МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра кібербезпеки

“До захисту допущено”
Завідувач кафедри
_____ Любчак В.О.
«___» _____ 20__р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 125 «Кібербезпека»
освітньо-професійної програми «Кібербезпека»
на тему: ”Захист web-ресурсів: виявлення вразливостей за допомогою методологій web-фаззінгу”

Здобувачки групи КБ-01 Пархоменко Олени Сергіївни.

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Пархоменко О.С.

Керівник старший викладач, кандидат фізико-математичних наук,
Коваль В.В. _____

Суми 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра кібербезпеки

ЗАТВЕРДЖУЮ
_____ Любчак В.О.
«___» _____ 20__р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр

зі спеціальності 125 «Кібербезпека»,
освітньо-професійної програми «Кібербезпека»
здобувачки групи КБ-01 Пархоменко Олени Сергіївни

1. Тема роботи: ”Захист web-ресурсів: виявлення вразливостей за допомогою методологій web-фаззінгу”
   затверджена наказом по СумДУ №_____ від «___» _____ 20___ р.
2. Термін подання студентом роботи «___» _____ 20___ р.
3. Вхідні дані до роботи: _____
   _____
   _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити):
   1. Огляд проблематики методик захисту інформації.
   2. Основи методології фаззінгу.
   3. Практичне дослідження методології фаззінгу.

5. Перелік графічного матеріалу (із зазначенням плакатів, презентацій тощо):
   _____
   _____

6. Консультанти до проекту (роботи), із зазначенням розділів, що їх стосуються

Розділ	**Прізвище, ініціали та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**

7. Дата видачі завдання «____» _____ 20 ___ р.

Здобувачка вищої освіти _____ Керівник _____

КАЛЕНДАРНИЙ ПЛАН

№	**Назва етапів кваліфікаційної роботи**	**Термін виконання**	**Примітка**
1	Огляд проблематики методик захисту інформації		
2	Основи методології фаззінгу		
3	Практичне дослідження застосування фаззінгу		

Здобувачка вищої освіти _____ Керівник _____

АНОТАЦІЯ

Кваліфікаційна робота виконана на 49 аркушах та містить 11 рисунків, 10 таблиць, 1 додаток та 23 джерела.

Об'єкт дослідження: застосуання різних методів виявлення вразливостей в інформаційних системах.

Мета роботи: дослідити методологію фаззінгу та оцінити її ефективність для процесів виявлення вразливостей та захисту інформації в умовах сучасного світу.

Метод дослідження: комбінація теоретичних, емпіричних і практичних методів.

Результати роботи:

Розглянуто основні концепції фаззінгу, інструменти, що застосовуються для його реалізації, та застосування цієї методики в сучасних умовах кібербезпеки.

Проведено огляд питання захисту інформації на основі методологію фаззінгу: методу виявлення вразливостей у web-додатках.

Проведено дослідження практичного застосування методології фаззінгу та виявлено вразливість ІС, яка полягає у наявності відкритого доступу до конфігураційного файлу FTP-серверу.

Розроблені рекомендації щодо усунення практично отриманих вразливостей з урахуванням методології.

Основні результати кваліфікаційної роботи обговорювалися на мыжнарожній науковій конференції ІМА-2024.

Ключові слова: фаззінг, тестування, вразливість, виявлення вразливостей, захист інформації.

ЗМІСТ

Зміст 5
Вступ 6
1 Огляд проблематики методик захисту інформації 7
2 Основи методології фаззінгу 11
2.1 Концепція фаззінгу 11
2.2 Види інструментів 14
2.3 Актуальність використання фаззінгу 19
3 Практичне дослідження застосування фаззінгу 20
3.1 Технічне завдання 20
3.2 Обґрунтування вибору інструменту 21
3.3 Опис обраного інструменту 22
3.3.1 Ключові поняття 25
3.3.2 Механізм роботи 26
3.4 Моделювання середовища тестування 30
3.4.1 Створення та підготовка ВМ 30
3.4.2 Встановлення утиліти ffuf 32
3.5 Тестування корпоративних ресурсів 33
Висновок 41
Список використаних джерел 43
Додаток А – Ключі керування утилітою 46

ВСТУП

Збільшення обсягу інформації, яка обробляється в онлайн середовищі, призводить до посилення уваги до питань кібербезпеки. Це відбувається на тлі постійного зростання кількості загроз з боку кіберзлочинців, які намагаються використовувати різноманітні техніки для отримання доступу до конфіденційної інформації або нанесення шкоди інформаційним системам. Саме тому забезпечення безпеки додатків стає критично важливим завданням для організацій та користувачів.

Однією з ключових складових забезпечення кібербезпеки є виявлення та усунення вразливостей у додатках. Через ці вразливості часто здійснюються атаки на інформаційні системи, що призводить до серйозних наслідків для роботи організацій та приватної безпеки користувачів. У цьому контексті методологія фаззінгу набуває все більшого значення, оскільки вона дозволяє систематично виявляти й усувати вразливості у додатках.

Мета кваліфікаційної роботи полягає у вивченні методології фаззінгу як методу виявлення вразливостей в інформаційних системах. Дослідження включає аналіз методів захисту інформації, теоретичні та практичні аспекти застосування фаззінгу.

Актуальність проведення даного дослідження підтверджується постійним зростанням кіберзагроз та необхідністю постійного вдосконалення заходів кібербезпеки відповідно до сучасних вимог. Розуміння та впровадження методології фаззінгу може стати важливим кроком у забезпеченні безпеки додатків та захисту конфіденційної інформації в онлайн середовищі.

1 ОГЛЯД ПРОБЛЕМАТИКИ МЕТОДИК ЗАХИСТУ ІНФОРМАЦІЇ

Сучасні методики захисту інформації є невід'ємною частиною сучасного цифрового середовища, де кіберзлочинці намагаються використовувати різноманітні технології для несанкціонованого доступу до конфіденційних даних.

Методики захисту інформації можна класифікувати за кількома категоріями, які охоплюють широкий спектр технічних та організаційних заходів.

– **шифрування даних** використовується для захисту інформації від несанкціонованого доступу;

– **керування правами доступу** ставить перед собою завдання перевірки ідентифікації користувачів та надання їм відповідних прав доступу;

– **керування мережевою безпекою** орієнтоване на захист мережевої інфраструктури від зовнішніх загроз;

– **керування фізичною безпекою** спрямоване на захист фізичних об'єктів, таких як приміщення та обладнання, від несанкціонованого доступу;

– **управління ризиками** включає розробку та реалізацію стратегій безпеки, а також проведення аудитів для перевірки відповідності встановленим стандартам безпеки.

Комбінація цих методик створює надійні системи захисту інформації, які забезпечують конфіденційність, цілісність та доступність даних для організацій та користувачів. [1]

Також широко використовується тестування додатків, оскільки воно дозволяє виявити та виправити потенційні вразливості на різних етапах життєвого циклу додатку.

У зв'язку з постійно зростаючими загрозами кібербезпеки, компанії та організації активно вдосконалюють підходи до тестування додатків для забезпечення їх надійності та безпеки. [1]

Static Application Security Testing (далі SAST) та Dynamic Application Security Testing (далі DAST) є двома основними методами тестування безпеки програмного забезпечення. [2]

SAST – це методика тестування безпеки, яка аналізує вихідний код, байт-код або двійковий код програми без її виконання.

Інструменти SAST сканують код програми, шукаючи уразливості та недоліки безпеки, такі як SQL-ін'єкції, XSS, неправильне керування пам'яттю тощо. Аналіз проводиться на ранніх етапах розробки, що дозволяє виявити уразливості ще до компіляції та виконання програми.

DAST – це методика тестування безпеки, яка аналізує роботу програми в реальному часі, під час її виконання.

Інструменти DAST імітують атаки на працюючу програму, щоб виявити уразливості, які можуть бути експлуатовані під час роботи системи. Вони аналізують поведінку програми, взаємодію з зовнішнім середовищем і користувачами, мережеві запити тощо. [2]

Порівняльна характеристика цих методик наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика методик тестування

Характеристика	**SAST**	**DAST**
Етапи тестування	використовується на етапі розробки та тестування коду	застосовується на етапі тестування або експлуатації готової програми
Знання про код	методика White Box testing	методика Black Box testing
Тип уразливостей	виявляє уразливості в статичному коді	фокусується на уразливостях, які можуть бути експлуатовані під час виконання програми

Продовження таблиці 1.1

Характеристика	**SAST**	**DAST**
Витрати на усунення вразливостей	більшість проблем можна виявити та усунути до того, як код потрапить на етап контролю якості (QA)	критичні вразливості можуть бути виправлені в рамках аварійного випуску
Вхідні дані	не вимагає виконання додатка	не вимагає вихідного коду чи бінарних файлів
Результати	дає розробникам змогу виправити код до його компіляції	допомагає знайти та усунути проблеми в вже працюючій системі

Використання обох методів у комплексі може значно підвищити рівень безпеки програмного забезпечення, забезпечуючи як глибокий аналіз коду, так і перевірку його безпеки під час виконання.

Стрімке збільшення обсягів даних створює значні виклики для використання цих методик, які вимагають від організацій вдосконалення методів аналізу безпеки, впровадження автоматизації та забезпечення високого рівня захисту конфіденційної інформації. [3]

Використання методів машинного навчання та штучного інтелекту може допомогти зменшити кількість помилкових спрацьовувань та підвищити точність виявлення вразливостей.

Автоматизація процесів тестування надає численні переваги, такі як:

- збільшення продуктивності;
- зниження витрат;
- підвищення якості та точності;
- оптимізація процесів.

