адміністратор може додавати нові сигнатури для підтримки актуального стану системи виявлення атак.

2.3 Варіанти обходу IDS

Розглянемо різноманітні механізми, які використовуються для уникнення виявлення, відомі як поліморфний код. Цей метод широко використовується для обману систем виявлення вторгнень та антивірусів. Основна ідея полягає у тому, щоб змінювати структуру коду, щоб він залишався функціональним, але не відповідав передбачуваному шаблону. Це може також означати, що код перетворюється кожен раз при запуску програми. Поліморфний код може приймати різні форми, що робить його важким для виявлення. Часто цей підхід включає в себе шифрування корисних фрагментів. Інший метод, відомий як ASCII-кодування, також може використовуватися для ухилення від систем виявлення вторгнень. Використання ASCII-кодування дозволяє представляти символи у вигляді, який не змінює їх функціональності, але ускладнює виявлення. Підхід ASCII-кодування, схожий на поліморфний код, забезпечує виконання певних дій, залишаючись майже невидимим для виявлення. Існує багато інших підходів до обфускації, які дозволяють приховати шкідливі функції під виглядом безпечних.

Одним із методів ухилення від засобів виявлення атак є XSS-поліґлот, який дозволяє зловмиснику мати різні уявлення одного і того ж шкідливого коду. XSS-поліґлоти є популярними серед новачків, оскільки для їх використання потрібно

лише скопіювати та вставити код. Хоча їх легко виявляє будь-який пристойний фільтр або WAF, вони можуть бути корисними для виявлення більшості випадків XSS [38].

В основному це фрагменти коду, які можна виконувати в кількох контекстах у програмі та все ще розглядати як дійсні дані. Вони корисні, оскільки з ними ви можете перевірити елементи керування введенням програми швидше та з меншим шансом бути поміченим. Існують поліглоти XSS, SQLi та файли (SWF, PDF тощо). У складних програмах введення користувача проходить через багато контрольних точок. Маршрут введення може бути від URL-адреси через фільтр, до бази даних і назад до декодера, перш ніж він взагалі відобразиться для перегляду користувачем. На діаграмі нижче показано, як може виглядати цей маршрут введення.

Поліглоти, як засіб ухилення від виявлення атак, стають все більш поширеними в кібербезпеці. Вони дозволяють зловмисникам створювати різні інтерпретації одного шкідливого коду, що ускладнює виявлення його системами безпеки. Це особливо привабливо для початківців, оскільки використання поліглотів не вимагає глибоких технічних знань – достатньо просто скопіювати та вставити код. Хоча фільтр системи захисту може легко виявити поліглоти, вони залишаються корисними для виявлення більшості випадків атак.

Поліглоти можуть бути представлені у вигляді фрагментів коду, які можна виконувати у різних контекстах програми, зберігаючи при цьому свою шкідливість. Це дозволяє швидше та з меншим ризиком бути поміченим здійснювати перевірку введення користувача на елементах управління програми. На практиці, поліглоти можуть існувати для різних типів атак, таких як XSS, SQLi, а також для різних форматів файлів, таких як SWF або PDF. У складних програмах шлях введення користувача може пройти через кілька контрольних точок, від URL-адреси через фільтри, до бази даних і назад до декодера перед тим, як він буде відображений для користувача. Такий маршрут введення може бути складним та повним перешкод для виявлення атак.

Щоб краще зрозуміти конструкцію поліглотів, розберемо один із них:

jaVasCript:/*-/*`/*\`/*'/*"/**/(/**/oNcliCk=alert())//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e

Для початку давайте розіб'ємо поліглот на його перші частини.

– jaVasCript: мітка в ECMAScript; інакше схема URI;

– /*-/*/*`/*’/*”/**/ – полілінійний коментар у ECMAScript; послідовність розривних літералів;

– (/* */oNcliCk=alert()) – зона виконання, укладена в дужки виклику;

– //%0D%0A%0d%0a// – однорядковий коментар у ECMAScript; подвійний CRLF у заголовках HTTP-відповідей;

– //</stYle/</titLe/</teXtarEa/</scRipt/--!> – послідовність, яка розриває теги HTML;

– x3csVg/<sVg/oNloAd=alert()//>x3e – елемент svg.

Аналогічно JavaScript, у мові SQL також можна створювати поліглоти, що ускладнюють виявлення атак. Наприклад, дослідник безпеки Матіас Карлссон навів такий приклад коду:

SLEEP(1) /*’ or SLEEP(1) or’” or SLEEP(1) or “*/

Це поліглотне корисне навантаження працює в контексті одинарних лапок, подвійних лапок і навіть у контексті "прямого запиту".

Існує ще один метод обходу, відомий як сеансовий сплайсинг. Одні IDS не відновлюють сеанс до порівняння зі зразком. Наприклад, якщо зловмисник затримує пакети настільки довго, що IDS перестає їх збирати, він може пройти мимо системи контролю IDS, приховуючи це корисне навантаження. Це лише один із способів ухилення. Зловмисники роблять атаки невидимими для IDS, щоб пройти повз них. Тепер перейдемо до іншої техніки ухилення – фрагментації пакетів.

Ми збираємося розглянути кілька різних способів обходу IDS шляхом розбиття IP-датаграм. Іншими словами, ми фрагментуватимемо пакети. Уявіть, що зловмисник

намагається обійти IDS, у якого тайм-аут фрагментації складає 15 секунд. Тут ми говоримо про тайм-аут повторного складання IP-фрагменту. Це стосується максимальної кількості часу, протягом якого фрагмент буде зберігатися в розібраному вигляді, перш ніж закінчиться термін його дії і він буде скинутий. Розглянемо будь-який хост, що розмовляє TCP з кількома фрагментами, що проходять через нього, які необхідно зібрати заново. Цей тайм-аут повторного складання фрагмента – це те, що не дає йому нескінченно утримувати незавершені фрагменти. Припустимо, ми маємо 15-секундний тайм-аут на IDS. Потім у нас є машина жертви з 30-секундним тайм-аутом повторного складання фрагментації. Зловмисник збирається відправити фрагмент номер один, який зберігатиметься в IDS, а потім передаватиметься жертві. У цьому фрагменті немає нічого поганого. Машина жертви також містить фрагмент номер один. Далі відбувається те, що зловмисник чекає більше 15 секунд, тому що через 15 секунд IDS збираються видалити фрагмент. Потім зловмисник повертається та відправляє фрагмент номер два у IDS, який потім проходить на машину жертви. IDS має фрагмент номер два і тільки номер два, тому що він втратив номер один. Але жертва має обидва фрагменти, і обидва фрагменти разом утворюють шкідливе корисне навантаження. IDS ніколи не бачив усе це разом, тому що він відкидав перший фрагмент після цих початкових 15 секунд. Отже, при атаці з фрагментацією зловмисник просто руйнує цей пакет і переконується, що IDS ніколи не отримає все це та зможе ідентифікувати шкідливе корисне навантаження.

Такий вид атаки використовується, коли час очікування на IDS коротший, ніж на машині-жертві. Однак зловмисник може впровадити цей метод і навпаки: коли час очікування на IDS довший, ніж на жертві. Зловмисник розбиває пакети, надсилає достатню кількість пакетів із шкідливим навантаженням, чекає, поки час очікування фрагментації на жертві буде перевищено, а потім надсилає додаткові пакети протягом цього періоду. IDS, які підмінені, примушують IDS відкидати весь пакет. Цей метод працює навіть у зворотньому напрямку, оскільки жертва – єдина машина, що врешті-решт підтримує всі фрагменти із корисним навантаженням, тоді як IDS може відкидати частину шкідливого навантаження або отримувати недійсні фрагменти.

Порядкові номери TCP допомагають упорядковувати наші пакети, і вони роблять це завдяки комбінації порядкових номерів, номерів підтвердження і розміру самого пакета. Коли все працює правильно, все шикується. Номер підтвердження стає сумою порядкового номера та довжини попереднього пакета. Але що відбувається, коли він не вишиковується правильно? Ну, це називається атакою фрагментів, що перекриваються. Назва говорить сама за себе, але давайте пройдемося нею. У цьому сценарії давайте знову додамо нашого зловмисника, нашу IDS та нашу жертву. У звичайних умовах зловмисник збирається надіслати дані до IDS, а потім знову відправити їх у вигляді фрагментованого пакета. Але ми маємо три фрагменти, і всі наші порядкові номери збігаються. IDS, природно, передасть їх на машину жертви і пакет буде повторно зібраний на іншому кінці. Але приберемо це і спробуємо трохи по-іншому. Ми збираємось змусити зловмисника відправити фрагменти таким чином, щоб вони перекривалися. Хитрість атаки з фрагментами, що перекриваються, полягає в тому, що різні машини будуть збирати фрагменти, що перекриваються, по-різному. Скажімо, пріоритет фрагмента три та фрагмента два над фрагментом один, але коли вони передаються на комп'ютер жертви, він робить це навпаки. Він віддає пріоритет першому фрагменту, а не трьом і двом, і ці фрагменти не лише перекриваються, а й вирушають не по порядку. Те, як ці машини збирають їх разом, визначає різницю між відновленням законного пакета, що містить шкідливе корисне навантаження, і відновленням пакета, який відкидається, оскільки він виглядає спотвореним. Хитрість полягає в побудові перекриваються фрагментів таким чином, що IDS – це той, хто не бачить шкідливе корисне навантаження в пакеті, а жертва – той, хто реконструює ці фрагменти, що перекриваються таким чином, щоб виконати вміст.

У цьому сценарії ми знову маємо наш зловмисник, нашу IDS і нашу жертву. Тепер атакуючий збирається відправити фрагмент із високим значенням TTL. Далі цей фрагмент пройде весь шлях до жертви. Значення TTL досить велике, щоб воно досягло пункту призначення. Потім зловмисник збирається відправити ще один фрагмент, але цього разу з низьким TTL та поганим корисним навантаженням. TTL буде досить високим, щоб досягти IDS, але досить маленьким, щоб не досягти жертви. Потім він відправить ще один

фрагмент, також із високим TTL, і він дійде до жертви. Зрештою, зловмисник може повторно відправити другу частину, але цього разу з розумним корисним навантаженням та високим значенням TTL, щоб вона пройшла весь шлях до жертви. Хитрість тут у тому, що IDS, отримавши це шкідливе корисне навантаження з початкового фрагмента з низьким TTL, ігноруватиме пакет. Це не виглядає дозволеним, але оскільки жертва ніколи не отримувала погане корисне навантаження, вона отримувала лише узгоджені фрагменти, а потім збирала ці фрагменти в пакет, який містив шкідливе корисне навантаження. Вся причина примусу IDS до читання неприпустимих пакетів іноді відома як атака вставкою, і це спроба заплутати її та приховати основну атаку, яка існує у цих інших заборонених пакетах.

