ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ ЗАБЕЗПЕЧЕННЯ ТЕСТУВАННЯ БЕЗПЕКИ WEB-САЙТІВ НА ОСНОВІ РІШЕННЯ OWASP ZAP»

Виконав студент 6 курсу, групи БСДМ-61
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)
Ахтьоров В.Ю
(прізвище та ініціали)
Керівник Довженко Н.М
(прізвище та ініціали)
Рецензент
(прізвище та ініціали)
Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ – 2023

ВСТУП

Актуальність дослідження. На сьогоднішній день через постійно зростаючу кількість кіберзагроз та їх інтенсивності, комплексності, виникає потреба в постійному покращенню і підтримці систем захисту інформації. Особливо гостро це постає для web-сайтів та інших ресурсів, які доступні з глобальної мережі для великої кількості користувачів. Забезпечення їх безпеки є складним процесом, який вимагає орієнтації на актуальні загрози та постійну адаптацію до нових умов.

Через це є потреба в рішеннях або практиках які б могли забезпечити та оцінити систему захисту інформації опираючись на актуальні загрози та специфіку окремого web-сайту або його компонентів.

Таким рішенням виступає тестування безпеки яке через природу свого проведення здатне адаптуватись під різні умови, вимоги та обмеження надаючи при цьому актуальну інформацію про рівень безпеки певної мережі або системи. Опираючись на тестування можливо ефективно розробити стратегію захисту та надати пріорітети саме тим елементам в системі, які цього потребують.

Мета роботи – розробка практичних рекомендацій щодо автоматизації тестування на основі OWASP ZAP.

Об'єкт дослідження – процес забезпечення захисту Web-сайтів на основі тестування безпеки.

Предмет дослідження – технологія проведення тестування на основі OWASP ZAP.

Наукові завдання:

- дослідити основні загрози для web-сайтів та динаміку їх зміни, проаналізувати статистику їх виявлення;

- проаналізувати специфіку проведення тестування безпеки, методології та підходи тестування, а також основні інструменти для його проведення;

- розробити рекомендації щодо тестування безпеки на основі рішення OWASP ZAP.

Практичне значення отриманих результатів полягає в можливості їх використання для оптимізації процесу тестування безпеки web-сайтів.

Апробація результатів магістерської роботи було оприлюднено на Всеукраїнській науковій конференції «Актуальні проблеми кібербезпеки», яка відбулася 27 жовтня 2022 року в Державному університеті телекомунікацій, м. Київ.

1 АНАЛІЗ СУЧАСНИХ ЗАГРОЗ ДЛЯ ВЕБ-САЙТІВ ТА МЕТОДІВ І ТЕХНОЛОГІЙ ЇХ ЗАХИСТУ

1.1 Аналіз загроз на основі OWASP TOP 10

Згідно до OWASP TOP 10 за 2021 рік, найпоширенішими загрозами для web-сайтів є [1]:

1) Порушення контролю доступу. Порушення контролю доступу це сценарій при якому порушується встановлені політикою безпеки розподілення доступів в системі і неавторизований користувач, або користувач без достатніх прав та привілеї отримує доступ до перегляду, модифікації, передачі або знищення інформації. Прикладом порушення контролю доступу є отримання інформації через модифікацію запиту до веб-сервера, яку не може отримати користувач без відповідного рівня привілеї. Порушення контролю доступу може призвести до фінансових та репутаційних втрат в разі витоку інформацію.

Контроль доступу та пов’язані з ним проблеми можна поділити на декілька категорій. Вертикальний контроль доступу – це механізми, які обмежують доступ до конфіденційних функцій, недоступних для інших типів користувачів. Завдяки вертикальному контролю доступу різні типи користувачів мають доступ до різних функцій програми. Наприклад, адміністратор може мати можливість змінювати або видаляти обліковий запис будь-якого користувача, тоді як звичайний користувач не має доступу до цих дій. Вертикальний контроль доступу може бути більш детальною реалізацією моделей безпеки, призначених для забезпечення дотримання бізнес-політик, таких як розподіл обов’язків і найменші привілеї.

Горизонтальний контроль доступу — це механізми, які обмежують доступ до ресурсів користувачам, яким спеціально дозволено доступ до цих ресурсів. Завдяки горизонтальному контролю доступу різні користувачі мають доступ до підмножини ресурсів одного типу. Наприклад, банківська програма дозволить

користувачеві переглядати транзакції та здійснювати платежі зі своїх власних рахунків, але не з рахунків будь-якого іншого користувача.

Згідно зі статистикою OWASP ця загроза була знайдена 318,487 та було знайдено 19,013 вразливостей які можуть призвести до цієї загрози.

2) Криптографічний збої. Криптографічний збій — це критична вразливість безпеки веб-програми, яка розкриває конфіденційні дані програми через слабкі криптографічний алгоритм. Це можуть бути паролі, медичні записи пацієнтів, комерційна таємниця, дані кредитної картки, адреси електронної пошти чи інша особиста інформація користувача.

Сучасні веб-сайти обробляють дані в стані спокою та при передачі їх під час запитів, що вимагає суворого контролю безпеки для комплексного пом'якшення загроз. У деяких розгортаннях використовуються слабкі криптографічні методи, які можна зламати протягом розумного періоду часу. Навіть за ідеальної реалізації криптографічних методів користувачі можуть уникати застосування найкращих методів захисту даних, що згодом робить конфіденційну інформацію сприйнятливою до крадіжки конфіденційних даних.