Завдяки цим перевагам організації можуть ефективніше досягати своїх цілей та бути більш конкурентоспроможними у сучасному динамічному середовищі. [4]

Фаззінг займає важливе місце в автоматизації тестування методиками SAST та DAST, значно посилюючи ці підходи. Інтеграція фаззінгу в процеси тестування дозволяє автоматизувати виявлення вразливостей, які можуть залишитися непоміченими при використанні лише цих методик.

Основна ідея фаззінгу полягає у намаганні випробувати програму на можливість неправильної обробки даних, які можуть викликати аварійну зупинку програми або витік інформації. [5]

Фаззінг може ефективно використовуватись як на етапах тестування методикою SAST, так і на етапах тестування методикою DAST. Він допомогає виявити потенційні вразливості на ранніх етапах розробки шляхом генерації великого обсягу різних даних, які можуть викликати небажану поведінку програми.

Для обох методик використання фаззінгу може значно підвищити рівень безпеки додатка та знизити ймовірність виникнення вразливостей. [6]

2 ОСНОВИ МЕТОДОЛОГІЇ ФАЗЗІНГУ

Методологія допомагає виявити вразливості, які можуть бути пропущені під час інших методів тестування безпеки, а також забезпечує більш повну охопленість заходів безпеки. Особливо в умовах швидкої розробки і впровадження програмного забезпечення, фаззінг є важливим інструментом для забезпечення безпеки додатків. [6]

Фаззінг став однією з найефективніших та найбільш використовуваних технік. Він відкриває двері до виявлення помилок та вразливостей, які можуть залишитися непоміченими під час звичайних тестів.

Це стає можливим завдяки автоматизованому введенню великої кількості випадкових або штучно змінених даних в програму або функцію з метою викликати непередбачувані або відхилення від очікуваного результати.

2.1 Концепція фаззінгу

Залежно від конкретного контексту та потреб тестування, може бути вибраний підхід до фаззінгу, який найбільш відповідає вимогам проекту. Загалом можна виділити наступні фактори, що впливатимуть на методику проведення фаззінгу:

Вид програмного забезпечення

Фаззінг може бути використаний для тестування різних типів програм, включаючи додатки, операційні системи, протоколи мережі, бібліотеки та інше.

Цілі тестування

Цілі фаззінгу можуть варіюватися від пошуку вразливостей безпеки до тестування на відповідність стандартам, відтворення певних сценаріїв використання або покриття коду.

Підхід до аналізу результатів

Якщо фаззер виявляє вразливості, важливо мати ефективний процес аналізу результатів для оцінки серйозності виявлених проблем та розробки заходів захисту.

Стратегія вибору вхідних даних

Підходи до вибору вхідних даних можуть варіюватися від випадкового вибору до врахування аналітичних або статистичних властивостей програми для покращення шансів на виявлення вразливостей.

Наявні ресурси

Ефективність фаззінгу може залежати від доступних обчислювальних та пам'яткових ресурсів для генерації, виконання та аналізу вхідних даних.

Методи генерації вхідних даних

Вибір методу генерації вхідних даних (наприклад, випадкове генерування, контрольована генерація або мутаційний підхід) може впливати на результати фаззінгу.

Способи взаємодії з програмою

Фаззінг може включати взаємодію з програмою через API, командний рядок, графічний інтерфейс або інші інтерфейси, що можуть впливати на ефективність та швидкість тестування.

Врахування цих факторів допомагає забезпечити вибір інструменту, що найкращим чином відповідає потребам проекту і забезпечує максимальну ефективність тестування програмного забезпечення. [7]

Хоча фаззінг є потужним інструментом для виявлення вразливостей, важливо розуміти його обмеження та враховувати їх при використанні в процесі тестування безпеки програмного забезпечення.

Фаззінг особливо корисний завдяки своїй здатності до автоматизації, що дозволяє одночасно запускати тестування і займатися іншими завданнями. Він

охоплює широкий спектр коду, виявляючи помилки, які можуть бути пропущені під час ручного тестування.

Таким чином, можна виділити наступні переваги фаззінгу:

- автоматизація процесу тестування;
- велика площа покриття;
- виявляє вразливості додатка.

Серед основних недоліків фаззінгу слід виділити складність генерації вхідних даних, особливо для складних програм. Цей метод може пропустити певні вразливості через відсутність повного контексту використання програми. Крім того, фаззінг потребує значних часових та ресурсних витрат для проведення ефективного тестування.

Основні недоліки методології фаззінгу:

- складність генерації вхідних даних;
- має обмеження умовами тестування;
- залежить від часу і наявних ресурсів.

Узагальнена інформація наведена у таблиці 2.1.

Таблиця 2.1 – Переваги та недоліки методології

Переваги	Недоліки
Автоматизація процесу тестування	Складність генерації запитів
Велика площа покриття	Залежність від часу та наявних ресурсів
Виявляє вразливості додатка	Обмежена умовами тестування

Ця технологія допомагає знайти проблеми, які можуть бути недооцінені у традиційних методах тестування. В результаті, вона підвищує рівень якості та безпеки розроблюваного продукту. [8]

Успішним прикладом використання фаззінгу для виявлення вразливостей у програмному забезпеченні є інцидент, пов'язаний з

Adobe Reader. [9] Дослідники за 50 днів виявили п'ятдесят вразливостей у цій програмі, навіть не маючи доступу до вихідного коду.

Даний приклад демонструє високу ефективність використання методології фаззінгу для виявлення вразливостей. Результати цього дослідження викликали значний інтерес у громадськості, зокрема серед фахівців у галузі інформаційної безпеки.

2.2 Види інструментів

Фаззери – це програмні засобами, які використовуються для фаззінгу. Основна мета фаззерів полягає у створенні тестових випадків, вводу випадкових або систематичних даних в програму, а також у виявленні помилок. [6]

Існує ряд класифікацій фаззерів, що базуються на різноманітних критеріях, які наведено нижче:

- За тестовим призначенням.
- За ціллю тестування.
- За алгоритмом генерації вхідних даних.
- За рівнем доступу до інформації.

Класифікація фаззерів допомагає зрозуміти їхню ефективність та обмеження кожного виду інструменту та забезпечити простоту вибору в залежності від поставлених вимог. [10]

В залежності від методу генерації вхідних даних, фаззери можуть використовувати різні підходи, такі як модель програми, мутаційний чи протокольний алгоритми, або їх поєднання в гібридному методі. Це дозволяє визначити найбільш ефективні методи генерації тестових даних для виявлення помилок.

У таблиці 2.2 наведено деталізацію видів фаззерів.

Таблиця 2.2 – Види фаззерів за алгоритмом генерації даних

Вид	**Опис**	**Перевага**	**Недолік**
Генерація на основі моделі	Генеруються вхідні дані відповідно до моделі програми.	Забезпечують глибоке тестування відповідно до специфікацій	Обмежені реалістичністю, можуть пропустити уразливості, які виникають у реальних сценаріях
Мутаційний підхід	Змінюються певні зразки вхідних даних, створюючи нові варіанти.	Простота та швидкість у створенні варіативних тестових наборів.	Можливе недостатнє покриття різних шляхів в програмі з мутаційними даними.
Протокольний підхід	Генеруються дані на основі протоколу, використовуючи зловмисно сформовані послідовності.	Дозволяють виявляти уразливості, що виникають внаслідок неправильної обробки протокольних даних	Можуть бути обмежені відображенням реальних сценаріїв використання програми
Гібридний підхід	Використовують або комбінують принципи попередніх підходів	Висока гнучкість та адаптивність до різних типів програм та вразливостей.	Складність налаштування та підтримки через різнорідність підходів.

За тестовим призначенням фаззери можуть фокусуватися на конкретних аспектах програми чи системи або ставити за мету охоплення якомога більшої частини функціоналу.

У таблиці 2.3 наведено деталізацію видів фаззерів.

Таблиця 2.3 – Види фаззерів за тестовим призначенням

Вид	**Опис**	**Перевага**	**Недолік**
Направлене тестування	Фокусуються на певному функціоналі програми, ігноруючи інші розгалуження.	Ефективне виявлення загальних вразливостей.	Може бути менш ефективним для специфічних вразливостей.
Тестування покриття	Спрямовані на охоплення якомога більшої частини функціоналу програми.	Гарантує високий рівень покриття коду програми.	Може пропустити певні типи вразливостей, які не пов'язані з логікою виконання програми.

Залежно від рівня доступу до інформації, фаззери можуть відноситися до категорій з різним ступенем знання внутрішньої структури програми. Це визначає рівень деталізації та точність аналізу, доступний під час процесу тестування.

У таблиці 2.4 наведено деталізацію видів фаззерів.

Таблиця 2.4 – Види фаззерів за рівнем доступу

Вид	**Опис**	**Перевага**	**Недолік**
White box	Використовуються дані про вихідний код програми.	Дозволяють точно виявити місце та причину помилок.	Обмежені доступом до вихідного коду, що може бути недоступним або непрактичним.
Gray box	Аналізують ПЗ для збору релевантної інформації, не використовуючи код.	Забезпечують частковий доступ до внутрішніх механізмів програми.	Можуть бути обмежені у доступі до деяких чутливих даних чи функціоналу.
Black box	Не залежать від інформації про ціль тестування.	Забезпечують об'єктивну оцінку зовнішнього поведінки програми.	Може пропустити внутрішні вразливості, які не виявляються через зовнішній інтерфейс.

Залежно від цілей тестування, фаззери можуть бути налаштовані на проведення тестування файлових форматів, web-застосунків, web-браузерів або мережевих протоколів відповідно до потреб програми або системи.

У таблиці 2.5 наведено деталізацію видів фаззерів.

