МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ФІЛІПЧУК Михайло Михайлович

Алгоритми тестування безпеки веб-ресурсів/ Algorithms for testing security of web resources

спеціальність: 125 – Кібербезпека

освітньо-професійна програма –Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
М.М.Філіпчук

Науковий керівник
д.т.н., професор В.В.Яцків

Кваліфікаційну роботу допущено
до захисту:

«____» _____ 2022 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ – 2022

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 - Кібербезпека
освітньо-професійна програма –Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
____" _____ 2021 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

ФІЛІППЧУК Михайло Михайлович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми тестування безпеки веб ресурсів / Algorithms for testing security of web resources
керівник роботи д.т.н., професор В.В Яцків

затверджені наказом по університету від 31 грудня 2021 року № 606

2. Строк подання студентом закінченої випускної кваліфікаційної роботи 16 листопада 2022 року.

3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
– дослідити наявні вразливості у веб-ресурсах;
– провести аналіз існуючих алгоритмів та методів для тестування та запобігання;
– розробити алгоритми для тестування та аналізу вразливостей;
– провести тестування розроблених методів;
– порівняти результати проведених досліджень існуючими методиками;
– провести аналіз продуктивності та доречності розроблених алгоритмів.

5. Перелік графічного матеріалу у роботі:
- основні типи вразливостей у веб ресурсах;
- дерево синтаксичного аналізу для атаки на основі логічних значень(Boolean based);
- дерево синтаксичного аналізу для XSS атаки;
- вивід результатів тестування на XSS та SQL-ін'єкції;

- взаємодія модуля між іншими частинами системи;
- послідовна робота алгоритму по запобіганню hijacking;
- схема роботи методу по виявленню фішингових сайтів.

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 11 жовтня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Аналіз та дослідження наявних вразливостей	12.2021 р. – 03.2022 р.	
2	Алгоритми тестування безпеки	03.2022 р. – 05.2022 р.	
3	Тестування та аналіз алгоритмів безпеки	05.2022 р. – 11.2022 р.	

Студент _____ Філіпчук М.М.
(підпис)

Керівник роботи _____ д.т.н., професор В.В. Яцків
(підпис)

АНОТАЦІЯ

Магістерська робота на тему “Алгоритми тестування безпеки веб-ресурів” зі спеціальності 125 –кібербезпека написана обсягом 83 сторінки і містить 22 ілюстрації, 12 таблиць, 2 додатки та 29 джерел за переліком посилань.

Метою роботи є розробка алгоритмів тестування безпеки веб-ресурів від тієї чи іншої потенційної вразливості, які на сьогоднішній день є актуальними і потребують вирішення.

Методи досліджень є теорія ймовірності та математичної статистики, методи математичної логіки та обчислювальної математики, теорія інформаційної безпеки.

Досліджено сучасні підходи до розв’язання проблеми виявлення та протидії атакам на веб-ресурси. Розроблено методи виявлення та протидії Cross-Site-Scripting атакам та SQL-ін’єкціям. Розроблено алгоритм виявлення та протидії вразливості Broken Authentification. Розроблено програму для виявлення, збору та фільтрації фішингових сайтів.

Проведено експерементальні дослідження розроблених методик, для визначення точності та повноти виявлення атак розробленим комплексом. Отримані результати порівнено з показниками існуючих аналогів і проаналізовано.

Дослідження та їх результати, описані в даній роботі, становлять інтерес для розробників та власників веб-ресурсів, а також для представників наукової спільноти у сфері веб-безпеки.

КЛЮЧОВІ СЛОВА: РЕСУРСИ, МЕРЕЖА, СЕРВЕР РОБОТИ, ЗАПИТИ, НАВЧАННЯ, СЕСІЯ.

ABSTRACT

The master's thesis on the topic "Algorithms for testing the security of web resources" from the specialty 125 - cyber security is written in the volume of 83 pages and contains 22 illustrations,12 tables and 29 sources according to the list of references.

The purpose of the work is the development of algorithms for testing the security of web resources against one or another potential vulnerability, which are currently relevant and require a solution

The methods of research are the theory of probability and mathematical statistics, methods of mathematical logic and computational mathematics, the theory of information security.

Modern approaches to solving the problem of detecting and countering attacks on web resources are studied. Methods for detecting and countering Cross-Site-Scripting attacks and SQL-injections have been developed. An algorithm for detecting and countering the vulnerability Violed authentication has been developed. A program has been developed to detect, collect and filter fishing sites.

Conducting experimental studies to determine the accuracy and completeness of detection of attacks by the developed complex and comparing the obtained results with the indicators of existing analogues.

Experimental studies of the developed methods were conducted to determine the accuracy and completeness of attack detection by the developed complex. The obtained results are compared with the indicators of existing analogues and analyzed.

KEYWORDS: RESOURCES, NETWORK, JOB SERVER, REQUESTS, LEARNING, NEURAL NETWORK, SESSION.

ЗМІСТ

ВСТУП

Актуальність роботи. У зв’язку з величезним обсягом даних, збільшенням транзакцій та постійним зростанням користувачів у всесвітній мережі інтернет, належне тестування безпеки веб додатків є дуже важливою складовою цього складного механізму. Тестування безпеки може знадобитись для того щоб дані всіх користувачів залишались збереженими та конфіденційними, або ж наприклад у користувача була можливість виконувати лише ті запити та завдання, якими він уповноважений адміністратором того чи іншого веб-ресурсу.

Мета і завдання дослідження. Метою роботи є розробка алгоритмів тестування безпеки веб-ресурів від тієї чи іншої потенційної вразливості, які на сьогоднішній день є актуальними і потребують вирішення.

Досягнення визначеної мети передбачає вирішення таких завдань:

- дослідити існуючі наявні вразливості веб-ресурсів;
- проаналізувати вразливості, з’ясувати їх сутність;
- дослідити методи, які вже наявні та використовуються для уникнення даних вразливостей;
- розробити вдосконалені алгоритми для запобігання вразливостей;
- провести тестування розроблених алгоритмів на веб-ресурсах.

Об’єкт дослідження – процес тестування веб-ресурсів на наявність вразливостей.

Предмет досліджень – алгоритми тестування веб-ресурс на наявність вразливостей.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано**:** існуючі алгоритми та методи для тестування, методи математичної логіки та обчислювальної математики, теорія інформаційної безпеки.

Наукова новизна одержаних результатів:

- розроблено та протестовано алгоритми для протидії тим чи іншим актуальним вразливостям у веб-ресурсах;

– побудовано аналітичні вирази запропонованої схеми відновлення десяткового числа за його залишками. Чисельний експеримент показав, що розроблений алгоритм дозволяє збільшити швидкодію між базисного переходу.

Практична цінність одержаних результатів:

– обґрунтовано підхід та розроблено алгоритми для тестування та запобігання вразливостям;

– реалізовано програмне забезпечення для виявлення найбільш актуальних вразливостей та їх аналізу.

Публікації та апробація КР.

1. Філіпчук М.М. Алгоритм захисту веб-ресурсів. Матеріали наукової конференції «Автоматизація та комп’ютерно-інтегровані технології» (АКІТ - 2022), Тернопіль, 2022. – С.110-113.

2. Філіпчук М.М. Алгоритми тестування безпеки веб-ресурсів. Матеріали наукової конференції «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2022), Тернопіль, 2022. – С.65-67.

1 АНАЛІЗ ТА ДОСЛІДЖЕННЯ НАЯВНИХ ВРАЗЛИВОСТЕЙ

1.1 Загальний аналіз вразливостей у веб-додатках

На сьогоднішній день вразливості у веб-додатках є найбільш поширеними та потребують сучасних рішень які допоможуть захистити дані того чи іншого користувача, або ж захистити сам веб-ресурс від зовнішнього втручання збоку зловмисників, яких з кожним роком стає все більше, а отже і способів та варіантів для запровадження цих вразливостей теж зростає.

Недооцінювати серйозності реалізації цих вразливостей є помилкою, а також одним із багатьох чинників низького рівня захищеності. Серед багатьох типів додатків, саме веб-додатки вимагають більшої безпеки, адже саме там ми спілкуємося з людьми, тим самим інколи надаючи конфіденційну інформацію серверу, який зберігає повідомлення у базі даних; робимо онлайн-транзакції у інтернет магазинах.

Для побудови досконалих та практичих алгоритмів, а також побудови правильної організації веб-захисту потрібно розібратись як саме виникають ці вразливості, зрозуміти їхню сутність, принцип дії та межі їх розповсюдження [8].

1.2 Види атак на веб-ресурси

1.2.1. XSS атака або між міжсайтовий скриптинг

Міжсайтовий сценарій (XSS) – це атака, під час якої зловмисник вставляє шкідливі виконувані сценарії в код довіреної програми або веб-сайту. Зловмисники часто ініціюють атаку XSS, надсилаючи зловмисне посилання користувачеві та спонукаючи користувача натиснути його. Якщо додаток або веб-сайт не очищають належним чином дані, зловмисне посилання виконує обраний зловмисником код у системі користувача [3].

Для уникання XSS атак важливо також дотримуватись наступних правил:

- будьте обережними з полями для вводу та вводьте конфіденційну

інформацію тільки на перевірених ресурсах;

- реалізуйте вихідне кодування;
- дотримуйтеся принципу глибокого захисту;

Також Cross-Site-Scripting атака має три основних типи:

- reflected-Cross-Site Scripting;
- stored Cross-Site Scripting;
- DOM-based Cross-Site Scripting.

Відображені (непостійні) атаки XSS відбуваються, коли зловмисне корисне навантаження включається в запит, надісланий до вразливої веб-програми, а потім відображається таким чином, що HTTP-відповідь сервера складається з корисного навантаження. Зловмисники використовують методи соціальної інженерії, такі як фішингові атаки, щоб змусити жертву включити шкідливий сценарій у свій запит до веб-сервера. Потім браузер жертви виконує шкідливий сценарій як HTTP-відповідь [7].

Під час атаки Stored XSS уразлива веб-програма отримує дані користувача з ненадійних джерел і зберігає їх. Цей шкідливий вміст також включається в пізніші HTTP-відповіді, надіслані сервером.

Щоб виконати збережену XSS-атаку, хакерам потрібно лише виявити вразливість у системі безпеки, яка дозволяє виконувати зловмисні запити. Це робить його більш придатним для використання, оскільки хакерам не потрібно створювати зовнішні методи для надання ненадійних вхідних даних цільовому серверу додатків.

Міжсайтовий сценарій на основі DOM – це тип уразливості, який з’являється в об’єктній моделі документа сторінки HTML. Він суттєво відрізняється від відображеного та збереженого XSS, оскільки під час атаки цього типу розробник не може знайти шкідливий сценарій у вихідному коді HTML або відповіді HTML. Його можна спостерігати лише на час виконання. Зловмисники зазвичай використовують цю техніку, коли введені користувачем дані перевіряються на теги сценарію. Візьмемо, наприклад, програму, яка приймає ім’я у полі введення та відображає повідомлення з введеними даними

текстове значення, коли користувач натискає «Надіслати». Якщо ми перевіримо відображене повідомлення, ми побачимо, що воно вбудоване у тегу <pre>.

Вихідне кодування також є ключовим для запобігання вразливостям XSS. Використовуйте бібліотеки вихідного кодування, які відповідають мовам програмування та структурам, які використовує ваша організація. Крім того, переконайтеся, що розробники залишаються в курсі передових методів запобігання XSS. Принцип роботи XSS атаки зображений на рисунку 1.1.

Рисунок 1.1 – Принцип роботи XSS

Якщо веб-програма використовує недезінфікованого користувача та відображає так само, як і вихід, програма може вважатися вразливою. XSS-атаки найчастіше зустрічаються на мові програмування JavaScript.

Проілюструвати це допоможе приклад. Загальнодоступний блог або розділ коментарів на форумі приймає введені користувачем дані та зберігає їх у своїй базі даних і відображає той самий вміст іншим користувачам, які відвідують блог або форум. Ці типи програми є ідеальною мішенню для зловмисників, які можуть

вставити шкідливий сценарій у поле введення та надіслати це так, що сценарій буде збережено в базі даних. Кожного разу, коли користувач відвідує блог або форум, скрипт виконується у веб-переглядачі цього користувача як частина веб-сторінки, надаючи зловмиснику доступ до конфіденційних даних користувача.

Важливо впровадити заходи безпеки на початку життєвого циклу розробки програми. Наприклад, виконувати заходи безпеки на етапі розробки програмного забезпечення, такі як аналіз ризиків архітектури та моделювання загроз. Не менш важливо провести тестування безпеки після завершення розробки програми.

1.2.2. SQL-ін’єкції

Атаки за допомогою SQL є однією з найпоширеніших загроз для веб-ресурів на сьогодні. SQL Injection є різновидом атаки, під час якої зловмисник використовує команди SQL, щоб отримати доступ або змінити дані. Це дозволяє зловмиснику отримати несанкціонований доступ до бази даних для зміни намічених запитів [23]. Внутрішня база даних часто містить конфіденційну інформацію, таку як номери безпеки, номери кредитних карток, фінансові дані, медичні дані [4]. Як правило, веб-користувач надає таку інформацію, як ім’я користувача та пароль. Ось деякі методи, за допомогою яких виконуються оператори SQL вводять у вразливі системи:

- введення ін’єкції через поле вводу;
- введення ін’єкції через серверні змінні;
- введення ін’єкції за допомогою полів у cookie сховищі;
- ін'єкція другого порядку.

У веб-середовищі конфіденційність кінцевого користувача є одним із найбільш суперечливих правових питань [5]. Зловмисник може здійснити цю атаку з багатьма різними намірами, наприклад:

1. Для отримання необхідних даних зловмисник має змогу отримати інформацію, збережену безпосередньо у базі даних, отримуючи доступ до даних когось з адміністрації. Тобто, виконується атака на базу даних вищого рівня, в

даному випадку – адміністрації. Якщо база адміністратора взламана, вся база даних стає вразливою.

2. Для отримання доступу до даних – зловмисники роблять все для того щоб отримати необхідні привілеї, права на управління та маніпулювання даними.

3. Відбиток пальця в базі даних – у цій атаці буде визначено версію та тип бази даних з боку нападника. Ця атака допомагає їм пробувати різні типи запитів у різних програмах.

4. Знайдено ін’єкційні параметри – за допомогою деяких автоматичних інструментів будуть знайдені уразливі параметри.

5. Обхід автентифікації – механізми автентифікації програми будуть обійдені для входу всередині бази даних.

5. За допомогою обходу автентифікації та механізмів автентифікації програми буде здійснено обхід всередині бази даних

Для кращого розуміння та аналізу роботи скрипту розглянемо наступний код, який виконується для отримання користувача на основі введених даних:

SELECT * FROM users WHERE login=’TestUser’ AND password=’12345’

На рисунку 1.2 зображено вікно входу, на якому користувач вводить вищевказані дані.

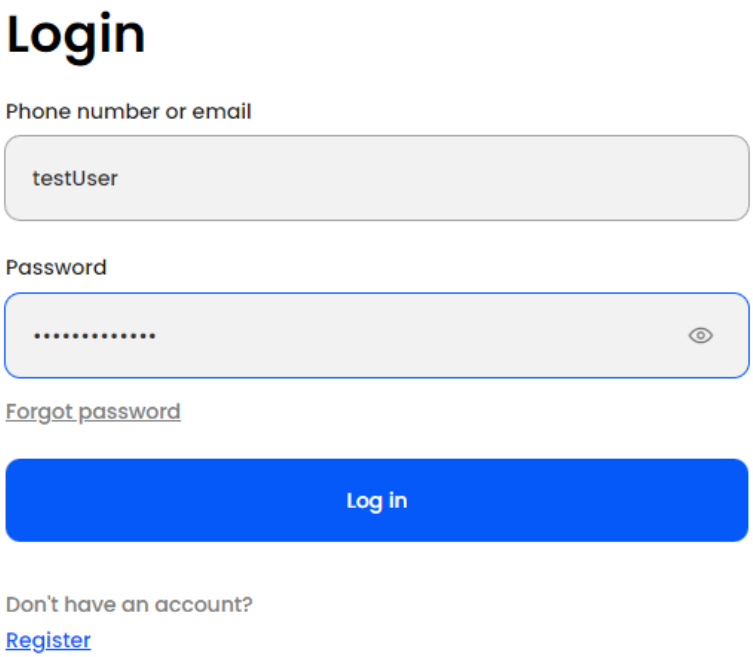

Рисунок 1.2 – Вікно входу користувача.

Це стандартний логічний запит, який дістане користувача із бази даних, якщо логін та пароль будуть співпадати [2].

Тут, коли користувач надаючи своє ім’я користувача та пароль, для перевірки генерується SQL-запит у серверній частині.

На рисунку 1.3 показано вікно зі сценарієм атаки. Тобто зловмисник замість того, щоб вводити дійсне ім’я користувача, використовує поле ін’єкції, наприклад «vulnerability» АБО «1’=’1’—» як ім’я користувача та «щось» як пароль. Після введення, сервер передасть до бази даних наступний код:

SELECT * FROM users WHERE login=’vulnerability’

or ‘1’=’1’ –‘ AND password=’testpass111’

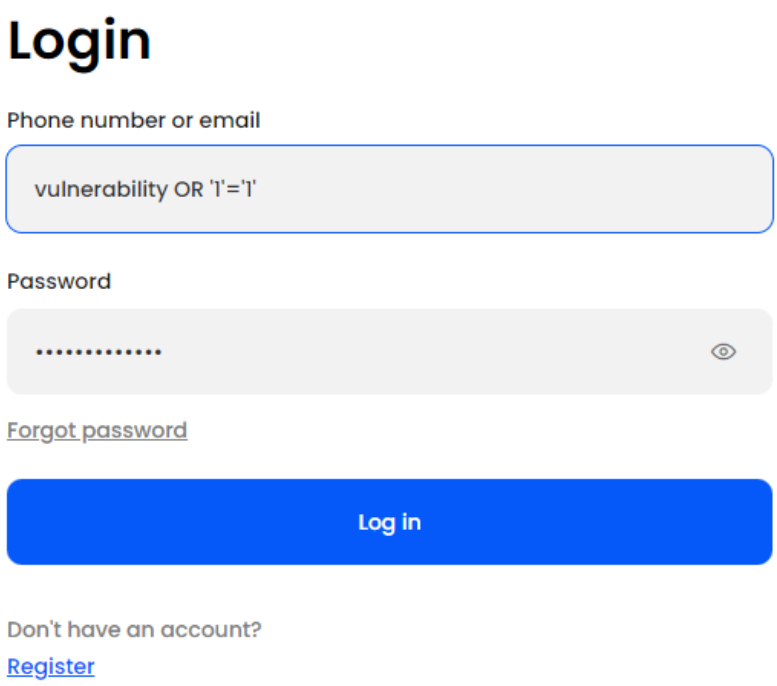

Рисунок 1.3 – Вікно входу користувача з підміною поля

Таким чином дійсна логіка запиту буде зламана, і база даних поверне користувача з таким іменем, навіть без паролю і зловмисник отримає доступ.