Погана криптографія безпосередньо впливає на безпеку програми та її даних. Відсутність безпеки може дозволити зловмисникам викрасти та змінити дані для шахрайства та крадіжки особистих даних, що може призвести до серйозних наслідків. Зловмисники намагаються викрасти ключі, здійснити атаки типу «людина посередині» або викрасти дані з сервера, під час передачі чи з браузера. Це знову призводить до компрометації конфіденційної інформації.

Вплив криптографічного збою не обмежується крадіжкою частини інформації від користувача. Зловмисники можуть заволодіти повною базою даних, що містить тисячі конфіденційної інформації, крадіжки даних, публічні списки, порушення та багато критичних проблем із даними, пов'язаними з бізнесом. Ви також можете уявити сценарій, коли облікові дані адміністратора викрадають, а зловмисник отримує повний контроль над сервером. Криптографічні збої можуть призвести до непоправної шкоди репутації та серйозних судових позовів.

Згідно зі статисткою OWASP ця загроза була знайдена 233,788 та було знайдено 3,075 вразливостей які можуть призвести до цієї загрози.

3) SQL — ін'єкції та інші типи ін'єкцій [2]- це вразливість веб-захисту, яка дозволяє зловмиснику втручатися в запити, які програма робить до своєї бази даних. Зазвичай це дозволяє зловмиснику переглядати дані, які вони зазвичай не можуть отримати. Це може включати дані, що належать іншим користувачам, або будь-які інші дані, до яких має доступ сама програма. У багатьох випадках зловмисник може змінити або видалити ці дані, викликаючи постійні зміни у вмісті або поведінці програми.

У деяких ситуаціях зловмисник може посилити атаку SQL-ін’єкції, щоб скомпрометувати базовий сервер чи іншу серверну інфраструктуру, або здійснити атаку типу «відмова в обслуговуванні».

Під час ін’єкційної атаки зловмисник може ввести зловмисний вхід у веб-програму (ввести її) та змінити роботу програми, змусивши її виконувати певні команди. Ін’єкційна атака може викрити або пошкодити дані та призвести до відмови в обслуговуванні або повної компрометації веб-сервера. Такі атаки можливі через уразливості в коді програми, яка дозволяє неперевірений ввід користувача.

Згідно зі статисткою OWASP ця загроза була знайдена 274,228 та було знайдено 32,078 вразливостей які можуть призвести до цієї загрози.

4) Небезпечний дизайн. Небезпечний дизайн зосереджений на ризиках, пов'язаних з недоліками в дизайні та архітектурі. Він зосереджений на необхідності моделювання загроз, безпечних шаблонів проєктування та принципів. Недоліки в незахищеному дизайні не можна виправити реалізацією. OWASP відрізняє незахищений дизайн від впровадження безпеки та контролює наступним чином: Незахищений дизайн неможливо виправити ідеальною реалізацією, оскільки, за визначенням, необхідні засоби контролю безпеки ніколи не створювалися для захисту від конкретних атак. Щоб використати незахищений дизайн, зловмисники можуть загрожувати робочим процесам моделі в

програмному забезпеченні, щоб виявити широкий спектр вразливостей і слабких місць.

5)Помилки в конфігурації безпеки. Помилки в конфігурації безпеки виникають, коли параметри безпеки неправильно визначені в процесі конфігурації або підтримуються та розгортаються з параметрами за замовчуванням. Це може вплинути на будь-який рівень стека програм, хмари чи мережі. Неправильно налаштовані хмари є основною причиною витоку даних, що коштує організаціям мільйони доларів. Уразливості зазвичай з'являються під час налаштування[4].

Типова вразливість неправильної конфігурації виникає під час використання таких засобів:

-Стандартні параметри, включаючи паролі, сертифікати;

-Застарілі протоколи та шифрування

-Відкрити екземпляри бази даних

-Повідомлення про помилки, що показують конфіденційну інформацію;

-Непотрібні функції, зокрема сторінки, порти.

6) Вразливі та застарілі компоненти. Програмний компонент — це частина системи або програми, яка розширює функціональні можливості програми, наприклад модуль, програмний пакет або API. Уразливості на основі компонентів виникають, коли програмний компонент не підтримується, застарів або вразливий до відомого експлойту. Ви можете випадково використовувати вразливі компоненти програмного забезпечення у виробничих середовищах, створюючи загрозу для веб-програми. Наприклад, організація може завантажити та використовувати програмний компонент, наприклад OpenSSL, і не в змозі регулярно оновлювати чи виправляти компонент, коли виявляються недоліки. Оскільки багато програмних компонентів працюють із тими самими привілеями, що й сама програма, будь-які вразливості чи недоліки в компоненті можуть призвести до загрози веб-програмі [5].

7) Помилки в ідентифікація та аутентифікації. Помилки в ідентифікації та аутентифікації виникають коли функціонал пов'язаний з ідентифікацію та

аутентифікацією користувачів, а також функції управління сеансами невірно імплементовані або мають помилки в конфігурації. Ця загроза проявляється коли зловмисник компрометувати паролі, токени сесії, використовує вразливості системи для надання собі прав або ідентифікації себе , як іншого користувача [6].

8) Проблеми з інтеграцією. Дана вразливість виникає при інтеграції нових модулів та оновлень в систему веб-додатку або веб-сайту без відповідної верифікації їх цілісності. Подібне може призвести до додавання шкідливого коду в веб-додаток при оновленні, створенню нових вразливостей в системі в цілому.