Таблиця 2.5 – Види фаззерів за ціллю тестування

Вид	**Опис**	**Перевага**	**Недолік**
Фазери файлових форматів	Тестують обробку файлів, намагаючись виявити помилки шляхом подачі невірно сформованих даних.	Виявлення вразливостей у форматах файлів.	Можуть не виявити складних уразливостей, які не відображаються у форматі даних
Фазери web-додатків	Спеціалізуються на виявленні вразливостей web-застосунків, охоплюючи і їхні інтерфейси та функціонал.	Здатність виявляти вразливості у web-додатках.	Обмежені у виявленні низькорівневих проблем, таких як недоліки в системних компонентах

Продовження таблиці 2.5

Вид	**Опис**	**Перевага**	**Недолік**
Фазери web-браузерів	Спрямовані на тестування вразливостей обробки HTML, виконання скриптів і форматів медіа-контенту.	Ефективні для виявлення вразливостей web-браузерів	Обмеженість у тестуванні функцій, що не взаємодіють з web-інтерфейсом.
Фазери мережевих протоколів	Тестують протоколи різних рівнів OSI.	Дозволяють виявляти уразливості в мережевому протоколі	Потребують значного обсягу ресурсів для виконання тестів на широкому спектрі протоколів

Використання відповідних видів фаззерів залежить від конкретних потреб та характеристик тестування програмного забезпечення. [10]

Візуальне представлення класифікації наведено на рис. 2.1.

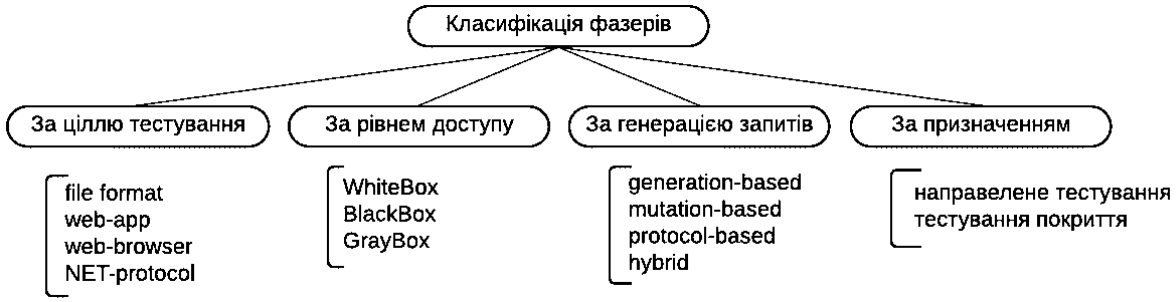

Рисунок 2.1 – Класифікація фаззерів

Така класифікація допомагає вибрати найкращий інструмент відповідно до конкретних потреб тестування, забезпечуючи ефективне виявлення вразливостей та оптимальне використання ресурсів.

Однак, як вибір інструменту, що відповідає поставленим вимогам, так і сам процес тестування – це лише мала частина всього процесу ідентифікації вразливостей.

Потрібно не тільки ідентифікувати потенційні вразливості, але й оцінити їх критичність, аби визначити, які з них потребують негайного виправлення. Крім того, важливо забезпечити точність аналізу, щоб мінімізувати ризик хибних результатів, які можуть вплинути на загальну безпеку web-додатка.

До основних проблем аналізу результатів можна віднести наступне:

Великий обсяг даних

Інструменти для фаззінгу можуть згенерувати величезну кількість даних. Аналіз кожного результату вручну може бути непосильним завданням, особливо якщо йдеться про великі проекти.

Хибні результати

Часто інструменти можуть ідентифікувати як вразливості ті місця, що насправді такими не є. Або, навпаки, деякі вразливості можуть залишитися непоміченими через обмеження інструментів або їх неправильну конфігурацію. Це може призвести до витрати часу на перевірку таких результатів.

Інтерпретація даних

Розуміння того, як саме web-додаток реагує на різні вхідні дані, може бути складним завданням. Для цього потрібен глибокий досвід та знання специфіки web-технологій.

Пріоритезація вразливостей

Виявлення багатьох вразливостей під час фаззінгу може створити проблему пріоритезації. Важливо вирішити, які з виявлених вразливостей потребують негайного виправлення, а які можна відкласти.

Для ефективного аналізу результатів фаззінгу необхідно використовувати не тільки автоматизовані інструменти, а й експертні знання. Це поєднання дозволяє правильно інтерпретувати результати, мінімізувати хибні спрацювання, а також ефективно пріоритезувати вразливості.

Інструменти можуть автоматизувати велику частину процесу, але людина залишається ключовим елементом для прийняття остаточних рішень і забезпечення надійного захисту web-додатків. [11]

2.3 Актуальність використання фаззінгу

У сучасному світі фаззінг виявляється дуже важливим інструментом для тестування програмного забезпечення. Фаззінг став необхідним інструментом для забезпечення надійної якості та безпеки додатків. [11]

Ключові аспекти ролі фаззінгу в сучасному світі:

Виявлення вразливостей безпеки

Фаззінг може виявити вразливості безпеки в програмах, такі як переповнення буфера, вразливості форматування рядка, недоліки у перевірці вводу даних та інші. Це допомагає розробникам виправити ці проблеми до того, як вони можуть бути використані зловмисниками.

Зменшення витрат на тестування

Автоматизований фаззінг може допомогти зменшити витрати на тестування, оскільки воно не потребує значних людських ресурсів для виконання. Це особливо важливо в сучасному світі, де швидкість розробки та випуску програмного забезпечення є надзвичайно важливою.

Підвищення якості програмного забезпечення

Фаззінг допомагає виявити помилки та непередбачені стани, які можуть впливати на якість та надійність програми. Виявлення цих проблем у ранніх стадіях розробки дозволяє їх виправити швидше та ефективніше.

Автоматизація тестування

Фаззінг дозволяє автоматизувати процес тестування, що дозволяє збільшити його ефективність та швидкість. Велика кількість тестів може бути виконана автоматично, що дозволяє розробникам зосередитися на інших аспектах розробки.

Отже, фаззінг відіграє важливу роль у сучасному світі, допомагаючи покращити якість та безпеку програмного забезпечення. [11]

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ ФАЗЗІНГУ

Перевірка ефективності цього підходу є важливим кроком у забезпеченні безпеки web-додатків. Успіх дослідників, що тестували Adobe Reader, може бути важко повторити. Здебышшого цьому заважає значне покращення як вимог до захисту додатків, так і розвиток методів захисту інформації.

Проте важливо розглянути алгоритм, за яким це дослідження було проведене.

Цей процес включав в себе наступні кроки:

1. Вибір цільового додатка
2. Аналіз протоколів і вхідних даних
3. Розробка фаззера
4. Запуск фаззера та аналіз результатів
5. Виявлення та документування вразливостей
6. Повторення процесу та перевірка результатів

Цей алгоритм дозволив дослідникам швидко і ефективно виявити велику кількість вразливостей у програмному забезпеченні Adobe Reader за короткий час. [9]

Таким чином, базуючись на даному алгоритмі було сформовано завдання для практичної перевірки ефективності методології фаззінгу в умовах сучасного світу.

3.1 Технічне завдання

Ціллю даного завдання є проведення практичної перевірки ефективності методології фаззінгу на прикладі інтернет-магазину з метою виявлення потенційних вразливостей у додатку та надання рекомендацій, щодо його захисту від атак.

Тестування було проведено у форматі black box тестування з використанням mutation based фазеру. Об'єктом тестування є web-ресурс інтернет-магазину автозапчастин.

Замість розробки власного фаззера, було вирішено використати вже готовий інструмент для фаззінгу – Fuzz Faster U Fool (далі FFUF). Інструмент дав змогу зосередити увагу на тестуванні файлвої структури ресурсу.

Кроки виконання завдання:

1. Конфігурація та налаштування інструменту фаззінгу для відповідності потребам тестування web-додатка.
2. Запуск фаззера та автоматичне генерування тестових векторів для виявлення потенційних вразливостей у web-додатку.
3. Аналіз результатів тестування, ідентифікація вразливостей та їх документування.
4. Надання рекомендацій, щодо виправлення вразливостей та впровадження заходів безпеки.

Таким чином вдалося перевірити ефективність методології фаззінгу на практиці та виявити можливі слабкі місця додатку інтернет-магазину з точки зору безпеки.

3.2 Обґрунтування вибору інструменту

Утиліта FFUF забезпечує можливість автоматизованого сканування web-ресурсів та виконання фаззінгу, або ж перебір параметрів, для виявлення потенційних вразливостей. [12]

Використання FFUF для тестування web-додатків має декілька переваг:

- швидкість та ефективність роботи;
- гнучкість налаштування;
- підтримка HTTP.

Інструмент має велику спільноту користувачів та активну підтримку, що забезпечує надійність використання.

3.3 Опис обраного інструменту

FFUF - це потужний інструмент для здійснення атак перебору файлів та каталогів на web-сайтах. Він широко використовується в тестуванні на проникнення та на етапах розробки додатків для автоматизації процесу виявлення прихованих ресурсів на web-серверах.

FFUF розроблений для сканування web-додатків і може бути використаний для знаходження різних видів вразливостей, таких як директорії з відкритим доступом, файлових ресурсів і багато інших. [12]

За класифікацією, наведеною у розділі 2.2, FFUF належить наступним категоріям:

- **За ціллю тестування:** "Web-App fuzzer", оскільки його основна функція полягає у фаззінгу додатку.
- **За наявністю апріорних знань:** "Black Box", оскільки він не вимагає знання або доступу до вихідного коду цільового додатку.
- **За алгоритмом генерації вхідних даних:** "Mutation-Based fuzzer", оскільки він використовує зразки HTTP запитів і змінює їх за певним алгоритмом для генерації нових варіантів.

