НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

До захисту допущено
В.о. завідувача кафедри
_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)
«_____»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною программою «Системи, технології та математичні методи кібербезпеки»

спеціальності: 125 «Кібербезпека»

на тему: Автоматизоване тестування ВебСокетів з використанням ZAP

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-73

Наумейко Лілія Андріївна _____
(підпис)

Керівник доцент Барановський Олексій Миколайович _____
(підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Здобувач вищої освіти _____

Київ - 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 125 «Кібербезпека»

Освітньо-професійна програма «Системи, технології та математичні методи кібербезпеки»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Микола ГРАЙВОРОНСЬКИЙ
(підпис)
«___»_____2021 р.

ЗАВДАННЯ
на дипломну роботу здобувачу вищої освіти

Наумейко Лілії Андріївни
(прізвище, ім'я, по батькові)

1. Тема роботи Автоматизоване тестування ВебСокетів з використанням ZAP, керівник роботи Барановський Олексій Миколайович, доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від « » _____ 2021 р. №

2. Термін подання здобувачем вищої освіти роботи 07 червня 2021 р.

3. Вихідні дані до роботи
   1. Попередні дослідження.
   2. Попередні тестування вебпрограм, що працюють по ВебСокет протоколу.

4. Зміст роботи

1. Вивчити технології ВебСокет протоколу;
2. Дослідити вразливості в ВебСокет протоколі;
3. Проаналізувати типи тестування вразливостей в ВебСокет протоколі;
4. Дослідити вебпрограми, які працюють по ВебСокет протоколу;
5. Вивчити типи скриптів для аналізу ВебСокетів в ZAP;
6. Написати скрипти для аналізу ВебСокетів;
7. Оцінити отриманий результат тестування використовуючи написані скрипти;
8. Порівняти ефективності різних типів тестування.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

Презентація

6. Дата видачі завдання 20.10.2020

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів дипломної роботи	Примітка
1	Дослідження існуючих проблем	20.10.2020	
2	Визначення завдання	06.11.2020	
3	Збір та обробка інформації	11.03.2021	
4	Дослідження предметної області	23.03.2021	
5	Побудова плану роботи	15.04.2021	
6	Вивчення інструментів використання	17.04.2021	
7	Виконання практичної частини	27.04.2021	
8	Проведення дослідження	06.05.2021	
9	Обробка результатів	17.05.2021	
10	Оформлення дипломної роботи	27.05.2021	
11	Отримання допуску до захисту	01.06.2021	

Здобувач вищої освіти _____ Лілія Наумейко

(підпис) (Власне ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ Олексій БАРАНОВСЬКИЙ

(підпис) (Власне ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота містить 49 сторінок, 22 ілюстрації, 4 таблиці, 3 додатки і 11 джерел посилань.

Об'єктом дослідження є вразливості у ВебСокет протоколі.

Предметом дослідження є проксі ZAP та написані скрипти.

Метою роботи є детальне вивчення тестування безпеки ВебСокетів, а саме детальне вивчення ВебСокет протоколу, типів тестування вразливостей у ВебСокет протоколі, практичне використання Zed Attack Proxy та написання скриптів для обраних типів вразливостей в існуючих вебпрограмах.

Як результат досягнення мети було написано скрипти, запущено на основі ZAP, та протестовано на вебпрограмах з ціллю оцінити отриманий результат.
Ключові слова: ВебСокети, тести на проникнення, ZAP, вразливості, OWASP10, скрипти, автоматизація процесу.

ABSTRACT

The thesis consists of 49 pages, 22 illustrations, 4 tables, 3 appendix and 11 reference sources.

The object of the study are vulnerabilities in the WebSocket protocol.

The subject of research is the ZAP proxy and written scripts.

The goal of the work is a research of WebSocket security testing, especially a detailed study of the WebSocket protocol, types of vulnerability testing in the WebSocket protocol, practical use of Zed Attack Proxy and writing scripts for selected types of vulnerabilities in existing web programs.

As a result of the achievement of the goal, scripts were written, run based on ZAP, and tested on web programs to evaluate the result.

Keywords: WebSocket, penetration tests, ZAP, vulnerabilities, OWASP10, scripting, automation of the process.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP - The Hypertext Transfer Protocol

HTTPS - The Hypertext Transfer Protocol over SSL(Secure)

WS - WebSocket

WSS - WebSocket Secure

SOP - Same Origin Policy

GDPR - General Data Protection Regulation

API - Application Programming Interface

REST - Representation State Transfer

CSRF - Cross-Site Request Forgery

XSS - Cross-Site Scripting

SSL - Secure Sockets Layer

TLS - Transport Layer Security

ВСТУП

Стан безпеки програм, що використовують ВебСокет протокол для комунікації, не є винятком та містить вразливості, які зловмисник може потенційно використати.

З ціллю захисти вебпрограму та уникнути потенційних вразливостей, потрібно вивчити як виявляються ті чи інші типи вразливостей, як їх експлуатувати та як від них захистити вебпрограму.

Актуальність

Досліджуючи поточний стан продуктів, що дозволють ефективно працювати з ВебСокет повідомленнями було зроблено висновок, що загальнодоступні засоби тестування безпеки ВебСокет протоколу не дають змоги цілком протестувати програму. Єдиним дієвим варіантом було перехоплювати кожне повідомлення окремо, та видозмінювати його.

Мета і завдання дослідження

Тому було вирішено розширити вже існуючі інструменти для оптимізації тестування безпеки у ВебСокет повідомленнях. Скрипти були розроблені з метою перевірки окремих вразливостей, виявлених в реалізаціях ВебСокет протоколу.

Варто зазначити, що скрипти не є комплексним рішенням для тестування, вони були написані для детального вивчення вразливостей у вебпрограмах, що працюють використовуючи ВебСокети. Для роботи з скриптами потрібно мати знання у реалізаціях ВебСокет протоколу, в процесі тестування безпеки вебпрограми та користування проксі-серверів, таких як ZAP.

Публікації

Результат досліджень були опубліковані у системі у системі DSpace Науково-технічної бібліотеки ім. Г. І. Денисенка НТУУ «КПІ імені Ігоря Сікорського» та частково представлені на конференції Всеукраїнській науково-практичній конференції студентів на базі Фізико-технічного інституту Національного технічного університету України “Київський політехнічний інститут імені Ігоря Сікорського”.

1 ОГЛЯД ВЕБСОКЕТ ПРОТОКОЛУ

1.1 Опис функціонування ВебСокет протоколу

ВебСокети – це одночасно двонаправлений протокол зв'язку, ініційований через HTTP. Вони зазвичай використовуються в сучасних веб-застосунках для потокової передачі даних та іншого асинхронного трафіку.

Вебсокети дозволяють клієнтському браузеру та серверу обмінюватись інформацією в реальному часі. На відміну від HTTP, клієнт отримує дані не лише після запиту на сервер, а й сервер сам може надсилати дані клієнту.

Більшість комунікацій між веб-браузерами та веб-сайтами відбуваються через HTTP(HTTPS). За допомогою HTTP клієнт відправляє запит, а сервер повертає відповідь. Як правило, відповідь відбувається негайно, і на цьому транзакція завершується. Навіть якщо з’єднання з мережею залишається відкритим, для наступної комунікації буде використаний окремий ланцюг запит-відповідь.

З’єднання ВебСокетів ініціюються через HTTP і, як правило, довговічні. Повідомлення можна надсилати в будь-який бік та у будь-який час. Зв'язок, як правило, залишатиметься відкритим, доки клієнт або сервер не будуть готові надіслати повідомлення. ВебСокети особливо корисні в ситуаціях, коли потрібні повідомлення із низькою затримкою або повідомлення, ініційовані сервером, наприклад, оновлення в реальному часі фінансових даних чи підтримка клієнтів в чатах.

Варто зазначити, що ВебСокет це рамковий протокол(Додаток А), що означає, що шматок даних (повідомлення) ділиться на декілька дискретних шматочків, розмір фрагмента кодується у кадрі. Кадр включає тип кадру, довжину корисного навантаження та частину даних.

1.2 Опис технічних специфікацій ВебСокет протоколу

Для аналізу та тестування вебпрограм, повідомлення яких передаються по ВебСокет протоколу, потрібно детально дослідити технічні специфікації протоколу.

Як було вже вказано раніше, перед початком обміну повідомленнями клієнт (браузер) відправляє на сервер HTTP-запит для ініціалізації[1] транспорту повідомлень по ВебСокет протоколу із заголовками вказаними в таблиці нижче (таблиця 1.1).

Таблиця 1.1 - Запит на ініціацію ВебСокет з’єднання

GET /chat HTTP/1.1 Host: testingsite.com **Upgrade**: websocket **Connection**: Upgrade **Sec-WebSocket-Key**:a2V5IGZvciBkaXBsb21hIA== Sec-WebSocket-Protocol: chat, superchat Origin: http://somehost.com **Sec-WebSocket-Version**: 13

Перші два заголовки - це стандартні заголовки протоколу HTTP. Відмінність від звичайного HTTP запиту починається вже з третього рядка запиту, а саме з заголовку Оновлення(Upgrade: websocket). Заголовок Оновлення використовується клієнтом(браузером), з ціллю переключити сервер на один із перерахованих протоколів у порядку спадання. Так як потрібно встановити з'єднання WebSocket, використовується websocket в ролі параметра заголовку.[2]

Наступним заголовком є Підключення(Connection: Upgrade). Заголовок Підключення зазвичай визначає, чи залишатиметься мережеве підключення відкритим після завершення поточної транзакції. Загальним значенням цього заголовка є підтримка активності, щоб переконатися, що з’єднання постійне, щоб дозволити подальші запити на той самий сервер. Під час відкриття рукостискання Вебсокет встановлюється заголовок з параметром Оновлення(Upgrade), сигналізуючи про те, що зв’язок потрібно зберегти і використовувати його для запитів, що не стосуються HTTP.

Важливим заголовком є Sec-WebSocket-Key, так як він виконує роль ключа, і є одноразовим випадковим значенням (не вираженим), яке генерується клієнтом. Це випадково вибране 16-байтове значення, яке було закодовано base64. [3]

З допомогою заголовку Sec-WebSocket-Protocol з клієнта йде усвідомлення сервера про ті реалізації протоколу, що він підтримує. Провівши паралель з HTTP протоколом, даний заголовок виконує схожу роль як і HTTP-заголовок Content-Type.

Заголовок Sec-WebSocket-Version визначає версію ВебСокет протоколу, що буде використовуватися для з'єднання. Варто зауважити, що єдиною прийнятою версією протоколу WebSocket є 13. Будь-яка інша версія, зазначена в цьому заголовку, недійсна.[1]

Переглянувши запит від клієнта на сервер для ініціалізації ВебСокет зв'язку потрібно також розглянути відповідь про вдале рукостискання[3]. Відповідь сервера, що підтримує ВебСокет з'єднання виглядає приблизно таким чином, як зображено в таблиці нижче(таблиця 1.2).

Таблиця 1.2 - Відповідь від сервера на запит для ВебСокет з'єднання

HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade **Sec-WebSocket-Accept**:aGFzaCBvZiBrZXkgZnJvbSBzZXJ2ZXI= **Sec-WebSocket-Protocol:** chat

Про успішне з'єднання з сервером та зміну протоколів свідчить HTTP код 101. В заголовках Підключення та Оновлення параметри вказують на те, що зєднання оновлене та підтверджене.

У заголовку Sec-WebSocket-Protocol сервер повідомляє обрану ним реалізацію протоколу.

В свою чергу заголовок Sec-WebSocket-Accept містить хеш ключа, відправленого клієнтом. Sec-WebSocket-Accept кодується base64, хеш-значення SHA-1. Це значення генерується шляхом об'єднання ключа, який передається з клієнта в параметрі Sec-WebSocket-Key та статичне значення 258EAFA5-E914-47DA-95CA-C5AB0DC85B11, визначене в RFC 6455. [5]

Існування Sec-WebSocket-Key і Sec-WebSocket-Accept є для того, щоб і клієнт, і сервер могли знати, що їх аналог підтримує ВебСокет протокол. Оскільки ВебСокет протокол повторно використовує з'єднання HTTP, існує потенційна загроза безпеці, якщо будь-яка сторона інтерпретує дані ВебСокет як запит HTTP.[4]

Тобто ці заголовки надсилається із сервера на клієнт, щоб повідомити, що сервер готовий ініціювати підключення до веб-сокета. Після такого обміну передача даних в HTTP подібному вигляді припиняється, і спілкування повністю переходить на ВебСокет.[5]

Для зупинки ВебСокет з'єднання, надсилається закриваючий кадр (код 0x08). Кадр закриття може містити тіло, яке вказує причину закриття. Якщо кожна сторона з'єднання отримує закритий кадр, вона повинна надіслати закритий кадр у відповідь, і більше даних не повинно надсилатися по з'єднанню. Після того, як кадр отримано обома сторонами, з'єднання TCP зривається. Сервер завжди ініціює закриття TCP-з'єднання.

1.3 Проблема дослідження

Досліджуючи та тестуючи вебпрограми було зрозуміло, що не існує жодного іншого варіанту, який зможе забезпечити справжній двосторонній зв’язок, в такому ж форматі як це виконує ВебСокет протокол.

Розглянувши попередні дослідження ВебСокет протоколу було зрозуміло, що існувало дві проблеми, що могли зупинити на шляху використання ВебСокетів у власній програмі, а саме відсутність підтримки браузера та невпевненість в безпеці протоколу. Однак ці проблеми є в минулому(або частково усунені), тому що використання ВебСокет протоколу у вебпрограмах збільшується, через те, що таким чином більша кількість повідомлень може бути надіслана за менший час, що зображено на рисунку нижче(рис.1.1).