Є багато інших способів обійти IDS, і один із цих методів може бути реалізований за допомогою атак типу «відмова в обслуговуванні». Атака DoS не обов'язково повинна відключати систему, і вона може призвести до того, що людям буде складно виконувати свою роботу. Ось приклад: зловмисник може вирішити залити IDS трафіком, який змусить IDS видавати попередження. Цей трафік, можливо, ніколи не призначався для використання будь-якого прихованого ризику в системі, але якщо IDS відправляє цілу купу попереджень адміністраторам, і вони зайняті їхньою обробкою, зловмиснику набагато легше прослизнути серед цього великого діапазону загроз. Існує ще одна атака, призначена для експлуатації цільової системи; іноді це називається атакою з флудом, тому що ви заливатимете IDS попередженнями. У минулому існували інструменти, такі як Stick та Snot, спеціально призначені для того, щоб IDS генерувала попередження. Оповіщення можуть ускладнити ідентифікацію законних атак через велику кількість шуму. Ви також можете побачити цю атаку, яка називається False Positive Generation. Так, IDS генерує оповіщення, ні, це не сповіщення про будь-які дії, які дійсно збираються використовувати систему, що стоїть за IDS. Генерація помилкових спрацьовувань як пов'язує адміністративні ресурси, а й обмежує здатність системи виявлення вторгнень перевіряти трафік. Це може призвести до відмови в

обслуговуванні IDS або, якщо генерується достатньо трафіку, це може призвести до вичерпання ємності диска.

2.4 Висновок

В даному розділі ми детально проаналізували архітектуру вебдодатку та перейшли до розробки програмного модуля для виявлення атак. Були розглянуті найбільш поширені методи виявлення атак на вебдодатки, включаючи їх особливості, переваги та недоліки. Також було розглянуто різні способи обходу систем виявлення атак на сервер. Цей аналіз надасть нам основу для подальшої розробки та вдосконалення програмного модуля виявлення атак, що є важливим етапом в забезпеченні безпеки вебдодатків.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ВИЯВЛЕННЯ АТАК НА ВЕБДОДАТКИ

3.1 Вибір інструментів

Досить важливим етапом розробки програмного продукту є правильний вибір засобів, за допомогою яких буде можливість реалізації необхідного функціоналу. Також якісні засоби розробки зможуть полегшити процес написання коду завдяки зручному інтерфейсу.

Python – сучасна мова програмування, яка працює на всіх популярних операційних системах. Мова програмування Python розроблюється вже більше 20 років. На сучасному етапі найбільш широко використовуються дві версії Python – більш пізня версія 2 і сучасна версія 3. Версія 2 більше не розвивається, проте досі використовується, оскільки дуже багато програмного забезпечення і бібліотек було розроблено саме для цієї версії. Між версіями не існує сумісності, в тому числі в синтаксисі команд введення-виведення, хоча вони багато в чому дуже схожі.

Python – це сучасний універсальний інтерпретована мова програмування. До його переваг відносять:

– багатоплатформність;

– доступність;

– простий синтаксис;

– великі можливості написання коротких та зрозумілих програм;

– багата стандартна бібліотека;

– легкість вивчення;

– динамічна типізація;

– підтримка об’єктно-орієнтованого програмування;

– відсутність витоків пам’яті;

– велика кількість готових модулів, які роблять процес написання коду набагато простішим.

– стандартна бібліотека, що встановлюється разом з Python має можливості для роботи з операційними системами, вебдодатками, базами даних, різними форматами даних, а також для оформлення графічного інтерфейсу програм тощо.

Pycharm є однією з найбільш інтелектуальних та популярних IDE мови програмування Python з повним набором засобів для якісної розробки. На даний момент є дві версії Pycharm:

– PyCharm Community Edition – безкоштовна версія;

– PyCharm Professional Edition, яка є платною версією та підтримує більший набір можливостей.

Pycharm виконує перевірку коду під час його написання, також має функцію автодоповнення, в тому числі опираючись на дані, які були раніше отримані під час виконання кода.

До ключових можливостей PyCharm відносять:

Потужний та багатофункціональний редактор коду з підсвічуванням синтаксису, автоформатуванням та автовідступами для мов програмування, які підтримуються IDE:

– проста та потужна навігація в коді;

– допомога під час написання коду;

– швидкий перегляд документації будь-якого елемента прямо у вікні редактору;

– велика кількість інспекцій коду;

– функціональний графічний відладчик;

– повна підтримка нових версій фреймворку Django;

– інтегроване Unit тестування;

– підтримка Flask фреймворку, а також мов Mako та Jinja2;

– багатоплатформність;

– велика та постійно зростаюча кількість плагінів.

У процесі розробки програмного модуля виявлення атак використовується система управління базами даних (СУБД) PostgreSQL для збереження та

управління даними на серверній стороні. PostgreSQL є однією з найпотужніших та надійних вільно розповсюджуваних СУБД з відкритим вихідним кодом, і має безліч переваг для розробників та підприємств [39, 40]:

– славиться своєю надійністю та стійкістю до відмов. Вона підтримує механізми транзакцій та контролю цілісності даних, що робить її ідеальним для додатків, які потребують надійної роботи;

– може легко розширюватися, дозволяючи розробникам додавати нові функції та розширення за допомогою розширень та процедур бази даних;

– має вбудовану підтримку геоданих, що робить її ідеальним вибором для додатків, які вимагають обробки географічних даних, таких як визначення маршрутів та відстеження геолокації;

– має відкритий вихідний код, що дозволяє розробникам адаптувати її до своїх потреб і вносити власні модифікації;

– підтримує стандарт ACID (Atomicity, Consistency, Isolation, Durability) для гарантування цілісності даних та коректності транзакцій;

– має активну спільноту розробників і користувачів, що означає, що завжди є доступ до допомоги, документації та оновлень;

Використання PostgreSQL дозволяє ефективно зберігати та керувати даними, які стосуються користувачів, замовлень, операцій оплати та багатьох інших аспектів нашого додатку. Велика продуктивність та надійність PostgreSQL роблять його ідеальним вибором для проектів, які потребують збереження та обробки великої кількості даних.

Django – високорівневий вебфреймворк, який дає можливість швидкого створення безпечних та зручних вебресурсів [41]. Даний фреймворк є безкоштовним та має відкритий джерельний код та чудову документацію для роботи. Спільнота, яка користується фреймворком постійно збільшується.

На звичайному інформаційному ресурсі вебдодаток очікує HTTP-запит від браузера. В момент отримання запиту, додаток розробляє те, що є необхідним на базі URL-адреси й, імовірно, даних в POST або GET запитах. В залежності від того,

що необхідно, далі він може читати або записувати інформацію із бази даних або виконувати інші задачі. Після цього додаток повертає відповідь браузеру, часто динамічно розроблюючи сторінку HTML для демонстрації в браузері, вставляючи отримані дані в HTML-шаблон.

Вебдодаток, написаний на Django, зазвичай групує код, який оброблює кожний з кроків в окремі файли (рис. 3.1):

Обробка запитів з різних URL-адрес може здійснюватися однією функцією, однак значно ефективніше написати окремі функції для обробки кожного конкретного ресурсу. Для цього використовується URL-маршрутизатор, який спрямовує HTTP-запити до відповідних функцій відображення на основі URL-адреси запиту. Додатково, URL-маршрутизатор здатен вилучати дані з URL-адреси згідно з визначеними шаблонами та передавати ці дані до відповідних функцій відображення у вигляді аргументів. Це дозволяє забезпечити більш гнучку і зрозумілу структуру коду.

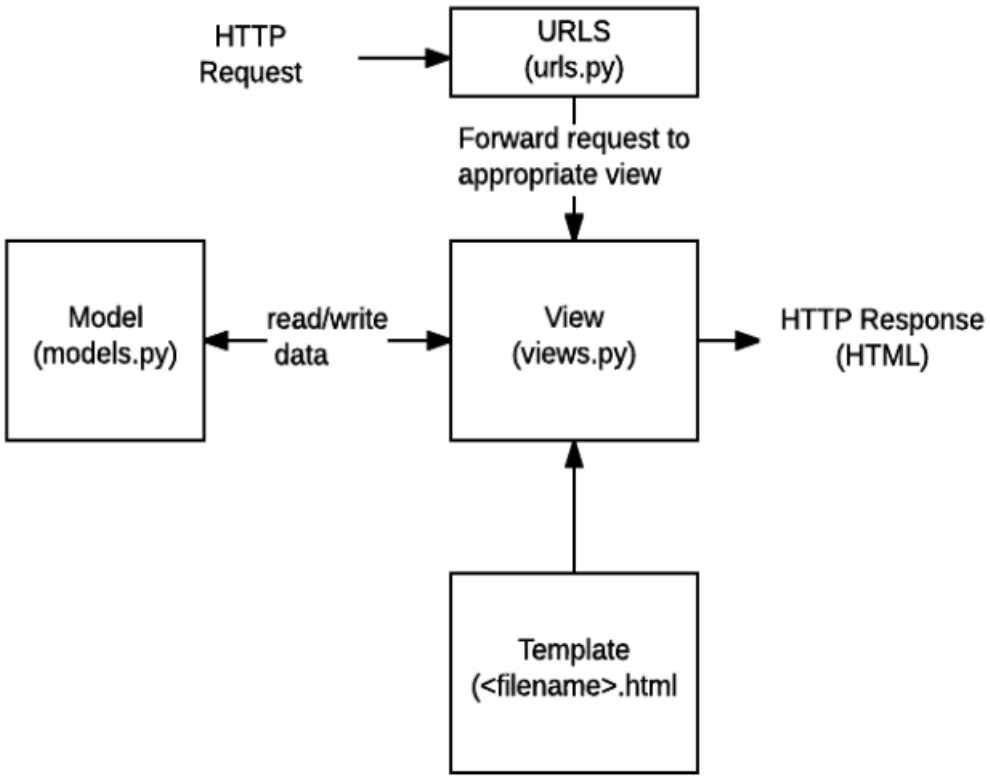


Рисунок 3.1 – Порядок виконання HTTP запиту [42]

View слугує функцією обробником запитів, яка отримує HTTP-запит і повертає відповіді. Функція view має доступ до даних, які є необхідні для задоволення запитів і делегує відповіді в шаблони через моделі.

Models являють собою об'єкти Python, які формулюють структуру даних додатку і надає механізми для керування і виконання запитів в базу даних.

Template – це файл, який формулює структуру сторінки (до прикладу css або html файл). View має можливість динамічно створювати HTML сторінки, беручи HTML-шаблони і заповнюючи їх інформацією з моделі.

Docker – це програмне забезпечення дозволяє використовувати віртуальні контейнери з додатковими функціями [43, 44]. Даний проєкт має основний контейнер з образом СУБД проєкту (Postgres 14), образом інструментарію пошуку у backend-частини проєкту та інструментарію адміністрування бази даних.

В нашому проекті використовується Docker Desktop 4.22.0 для Windows (рис. 3.2).