FFUF може бути використаний як для "Направленого тестування", коли фокусується на певному функціоналі або критичних точках додатку, так і для "Тестування покриття".

Взаємодія з інтерфейсом утиліти FFUF в основному відбувається через командний рядок, що робить її досить гнучкою і зручною для автоматизації та інтеграції у скрипти.

Основні параметри та опції можна передати як аргументи командного рядка, дозволяючи вказувати URL-адреси, словники, параметри запитів та інші налаштування. [13]

Завдяки цьому, можна виділити наступні основні особливості:

- **Швидкість:** FFUF розроблений для швидкого та ефективного сканування. Він може відправляти тисячі запитів за секунду, що робить його ідеальним для великих або складних web-застосунків.
- **Гнучкість:** FFUF має багато параметрів, які можна налаштувати відповідно до потреб користувача. Це включає установку заголовків, методів запитів, розподіл словників та багато іншого.
- **Підтримка мультиплексування:** FFUF може працювати зі списками URL-адрес, що дозволяє йому сканувати кілька web-сайтів або сторінок одночасно.
- **Підтримка HTTPS:** Він може відправляти запити через захищений протокол HTTPS, що робить його придатним для тестування захищених web-застосунків.
- **Результати в реальному часі:** FFUF може надавати результати сканування в реальному часі, що дозволяє оперативно відслідковувати прогрес тестування.
- **Підтримка власних словників:** Користувачі можуть використовувати власні словники або списки для генерації варіантів запитів.
- **Підтримка проксі:** FFUF може взаємодіяти з проксі-серверами, що дозволяє аналізувати та перехоплювати запити та відповіді.

FFUF має численні ключі та параметри для налаштування та керування фаззінг-процесом.

Перелік всіх ключів та їх детальний опис наведено у додатку А.

Використання утиліти FFUF має свої переваги та недоліки, які варто враховувати перед її використанням. Переваги та недоліки утиліти наведено у таблиці 3.1.

Таблиця 3.1 – Переваги та недоліки FFUF

Переваги	Недоліки
Швидкість FFUF - дуже швидкий інструмент для фаззінгу, який може генерувати велику кількість запитів в секунду. Це робить його ідеальним для тестування на вразливості та сканування великих ресурсів.	**Вимоги до знань** Для ефективного використання FFUF необхідно мати базові знання HTTP, методів фаззінгу та безпеки, оскільки неправильне налаштування може призвести до негативних наслідків.
Гнучкість FFUF має багато параметрів та налаштувань, що дозволяє йому адаптуватися до різних сценаріїв тестування. Ви можете налаштувати його для роботи з різними HTTP-методами, заголовками, тілами запитів та багато іншого.	**Потенційний зловживання** Помилкове використання FFUF може призвести до зловживання та негативно вплинути на web-сервери та ресурси. Важливо використовувати його відповідально та в межах законних та етичних норм.
Підтримка рекурсивного фаззінгу FFUF може бути використаний для виявлення файлів та каталогів на web-сервері шляхом рекурсивного фаззінгу. Це корисно для сканування web-додатків та виявлення ресурсів.	**Незабезпечені сервери** Фаззінг може виявити вразливості на серверах, які не відповідають вимогам безпеки, і викликати атаки на ці сервери. Тому важливо використовувати FFUF лише на серверах і ресурсах, до яких ви маєте право доступу.
Фільтрація результатів FFUF дозволяє фільтрувати результати за допомогою статус-кодів, регулярних виразів, довжини відповідей та інших параметрів. Це допомагає зосередитися на важливих запитах та відповідях.	**Вимоги до ресурсів** Велика швидкість та кількість запитів, які може генерувати FFUF, можуть призвести до надмірного споживання ресурсів, таких як мережевий трафік та процесор. Це може вплинути на продуктивність системи або призвести до блокування IP-адреси сервера.
Підтримка HTTP-заголовків FFUF дозволяє вам додавати та керувати HTTP-заголовками для кожного запиту. Це корисно для імітації конкретного користувача чи змінних в ході фаззінгу.	

FFUF є потужним інструментом, який знаходить застосування в різних аспектах кібербезпеки та тестування програмного забезпечення, допомагаючи виявляти та виправляти вразливості та забезпечувати безпеку web-додатків. [14]

3.3.1 Ключові поняття

FUZZ - це ключове поняття у використанні FFUF, яке вказує місце для вставки значень зі словника або списку в HTTP-запитах. Використовуючи "FUZZ", можна автоматично заміняти цей плейсхолдер різними значеннями під час генерації запитів.

Словник (Wordlist) - це файл, що містить список можливих шляхів або значень, які FFUF буде перебирати в запитах. Використання правильного словника дозволяє швидше виявляти вразливості або приховані ресурси.

Словники для використання з утилітою FFUF можуть бути зроблені самостійно, знайдені в Інтернеті або використані з публічно доступних джерел, наприклад:

- **RockYou:** один з найвідоміших та найширше використовуваних словників для атак на паролі. Він містить багато популярних паролів і є загальнодоступним.
- **SecLists**: проект, який містить багато різних словників для тестування на вразливості та фаззінгу. Вони охоплюють різні аспекти безпеки, такі як паролі, патерни, секретні ключі та багато інших. Список можна знайти на GitHub.
- **FuzzDB:** проект, який містить словники та патерни для фаззінгу різних видів даних та запитів. Він розроблений для тестування на вразливості web-додатків.

Цільова URL адреса - це URL-адреса цільового web-сайту або додатку, який необхідно просканувати. Встановлення правильного цільового URL допомагає FFUF зосередитися на конкретному ресурсі або додатку. [12]

3.3.2 Механізм роботи

Архітектура клієнт-сервер (рис. 3.1) є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених web-додатків і передбачає взаємодію та обмін даними між ними.

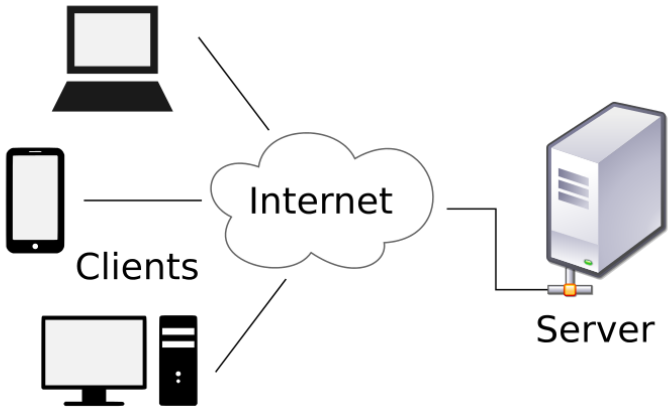

Рисунок 3.1 – Клієнт-серверна архітектура

Існує чимало протоколів, які регулюють процеси обміну інформацією в Інтернеті. Серед них основними є протоколи стеку TCP/IP та HTTP/HTTPS. Вони забезпечують обмін інформацією та її відображення у клієнта. [15]

HTTP (Hypertext Transfer Protocol) - це протокол передачі даних, який використовується для комунікації між web-клієнтами та web-серверами. Він є основою для передачі гіпертекстових документів, таких як web-сторінки, зображення, відео та інші ресурси. HTTP працює за схемою "запит-відповідь" (рис. 3.2).

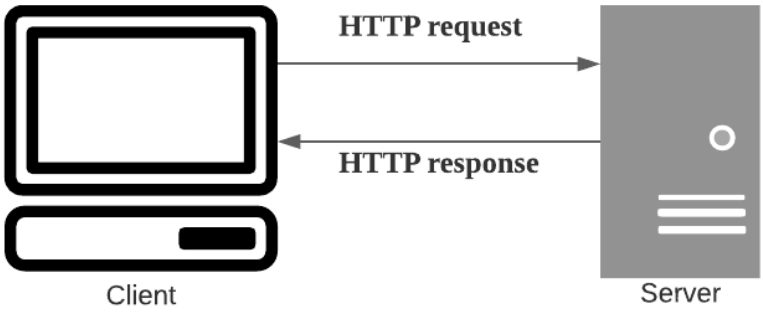

Рисунок 3.2 – Схема роботи протоколу HTTP

Коли клієнт відправляє запит на сервер, він включає різні методи (GET, POST, PUT, DELETE тощо), заголовки та інші параметри, щоб ідентифікувати необхідні ресурси та дії, які потрібно виконати.

Після того, як сервер отримує запит, він генерує відповідь, яка включає код статусу (наприклад, 200 для успішного запиту), заголовки та іншу необхідну інформацію. [16]

HTTP-запит має загальну структуру, що складається з наступних частин:

- Метод запиту
- URL
- Версія протоколу
- Заголовки
- Тіло запиту (за необхідності)

Простий приклад HTTP-запиту:

```
GET /path HTTP/Version
Host: exsample.com
User-Agent: App/Version (OS; version)
{body}
```

Основний принцип роботи FFUF (рис. 3.3) полягає в тому, щоб автоматично генерувати велику кількість різних HTTP-запитів на основі заданого списку словників з різними параметрами запиту. [16]

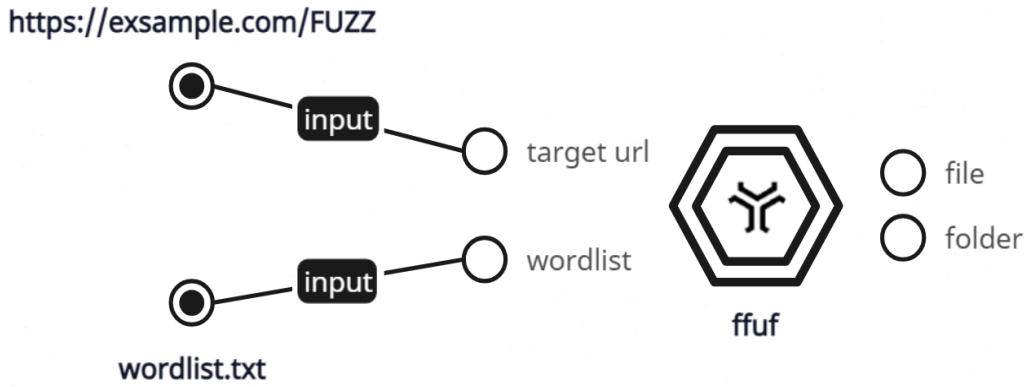

Рисунок 3.3 – Схематичне зображення механізму роботи утиліти FFUF

Як видно з рис.3.3, базові необхідні дані для формування запитів включають наступне:

- Цільова URL-адреса
- Словник

Ці дві основні складові надають достатньої інформації для того, щоб FFUF міг генерувати HTTP-запити та почати тестування.