9) Проблеми з логуванням та моніторингом. Ця вразливість проявляється в будь-яких помилках при логуванні , моніторингу та звітуванні системних подій пов’язаних з безпекою.

10) Підробка запитів зі сторони сервера. Підробка запитів на стороні сервера (також відома як SSRF) — це вразливість веб-безпеки, яка дозволяє зловмиснику спонукати програму на стороні сервера робити запити до ненавмисного місця. У типовій атаці SSRF зловмисник може змусити сервер встановити з’єднання лише з внутрішніми службами в інфраструктурі організації. В інших випадках вони можуть змусити сервер підключатися до довільних зовнішніх систем, що потенційно може призвести до витоку конфіденційних даних, таких як облікові дані авторизації.

Динаміка загроз також постійно змінюється і для забезпечення достатнього рівня захисту потрібно постійно відштовхуватись від актуальних даних. Таким чином, загрози які були актуальним в минулі роки можуть поступатися позиціями на сьогодні, бути об’єднані в більш широку категорію [7].

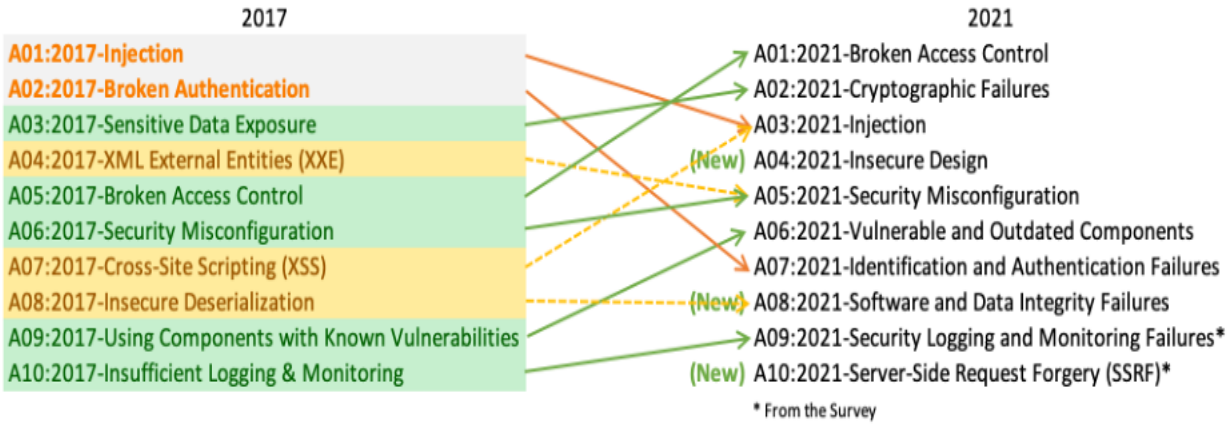


Рис. 1.1. Порівняння загроз згідно OWASP ZAP за 2017 та 2021 роки

При порівнянні актуальних загроз для 2017 року та 2021 року, можна помітити, що найкритичніша загроза – ін’єкції, перейшла від 1-го місця в OWASP TOP 10 , до 4-го. Також в списку зникли деякі з загроз та з’явилися нові, такі як небезпечний дизайн та підробка запитів зі сторони сервера. Подібну динаміку можна відслідкувати не тільки на дистанції в декілька років. Для порівняння, OWASP TOP 10 2020 представляє SQL та інші типи ін'єкцій веб-сайтів, як надкритичну загрозу, в порівняні з її 3 місцем в останній версії списку. Подібні зміни можна прослідкувати в багатьох пунктах списку. Таким чином порушення контрою доступу виступало лише 5 в списку загроз, а проблеми з логуванням та моніторингом — 10.

Також варто помітити, що змінюється не лише порядок загроз. Багато вказаних в OWASP TOP 10 2021 загроз є новими для списку, в порівнянні з минулим роком. Підробка запитів зі сторони сервера. проблеми з інтеграцією, збої в криптографічних алгоритмах, небезпечний дизайн, помилки в ідентифікація та аутентифікації є новими актуальними вразливостями які не мало такої критичності в минулому.

Статистика кібератака також є важливим фактором при побудові системи захисту для веб-сайту. Так , за останній час все більше зростає кількість атак з використанням злоякісних програм-вимагачів. Лише за третій квартал 2022 року було зафіксовано 48415364 атаки цього типу. Також за цей квартал зросла середня

кількість атак на організації. В цілому фіксується 800 атак за неділю на організацію. Це є на 20.5% вищим ніж минулий квартал [8].

Цікавим є розподілення атак за індустрією. Серед найбільш вразливих виділяється охорона здоров'я та роздрібний продаж. В їх системах виявлено найбільша кількість вразливостей та проведена найбільша кількість атак за останній рік. Серед основних проблем при забезпеченні кібербезпеки відносять нестачу робочої сили, вибір програмного забезпечення виробника [9].

Розглядаючи основні вразливості, які можна зустріти в сучасних веб-сайтах та веб-додатках можна виділити цілий ряд , як незначних і простих для вирішення, так і більш комплексних вразливостей. До основних вразливостей, які можуть бути використані для реалізації згаданих загроз можна віднести :

CWE-22 - Невірне обмеження шляху в директорію з обмеженим доступом. Ця вразливість виражається в тому, що веб-сервер і його програмне забезпечення використовує зовнішні вхідні дані для створення імені шляху, призначеного для ідентифікації файлу чи каталогу, розташованого під обмеженим батьківським каталогом, але програмне забезпечення не нейтралізує належним чином спеціальні елементи в імені шляху, які можуть призвести до визначення шляху до розташування, яке знаходиться поза межами обмеженого каталогу.