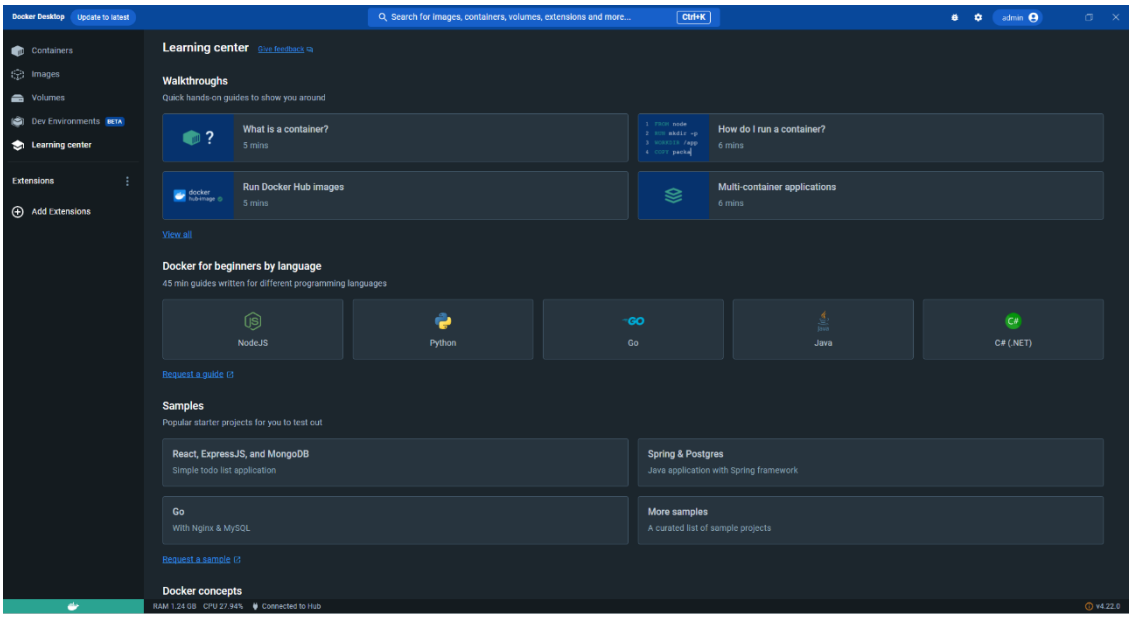


Рисунок 3.2 – Інтерфейс Docker

За допомогою Docker ми отримуємо:

– ізольованість програми на рівні процесу, що дозволяє безпечно виконувати небезпечний код;

– приховування процесів, оскільки для кожного контейнера можна встановити власні методи обробки даних, забезпечуючи конфіденційність фонових процесів;

– швидке розгортання, оскільки можна використовувати готові образи замість встановлення додаткових компонентів; просте масштабування, дозволяючи легко розширювати проект за допомогою нових контейнерів;

– зручний запуск програми на будь-якому docker-хості;

– оптимізацію файлової системи за рахунок складання образів з шарів, що дозволяє ефективно використовувати файлову систему.

3.2 Налаштування проекту та проектування бази даних

Для початку створимо порожній проект використовуючи команду django-admin startproject ids_system. Результатом виконання є створена папка ids_system зі структурою Django проекту (рис. 3.3).

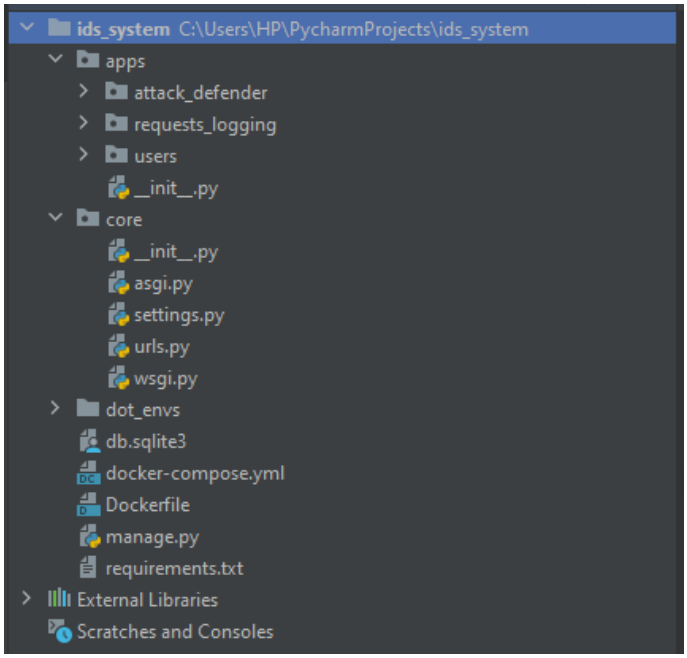


Рисунок 3.3 – Структура проекту

Створимо базу даних PostgresSQL та таблиці Users, UserGroups, UserPermissions, RequestLog, Blacklist (рис. 3.4).

Таблиця Users буде містити в собі дані про користувачів сайту, включаючи особисту інформацію, статус облікового запису та права доступу. Вона слугує для зберігання та управління даними користувачів, забезпечуючи автентифікацію та авторизацію на сайті.

Таблиця UserGroups дозволяє керувати доступом користувачів до різних функціональних частин сайту або групувати їх за певними параметрами. Кожна група може мати деякі специфічні атрибути або права, які спільно належать їм усім.

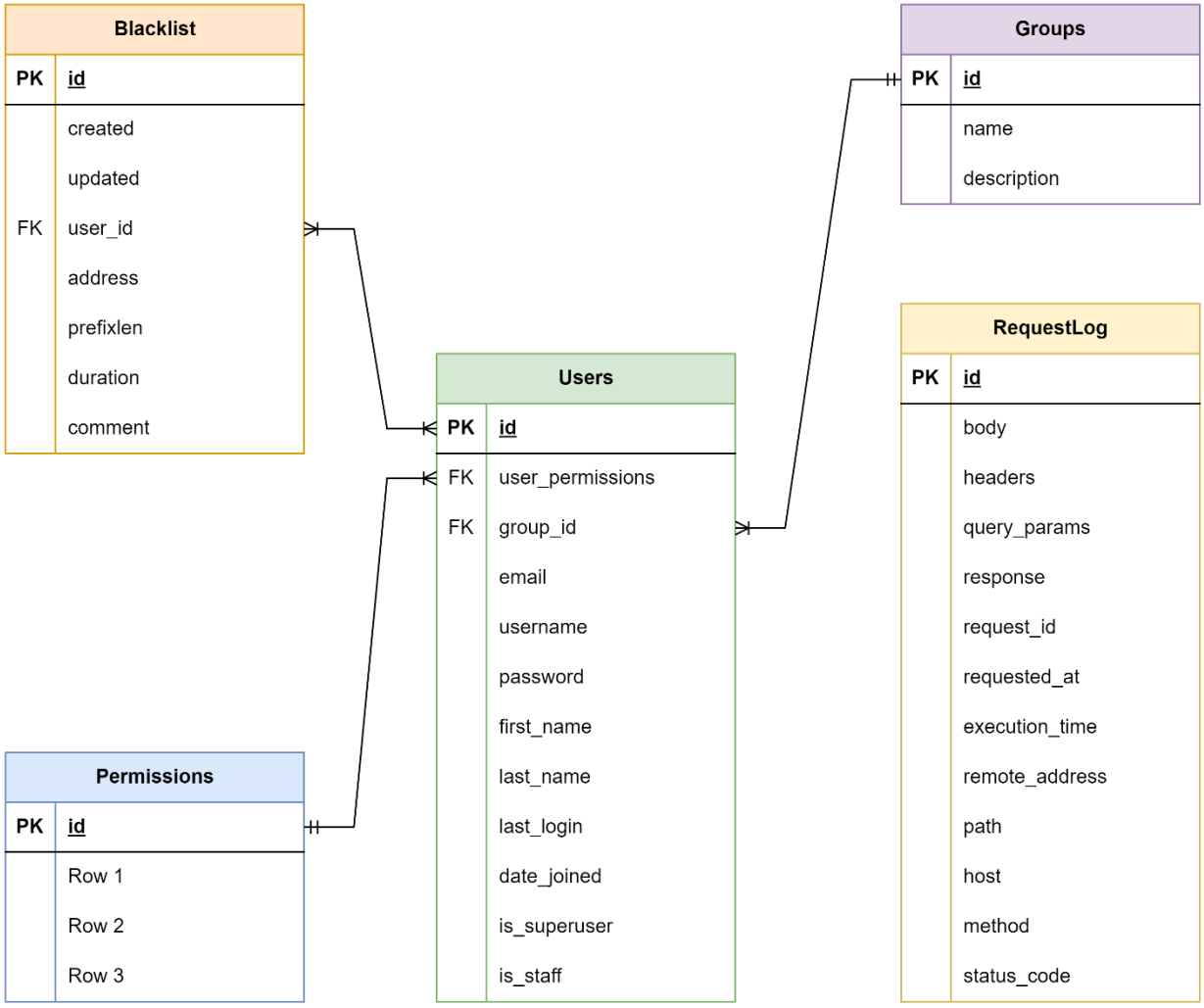


Рисунок 3.4 – Діаграма бази даних модуля виявлення атак

Таблиця UserPermissions відповідає за керування правами доступу користувачів. Кожен дозвіл визначає конкретну дію або функціонал, до якого користувач може мати доступ, такий як перегляд, редагування, видалення тощо. Ці дозволи можуть бути призначені окремим користувачам або групам користувачів.

Таблиця RequestLog використовується для зберігання інформації про всі вхідні запити до API вебресурсу. Це дозволяє адміністратору відстежувати та аналізувати активність користувачів, виявляти потенційні проблеми та забезпечувати безпеку системи.

Таблиця Blacklist потрібна для зберігання інформації про заборонені IP-адреси та користувачів, яким заборонений доступ до системи.

Створимо конфігурацію для файлу .env (рис. 3.5), надамо йому дані для підключення до БД.

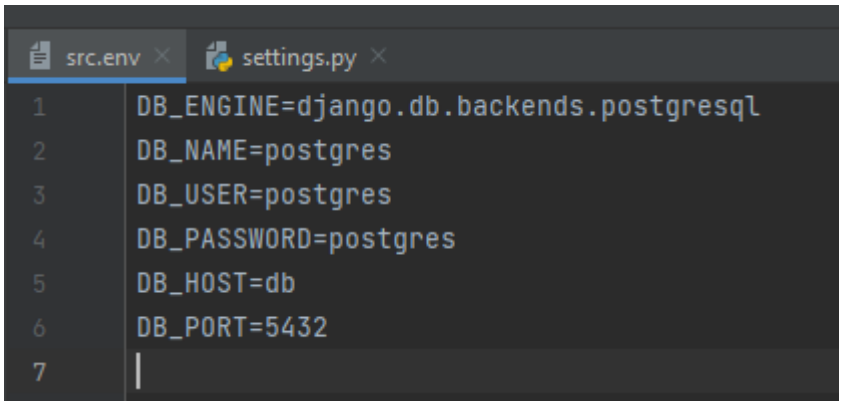


Рисунок 3.5 – Зміні віртуального середовища

Поле DB_ENGINE визначає тип бази даних, яка використовується в проекті. В нашому випадку це база даних PostgresSQL.

Поле DB_NAME вказує на назву бази даних з якою ми будемо взаємодіяти.