1.2.3 Зламування автентифікації (Broken authentification)

Ще одна, не менш загрозлива вразливість яка є нагальною проблемою будь якої програми. Адміністратору слід переконатись що певний користувач є справді тим, за кого себе видає, адже правильно реалізована зламана аутентифікація є досить серйозною вразливістю у плані безпеки веб-додатків [9, 21]. Якщо коротко, то порушена або зламана автентифікація це взагалі є загальний термін для різноманітних вразливостей у системі, які зловмисники стараються впровадити щоб видати себе за звичайного користувача [1, 22]. Тому правильним рішенням було б розділити ці вразливості по окремості та проаналізувати.

1. Викрадення сеансу (hijacking) – це техніка, яку використовують хакери для отримання доступу до комп'ютера або облікових записів цільової особи. Під час атаки з перехопленням сесії хакер бере під контроль сеанс веб-перегляду користувача, щоб отримати доступ до його особистої інформації та паролів. Для уникання такого виду атаки слід дотримуватись наступних запобіжних рекомандацій:

- використання VPN. Це допоможе запобігти перехоплення трафіку збоку зловмисників, ускладнюючи викрадання інформації у сесії;

- використання HTTPS забезпечує шифрування SSL/TLS у всьому трафіку сеансу. Зловмисники не зможуть перехопити відкритий текстовий ідентифікатор сеансу, навіть якщо трафік жертви відстежувався;

- використання атрибуту HtppOnly запобігає злому збережених файлів а також сесій авторизації, це допоможе уникнути викрадення сеансу але через це зловмисники можуть вдатись до атаки на міжсценарного скриптингу;

- двофакторна аутентифікація та складні паролі можуть також бути у нагоді, навіть у випадку якщо сесію було перехоплено;

- постійне оновлення програмного та апаратного забезпечення з найновішими методами в плані безпеки;

- зміна ключа сеансу після успішного входу ускладнює викрадачам сеансу стежити за сеансом користувача, навіть якщо він знає оригінальний ключ. Навіть

якщо зловмисник надсилає фішингове посилання, на яке натискає користувач, зловмисники не можуть захопити сеанси за допомогою самостійно згенерованих ключів у таких налаштуваннях.

Рекомендації OWASP щодо порушення автентифікації чітко стверджують, що ідентифікатор сеансу, виданий користувачеві, який увійшов у систему, тимчасово еквівалентний вихідним обліковим даним користувача для входу. Крім того, його можна легко використати, щоб видати себе за користувача в додатку.

Такими ідентифікаторами сеансу потрібно ретельно керувати. Хакери, швидше за все, маніпулюють будь-якими слабкими місцями або лазівками.

2. Надсилання облікових даних (credential stuffing) – ще один відомий метод кібератаки у плані зламування аутентифікації під час якого зловмисник може використати списки скомпрометованих даних користувачів для доступу у систему [1]. Часто використовуються боти для того щоб вже відбувалось автоматично і машстабовано(багато ботів). Сам процес роботи цієї атаки наступний:

- зловмисник налаштовує бота, який буде автоматично входити в кілька облікових записів паралельно, підроблюючи IP адреси;

- запуск автоматизованого процесу на багатьох сервісах, щоб перевірити правильність та валідність даних на багатьох веб-сайтах;

- зловмисник або бот відстежує успішні входи на веб-ресурсах та отримує особисту інформацію;

- зловмисник зберігає отриману зламану інформацію для її подальшого використання, наприклад під час фішингових атак та ін.

Для уникання даного виду атаки можна вдатись до наступних методів:

– використання різних унікальних та складних паролів на різних сервісах. У такому разі, якщо зловмисник отримає пароль, він не буде співпадати на всіх ресурсах;

– багатофакторна автентифікація (MFA) є, безперечно, найкращим захистом від більшості атак, пов’язаних із паролями, зокрема підміни облікових

даних і розпилення паролів, а аналіз, проведений Microsoft, показує, що вона зупинила б 99,9% зламу облікових записів. Замість того, щоб мати лише один пароль для захисту облікового запису, двофакторна автентифікація (2FA) або багатофакторна автентифікація (MFA) запитує одну або кілька додаткових частин інформації для входу на додаток до пароля. Ця додаткова інформація може приймати різні форми. Таким чином, його слід впроваджувати скрізь, де це можливо; однак, залежно від аудиторії програми, застосування MFA може бути непрактичним або неможливим;

– відстеження витоку облікових даних. Постачальники послуг можуть використовувати рішення безпеки, які автоматично відстежують дані для входу користувачів у порівнянні з величезними базами даних витоку облікових даних, загальнодоступних у темній мережі. Якщо буде виявлено викрадені облікові дані, які збігаються з даними для входу користувачів, цих користувачів можна негайно повідомити. Такі рішення не обмежуються постачальниками послуг. Кінцеві користувачі також можуть вводити свої адреси електронної пошти на HaveIBeenPwned.com, щоб перевірити (безкоштовно!), чи були будь-які облікові записи, пов’язані з адресою електронної пошти, скомпрометовані в будь-яких попередніх витоках даних. Якщо збіг знайдено, користувач повинен негайно змінити всі свої паролі, які ідентичні або схожі на зламаний пароль, щоб запобігти надходженню облікових даних. Однак така перевірка працює, лише якщо зламана база даних була опублікована в Інтернеті. Він не може виявити скомпрометовані облікові дані, які були продані приватно та ніколи не публікувалися.

На рисунку 1.4 зображений сам процес атаки з надсиланням скомпрометованих даних, де зловмисник надсилає ці дані ботам, а вони в свою чергу використовують їх на різних багатьох веб-ресурсах.

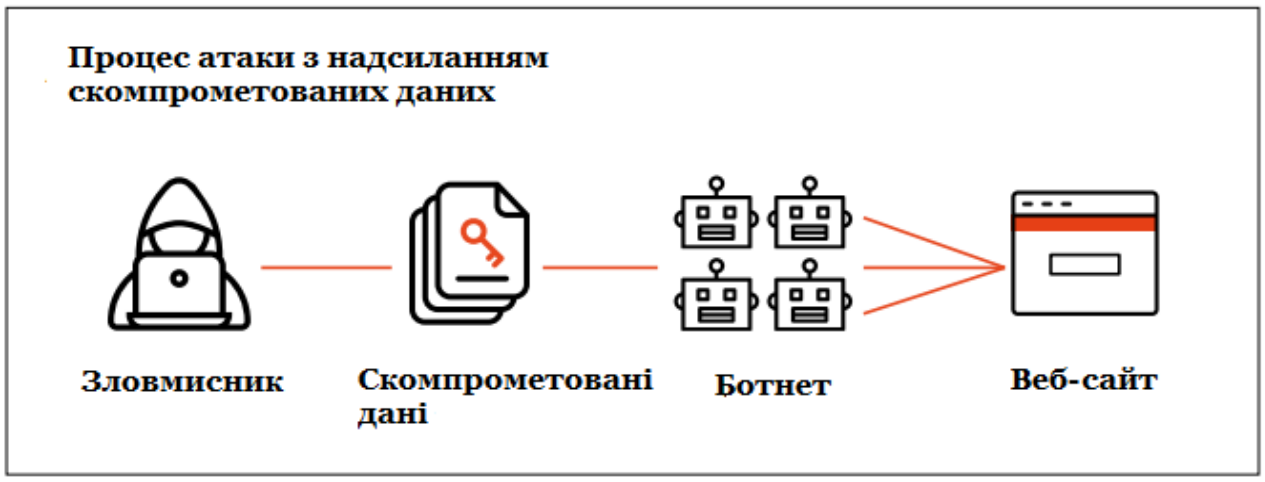

Рисунок 1.4 – Процес атаки з надсиланням даних

3. Password spraying – це вид атаки, яку використовують зловмисники, щоб отримати доступ до облікового запису жертви спробувати паролі, які користувачі, швидше за все, використовуватимуть. Зловмисники за одну спробу можуть спробувати отримати доступ до кількох облікових записів, перевіривши їх на декілька часто використовуваних паролів (наприклад, пароль, 12345 тощо). На відміну від атаки повним перебором(brute foce), цей підхід виявився дуже успішним, оскільки він дозволяє зловмисникам залишатися прихованими та уникати швидке блокування облікового запису [21]. Розпорошення паролів є поширеним методом, який використовують кіберзлочинці для отримання несанкціонованого доступу до комп'ютерних систем. Наприклад, у звіті IBM/The Ponemon Institute за 2020 рік про вартість витоку даних було встановлено, що 19% усіх витоків даних були результатом слабких або скомпрометованих облікових даних.

У звіті Verizon про порушення даних за 2020 рік було виявлено, що понад 80% усіх порушень, пов'язаних зі зломом, стосувалися методів грубої сили, наприклад розпилення пароля [22]. На рисунку 1.5 показаний приклад такої атаки, та її відмінності від атаки з повним перебором.

Рисунок 1.5 – Відмінності від атаки з повним перебором

Зловмисники, як правило, виконують три кроки, щоб успішно здійснити атаку з розпиленням пароля, зокрема:

- зловмисники спочатку спробують отримати список імен користувачів різними способами. Компанії часто використовують формалізовані імена користувачів, якими зазвичай є адреси електронної пошти користувачів. Наприклад, поширений формат електронної пошти: ім’я.прізвище@компанія.com. Зловмисники часто можуть знайти цю інформацію, шукаючи підказки на веб-сайті компанії або переглядаючи веб-сайти соціальних мереж, наприклад LinkedIn. У деяких випадках зловмисники можуть придбати список імен користувачів у темній мережі. Зловмисники зазвичай використовують програмне забезпечення, яке може перевірити точність імен користувачів перед здійсненням атаки;
- отримавши список імен користувачів, вони отримають список найбільш часто використовуваних паролів і почнуть розпилювати. Вони також можуть налаштувати список відповідно до певних факторів, таких як географічний регіон, де знаходяться користувачі, що може брати до уваги регіональні діалекти, популярні регіональні спортивні команди/гравці тощо. Щоб уникнути спрацьовування будь-яких тривог, зловмисники зазвичай чекають принаймні 30 хвилин, перш ніж спробувати знову;

- якщо припустити, що атака розпилення була успішною і зловмисник тепер має доступ до одного або кількох облікових записів, він почне досліджувати тип доступу, який він має, зокрема, до яких систем, даних і програм він має доступ. Вони також намагатимуться використовувати свій доступ, щоб підвищити свої привілеї, що зазвичай передбачає використання вразливостей програмного забезпечення, неправильних налаштувань або виявлення будь-яких слабких засобів контролю доступу. Враховуючи те, що тепер вони мають легітимний доступ до мережі, включаючи «власний» обліковий запис електронної пошти, вони також можуть спробувати підвищити свої привілеї, надсилаючи електронні листи колегам із привілейованими обліковими записами та намагаючись обманом змусити їх передати свої облікові дані. По суті, будь-яка додаткова інформація, яку вони можуть отримати про мережу, дасть їм більше шансів досягти своєї мети.

Наприклад маючи необхідні дані, зловмисник може спробувати ввести найпоширеніші паролі для кожного зі знайдених користувачів [9]. Якщо у зловмисника є облікові дані користувача або доступ до оболонки як користувача домену, він можете отримати політику паролів за допомогою наступного коду:

```
# From Linux

crackmapexec <IP> -u 'user' -p 'password' --pass-pol

enum4linx -u 'username' -p 'password' -P <IP>

rpcclient -U "" -N 10.10.10.10;

rpcclient $>querydominfo

ldapsearch -h 10.10.10.10 -x -b "DC=DOMAIN_NAME,DC=LOCAL" -s sub
"*" | grep -m 1 -B 10 pwdHistoryLength

# From Windows

net accounts

(Get-DomainPolicy)."SystemAccess" #From powerview
```

Успішне захоплення облікового запису дає змогу шахраям використовувати зламаний обліковий запис за бажанням. На додаток до фінансової вигоди від виведення коштів з облікового запису, шахраї можуть отримати доступ до збережених даних, таких як адреси електронної пошти зі зламаного облікового запису, щоб атакувати більше користувачів [11].

Хоча атаки з розпиленням пароля є поширеними, їм можна запобігти. Наведені нижче технології позбавлять кіберзлочинців можливості використовувати розпилення паролів для зламу систем будь-якої організації.

1. Автентифікація без пароля. Безпарольна автентифікація — це метод автентифікації, який дозволяє користувачеві отримати доступ до програми чи IT-системи без введення пароля чи відповідей на таємні запитання. Замість цього користувач надає будь-яку іншу форму доказу, наприклад відбиток пальця, значок наближення або код апаратного маркера. Автентифікація без пароля часто використовується в поєднанні з багатофакторною автентифікацією (MFA) і рішеннями Single Sign-On для покращення взаємодії з користувачем, посилення безпеки та зменшення витрат і складності IT-операцій.

2. Мультифакторна аутентифікація. Як випливає з назви, багатофакторна автентифікація (MFA) — це використання кількох факторів для підтвердження особи особи, яка запитує доступ до програми, веб-сайту чи іншого ресурсу. Багатофакторна автентифікація – це різниця між, наприклад, введенням пароля для отримання доступу та введенням пароля та одноразового пароля (OTP) або пароля та відповіді на секретне запитання. Вимагаючи від людей підтверджувати особу кількома способами, багатофакторна автентифікація забезпечує більшу впевненість, що вони справді ті, за кого себе видають, що зменшує ризик несанкціонованого доступу до конфіденційних даних. Зрештою, одна справа ввести викрадений пароль, щоб отримати доступ; зовсім інша справа – ввести викрадений пароль, а потім також вимагати ввести одноразовий пароль, надісланий на смартфон законного користувача.

3. Фішингова атака (phishing attack). Термін «фішинг» походить від слова «рибалка». На відміну від рибалок, фішери не ловлять рибу. Фішери

шукають цінну інформацію. За визначенням, фішинг – це спроба викрасти конфіденційні дані шляхом обману особи, щоб розкрити паролі чи дані кредитної картки, або завантажити комп'ютерний вірус. Фішингові атаки починаються з електронного листа. Фішингове повідомлення може бути спокусливим та запропонувати або вимагати негайних дій, щоб змусити вас заповнити якусь електронну форму або ж перейти за невідомим посиланням, яке може бути замасковане під надійне [12, 24]. Ви можете отримувати електронні листи, у яких повідомляється, що ваш (електронний, банківський або будь-який інший) обліковий запис буде видалено, якщо ви негайно не змініте пароль. І це легко – ви можете просто скористатися формою, що додається. Фішери намагаються будувати та підвищувати тиск, щоб спонукати вас до імпульсивних дій. Пам'ятайте, ніколи не відповідайте на електронні запити щодо паролів, кодів безпеки, пін-кодів тощо. На рисунку 1.6 зображено приклад фішингової атаки.

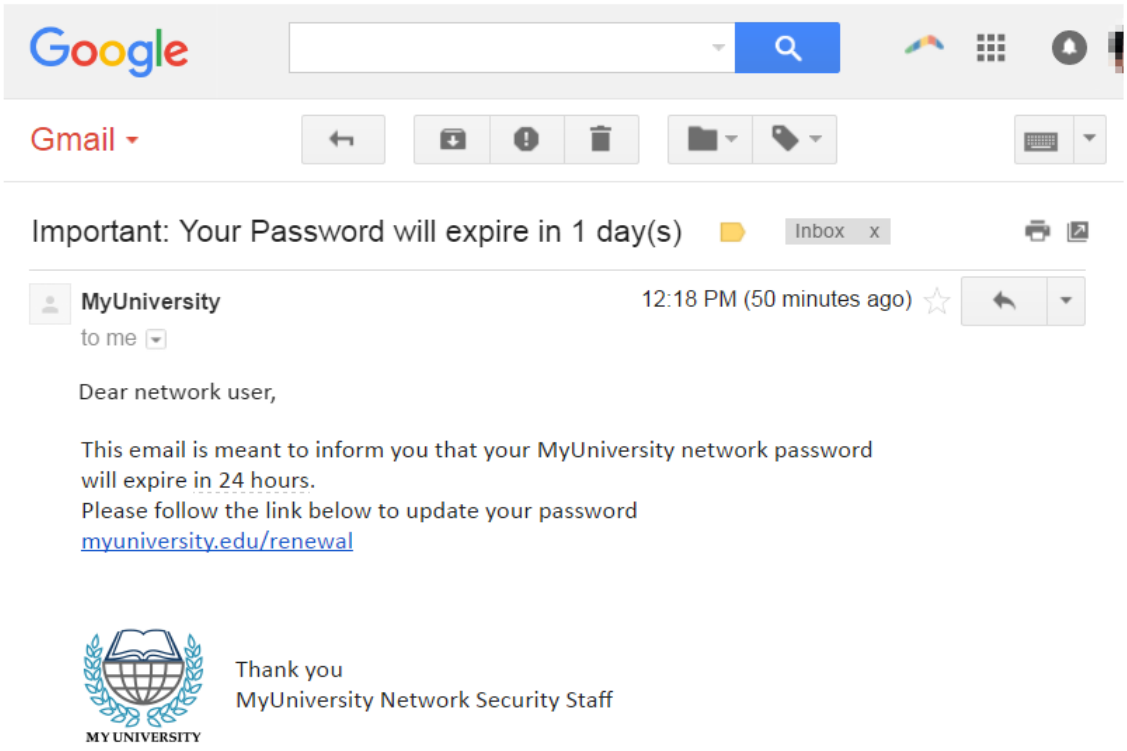

Рисунок 1.6 – Приклад фішинг-атаки з використанням електронного листа

На рисунку зображено підроблений електронний лист, який повідомляє що

нібито термін дії паролю на сайті університету закінчиться через 24 години, тому щоб його оновити потрібно перейти за посиланням. Саме це посилання і є методом фішингу, і при переході за ним зловмисник може викрасти дані. Сценаріїв такого процесу може бути багато, але для прикладу розберемо найбільш актуальні:

1. Користувач перейде за фішинг-посиланням та перенаправиться на підроблену сторінку університету, яка може виглядати так само, але URL адреса може бути змінена всього на одну букву, і користувач може просто не помітити це. Наприклад якщо реальна сторінка сайту університету це: myuniversity.edurenewal.com, то підробленою можуть бути myunersity.edurenewal.com, myuniversity.edurenewai.com, myuniverslty.edurenewal.com і. т.д.

2. Нічого не підозрюючи, користувач переходить на сторінку відновлення паролю, однак під час перенаправлення шкідливий сценарій активується у фоновому режимі, щоб захопити сеансовий файл cookie користувача. Це призводить до відображеної атаки XSS, яка дає зловмиснику привілейований доступ до університетської мережі.