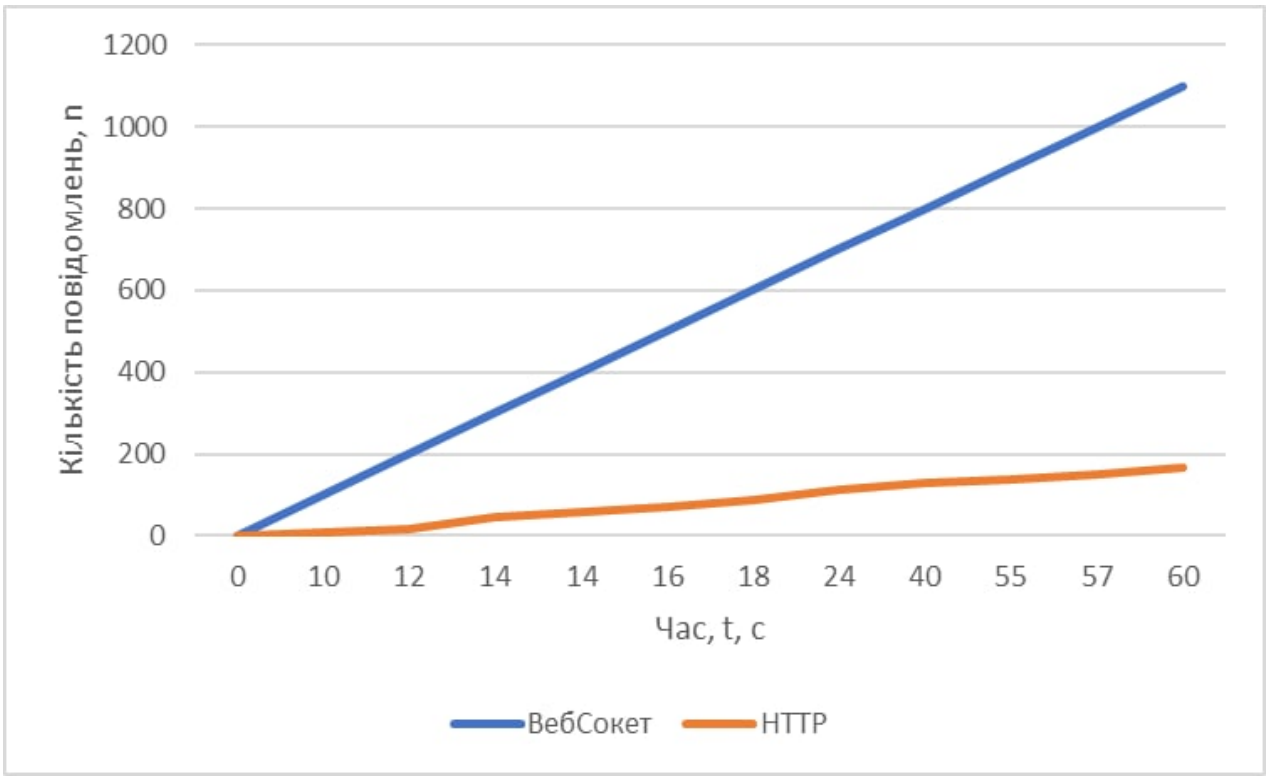


Рисунок 1.1 - Кількість надісланих повідомлень двома протоколами за один і той же час

Зображена на графіку відмінність пояснюється тим, що кожного разу, коли надсилаються дані з RESTful API, потрібно створити нове TCP-з'єднання з сервером, надіславши запит HTTP. Після отримання відповіді від API через HTTP підключення TCP припиняється. ВебСокети ж, з іншого боку, модернізують з'єднання HTTP і підтримують з'єднання в житті, щоб клієнт і сервер могли спілкуватися через одне і те ж з'єднання TCP. Це призводить до величезного виграшу в часі, необхідному для обробки великої кількості повідомлень.[6]

Проблема дослідження ВебСокетів зростає, якщо розглянути, як часто та в яких вебпрограмах використовується цей протокол, тому що саме від вебпрограми, та даних, які передаються, і стає зрозуміло наскільки важливим є захистити цілісність, конфіденційність та доступність. В першу чергу, це соціальні канали, інформація повинна оновлюватися в реальному часі без затримки, однак передається особиста інформація та задіяна велика кількість людей, тому вплив від загрози може бути великий, через витік інформації та обмежений в часі доступ до інформації.

Багатокористувацькі ігри також вимагає швидкого та чіткого інтерактиву, тому веб-розробники впроваджують високопродуктивні ігри у браузері. Треба

зауважити, що в такий тип вебпрограм додаються фінанси, що вимагає дотримання окремих стандартів за GDPR, а саме захисту конфіденційності, цілісності та доступності.

Також ВебСокети використовуються при спільному редагування, наприклад системи контролю версій, так як це уможливлює роботу над одним документом, уникаючи конфліктів об’єднання. Однак під загрозою в такому випадку може бути цілісність та доступність системи.

Окремо ВебСокет протокол застосовується для відтворення взаємодії сервера із клієнтом, що може містити чутливу інформацію, яка корисною зловрендному користувачеві, що зможе порушити конфіденційність інформації.

Висновок до розділу 1

У цьому розділі було розглянуто технічні специфікації ВебСокет протоколу, особливість встановлення зв'язку клієнт-сервер по ВебСокет протоколу, також було порівняно швидкість роботи ВебСокет протоколу з HTTP протоколом та вивчено, в яких веброграмах може використовуватися ВебСокет протокол, та які із властивостей інформаційної безпеки можуть бути порушені.

ВебСокет з’єднання ініціюється через HTTP шляхом рукостискання: надісланий запит на сервер з визначеним заголовками, та при підтримці ВебСокет протоколу сервером отримана відповідь з початком сесії. Після цього етапу, ВебСокет з’єднання існує відокремлено від HTTP, та відрізняється швидкістю надсилання двонаправлених повідомленнь. Залежно від інформації та вебпрограми можуть порушуватися всі три властивості інформації.

Часте використання ВебСокетів вимагає належних методів захисту. Для дослідження безпеки ВебСокет протоколу потрібно тестувати вразливості, для усунення знайдених вразливостей та покращення безпеки вебпрограми чи системи загалом.

2 АНАЛІЗ ТА ТЕСТУВАННЯ ВБСОКЕТ ПРОТОКОЛУ

Поточний стан ВебСокет протоколу не є винятком, коли мова йде про проблеми безпеки у вебпрограмах. Існують вразливості з списку OWASP10, які зазвичай існують у вебпрограмах, що використовують ВебСокети, які будь-який зловмисник може потенційно використати. Саме тому завданням було дослідити вразливості та типи їх тестування у ВебСокетах для покращення безпеки вебпрограм, що працюють по ВебСокетах.

2.1 Огляд вразливостей у ВебСокет повідомленнях

Для огляду вразливостей у ВебСокет протоколі був використаний список OWASP10 та ресурси в інтернеті про знайдені вразливості у вебпрограмах, що працюють по ВебСокет зв’язку.

Автентифікація

Як описувалося в розділі реалізації ВебСокет протоколу, авторизація та автентифікація не обробляється ВебСокетами, і тому для них слід проводити звичайні тести без додаткових знаннь про веб програму(black box). Це означає, що якщо ВебСокет протокол відкрий, він не отримує жодної авторизації та автентифікації, і потрібно зробити деякі додаткові кроки, щоб захистити з'єднання ВебСокет.

Розкриття чутливої інформації

Через ВебСокет повідомлення передається різні типи інформації. Замість того, щоб безпосередньо атакувати вебпрограму, можна вкрасти чутливу інформацію, що передається або зберігається, виконуючи атаки "людина посередині" або викравши чисті текстові дані з сервера або у клієнта користувача.

Найпоширенішим недоліком є просто не шифрування конфіденційних даних. Помилка у відсутності шифрування ВебСокет повідомлень часто компрометує всі дані, які мали бути захищені. Як правило, ця інформація включає конфіденційні дані про особисту інформацію (ІПІ), такі як медичні записи, облікові дані, особисті дані та кредитні картки, які часто потребують захисту, як це визначено законами або нормативними актами, такими як GDPR або місцевим законодавством про конфіденційність.

CSRF

ВебСокет використовує модель безпеки на основі походження, яка зазвичай використовується в браузерах. Це означає, що коли користувач виконує рукостискання, що відкриває WebSocket запит, веб-браузер додає до запиту заголовок HTTP із назвою Origin. Поле заголовка джерела використовується для захисту від несанкціонованого використання джерела сервера використовуючи API ВебСокет веб-браузері.[3] Якщо під час початкового рукостискання ВебСокет заголовок джерела не буде перевірений, сервер може приймати підключення з будь-якого джерела, що може спричинити серйозні вразливості безпеки.

Шифрування трафіку

Цілісність та конфіденційність забезпечує доступ до інформації з розмежування прав та в такому вигляді, в якому інформація була розміщена на сервері. Для того, щоб зберегти конфіденційність та цілісність інформації протягом усього процесу, важливо перевірити, що з'єднання ВебСокет використовує SSL для передачі конфіденційної інформації wss://. Також важливо перевірити реалізацію SSL на наявність проблем безпеки.

Атаки ін'єкції

У контексті вразливостей у веб-програмах перевірка вводу даних відноситься до відмови належним чином обробляти вхідні дані, що надходять від клієнта або сервера. Неправильна обробка вхідних даних може призвести до різного роду вразливостей у веб-додатках, таких як ін’єкіцй даних (наприклад, SQL, LDAP або HTML), атаки на файлову систему та переповнення буфера. Сервер ніколи не повинен довіряти даним, що надходять від клієнта.[4]

Атаки ін'єкції вхідних даних такі ж вірогідні через ВебСокети, як і за будь-яким іншим механізмом, таким як з'єднання HTTP. Три із вразливостей веб-сайтів від OWASP10, а саме XSS, SQLi та RCE, можна віднести до не правильної обробки випадкових вхідних даних.

Потрібно враховувати, що атаки, спрямовані на перевірку вхідних даних, можна розділити на дві основні категорії:

- Атаки, спрямовані на користувача програми.
- Атаки, спрямовані на сервер програми.

2.2 Огляд попередніх типів тестування ВебСокет повідомлень

Розглядаючи можливість тестувати вебпрограми, що працюють через ВебСокети вибір інструментів не великий. В більшості випадків використовуються ZAP та Burpsuite для проксіювання трафіку, що йде з клієнта на сервер та навпаки. І основним методом є ручне тестування, шляхом окремої обробки ВебСокет повідомлень та додаткової зміни параметрів в них.

Мануальне тестування

Інструменти безпеки WebSocket, такі як Burp Suite та OWASP ZAP, використовують функціональність перехоплення трафіку для досягнення необхідних функціональних можливостей з ціллю перевірки безпеки.

В рамках тестування веб-безпеки проксі - це інструмент, який можна використовувати для перенаправлення трафіку через проксі-сервер, який виконує роль посередника, який можна використовувати для перегляду історії, перехоплювати, змінювати або блокувати трафік, що відбувається між двома сторонами.

Коли проксі-сервер налаштований для браузера клієнта, щоб бачити трафік між двома сторонами в інструменті проксі, то трафік із сервера та клієнта проходить через налаштований проксі-сервер так, як зображено на рисунку нижче(рис.2.1).

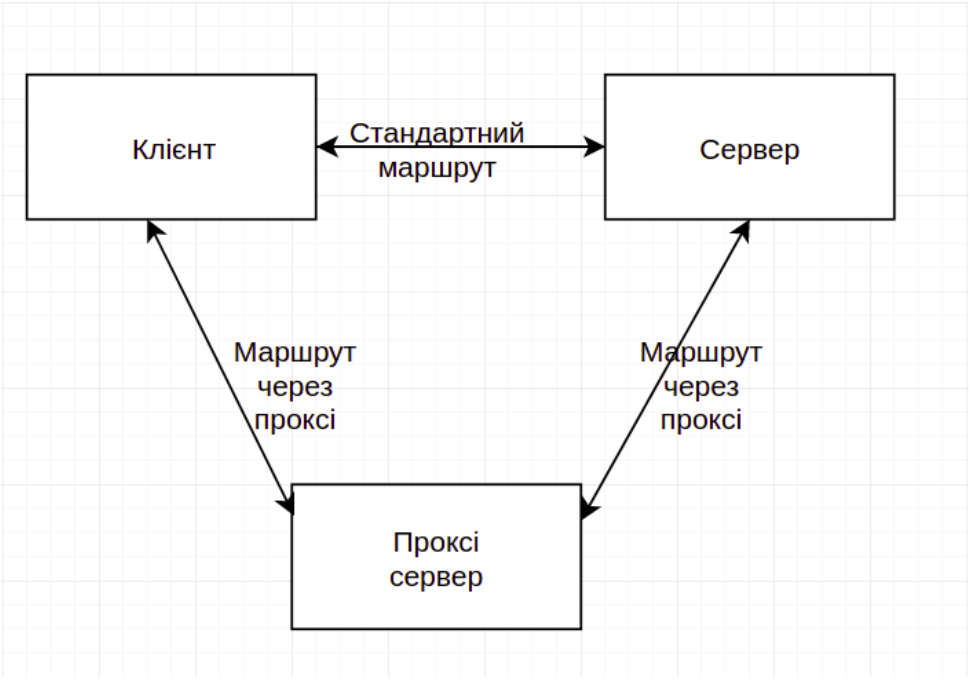


Рисунок 2.1 - Принцип функціонування проксі-сервера.