Поле DB_HOST містить адресу, за якою розміщується сервер бази даних. Однак, оскільки ми використовуємо Docker контейнер для бази даних, ми вказуємо ім'я контейнера, яке використовується для доступу до бази даних у середовищі Docker.

Поле DB_PORT відповідає за порт, через який ми будемо підключатись до бази даних.

Поля DB_USERNAME та DB_PASSWORD містять логін і пароль для підключення до бази даних.

Тепер, коли ми підготували Django проект та налаштували віртуальне середовище, настав час перейти до Docker. Ця технологія дозволить нам упакувати наш проект та всі його залежності у контейнери, що забезпечить легке та безпечне розгортання проекту на будь-якому сервері або хмарній платформі. Створимо в директорії проекту два файли: DockerFile та docker-compose.yml. Dockerfile містить список інструкцій для образу, інакше кажучи, що відбувається в середовищі контейнера (рис. 3.6).

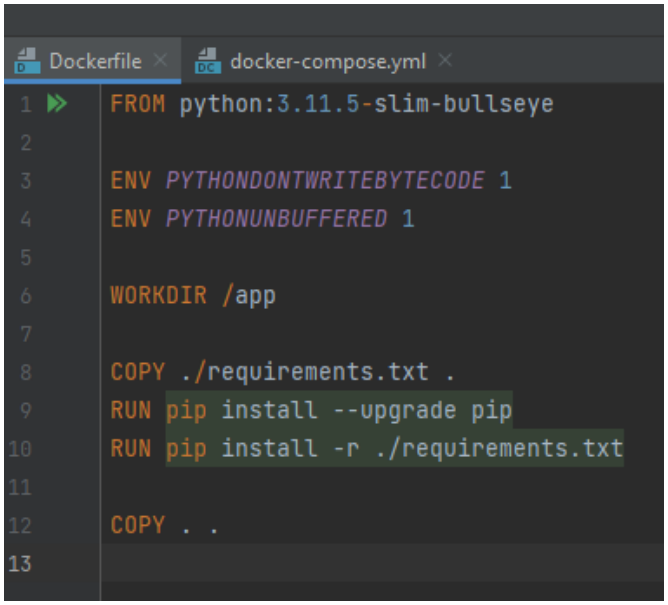


Рисунок 3.6 – Налаштування Dockerfile

Ми використовуємо офіційний Docker-образ для Python 3.11, щоб створити контейнер для нашого Django-проекту. Змінні середовища PYTHONDONTWRITEBYTECODE та PYTHONUNBUFFERED налаштовані таким чином, щоб уникнути створення файлів .pyc та забезпечити негайний вивід логів. Робочий каталог встановлено в /app. Ми копіюємо requirements.txt,

оновлюємо pip, встановлюємо залежності і копіюємо весь код проекту. Це забезпечує ефективне середовище для запуску нашого Django-проекту.

Тепер нам залишилося сворити найважливіший файл – docker-compose.yaml. docker-compose.yaml – це файл конфігурації для Docker Compose, інструмента, що дозволяє вам описувати та запускати багатоконтейнерні додатки з легкістю. Відкриємо текстовий редактор та створимо новий файл з назвою docker-compose.yaml у кореневій папці проекту.

Цей Docker Compose файл налаштовує три сервіси для Django-проекту:

Сервіс db використовує образ postgres:16.1-alpine для запуску бази даних PostgreSQL. Всі дані бази даних зберігаються в томі ids_pgdata, що забезпечує стійкість даних між перезапусками контейнера. Змінні середовища для бази даних завантажуються з файлу db.env.

Сервіс redis використовує стандартний образ redis для запуску сервера Redis. Дані Redis також зберігаються в томі ids_redis, що дозволяє зберігати дані між перезапусками контейнера.

Сервіс app будується з поточного контексту і запускає сервер Django. Він має доступ до мереж ids_pgdata і ids_redis. Порт 8000 контейнера мапується на порт 8000 хоста для доступу до додатку. Весь код проекту змонтовується в /app контейнера. Команда command виконує послідовність дій: міграцію бази дани та запуск сервера Django. Сервіс app залежить від сервісів db і redis для правильного запуску.

Перейдемо тепер до налаштування системи автентифікації користувачів на основі JWT токенів. JWT (JSON Web Token) – це стандарт для створення доступу для RESTful вебслужб в безстандартному способі, оскільки вони використовують JSON для передачі даних. JWT складається з трьох частин: заголовка, корисних навантажень та підпису (рис. 3.7). Для початку потрібно встановити необхідну бібліотеку за допомогою команди pip install djangorestframework_simplejwt.

Наступним кроком необхідно додати rest_framework_simplejwt до списку встановлених додатків у вашому Django проекті. Крім того, у файлі settings.py потрібно налаштувати параметри для використання JWT токенів.

Рисунок 3.7 – Структура jwt токену [45]

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_simplejwt',
    'apps.requests_logging',
    'apps.users',
    "apps.attack_defender",
    'blacklist',
]

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=1),
```

```
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'TOKEN_USER_CLASS': 'rest_framework_simplejwt.models.TokenUser',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True
}
```

Далі необхідно вказати Django, яку модель користувача необхідно використовувати для автентифікації AUTH_USER_MODEL = 'users.User'. Тепер підключимо кінцеві точки для токенів JWT у файл urls.py.

```
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path(route="api/v1/users/", view=include(arg="apps.users.urls")),
]
```

Middleware є важливим компонентом у веброзробці, що діє як проміжний шар між запитом і відповіддю. У Django middleware може використовуватися для

виконання різних завдань, таких як обробка запитів перед їх передачею до представлення або маніпуляція відповідями перед їх відправленням клієнту. Один із поширених випадків використання – це логування вхідних запитів та блокування шкідливих запитів.

Клас LoggingMiddleware логує детальну інформацію про кожен вхідний запит, включаючи метод запиту, шлях, заголовки, параметри запиту, тіло, дані відповіді, код статусу та час виконання. Ця інформація зберігається у моделі APIRequestLog, яка забезпечує збереження записів про всі залоговані запити.

Клас SignatureDetectionMiddleware реалізує захист від підозрілих запитів шляхом перевірки параметрів запиту на наявність заборонених сигнатур за допомогою регулярних виразів.

Додамо LoggingMiddleware та SignatureDetectionMiddleware до конфігурації проекту, включаючи їх у список MIDDLEWARE.

Для захисту від атак перерахування, можна використати бібліотеку ratelimit у Django для обмеження кількості запитів, які можуть бути виконані користувачем за певний час. Встановлюючи ліміт запитів на певний період, наприклад, 100 запитів за годину для кожного користувача, можна ефективно запобігти зловживанню та зменшити ризик успішних атак на систему.