Напад може мати руйнівні наслідки. Для фізичних осіб це включає несанкціоновані покупки, крадіжку коштів або виявлення крадіжки. Організація, яка піддається такій атаці, зазвичай зазнає серйозних фінансових втрат на додаток до зниження частки ринку, репутації та довіри споживачів. Залежно від масштабу спроба фішингу може перерости в інцидент безпеки, від якого компанії буде важко оговтатися [19, 25]. Для уникнення такого роду атаки, слід дотримуватись наступних рекомандацій:

- встановити захисне програмне забезпечення з останніми оновленнями безпеки для виявлення та блокування;
- захистіть свої облікові записи за допомогою багатофакторної автентифікації. Деякі облікові записи пропонують додатковий захист, вимагаючи двох або більше облікових даних для входу в обліковий запис;
- завдяки багатофакторній автентифікації шахраям буде важче ввійти у

ваші облікові записи, якщо вони отримають ваше ім’я користувача та пароль;

- створення резервних копій;
- відкривайте лише вкладення електронної пошти, надіслані з адрес, яким ви довіряєте. Деякі файли можуть інсталювати вірус на ваш комп’ютер, просто відкривши їх. Так що потрібно бути обережними, щоб не відкривати підозрілі файли, прикріплені до електронних листів. Можна отримати електронний лист із рахунком за щось, що ви ніколи не замовляли, або документ на отримання посилки. Фішери знають, як змусити вас відкрити вкладення;
- не довіряйте електронним листам від знайомих але з незвичайної електронної адреси. Деякі фішингові листи використовують маскування, щоб дістатися до вас. Ви можете побачити логотип вашого банку або улюбленого бренду. Але чи це та сама адреса електронної пошти банк чи компанія зазвичай використовує? Перевірте адресу відправника та порівняйте її з відомими вам адресами.

2 АЛГОРИТМИ З ТЕСТУВАННЯ БЕЗПЕКИ

2.1 Алгоритм тестування SQL-ін’єкцій та XSS атак

Як задокументовано в літературі, запобігання впровадження SQL вразливості, а також атаки XSS в основному досягається за допомогою використання алгоритмів шифрування даних, функцій екранування PHP, алгоритмів зіставлення шаблонів і рандомізації набору інструкцій. Розробники використовували алгоритм хешування SHA-1 щоб запобігти впровадженню пакетного запиту SQL. Це працює за допомогою вилучення значень атрибутів запиту із збережених вхідних даних; їх було хешовано за допомогою алгоритму хешування SHA-1. Після цього будь-який інший вхід також буде хешовано і порівняно з початково хешованими збереженими вхідними даними перед подальшим виконанням. Якщо порівнювані хешовані вхідні дані однакові, запит SQL буде виконано, інакше відхилено. За допомогою цієї техніки помилкові введення не можуть бути оброблені безпосередньо запитом SQL, а спроба отримати збережені дані вхідних значень поверне хешовані значення, які вже зашифровані. Подібним чином, алгоритм зіставлення рядків Боєра-Мура для SQL виявлення та попередження атак ін’єкцій було представлено раніше. У таблиці 2.1 наведено найбільш використовувані спеціальні символи, які використовуються у коді SQL-injection.

Таблиця 2.1 – Спеціальні символи, які використовуються в SQL-ін’єкціях

S/N	Символ	Опис
1	‘	Індикатор рядка символів
2	-- or #	Однорядковий коментар
3	/*…*/	Багаторядковий коментар
4	%	Індикатор атрибута підстановки
5	;	Термінатор запиту
6	+ or ||	Об’єднання рядків
7	=	Оператор присвоювання
8	>, <=, >=, ==, <> or ! =	Оператори порівняння

Подібним чином, техніка запобігання SQL-ін’єкції використання кодів ASCII було запропоновано Шріваставою. Дані введені користувачем було спочатку перетворено на значення ASCII перед збереженням у базі даних. Подальше введення буде перетворено на значення ASCII і зіставлено зі збереженими значеннями ASCII [6]. Якщо буде будь-яка різниця, вхідний SQL-запит буде відхилено. Аналогічно було розібрано метод перевірки та перетворення коду які запропоновані для виявлення SQL-ін’єкції та атак XSS [16]. Однак перетворення коду ASCII споживає ресурси, і техніка не може обробляти закодовану SQL ін'єкцію. Автори в [10] застосували синтаксичний аналіз для виявлення та запобігання ін'єкціям SQL. Кожен вхід SQL-запит було проаналізовано за допомогою граматики, спеціально написаної програми для виявлення SQL. Проте пропонована техніка не була розроблена для обробки інших форм SQL-ін'єкції та XSS-атак

Автори в [13] запропонували методику запобігання SQL-ін'єкцій та XSS-атак. Веб-додатки були розділені на дві категорії: ті, чий запит не змінюється незалежно від часу та параметра класифікувалися як статичні, тоді як ті, чий запит змінюється через час або дані, що передаються в нього, називалися динамічною мережею програми. Були статична і динамічна моделі відображення використовується для виявлення та запобігання вразливостям обох категорій. Подібна техніка з використанням шаблону Ахо–Корасіка Техніка відповідності була запропонована Prabakar et al. Автори в [12] використовували рандомізацію набору інструкцій запобігти ін'єкції SQL другого порядку. Техніка динамічно створює набори інструкцій SQL з довіреного SQL ключові слова.

Вхідний SQL-запит було збережено на проксі-сервері, а не безпосередньо в базі даних. Тому вміст проксі-сервера буде перевірено на будь-які злісні укуси. Однак ця техніка може запобігти лише атаці SQL-ін’єкції на основі логічних значень. Також була подібна техніка з використанням рандомізації набору інструкцій повідомляється в [13–15]. Також у таблиці 2.2 показані команди які використовуються для SQL ін’єкцій

Таблиця 2.2 – Ключові слова які використовуються для SQL-ін’єкцій

№	Ключове слово	Опис
1	OR	Використовується на основі логічних значень
2	UNION	Використовується на основі об'єднання
3	DROP	Використовується для знищення всієї таблиці
4	DELETE	Використовується для видалення рядків у таблиці бази даних
5	TRUNCATE	Використовується для очищення певної таблиці в базі даних
6	SELECT	Використовується для отримання запису з таблиці бази даних
7	UPDATE	Використовується для зміни запису в таблиці бази даних
8	INSERT	Використовується для додавання запису до таблиці бази даних
9	LIKE	Використовується разом із символом підстановки (%) для вибору запису, який містить певний шаблон рядка
10	COMMENT	Використовується під час ін’єкції SQL на основі помилок, щоб змусити сервер бази даних відображати деякі повідомлення про помилки.

Для розробки даного алгоритму, який міг би виявляти та запобігати різним варіантам та формам SQL ін’єкції, було проаналізовано існуючі методики запобігання [27], а також вивчено шаблони для різних видів цієї атаки, і на основі цього запропоновано наступні методи. Методологія, використана в цьому дослідженні, має п’ять етапів: формування шаблону рядка SQL-ін’єкцій, проектування дерева синтаксичного аналізу для різних форм атак, виявлення SQL-ін’єкцій та XSS-атак, запобігання SQL-injection та XSS-атак за допомогою алгоритму KMP та формулювання функцій фільтра.

Формування шаблонів рядків SQL ін'єкції. Кожна форма атак має певні символи та ключові слова(були продеменстровані вище) якими хакери маніпулюють, щоб увічнити свої атаки. Ці символи та ключові слова використовуються для створення шкідливих кодів, які використовуються для виконання різних форм атак. Ідентифікація цих кодів ін'єкцій допоможе придумати, як виявити та запобігти цим нападам. Також у таблиці 3 наведено коди різновиди форм ін'єкційного коду з їх загальними шаблонами.

Таблиця 2.3 – Різні види форм ін'єкцій згідно шаблонів

№	Тип ін'єкції	Шаблон	Приклад
1	Boolean-based	OR '…' = | > | > =| < | < =|<>|! = '…';#	OR '1' = '1';# 123' OR 'a' <> 'b' ;# ' OR '2+3' < = '10' ;#
2	Union-based	`union select … from …;#	' union select * from users; # ' union select name from a;#
3	Error-Based	'…convert (|avg(| round(...	111' convert(int, 'abcd') A' avg('&%$#@*')
4	Batch query	'; drop | delete | insert | truncate | update | select…;#	aaa' ; delete * from users; # ' ; drop table users; #
5	Like-based	'OR … LIKE '…%';#	' OR username LIKE 'S%'#
6	XSS	<script> …'…;</script>	<script>alert('Xss');</script>

Розробка дерева аналізу для різних форм атак. Дерево аналізу використовувалося для представлення синтаксичного шаблону різних форм SQL-ін'єкцій і міжсайтового використання. Нижче представлений сам код дерева аналізу:

start:

input array of characters S (the text to be searched) and array of characters W (the word sought)

'output: array of integers P (postions in at which W is found); an integer nP (number of positions)

define variables:

an integer j ← 0 (the position ofthe current character in S)

an integer k ← 0 (the position ofthe current character in W)

an array of integers T (the table computed elsewhere)

let $nP \leftarrow 0$

while j < length(s) do

if W[k] = S[j] then let j ← j + 1, k ← $k + 1$

If k= length(W) then let PnP} ← j - k, nP ← $Np + 1$

(occurrence found if only first occurrence is needed, may be returned here)

Let K ← $T[k]$ if k < 0 then let J ← j+1, let k ← k+1

Let K ← $T[k]$ (T[length (W)] can't be -1)

Else

Let K ← $T[k]$ if k < 0 let j ← j+1, let k ← k+1

Stop

Union-based або атаки на основі об'єднання мають ординарні лапки, а також ключове слово UNION(об'єднання), а далі вже стандартизований SQL запит по типу: SELECT * FROM products OR ' UNION select title from products.

Для запобігання цього виду атак, було запропоновано наступну функцію:

```
UnionBasedMethod(input: any) {
    injPattern[] = {"`", "UNION", "SELECT", "FROM", "#"}
    for(i=0; i<injPattern.length; i++) {
        if(KMPSearch(input, injPattern[i]) > 0) {
            if((i+1) ==  injPattern.length){
                result=true;
            }
        input=end(slice(injiPattern[i], input, 2));
        }
        Else {
```

```
                result=false; break;
            }
            Return result;
        }
}
```

Error-based або ін’єкція на основі помилок. Поле приймає в себе ординарні лапки, далі функції SQL яких може і взагалі не бути, вказують на наявність SQL-ін’єкцій на помилковій основі. Чергову функцію було розроблено, та підігнано під потрібні характеристики даної моделі атаки:

```
ErrorBasedMethod(input) {
    injPattern[] = {“‘`”,”)”};
    sqlFn[] = {“convert(“,avg(“,”round(,”sum(“,”max(“,”min(“};
    for(I = 0; I < injPattern.length; i++) {
        if(KMP$Search(input, injPattern[i] > 0)$ {
    if(i==0) {
        counter = 0;
        }
        for(j = 0; j< sqlFn.length; j++) {
    if(KMP$Search(input, sqlFn[j] > 0)$ {
        counter ++;
    }
    if(counter == 0 {
        result = false; break; }
        }
    If((i+1) == injPattern.length) {
        result = true
        }
    input = end(slice(injPattern[i], input, 2));
```

```
    }
    else {
        return = false; break; }
    }
    return result;
}
```

Атаки на основі логічних значень або Boolean-based. Згідно з таблицями можна дійти висновку що рядки цієї ін’єкції також мають ординарні лапки за яким іде логічний оператор АБО та істинне твердження, наприклад «1» = «1»; #, «a» <> «b» ; #, «2+3» < = «10». Відповідно для цього рядка, слід спробувати наступну функцію-алгоритм:

```
BooleanBasedMethod(input: any) {
        injPattern[] = {“‘”, “‘”, “‘”, “ = “”‘”, “‘”, “‘”, “#”}
        lOprt[] = {“or”, “||”};
        rOprt[] = {`=`, `>`, `>=`, `<`, `<=`, `<>`, `!=`};
        for(i=0; i<injPattern.length; i++) {
            if(i ==0) {
                counter = 0;
                for(j=0; j<= Oprt.length; j++) {
        if(KMP_Search(input, lOprt[j] > 0 { counter ++; }  ))
            if(conter == 0) {
                return result = false; break;
            }
            If(i == 2) return counter 0;
                }
            for(k=0; k< rOprt.length; k++){
                if(KMP$Search(input, rOprt[k]) > 0$) { counter ++; }
            }
```

```
                If(counter ==0 { result =false; break; })
                If((i+1) == injPattern.length) { result = true }
                input = end(slice(injPattern[i], input, 2)); }
                else { result=false; break; }
                return result; }
        }
}
```

Like-based атака з використанням ключового слова LIKE. Дана атака виконується також з однією лапкою за якою йде логічний оператор OR за який іде один або більше ідентифікаторів та за цим всім ключове слово LIKE. Для запобігання використовується наступна функція:

```
LikeBasedMethod(input) {
injPattern[] = {“`”, “LIKE”, “`”, “%”, “`”, “#”};
lOprt[] = {“OR”, “||”}
for(i=0; i< injPattern.length; i++){
    if(KMP$Search(input, injPattern[i]) > 0$) {
        if(i==0) { counter =0;
    for(j=0; j< lOprt.length; j++)
    {
        If(KMP$Search(input, lOprt[j]) > 0$) { counter ++; }
        if (counter == 0) {result = false; break; }
        if (i+ 1) == injPattern, length){result = true; }
        input = end(slice(injPattern[i], input, 2));
    else { result = false; break;}
    return result;
    }
}
```

Атаки з використанням пакетного впровадження або Batch query. Запит містить ординарні лапки за якими можуть йти наступні ключові слова: INSERT, DELETE, DROP, далі ідентифікатор та в кінці обов'язково крапка з комою та хештег #. Наступна функція шукає необхідний запит та призначається для запобігання такої вразливості:

```
BatchQueryMethod(input) {
    injPattern[] = {“`”, “;”, “,”, “;”, “#”}
    sqlk[] = {"delete", "drop","insert", "truncate", "update", "select", "alter"};
for(i = 0;i < injPattern, length; i + +)
{
    if(KMP$Search(input, injPattern[i] > 0)$ {
        if(i ==0) { counter 0;
            for(j=0; j< sqlk.length; j++) {
        if(KMP$Search(input, sqlk[j] > 0)$) {
            counter ++
        }
        if (counter == 0){result = false; break; }
        if (i+ 1) == injPattern, length) {result = true; }
        input = end(slice(injPattern{i], input, 2));
        }
    else
    {result = false; break;}
    return result; }
```

Cross-Site-Scripting атака. Виявляється коли у вхідному рядку програми зустрічається тег <script>, після якого йде набір наступних команд. Якщо це буде закодована атака XSS, вона матиме відкритий тег JavaScript «<script>», за яким слідуватиме один або більше кодів ASCII, шістнадцяткове число, одинарна лапка. Алгоритм у вигляді звичайної функції виглядає наступним чином:

```
CheckCrossSideMethod(input){
    injPattern[] = {“</script>”, “`”, “</script>”};
    for(i=0; i<injPattern.length; i++) {
        if(KMP_Search(input, injPattern[i] > 0) {
            if((i+1) == injPattern.length) {
                result { false; break;
    }
        }
    }
    return result;}
```

Також для поєднання всіх функцій, слід застосувати алгоритм порівняння рядків KMP(Knuth–Morris–Pratt algorithm), який використовується для порівняння введеного користувачем рядка з різними сформульованими шаблонами SQL-ін’єкції та атак XSS. Алгоритм Кнута-Морріса-Пратта (KMP) — це алгоритм, який використовується для пошуку підрядка (W) у заданому рядку (s), в O(m+n), де m та n це довжини W та S. Сам код даного алгоритму наведено нижче:

$I = \sum_{i=0}^{n} f_i$ – де f це поле користувача з кожної форми у текстовому полі.

```
filter(I) {data = convertASClItoString (I);
    if (data <>""){
        a = UnionBasedMethod (data)
        b = ErrorBasedMethod(data);
        c = BooleaBasedMethod(data);
        d = LikeBasedMethod (data);
        e = BatchQueryMethod(data);
        f = checkXss(data);
if (true(a|||b|c||d|||e||f)) {
    blockUser();
```

```
        resetHTTP();
        warningMessage();
}
else{
        grantAccess();
}
```

Функція filter(). Була розроблена також для запобігання SQL-ін’єкції та атак XSS. Вона містить в собі також вкладені функції, які були написані для запобігання певної форми атак. Тому, щоб виявити та запобігти будь-яким атакам, кожен вхідний рядок буде передано через усі сформульовані функції. Якщо хоча б одна функція повертає True, тоді будуть запущені такі функції: blockUser(), resetHTTP() і warningMessage(). Ці функції використовуються блокування користувача, знищення робочої сесії та повідомлення адміністрації.

2.2 Алгоритм тестування зламаної аутентифікації

Реалізуємо механізми та методи для виявлення та запобігання цієї атаки на обох сторонах: фронтенд частині та бекенд частині, з використанням двох стратегій – мережева та позамережева. Як ми аналізували, кожен підхід у дослідницькій галузі для виявлення викрадення сесії, розробники або запропонували механізм виявлення для певної мережі, яка знаходиться в межах локальної мережі, або для зовнішньої мережі. Але цей метод пропонує підхід, який захистить цілі від атак захоплення сеансу як зсередини мережі, а також поза мережею.

Сеанс – це тривалий зв’язок між користувачем (або користувачем, тобто, браузер) і сервер, зазвичай передбачаючи обмін багатьма запитами та відповідями. Типовий сервер підтримує його, а на сервері є сховище даних або таблиця для зберігання стану користувача та інформації про нього. У табличці кожен рядок містить інформацію про певні сесії. Індекс цієї таблиці відомий як

ідентифікатор сеансу або ключ сеансу. Ключ генерується за першим запитом або після процесу автентифікації та обмінюється між браузером і сервером на кожен запит. Найбільш поширеним метод обміну ідентифікаторами сеансу здійснюється за допомогою файлів cookie HTTP. Викрадення сеансу означає використання дійсного комп'ютерного сеансу, коли зловмисник переймає на себе і повністю контролює його. Це робиться шляхом викрадення ідентифікатора сеансу, який можна використовувати для входу в систему та перегляду особистих даних. Запропонований метод використовує евристичний підхід, для того щоб захистити потенційно вразливі веб сайти та спробувати захистити випадкових користувачів у мережах. Нашу систему можна розділити на дві частини – клієнт системи на стороні та сервері.

Для запобігання атаки hijacking пропонована нами модель складається з двох модулів:

1) зворотний проксі-сервер: RPS відповідає за видачу і перевірку OTC, або ж One-Time-Cookies. Дотримуємося дизайну реверсу проксі-сервера, запропонованого в [14].

2) криптографічний операційний модуль (COM): він відповідає за забезпечення конфіденційності, автентичності та цілісності для OTC. Весь зв'язок між браузером клієнта та внутрішнім сервером маршрутизується через RPS. OTC видається на браузер клієнта за запитом. RPS працює в парі з COM для забезпечення конфіденційності, достовірності та цілісності OTC. Взаємодії RPS з іншими компонентами системи показано ни рисунку 2.1 [28, 29].

Функціональні можливості запропонованого алгоритму наступні:

- колекція IP-адрес та даних з браузера на стороні клієнта;
- генерація ідентифікатора сесії (ID);
- пошук збігів по IP адресам, даних браузера і.т.п.

Для тестування та реалізації RPS використовувалась програма TwistedWeb. Коли надходить запит, генерується OTC і встановлюється в RPS.