Коли проксі-сервер налаштований, наприклад, для клієнта браузера, вхідний і вихідний трафік браузера спрямовується через проксі-сервер. Проксі-сервер може вільно переглядати, перехоплювати, змінювати та блокувати трафік.

Слід зазначити, що якщо проксі-сервер використовується із зашифрованими з'єднаннями (наприклад, HTTPS або WSS), клієнт повинен довіряти центру сертифікації, який підписав сертифікат проксі-сервера. У разі тестування безпеки, кореневий сертифікат, що використовується з проксі-сервера потрібно експортувати та встановити клієнту, щоб уникнути будь-яких проблем з довірою сертифікатах.

При мануальному тестуванні процес включає в себе підключення до служби ВебСокетів, шляхом ініціації зв’язку, а потім зупинку запитів, що проходять через проксі-сервер, та відкриття їх в окремому вікні для модифікації та пересилання на сервер вебпрограми.

Щоб перевірити вручну вразливості автентифікації у вебпрограмі, що працює по ВебСокет протоколу, потрібно обрати для пересилання запит на рукостискання ВебСокет зв’язку, який містить заголовок Cookie, і тестувати шляхом обходу контролю автентифікації, а саме видалення всього заголовку, видалення параметру заголовку, пустий параметр заголовку чи модифікація змінної, як зображено на рисунку нижче(рис.2.2).

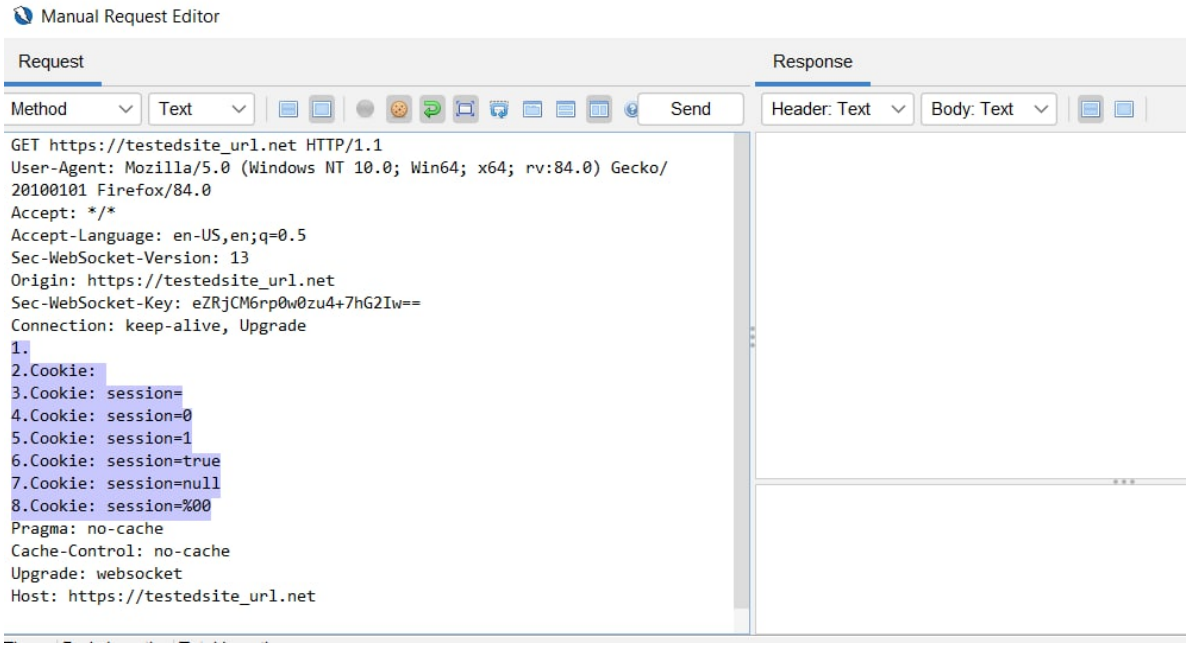


Рисунок 2.2 - Приклад обходу контролю автентифікації

Щоб протестувати міждоменні запити потрібно врахувати, що за замовчуванням у ВебСокет протоколі, єдине, що керує міждоменним запитом, це заголовок Джерела(Origin).

Заголовок Джерела надсилається в рамках запиту на відкриття рукостискання ВебСокет клієнтом і перевіряється сервером. Використовуючи проксі-інструменти можна маніпулювати вільно заголовками, як зображено на рисунку нижче(рис.2.3), і якщо на сервері не перевіряється заголовок Джерела та обробляє будь-які параметри, можна вважати це вразливістю.

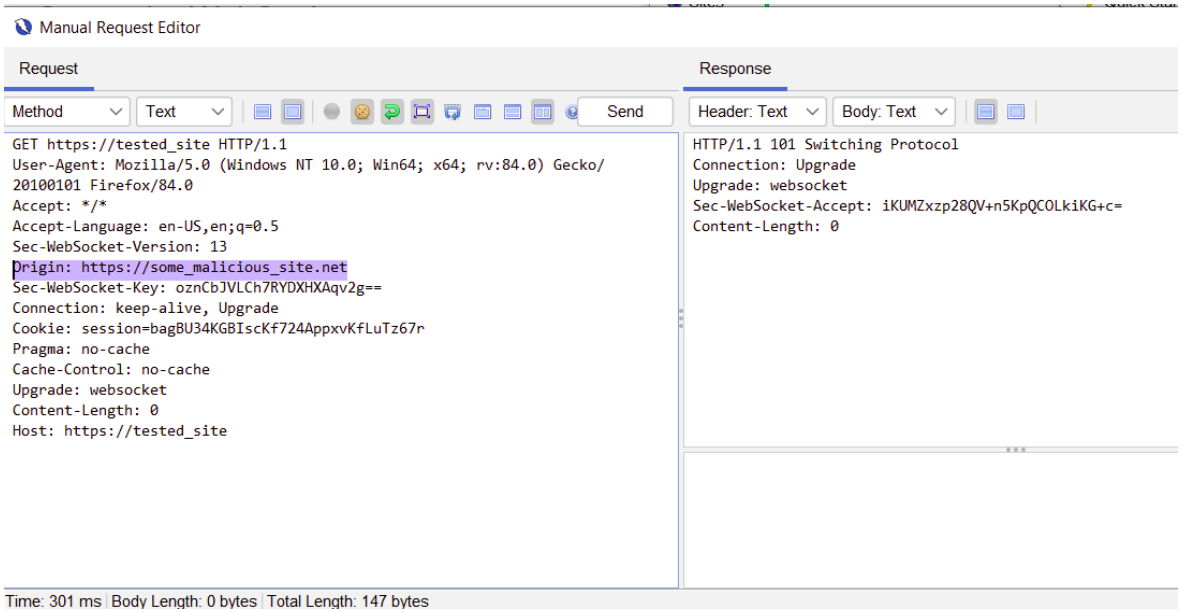


Рисунок 2.3 - Приклад успішної маніпуляції заголовком джерела

За замовчуванням у ВебСокет протоколі якщо сервіс ВебСокет не використовує зашифроване з'єднання TLS (URI: wss: //) потрібно перевіряти, чи взагалі підтримує сервіс TLS шифрування взагалі.

Після встановлення зв’язку потрібно переглянути історію проксі-сервера та редагувати запит для рукостискання. В окремому вікні, як зображено на рисунку нижче(рис.2.4) змінюючи налаштування деталей зв'язку потрібно обрати необхідний порт та використання HTTPS.

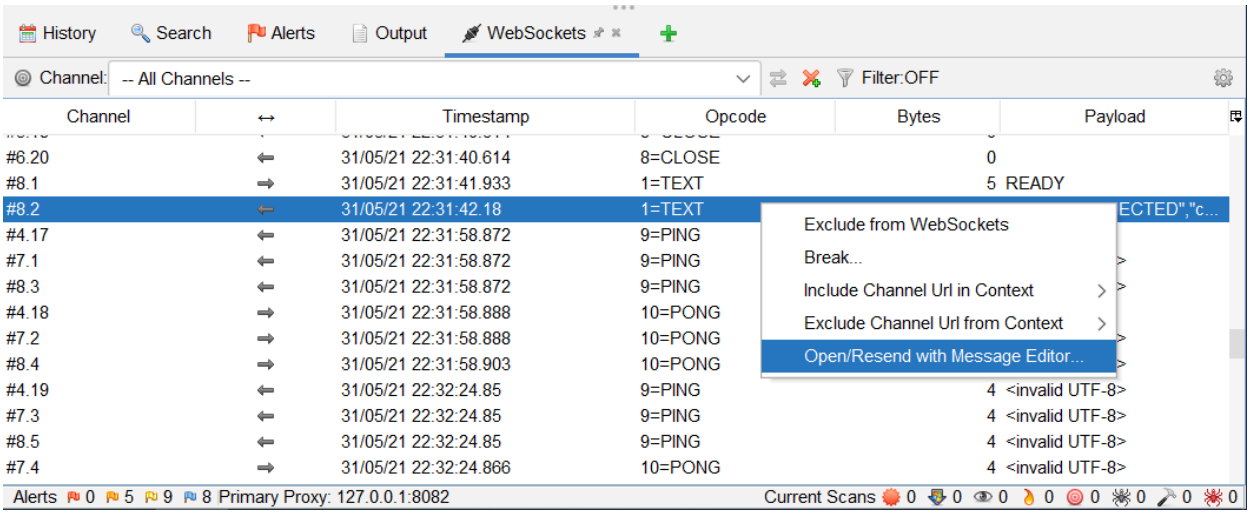


Рисунок 2.4 - Мануальне пересилання ВебСокет повідомлень в ZAP

Якщо відповідь на змінений запит схожа на відповідь зображену в таблиці нижче (таблиця 2.1), то ВебСокет сервіс може працювати з шифруванням TLS також.

Таблиця 2.1 – Успішна ініціація ВебСокет з’єднання

HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade **Sec-WebSocket-Accept**:aGFzaCBvZiBrZXkgZnJvbSBzZXJ2ZXI= **Sec-WebSocket-Protocol:** chat

У всіх службах, де шифрування TLS не використовується за замовчуванням або TLS не підтримується, можна вважати це проблемою безпеки. Завжди слід використовувати шифрування.

Щоб мати можливість перевірити проблеми перевірки вводу у ВебСокет сервері, потрібно проаналізувати на наявність типів ВебСокет повідомлень, що надсилаються з клієнта на сервер при звичайній роботі вебпрограми (сюди входять усі повідомлення, які надсилаються з всіма правами користувачів).

Отримавши список повідомлень, що використовуються службою, його потрібно переглянути на потенційно проблемні повідомлення з точки зору перевірки вхідних даних. Теоретично кожне повідомлення може піддатися до ін'єкцій, але через обмеження за часом або обсягом може потрібно відфільтрувати. У таких

випадках пріоритетність повинна бути зроблена для потенційно більш проблемних повідомлень. Наприклад, якщо є повідомлення із вмістом userId = 123, воно має бути хорошою відправною точкою для перевірки перевірки вхідних даних.

Щоб мати можливість перевірити повідомлення, повинен бути доступний список шкідливих корисних навантажень(payloads), які є за замовчування у прокс-серверах для фазингу потенційних проблем при перевірці вхідних даних. Таке тестування повинно містити тести для різного роду вразливостей (наприклад, ін'єкція SQL, XSS, path traversal та подібні).

Для ілюстрації функції повторного надсилання в OWASP ZAP в табі Fuzz, потрібно виділити обране повідомлення та правою кнопкою миші обрати Fuzz, як зображено на рисунку нижче (рис.2.5).

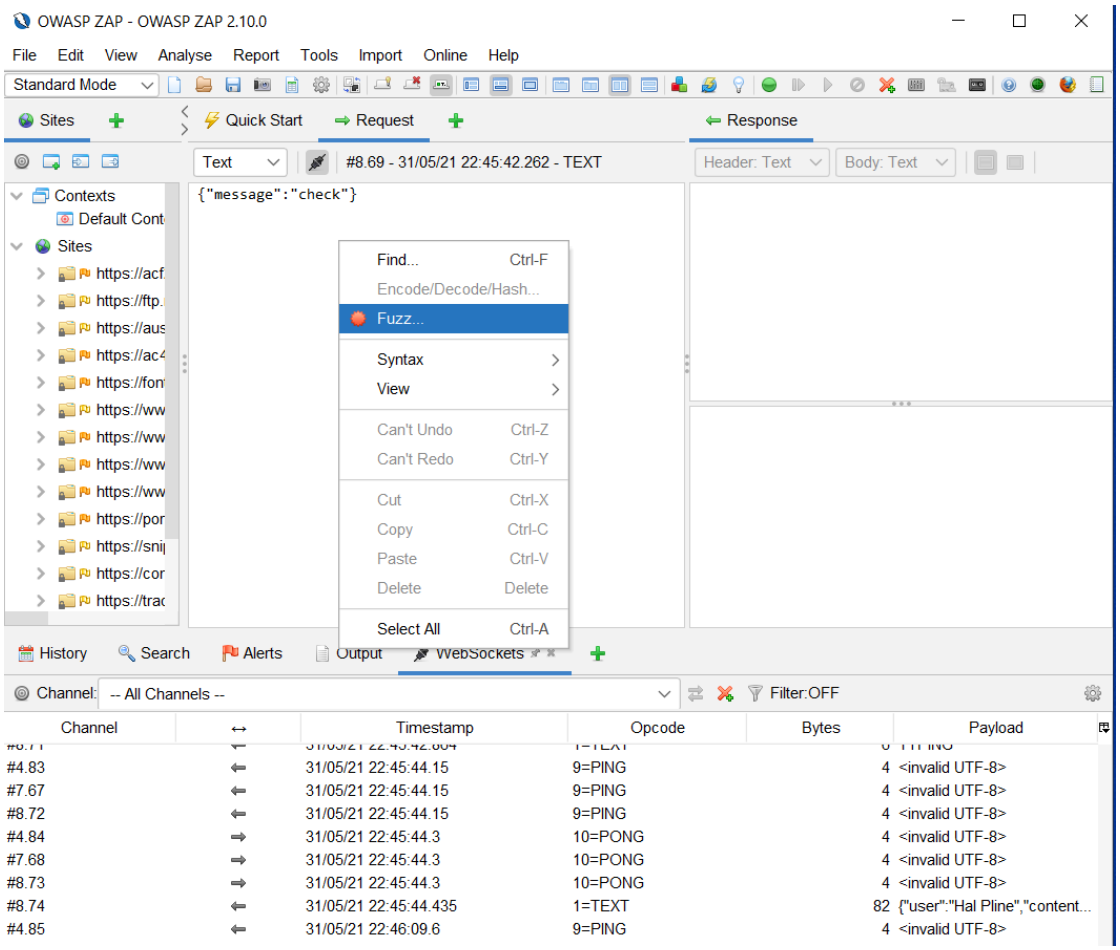


Рисунок 2.5 - Функція повторного надсилання повідомлення WebSocket в проксі-сервері OWASP Zed Attack.