CWE-324 - Використання ключа шифрування після кінця терміну придатності Вразливість визначається в використанні криптографічних ключів , які вийшли за межі свого терміну придатності. Це не обов'язково означає що вони були скомпрометовані , проте значно підвищує шанси цього.

CWE-20 Невірна валідація вводу Веб-сайт отримує вхідні дані , але не перевіряє або невірно перевіряє, чи вхідні дані мають властивості, необхідні для безпечної та правильної їх обробки [10].

1.2 Методи, підходи та технології забезпечення захисту веб-сайтів

Всі підходи до захисту веб-сайту можна поділити на декілька категорій за

різною класифікацією, проте в основному виділяються системи і технології захисту веб-серверу, захист від DoS та DDoS, організаційні заходи, підтримка системи та резервне копіювання.

До систем та технологій захисту веб-серверу можна віднести IPS , IDS , міжмережеві екрани. SIEM-системи. Вони забезпечують основний захист мережевої інфраструктури на якій зберігається та підтримується веб-сайт або веб-додаток.

Тестування безпеки веб-сайтів — процес тестування, аналізу та звітування рівня безпеки веб-сайту. Якщо брати більш точне визначення, то тестування безпеки це процес порівняння стану системи по певним критеріям. Критеріями в даному випадку можуть виступати стандарти, політики безпеки, моделі загроз, наявність в системі певних критичних вразливостей.

Тестування є комплексним процесом, яке включає в себе різноманітну та різно правлену діяльність. Воно може проводитись на самих ранніх стадіях розробки проекту та до його останньої стаді підтримки. Через це, тестування безпеки веб-сайту здатно надавати актуальну інформацію про стан та вразливості. Також тестування може адаптуватись під нові критерії і постійно змінюючись кіберпростір [11].

Як вже згадувалось раніше тестування проводиться на всіх етапах життєвого циклу розробки веб-сайту. В загальному вигляді цей цикл включає в себе наступні етапи:

- Визначення та аналіз вимог — формування загального представлення про майбутній проект;

- Дизайн — конкретизація структури та взаємодій на веб-сатйі, деталізації функціоналу;

- Розробка — власне розробка зі створенням фронт-енд та бек-енд частин веб-сайту. На цьому етапі пишеться вихідний код, розробляються бази даних, основні сторінки та функціонал веб-сайту;

- Впровадження — додання нового розробленого функціоналу до вже існуючого веб-сайту або винесення розробленого веб-сайту на платформу (веб-сервер)б де він буде доступним для користувачів.

- Підтримка — підтримка актуальної версії веб-сайту та розробка нового функціоналу для нього. Також даний етап включає в себе збір інформації та постійний моніторинг про стан веб-сайту , його недоліки та уразливості, як зі сторони безпеки, так і зі сторони некоректного функціоналу певних елементів, що не несе в собі загрози безпеці.

Тестування на самих ранніх етапах являє собою аналіз вимог та критеріїв для веб-сайту на предмет наявності в них вразливих елементів, які пізніше будуть додані на етапі дизайну та розробки. Також під час такого тестування проходить формування тест-кейсів, критеріїв, графіків та підбір інструментів для майбутнього тестування[12].

Тестування на етапі дизайну схоже з минулим етапом, проте в даному випадку у тестувальників є в наявності більш чіткі дані про веб-сайт та його структуру, а отже більше можливостей знайти відповідні вразливості, які все ще не були додані на етапі розробки. Головна задача тестування на цьому етапі це попередження майбутніх загроз та вразливостей які виникнуть при впровадженні незахищеного дизайну.

Тестування під час етапу розробки являє собою тестування окремих розроблених елементів веб-сайту на предмет їх вразливостей (Unity Testing), тестування інтеграції різних елементів між собою (Integration Testing) та тестування веб-сайту в цілому (System Testing) на самих пізніх етапах розробки. Цей етап тестування надає найбільшу кількість конкретних даних про стан веб-сайту адже об’єктом тестування є вже готові елементи функціоналу, які можуть бути додані до актуальної версії.

Тестування на етапі впровадженні та підтримки це постійний процес моніторингу стану веб-сайту, знаходження нових вразливостей, проведення регресивного тестуванні при провадженні нового функціоналу або проведення

тестування за новими виниклими критеріями для запобігання реалізації поточний критичних загроз [13].

До основних принципів тестування можна віднести:

- Ціна помилки або вразливості зростає з кожним етапом при якому вона не була виявлена або виправлена. Подібне зростання пов’язане з тим що коли існуючу вразливість реалізують на певному із етапів складність її видалення на наступних, коли у функційного елемента з даною вразливістю вже є інтеграції і взаємозв’язки в системі складно. Особливо яскраво це виділяється при вразливостях створених на етапі дизайну, або формування вимог, але виявлених лише на дуже пізніх етапах. Подібна вразливість буде лежати в самій архітектурі веб-сайту і її виправлення може означити повну зміну основних принципів і вимог на яких був побудований веб-сайт. При управлінні ризиками для веб-сайту необхідно враховувати цю особливість і проводити тестування на самих ранніх етапах життєвого циклу щоб мінімізувати її вплив;

- Важливо знати, який рівень безпеки вимагатиме певний проект. Активи, які підлягають захисту, мають бути надано класифікацію, яка визначає, як з ними потрібно поводитись (наприклад, конфіденційно, секретно, цілком таємно). Подібна класифікація активів дозволить пріорі тезувати задачі розробки, надати відповідний рівень захисту веб-сайту та надасть можливість виділити компоненти які потребують найбільш затрат ресурсів та часу при тестуванні [14].