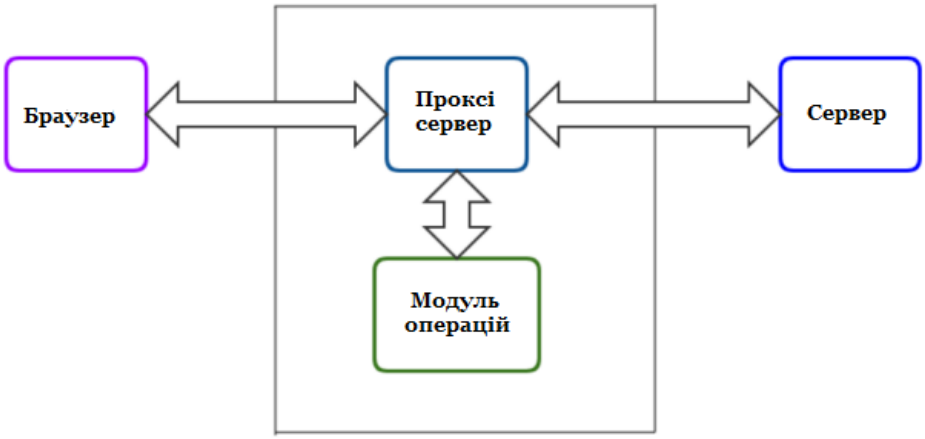

Рисунок 2.1 – Взаємодія RPS з іншими частинами системи

Фрагмент коду реалізації RPS наведено нижче:

```
def render(self, request):
    if self.port == 80:
        host = self.host
    else:
        host = u"%s:%d" % (self.host, self.port)
    request . requestHeaders.setRawHeaders(b"host", [host.encode('ascii')])
    gidd = request.getCookie(b'_gid')
    print (‘gid cookie:', gidd)
    for key in request.received_cookies.keys():
        print ("cookies:",key," ", request.received_cookies.get (key))
    request.content.seek(0, 0)
    qs = urllib_parse.urlparse (request .uri) [4]
    if qs:
        rest = self.path + b'?' + qs
    else:
        rest = self.path
    clientFactory =self.proxyClientFactoryClass(
        request.method, rest, request.clientproto,
        request.getAllHeaders(), request.content.read(), request)
```

self.reactor.connectTCP(self.host, self.port, clientFactory)

return NOT_DONE YET

Коли користувач надсилає свій запит на вхід у систему, Reserve Proxy Server робить перенадсилання цього запиту на сервер. Якщо сервер ідентифікує користувача, тоді сервер згенерує йому одноразову cookie сесію (OTC) та зашифрує конфіденційні дані які складаються з ID(ідентифікатора сеансу) та Expiration Time(термін закінчення дії токену) [9, 15].

Після того як вищезазначені дії та операції будуть застосовані Reverse Proxy Server(RPS) створить цифровий підпис на основі зазначеного та згенерованого вмісту у cookie файлах. Згенерований підпис сервер відправляє назад до користувача у cookie. Тоді браузер перевірить цифровий підпис, дістаючи кукі дані, і у разі валідності перевірка буде завершена успіхом. Тепер після того як автентичність даних було збережено, браузер збереже їх, та буде надсилати на сервер при наступних запитах до нього.

Кожен наступний запит і відповідь між клієнтом і сервером відбувається за цією ж процедурою до завершення сеансу. На рисунку 2.2 показано яка послідовність всіх операцій у запропонованій архітектурі

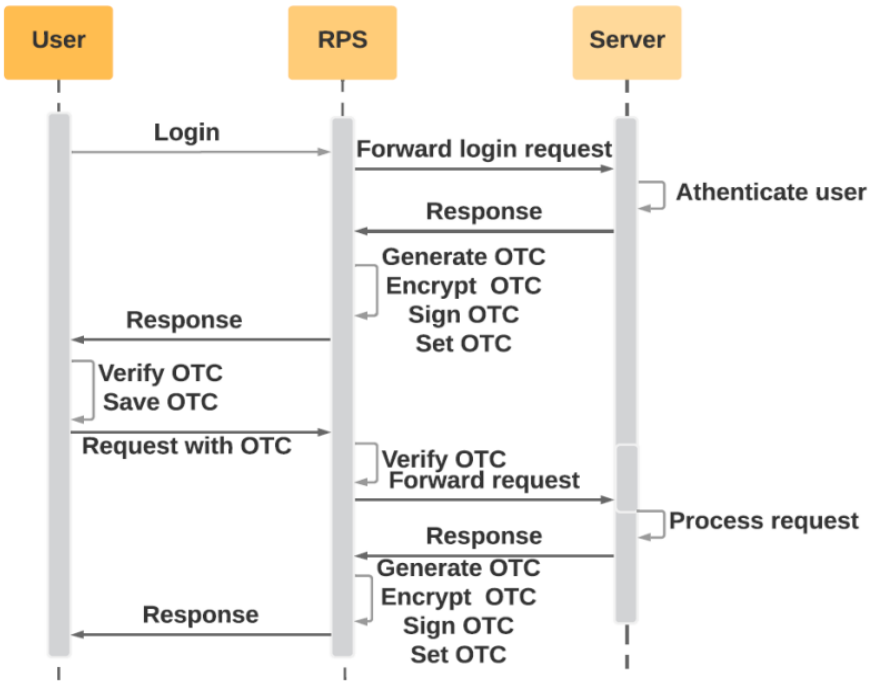

Рисунок 2.2 – Послідовна схема роботи алгоритму OTC

Після того як була складена теоретична частина та спланована схема роботи запропонованого методу було реалізована програмна частина на мові Python. Для початку потрібно згенерувати довгострокового асиметричного ключа, наведений нижче:
Генерування параметру:

```
def get_secret_parameter(p) =
        while True:
        k = random.randrange(1, p - 1)
        if gcd(k, (p - 1)) == 1
                break;
return k;
```

Генерування ключа:

```
def generate_secret_key(data, k):
value = data + str(k)
h = hashlib. sha256(value.encode(‘ascii')) digest ()
return base64.urlsafe_b64encode (h) .decode('ascii')
```

Симетричний ключ використовується в модулі шифрування Python для шифрування конфіденційного вмісту. Його можна описати як Ti = E_{SK} (C_{i2}). Тут E_{SK}() – це функція шифрування. Цифровий підпис (r,s) ОТС можна згенерувати за наступний рівнянням:

$$r = g^k \bmod p.$$

Під час генерації кожного ОТС є секретний параметр k обчислюється так, що k ∈ [1, p − 1] і gcd(k, p − 1) == 1. Нечутливий вміст ОТС і k об’єднані і хешовані за допомогою алгоритму SHA256 для створення симетричного ключа. Цей

процес можна виразити так:

$$SK = h(Ci1||k),$$
$$s = x * (r + h(C_{i1}||T_i) - k\,mod\,(p - 1)$$

де, C_{i1}, T_i, r, s - надсилаються клієнту як частина ОТС. Також для перевірки автентичності поточного клієнта у браузері, можна застосувати наступне рівняння:

$$y^{r+h(Ci1||Ti)} = \mathrm{r} * g^{s} \bmod \mathrm{p}$$

Також у браузері клієнта є наданий ОТС, у наступному запиті. Під час верифікації ОТС, k – вигятується з цифрового підпису, згідно наступного рівняння:

$$\mathrm{k} = \mathrm{x} * (\mathrm{r} + \mathrm{h}(\mathrm{C}_{\mathrm{ш}1}||T_i) - \mathrm{s} \bmod (\mathrm{p} - 1)$$

У коді нижче приведена реалізація вищезгаданих формул для створення цифрового підпису та отримання секретного параметру на мові Python:

```
def CreateDigitalSignature(data, g, k, x,p):
    r = power(g,k,p)
    m = int(hashlib.sha1(data.encode(‘utf-8)).hexdigest(), 16)
    v_1 = x * (r + m)
    v_2 = k % (p - 1)
    s = v_1 – v_2;
    return r, s;
```

Відповідно, функція для отримання секретного параметру:

```
def GetSecretParameter(data, r,s,x,p):
    m = int(hashlib.sha1(data.encode(‘utf-8)).hexdigest(), 16)
    v_1 = x * (r + m)
    v_2 = k % (p - 1)
```

s = v_1 – v_2; return r, s;

Далі реалізовано симетричне шифрування, у якому лише один ключ (секретний ключ) використовується як для шифрування, так і для дешифрування електронних даних. Суб’єкти, які спілкуються за допомогою симетричного шифрування, повинні обмінятися ключем, щоб його можна було використовувати в процесі дешифрування. Цей метод шифрування відрізняється від асиметричного шифрування, де для шифрування та дешифрування повідомлень використовується пара ключів – один відкритий і один приватний. У цьому модулі симетричний ключ використовується для того щоб розшифрувати зашифрований конфіденційний вміст, у нашому випадку це можна описати наступною формулою:

$$\mathrm{C}_{i2}) \; = \; D_{SK}(T_i, \text{ - де } D_{SK} \text{ це і є функція дешифрування}$$

Відповідно до цієї формули, та згаданих вище, можна написати функції для шифрування та дешифрування даних:

```
def SymmetricEncryption(text, key):
    text_bytes = text.encode(‘utf-8’)
    f = Fernet(key)
    cipher_text_bytes = f.encrypt(text_bytes)
    cipher_text = cipher_text_bytes.decode(‘utf-8’)
    return cipher_text

def SymmetricDecryption(cipher_text, key):
    cipher_text_bytes = cipher_text.encode(‘utf-8’)
    f = Fernet(key)
    text_bytes = f.encrypt(text_bytes)
    text = text_bytes.decode(‘utf-8’)
    return text
```

2.3 Алгоритм тестування фішинг атак

Модуль аналізу URL адреси та доменного ім’я. Ми створимо програму, яка включає механізм виявлення фішингу в реальному часі, розміщений на стороні клієнта. Наша програма додаватиме підроблені облікові дані на сторінки входу та порівнюватиме відповіді веб-сайтів, щоб переконатися, що вони справжні.

Оскільки фішингові веб-сайти не мають жодної інформації про облікові дані жертви, під час входу жертви очікується одна з двох речей [18]. Вони або добровільно надають свої облікові дані, або розуміють, що облікові дані, які вони ввели, насправді не є їхніми. Отже, очікуться один з таких сценаріїв:

- фішинговий сайт відображає повідомлення про успіх;
- фішинговий веб-сайт переспрямовує на інший веб-сайт;
- фішинговий сайт відображає повідомлення про помилку;
- фішинговий веб-сайт знову показує ту саму сторінку входу.

Візуальне відображення даної методології зображено на рисунку 2.3.

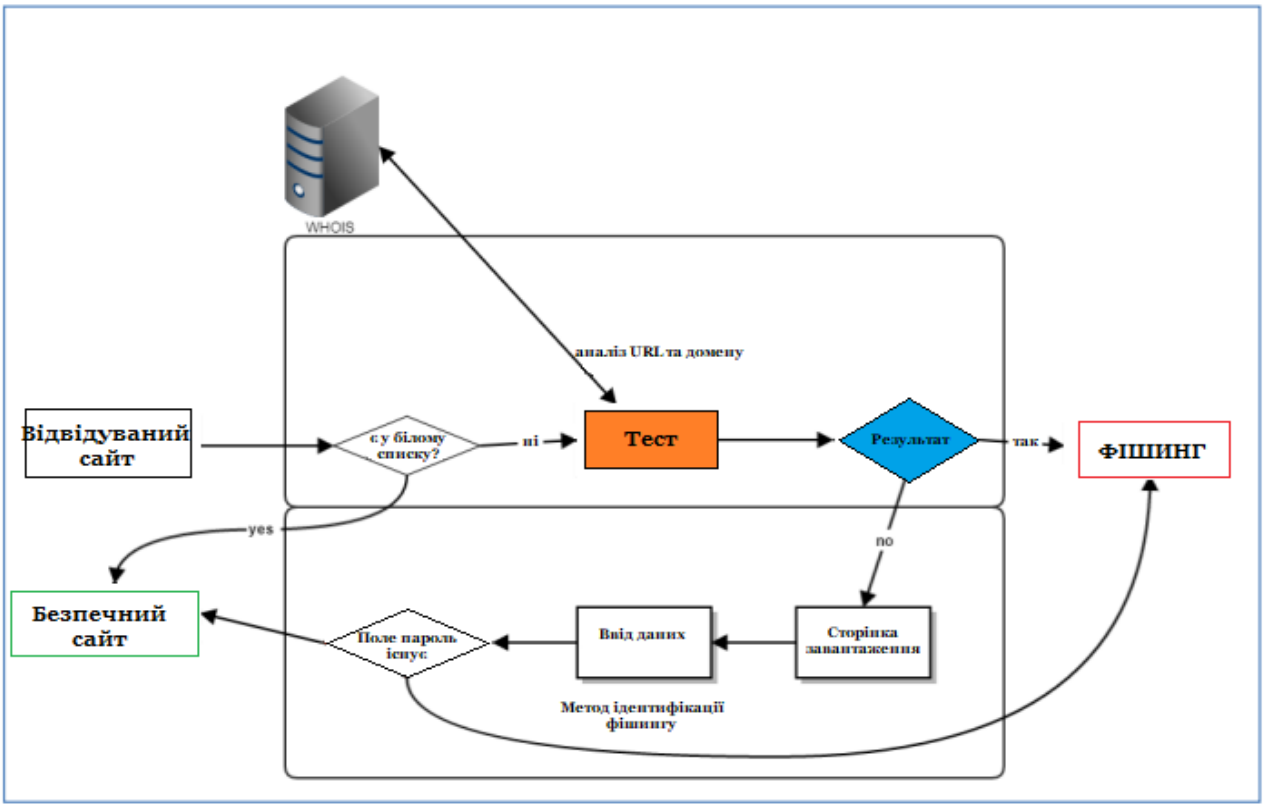

Рисунок 2.3 – Методологія по запобіганню фішинг атак

Отже, згідно рисунку вище, розберемо по кроках необхідні операції:

1. Користувач переходить на вебсайт.
2. Відбувається аналіз URL адреси та домену, перевірка на його наявність у білому листі.
3. Якщо відвідуваний веб-сайт знайдено у білому списку, тоді від позначається як “Легітимний” або “безпечний”.
4. Якщо відвідуваний веб-сайт не знайдено у білому списку, тоді модуль перевірки посилання та домену перевірить наступні властивості:
   a) доменне ім’я було створене меньш ніж 365 днів тому;
   b) закінчення терміну дії домену відбудеться меньш ніж через 180 днів;
   c) сервіс перевірки WHOIS не знайшов такого домену;
   d) у доменному імені перевищено допустимий ліміт використання крапок(більше ніж 5);
   e) у доменному імені присутні спеціальні символи.

1. Кожен атрибут матиме певну вагу, для простоти ми припускаємо, що всі атрибути мають однакове значення ваги 1.
2. Якщо припустити, що всі атрибути присутні, буде обчислено вагу суми, а потім сума буде між 0 і 5 залежно від повернутого атрибута.
3. Сума ваг атрибутів порівнюватиметься із заданим порогом (скажімо, 3), і якщо сума буде більшою за 3, URL-адресу класифікуватимуть як фішингову.
4. Модуль ідентифікації фішингу введе n фальшивих комбінацій паролів електронної пошти.

   a) модуль виявлення фішингу виявляє всі текстові поля, що вводяться, з атрибутом типу адреса електронної пошти або атрибутом імені адреси електронної пошти, користувач, ім'я користувача, ідентифікатор та ідентифікатор користувача;
   b) модуль додає підроблені облікові дані до текстового поля введення та очікує завантаження сторінки:
      i. Якщо сторінка завантажується без поля пароля, це буде враховано.
      ii. Якщо сторінка перенаправляє на інший сайт, це рахується

фішингом

iii. Якщо поле пароля закривається після спроб n, тоді сайт вважається безпечним.

iv. Коли сторінка перезавантажується з текстовим полем пароля, програма вводить нові підроблені облікові дані n разів.

Модуль для аналізу адреси посилання та домену. Мета модуля аналізу URL та доменів - звести до мінімуму кількість помилкових спрацьовувань та скоротити час аналізу. Модуль аналізу URL-адрес і доменів діє як фільтр для явно незаконних URL-адрес перед тестуванням за допомогою модуля виявлення фішингу. Наш фільтр складається з евристики, наведеної в таблиці 2.1 нижче.

Таблиця 2.1 – Евристика для модуля аналізу URL і домену

Евристика	Підозра у фішингу
Дата створення домену	Меньше ніж 365 днів
Закінчення терміну дії	Меньш ніж через 180 днів
Спеціальні символи у адресі	Більш ніж 1
Крапки у домені	Більш ніж 5
Перевірка WHOIS	Не знайдено даних

Існує багато алгоритмів, які можна використовувати для визначення оптимальної ваги для евристик, показаних у таблиці вище, однак для простоти ми використовуватимемо прямолінійну модель, формула якої наведена нижче:

$$C = th_x(\sum we_i \times he_i)$$

де he_i – евристична змінна, варіюється від 0 до 1,

we_i – вага певної евристики, ціле число,

th_x – порогова функція, x – значення, логічний тип,

C– функція класифікатора, логічне значення.

Наступним кроком є обчислення ваги для кожної евристики. Фактично, що ефективніша евристика, то більшу вагу вона має. Евристика з більшими вагами має бути більш точною і мати менше неправдивих спрацьовувань при виявленні фішингових сайтів. Щоб виміряти ефективність евристики, розраховується різниця між істинними та хибними позитивними результатами. Це простий підхід, аналогічний тому, що використовувався в іншому звіті про панелі інструментів для захисту від фішингу [22]. Розрахуйте кожну вагу пропорційно.

Формула має вигляд:

$$we_i = round(\frac{TRP_i - FPR_i}{10})$$

Ми використовували рівняння 1, 2 для визначення оптимальних ваг кожної евристики. Ми використовували 100 фішингових URL-адрес, вибраних з PhishTank, і 100 законних сторінок з бази даних Alexa Top 500. 200 URL-адрес були сайтами різними мовами для забезпечення узгодженості. Використовувався поріг. Для порогу x = 8 була досягнуто найкраща точність 96,66%.

Модуль аналізу URL використовує евристику для перевірки URL-адреси. Модуль аналізує вказану URL-адресу та витягує домен. База даних WHOIS запитується з відкликаним доменом для дати створення та дати закінчення терміну дії домену. Кожне евристичне правило перевіряється та розраховується загальна вага. Класифікатор порівнює суму з порогом x = 8. Якщо x > 8, C = 1, то сторінка класифікується як фішингова, і якщо x > 8, C = 0, сторінка класифікується як легітимна.