Результат відправленого повідомлення повинен бути перевірений з історії ВебСокет повідомлень на потенційно проблемну відповідь. Якщо відповідь на

надіслане повідомлення повертає повідомлення про помилку SQL, воно може бути ін’єкційним для SQL, і вимагатиме додаткового тестування, щоб перевірити, чи це справді так.

Сценарії перевірки вхідних даних з різними корисними навантаженнями вимагають більш глибоких досліджень різного роду вразливостей, як їх можна перевірити та як вони можуть вплинути на систему. Не ефективно описувати всі можливі корисні навантаження для кожного типу вразливостей у цьому дослідженні, оскільки їх кількість величезна, і вони є також сильно залежать від контексту.

Щоб мати можливість протестувати проблеми з вичерпанням ресурсів у ВебСокет сервісі, необхідно встановити кілька ВебСокет з’єднань з метою проведення тестування. Спочатку потрібно протестувати, якщо активне ВебСокет з'єднання обмежуються одним підключенням за сеанс чи ні.

Використовуючи веб-браузер та JavaScript можна легко перевірити відкриття одного і того ж сервісу для кількох вкладок та перевірка, якщо він використовується у кожній вкладці.

Автоматизоване тестування

Автоматизація тестування потрібна для покращення процесу тестування, та для первірки більшої частини вебпрограми та її функцій. При монотонному тестування, автоматизоване тестування зберігає зусилля витрачені на однотипне тестування різних гілок вебпрограми чи абсолютно інших вебпрограм, де імплементація ВебСокетів є однаковою або з малою відмінністю.

Майже весь процес мануального тестування може бути автоматизований, шляхом написання скриптів.

Для автоматизації пошуку чутливої інформації та вразливостей, що пов’язані з заголовками запиту, потрібно розширювати функціонал скриптів OWASP ZAP для пасивного тестування ВебСокет повідомлень.

Щоб автоматизувати процес тестування ін’єкції даних, потрібно використовувати список корисних навантажень, за допомогою інструменту Fuzz OWASP ZAP із індивідуальним списком корисного навантаження, та перевірку відображення у відповідях чи на сторінці вебпрограми.

Процес неодноразового відкриття ВебСокет сесії для тестування виснаження сервера може бути автоматизованим додатковим скриптом, що працюватиме незалежно від OWASP ZAP.

Щоб наглядно зрозуміти різницю мануального тестування та автоматизованого було зображено на рисунку нижче(рис.2.6) відмінність процесу автоматизованого та ручного тестування.

Рисунок 2.6 - Процес ручного та автоматизованого тестування

Зусилля тестувальника при автоматизованому тестуванні необхідні для оцінки вебпрограми, написання та оптимізація скрипту та додаткового перегляду результат, так як потрібно враховувати присутність хибно-позитивних попереджень.

Висновок до розділу 2

У цьому розділі було розглянуто вразливості за OWASP10, що можуть бути присутніми у вебпрограмі, що працює по ВебСокет протоколу, та вимагають тестування. Майже всі вразливості з списку OWASP10 зазвичай можуть бути у вебпрограмах, що використовують ВебСокети, які будь-який зловмисник може потенційно використати.

Також було розглянуто два типи тестування цих вразливостей, а саме мануальне тестування, та автоматизоване. Автоматизоване тестування покращує процес тестування шляхом виконання більшого об’єму роботи та циклічної перевірки вебпрограми та її функцій.

3 СТВОРЕННЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ

OWASP ZAP (Zed Attack Proxy) - один з найпопулярніших в світі інструментів безпеки. Він призначений для користувачів з широким спектром досвіду в області безпеки, тому відмінно підходить для написання власних скриптів.

ZAP створює проксі-сервер між клієнтом і вразливим сайтом. Під час переміщення по веб-сайту, ZAP фіксує всі дії, а потім атакує сайт відомими методами.

З допомогою ZAP були написані скрипти, для автоматизованого аналізу вебпрограм, що працюють по ВебСокет протоколу.

ZAP виконує роботу людини в середині(Man-in-the-middle), тобто не лише дозволяє переглядати запити, а й модифікувати запити, шляхом видалення чи додавання необхідних параметрів чи заголовків. ZAP містить три важливі складові програмного сканування(рис.3.1): павук(spider), пасивний сканер (passive scanner) і активний сканер (active scanner).

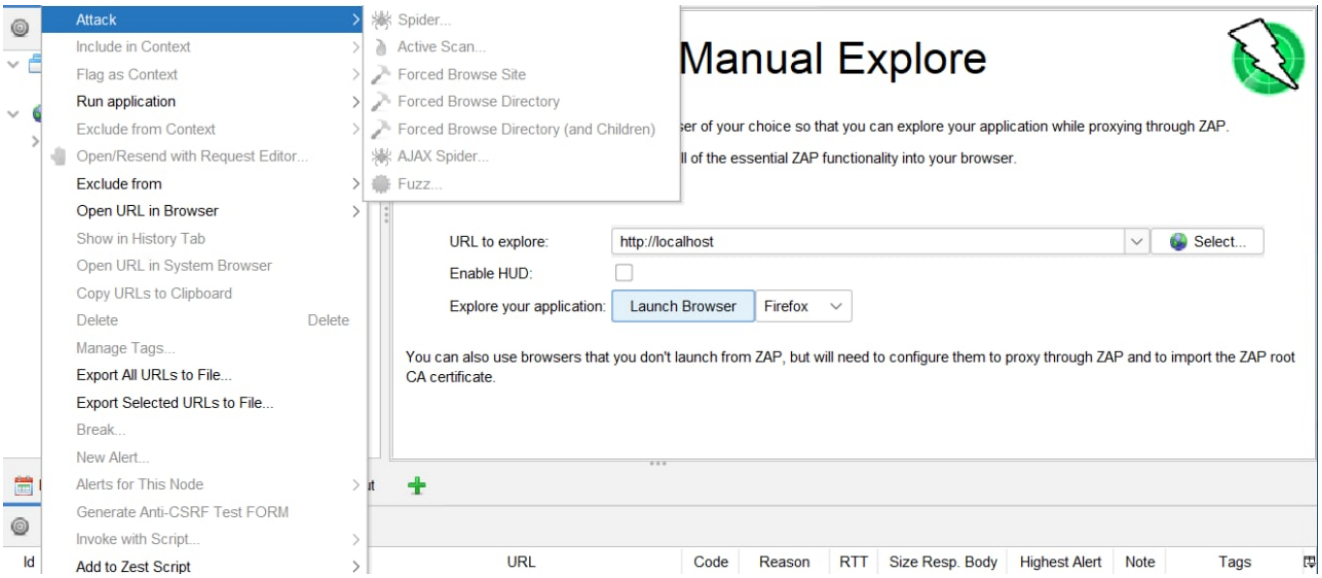


Рисунок 3.1 - Складові сканування веб-програми

Павук надає можливість отримати додаткову інформацію за рахунок переходу за посиланнями та шляхом заповнення форм. Пассивний сканер лише аналізує уже надіслані запити та отримані відповіді, тобто не має жодного шкідливого впливу на програму. Однак активний сканер навпаки шляхом заповнення

небезпечним змістом різних параметрів та заголовків модифікує HTTP запити та надсилає їх на сервер.

3.1 Мови скриптів, що підтримуються в ZAP

Для початку роботи з Zed Attack Proxy (ZAP) потрібно скачати та правильно налаштувати Java залежності та роботу сертифікатів в браузері.

ZAP містить вбудовані скрипти та розширення. Однак є також можливість написання кастомних скриптів, які виконуватимуть вузьконаправлені завдання. Є чотири мови, які підтримуються ZAP для написання скриптів: JavaScript, Zest, Jython і Ruby. Наступним завданням є вибір оптимального варіанту мови скрипту.

Zest – це експериментальна мова скриптів, яка записується графічно. Вбудованою в ZAP Zest мовою скрипти можуть бути створені з допомогою шаблонів, обираючи і додаючи запити, дублікуючи вже існуючі скрипти чи шляхом запису(зображено на рисунку). Мова Zest містить цикли, умови, призначення та навіть запуск браузера.

JavaScript – це також мова скриптів, яка наявна в ZAP за замовчування. Скрипти на мові JavaScript широко використовуються в усіх браузерах (а також у Firefox). Тобто плюсом такої мови є простота в написанні. ZAP за замовчуванням містить шаблони написані на JavaScript.

Jython – це мова скриптів, яка вимагає додаткового скачування розширень, які пропонує опенсорсний ZAP. Jython містить в собі потужність мови Java та легкість написанння мови Python. ZAP не надає шаблони сканування ВебСокетів на мові Jython, однак потрібно виділити час на написання власного скрипту для неодноразового використання скрипту в різних веб програмах. Процес встановлення розширення для написання скриптів на мові Jython зображено в додатку Б.

3.2 Типи скриптів

Залежно від функції використання скрипту ZAP поділяє їх на різні типи описані в таблиці нижче(таблиця 3.1).

Таблиця 3.1 - Типи скриптів, що підтримує ZAP

Назва	Виклик
Stand Alone	Лише шляхом мануального запуску
Targeted	Для окремих запитів(через обмеження URL)
Proxy	Для всіх запитів, що проксіюються
HTTP Sender	Для всіх запитів
Active Scan	Під час активного скану
Authentication	Під час запитів, які вимагають автентифікації
Passive Scan	Під час пасивного скану
Fuzzer Websocket Processor	Під час фазингу ВебСокетів
Websocket Sender	Під час кожного надсилання ВебСокетів

Для тестування ВебСокетів було написано скрипти, що відносяться до Passive Scan(Пасивне сканування), WebSocket Sender(Надсилання ВебСокетів), HTTP Sender та WebSocket Fuzzer(Одночасне надсилання ВебСокет повідомлень).

Кожен із скриптів має свої нюанси, що потрібно було враховувати під час написання скрипту. Що відноситься до WebSocket Sender, спочатку було потрібно клонувати запит msg = msg.cloneRequest(); і лише потім змінювати параметри та пересилати унікальний запит, як новий до веб-сервера.

Також було досліджено, що навіть коли ZAP містить вбудований генератор пейлоадів для фазингу, знаючи структуру кастомного пейлоаду для генерації

запитів можна створити власний генератор шкідливих навантажень в **payload generator** табі.

Щоб працювати з скриптами в ZAP потрібно відкрити нову табу Scripts, як зображено на рисунку нижче(рис.3.2) та писати скрипти відповідно до їх типу в окремих розділах.

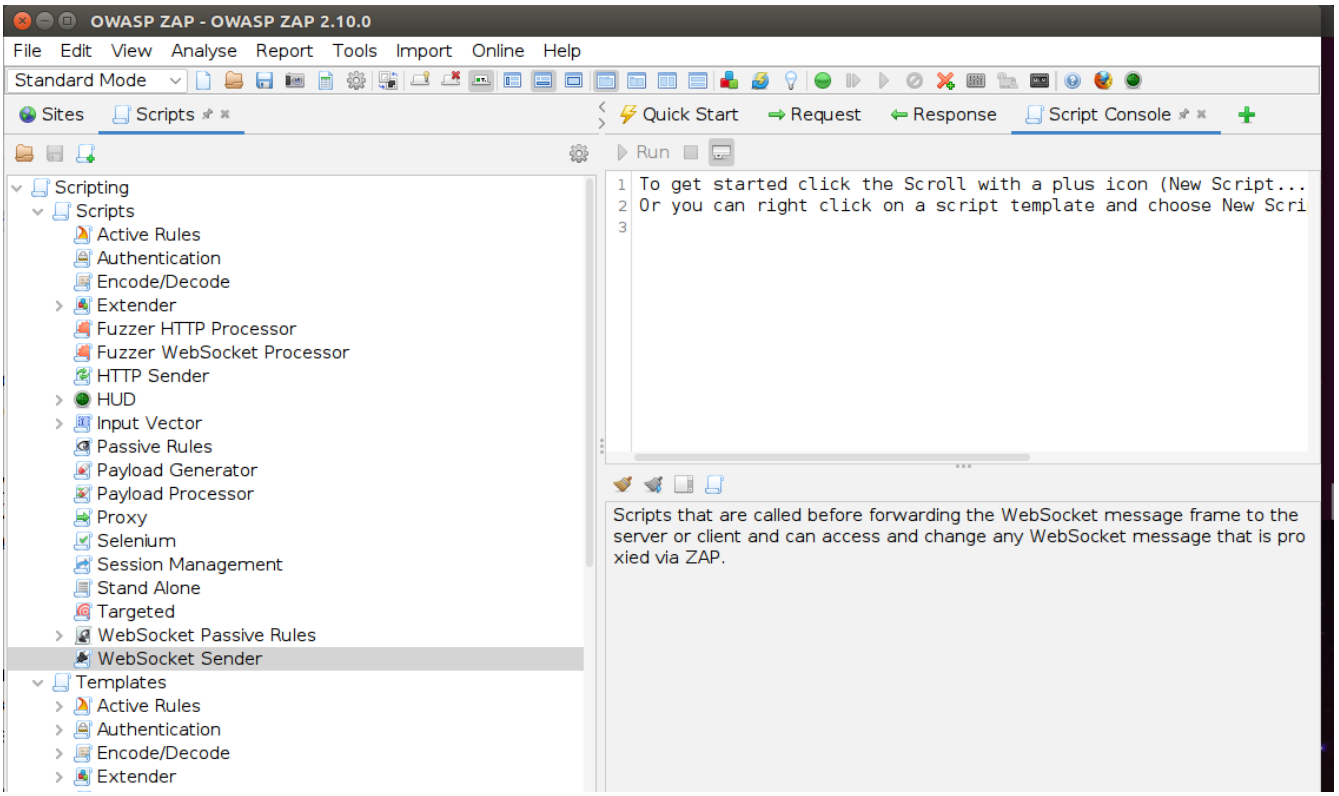


Рисунок 3.2 – Структура скриптів у ZAP

Скрипти для сканування Вебсокетів

Отже завданням дипломної роботи було розробити кастомні скрипти для роботи автоматизованого аналізу ВебСокетів, а саме детальним вивченням вказаних типів написання скрипту.