Без проведення подібної оцінки та класифікації неможливо провести тестування за критеріями.

- Важливою частиною хорошої програми безпеки є здатність визначити, чи є покращення. Важливо відслідковувати результати тестування та розробити показники, які виявлятимуть тенденції безпеки додатків у межах організації. Хороші показники покажуть:

- Чи потрібне додаткове навчання персоналу та розгляду проблем безпеки;

- Чи існує певний механізм безпеки, який не зовсім зрозумілий команді розробників;

- Чи загальна кількість виявлених проблем, пов’язаних із безпекою, зменшується.

Послідовні показники, які можна згенерувати автоматизованим способом із доступного вихідного коду, також допоможуть організації в оцінці ефективності механізмів, запроваджених для зменшення помилок безпеки в програмному забезпеченні розвитку [15].

- *Необхідність документації*. Щоб завершити процес тестування, важливо створити формальний запис про те, які дії тестування було виконано, ким, коли вони були виконані, і деталі результатів тесту. Доцільно домовитися про прийнятний формат звіту, який корисний для всіх зацікавлених сторін, серед яких можуть бути розробники, керівники проектів, власники бізнесу, IT відділ. Звіт має чітко визначити для власника бізнесу, де існують суттєві ризики, і зробити це у спосіб, достатній для отримати їхню підтримку для подальших дій із пом’якшення наслідків. Звіт також має бути зрозумілим для розробника, щоб точно визначити точна функція, на яку впливає вразливість, і відповідні рекомендації щодо вирішення проблем у мові що розробник зрозуміє. Звіт також має дозволити іншому тестувальнику безпеки відтворити результати. Написання звіт не повинен бути надто обтяжливим для самого тестувальника безпеки. Тестери безпеки, як правило, не є відомі своїми навичками творчого письма, і узгодження складного звіту може призвести до випадків, коли результати тестів будуть такими не оформлено належним чином. Використання шаблону звіту про перевірку безпеки може заощадити час і гарантувати документальне оформлення результатів точно й послідовно, а також у форматі, який підходить для аудиторії.

До основних підходів проведення тестування можна віднести:

- *Тестування за принципом чорного ящика*. Тестування за принципом чорного ящика тестувальнику не надається доступ до бек-енду або вихідного коду веб-сайту, а при тестуванні на проникнення навіть прямого доступу до мережевої інфраструктури. Тестер повинен виконати розвідку, щоб отримати інформацію про систему та самостійні знайти недоліки та вразливості, які дозволять йому

отримати подальший доступ. Цей вид тестування є найбільш реалістичною симуляцією кібератаки. Однак це також вимагає багато часу та має найбільший потенціал, щоб не помітити вразливість, яка існує у внутрішній частині архітектури веб-сайту. Реальний зловмисник зазвичай не має часових обмежень і може витратити місяці на розробку плану атаки в очікуванні слушної нагоди.

- *Тестування за принципом білого ящика*. Тестування за принципом білого ящику дозволяє тестеру мати повний відкритий доступ до всіх програм і систем. Тестеру надається доступ високого рівня до мережі, внутрішньої архітектури та може переглядати вихідний код. Тестування за принципом білого ящику спрямоване на виявлення потенційних слабких місць у різних сферах, таких як логічні вразливості, потенційні ризики безпеки, неправильні конфігурації безпеки, погано написаний код розробки та відсутність захисних заходів. Цей тип оцінки є більш комплексним, оскільки як внутрішні, так і зовнішні вразливості оцінюються з «залаштунковий» точки зору, яка недоступна типовим зловмисникам. Знову ж таки, оскільки для ретельного аналізу всіх аспектів системи потрібно так багато часу, тестування білої скриньки, як правило, зарезервовано для систем високого ризику або тих, які обробляють конфіденційні дані .

- *Тестування за принципом сірого ящика*. Тестування за принципом сірого ящику тестування це підхід до тестування веб-сайту або веб-додатку з частковим знанням внутрішньої структури програми. Метою тестування сірого ящика є пошук і виявлення дефектів через неправильну структуру коду або неправильне викання вихідного коду.

До основних типів тестування безпеки можна віднести:

- *Ручні перевірки*. Ручні перевірки робляться експертами та перевіряють впроваджені політики, процеси, а також дизайн та архітектуру веб-сайту або веб-додатку. Подібний тип тестування можна проводити з самих ранніх етапів життєвого циклу проекту.

- *Моделювання загроз*. Моделювання загроз – це перевірка системи та аналіз ризиків на основі потенційної реалізації певної загрози. Для проведення такого

тестування необхідно формувати чіткий опис загроз, які будуть актуальними для даної системи.

- *Перегляд вихідного коду*. Перегляд вихідного коду може проводитися лише зі стадії впровадження та є доволі затратною формою тестування, проте він надає чіткі дані про вразливості в системі на рівні коду, адже будь яка загроза яка є в системі може бути знайдена на рівні вихідного коду.