Запит надсилається на вказану адресу, наприклад **http://exsample.com/.file**, використовуючи вказану версію протоколу HTTP.

Коли FFUF відправляє запити та отримує відповіді. Найпростіший шаблон відповіді серверу виглядає наступним чином [17]:

HTTP/Version Response Status
Date: Thu, 16 Apr 2024 10:20:01 GMT
Server: OS/Version
{body}

Утиліта аналізує їх і може фільтрувати результати залежно від налаштувань, які встановив користувач.

Типова інформація, що надає утиліта виглядає наступним чином:

[Status: 0, Size: 0, Words: 0, Lines: 0, Duration: 0 ms]

***FUZZ: .file**

Запит виконується з використанням плейсхолдера FUZZ, який в даному випадку шукає

Також надається наступна інформація:

- **Status:** HTTP-код стану відповіді сервера.
- **Size:** розмір відповіді, який сканер отримав від сервера.
- **Words:** кількість слів у відповіді від сервера.
- **Lines:** кількість рядків у відповіді від сервера.
- **Duration:** тривалість запиту, виміряна в мілісекундах.

Таким чином, відповідь на запит надає детальну інформацію, щодо файлу або ресурсу, який має шлях "https:/example.com/.file".

В залежності від розміру словника, утиліта надає різну кількість результатів. Зважаючи на це може виникнути проблеми під час аналізу даних такі як:

Обсяг даних

FFUF може згенерувати велику кількість результатів. Це залежить від кількості запитів, а також розміру словника. Аналіз такої великої кількості даних може бути затратним з точки зору обробки та зберігання.

Хибні спрацювання

FFUF може повертати багато хибних результатів, особливо якщо цільовий web-сайт має динамічну поведінку або функції, які можуть змінювати URL-адреси.

Частково уникнути цих проблем можливо за допомогою ретельного планування процесу фаззінгу, використання правильних параметрів FFUF та ефективної обробки та аналізу отриманих результатів. [14]

Таким чином, накладається ряд вимог до експерта, що проводить тестування використовуючи дану методологію та утиліту:

- Знання HTTP-протоколу.
- Знання функціоналу інструменту.
- Розуміння контексту та цілей аналізу.
- Знання технік тестування безпеки.
- Знання різних методів фаззінгу.
- Уважність та систематичний підхід.

Комбінація цих навичок дозволить аналітику максимально ефективно використовувати інструменти фаззінгу та мінімізувати можливі проблеми під час аналізу отриманих результатів. [14]

3.4 Моделювання середовища тестування

Моделювання середовища тестування - це процес створення віртуальних образів серверів, мереж та інших компонентів, які імітують реальне виробниче середовище, де розгорнуто додаток.

З іншого боку, такі середовища широко застосовуються для ізоляції процесів тестування від основних компонентів операційної системи під час проведення тестування на проникнення. Завдяки цьому зловмиснику або експерту з питань безпеки вдається забезпечити собі деяку анонімність.

Завдання належить до Black-Box тестування. Для його виконання та ізоляції процесів тестування від основних процесів системи було вирішено створити віртуальну машину (далі ВМ).

3.4.1 Створення та підготовка ВМ

Oracle VirtualBox - це безкоштовна та відкрита віртуалізаційна програма, яка дозволяє створювати та управляти ВМ на різних операційних системах. Вона підтримує різні гостьові операційні системи, включаючи Windows, Linux, macOS та інші. [18]

Процес створення ВМ ідентичний базовому алгоритму встановлення операційної системи (далі ОС). Ключові налаштування ВМ та встановленої ОС наведено в таблицях 3.2 та 3.3 відповідно.

Функціонал утиліти розглянуто в середовищі дистрибутиву Linux – Manjaro XFCE. [19]

Для полегшення процесу роботи під час тестування та забезпечення захисту інформації, було додатково встановлено VPN - Proton VPN. У таблиці 3.4 наведено детальну інформацію, щодо встановленого пакету програмних засобів.

Таблиця 3.2 – Параметри налаштування ВМ

Параметр	**Значення**	**Примітка**
Ім’я машини	FFUF	
Тип ос	Linux	
Версія ос	Arch x64 bit	Залежить від скачаного образу ос
Оперативна пам’ять	8000 мб	
Процесори	6 цп	
Розмір жорсткого диску	50 гб	
Графічний контролер	Vboxsvga	
Буфер обміну	З основної ОС в ВМ	Таке налаштування прискорить процес аналізу результатів
Операційна система	Manjaro xfce	

Таблиця 3.3 – Параметри налаштування ОС

Параметр	**Значення**	**Примітка**
Мова інтерфейсу	Українська	
Регіон	Європа	
Зона	Київ	
Мовні пакети	Ukrainian default	Англомовний пакет встановлено за замовчуванням
Пристрій зберігання	Virtualbox harddisk	
Метод встановлення ОС	З очисткою обраного диску	
Ім’я користувача	User	
Логін	User	
Назва цього комп’ютера	User-virtualbox	
Пароль	1234	
Пароль адміністратора	1234	

Таблиця 3.4 – Параметри налаштування ОС

Програма	**Версія**	**Джерело**	**Примітка**
Yay	12.1.3.1	Pacman	Використовується для роботи з пакетами AUR
FFUF	2.0.0	AUR	
Protonvpn		Pacman	

Після створення ВМ, було оновлено ОС, як це показано на рис. 3.4, до актуальної версії з використанням команди: **sudo pacman -Suuyy**.

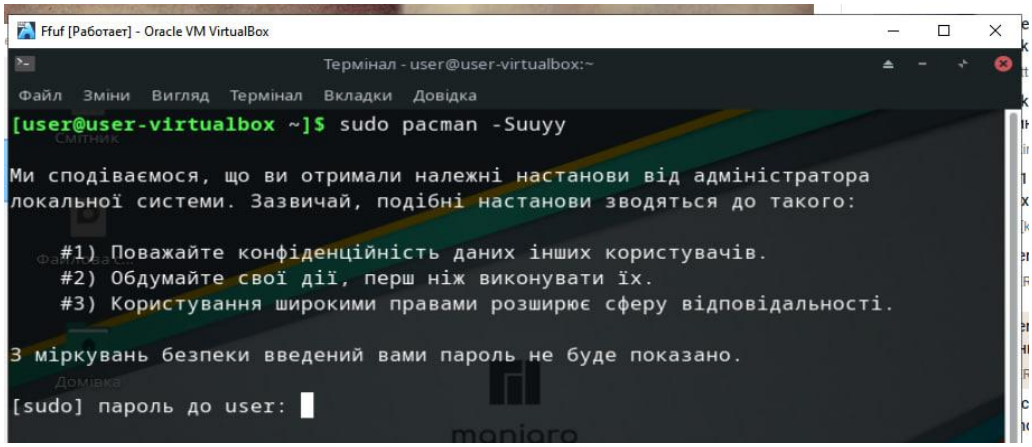

Рисунок 3.4 – Оновлення пакетів ОС

Перед використанням утиліти FFUF на ВМ встановлення VPN рекомендується з міркувань безпеки та конфіденційності. Тож для реалізації даної рекомендації було обрано Proton VPN. Ця утиліта була встановлена за допомогою команди: **sudo pacman -S protonvpn**.

3.4.2 Встановлення утиліти FFUF

Для встановлення утиліти FFUF на Linux, можна використовувати кілька методів.

Перший спосіб описано автором утиліти на сервісі GitHub, де і розміщено відкритий код програми. Таке встановлення є доволі простим, але може викликати ряд проблем, через відсутність автоматичного завантаження всіх необхідних залежностей. [13]

Команди для встановлення утиліти виглядають наступним чином:

go install github.com/FFUF/FFUF/v2@latest

git clone https://github.com/FFUF/FFUF ; cd FFUF ; go get ; go build

Другий спосіб полягає у використанні пакетного менеджеру дистрибутиву. Його перевага полягає у автоматичному завантаженні всіх додаткових пакетів, які необхідні для коректної роботи утиліти.

В Manjaro такий пакетний менеджер заміняє AUR (Arch User Repository), який є централізованою системою керування пакетами. [20]

Утиліту було встановлено другим методом з використанням наступної команди: **yay -S FFUF.**

Процес встановлення утиліти показано на рис. 3.5.