Для написання скриптів для тестування вебпрограм, що працюють по ВебСокет протоколу, було вивчено відмінність типів скриптів. Скрипти WebSocket Sender викликаються перед пересиланням ВебСокет повідомлень на сервер або клієнт, і можуть отримати доступ до будь-якого повідомлення ВебСокет, що проксіюється через ZAP, і змінити його.

Скрипти Fuzzer WebSocket Processorвикликаються перед тим, як переслати декілька повідомлень ВебСокет одночасно на клієнт чи сервер. Для кожного повідомлення використовується корисне навантаженняz з підготовленого списку і, як зображено на рисунку нижче(рис 3.3), почергово перевіряється на можливий вплив на вебпрограму.

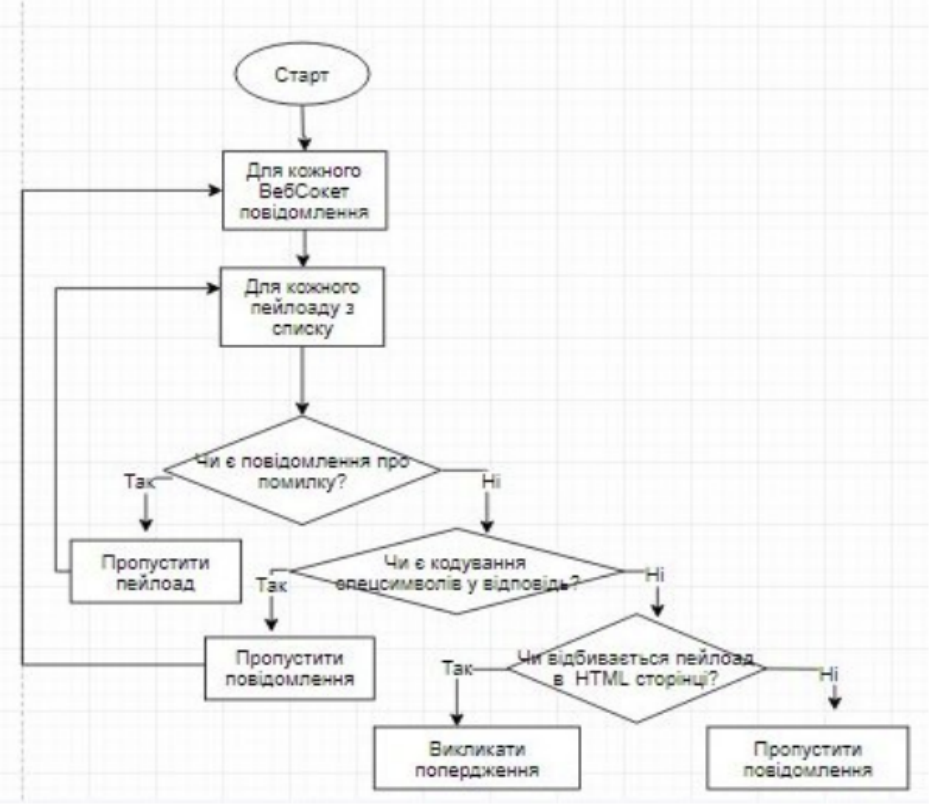


Рисунок 3.3 - Блок-схема для активного скрипту

Скрипти пасивного сканування ВебСокет повідомлень викликаються, як зображено на рисунку нижче(рис.3.4), кожного разу, коли ВебСокет повідомлення передається через з'єднання ВебСокет. Скрипти можуть отримувати доступ до повідомлень ВебСокет, щоб перевірити присутність певної інформації та викликати попередження, однак не мають жодного навантаження на веб-сервер.

Рисунок 3.4 - Блок-схема пасивного скрипту

3.3 Скрипти для пасивного сканування

За замовчуванням ZAP містить деякі сценарії пасивного сканування ВебСокет протоколу. Сценарії за замовчуванням пасивно сканують уже надіслані ВебСокет повідомлення, щоб розкрити інформацію, яка може бути корисною для користувача під час тестування веб-програми, та в разі виявлення такої.

При написанні пасивного скрипту, щоб ідентифікувати витік чутливої інформації потрібно вказати тип регулярного виразу в скрипті на такий, що буде відповідати потрібному формату інформації(банківський рахунок, ідентифікатор студента КПІ чи будь-який тип національного номеру - паспорт, ІПН).

Після запуску скрипту, якщо на іншій сторінці, чи на іншій http відповіді буде знайдений інший варіант тексту, що відповідає регулярному виразу, інше сповіщення буде зображене, тому що параметр оповіщення(evidence) відрізняється.

Виконуючи скрипт пасивного сканування, якщо в ВебСокет повідомленнях, що передаються, буде знайдений варіант тексту, що відповідає регулярному виразу, сповіщення буде зображене в окремій панелі ZAP.

При автоматизованому тестуванні потрібно враховувати присутність хибно-позитивних попереджень. Щоб детальніше зрозуміти значення хибно-позитивних попереджень потрібно розглянути два різні вебсокет

повідомлення, які зображені на рисунку нижче(рис.3.5), де тривіально відображається як регулярний вираз реагує лише по шаблону, і після роботи скрипту потрібно додатково перевірити, що всі попередження, які спрацювали є реальними. Тому що лише одне із повідомлень містить електронну пошту, що є чутливою інформацією.

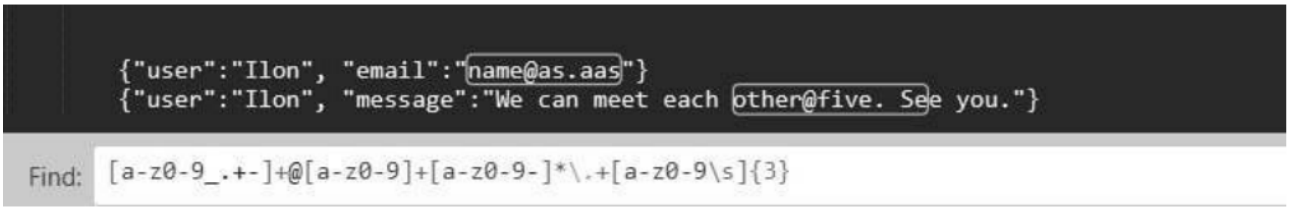


Рисунок 3.5 - Приклад хибно-позитивного попередження

Чутлива інформація

Чутлива інформація включає конфіденційні дані про особисту інформацію (ІПІ), такі як медичні записи, облікові дані, особисті дані та кредитні картки, які часто потребують захисту.

Щоб створити успішний скрипт пасивного сканування недостатньо володіти мовою JavaScript та Zed Attack Proxy. Одним з ключовим моментів було розвинути здібності в написанні регулярних виразів для побудови необхідних шаблонів пошуку інформації.

Прикладом чутливої інформації був використаний ідентифікатор студента Київського Політехнічного інституту, та побудований регулярний вираз для автоматизованого пошуку з допомогою скрипта, який зображений нижче(рис.3.6), ідентифікатора по ВебСокета повідомленнях, що проходять через проксі-сервер.

```
Run  ECMAScript : Student_ID.js
7  CONFIDENCE_HIGH = 3;
8
9  var student_ID_Regex = new RegExp("([A-Z]{2}[0-9]{8})", 'gmi');
10
11 function scan(helper,msg) {
12
13     if(msg.opcode != OPCODE_TEXT || msg.isOutgoing){
14         return;
15     }
16     var message = String(msg.getReadablePayload());
17     var matches;
18
19     if((matches = message.match(student_ID_Regex)) != null) {
20         matches.forEach(function(evidence){
21             helper.newAlert()
22                 .setRiskConfidence(RISK_INFO, CONFIDENCE_HIGH)
23                 .setName("Student ID was found in WebSocket message (script).")
24                 .setDescription("Student ID  was found in a WebSocket Message.")
25                 .setSolution("Remove Student ID from public access.")
26                 .setEvidence(evidence)
27                 .setCweId(200) //Information Exposure
28                 .setWascId(13) // Information Leakage
29                 .raise();
30         });
```

Рисунок 3.6 - Фрагмент скрипту для пасивного сканування ВебСокетів

Успішний результат роботи скрипти був зображений в табі попереджень(рис.3.7) з підтвердженням знайденого ідентифікатора, та деталями вразливості вказаними в скрипті.

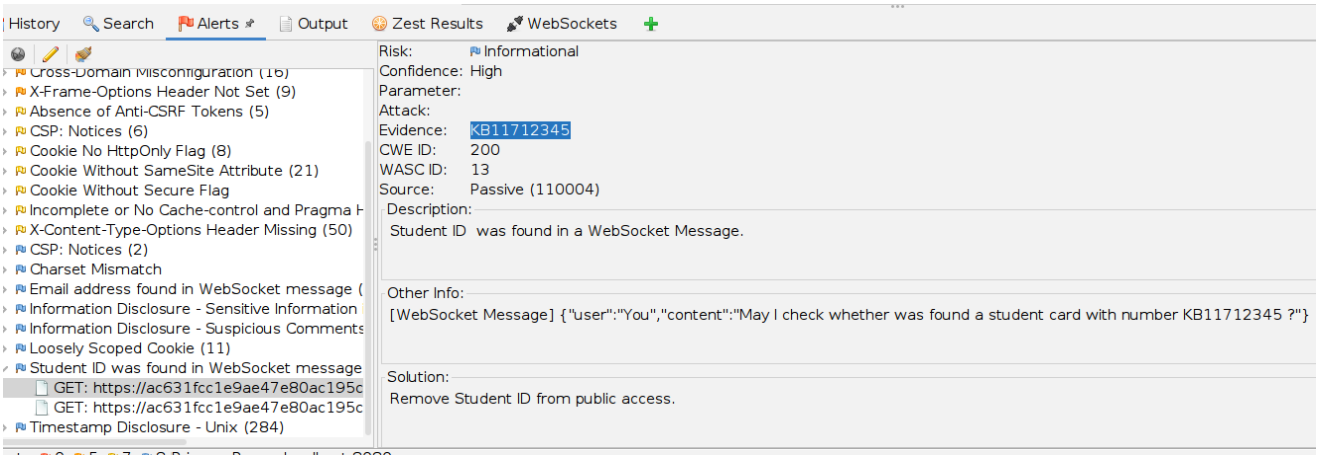


Рисунок 3.7 - Результат роботи пасивного скрипту

Для роботи скрипту потрібно встановити ZAP, додати скрипт в папку скриптів розділ пасивне сканування, зберегти скрипт так увімкнути. Під час роботи

вебпрограми, що використовує ВебСокети, в разі виявлення вразливості сповіщення буде зображене в табі Alerts.

SSL в ВебСокетах

ВебСокет з’єднання може ініціюватися як через HTTP так і через HTTPS. Однак якщо сторінка є доступною через HTTP - ВебСокет сесія може використовувати WS чи WSS(Безпечний ВебСокет - ВебСокет через TLS). Однак, коли сторінка завантажується через HTTPS - лише WSS може використовуватися, тому що браузер не дозволяє зменшити рівень безпеки.

ВебСокет зв'язок через TLS налаштований тоді, коли включений словник TLS із (обов’язковим) ключем атрибутів та сертифікатом(рис.3.8). Ключ вказує на файл приватного ключа сервера (формат PEM, без парольної фрази), а сертифікат вказує на файл сертифіката сервера (формат PEM).

```
{
  "type": "websocket",
  "endpoint": {
    "type": "tcp",
    "port": 443,
    "tls": {
      "key": "server_key.pem",
      "certificate": "server_cert.pem"
    }
  },
  "url": "wss://example.com"
}
```

Рисунок 3.8 - Приклад налаштування ВебСокетів через TLS

У вебпрограмах, де використовуються конструктор JavaScript для ініціалізації ВебСокет протоколу, саме вказуючи протокол(рис.3.9) визначається чи будуть іти повідомлення через TLS. Після виконання функції, exampleSocket.readyState буде мати значення CONNECTING. readyState змінюється на OPEN як тільки з'єднання станет готовим до передачі даних.

```
var exampleSocket = new WebSocket("ws://www.example.com/socketserver", "protocolOne");
```

Рисунок 3.9 - Використання конструктуру JS для ініціалізації ВебСокету

Коли з'єднання встановлено (що відповідає, readyState OPEN), exampleSocket.protocol повідомляє, який протокол вибрав сервер.