- *Тестування на проникнення*. Тестування на проникнення - це імітована кібератака на комп'ютерну систему з метою перевірки на наявність уразливостей, які можна використовувати. У контексті безпеки веб-додатків тестування на проникнення зазвичай використовується для посилення брандмауера веб-додатків (WAF). Подібне тестування можна проводити лише після етапу впровадження, але воно надає найбільш практичні данні про вразливості системи.Тестування на проникнення вважається проактивним заходом кібербезпеки, оскільки передбачає послідовні само ініціативні вдосконалення на основі звітів, які створює тест. Це відрізняється від не проактивних підходів, які не виправляють недоліки, коли вони виникають. Наприклад, не проактивний підхід до кібербезпеки передбачав би, щоб компанія оновлювала свій брандмауер після витоку даних. Метою профілактичних заходів, таких як тестування пера, є мінімізація кількості ретроактивних оновлень і максимізація безпеки організації.

Визначення цілей для показників і вимірювань тестування безпеки є необхідною умовою для проведення і подальшого використання даних з тестування безпеки.

Прикладом показника для тестування є кількість знайдених в веб-сайті вразливостей або кількість вразливостей певного рівня критичності. Подібні метрики також дозволяють визначитись з напрямом використання даних тестування для зміни системи захисту інформації.

Ще одним напрямок використання метрик та показників є порівняння стану об'єкту тестування до і після внесення певних змін, при цьому тести якими визначався початковий стан безпеки можуть бути замінені на тести іншого

формату. Наприклад, за допомогою тестування на проникнення було визначено ряд вразливостей в веб-сайті та їх критичність. Провівши аналіз вихідного коду та ще одне тестування на проникнення можна визначити чи відбулися зміни в загальному стані безпеки після оновлення.

При звичайному тестуванні кількість виявлених дефектів може виступати показником якості об’єкту тестування. Подібним чином тестування безпеки може визначити рівень безпеки веб-сайту або веб-додатку. Також всі знайдені недоліки в веб-сайті та вразливості можуть бути проаналізовані для визначення корінної причини їх виникнення. Корінною причиною може виступати недолік в дизайні або структура самого вихідного коду. Також всі знайдені вразливості необхідно класифікувати за ресурсами які потрібно витратити для їх усунення. Подібне є загальною характеристикою як для звичайного тестування, так і для тестування безпеки.

Головною відмінністю між тестовими даними отриманими під час звичайного тестування і тестування безпеки є розгляд всіх даних з позицію ризиків для інформації в системі під час тестування безпеки.

Тестування безпеки полягає в управлінні ризиками, визначення їх критичності та необхідних дій для роботи з ними. . З цієї причини дані тестування безпеки повинні підтримувати загальну стратегію управління ризиками безпеки на критичних контрольних точках під час SDLC.

Наприклад, уразливості, виявлені у вихідному коді за допомогою аналізу вихідного коду, є початковим показником ризику. Рівень ризику (наприклад, високий, середній, низький) для вразливості можна розрахувати шляхом визначення фінансовими або іншими втратами при його реалізації та ймовірності цієї реалізації, а також шляхом перевірки вразливості за допомогою тестів на проникнення.

Подібний розгляд ризику дозволяє ефективно оцінити подальший напрям роботи з ним, такий як ресурси які можуть бути витрачені на усунення, прийняття ризику без додаткових дій, моніторинг ризику.

Оцінюючи рівень безпеки веб-сайту, важливо брати до уваги певні фактори, як розмір самого проекту, що розробляється. Управління ризиками і проведення тестування безпеки є менш ресурсозатратним для маленьких проектів, тому всі тестові данні мають аналізуватися з врахуванням цього факту.

Коли тестування безпеки виконується на кількох етапах SDLC, тестові дані з різних тестів можуть підтвердити наявність однакових вразливостей. Таку особливість можна використати ,як для економії ресурсів під час тестування, так і для і підтвердження, що певна вразливість не перейшла з одного етапу життєвого циклу на інший. Дані випробувань також можуть підтвердити ефективність видалення уразливості шляхом впровадження контрзаходів на різних контрольних точках SDLC.

Висновки до розділу 1

Розглянуто основні загрози та статистику кібератак на web-сайти та web-додатки, досліджено процес проведення тестування безпеки та основні методи та підходи.

Проаналізовано динаміку змін актуальних загроз для web-сайтів та web-додатків. Визначено, що критичність та поширеність загроз проходить серйозні зміни кожного року, а тому виникає необхідність в постійній адаптації системи захисту інформації під нові умови для забезпечення достатнього рівня безпеки. Виокремлено основні етапи життєвого циклу веб-сайту та впровадження тестування безпеки на цих етапах.

Підкреслено основні підходи та методи проведення тестування. Виведено основні відмінності та дані які можуть бути отримані при використанні певних методів та підходів.

2 ТЕХНОЛОГІЇ ТА ІНСТРУМЕНТИ ТЕСТУВАННЯ БЕЗПЕКИ WEB-САЙТІВ

2.1 Загальний огляд інструментів та технологій тестування

Оскільки тестування безпеки включає в себе різноманітну діяльність на різних етапах життєвого циклу проекту і воно торкається різних технологій, функцій та компонентів самого веб-сайту необхідний широкий набір інструментів для забезпечення його проведення.