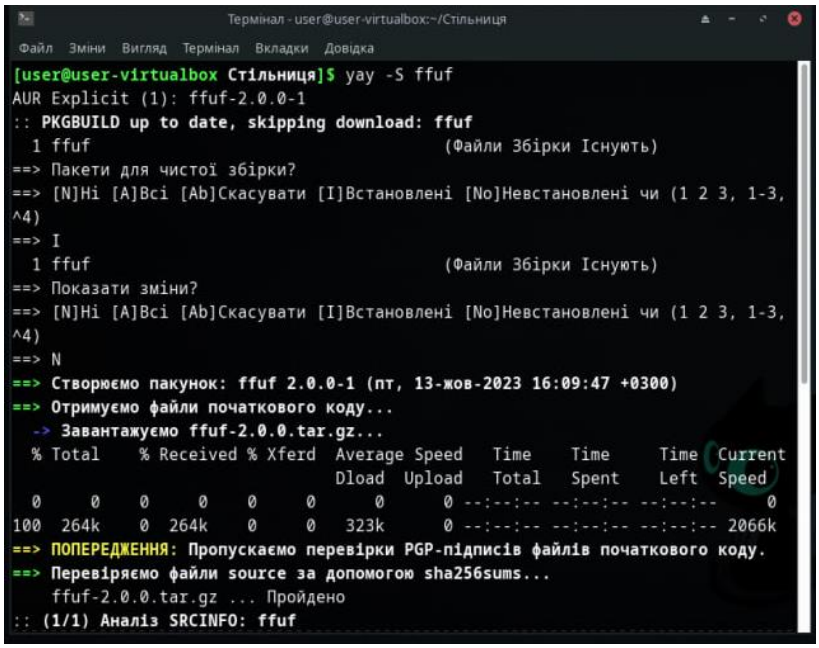

Рисунок 3.5 – Процес встановлення утиліти FFUF

Після закінчення встановлення перевірити роботу утиліти можна різними методами, наприклад виконавши наступну команду: **FFUF -h.**

3.5 Тестування корпоративних ресурсів

Для перевірки ефективності методології фаззінгу було проведено тестування web-ресурсу інтернет-магазину автозапчастин.

Крок 1. Пошук інформації про цільовий ресурс

Попередній пошук відкритої інформації є невід'ємною частиною Black box тестування. Це сприяє більш успішному та безпечному тестуванню і забезпечує кращу підготовку до нього.

Існує безліч спеціальних інструментів, таких як Whois, Shodan та багато інших, які допомагають збирати інформацію про домени, IP-адреси, технології та інші зв’язки з ресурсом даних.

А також утиліти такі як Nmap, можуть бути використані для визначення відкритих портів на сервері ресурсу. Це може дати представлення про те, які служби працюють на сервері та які вразливості можуть бути пов’язані з цими службами. [21]

Попередній аналіз ресурсу надав наступні результати:

- усі порти закриті, окрім стандартних портів, які зазвичай залишають відкритими;
- порт 21 також відкрито для обміну файлами по протоколу FTP.

Забігаючи наперед, саме ця інформація для нас є ключовою.

Крок 2. Тестування цільового ресурсу

Коректне налаштування запиту є критично важливим, оскільки воно визначає, які саме ресурси та параметри будуть виконуватися. Неправильно налаштований запит може призвести до пропуску вразливостей або отримання неточних результатів сканування.

Ось кілька причин, вважати даний крок важливим:

- Зосередження уваги на ділянках системи
- Раціональне використання ресурсів
- Мінімізація шуму та втрати часу

Утиліта надає багато результатів, кількість яких здебільшого залежить від розміру словника.

На серверах web-ресурсів часто існує багато зайвих файлів, які лише заважають швидко проаналізувати дані та виявити вразливості.

Для запобігання втраті важливих файлів серед великої кількості нерелевантних даних, необхідно оволодіти навичками ефективного фільтрування інформації. Це є ключовим аспектом у процесі налаштування запиту за умови, що не потребується додатковий контроль витрат ресурсів.

Основні ключі для керування утилітою, розглянемо на прикладі вже готової робочої команди.

FFUF -w dirs.lst –u https://.../FUZZ -fl 1 -fc 403 –c

Ключ **-w, --wordlist** вказує шлях до словника, який буде використовуватися для перебору, а **-u, --url** використовується для заповнення інформації про цільовий ресурс.

Ключі **-fl, --filter-status** та **-fc, --filter-words** дозволяють фільтрувати результати за довжиною вмісту HTTP-відповідей та за HTTP-статусами відповідно. Обидві команди виключають переліч ені параметри із загального списку результатів.

Ключ **-c** використовується для візуальної фільтрації результатів.

В утиліті FFUF кольорове виділення виводу має стандартні значення, що сприяють кращому розумінню результатів сканування:

- **Червоний:** використовується для позначення невдалих запитів.
- **Зелений:** використовується для успішних запитів.
- **Синій:** використовується для важливих повідомлень, які не є ані помилковими, ані абсолютно успішними.

Таке фільтрування дозволяє швидше аналізувати результати та в першу чергу звертати уваги на файли з успішними та помилковими запитами.

Це базові ключі, які дозволяють налаштовувати та керувати процесом сканування залежно від конкретних потреб користувача.

З використанням всіх перелічених вище налаштувань, було отримано результати, які наведено на рис. 3.6.

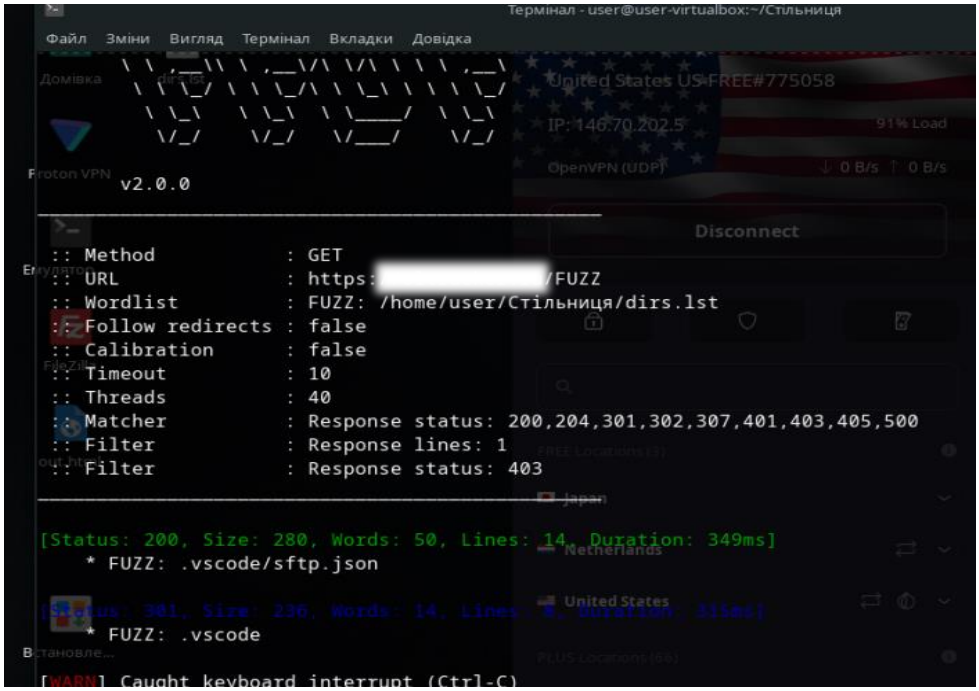

Рисунок 3.6 – Результати тестування

В даному випадку, було використано словник dirs.lst на 10 000 варіантів шляхів, що відповідно призвело до появи аналогічнох кількості результатів. Проте, використовуючи методи фільтрації, вдалося значно зменшити обсяг даних для аналізу. [22]

Крок 3. Збір результатів тестування

FFUF зазвичай виводить результати у текстовому форматі напряму в консоль, що може бути зручно для швидкого ручного аналізу. За необхідності надані результати можна експортувати в будь-який зручний формат та провести аналіз за допомогою інших інструментів, наприклад MS Excel.

FFUF підтримує різні формати для збереження результатів виводу, наприклад csv, html, txt, json та інші.

За цю функцію відповідають ключі **-o, --output** та **-of**, які задають назву вихідного файлу та його формат відповідно.

При аналізі відповідей слід в першу чергу звертати увагу на:

- Файли конфігурації (Configuration files).
- Файли журналу (Log files).
- Файли відповідей (Response files).

Всі ці типи файлів можуть стати вразливостями за відсутності надійного захисту.

Було проаналізовано наступну відповідь, що надала утиліта:

[Status: 200, Size: 280, Words: 50, Lines: 14, Duration: 349 ms]

***FUZZ: .vscode/sftp.json**

Відповідь вказує на успішний запит (Status: 200) з розміром відповіді 280 байт, що містить 50 слів та 14 рядків. Час відповіді склав 349 мілісекунд. Утиліта FFUF намагалася звернутися до файлу зі шляхом .vscode/sftp.json, використовуючи FUZZ в якості плейсхолдера для перебору можливих шляхів.

З урахуванням назви файлу, ймовірно, sftp.json вказує на конфігураційний файл для налаштування з'єднання до серверу, що забезпечує обмін файлами. Такі файли часто містять конфіденційні дані, такі як імена користувачів, паролі, адреси серверів та іншу чутливу інформацію.

Крок 4. Аналіз результатів тестування

Головною проблемою утиліти під час аналізу результатів є актуальність отриманих даних, що ускладнює процес виявлення потенційних вразливостей або проблем з безпекою додатку.

Перевірка актуальності даних є критично важливою, оскільки зміни у програмному чи апаратному забезпеченні та оновлення систем безпеки можуть значно впливати на наявність і значущість виявлних вразливостей.

Тож було вирішено провести перевірку актуальності отриманих даних шляхом перевірки вмісту знайденого файлу (рис. 3.7) та аналізу отриманої інформації.