У прикладі WS заміняє HTTP, так само WSS замінює HTTPS. Встановлення ВебСокет протоколу покладається на механізм оновлення HTTP(Upgrade), тому запит на оновлення протоколу є неявним, коли звертаються до сервера HTTP як **ws:** //www.example.com або **wss:** //www.example.com.

Підключення WebSocket можна встановити лише до URI, які відповідають цій схемі. Тобто, якщо в вебпрограмі є URI зі схемою ws: // (або wss: //), то і клієнт, і сервер слідують протоколу з'єднання WebSocket, щоб відповідати специфікації WebSocket.

Щоб автоматизувати тестування наявності шифрування був написаний скрипт(рис.3.10), що змінює HTTP на HTTPS в запиті рукостискання ВебСокетів, та при не успішній відповіді повертає попередження про непідтримку шифрування.

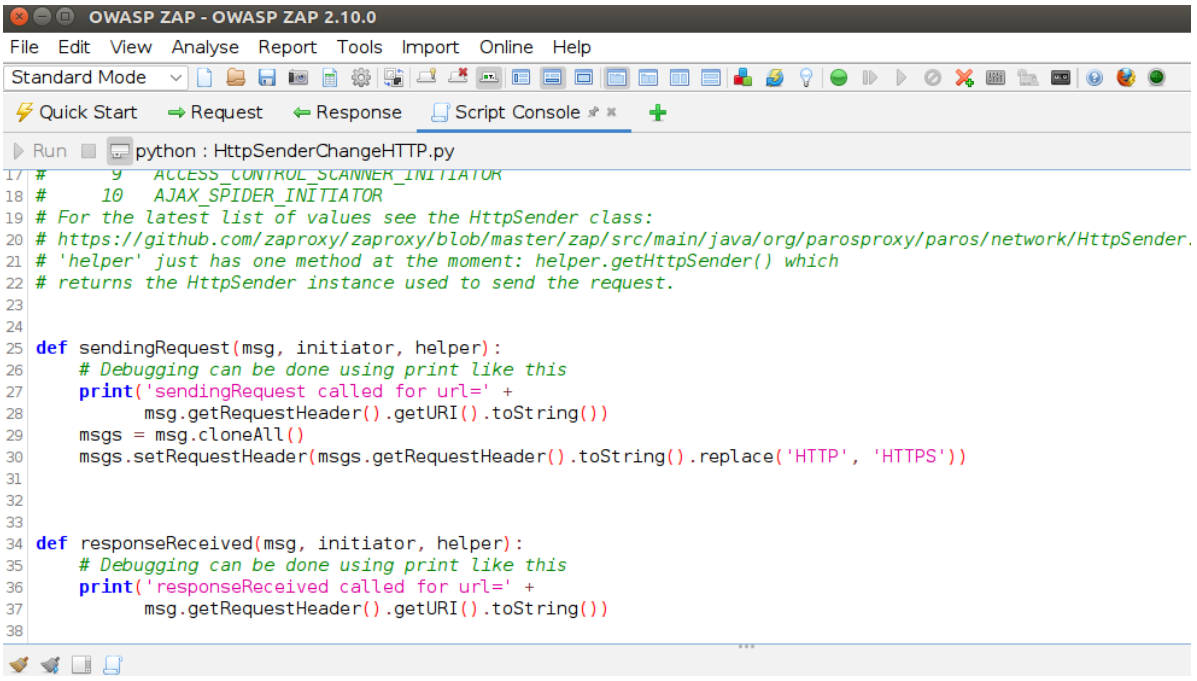


Рисунок 3.10 – Скрипт для перевірки підтримки шифрування

Для роботи скрипту потрібно встановити ZAP, додати скрипт в папку скриптів розділ HTTP Sender, зберегти скрипт так увімкнути. Під час роботи вебпрограми, що використовує ВебСокети, в разі виявлення вразливості сповіщення буде зображене в табі Alerts.

Заголовок джерела

Стандарт ВебСокет протоколу визначає поле заголовка Джерела(Origin), яке веб-браузери встановлюють за URL-адресою, що ініціює запит ВебСокет. Це може бути використано для розмежування між підключеннями від різних хостів або між підключеними через браузер та іншим видом мережевого клієнта.

Можна розглядати заголовок Джерела як приблизно аналогічний заголовку X-Request-With, який використовується запитами AJAX. Веб-браузери надсилають заголовок X-Request-With: XMLHttpRequest, за допомогою якого можна розрізнити запити AJAX, зроблені браузером, та ті, що зроблені безпосередньо. Однак цей заголовок легко встановлюється клієнтами, що не переглядають браузер, і тому він не є надійним джерелом автентифікації.

Таким же чином можна використовувати заголовок Джерела як дорадчий механізм, який допомагає диференціювати ВебСокет запити з різних місцеположень та хостів, але не слід покладатися на нього як на джерело автентифікації.

Перевірка заголовка Джерела перешкоджає використанню ВебСокетів іншим веб-сайтом, який користувач також відвідує (наприклад, для отримання даних). Тому був написаний скрипт(рис.3.11), що в запиті на рукостискання перевіряє наявність заголовку Джерела.

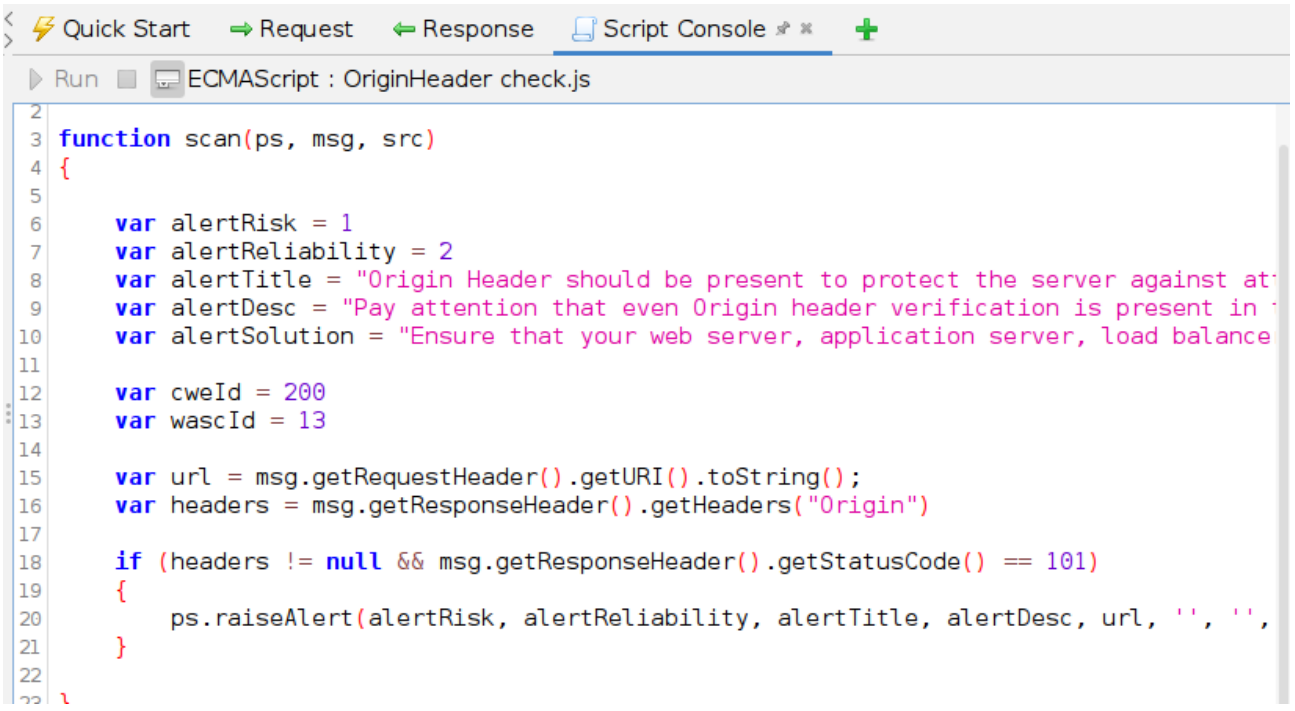


Рисунок 3.11 - Скрипт перевірки заголовку Джерела

ВебСокет протокол не передбачає обмеження політикою того самого походження(SOP).Політика дозволяє скриптам, які працюють на сторінках, що згенеровані на одному сайті, отримати доступ до методів та атрибутів один одного без особливих обмежень, але забороняє доступ до більшості методів та властивостей на сторінках інших сайтів.[7]

Це пов’язано з тим, що запит на оновлення протоколу матиме доступ до файлів cookie користувача, тому, якщо не перевіряється походження, запит може бути зроблений з somemalicioussite.com, а не з сайту safesite.com.

Коли є ВебСокет сервіс, який повертає деякі приватні дані, за допомгою шкідливого сайту, який користувач відкриє, можна зчитати дані з вебпрограми, оскільки користувач ввійшов у систему.

Регулярні вирази

Регулярний вираз (іноді називають раціональним виразом) - це послідовний набір символів, що визначають шаблон пошуку, головним чином для використання у збігу шаблонів із рядками або збігу рядків, тобто операцій, подібних до "пошуку та заміни".

Регулярні вирази - це узагальнений спосіб узгодження зразків із послідовностями символів. Він використовується в усіх мовах програмування, таких як C ++, Java та Python.

Таблиця 3.2 - Регулярний вираз для пошуку адреси електронної пошти:

^ ([a-zA-Z0-9 _ \ - \.] +) @ ([a-zA-Z0-9 _ \ - \.] +) \. ([a-zA-Z0-9] {2,5}) $

Зазначений в таблиці вище(таблиця 3.2) регулярний вираз можна використовувати для перевірки, чи є заданий набір символів адресою електронної пошти чи ні. Щоб правильно писати регулярні вирази, потрібно знати певні правила, як і в кожній мові.

Повторювачі (*, + та {}) - це символи, що виконують роль повторювачів і повідомляють програмі, що попередній символ слід використовувати не лише один раз. Символ зірочки (*) повідомляє програмі відповідати попередньому символу (або набору символів) 0 або більше разів (до нескінченності).[8]

Символ плюс (+) повідомляє комп'ютеру повторити попередній символ (або набір символів) принаймні один або кілька разів (до нескінченності).

Фігурні дужки {…} повідомляють комп'ютеру повторювати попередній символ (або набір символів) стільки разів, скільки значень у цій дужці. {7} означає, що попередній символ слід повторити 7 разів, {min,} означає, що попередній символ відповідає min або більше разів. {min, max} означає, що попередній символ повторюється в мінімум min і максимум max разів.

Узагальнюючий знак(.), а саме крапковий символ може мати місце будь-якого іншого символу, саме тому він може називатися символом підстановки. Необов’язковий символ (?) повідомляє комп'ютеру, що попередній символ може або може не бути в рядку, що підлягає зіставленню. Записаний формат файлу документа як - “docx?”, де знак «?» повідомляє комп’ютеру, що x може бути, а може і не бути присутній у назві формату файлу.

Символ каретки (^) означає встановлення позиції для збігу і повідомляє комп'ютеру, що збіг повинен починатися на початку рядка або рядка.

Символ долара ($) повідомляє комп'ютеру, що збіг має відбуватися в кінці рядка або перед \n в кінці рядка або рядка.

3.4 Скрипти для ін’єкції вхідних даних

Атаки ін’єкції вхідних даних

У багатьох випадках правильне кодування може знешкодити атаки, які покладаються на відсутність перевірки вхідних даних. У разі введення HTML, якщо використовується кодування сутності HTML, перш ніж воно буде відправлене до браузера, це запобіжить більшості атак введення HTML (наприклад, скриптинг між сайтами [XSS]). Кодування сутності HTML означає заміну ASCII символи з їхніми еквівалентами "HTML Entity" (наприклад, замінюючи символ "<" на "& Lt;"). [4]

Однак для перевірки того, чи є введений захист від шкідливого вводу даних був написаний скрипт (рис.3.12), що надсилає ВебСокет запит з уже записаним шкідливим навантаженням замість існуючого параметра. В роботі був використаний JavaScript тег картинки з активною подією, що спрацьовує при неправильному посиланні.

```
function onMessageFrame(msg, helper){
    var msg = new WebSocketMessageDTO()
    msg.isOutgoing = true
    msg.opcode = WebSocketMessage.OPCODE_TEXT
    msg.payload = "<img src onerror=alert(1)>"

    proxy = getWebSocketProxy.invoke(ext, helper.getChannelId())
    proxy.send(msg)
```

Рисунок 3.12 - Скрипт з шкідливим навантаженням XSS

Результат роботи(рис.3.13) був перевірений на сторінці вебпрограми, однак це можна було зробити використовуючи табу пошуку із обмеженням на відповіді від сервера.