Модуль ідентифікації та тестування фішингу. Веб-сторінки повністю скануються та класифікуються модулем виявлення фішингу. Модуль виявлення фішингу перевіряє сторінку входу, багаторазово вводячи підроблені облікові дані в полі входу, і на основі відповіді класифікує сторінку як фішингову чи

легітимну. Сам модуль виявлення фішингу написано на Python і він використовує веб-драйвер Selenium. Selenium використовується для імітації звичайних дій користувача (натискання кнопок, вставлення тексту, надсилання форм, перетягування). Крім того, Selenium використовує безліч браузерів (Firefox, Chrome) для відображення веб-сторінок [16, 17]. Це корисно, тому що тести виконуються на сторінці, що повністю відображається. Модуль використовує Selenium для перевірки цієї URL-адреси. Однак, перш ніж Selenium розпочне тестування веб-сторінки, він перевіряє білий список, що зберігається в базі даних, щоб дізнатися, чи є домен у білому списку. Якщо база даних не містить запису для поточного домену, продовжується перевірка URL-адреси. Наприклад, у коді наведеному нижче міститься два поля елемента введення HTML з типами атрибутів електронної пошти та пароля, які ми шукаємо на сторінці:

<input name=”email” type=”text” type=”email” >
<input name=”password” type=”text” type=”email” >

Якщо поля для імені вхідного тега відсутнє, сторінка не буде розглядатися для цього тесту, а URL-адреса буде зареєстрована в базі даних. З іншого боку, якщо ім'я вхідного тега завершується, програма фільтрує всі імена вхідних тегів з типами атрибутів або іменами, що містять електронну пошту, користувача, ім'я користувача, ідентифікатор користувача і пароль використовуючи їхні так звані XPath лінки [26].

Якщо поля пароля немає або взагалі немає поля введення, поточна сторінка вважається сторінкою без форми входу і не підлягає цій перевірці. Використовуючи жорстко запрограмований список визначених адрес електронної пошти та паролів, програма перевіряє URL-адреси, використовуючи наступний список підроблених адрес електронної пошти та звичайних випадкових паролів:

test_emails = [‘first.last@name.com’, ‘test@test.com’, ‘python@great.com’]

Також для кращого розуміння, на рисунку 2.4 представлено схему роботи модуля по ідентифікації фішингових атак.

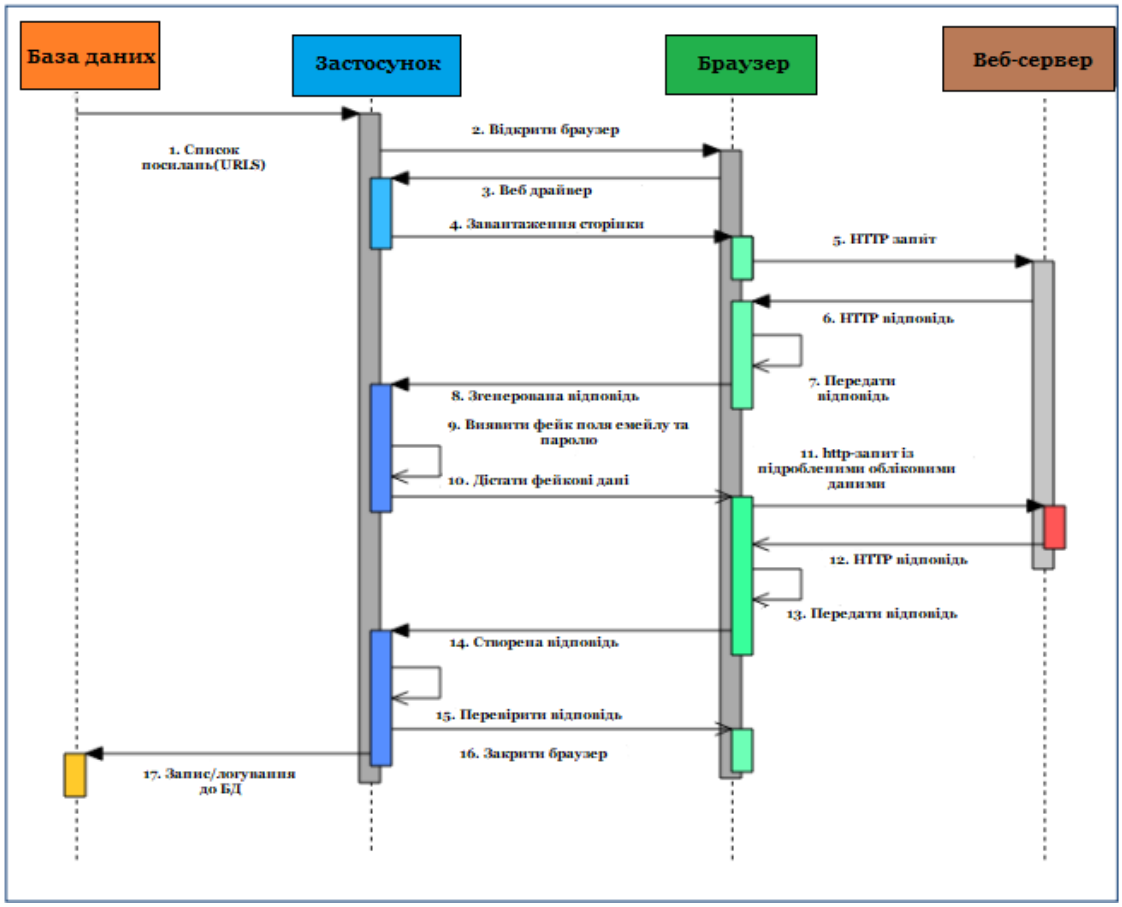

Рисунок 2.4 – Схема роботи модуля по ідентифікації фішингових атак

Отже, у даному розділі було представлено схеми тестування та ідентифікації найбільш актуальних проблем та вразливостей у сфері вебу. Було запропоновано методи для виявлення та тестування веб-ресурсів на проникність від XSS атак, SQL ін’єкцій, а також механізми та методи для виявлення та запобігання зламу аутентифікації. Це було організовано на фронтенд та бекенд частині, з використанням двох стратегій – мережева та позамережева. Для виявлення фішингових сайтів було використано існуючий модуль аналізу URL адреси та доменного ім’я, з подальшим використанням його у запропонованому модулі для ідентифікації потенційно загрозливих фішингових сайтів.

3 ТЕСТУВАННЯ ТА АНАЛІЗ АЛГОРИТМІВ З БЕЗПЕКИ

3.1 Тестування ресурсу та аналіз на вразливості Cross-Site-Scripting та SQL ін’єкцій

Запропонована методика була реалізована за допомогою мови програмування PHP та Apache XAMPP Server. PHP було обрано як мову програмування, оскільки це одна з найбільш широко використовуваних мов програмування для створення веб-додатків, керованих базами даних, тоді як сервер Apache XAMPP було обрано через його кросплатформенність та може працювати з різними операційними системами.

Середовище тестування. Веб сервер Apache та інформаційний сервер(ISS) були використані для того щоб розмістити систему під час тестування методики. Тест проводився як на віддаленому сервері, так і на локальному сервері WAMP/XAMPP. Щоб перевірити запропонований метод, спеціально було розроблено вразливий веб-додаток, показаний на рисунку 3.1

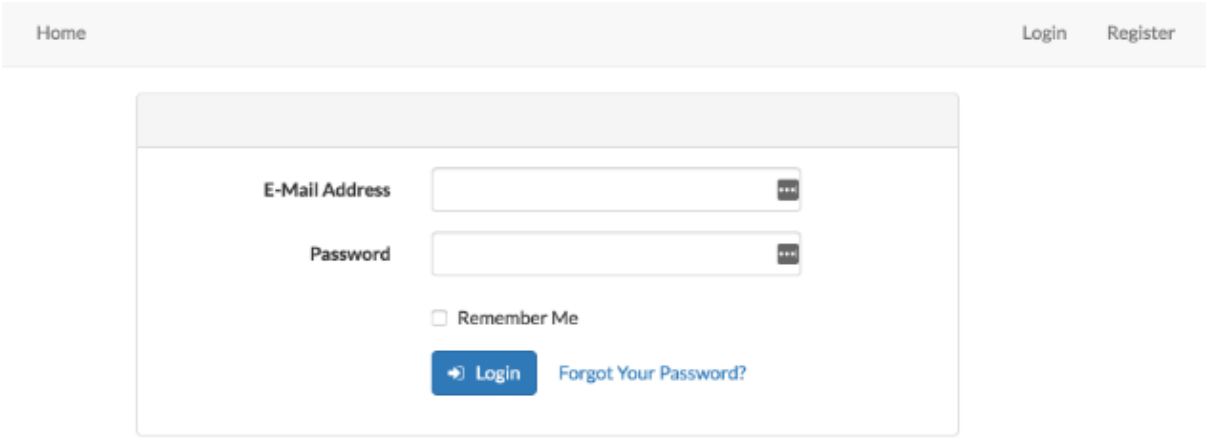

Рисунок 3.1 – Вікно входу у вразливий додаток

Зроблені різні спроби подати різні відомі SQL шаблони ін’єкцій і XSS-атаки з використанням різних типів вразливого коду, які показано в таблиці 3.1. Тестування складається з тестових випадків який містить рядок введення вразливого коду.

Таблиця 3.1 – Список проведених тестів у веб-додатку

№	Boolean-based SQLi	’ OR “ = “; #
1	Like-based SQLi	a‘ OR username LIKE ‘S%’;#
2	Union-based SQLi	UNION select cardNo, pin from customer; #
3	Error-based SQLi	‘ round((select username from users), 3)
4	Batch query SQLi	‘ ; insert into users values (‘Bala’, ‘1234’);#
5	Batch query SQL Injection	; update table users set username = ‘Bala’, password =’123’ ; #
6	Encoded cross-site scripting	<script> alert(" XSS ") </script>
7	Encoded SQL injection	& # x39 & # x85 & # x78 & # x73 & # x79 & # x78 & # x32 & # x83 & # x69 & # x76 & # x69 & # x67 & # x84 & # x32 & # x 42 & # x32 & # x70 & # x82 & # x79 & # x77 & # x32 & # x117 & # x115 & # x101 & # x114 & # x115 & # x45 & # x45
8	Cross-site scripting	<script> alert(‘XSS‘) </script>
9	Cross-site scripting	<script>myFunction( );</script>

Рядки введення були надані через поля для входу у аккаунт, у вразливому додатку. У разі виявлення рядків коду з таблиці вище, її виконання блокувалося, а MAC-адреса систем, що використовуються для здійснення атаки, типи атак, надані вхідні рядки, позначка часу та статус злому були задокументовані в таблиці бази даних, показаній на рисунку 3.2.

Ґрунтуючись на результатах, отриманих від різних спроб атак, запропонована техніка змогла успішно повністю виявити та запобігти всім атакам.

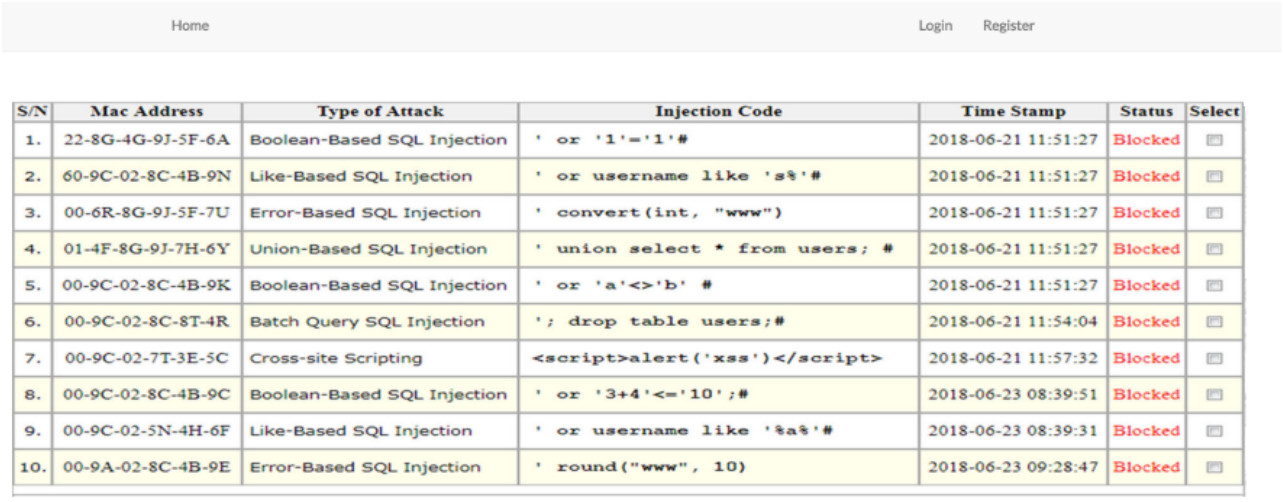

Рисунок 3.2 – Вивід результатів з бази даних

Результати, отримані за допомогою запропонованої методики, порівнювали з наявними в існуючій літературі. Сім існуючих літературних джерел, які використовували алгоритми зіставлення шаблонів, шість літературних джерел, які використовували алгоритми шифрування даних, дві літературні джерела, які використовували рандомізація набору інструкцій і одна література які використовували функції екранування PHP, використовувалися для порівняння.

Було створено новий підхід до виявлення та запобігання SQL-ін’єкцій і атак XSS. Спочатку були вивчені різні типи та шаблони атак, а потім було розроблено дерево синтаксичного аналізу для представлення шаблонів. На основі ідентифікованих шаблонів була сформульована функція filter() яка виявляє та запобігає різних форм SQL-ін’єкцій та XSS атак.

3.2 Тестування ресурсу та аналіз на вразливості Broken Authentification

Запропоновані алгоритми дозволяють забезпечити конфіденційність, автентичність та цілісність, тому було проведено аналіз системи безпеки. Перш за все, це забезпечення конфіденційності, адже саме заходи конфіденційності

захищають інформацію від несанкціонованого доступу та зловживання. На сьогодні багато інформаційних систем зберігають інформацію, яка має певний ступінь конфіденційності. Це може бути конфіденційна бізнес-інформація, яку конкуренти можуть використати у своїх інтересах, або особиста інформація про працівників, клієнтів або клієнтів організації.

Конфіденційна частина одноразових файлів cookie, яка містить ідентифікатор сеансу та термін придатності шифрується секретний ключем не зберігається на зворотньому проксі сервері і не передається клієнту по мережі. Отже, зловмисник не зможе отримати доступ до даних по лінії передачі.

На рисунку 3.3 показано поведінку операцій криптографії під час використання одноразових файлів cookie. На даній лінійній діаграмі, кількість запитів яка розміщена на осі X, і час витрачений на виконання відповідних операцій, розміщені на осі Y. Кожна окрема операція представлена лініями різних кольорів. Також на даному рисунку можна побачити що потрібно дуже мало часу для вибору секретного параметру та генерації секретного ключа.

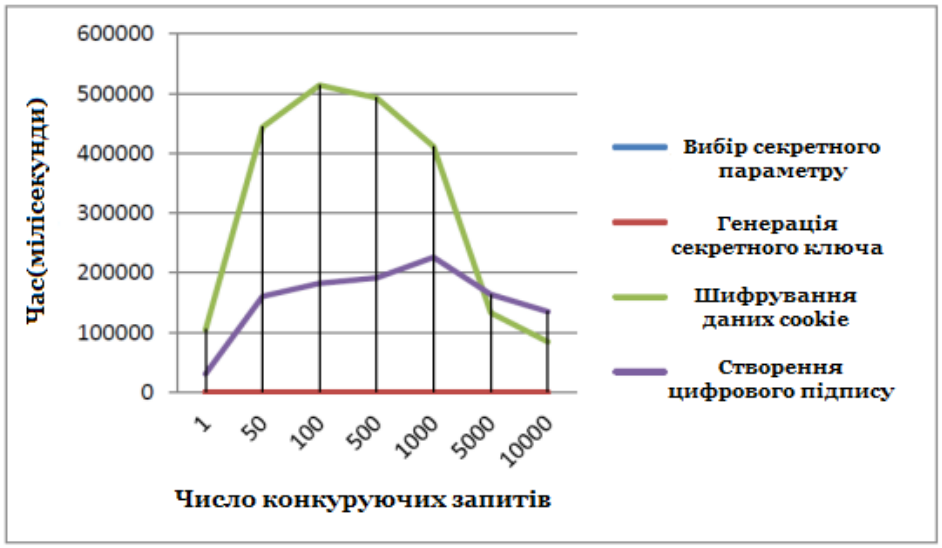

Рисунок 3.3 – Діаграма поведінки проведених операцій

Ці дві операції виконуються майже одночасно і швидко. З іншого боку, лінії для шифрування даних файлів cookie та створення цифрового підпису мають дещо параболічні форми. Час, необхідний для шифрування файлів cookie

та створення цифрового підпису, спочатку збільшується разом із кількістю запитів, а після отримання певного значення зменшується, незважаючи на збільшення кількості запитів.

Також на рисунку 3.4 наведено лінійну діаграму, яка показує час, необхідний для різних криптографічних операцій під час використання одноразових cookie файлів у фазі верифікації

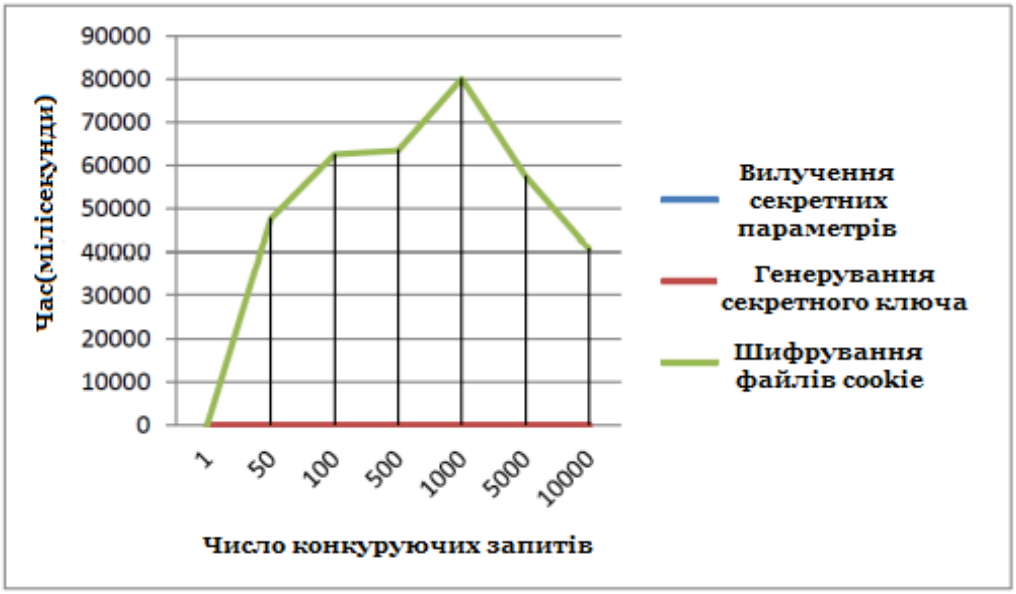

Рисунок 3.4 – Діаграма поведінки проведених операцій на етапі верифікації

У нас є різноманітна кількість одночасних запитів і показаний час, витрачений на відповідні операції. Згідно вищенаведеної діаграми, можна побачити, що вилучення секретного параметра та генерація секретного ключа вимагає дуже мало часу, і цей час є постійним незалежно від кількості одночасних запитів. З іншого боку, лінія для розшифровки файлів cookie має майже параболічну форму. Час, необхідний для шифрування даних файлів cookie спочатку зростає зі збільшенням числа одночасних запитів і після отримання певного значення він зменшується, незважаючи на збільшення кількості запитів.