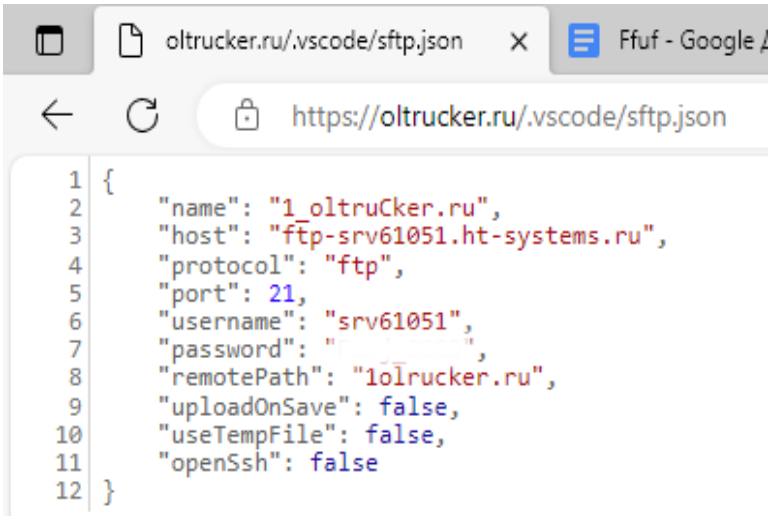

Рисунок 3.7 – Вміст файлу sftp.json

Файл містить конфігураційні дані для FTP-підключення. Щоб переконатися у їх актуальності, було проведено спробу підключитися до сервера за допомогою цих параметрів.

В залежності від обраної утиліти набір даних, що можуть знадобитися буде відрізнятися. Для прикладу було обрано утиліту CrossFTP. Вона потребує від користувача такі дані як: тип з'єднання, логін користувача, ім’я хоста, пароль, порт та remotePath.

Дані, що потребуються для з’єднання виведено нижче.

- protocol
- name
- host
- port
- username
- password
- remotePath

Як видно, ці всі дані було отримано з файлу sftp.json, тому залишається лише налаштувати з’єднання з сервером.

Щоб налаштувати з’єднання з сервером потрібно натиснути на відповідну іконку в правій частині екрану з позначенням інтернет мережі та ввести необхідні параметри, як це показано на рис. 3.8.

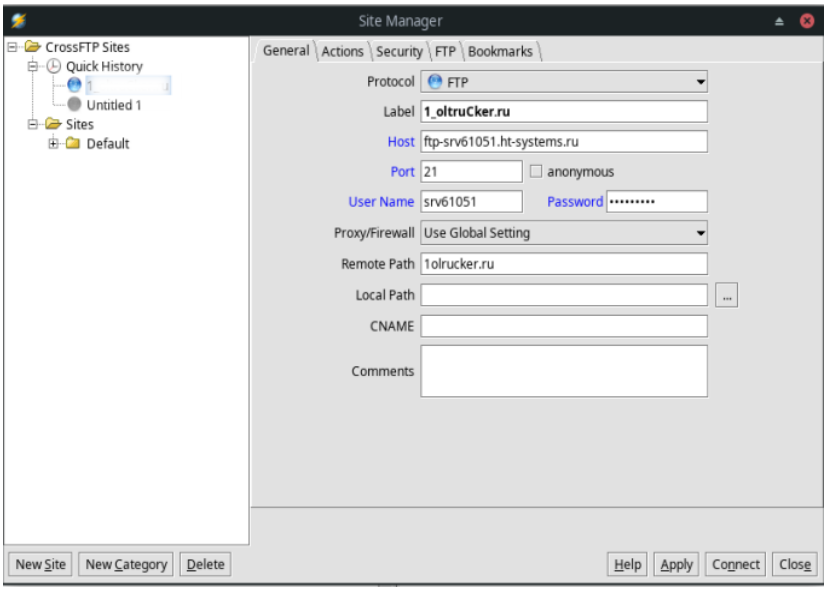

Рисунок 3.8 – Налаштування з’єднання до серверу FTP

Було отримано повний доступ до FTP серверу web-додатку (рис. 3.9). Така поведінка системи не є нормою і потребує негайного виправлення.

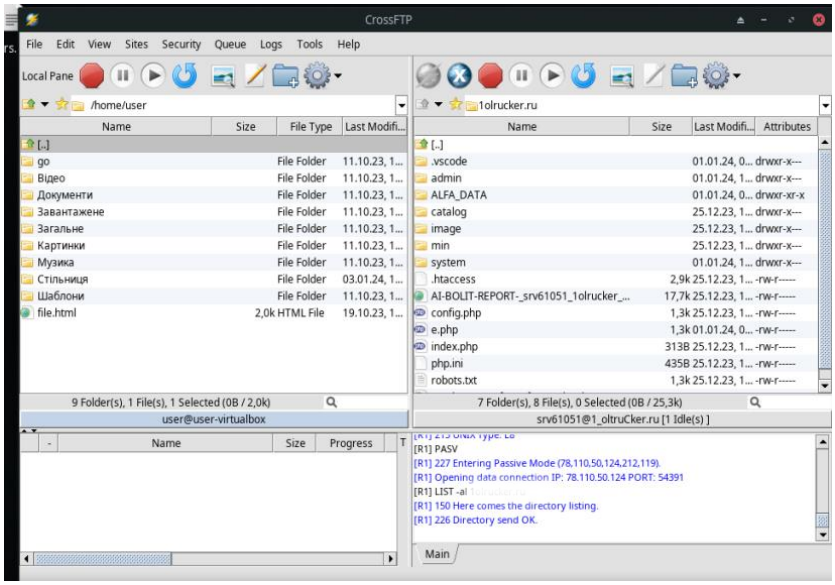

Рисунок 3.9 – Процес під’єднання до серверу FTP

Потенційний витік конфігураційних даних може призвести до серйозних проблем з безпекою. Зловмисники можуть використовувати ці дані для

несанкціонованого доступу до FTP-сервера, що може призвести до втрати конфіденційної інформації або навіть до втрати контролю над сервером.

Отримані дані є актуальними, а тому відсутність захисту на такому файлі є вразливістю, оскільки зловмисники зможуть будь-яким чином маніпулювати всією файловою системою.

Важливо дотримуватися найвищих стандартів безпеки при збереженні конфігураційних даних і переконатися, що доступ до таких файлів обмежений та захищений.

Для того, щоб усунути дану вразливість, перш за все, необхідно негайно обмежити доступ до цього файлу або взагалі видалити його та змінити всі дані для підключення до сервера, щоб унеможливити невповноваженим особам отримати доступ до системи.

Після цього рекомендується вжити наступних заходів для запобігання подібним проблемам у майбутньому:

- регулярне проведення аудиту безпеки;
- обмеження доступу до файлів;
- регулярна перевірка прав доступу до наявних ресурсів;
- впровадження механізмів шифрування конфіденційних даних;
- зміцнення механізмів аутентифікації та авторизації;
- регулярно проводити оновлювлення сайту
- регуряне проведення навчання персоналу.

Застосування цих заходів допоможе зменшити ризики безпеки, пов'язані з вразливістю, та забезпечити безпечну роботу web-додатку. [23]

ВИСНОВОК

У результаті виконання кваліфікаційної роботи бакалавра, було розглянуто основні концепції фаззінгу, інструменти, що застосовуються для його реалізації, та застосування цієї методики в сучасних умовах кібербезпеки.

Проведено огляд захисту інформації на основі методології фаззінгу: методу виявлення вразливостей у web-додатках.

Проведено дослідження ефективності застосування методології фаззінгу. Для цього були визначені цільовий ресурс та умови тестування.

У результаті було виявилено вразливість у додатку, яка полягає у відсутності механізмів захисту на критично важливому файлі – sftp.json. Він містить конфігураційні дані для налаштування підключення до FTP-серверу web-ресурсу. Ця вразливість дозволяє несанкціонованим особам отримувати доступ до конфіденційної інформації.

Для першочерногово усунення вразливості було надано рекомендацію негайно змінити дані користувача, а також обмежити доступ до вразливого файлу sftp.json.

Таким чином вдалося продемонструвати ефективність даної методології для вирішення поставлених питань захисту інформації та її кореляцію з заданою темою. Однак слід зазначити, що її ефективність все ще залежить від конкретного додатку та контексту використання.

Сама по собі методологія не може забезпечити повну картину щодо захищеності додатка. Для отримання повної картини з питань безпеки додатка необхідно також застосовувати інші методи захисту.

Методологія розвивається й інтегрується з іншими технологіями, що відкриває широкі перспективи у сферах тестування та захисту додатків. Деякі з потенційних напрямків використання фаззінгу включають наступне:

- Автоматизація процесів тестування.
- Машинне навчання та штучний інтелект.
- Розширення сфери застосування.
- Інтеграція з DevSecOps.

Таким чином, методологія фаззінгу має потенціал стати важливим інструментом в арсеналі кібербезпеки, що сприятиме розробці більш захищених та надійних програмних додатків.