```
@ratelimit(key='ip', rate='100/h', block=True)
def retrieve(self, request, pk) -> Response:
    instance = get_object_or_404(self.queryset, pk=pk)
    self.check_object_permissions(request, instance)
    serializer = self.serializer(instance)
    return Response(serializer.data, status=status.HTTP_200_OK)
```

У цьому прикладі ми використовуємо декоратор ratelimit для функції, яка відповідає за отримання інформації про користувача. Параметр key='ip' вказує, що ліміт застосовується до кожного IP-адреси. Параметр rate='100/h' встановлює ліміт

в 1000 запитів на годину. Параметр block=True вказує, що якщо ліміт буде перевищений, буде виконано блокування користувача.

3.3 Тестування системи

Щоб провести тестування системи виявлення атак на Django, нам знадобляться такі інструменти, як Kali Linux, VirtualBox, ngrok та zap proxy.

Спочатку відкриємо програму VirtualBox та запустимо віртуальну машину з операційною системою Kali Linux (рис 3.8).

Рисунок 3.8 – Налаштування віртуальної машини

Паралельно на основній машині запустимо Django проект у Docker контейнері (рис. 3.9) та відкриємо порт 8000 через ngrok (рис. 3.10).

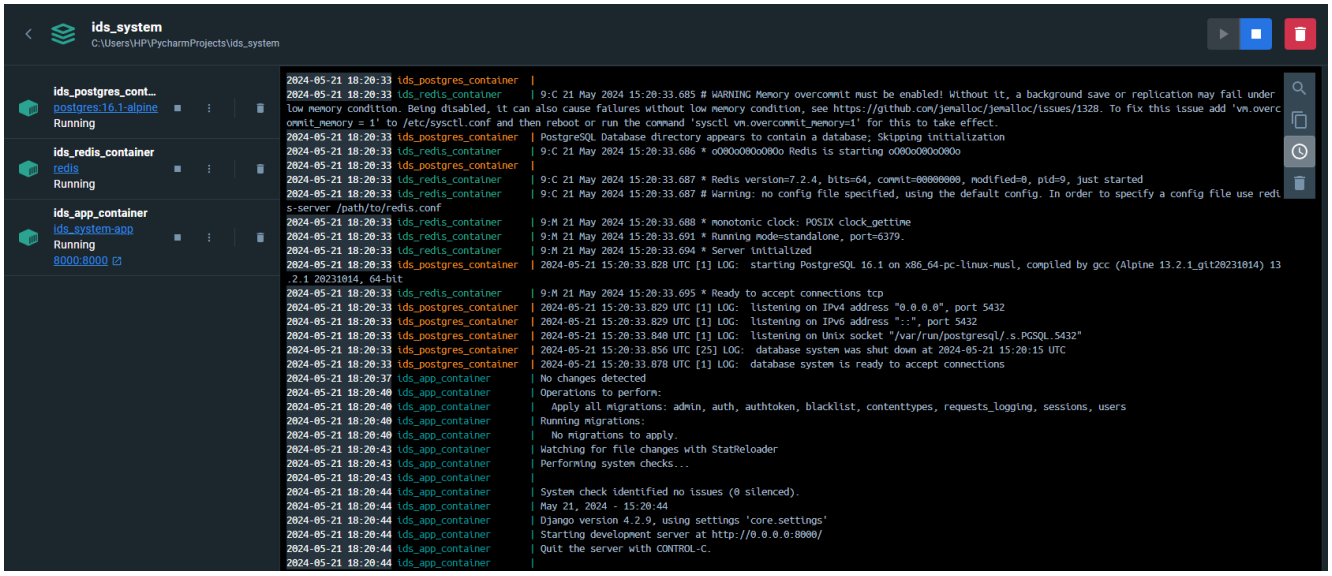


Рисунок 3.9 – Запуск докер контейнеру

Рисунок 3.10 – Запуск ngrok

Після того як ми запустили проект в докері, віртуальну машину для здійснення атаки та прокинули порти ми відкриваємо сканер вразливостей zap proxy та вказуємо адрес, який ми отримали в ngrok (рис. 3.11).

Рисунок 3.11 – Запуск сканера ZAP

Zap розпочинає свою діяльність із фази сканування, що включає в себе аналіз вебдодатку з метою ідентифікації наявних директорій та ресурсів (рис. 3.12). Цей процес зазвичай використовує методи, такі як перебірка, інтелектуальний аналіз та інші техніки для систематичного огляду структури вебдодатку з метою виявлення можливих точок входу або недоліків безпеки.

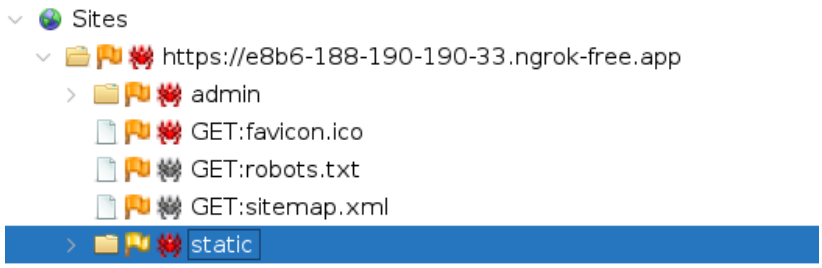


Рисунок 3.12 – Виявлена структура каталогів сайту

Після успішного сканування ZAP Proxy переходить до етапу аналізу вразливостей, де він створює розширені запити та перехоплює відповіді сервера для виявлення можливих слабкостей у вебдодатку (рис. 3.13). Цей процес може включати тестування на вразливості, такі як кросс-сайтовий скриптинг, ін'єкції SQL, витік конфіденційної інформації та інші атаки.

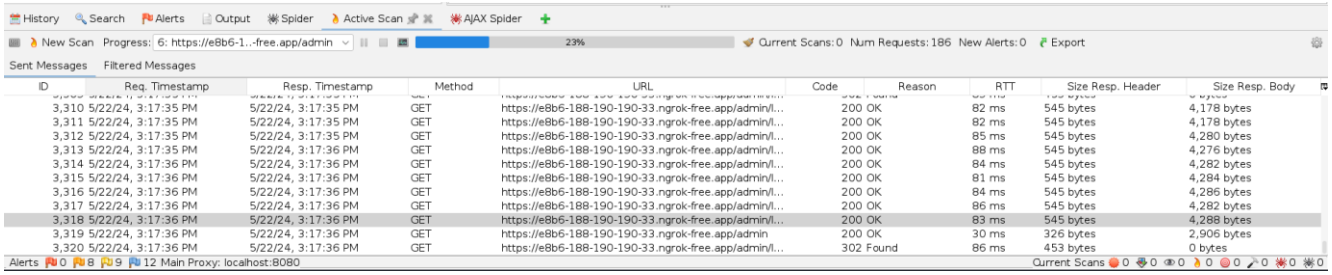


Рисунок 3.13 – Сканування вразливостей сайту

Під час тестування ZAP Proxy надсилає різноманітні запити, які спрямовані на виявлення вразливостей у вебдодатку. Це включає запити для тестування на кросс-сайтовий скриптинг, SQL ін'єкції, витік конфіденційної інформації та інші можливі атаки. ZAP створює запити з ін'єкцією шкідливих скриптів чи SQL-коду

(рис. 3.14), а також аналізує відповіді серверу, щоб виявити можливі слабкості. Крім того, він може модифікувати параметри запитів та заголовки HTTP для оцінки стійкості додатку до різних видів атак, зокрема аналізує реакцію на неправильні чи підозрілі запити.

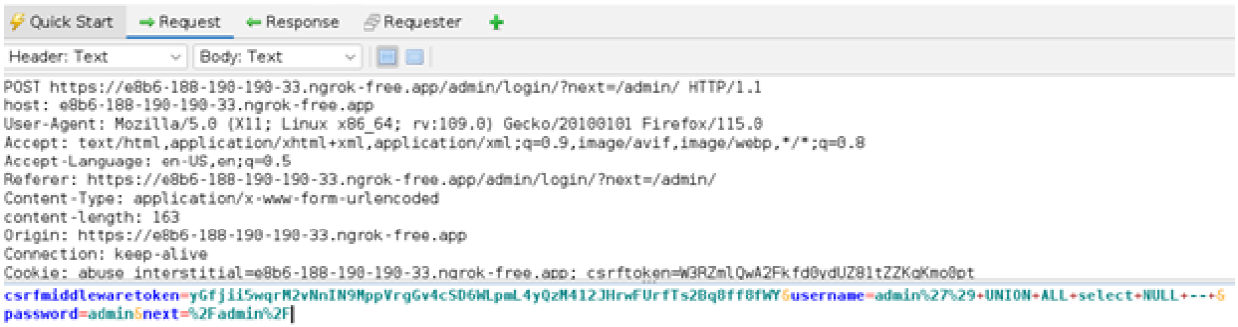


Рисунок 3.14 – Шкідливий запит до вебдодатку

Атакований сервер отримав підозрілий запит, який містив шкідливі параметри та виявив спробу здійснення атаки. У відповідь на це сервер негайно заблокував IP-адресу користувача, який надіслав шкідливий запит (рис. 3.15).

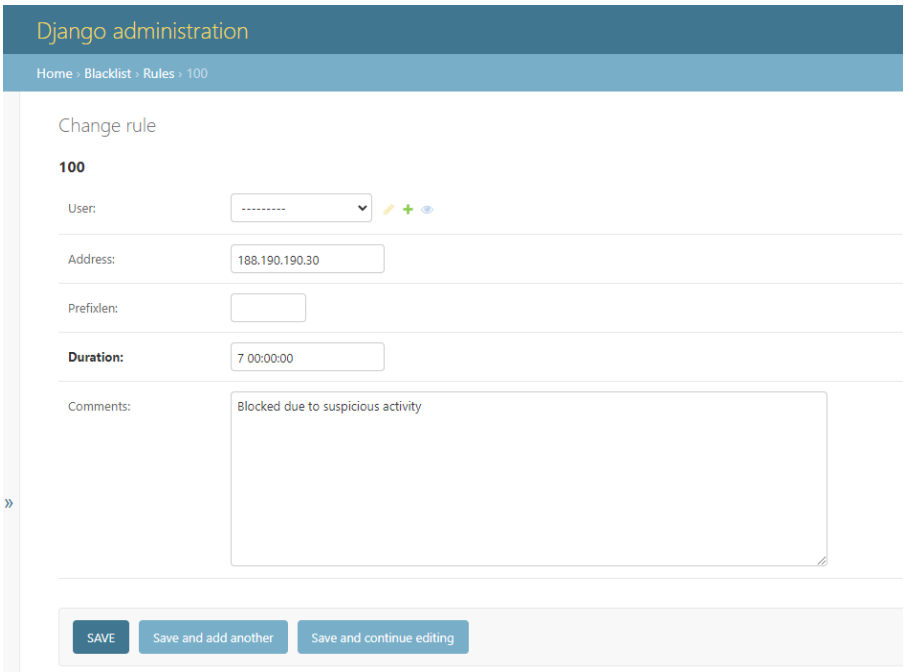


Рисунок 3.15 – Перегляд заблокованого користувача

Відповідно до налаштованих політик безпеки, цей IP-адрес було внесено до чорного списку, що призвело до блокування користувача та запобігло подальшим спробам взаємодії з сервером з цієї IP-адреси (рис. 3.16).

Рисунок 3.16 – Блокування входу

3.4 Висновок до розділу

У цьому розділі ми обрали найефективніші інструменти для розробки нашого проекту. Використання сучасних технологій гарантує не лише швидкість, а й надійність процесу розробки, а також забезпечує ефективне управління даними та просте розгортання проекту.

Після встановлення необхідних інструментів ми розпочали реалізацію програмного модуля для виявлення атак. Спочатку ми спроектували структуру бази даних, а потім створили та налаштували проект за допомогою Django. Для забезпечення контейнеризації нашого додатку та його залежностей ми створили Dockerfile та docker-compose.yml. Автентифікація користувачів у системі здійснювалася за допомогою токенів JWT. Після цього було створено проміжне програмне забезпечення для реєстрації запитів та захисту від шкідливих атак. Ми обмежили кількість запитів, які користувач може надіслати до API протягом певного проміжку часу.

Під час тестування системи на вразливості за допомогою інструментів, таких як Kali Linux, VirtualBox, ngrok та zap proxy, було виявлено, що zap не зміг виявити жодної вразливості та система успішно справилася з виявленням шкідливих запитів. Заходи безпеки дозволили системі ідентифікувати та блокувати користувачів, які здійснювали атаку, запобігаючи подальшим можливим загрозам.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи було проведено дослідження предметної області та інформаційних загроз. Зокрема, були розглянуті різні типи атак на вебсервери, сканери вразливостей, а також потенційні наслідки таких атак. На основі проведеного аналізу був розроблений програмний модуль для виявлення атак, що дозволяє підвищити рівень безпеки вебресурсів.

Метою проекту є створення ефективного інструменту для моніторингу та виявлення шкідливих запитів до вебсерверу, що забезпечить своєчасне реагування на потенційні загрози та зниження ризиків компрометації інформаційної системи.

У ході виконання кваліфікаційної роботи було досліджено структуру та різні типи архітектур вебдодатків. Розглянуто методи виявлення атак, зокрема сигнатурний метод, який був обраний для реалізації в програмному модулі. Окрім того, було вивчено різноманітні механізми, що використовуються для обходу систем виявлення атак.

Під час розробки програмного модуля ми обрали необхідні інструменти та покроково описали налаштування і процес розробки проекту. Після завершення розробки було проведено тестування системи, де вона успішно справилася з виявленням шкідливих запитів. Заходи безпеки дозволили системі ідентифікувати та блокувати користувачів, які здійснювали атаки, запобігаючи подальшим загрозам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. NIST Special Publication 800-53 Revision 5 (2020). Security and Privacy Controls for Information Systems and Organizations. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf (дата звернення: 01.03.2024).

2. Про захист інформації в інформаційно-комунікаційних системах: Закон України від 05.07.1994 р. № 80/94-ВР. URL: https://zakon.rada.gov.ua/laws/show/80/94- %D0%B2%D1%80#Text (дата звернення: 04.03.2024).

3. Про захист персональних даних: Закон України (Відомості Верховної Ради України (ВВР), 2010, № 34, ст. 481) URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text 4. Про державну таємницю: Закон України від 21.01.1994 р. № 3855-XII. URL:https://zakon.rada.gov.ua/laws/show/3855-12#Text (дата звернення: 04.03.2024).

4. ДСТУ ISO/IEC 27032:2016 (ISO/IEC 27032:2012, IDT). Інформаційні технології. Методи захисту. Настанови щодо кібербезпеки. – Чинний від 2016-27-12. – Київ : ДП «УкрНДНЦ», 2018. – [50] с. (дата звернення: 07.03.2024).

5. Web Application Risk – the Threat of and Solution to Sensitive Data Exposure. URL: https://www.immuniweb.com/blog/OWASP-sensitive-data-exposure.html. (дата звернення: 09.03.2024).

6. ДСТУ ISO/IEC 27032:2016. Інформаційні технології. Методи захисту. Настанови щодо кібербезпеки. – Чинний від 2016-27-12. – Київ : ДП «УкрНДНЦ», 2018. – [50] с. (дата звернення: 10.03.2024).

7. Гапак О. М., Болога С.І. Захист інформації в комп’ютерних системах : підручник. Ужгород : ДВНЗ «Ужгородський національний університет», 2021. 184 с. (дата звернення: 12.03.2024).

8. OWASP (Open Web Application Security Project). Список найпоширеніших ризиків для вебдодатків. URL: https://owasp.org/www-project-top-ten/. (дата звернення: 12.03.2024).

9. Киричок Р. В., Ахтьоров В.Ю. Аналіз шкідливого програмного забезпечення як метод протидії кібератакам. Матеріали науково-практичної інтернетконференції «Цифрова трансформація кібербезпеки». К.: ДУТ, 2020. 88-91 с. (дата звернення: 14.03.2024).

10. Тарнавський Ю.А. Технології захисту інформації: підручник. К.: КПІ ім. Ігоря Сікорського, 2018. 162 с. (дата звернення: 15.03.2024).

11. Захист вебсервісів: лабораторний практикум І.А. Терейковський, Л.О. Терейковська, К.О. Радченко, С.О. Гнатюк. – Київ: КПІ ім. Ігоря Сікорського, 2018 URL: https://ela.kpi.ua/bitstream/123456789/22234/1/Zahist_web_servisiv_Laboratornyi_praktikum.pdf (дата звернення: 15.03.2024).

12. Янчев А.В. Контроль інформаційної безпеки в системах електронного документообігу. Бізнес Інформ 2015. 199-204с. URL: http://www.business-inform.net/export_pdf/business-inform-2015-4_0-pages199_204.pdf (дата звернення: 16.03.2024).

13.OWASP CheatSheetSeries. URL: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md. (дата звернення: 20.03.2024).

14. SQL Injections Top Attack Statistics. URL: https://www.darkreading.com/cyber-risk/sql-injections-top-attack-statistics. (дата звернення: 22.03.2024).

15. Best Practices to prevent SQL-injections. URL: https://tableplus.com/blog/2018/08/best-practices-to-prevent-sql-injection-attacks.html. (дата звернення: 22.03.2024).

16. Duchene F. Xss vulnerability detection using model inference assisted evolutionary fuzzing. / F. Duchene, R. Groz, S. Rawat, and J. L. Richier // IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012. 815–817p. (дата звернення: 24.03.2024).

17. Guaman D. Implementation of techniques and owasp security recommendations to avoid sql and xss attacks using j2ee and ws-security / D. Guaman,

F. Guam´an, D. Jaramillo, M. Sucunuta // 12th Iberian Conference on Information Systems and Technologies (CISTI), 2017. 1–7p. (дата звернення: 25.03.2024).

18. Cross-Site Request Forgery (CSRF). URL: https://www.owasp.org/index.php/CrossSite_Request_Forgery_(CSRF). (дата звернення: 26.03.2024).

19. Testing for CSRF (OTG-SESS-005). URL: https://www.owasp.org/index.php/Testing_for_CSRF_(OTG-SESS-005). (дата звернення: 27.03.2024).

20. WhiteHat Security's Approach to Detecting Cross-Site Request Forgery (CSRF). URL: https://www.whitehatsec.com/ - (дата звернення: 01.04.2024)..

21. Cross site request forgery (CSRF). URL:https://learn.snyk.io/lesson/csrf-attack/ (дата звернення 20.05.2023 р) (дата звернення: 02.04.2024).

22. Directory traversal. URL: https://learn.snyk.io/lesson/directory-traversal. (дата звернення: 04.04.2024).

23. Path Traversal. URL: https://owasp.org/wwwcommunity/attacks/Path_Traversal (дата звернення: 05.04.2024).

24. Тестування на проникнення. URL: https://krlabs.com.ua/blog/testuvannya-na-pronyknennya-pentest-vid-a-do-ya/. (дата звернення: 07.04.2024).

25. Professional Pen Testing for Web Applications (Programmer to Programmer)/ Andres Andreu. – Wrox, 2016. 548 p. (дата звернення: 08.04.2024).

26. The Web Applicaion Hackers's Handbook: Finding and Exploiting Security Flaws/D. Stuttard, M. Pinto. - John Wiley & Sons, Inc, 2011. 877 p. (дата звернення: 10.04.2024).

27. OWASP ZAP. URL: https://www.zaproxy.org/docs/ (дата звернення: 10.04.2024).

28. ZAP Penetration Testing Tutorial to Detect Vulnerabilities. URL: https://toobler.medium.com/zap-penetration-testing-a-simple-tutorial-to-detect-vulnerabilities-e9a8311182a9 (дата звернення: 10.04.2024).

29. Nessus. URL: https://www.techtarget.com/searchnetworking/definition/Nessus. (дата звернення: 11.04.2024).

30. Tenable Inc. Nessus 8.0.x User Guide / Tenable Inc., 2019. 396 p. (дата звернення: 15.04.2024).

31. Khawaja G. Practical Web Penetration Testing: Secure web applications using Burp Suite, Nmap, Metasploit, and more. – Packt Publishing, 2018. 294 p. (дата звернення: 17.04.2024).

32. Nikto2. URL: https://cirt.net/Nikto2 (дата звернення: 17.04.2024).

33. Дослідження вразливостей Web-сайтів та методів їх усунення. URL: http://phone.kpi.ua/wpcontent/uploads/2014/06/4.pdf (дата звернення: 18.04.2024).

34. Web Application Architecture: The Latest Guide 2024. URL: https://www.clickittech.com/devops/web-application-architecture/ (дата звернення: 20.04.2024).

35. Системи моніторингу та управління безпекою. URL: http://integritysys.com.ua/security/siem/. (дата звернення: 21.04.2024).

36. А. Завада, О. Самчишин, В. Охрімчук, "Аналіз сучасних систем виявлення атак і запобігання вторгненням", Інформаційні системи, Житомир: Збірник наукових праць ЖВІ НАУ, 2012. 97-106р. (дата звернення: 22.04.2024).

37. І.Терейковський, А.Корченко, Т.Паращук, Є.Педченко, Аналіз відкритих систем виявлення вторгнень, DOI: 10.18372/2225-5036.24.13431, УДК 004.056.53(045), 2018. 201-216с. (дата звернення: 25.04.2024).

38.Building XSS Polyglots. URL: https://brutelogic.com.br/blog/building-xss-polyglots/. (дата звернення: 27.04.2024).

39. Postgres документація. URL: https://www.postgresql.org/docs/. (дата звернення: 03.05.2024).

40. PostgreSQL Configuration : Best Practices for Performance and Security – Berkley, United States: aPress, 2020. 226 p. (дата звернення: 07.05.2024).

41. Security in Django, Django Software Foundation. URL:

https://docs.djangoproject.com/en/2.2/topics/security/. (дата звернення: 08.05.2024).

42. An introduction to Django URL: https://medium.com/@sundaram.2911/an-introduction-to-django-b17b51e3a7dc (дата звернення: 10.05.2024).

43. Docker документація. URL:https://docs.docker.com/get-started/. (дата звернення: 12.05.2024).

44. Docker : Complete Guide To Docker For Beginners And Intermediates, 2020. 140 p (дата звернення: 14.05.2024).

45. Components of JWTs Explained. URL: https://fusionauth.io/articles/tokens/jwt-components-explained (дата звернення: 17.05.2024).

ДОДАТОК А

Фрагменти програмного коду системи виявлення атак

Middleware для логування усіх вхідних запитів та відповіді від сервера:

```
import string
import random
from django.utils.timezone import now
from apps.requests_logging.models import APIRequestLog
from apps.requests_logging.utils import is_allowed_url


class LoggingMiddleware:
    def __init__(self, get_response):
        self.log = {}
        self.get_response = get_response

    def __call__(self, request):
        self.log = {"requested_at": now()}
        response = self.get_response(request)
        is_allowed_url(request.path_info)
        request_id = generate_random_string(length=24)
        print(request.META)
        response.data['requestId'] = request_id
        client_ip = self._get_client_ip(request)
        self.log.update(
            {
                "request_id": request_id,
                "method": request.method,
                "path": request.path_info,
                "host": request.get_host(),
                "remote_address": client_ip,
                "body": request.POST,
                "headers": dict(request.headers),
                "query_params": request.GET,
                "response": response.data,
                "status_code": response.status_code,
                "execution_time": self._get_response_ms(),
            }
        )
        self.handle_log()
        return response

    def _get_client_ip(self, request):
        if request.META.get('HTTP_X_FORWARDED_FOR'):
            ip = request.META.get('HTTP_X_FORWARDED_FOR')
        else:
            ip = request.META.get('REMOTE_ADDR')
```

```
        return ip

    def _get_response_ms(self):
        response_timedelta = now() - self.log["requested_at"]
        response_ms = int(response_timedelta.total_seconds() * 1000)
        return max(response_ms, 0)

    def handle_log(self):
        APIRequestLog(**self.log).save()


def generate_random_string(length=24):
    alphabet = string.ascii_lowercase + string.digits
    return ''.join(random.choices(alphabet, k=length))
```

Django модель для зберігання інформації про вхідний запит:

```
from datetime import datetime
from django.db import models


class APIRequestLog(models.Model):
    request_id = models.CharField(max_length=24)
    requested_at = models.DateTimeField(default=datetime.now, db_index=True)
    execution_time = models.PositiveIntegerField(default=0)

    remote_address = models.GenericIPAddressField()
    path = models.CharField(
        max_length=200,
        db_index=True,
        help_text="url path",
    )
    host = models.URLField()
    method = models.CharField(max_length=10)

    body = models.JSONField(null=True, blank=True)
    headers = models.JSONField(null=True, blank=True)
    query_params = models.JSONField(null=True, blank=True)
    response = models.JSONField(null=True, blank=True)

    status_code = models.PositiveIntegerField(db_index=True)

    class Meta:
        verbose_name = "API Request Log"

    def __str__(self):
        return f"{self.method} {self.path}"
```

Програмний модуль виявлення атак:

```
import re
from django.http import JsonResponse
from apps.attack_defender.utils import load_blacklisted_keywords
from blacklist.models import Rule

import ipaddress
import threading
from datetime import datetime, timedelta
from typing import Optional, Dict, Union
import logging

from django.core.exceptions import SuspiciousOperation
from django.db.models import F, Max, DateTimeField
from django.utils.deprecation import MiddlewareMixin
from django.utils.timezone import now
from django.shortcuts import render
from django.conf import settings



logger = logging.getLogger(__name__)


_RELOAD_PERIOD = timedelta(seconds=getattr(settings,
'BLACKLIST_RELOAD_PERIOD', 60))


_user_blacklist: Dict[int, datetime] = {}
_addr_blacklist: Dict[Optional[int], Dict[Union[ipaddress.IPv4Network,
ipaddress.IPv6Network], datetime]] = {}

_loaded: Optional[datetime] = None

_blacklist_lock = threading.Lock()


class Blacklisted(SuspiciousOperation):
    pass


class BlacklistMiddleware(MiddlewareMixin):
    def process_request(self, request):
        if getattr(settings, 'BLACKLIST_ENABLE', True):
            current_time = now()
```