Отже для забезпечення конфіденційності, автентичності та цілісності файлів cookie та запобігання викраденню сеансу через повторне відтворення та hijacking атаки було запропоноване та протестоване нове рішення, яке включає в себе провадження зворотного проксі серверу та перевірки одноразових файлів

cookie(OTC) та використано спеціальний модуль криптографічних операцій для керування зашифрованими одноразовими файлами cookie. Виконаний аналіз безпеки для перевірки запропонованого методу.

3.3 Тестування ресурсу та аналіз на вразливості фішинг-атак

Веб-скрапінг фішингових сайтів. Для збору даних для тестування використовувалися два різні сервіси веб-скрапінгу. Парсер на основі Ruby для збору фішингових сторінок реалізований за допомогою модуля ruby Marlonso5 Phishtank. Scrapper отримує та витягує дані з перших 50 сторінок Phishtank.

Phishtank відображає дані в таблиці HTML, як показано на рисунку 3.5. Парсер видаляє дані тільки в тому випадку, якщо останній осередок у рядку містить «ONLINE» як вміст елемента HTML.

Парсер витягує всі комірки (id, url, created_at, submit_by і т.д.) та експортує їх у формат JSON. Це використовуватиметься пізніше під час тестування.

Загалом, всі отримані дані будуть мати наступний вигляд:

[

{

"id":"3455967",

"URL":"http://cafeim.co.kr/wp/caixa.gov.br/pages/inter/index.php",

"created_at":"added at on September 7th 2022 5:52 PM",

"submitter":"test33",

"valid":"",

"online":"ONLINE"

},

{…}

]

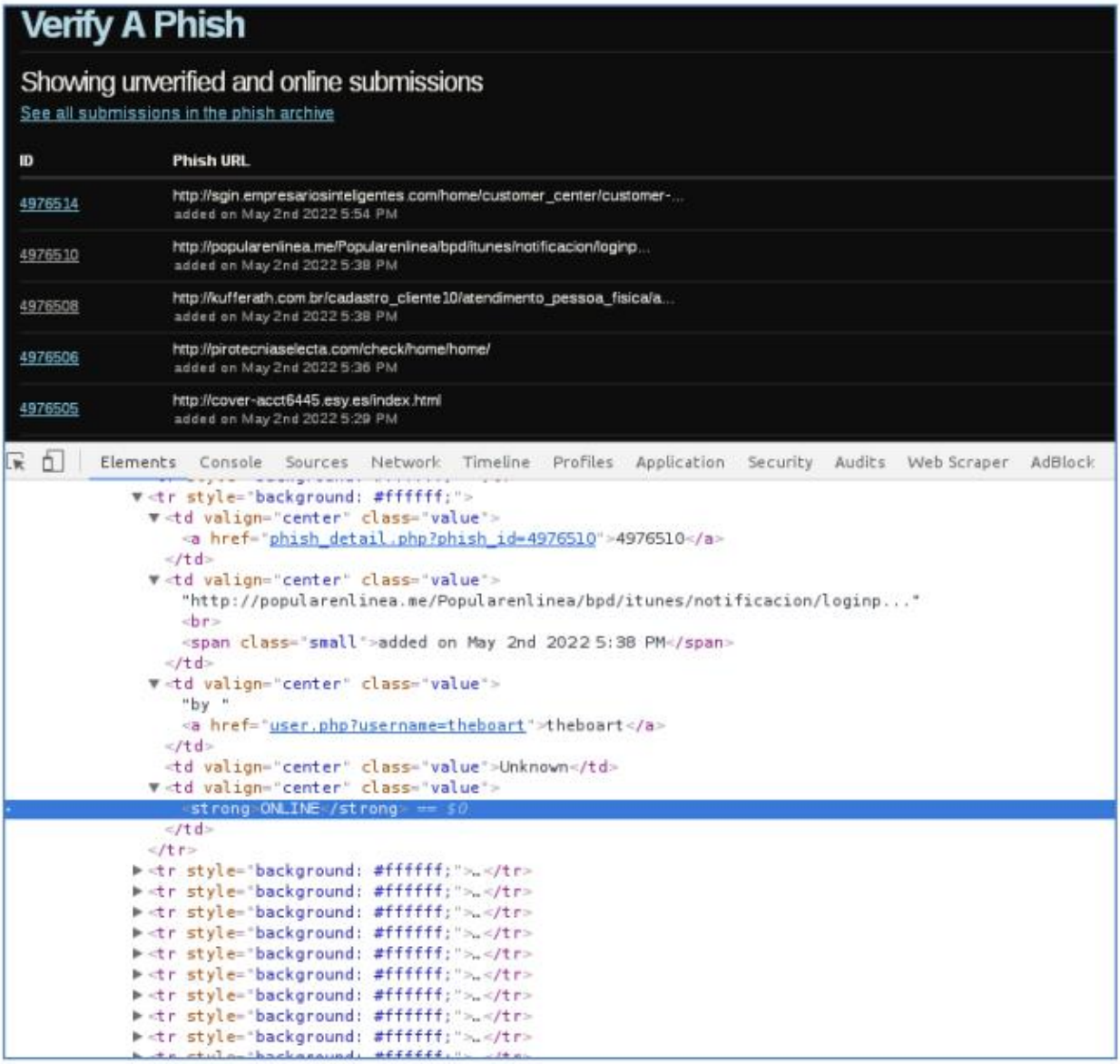

Рисунок 3.5 – Результати скрапінгу PhishTank

Парсер видаляє сайти Phishtank, але нас цікавлять лише фішингові сторінки із формами входу. Це фільтрується запропонованим модулем виявлення фішингу.

Веб-скрапер легітимних сайтів. Скарпер легітимних сайтів базується на двох різних скраперах другого порядку. Перший використовується для заповнення бази даних Alexa з 500 найкращих доменів. Цей скрапер заснований на бібліотеці Python6 компанії VivekPatani і доданий до списку доменів. Потім

другий парсер отримає список різних сайтів використовуючи Google, та шукає доменні ім’я адрес яка містять в собі також таку частину як “/login”(Сторінка авторизації). Ключове слово «логін» та доменне ім'я. Результати експортуються у файл JSON, який має приблизно наступний вигляд:

"data": [
{
"domain": "myaccount.payoneer.com",
"id": "432",
"link": "https://myaccount.payoneer.com/",
"link_type": "results",
"rank": "1",
"serp_id": "455",
"snippet": "Payoneer Inc.",
"title": "Payoneer Inc.",
"visible_link": "https://myaccount.payoneer.com"
}
]

Отримані дані також використовувались у наборі даних які були представлені у таблиці 3.2. На рисунку 3.6 представлено схему роботи двох використовуваних компонентів, а саме Google Scrapper та запропонованих модуль ідентифікації фішингових атак:

Загальна кількість URL-адрес, видалених з бази даних Alexa, становить 412 і 940 з Phishtank. Однак, як показує збір даних, лише 532 сторінок були доступні в Інтернеті, а інші були відсутні або були недоступні. Оскільки фішингові сторінки мають дуже обмежений час онлайну, з Phishtank було видалено більше фішингових сторінок та відфільтровано лише сторінки з формами входу, тобто з вставками “/login”, в результаті чого було отримано 232 результатів з Phishtank.

Крім того, були відфільтровані URL-адреси з набору даних Alexa скрапера зі сторінками входу користувача(“/login”), в результаті отримано 298 сторінок.

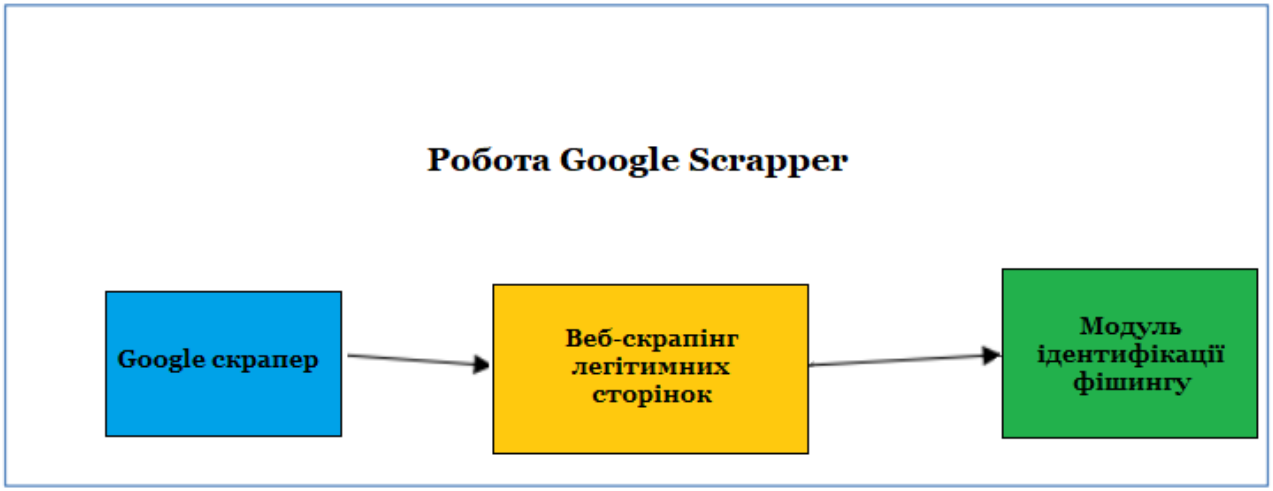

Рисунок 3.6 – Схема роботи Google скрапера

Набір даних , показаний у таблиці 3.4, використовувався для визначення евристичних ваг для модуля аналізу URL і домену. Це перші 100 сторінок фішингу та перші 100 законних сторінок з набору даних таблиці 3.3. Набір даних з таблиці 3.5 використовує ті ж 298 легітимних сторінок, що й набір даних з таблиці 3.3, нещодавно отримані URL-адреси з Phishtank та відфільтровані для сторінок входу, в результаті чого виходить 431 URL-адреса, як показано в таблиці 3.5.

Таблиця 3.2 – Набір даних станом на 12/09/22

База даних	Число посилань (URL)	Фішинг/безпечний
Phishtank	532	Фішинг
Alexa	412	Легітимні

Таблиця 3.3 – Набір даних станом на 23/09/22

База даних	Число посилань (URL)	Фішинг/безпечний
Phishtank	232	Фішинг
Alexa	298	Легітимні

Таблиця 3.4 – Набір даних станом на 05/10/22

База даних	Число посилань (URL)	Фішинг/безпечний
Phishtank	100	Фішинг
Alexa	100	Легітимні

Таблиця 3.5 – Набір даних станом на 24/10/22

База даних	Число посилань (URL)	Фішинг/безпечний
Phishtank	431	Фішинг
Alexa	298	Легітимні

Весь проект був розроблений на платформі Debian (середовище розробки), розгорнутий та протестований у віртуальному середовищі Ubuntu. Віртуальні середовища Ubuntu базуються на віртуальних знімках екрана або станах. Це означає, що при кожному запуску віртуального середовища використовуватимуться лише встановлені бібліотеки та інструменти. код останнього запуску видаляється. Як середовище розробки було обрано операційну систему Debian10 GNU/Linux версії «10.2». Причина вибору Debian Linux полягає в тому, що це надійна, стабільна, надійна та одна з найпопулярніших серверних операційних систем. Розробники Debian також стверджують, що Debian має найкращу систему пакування у світі. Крім того, система дуже ефективно використовує оперативну пам'ять, особливо, порівняно з іншими дистрибутивами Linux. Процесор Pentium – це мінімум, необхідний для належної роботи даної операційної системи.

Debian також підтримує всі пакети та модулі Python, необхідні для розробки інструментів.Усі необхідні пакети можна встановити у спеціальному менеджері програм.

Для розробки модуля була обрана мова програмування Python з інтерпретатором версії 3.5. У професійному плані Python чудово підходить для веб-розробки на стороні сервера, штучного інтелекту, аналізу даних, моделювання даних і наукових обчислень. Оскільки так багато програмістів

використовують Python для створення продуктивних програм, ігор і інструментів для робочого столу, є багато ресурсів і бібліотек, які допоможуть вам почати роботу. Крім того, Python є кросплатформним і працює на всіх популярних платформах. Крім того, майже в усіх дистрибутивах Linux за замовчуванням встановлено Python, що спрощує розгортання інструменту. З основних плюсів Python є:

- інтерпретація. Інтерпретатор обробляє вихідний файл під час виконання, зчитуючи його рядок за рядком і виконуючи його вміст. Подібно до Perl і PHP, Python не вимагає від вас компілювати програму перед її запуском. Тому немає необхідності викликати компілятор. Замість запуску компілятора, який допомагає перетворити вихідні файли на скомпільовані файли класів, просто запустіть файл .py. Компіляція байт-коду Python є автоматичною та повністю неявною;
- ООП. Об’єктно-орієнтоване — ця парадигма програмування пропонує загальну орієнтацію сценарію та потужну структуру коду. Цей об’єктно-орієнтований підхід дозволяє нам думати про проблеми в термінах класів та об’єктів. Потім об’єкти об’єднуються для створення складних комп’ютерних програм. Окрім об’єктно-орієнтованого програмування, Python також підтримує процедурну парадигму. Вибір підходу до об’єктно-орієнтованого програмування може зробити програмування на Python більш просунутим, оскільки ООП є лише одним із варіантів;
- потужний набір інструментів. По суті, Python є текстовим файлом, що містить інструкції для інтерпретатора, написані в текстовому редакторі або IDE. IDE є повнофункціональною і пропонує вбудовані інструменти, такі як засоби перевірки синтаксису, відладчики та браузери коду. Текстові редактори зазвичай не включають функції IDE, але їх можна настроювати. Python також має величезний набір сторонніх пакетів, бібліотек та фреймворків, які спрощують процес розробки. Таким чином, ці оптимізації роблять Python ідеальним для великих проектів.

База даних, яка використовується для зберігання URL-адрес із білого

списку, — MongoDB10 версії 3.4.3. MongoDB - це документно-орієнтована база даних, яка зберігає дані в колекціях, створених з окремих документів, а не в таблицях, побудованих з окремих рядків, як це роблять реляційні бази даних.

Системне використання. Користувачеві буде запропоновано, як показано на рисунку 3.7 ввести «l», щоб перевірити наявність легітимних сторінок. Якщо користувач вводить "p", програма перевіряє наявність фішингових сторінок.

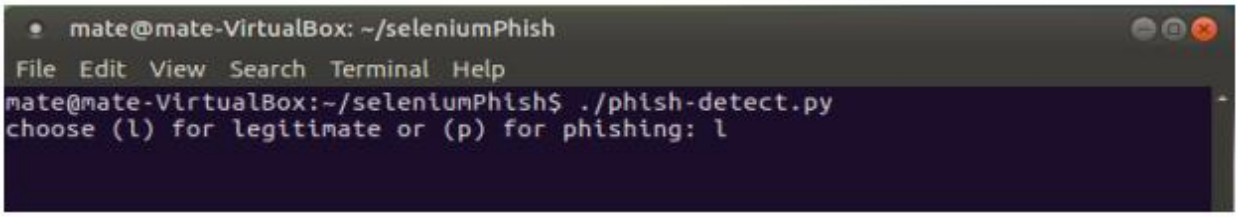

Рисунок 3.7 – Головне вікно програми по ідентифікації фішингу

Після однієї з цих операцій, модуль перевірки фішингових посилань дістане посилання та почне перевірку, процес перевірки зображено на рисунку 3.8.

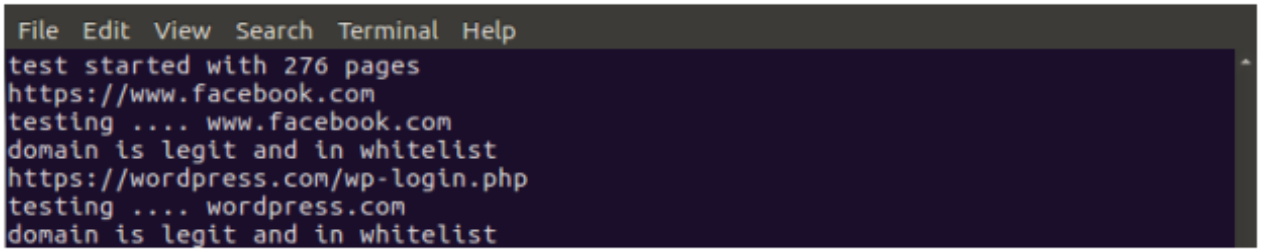

Рисунок 3.8 – Процес перевірки посилань

Браузер Google Chrome знайде поля адреси електронної пошти та паролю і введе їх у відповідні поля. Потім завдяки Selenium, буде змодельовано натискання на клавішу Enter, і дані введені дані відправляться на сервер. Процес зображено на рисунку 3.9

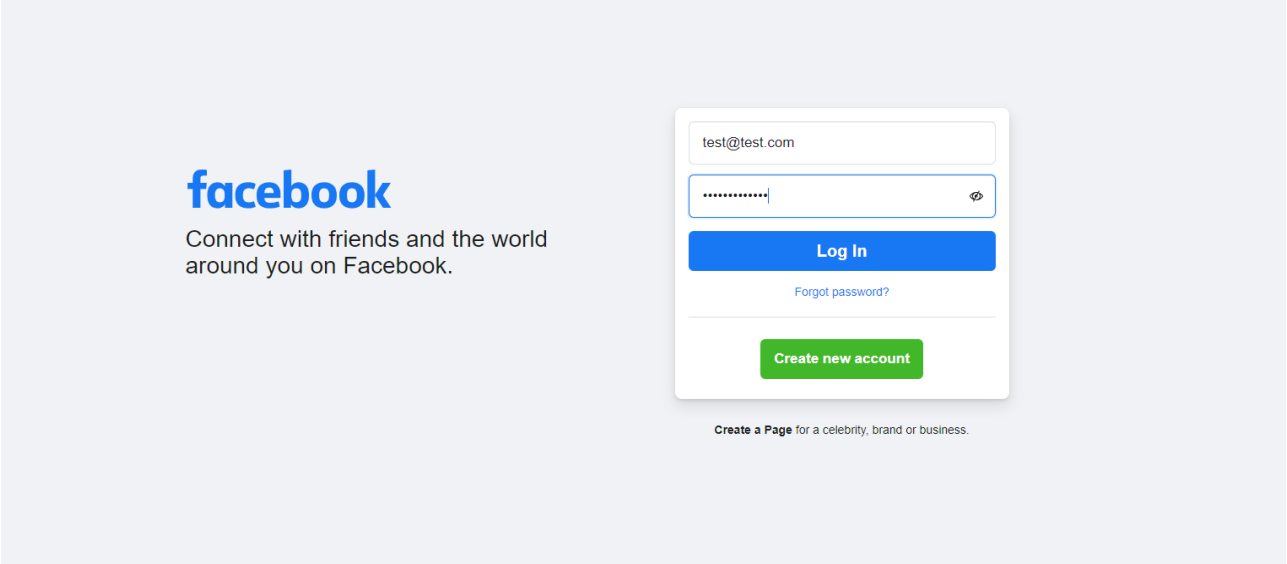

Рисунок 3.9 – Сторінка входу Facebook, автоматично заповнена Selenium модулем

Веб-драйвер продовжує вводити адресу електронної пошти та пароль та надсилати форму n разів. Потім перевіряється наявність символів у полі для паролю (див. рис. 3.10). Якщо пароль ще існує, веб-сторінка вважається легітимною. В іншому випадку він класифікується як фішинговий.