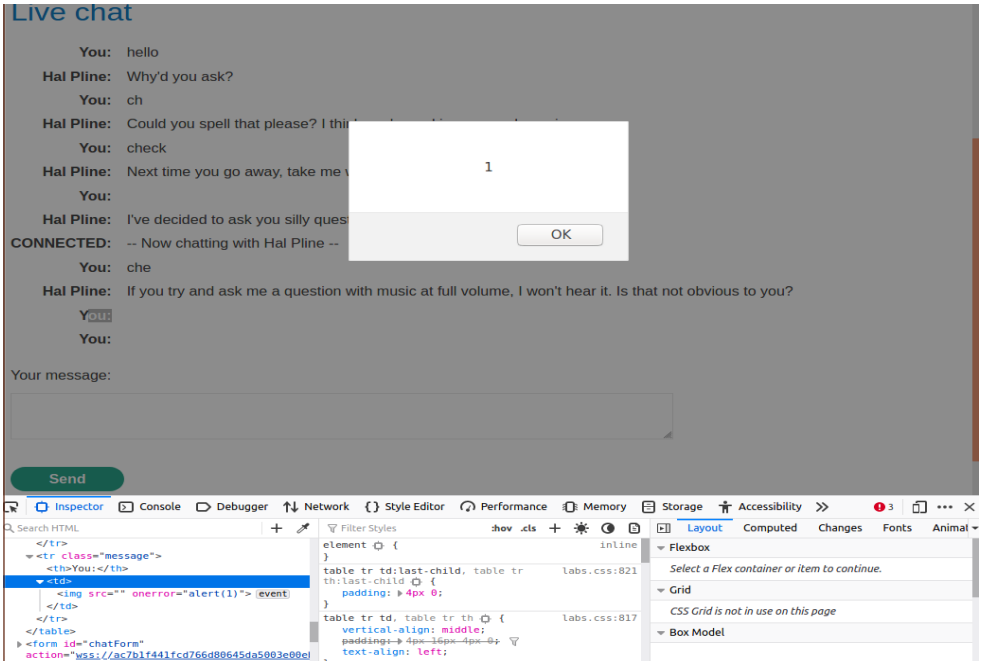


Рисунок 3.13 – Робота JavaScript вставленого з клієнтської сторони

Напівавтоматичне тестування

Написання скрипту для перевірки атак ін’єкцій вхідних даних не завжди буде оптимальним для тестування, тому під час роботи над дипломом було використано функціонал ZAP для автоматизації тестування шляхом надсилання ВебСокет повідомлень з попередньо зібраним списком, що містить шкідливе навантаження.

Протягом дослідження було написано скрипт, що містить мінімальну кількість змінних, але включає всі типи вразливостей, що відносяться до атаки ін’єкцій.

Процес тестування було автоматизовано за допомогою інструменту Fuzz (рис.3.14) OWASP ZAP із індивідуальним списком корисного навантаження, та перевірку відображення у відповідях чи на сторінці вебпрограми. [9]

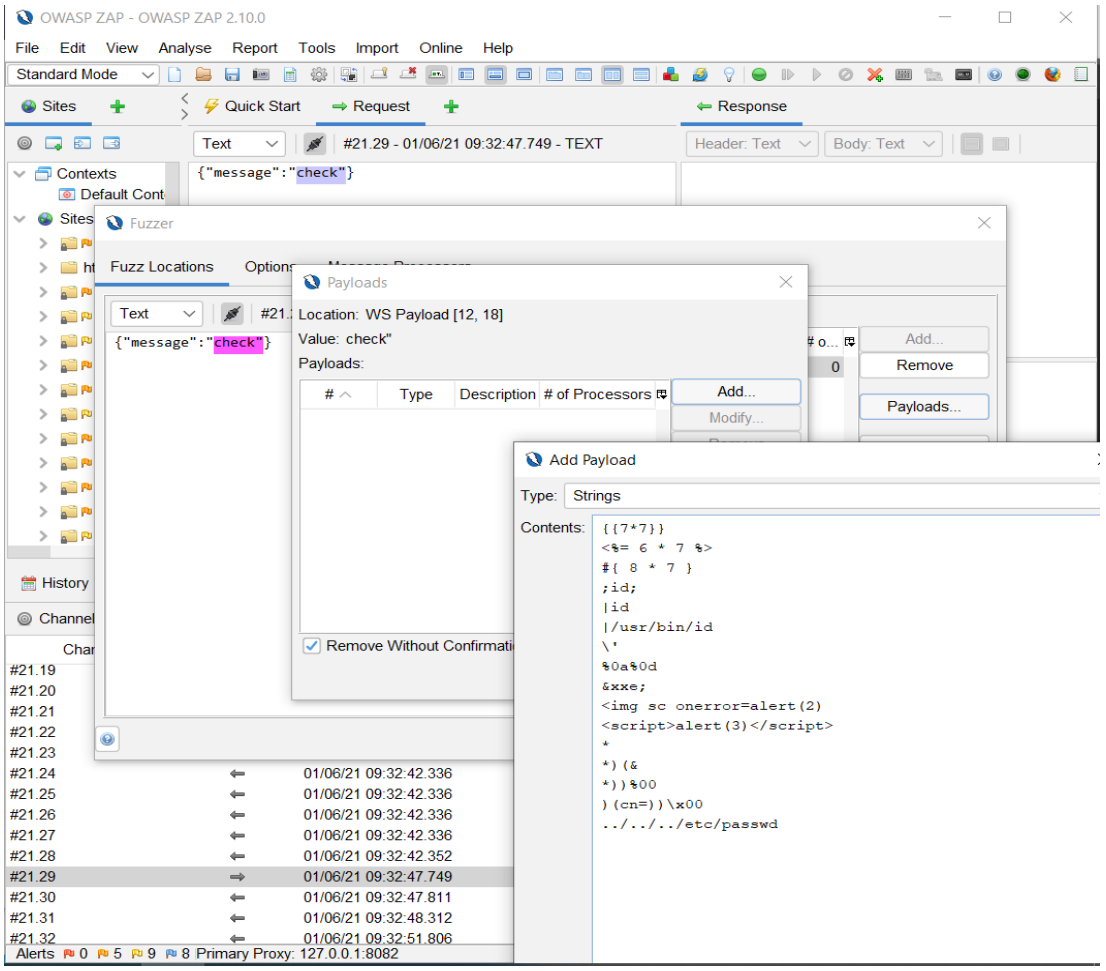


Рисунок 3.14 - Програма автоматизації надсилання ВебСокет повідомлень в ZAP

Завдяки такому типу автоматизації було визначено, що програма, що тестувалася не вразлива до шаблонної ін’єкції, SQLi, командної ін’єкції, LDAP ін’єкції, і на прикладі є вразливою(рис.3.15) до JavaScript ін’єкції(XSS).

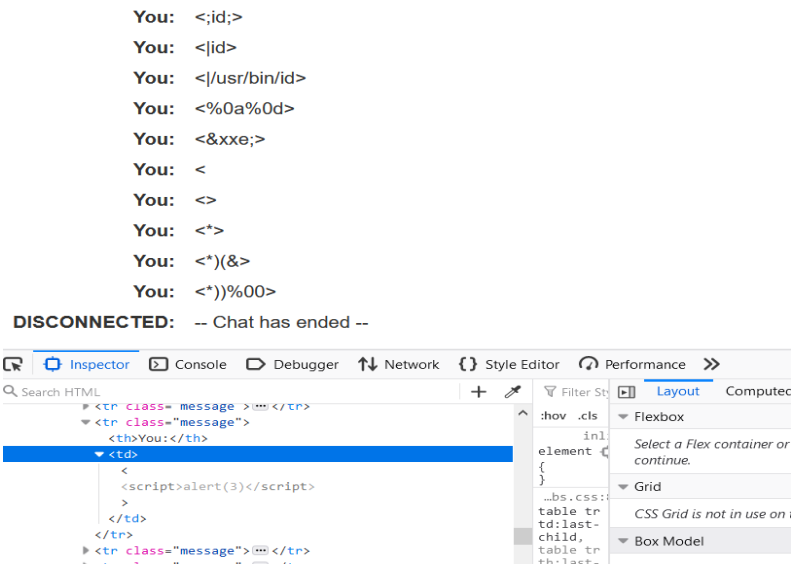


Рисунок 3.15 - Результат роботи ін’єкції із створеним списком як шкідливе навантаження

3.5 Виклик попередження з різними аргументами

Попередження(alert) - це не лише результат успішної роботи скрипту, а й зображені деталі знайденої вразливості для інших тестувальників та веб-розробників. Попередження має містити підтвердження знайденої вразливості, опис вразливості, можливі шляхи усунення та ще додаткові деталі, що можуть бути важливими.

Залежно від типу скрипту було детально вивчено методи виклику сповіщення. Використовуючи інформацію про класи в ZAP було знайдено(рис.3.16) опис компоненти пасивного сканування, та переглянуто детально метод виклику сповіщення.

Constructors

Constructor and Description
ScriptsPassiveScanner()

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	addTag(String tag)
boolean	appliesToHistoryType(int historyType)
String	getName()
int	getPluginId() Returns the ID of the plug-in.
void	raiseAlert(int risk, int confidence, String name, String description, String uri, String param, String attack, String otherInfo, String solution, String evidence, int cweId, int wascId, HttpMessage msg)
void	scanHttpRequestSend(HttpMessage msg, int id)
void	scanHttpResponseReceive(HttpMessage msg, int id, net.htmlparser.jericho.Source source)
void	setParent(PassiveScanThread parent)

Рисунок 3.16 - Методи ZAP для пасивного скану

Параметр risk є integer(цілі числа),та використовується для сортування знайдений вразливостей по ризику, тому потрібно використовувати правильну інформацію, яка цифра відповідає певному значення ризику. Використовуючи пошук по константним змінних знаходимо таблицю відповідностей ризиків(Додаток В). [10]

Залежно від скрипту були використані різні параметри, що відносяться до методу виклику сповіщення. Для параметру Evidence, який відповідає за доказ, зазвичай використовувалося шкідливе навантаження,що використовувалася успішно при запуску скрипту, або ж знайдена інформація в повідомленні за регулярним виразом.[11]

Висновок до розділу 3

У цьому розділі було розглянуто та використано функціонал ZAP для написання скриптів для автоматизованого тестування вебпрограм, що працюють по ВебСокет протоколу. Було вибрано мови для написання скриптів з тих, що підтримує ZAP з урахуванням гнучкості мови та легкості написання скриптів.

Для тестування ВебСокетів було обрано окремі типи скриптів, які виконують різні функції та мають певний шаблон, залежно від вразливостей, що тестувалися. Результати скриптів були додані в розділ, як і додаткові деталі щодо написання скриптів, такі як регулярні вирази та виклик сповіщень.

4 АНАЛІЗ ОТРИМАНОГО РЕЗУЛЬТАТУ

В результаті роботи над написанням скриптів для тестування ВебСокет повідомлень було успішно отримано сповіщення про присутність вразливостей чи відображення вставленого шкідливого навантаження на сторінках вебпрограми.

Тобто були виконані правильно всі пункти поставленого завдання, а саме ознайомлення з середовищем для тестування; вивчення Вебсокет протоколу; дослідження вразливостей в ВебСокет протоколі; вивчення типів тестування вразливостей в ВебСокет протоколі; дослідження вебпрограм, які працюють по ВебСокет протоколу; вивчення типів скриптів для аналізу ВебСокетів в ZAP; написання скриптів для аналізу ВебСокетів;

4.1 Порівняння ефективності різних типів тестування

Під час дослідження було розглянуто два типи тестування ВебСокет протоколу на вразливості з двох причин, а саме для більш детального дослідження вразливостей присутніх у вебпрограмах, що працюють через ВебСокет повідомлення та для порівняння ефективності.

Для розуміння міри порівняння ефективність тестування була визначена як залежність знайдених вразливостей від витраченого часу. Для цього було зображено графік залежності(рис.4.1), побудований на даних зібраних під час тестування двома методами одні і ті ж вебпрограми.

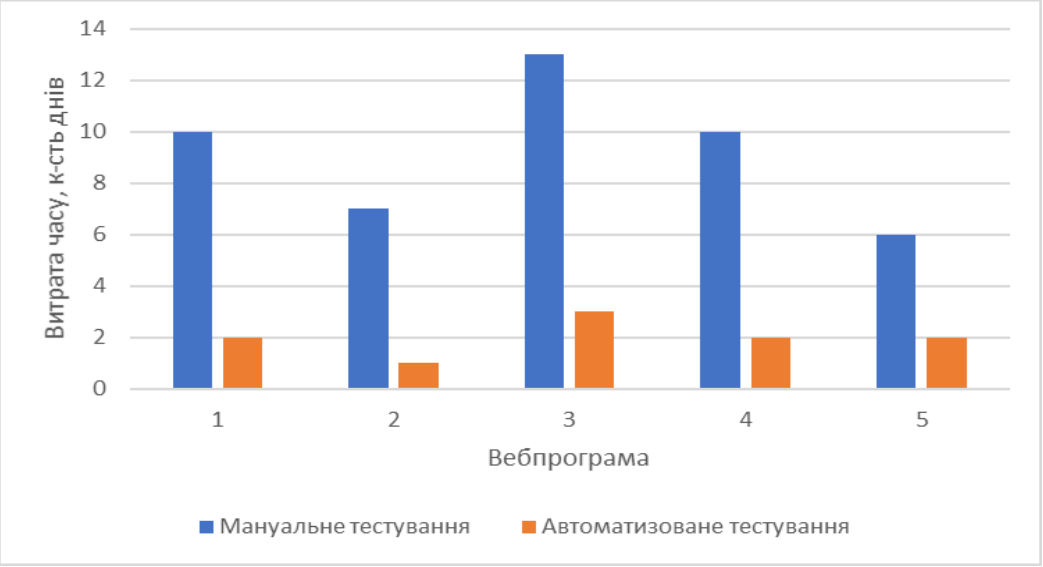


Рисунок 4.1 - Витрата часу на тестування вебпрограми при різному типі тестування

Для побудови такого типу залежності було використано п’ятеро вебпрограм наданих компанією на практиці. Витрати часу при кожному з тестів могла бути не точним значенням, тому що при кожному з тестів була задіяна людина, яка може використовувати час не раціонально. Кількість днів вказаних в репортах містить лише робочі восьмигодинні дні.