До основних сучасних інструментів можна віднести:

Nmap («Network Mapper») - це утиліта з відкритим вихідним кодом для дослідження мережі та перевірки безпеки. Вона була розроблена для швидкого сканування великих мереж, хоча чудово справляється і з одиничними цілями. Nmap використовує «сирі» IP пакети оригінальним способом, щоб визначити які хости доступні в мережі, які служби (назва програми та версію) вони пропонують, які операційні системи (і версії ОС) вони використовують, які типи пакетних фільтрів/брандмауерів використовуються і ще багато інших характеристик. У той час, як Nmap зазвичай використовується для перевірки безпеки, багато системних адміністраторів знаходять її корисною для звичайних завдань, таких як контроль структури мережі, управління розкладами запуску служб і облік часу роботи хосту або служби.

В Nmap є такі функції - Сканування цільової мережі або хосту, виявлення нових хостів, настройка типів сканування, визначення портів, визначення служб та їх версій, сканування з скриптами, визначення ОС, обхід брандмауера.

Під час тестування безпеки методом тестування на проникнення, вона дозволяє отримати загальну картину про мережу, сервіси , операційні системи та порти доступні хостах , які зможе різниться зловмисник під час проведення атаки

Це дозволить формувати подальші тести відштовхуючись від знайденої інформації, та надасть данні для доповнення тестів в цілому.

Щодо сканування та знаходження хостів має наступні особливості. Оскільки завдання, що вимагають виявлення хостів настільки різні, Nmap надає велику різноманітність опцій для різних методів. Завдання виявлення хостів іноді називають пінг скануванням (ping scan), проте вона набагато перевершує використання звичайних ICMP запитів, що асоціюються з ping утилітами. Користувачі можуть повністю пропустити крок пінг сканування за допомогою опції сканування з метою складання списку (-sL) або просто відключивши його (-PN), або сканувати мережу за допомогою довільних комбінацій мультипортових TCP SYN/ACK, UDP та ICMP запитів. Метою всіх цих запитів є отримання відповідей, що вказують, що IP-адреса в даний час активна (використовується хостом або мережевим пристроєм). У більшості мереж лише невеликий відсоток IP-адрес активний постійно. Це особливо притаманно адресних просторів виду 10.0.0.0/8. Такі мережі мають 16 млн IP адрес, але я бачив, як вони використовуються компаніями, в яких не більше тисячі машин. Функція виявлення хостів може знайти ці машини у цьому неосяжному морі IP адрес.

Якщо брати до уваги сканування портів, то більшість типів сканування доступні лише привілейованим користувачам, тому що надсилаються та приймаються сирі пакети, що потребує прав користувача root на Unix системах. Під Windows рекомендується працювати з обліковим записом адміністратора, хоча іноді Nmap працює і з непривілейованими користувачами, коли в ОС вже завантажена утиліта WinPcap. Вимога root привілеїв було серйозним обмеженням, коли Nmap вийшла друком у 1997, т.к. багато користувачів мали доступ тільки до облікових записів. Нині світ змінився. Комп'ютери стали дешевшими, багато користувачів мають постійний доступ до Інтернету, а Unix системи для домашніх комп'ютерів (включаючи Linux та Mac OS X) тепер широко поширені. Також тепер доступна Windows версія Nmap, що дозволяє запускати її на більшій кількості комп'ютерів. З цих причин, користувачам немає необхідності запускати облікових

записів. Це великий успіх, т.к. функції, що вимагають привілейованого доступу, роблять Nmap набагато більш потужною і гнучкою.

Коли Nmap робить спробу видати правильні результати, треба мати на увазі, що вся інформація базується на пакетах, повернутих цільовими машинами (або брандмауер перед ними). Такі хости можуть бути ненадійними і надсилати відповіді з метою ввести Nmap в оману. Набагато більш поширеною нагодою є не сумісні з RFC хости, які відповідають на запити Nmap не так, як повинні. Сканування типу FIN, NULL і Xmas найбільш сприйнятливі до таких проблем.

Ще одним з інструментів може виступати сканер безпеки Nikto. Nikto — це сканер веб-серверів із відкритим вихідним кодом (GPL), який виконує комплексні тести на веб-серверах для багатьох елементів, у тому числі понад 6700 потенційно небезпечних файлів/програм, перевіряє наявність застарілих версій понад 1250 серверів і перевіряє проблеми, пов’язані з версією, на понад 270 серверах. Він також перевіряє елементи конфігурації сервера, такі як наявність кількох індексних файлів, параметри HTTP-сервера, і намагається визначити встановлені веб-сервери та програмне забезпечення. Елементи сканування та плагіни часто оновлюються та можуть оновлюватися автоматично.

Nikto не розроблений як прихований інструмент. Він якнайшвидше перевірить веб-сервер і буде очевидним у файлах журналу або в IPS/IDS. Однак існує підтримка методів LibWhisker проти IDS, якщо захочете спробувати (або перевірити свою систему IDS).

Не кожна перевірка є проблемою безпеки. Є деякі елементи, які є перевірками типу «лише інформація», які шукають речі, які можуть не мати недоліків безпеки, але тестувальник може не знати про наявність на сервері. Ці елементи зазвичай позначаються відповідним чином у надрукованій інформації. Існують також деякі перевірки невідомих елементів, які були відскановані у файлах журналу.