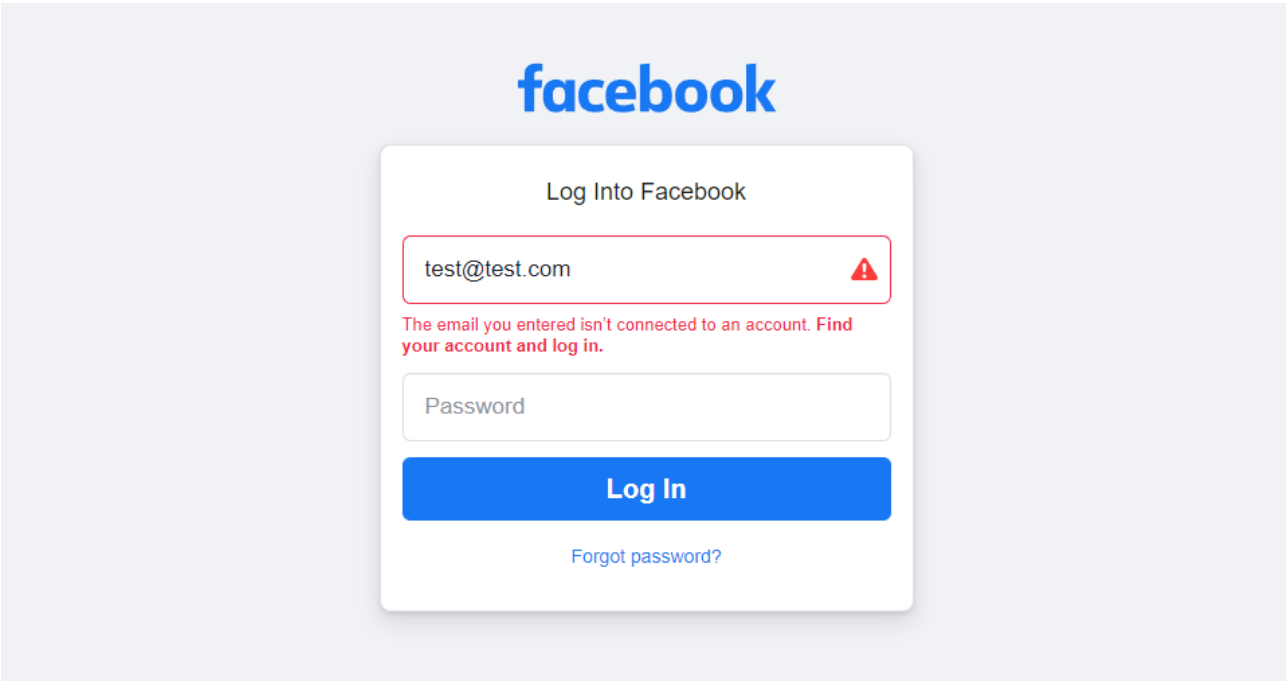

Рисунок 3.10 – Відповідь сторінки після відправки форми

Результати перевірки на фішинг відображаються на вихідному терміналі,

як показано на рисунку 3.11

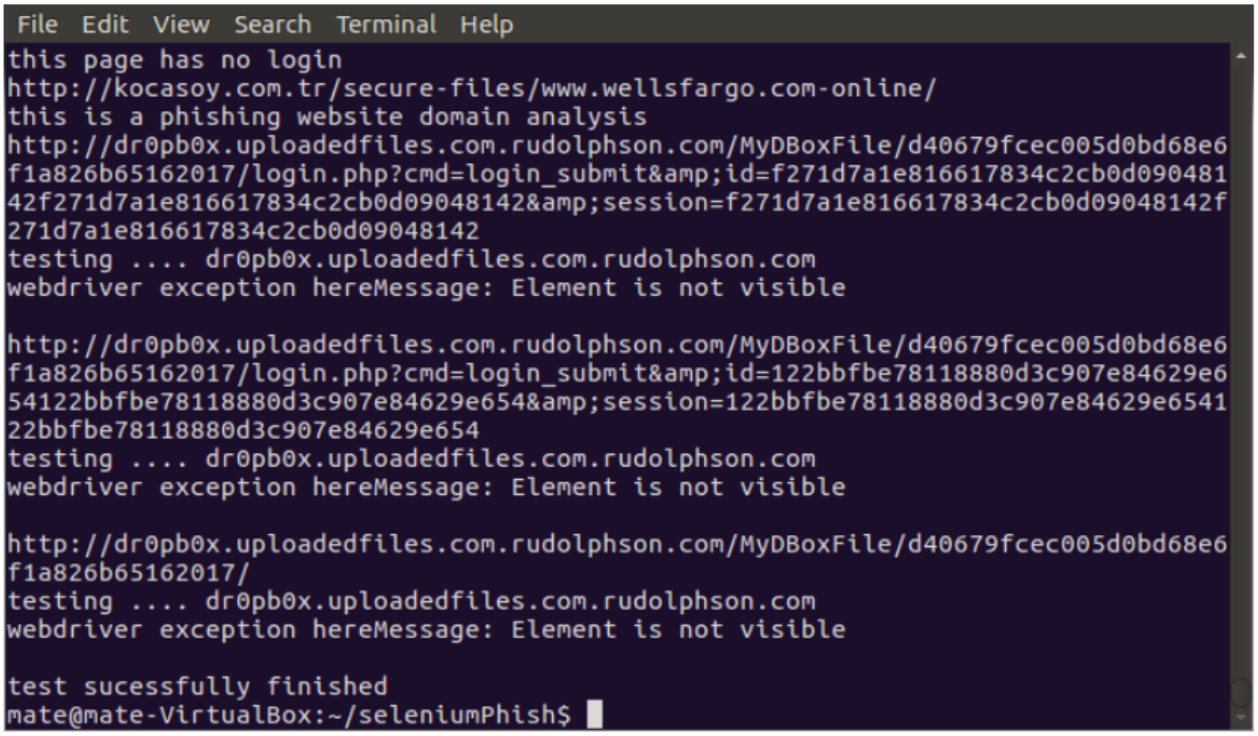

Рисунок 3.11 – Результати тестування

Ми використали Grafana для візуалізації даних у реальному часі. Grafana — це графічний інструмент, інтегрований з базою даних журналів часових рядів входу та виходу, для відображення результатів у реальному часі, результати по продуктивності наведено на рисунку 3.12.

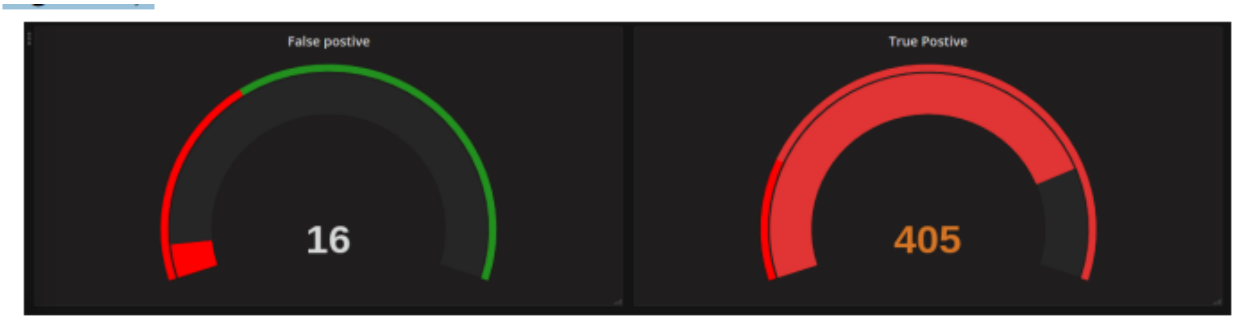

Рисунок 3.12 – Результати продуктивності програми

У таблиці 3.6 показано результати, використані для розрахунку евристичних ваг парсерів URL-адрес і доменів залежно від їх продуктивності. За результатами тестування найбільш ефективною евристикою виявилася дата

створення домену, з 85% позитивними результатами. Точки у адресі URL – це ефективна евристика, яка виявляє 44% фішингових веб-сторінок.

Таблиця 3.6 – Результати ваги евристик

Евристика	Позитивні	Негативні	Ефект	Вага
Дата створення домену	85%	32%	53	5
Термін придатності домену	23%	5%	18	2
Спеціальні символи у URL	10%	0%	10	1
Крапки у URL	48%	3%	45	1
Наявність у WHOIS	10%	7%	3	0

Ми протестували систему 5 разів зі 100 фішинговими URL-адресами та 100 законними URL-адресами (3), показаними в таблиці 3.4.

Межі для вибору порогових значень не були обрані випадково. Дата створення домену має евристичну вагу 5, і немає сенсу мати порогове значення «x» 5, оскільки класифікація базуватиметься лише на цій евристиці. Таким чином, мінімальний поріг для цього тесту дорівнює 6, що є максимальною вагою. Максимальне значення x для цього тесту дорівнює 10. Однак, коли x більше 8, справжній позитивний показник падає, оскільки аналізатори URL та доменів класифікують законні URL-адреси.

Відповідний відсоток справжніх позитивних результатів становить 93%, як і за використання лише модуля ідентифікації фішингу. Тому вищі тести з вищими порогами не розглядалися.

У таблиці 3.7 нижче показані результати тестування модуля виявлення фішингу тільки на наборі даних показаних у таблиці 2. Істинний негативний показник становить 94,19%, що дуже багато, як і очікувалося. Це пов'язано з тим, що на безпечних та легітимних веб-сайтах завжди має відображатися поле

пароля під час входу до системи якщо форму було надіслано з неправильними обліковими даними.

Таблиця 3.7 – Результати роботи модуля по ідентифікації фішингу

Всього протестовано URL	542
Фішинг	258
Легітимні	284
Істинні позитивні	241
Хибні негативні	17
Хибні позитивні	9
Істинні негативні	275
Рейтинг істинно позитивних	93,42%
Рейтинг хибно негативних	5,21%
Рейтинг хибно позитивних	95,23%
Рейтинг хибно негативних	5,26%
Загальний рейтинг	92,4%

Вся система (модуль аналізу URL-адрес та доменів і модуль ідентифікації фішингу) була протестована на наборі даних у таблиці 4. Модуль аналізу URL-адрес та доменів підвищив загальну продуктивність до 96,66%, за рахунок зниження кількості помилкових спрацьовувань з 3,16% до 2,65%.

Також результати аналізу URL-адрес та доменів відображено у таблиці 3.8.

Таблиця 3.8 – Результати ідентифікації фішингу за допомогою модуля аналізу URL-адрес

Фішинг	421
Легітимні	284
Істинні позитивні	405
Хибні негативні	15
Хибні позитивні	9
Істинні негативні	284
Рейтинг істинно позитивних	95,2%
Рейтинг хибно негативних	3,41%
Рейтинг хибно позитивних	98,51%
Рейтинг хибно негативних	2,56%
Загальний рейтинг	96,6%

Отже у даному розділі були проведені тести запропонованих методик та алгоритмів для виявлення та запобіганню найбільш актуальних на сьогоднішній день вразливостей у веб-системах. Було протестовано алгоритми та методи для виявлення XSS атак, SQL ін’єкцій, вразливість Broke Authentification, запропоновано алгоритми які дозволяють забезпечити конфіденційність, автентичність та цілісність. З допоміжними сервісами, а саме PhishTank, було отримано список веб-сайтів, які включали в себе також і фішингові. Дані було оброблено та протестовано на розробленому модулі по ідентифікації фішингових ресурсів.

ВИСНОВКИ

Досліджено існуючі наявні вразливості у сфері веб ресурсів, проведено аналіз різноманітних методик та алгоритмів для їх виявлення та запобігання.

Досліджено актуальні методи для виявлення та тестування веб-ресурсів на проникність від Cross-Site-Scripting(XSS) атак.

Досліджено алгоритми та стратегії для тестування веб-додатків на наявність вразливості SQL-ін’єкцій.

Проаналізовано вразливість зі зламування аутентифікації(Broken Authentification) та її типи, а саме: викрадення сеансу(hijacking), надсилання облікових даних(credential stuffing), атака з розпиленням паролю (password spraying) та фішинг(phishing).

Досліджено роботу алгоритму Кнута-Моріса-Пратта(KMP) для формулювання фукнцій фільтра, який було використано у алгоритмі для тестування та запобігання XSS-атак та SQL-ін’єкцій

Розроблено методи для виявлення та тестування веб-ресурсів на проникність від XSS атак, SQL ін’єкцій, а також механізми та методи для виявлення та запобігання зламу аутентифікації. Для виявлення фішингових сайтів було використано існуючий модуль аналізу URL адреси та доменного ім’я, з подальшим використанням його у запропонованому модулі для ідентифікації потенційно загрозливих фішингових сайтів. Для кращого розуміння було наведено схеми роботи різних модулів та методів.

Проведено програмне тестування розроблених алгоритмів та методів, результати яких показали високу продуктивність та ефективність в межах проведеної роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. J. S. Park and R. Sandhu, “Secure cookies on the web,” IEEE Internet computing, vol. 4, no. 4, pp. 36–44, 2000.

2. O.P. Voitovych, O.S. Yuvkovetskyi, L.M. Kupershtein, SQL injection prevention system, International Conference “Radio Electronics & InfoCommunications” (UkrMiCo) (2016), pp. 2–5

3. Защита от XSS [Електронний ресурс] – Режим доступу: URL: http://www.spysoft.net/zashhita-ot-xss/ Назва з екрану.

4. G. Ahmad, A hybrid method for detection and prevention of SQL injection attacks, Computing Conference (London, 2017), pp. 833–838

5. Y. Jang, J. Choi, Detecting SQL injection attacks using query result size. Comput Security, 1–15 (2014) https://doi.org/10.1016/j.cose.2014.04.007

6. Testing for SQL Injection (OTG-INPVAL-005) – OWASP. [Електронний ресурс]. Режим доступу: https://www.owasp.org/index.php/103.

7. What is Cross Site Scripting (XSS). URL: https://www.geeksforgeeks.org/what-is-cross-site-scripting-xss/

8. Jovanovic N., Kruegel, C.; Kirda, E. Pixy: A static analysis tool for detecting web application vulnerabilities. Proceedings of the 2006 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, 2006.

9. A.O. Christiana, A.N. Oluwatobi, G.A. Victory, O.R. Oluwaseun, A Secured One Time Password Authentication Technique using (3, 3) Visual Cryptography Scheme. IOP Conf. Series: Journal of Physics: Conf. Series 1299, 1–10 (2019 https://doi.org/10.1088/1742-6596/1299/1/012059)

10. C. Ping, A second-order SQL injection detection method, 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (2017), pp. 1792–1796

11. Clickjacking [Електронний ресурс] – Режим доступу до ресурсу: https://www.imperva.com/learn/application-security/clickjacking

12. H. Huang, L. Qian, Y. Wang, “A SVM-based technique to detect phishing URLs,” Inf.Technol. J. vol. 11, no. 7, pp. 921–925, 2012.

13. Justin Clarke, Rodrigo Marcos Alvarez, Dave Hartley, Joseph Hemler, Alexander Kornbrust, Haroon Meer, Gary O’Leary-Steele, Alberto Revelli, Marco Slaviero, Dafydd Stuttard «SQL Injection Attacks and Defense», pp. 148-156 ,2009

14. Content Security Policy (CSP) Quick Reference Guide [Електронний ресурс] – Режим доступу до ресурсу: https://content-security-policy.com/

15. APWG, "Phishing Activity Trends Report", APWG, 2016

16. M. Khonji, Y. Iraqi and A. Jones, "Phishing Detection: A Literature Survey", IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2091-2121, 2013.

17. D. L. Cook, V. K. Gurbani, and M. Daniluk, “Phishwish: A stateless phishing filter using minimal rules,” in Financial Cryptography and Data Security, G. Tsudik, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 182–186

18. SQL инъекции. Проверка, взлом, защита. [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/130826/.

19. "History of Phishing | Phishing.org", Phishing.org, 2017. [Електронний ресурс] – Режим доступу до ресурсу: http://www.phishing.org/history-of-phishing/. [Accessed: 11- Feb- 2017].

20. Hadnagy, C. (2010). Social engineering: The art of human hacking. Indianapolis, IN: John Wiley & Sons.

21. Jerry Louis, “Detection of Session Hijacking,” in University of Bedforshire, January 2011.

22. Chirag R Desai, Dr. Narendra M Shekokar, “Prevention of session hijacking attack using enhanced binding technique”

23. М. Егоров, “Выявление и эксплуатация SQL-инъекций в приложениях”, Защита информации. INSIDE, № 2, с. 2-8, 2011

24. McCluskey, L.; Thabtah, F.; Mohammad, R.M. Intelligent Rule-based Phishing Websites Classification. IET Inf. Secur. 2014,8, 153–160.

25. A. Y. Fu, L. Wenyin, and X. Deng, “Detecting phishing web pages’ visual similarity assessment based on earth mover’s distance (emd),” IEEE Trans. Dependable Secure. Comput. vol. 3, no. 4, pp. 301–311, Oct. 2006.

26. M. Khonji, Y. Iraqi and A. Jones, "Phishing Detection: A Literature Survey", IEEE Communications Surveys & Tutorials, vol. 15, no. 4, pp. 2091-2121, 2013

27. A. Prabakar, M. KarthiKeyan, K. Marimuthu, An efficient technique for preventing SQL injection attack using pattern, International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN) (2013), pp. 503–506.

28. Філіпчук М.М. Алгоритм захисту веб-ресурсів. Матеріали наукової конференції «Автоматизація та комп’ютерно-інтегровані технології» (АКІТ - 2022), Тернопіль, 2022. – С.110-113.

29. Філіпчук М.М. Алгоритми тестування безпеки веб-ресурсів. Матеріали наукової конференції «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2022), Тернопіль, 2022. – С.65-67.

ДОДАТОК А.

Копія виданих публікацій

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ ПОЛІТЕХНІЧНИЙ УНІВЕРСИТЕТ
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2022)

науково-практична конференція
молодих вчених, аспірантів та студентів

29–31 серпня 2022
Тернопіль

УДК 004.056

***Філіпчук М.М.*[1]**

[1]*Західноукраїнський національний університет*

АЛГОРИТМИ ТЕСТУВАННЯ БЕЗПЕКИ ВЕБ-РЕСУРСІВ

Вступ. За останні два десятиліття програмне забезпечення охопило весь світ і стикається з багатьма захоплюючими проблемами. Веб-програми стали обов'язковими для повсякденного життя людей, а деякі з них часто використовують функціональні веб-додатки, такі як онлайн-банкінг, веб-пошта, онлайн-аукціони, онлайн-продажі, соціальні мережі.

Мережі та блоги є головним цільовим місцем для зловмисників. Необхідно виконати належну обробку входу, перевірити синтаксис та дотримуватись вказівок із безпеки, щоб захиститися від основних лазівок на етапі програмування [1-5].