Загалом, дослідження демонструє перспективність та актуальність вивчення даної методології як напряму для виявлення вразливостей та підвищення рівня захищеності інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Repository Simon Kuznets Kharkiv National University of Economics: Технології захисту інформації. Repository Simon Kuznets Kharkiv National University of Economics: Главная страница. URL: http://repository.hneu.edu.ua/bitstream/123456789/22547/1/Технології%20захисту%20інформації.pdf (дата звернення: 17.05.2024).
2. Comparative analysis of SAST and DAST. NCI Library. URL: https://norma.ncirl.ie/5956/1/lyubkadencheva.pdf (date of access: 23.05.2024).
3. Analysis and testing of Web applications. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/919078/ (date of access: 17.05.2024).
4. Автоматизоване тестування web-додатків. CORE – Aggregating the world’s open access research papers. URL: https://core.ac.uk/download/pdf/324214889.pdf (дата звернення: 17.05.2024).
5. Учасники проектів Вікімедіа. Фаззінг – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Фазинг (дата звернення: 17.05.2024).
6. What is Fuzzing?. Packetlabs. URL: https://www.packetlabs.net/posts/what-is-fuzzing/ (date of access: 17.05.2024).
7. Центр Систем Безпеки. Фаззінг. Частина 1: ідея, техніка. Хабр. URL: https://habr.com/ru/companies/ussc/articles/771778/ (дата звернення: 17.05.2024).

8. NahamSec. What is Fuzzing (using FFUF), 2024. YouTube. URL: https://www.youtube.com/watch?v=0v1CTSyRpMU (date of access: 17.05.2024)
9. 50 CVEs in 50 Days: Fuzzing Adobe Reader - Check Point Research. Check Point Research. URL: https://research.checkpoint.com/2018/50-adobe-cves-in-50-days/ (date of access: 17.05.2024).
10. Гах В. О. Методи та засоби тестування захищеності REST API. Фазінг REST API. DSpace :: ELAKPI :: Репозитарій КПІ ім. Ігоря Сікорського. URL: https://ela.kpi.ua/server/api/core/bitstreams/921b51ac-df1b-44c9-a206-9778723fb737/content (дата звернення: 17.05.2024).
11. veneramuholovka. Фаззінг – важливий етап безпечної розробки. Хабр. URL: https://habr.com/ru/companies/dsec/articles/450734/ (дата звернення: 17.05.2024).
12. FFUF Tool | Docs | Trickest. Automate OffSec, EASM, and Custom Security Processes. URL: https://trickest.com/docs/knowledge-hub/tools/FFUF/ (date of access: 17.05.2024).
13. Home. GitHub. URL: https://github.com/FFUF/FFUF/wiki (date of access: 17.05.2024).
14. Tech Raj. Hacking Websites with FFUF! (FUZZING), 2023. YouTube. URL: https://www.youtube.com/watch?v=eTj9A8c9tCM (date of access: 17.05.2024).
15. The OSI reference model. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/1457043 (date of access: 17.05.2024).
16. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc7231 (date of access: 17.05.2024).

17. HTTP: The Definitive Guide. Google Books. URL: https://cutt.ly/Sw7ZkTzm (date of access: 17.05.2024).
18. Documentation – Oracle VM VirtualBox. Oracle VM VirtualBox. URL: https://www.virtualbox.org/wiki/Documentation (date of access: 17.05.2024).
19. Manjaro. Main Page. Manjaro. URL: https://wiki.manjaro.org/index.php/Main_Page/ru (date of access: 17.05.2024).
20. AUR (en) - FFUF. AUR (en) - Home. URL: https://aur.archlinux.org/packages/FFUF (date of access: 17.05.2024).
21. Ethical Hacking and Penetration Testing Guide | Rafay Baloch | Taylor. Taylor & Francis. URL: https://www.taylorfrancis.com/books/mono/10.4324/9781315145891/ethical-hacking-penetration-testing-guide-rafay-baloch (date of access: 17.05.2024).
22. TheWay. Офіційний телеграм української хакерської групи C.A.S. Telegram. URL: https://t.me/cyber_anarchy_squad (дата звернення: 20.05.2024).
23. Cybersecurity for Critical Infrastructures: Attack and Defense Modeling. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/5477189 (date of access: 17.05.2024).

ДОДАТОК А.1 – Ключі керування утилітою

HTTP OPTIONS

-H	Заголовок `"Назва: значення"`, розділений двокрапкою. Приймаються кілька позначок -H.
-X	Метод HTTP для використання
-b	Дані файлів cookie `"NAME1=VALUE1; NAME2=VALUE2"` для копіювання як функції curl.
-cc	Клієнтський сертифікат для автентифікації. Щоб це працювало, необхідно також визначити ключ клієнта
-ck	Ключ клієнта для автентифікації. Щоб це працювало, потрібно також визначити сертифікат клієнта
-d	дані POST
-http2	Використовувати протокол HTTP2 (за замовчуванням: false)
-ignore-body	Не отримувати вміст відповіді. (за замовчуванням: false)
-r	Слідувати за перенаправленнями (за замовчуванням: false)
-raw	Не кодувати URI (за замовчуванням: false)
-recursion	Сканувати рекурсивно. Підтримується лише ключове слово FUZZ, URL (-u) має закінчуватися ним. (за замовчуванням: false)
-recursion-depth	Максимальна глибина рекурсії. (за замовчуванням: 0)
-recursion-strategy	Стратегія рекурсії: «за замовчуванням» для перенаправлення на основі та «жадібний» для рекурсії для всіх збігів (за замовчуванням: за замовчуванням)
-replay-proxy	Відтворення відповідних запитів за допомогою цього проксі.
-sni	Цільовий TLS SNI, не підтримує ключове слово FUZZ
-timeout	Тайм-аут запиту HTTP в секундах. (за замовчуванням: 10)
-u	Цільова URL-адреса

ДОДАТОК А.2 – Ключі керування утилітою

GENERAL OPTIONS

-V	Показати інформацію про версію. (за замовчуванням: false)
-ac	Автоматично калібрувати параметри фільтрації (за замовчуванням: false)
-acc	Спеціальний рядок автоматичного калібрування. Можна використовувати кілька разів. Має на увазі -ac
-ack	Ключове слово автокалібрування (за замовчуванням: FUZZ)
-acs	Спеціальні стратегії автоматичного калібрування. Можна використовувати кілька разів. Має на увазі -ac
-c	Розфарбувати вихід. (за замовчуванням: false)
-config	Завантажити конфігурацію з файлу
-maxtime	Максимальний час роботи в секундах для всього процесу. (за замовчуванням: 0)
-maxtime-job	Максимальний час роботи в секундах на завдання. (за замовчуванням: 0)
-noninteractive	Вимкнути функціональність інтерактивної консолі (за замовчуванням: false)
-p	Секунди затримки між запитами або діапазон випадкової затримки. Наприклад, "0,1" або "0,1-2,0"
-rate	Швидкість запитів за секунду (за замовчуванням: 0)
-s	Не друкувати додаткову інформацію (тихий режим) (за замовчуванням: false)
-sf	Зупинити, коли > 95% відповідей повертають 403 заборонено (за замовчуванням: false)
-t	Кількість одночасних потоків. (за замовчуванням: 40)
-v	Детальний висновок, друк повної URL-адреси та місця перенаправлення (якщо є) з результатами. (за замовчуванням: false)

ДОДАТОК А.3 – Ключі керування утилітою

MATCHER OPTIONS

-mc	Зіставляти коди статусу HTTP або "всі" для всього. (за замовчуванням: 200-299,301,302,307,401,403,405,500)
-ml	Збігається з кількістю рядків у відповіді
-mmode	Оператор набору відповідників. Будь-який із: і, або (за замовчуванням: або)
-mr	Відповідає регулярному виразу
-ms	Відповідає розміру відповіді HTTP
-mt	Відповідає кількості мілісекунд для першого байта відповіді, більше або менше. НАПРИКЛАД: >100 або <100
-mw	Відповідає кількості слів у відповіді

FILTER OPTIONS

-fc	Фільтрувати коди стану HTTP з відповіді. Список розділених комами кодів і діапазонів
-fl	Фільтрувати за кількістю рядків у відповіді. Розділений комами список кількості рядків і діапазонів
-fmode	Оператор набору фільтрів. Будь-який із: і, або (за замовчуванням: або)
-fr	Регулярний вираз фільтра
-fs	Фільтрувати розмір відповіді HTTP. Розділений комами список розмірів і діапазонів
-ft	Фільтрувати за кількістю мілісекунд до першого байта відповіді, більше або менше. НАПРИКЛАД: >100 або <100
-fw	Фільтрувати за кількістю слів у відповіді. Розділений комами список кількості слів і діапазонів

ДОДАТОК А.4 – Ключі керування утилітою

INPUT OPTIONS

-D	Режим сумісності зі списком слів DirSearch. Використовується в поєднанні з прапорцем -e. (за замовчуванням: false)
-e	Список розширень, розділених комами. Розширює ключове слово FUZZ.
-enc	Кодери для ключових слів, напр. "FUZZ:urlencode b64encode"
-ic	Ігнорувати коментарі списку слів (за замовчуванням: false)
-input-cmd	Команда, що створює вхідні дані. --input-num необхідний під час використання цього методу введення. Перевизначає -w
-input-num	Кількість входів для перевірки. Використовується в поєднанні з --input-cmd. (за замовчуванням: 100)
-input-shell	Оболонка, яка буде використовуватися для запуску команди
-mode	Режим роботи зі списком слів. Доступні режими: касетна бомба, вила, снайпер (за замовчуванням: касетна бомба)
-request	Файл, що містить необроблений http-запит
-w	Шлях до файлу Wordlist і (необов’язково) ключове слово, розділене двокрапкою. напр. '/path/to/wordlist:KEYWORD'

OUTPUT OPTIONS

-debug-log	Записати весь внутрішній журнал у вказаний файл.
-o	Записати вихідні дані у файл
-od	Шлях до каталогу для збереження відповідних результатів.
-of	Формат вихідного файлу. Доступні формати: json, ejson, html, md, csv, ecsv (або «всі» для всіх форматів) (за замовчуванням: json)
-or	Не створювати вихідний файл, якщо ми не маємо результатів (за замовчуванням: false)