```
        if _needs_reload(current_time):
            _load_blacklist()

        try:
            _filter_client(request, current_time)

        except Blacklisted as exception:
            template_name = getattr(settings, 'BLACKLIST_TEMPLATE', None)

            if template_name:
                if getattr(settings, 'BLACKLIST_LOGGING_ENABLE', True):
                    logger.warning(exception)

                context = {'request': request, 'exception': exception}
                return render(request, template_name, context, status=400)

            raise


++def _filter_client(request, current_time):
    user = request.user
    user_id = user.id

    addr = _get_client_address(request)

    # no logging here, because the event will be logged either by the caller, or by django.request

    until = _user_blacklist.get(user_id)
    if until is not None and until > current_time:
        raise Blacklisted('Blacklisted user: %s' % user.username)

    for prefixlen, blacklist in _addr_blacklist.items():
        network = Rule(address=addr, prefixlen=prefixlen).get_network()

        until = blacklist.get(network)
        if until is not None and until > current_time:
            raise Blacklisted('Blacklisted address: %s' % addr)


def _get_client_address(request):
    source = getattr(settings, 'BLACKLIST_ADDRESS_SOURCE', 'REMOTE_ADDR')

    if request.META.get('HTTP_X_FORWARDED_FOR'):
        ip = request.META.get('HTTP_X_FORWARDED_FOR')
    else:
        ip = request.META.get('REMOTE_ADDR')

    return ip
```

```
def _needs_reload(current_time):
    return _loaded is None or current_time >= _loaded + _RELOAD_PERIOD


def _load_blacklist():
    global _loaded, _user_blacklist, _addr_blacklist

    with _blacklist_lock:
        current_time = now()

        if _needs_reload(current_time):
            until = Max(F('created') + F('duration'), output_field=DateTimeField())
            rules = Rule.objects.values('user_id', 'address', 'prefixlen').annotate(until=until)
            rules = rules.filter(until__gt=current_time)

            user_blacklist = {}
            addr_blacklist = {}

            for rule in rules:
                user_id = rule['user_id']
                prefixlen = rule['prefixlen']
                network = Rule(address=rule['address'], prefixlen=prefixlen).get_network()
                until = rule['until']

                _add_client(user_blacklist, addr_blacklist, user_id, prefixlen, network, until)

            _user_blacklist = user_blacklist
            _addr_blacklist = addr_blacklist
            _loaded = now()


def _add_client(user_blacklist, addr_blacklist, user_id, prefixlen, network, until):
    if user_id is not None:
        current_until = user_blacklist.get(user_id)
        if current_until is None or current_until < until:
            user_blacklist[user_id] = until

    if network is not None:
        blacklist = addr_blacklist.setdefault(prefixlen, {})
        current_until = blacklist.get(network)
        if current_until is None or current_until < until:
            blacklist[network] = until


def _add_rule(rule):
    global _addr_blacklist
```

```
    with _blacklist_lock:
        user_id = rule.user_id
        prefixlen = rule.prefixlen
        network = rule.get_network()
        until = rule.get_expires()

        addr_blacklist = _addr_blacklist.copy()
        _add_client(_user_blacklist, addr_blacklist, user_id, prefixlen, network, until)
        _addr_blacklist = addr_blacklist



class SignatureDetectionMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response
        self.blacklisted_keywords = load_blacklisted_keywords()

    def __call__(self, request):
        response = self.get_response(request)
        return response

    def process_view(self, request, view_func, view_args, view_kwargs):
        if request.method == 'GET':
            params = request.GET
        elif request.method == 'POST':
            params = request.POST
        else:
            params = {}

        for param_name, param_value in params.items():
            if self.contains_signature(param_value):
                self.block_user(request)
                return JsonResponse(
                    {"error": "Signature detected in parameter: {}".format(param_name)},
                    status=400
                )
        return None

    def contains_signature(self, data):
        for keyword in self.blacklisted_keywords:
            # Створюємо регулярний вираз для пошуку ключового слова
            pattern = re.compile(r'\b' + re.escape(keyword) + r'\b', re.IGNORECASE)
            # Перевіряємо, чи є ключове слово у рядку
            if pattern.search(data):
                return True
        return False

    def block_user(self, request):
        ip = self.get_client_ip(request)
```

```
        # Додаємо правило блокування IP-адреси
        Rule.objects.create(
            address=ip,
            duration=timedelta(days=7),  # Блокуємо на 7 днів
            comments="Blocked due to suspicious activity"
        )

    def get_client_ip(self, request):
        if request.META.get('HTTP_X_FORWARDED_FOR'):
            ip = request.META.get('HTTP_X_FORWARDED_FOR')
        else:
            ip = request.META.get('REMOTE_ADDR')
        return ip
```

Налаштування проекту:

```
from datetime import timedelta
from pathlib import Path
import environ

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

env = environ.Env()
environ.Env.read_env(env_file=f"{BASE_DIR}/dot_envs/src.env")


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-sq2!ecm-
ys=w!m0!#b2*u#kj25m@gpwhwx&^5ykmk68wl481d%'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = ["*"]


# Application definition
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

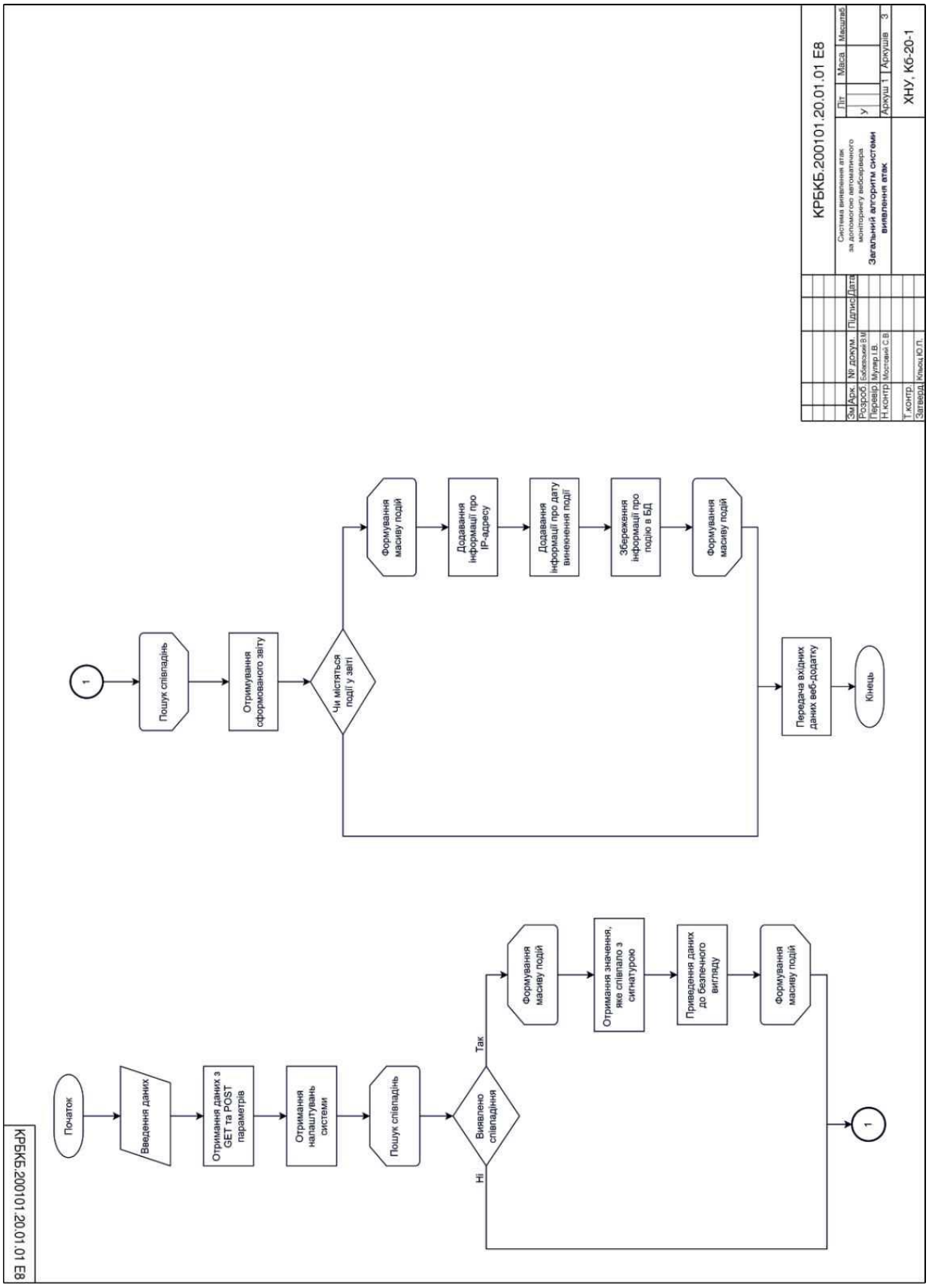
```
    'rest_framework',
    'rest_framework.authtoken',
    'rest_framework_simplejwt',
    'apps.requests_logging',
    'apps.users',
    "apps.attack_defender",
    'blacklist',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'apps.attack_defender.middleware.BlacklistMiddleware',
    "apps.attack_defender.middleware.SignatureDetectionMiddleware",
    'apps.requests_logging.middleware.LoggingMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'core.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'core.wsgi.application'


# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    "default": {
        "ENGINE": env("DB_ENGINE"),
```

```
        "NAME": env("DB_NAME"),
        "USER": env("DB_USER"),
        "PASSWORD": env("DB_PASSWORD"),
        "HOST": env("DB_HOST"),
        "PORT": env("DB_PORT"),
    }
}


# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]


# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

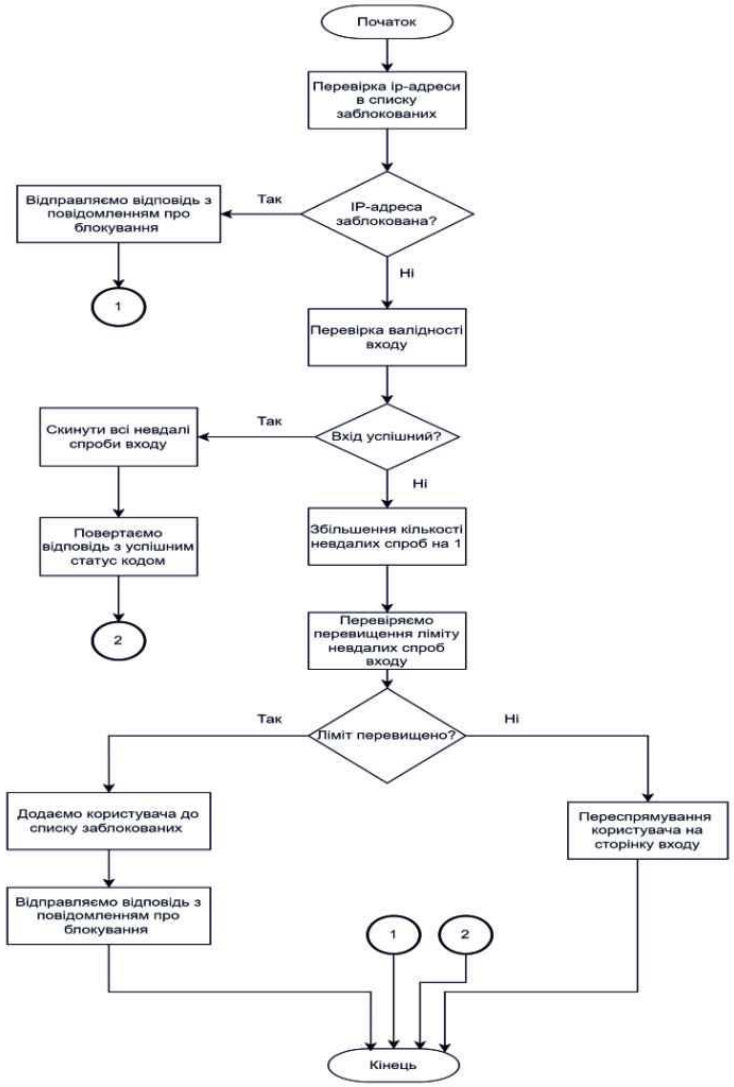
```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}