В результаті, при мануальному тестуванні ВебСокетів витрата часу йде більша, тоді коли серед отриманого результату також є хибно позитивні знайдені вразливості, тому кількість знайдених вразливостей зменшується. При автоматизованому тестування з допомогою написаних скриптів, не враховуючи час на написання скриптів та налаштування програмного забезпечення, приблизно та ж сама кількість знайдених вразливостей проте з меншою кількістю затрат часу та зусиль. Також треба враховувати, що вразливості та структура вебпрограм була відома перед написання скриптів.

4.2 Оцінка отриманого результату

Початкові вимогами були автоматизувати обробку різних видів сценаріїв тестування безпеки. Всі пункти, включені в план роботи, були успішно реалізовані під час роботи над дипломною роботою. З метою підтвердження достовірності результатів за роботою скриптів, були проведені всі тести мануально для перевірки роботи скриптів на наявність помилок.

Після закінчення написання скриптів, їх можна використовувати при тестуванні ВебСокет безпеки, коли є завдання підвищення ефективності тестування (з точки зору часу та зусиль, проте не як комплексний засіб тестування. З допомогою скриптів можна тестувати майже всі відповідні сценарії тестування безпеки, що стосуються ВебСокет під час тестів на проникнення. Однак користувач скриптів повинен мати достатню кількість розуміння того, як працює ВебСокет протокол та як підключаються скрипти в ZAP. Це не проблема, оскільки засіб був призначений для використання експертами в Інтернеті поле тестування безпеки додатків.

До створення скриптів потрібно було перевіряти безпеку вебпрограм мануально. Однак використовуючи скрипти цей процес пришвидшується, та витрати зусиль тестувальника зменшуються. У всіх аспектах ці скрипти корисні для тестування ВебСокет повідомлень та їх вивчення. З будь-якої точки зору, дослідження в даній роботі є успішними.

Висновок до розділу 4

В цьому розділі було проаналізовано результат практичної частини дослідження, а саме написання скрипті та їх роботу. Також було порівняно ефективність роботи написаних скриптів відносно ручного тестування.

Початковим завданням було написати скрипти для автоматизації тестування безпеки ВебСокетів. Як підтвердження результатів роботи скриптів, були проведені всі тести мануально для перевірки роботи скриптів на наявність помилок.

При автоматизованому тестування з допомогою написаних скриптів, не враховуючи час на написання скриптів та налаштування програмного забезпечення, знайдена та ж сама кількість вразливостей, як і при мануальному тестуванні, проте з меншою кількістю затрат часу та зусиль.

ВИСНОВКИ

Отже, результатом роботи є написані скрипти для тестування вебпрограми, що працює по ВебСокет протоколу, та перевірка їх ефективності. Отриманий результат може бути використаний та застосований в автоматизованих тестах на проникнення.

Після закінчення дослідження були визначені напрямки покращення роботи:

- розширення можливостей скриптів та їх оптимізація.
- врахування особливостей окремої вебпрограми.
- побудова окремого фреймворку, що працюватиме з консолі та використовуватиме ZAP API для автоматизованого тестування використовуючи різні параметри(скрипти, потоки, ресурси, порти).

В даний час стан безпеки ВебСокет протоколу стабільний, тобто нові типи вразливостей не були знайдені останнім часом. Однак, якщо новий тип вразливості буде знайдений у ВебСокет протоколі, потрібно буде додатково вивчати методи та можливі шляхи автоматизації тестування вразливості.

Також потрібно дослідити, як підтримується тестування у різних вебпрограмах, де ВебСокет з’єднання може бути реалізовано по різному.

Одною із функцій в майбутньому дослідженні є зробити більш автоматизований інструмент на основі ZAP API. В даний час скрипти запускаються вручну і це означає, що користувач повинен знати, як правильно користуватися ZAP та ВебСокетами.

Однак не всі тести повинні бути автоматизовані, оскільки застосування ВебСокетів широко відрізняється в кожного вбр-розробника. Деякі вебпрограми можуть використовувати ВебСокет з’єднання лише для потокового відео, тоді коли інші можуть використовувати його для прямого доступу до бази даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1.The Definitive Guide to HTML5 WebSocket. – 2 edition. – 2013. – P. 227 c. – Vanessa Wang – Access mode: https://www.amazon.com/Definitive-Guide-HTML5-WebSocket/dp/1430247401

2. Blokdyk Gerardus. WebSocket Second Edition Paperback. — 2 edition. — 2018. — P. 280 c. — Access mode: https://www.apress.com/gp/book/97814 30247401.

3. Vangos Pterneas. Getting Started with HTML5 WebSocket Programming. — 2013. — P. 110 c. — Access mode: https://www.amazon.com/Getting-S tarted-HTML5-WebSocket-Programming/dp/1782 166963.

4. Stuttard Dafydd, Pinto Marcus. The Web Application Hacker's Handbook Second Edition. — 2 edition. — 2011. — P. 912c. — Access mode: https: //portswigger.net/web-security/web-appli cation-hackers-handbook

5. A. Melnikov. Proposed Standard The WebSocket Protocol. - December 2011. - Access mode:https://datatracker.ietf.org/doc/html/rfc6455

6.Tony Hsiang-Chih Hsu. Practical Security Automation and Testing: Tools and techniques for automated security scanning and testing in DevSecOps. – Ferbruary 2019. – P.256 c. – Access mode:https://www.amazon.com/Practical-Security-Automation-Testing-techniques/dp/1789802024

7. Access mode: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Sec-WebSocket-Accept

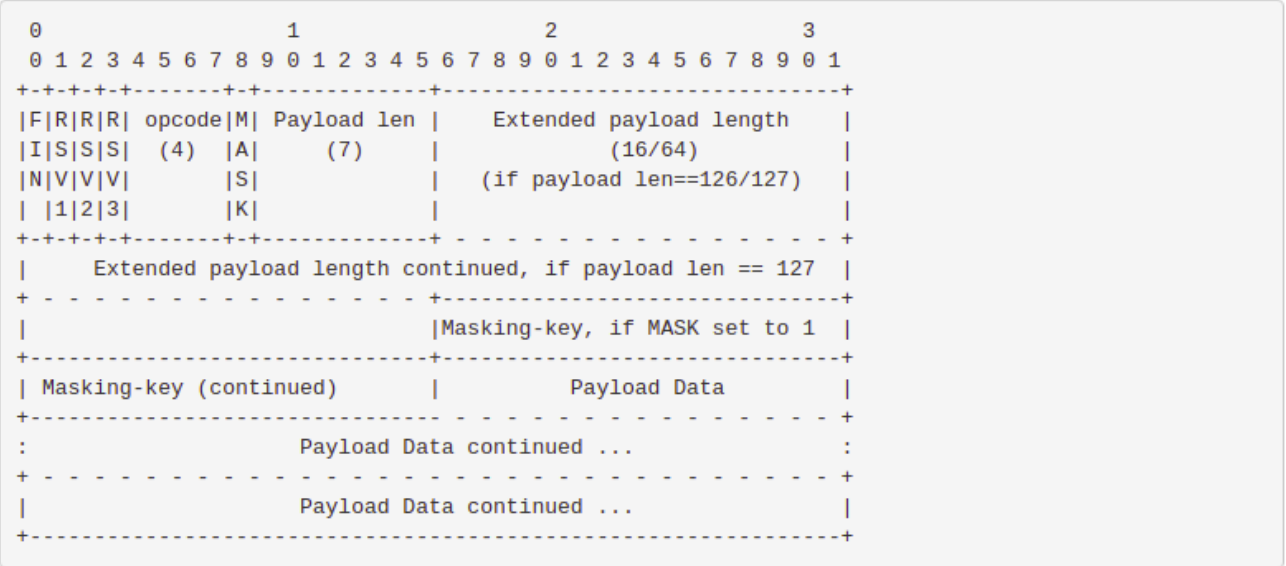
8. Access mode: https://tools.ietf.org/id/draft-ietf-hybi-thewebsocketprotocol-06.html

9. Access mode: http://www.w3.org/Security/wiki/Same_Origin_Policy

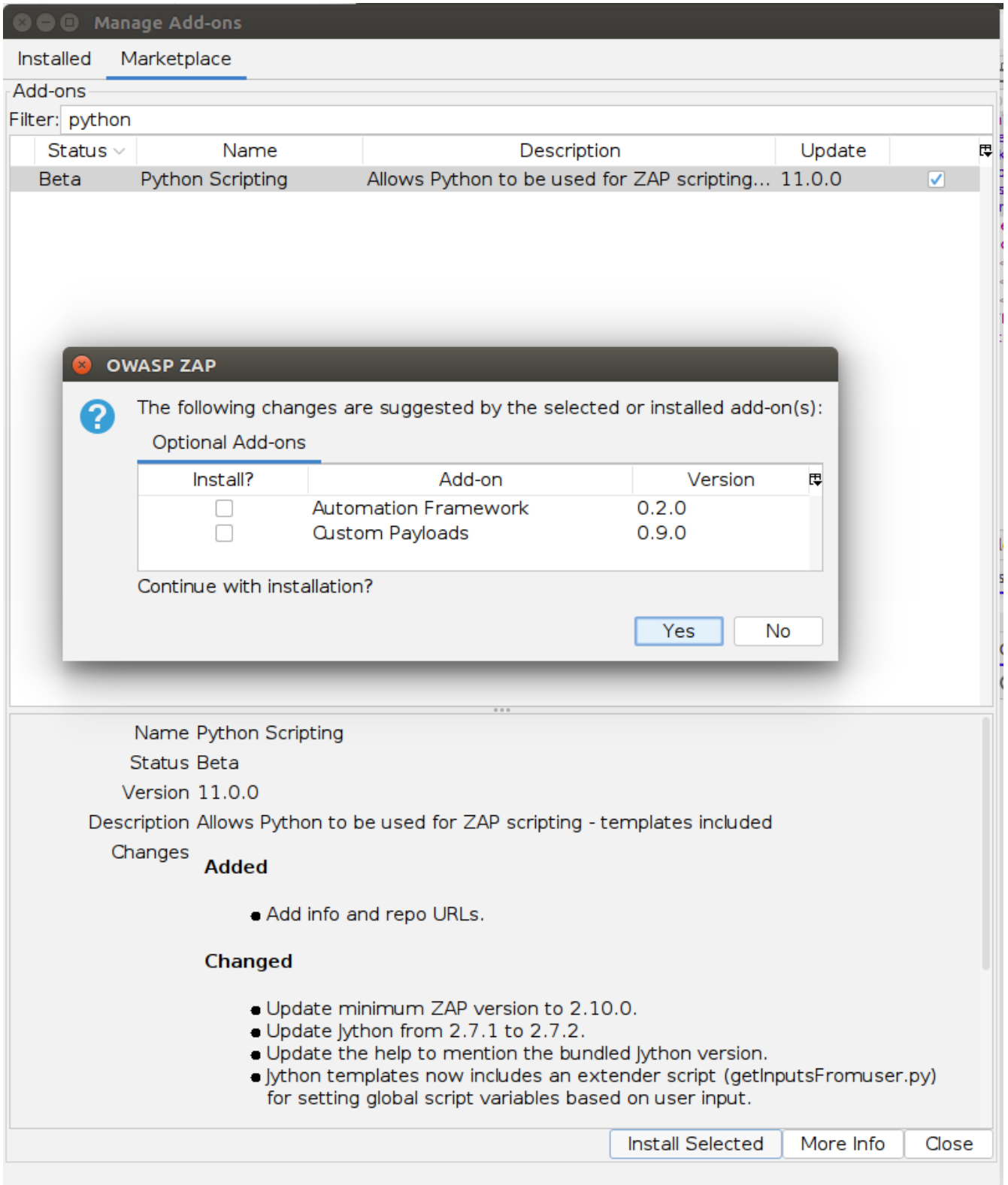
10. Access mode: https://javadoc.io/doc/org.zaproxy/zap/2.7.0/index.html

11. Access mode: https://www.zaproxy.org/docs/alerts

ДОДАТОК А


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
F	R	R	R	opcode	M	Payload len	Extended payload length
I	S	S	S	(4)	A	(7)	(16/64)
N	V	V	V		S		(if payload len==126/127)
	1	2	3		K		
Extended payload length continued, if payload len == 127
Masking-key, if MASK set to 1
Masking-key (continued)	Payload Data
: Payload Data continued ... :
| Payload Data continued ... |

ДОДАТОК Б


Manage Add-ons
Installed
Marketplace
Add-ons
Filter: python
Status
Name
Description
Update
Beta
Python Scripting
Allows Python to be used for ZAP scripting...
11.0.0
OWASP ZAP
The following changes are suggested by the selected or installed add-on(s):
Optional Add-ons
Install?
Add-on
Version
Automation Framework
0.2.0
Custom Payloads
0.9.0
Continue with installation?
Yes
No
Name Python Scripting
Status Beta
Version 11.0.0
Description Allows Python to be used for ZAP scripting - templates included
Changes
Added
Add info and repo URLs.
Changed
Update minimum ZAP version to 2.10.0.
Update Jython from 2.7.1 to 2.7.2.
Update the help to mention the bundled Jython version.
Jython templates now includes an extender script (getInputsFromuser.py) for setting global script variables based on user input.
Install Selected
More Info
Close

ДОДАТОК В

public static final int	**RISK_HIGH**	3
public static final int	**RISK_INFO**	0
public static final int	**RISK_LOW**	1
public static final int	**RISK_MEDIUM**	2