Мета: Тестування запропонованого алгоритму проти атаки SQL-ін'єкції для забезпечення інформаційної безпеки.

1. Аналіз останніх досліджень і публікацій

Атака SQL-ін'єкцій (впровадження коду) є найпоширенішим і найпростішим типом техніки уразливості прийняті зловмисниками через веб-додатки. За допомогою простих команд SQL такі як Select, Where, Insert, Delete та Update, зловмисники ефективно реструктурують фактичний код SQL (вирази) і виконують вразливий код у веб-додатках.

Атаки SQL-ін'єкції є більш прибутковими для зловмисників в основному зосереджені на викрадених банківських рахунках, номерах кредитних карток тощо. Найчастіше користуються перевагами помилково переданих параметрів, помилкового типу обробок, помилкове використання операторів SQL, наприклад (‘ АБО) 1 = -- ’). Подолання цих типів атак не є простим, оскільки зловмисник фактично змінює поведінку попередньо визначених запитів SQL [5, 6 - 8].

2. Алгоритм тестування безпеки

Багато дослідників вивчали низку методів виявлення та запобігання SQLIA; найбільший обрані методики: статичний аналіз, динамічний аналіз, комбінований статичний і динамічний аналіз, веб фреймворк, захисне програмування та методи машинного навчання. Перевірка моделі, аналіз потоку даних, абстрактна інтерпретація та використання тверджень у вихідному коді є кількома методами статичного аналізу коду.

Метод динамічного аналізу може виконуватися автоматично шляхом аналізу уразливості під час виконання веб-додатків, що дозволяє уникнути багато тестів, виконуючись кілька разів вручну. Нещодавно запропонований алгоритм заснований на динамічній техніці, яка порушує напади з використанням SQL-ін'єкції.

Алгоритм складається з наступних кроків, які наведено нижче [9, 10]:

01 Вхідні дані: Fm позначає набір форм із набором полів.

02 Вхідні дані: Enumerate Form_Status FS = {attack, free}

03 Вхідні дані: значення за замовчуванням FS як вільний

04 Вихідні дані: FS

05 for each fm' □ Fm do

06 for each f' □ fm' do

07 f' к fields contains the values

08 if f' is not empty string then

09 FS к output from the method CheckVulnerability (f')

10 if FS as attack

11 D к Add the field f' in the collection

12 Reset the Http requests to issue warning;

13 return FS;

Цей алгоритм зосереджений на розробці атак IF (Injection Free), де як особливий вид тесту, набір розроблено для виявлення атак SQL-ін'єкцій. Алгоритм виконує свою функцію, призначаючи статус форми (FS) як атаку, отриманих із колекції форм. Форма (fm') отримується з набору форм (Fm), а значення кожного поля – з форми (Fm'). Три чітко визначені функції генеруються всередині методу під назвою CheckVulnerability (f') для перевірки будь-яких спеціальних символів, ключові слова та логічні символи.

```
CheckVulnerability (f'):
        for each f' fields do
if f' is a non-empty string then
        // to check for special characters in the input fields and parameters
        p = collection of compiled special characters like {([',&+=<><=>=])}
 for each tokens ff' ™ f' then
         ff'' = compile ff' to make all the input tokens neutralize.
         v = comparison of P and ff';
 if v is not true then return v;
        // to check for keywords in the input fields and parameters
         k = collection of keywords {union |select |intersect|insert|update|delete|drop|truncate}
 for each tokens ff' ∟ f' then
        ff'' = compile ff' to make all the input tokens neutralize.
         v = compare ff'' with k;
        if v is not true then return v;
        // to check for Boolean characters in the input fields and parameters
         b = collection of Boolean characters {' or '|'or'|'AND'|'and'"}
        for each tokens ff' □ f' then
```

return v.

Висновок. Запропонований загальний алгоритм характеризується простотою механізму виявлення атаки SQL ін'єкції. Тестування веб-додатків на атаку SQL-ін'єкцій є важливим кроком для забезпечення його продуктивності та якості. Запропонований алгоритм працює набагато швидше і наділений кваліфікованим рішенням проти атак SQL-ін'єкцій.

Перелік використаних джерел.

1. Stephen Thomas, Laurie Williams, Tao Xie, On automated prepared statement generation to remove SQL injection vulnerabilities, Journal of Information and Software Technology, Elsevier Ltd, 2009, pp. 589-598.

2. Abdul Bashah Mat Ali , Ala' Yaseen Ibrahim Shakhatrehb, Mohd Syazwan Abdullahc, Jasem Alostadd, SQL-injection vulnerability scanning tool for automatic creation of SQL-injection attacks, Journal of Procedia Computer Science, Elsevier Ltd, 2010, pp. 453-458.

3. Dimitris Mitropoulos, Diomidis Spinellis, SDriver: Location-specific signatures prevent SQL injection attacks, Journal of Computers & Security, Elsevier Ltd, 2009, pp.121-129.

4. Joa˜o Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, and Rui Neves, Vulnerability Discovery with Attack

5. Injection, IEEE Transactions on Software Engineering, 2010, Vol. 36, pages: 357- 370.

6. Inyong Lee, Soonki Jeong, Sangsoo Yeo, Jongsub Moon, A novel method for SQL injection attack detection based on removing SQL query attribute values, Journal of Mathematical and Computer Modeling, Elsevier Ltd, 2011, pp. 1-11.

7. Shaukat Ali, Azhar Rauf, Huma Javed, SQLIPA: An Authentication Mechanism against SQL Injection, European Journal of Scientific Research, 2009, Vol.38, pp. 604-611.

8. Ivano Alessandro Elia, José Fonseca, Marco Vieira, Comparing SQL Injection Detection Tools Using Attack Injection:

9. An Experimental Study, 21stIEEE International Symposium on Software Reliability Engineering, 2010, pp. 289-298.

10. J. Park, B. Noh, SQL injection attack detection: profiling of web application parameter using the sequence pairwise alignment, Journal of Information Security Applications, LNCS, 2007, vol. 4298, pp. 74-82.

11. Z. Su, G. Wassermann, The essence of command injection attacks in web applications, 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Charleston, SC, USA, 2006, pp. 372–382.

12. W.G.J. Halfond, A. Orso, P. Manolios, WASP: protecting web applications using positive tainting and syntax-aware.

ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ПРИКАРПАТСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТЕФАНИКА
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ
НАЦІОНАЛЬНИЙ ТРАНСПОРТНИЙ УНІВЕРСИТЕТ
НАДВІРНЯНСЬКИЙ КОЛЕДЖ НТУ
ГАЛИЦЬКИЙ КОЛЕДЖ ІМ. В. ЧОРНОВОЛА

Проблемно-наукова міжгалузева конференція

АВТОМАТИЗАЦІЯ ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

(АКІТ – 2022)

21—23 лютого 2022 року

Тернопіль

БЕЗПЕКА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

УДК 004.056

***Філіпчук М.М.*[1]**

[1]*Західноукраїнський національний університет*

АЛГОРИТМИ ЗАХИСТУ ВЕБ-РЕСУРСІ

Вступ. Сьогодні Інтернет наповнений різними веб-додатками. Одна категорія речей, які можуть спричинити багато проблем є помилки в безпеці. Деякі з них зумовлені помилками в програмуванні, деякі через недосвідченість або іншим чином нездатність захистити систему від шкідливих наслідків. Незалежно від того, чому здійснюються атаки, дуже важливо переконатися, що атаки не будуть успішними. Безпека веб-інформаційних систем є особливо актуальною проблемою. Вона зводиться до методів і алгоритмів забезпечення кожного з трьох рівні безпеки, які повинна мати кожна інформаційна система – автентифікація, авторизація та безпека даних. Запропоновані алгоритми автентифікації користувача, авторизація та доступ до даних, які об'єднані в один повний алгоритм для забезпечення безпеки інформаційної системи.

Мета: забезпечення інформаційної безпеки використовуючи алгоритми автентифікації користувача, авторизації та доступу до даних, які об'єднані в один алгоритм.

1. Аналіз останніх досліджень і публікацій

Основною проблемою в Інтернеті є ідентифікація користувача. Клієнта в магазині можна впізнати за зовнішнім виглядом, але це неможливо в Інтернеті. Хоча в реальному житті людина може представляти себе за іншу людину, це також ще простіше в Інтернеті. Ніхто не може бути впевнений щодо особи іншої людини в Інтернеті, якщо не використовується допоміжна технологія. Цифровий сертифікат є частиною технології, що називається інфраструктурою відкритих ключів (PKI). Це цифровий аналог так званих ідентифікаційних карток (ID) [1].

Подібно до цих карток, сертифікат клієнта містить персональні дані, пов'язані з користувачем, який використовує сертифікат, а також дані про організацію, що видала сертифікат. Схожий до особистої картки, сертифікат також буде належним чином «керований»; він не буде переданий іншим особам і буде захищений від втрати або пошкодження. Якщо виникне проблема, організація, що видає, буде негайно повідомлена, щоб зробити його недійсним і видати новий сертифікат [4]. Атаки на веб-системи можуть бути спрямовані на операційну систему, додаток інформаційної безпеки або базу даних (БД).

2. Алгоритм захисту

Існує велика різноманітність підходів, які можна використовувати для забезпечення безпеки веб-програми. Кожен з них має свої переваги та недоліки. Забезпечення безпеки програми має особливе значення для веб-інтерфейсу інформаційної системи. Для цього будуть використані три рівні механізмів забезпечення безпеки кожного з них, які повинна мати кожна інформаційна система – автентифікація, авторизація та безпека даних [2]. Метою цієї розробки є розробка алгоритмів забезпечення кожного з трьох рівні безпеки - автентифікація, авторизація та доступ до даних в інформаційних системах на базі Інтернету.

Алгоритми реалізовані на прикладному рівні і таким чином, вони стають незалежними від платформи та технологій, що використовуються для системи впровадження. Система аутентифікації користувача здійснюється на основі введеного імені користувача та пароля, а також цифровий сертифікат клієнта [3]. Захист інформації, що зберігається в БД здійснюється з використанням цифрових підписів. Для захисту від так званої DoS (Denial of Service) атаки, спрямованої на модуль автентифікації користувача. Кількість невдалих спроб входу в систему з даної IP-адреси визначається системою моніторингу [8, 9]. У разі певної кількості невдалих спроб інформаційна система ігнорує всі запити, надіслані відповідною IP- адресою користувача. Алгоритм характеризується тим, що в разі невдалої авторизації користувач вже авторизованого системою, сеанс користувача закрито, користувач отримує блокування доступ до відповідної IP-адреси, про що повідомляється адміністрація системи. Алгоритм захисту інформаційної системи з веб-інтерфейсом складається з наступних кроків:

1. Вхід користувача в систему. Ім'я користувача, пароль і клієнту потрібен цифровий сертифікат.

2. Перевірка заблокованого доступу з цієї IP-адреси. Якщо перевірка пройшла успішно, система переходить до пункту 15; інакше система переходить до пункту 3.

3. Виконується процедура аутентифікації користувача. У разі успішної аутентифікації система переходить на пункту 4; інакше система переходить до пункту 10.

4. Вхід користувача в інформаційну систему фіксується.

5. Лічильник, який фіксує кількість невдалих спроб входу користувача в систему за останні M хвилин, скидається.

6. Створюється зашифрований сеанс користувача, у межах якого здійснюється зв'язок між користувачами та ІБ.

7. Виконується процедура авторизації користувача. У разі успішної авторизації, система переходить до пункту 8; інакше система переходить до пункту 13.

8. Виконано процедуру надання доступу до ресурсів ІБ.

9. Для виходу з системи відбувається перехід до пункту 16; інакше система переходить до пункту 7.

10. Невдала спроба входу в ІС з відповідної IP-адреси.

11. Виконується перевірка N невдалих спроб входу в систему зроблених даною IP- адресою за останні хвилини. Якщо перевірка пройшла успішно, система переходить до пункту 12; інакше система переходить до пункту 1.

12. Сеанс користувача закрито.

13. Доступ користувача з цієї IP-адреси заблоковано на певний період часу.

14. Надсилається повідомлення адміністрації системи про спробу несанкціонованого входу в систему. Система переходить до пункту. 1.

15. Користувачеві заборонено доступ до ресурсів ІБ. Виводиться повідомлення про помилку, система переходить до пункту 17.

16. Сеанс користувача закрито.

17. Кінець.

Запропонований алгоритм забезпечує захист БД на основі цифрових підписів. Реалізується на прикладному рівні. Це дозволяє підписати лише частину даних, що зберігаються в БД, що підвищує продуктивність системи. Алгоритм особливо підходить для інформаційних систем, де є кілька авторизованих користувачів та надає доступ для модифікації різних частин даних. Приватний і відкритий ключ авторизованого користувача ІС, необхідного для цифрового підпису та перевірки даних, зберігаються в базі даних [7]. Відкритий ключ, який використовується для перевірки підпису, зберігається незашифрованим, при цьому закритий ключ у ньому, який використовується для цифрового підпису даних, записується в БД у зашифрованому вигляді.

Шифрування реалізовано за допомогою симетричного ключа. Симетричний ключ шифрування/дешифрування (SEDK) користувача, закритий ключ генерується за певним алгоритмом. В даних отриманих від користувача використовується цифровий сертифікат. Ця техніка не вимагає зберігання SEDK в базі даних, у програмі або на диску [5, 6].

Висновки. В роботі запропоновано алгоритм забезпечення безпеки інформації системи з веб-інтерфейсом. У обговоренні було запропоновано можливі алгоритми автентифікації, авторизації та захисту даних. Використовуються цифрові сертифікати клієнтів. Запропонований алгоритм характеризується тим, що сертифікати клієнтів перевіряються веб-сервером. Усі запити проходять фільтрацію, незалежно від того, хто їх відправник та блокуються у разі спроби несанкціонованого доступу. Автентичність даних у базі даних забезпечується цифровим підписом. Запропоновані методи та алгоритми можуть бути реалізовані в кожному веб-додатку незалежно від його конкретного призначення, а також більш удосконалено в плані безпеки для користувача.

Перелік використаних джерел.

1. Eugene Lebanidze, Securing Enterprise Web Applications the Source: An Application Security Perspective, OWASP.

2. WEB APPLICATION SECURITY, The Government of the Hong Kong Special Administrative Region, 2008, available: http://www.infosec.gov.hk/english/ technical/files/web app.pdf.

3. American Bar Association, Digital Signature & PKI Assessment Guidelines, http://www.abanet.org/scitech /ec/isc/dsg-tutorial.html.

4. Judith V. Boettcher, Amanda Powell, Digital Certificates, http://www.cren.net/ crenca/docs/syllabus.pdf, 2002.

5. Gary C. Kessler, An Overview of Cryptography, available: http://www.garykessler.net /library/crypto.html, 2014.

6. Laboratories, R. PKCS #1 v2.1: RSA Cryptography Standard, Available from: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

7. Emil Burtescu, database security - attacks and control methods, Department of Accounting and Management Informatics, University of Pitesti, Pitesti, Romania.

8. Qijun Gu, Peng Liu, Denial of Service Attacks, 2007, available: http://s2.ist.psu.edu/paper/DDoS-Chap-Gu-June-07.pdf.

9. Common Attack Pattern Enumeration and Classification [Електронний ресурс] – Режим доступу: https://capec.mitre.org/

ДОДАТОК Б.

Лістинг коду програми

Phishing scraper

```
require 'phishtank_scraper/phishing_set'
require 'phishtank_scraper/site'

# Director interface for scraping
class PhishtankScraper
  attr_reader :site, :range

  def initialize(url="http://phishtank.com")
    @site = Site.new(url)
    @range = (0..0)
  end

  # returns an array of detections in the pages range
  # options:
  # active: "All", "n", "y", "u"
  # valid: "All", "n", "y", "u"
  def page_scrape(range=@range, options={})
    build_range(range).map do |page_index|
      PhishingSet.new(@site.build_path(page_index, options)).all
    end.flatten
  end

  # returns an array of detections from id to last submitted id
  # options:
  # active: "All", "n", "y", "u"
  # valid: "All", "n", "y", "u"
  def id_scrape(since, options={})
```

```
    since = since.to_i
    page_at = PhishingSet.new(@site.home).page_at_id(since)

    phset = (0..page_at).map do |page_index|
      PhishingSet.new(@site.build_path(page_index, options)).all
    end.flatten

    phset.delete_if {|ph| ph[:id].to_i < since}
  end

  private
  def build_range(value)
    @range = value.class.eql?(Range) ? value : (value..value)
  end
end


# Prvides routes and URLs
class Site
  attr_reader :domain
  def initialize(url)
    @domain = url
  end
  def home
    build_path
  end
  def build_path(page_index=0, options={})
    active = options[:active]
    valid = options[:valid]
    path = if active or valid
```

```
      actives =  "&active=" + (active || "y")
      valid =  "&valid=" + (valid || "y")
      "phish_search.php?page=#{page_index}#{active}#{valid}&Search=Search"
    else
      "phish_archive.php?page=#{page_index}"
    end
    "#{@domain}/#{path}"
  end
  def build_detail_path(submission_id)
    "#{@domain}/phish_detail.php?phish_id=#{submission_id}"
  end
end
require 'nokogiri'
# require 'random_user_agent'
require 'open-uri'

# Creates a collection of phishing hashes for each phishtank page
# Example
# {
#   id: "4141251",
#   url: "http://bintango.xyz/AIsaE",
#   created_at: "added on Jun 3rd 2016 3:57 PM",
#   submitter: "PhishReporter",
#   valid: "Unknown",
#   online: "ONLINE"
# }

class       PhishingSet
  include Enumerable
  attr_reader :url, :all
```

```
def initialize(url)
  @url = URI(url)
  @page = Nokogiri::HTML(open(@url.to_s))
  @all = scrape_parse
end
def scrape_parse
  rows = @page.at('.data').search('tr')
  rows.shift #removes header

  rows.collect do |row|
    id = row.at_xpath('td[1]/a/text()').to_s.strip
    url_id = row.at_xpath('td[1]/a/@href').to_s.strip
    url = row.at_xpath('td[2]/text()').to_s.strip
    url = scrape_detail(url_id) if url[-3,3] == "..." # incomplete url
    {
      id: id,
      url: url,
      created_at: row.at_xpath('td[2]/span/text()').to_s.strip,
      submitter: row.at_xpath('td[3]/a/text()').to_s.strip,
      valid: row.at_xpath('td[4]/text()').to_s.strip,
      online: row.at_xpath('td[5]/strong/text()').to_s.strip
    }
  end
end
def scrape_detail(url)
  uri = URI(url)
  uri = uri.host ? uri : "#{@url.scheme}://#{@url.host}/#{uri}"
  detail_page = Nokogiri::HTML(open(uri))
  detail_page.at("#widecol").at_xpath("div/div[3]/b/text()").to_s
```

```
  end
  def first
    @all.first
  end
  def page_at_id(id)
    last_subm_id = self.first[:id].to_i
    ((last_subm_id - id + 1)/20).round # 20 items per page
  end
  def each
    @all.each{ |ph| yield ph }
  end
end
```