# User
AUTH_USER_MODEL = 'users.User'

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=1),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'TOKEN_USER_CLASS': 'rest_framework_simplejwt.models.TokenUser',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True
}
```

ДОДАТОК Б

Копії графічної частини

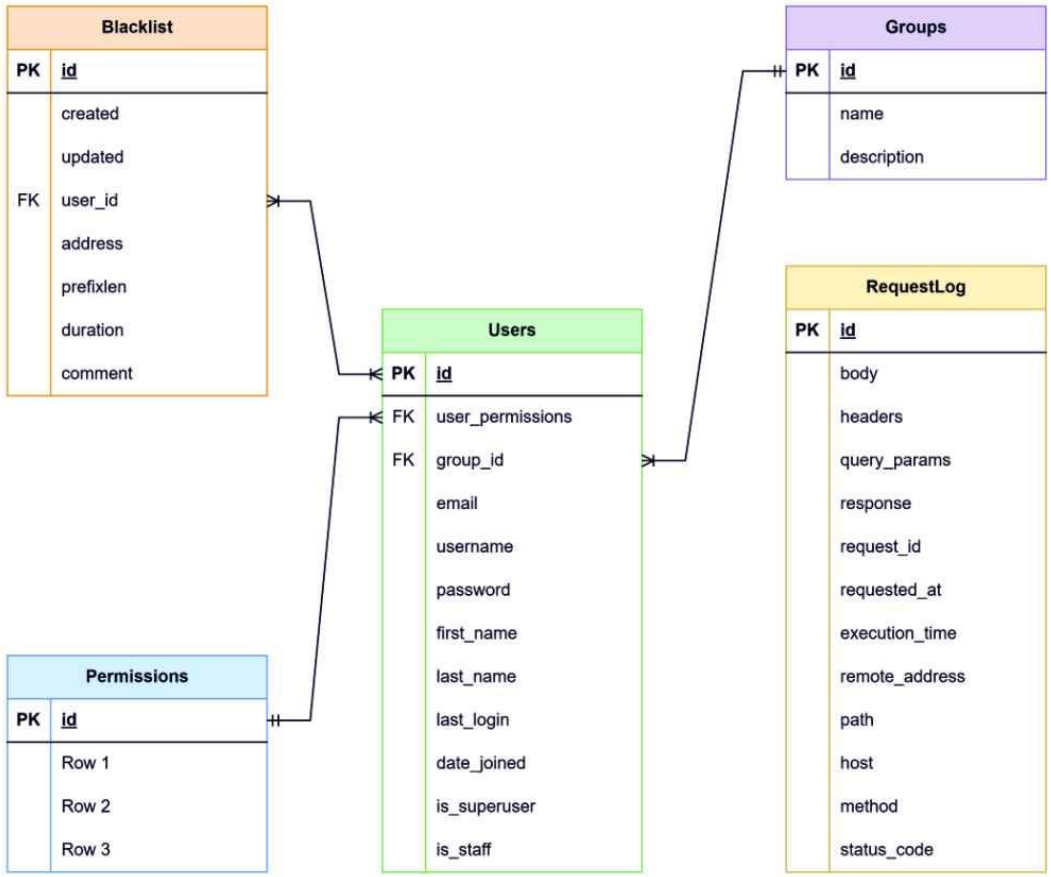


КРБКБ.200101.20.01.01 Е8



						КРБКБ.200101.20.01.01 Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення атак за допомогою автоматичного моніторингу вебсервера	Літ	Маса / Масштаб
Розроб.		Бабаєвський В.М.			Алгоритм захисту від атак перебору паролів	У	
Перевір.		Муляр І.В.				Аркуш 2	Аркушів 3
Н.контр.		Мостовий С.В.				ХНУ, Кб-20-1	
Т.контр.							
Затверд.		Кльоц Ю.П.					

КРБКБ.200101.20.01.01 Е8



					КРБКБ.200101.20.01.01 Е8		
Зм.	Арк.	№ докум.	Підпис	Дата	Система виявлення атак за допомогою автоматичного моніторингу вебсервера **Діаграма бази даних модуля виявлення атак**	Літ	Маса / Масштаб
Розроб.		Бабевський В.М.				У	
Перевір.		Мулер І.В.				Аркуш 3	Аркушів 3
Н.контр.		Мостовий С.В.					
Т.контр.						ХНУ, Кб-20-1	
Затверд.		Кльоц Ю.П.					

Завідувачу кафедри кібербезпеки
к.т.н., доц. Кльоцу Ю.П.
Бабаєвського Віталія Михайловича
ПІБ здобувача вищої освіти

Студента ФІТ, 4 курсу, групи КБ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у хмельницькому національному університеті» від 31.08.2023, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

17.06.2024
дата

підпис

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 9%**

ID: 131420 Назва: Система виявлення атак за допомогою автоматичного моніторингу вебсервера Додано в БД: 2024-06-18 Автора: Бабаєвський В.М. Керівники: Муляр І.В. Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	74011	642	891 (1%)	13 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра кібербезпеки

ID перевірки:
1016373398

Дата перевірки:
18.06.2024 22:24:54 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
18.06.2024 22:34:14 EEST

ID користувача:
100008300

Назва документа: **Бабаєвський_На плагіат**

Кількість сторінок: **56** Кількість слів: **10476** Кількість символів: **81795** Розмір файлу: **2.00 MB** ID файлу: **1016180881**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

10%
Схожість

Найбільша схожість: **4.88%** з Інтернет-джерелом (https://ela.kpi.ua/handle/123456789/44087)

9.56% Джерела з Інтернету	236	Сторінка 58
1.14% Джерела з Бібліотеки	101	Сторінка 60

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4
Підозріле форматування	11 сторінок

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

КАФЕДРИ КІБЕРБЕЗПЕКИ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Система виявлення атак за допомогою автоматичного моніторингу вебсервера

Автор: Бабаєвський Віталій Михайлович

Спеціальність: 125 – Кібербезпека

Освітня програма: освітньо-професійна

Науковий керівник: Муляр Ігор Володимирович, канд. техн. наук, ст. викладач

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Оригінальність тексту роботи за результатами перевірки системою Unichek складає 90%, оригінальність тексту роботи за результатами перевірки системою Anti-Plagiarism v-15.257 складає 99%.

Згідно з Положенням про систему забезпечення академічної доброчесності у ХНУ (https://khmnu.edu.ua/wp-content/uploads/normatyvni-dokumenty/polozhennya/pro-systemu-zabezpechennya-akademichnoyi-dobrochesnosti.pdf, Додаток В) кваліфікаційна робота, виконана за , освітньо-професійною програмою, кількісні показники рівня унікальності тексту у відсотках до загального обсягу матеріалу в якій складає 75-100 %, визнається роботою з високою унікальністю тексту: «Текст вважається унікальним і не потребує додаткових дій щодо запобігання неправомірним запозиченням».

Керівник роботи _____ Ігор МУЛЯР

Завідувач кафедри кібербезпеки _____ Юрій КЛЬОЦ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Бабаєвський Віталій Михайлович

Тема Система виявлення атак за допомогою автоматичного моніторингу вебсервера

Спеціальність 125 – Кібербезпека

Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:

кількість листів креслень 3; кількість сторінок записки 82.

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі була розроблена система виявлення атак за допомогою автоматичного моніторингу вебсервера. Дана система здатна виявляти популярні типи атак та сканування сканерами вразливостями. У процесі проєктування були розроблені такі компоненти: модуль виявлення атак, модуль управління системою виявлення атак, модуль реєстрації подій для моніторингу активності вебдодатку. Крім того, проведено тестування розробленої системи, що підтвердило її ефективність.

2. Висновок про відповідність кваліфікаційної роботи завданню. У кваліфікаційній роботі було виконано поставлене завдання як у теоретичній, так і в практичній частині. Було дотримано усіх вимог щодо виконання

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У кваліфікаційній роботі виконано низку завдань, таких як наведення загальної характеристики задачі, визначення проблематики, розгляд предмета та методів дослідження. Виконані задачі сприяли створенню рішення для системи виявлення атак на вебдодатки з метою підвищення рівня інформаційної безпеки. У першому розділі проведено аналіз предметної області, зокрема загроз інформаційної безпеки, типів атак на вебдодатки та існуючих сканерів вразливостей. На основі проведеного аналізу була поставлена задача створення ефективного програмного рішення для виявлення атак. У другому розділі, на основі результатів аналізу та визначеного завдання, спроектовано структуру вебдодатку, розроблено методи виявлення атак на вебресурси та розглянуто можливі варіанти обходу систем виявлення вторгнень. У третьому розділі описано практичні аспекти розробки та реалізації компонентів системи, а також проведено тестування розробленої системи, що підтвердило її ефективність у виявленні атак.

4. Позитивні сторони роботи Розроблена система виявлення атак за допомогою автоматичного моніторингу вебсервера є актуальною та цікавою темою, що відповідає потребам сучасних вебресурсів у забезпеченні інформаційної безпеки. Ця система спрямована на підвищення рівня захисту від потенційних атак, що може значно покращити безпеку користувачів та надійність їхніх даних.

5. Негативні сторони роботи В системі відсутня можливість ефективного захисту від DDOS-атак, що може причинити відмови в обслуговуванні користувачів, що є критичним аспектом для стабільності та надійності інформаційних систем. У системі відсутній графічний інтерфейс та документації, тому користувачам з недостатнім рівнем знань потрібен час на вивчення системи.

6. Оцінка графічного оформлення та пояснювальної записки роботи. Графічне оформлення кваліфікаційної роботи відповідає темі роботи та виконане з дотриманням стандартів. У цілому графічне оформлення є чітким та якісним, а пояснювальна записка відповідає нормам оформлення.

7. Відгук про роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки, оскільки весь матеріал роботи є структурованим, чітким та послідовним. Усі розділи роботи мають логічну послідовність, що сприяє зрозумінню викладеного матеріалу в рамках теми роботи. Графічний матеріал допомагає наочно продемонструвати доцільність та ефективність прийнятих рішень для досягнення мети.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Ураховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінки «відмінно»

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи)
Підченко Сергій Костянтинович, завідувач кафедри ТМІТ, доктор технічних наук, професор

« 17 » 06 2024. (підпис)