Основною функцією цього рішення є дослідження веб-сервера для виявлення потенційних проблем і вразливостей безпеки, зокрема:

-Неправильна конфігурація сервера та програмного забезпечення

-Файли та програми за замовчуванням

-Незахищені файли та програми

-Застарілі сервери та програми

-Вказівки, які допоможуть людині-тестеру покращити ручне тестування.

Nikto побудовано на LibWhisker2 (від Rain Forest Puppy) і може працювати на будь-якій платформі, яка має середовище Perl. Він підтримує автентифікацію хосту, кодування атак тощо.

SSLLabs-Scan — відомий постачальних рішень щодо хмарної безпеки для сканування безпеки мережі та управління вразливостями. Він перевіряє деталі веб-сайтів, які підтримують HTTPS з'єднання. Це рішення має наступний алгоритм перевірки :

1) Перевірка сертифікату, щоб переконатися, що він дійсний і випущений надійним центром сертифікації.

2) Перегляд конфігурації сервера в трьох категоріях: підтримка протоколів, підтримка обміну ключами та підтримка шифру.

3) Розрахування оцінки на від 0 до 100, яку можна винести на основ проведеної перевірки

4) Застосування низку правил, щоб призначити буквену оцінку (A+, A-B, C, D, E або F) до деяких аспектів конфігурації сервера які не можна оцінити за допомогою числових балів. Оцінка понижається за використання ненадійної конфігурації.

Ще одним напрямком тестування безпеки перехопленням пакетів в мережі. Аналіз пакетів, який часто називають перехопленням пакетів або аналізом протоколів, описує процес захоплення та інтерпретації трафіку, який проходить через мережу щоб краще зрозуміти, що відбувається в цій мережі. Аналіз пакетів зазвичай виконується сніфером пакетів, інструментом, який використовується для захоплення необроблених даних мережеві дані. Аналіз пакетів може допомогти з наступним:

- Розуміння характеристик мережі;

- Визначення хостів в мережі та їх сервісів;

- Визначення того, хто або що використовує доступну пропускну здатність;

- Визначення часу пікового використання мережі;

- Виявлення можливих атак або зловмисної діяльності;

- Пошук незахищених веб-додатків.

Wireshark — це сніфер пакетів з відкритим кодом. Wireshark фіксує дані, що надходять або проходять через мережеві карти на своєму пристрої, використовуючи базову бібліотеку захоплення пакетів. За замовчуванням Wireshark збирає лише дані на пристрої, але він може захоплювати майже всі дані в локальній мережі, якщо працювати в безладному режимі. Зараз Wireshark використовує бібліотеку захоплення пакетів NMAP (називається npcap).

SQLMap - це інструмент тестування на проникнення з відкритим вихідним кодом, який автоматизує процес виявлення та використання недоліків впровадження SQL і захоплення серверів баз даних. Він постачається з потужним механізмом виявлення, багатьма спеціальними функціями для тестування на проникнення та широким набором конфігурацій і компонентів, починаючи від відбитків бази даних, запитом даних і закінчуючи доступом до основної файлової системи та виконанням команд в операційній системі через вихідні поза діапазонні з'єднання [16].

Grep Rough Audit - це простий скрипт і набори сигнатур, які дозволяють знаходити потенційні недоліки безпеки у вихідному коді за допомогою утиліти GNU grep. Його можна порівняти з іншими програмами для статичного аналізу, такими як RATS, SWAAT і дефектоскоп, проте основною різницею між ним і іншими рішеннями є мінімальний інтерфейс і навантаження по ресурсам.

SonarQube — це інструмент із відкритим кодом для безперервної перевірки коду. Він збирає й аналізує вихідний код і надає звіти про якість коду проектів. За умови регулярного використання SonarQube може бути налаштований під

стандарти написання коду в організації та допомогти в знаходженні вразливостей на самих ранніх етапах створення коду.

SonarQube оцінює код за набором правил, які називаються профілями якості. Для профілів можна встановити глобальні параметри за замовчуванням або їх можна унікально налаштувати для певної мови чи проекту. Рівні критичності показують, наскільки важливе правило, яке було порушено. SonarQube також оцінює код за набором критеріїв, які називаються воротами якості. Ці показники можна налаштувати на основі профілю якості, за проектом або встановити глобальні значення за замовчуванням. Глобальні параметри за замовчуванням включають ремонтопридатність, надійність, безпеку, покриття коду та дубльовані рядки.

Використання всіх вищенаведених інструментів дозволяє оцінити рівень безпеки веб-сайту, при цьому більшість інструментів можуть бути замінені лише на свої прямі аналоги, адже кожний із згаданих інструментів і технологій покривають різні аспекти функціоналу веб-сайту. Цей факт важливо врахувати при формуванні стратегії тестування і покриття тестування.

2.2 Автоматизація тестування безпеки

Автоматизація тестування безпеки є одним з основних напрямків тестування. На відміну від ручного тестування при якому кожен тест виконується тестувальником при прямій взаємодії з веб-сайтом та веб-додатком, а результати тестування є оцінкою відповідно до певних критеріїв заданих для тесту, автоматизоване тестування є наступним етапом в розвитку тестування, яке дозволяє віддати більшу частину задач тестування спеціалізованому програмному забезпеченню.

Деякі частини тестових завдання можуть бути автоматизовані. На даний момент автоматизація виконання тестів досягла вже високого ступеня. Наприклад, існують фреймворки для модульного тестування, наприклад, JUnit для мови

програмування Java, а також для тестування інтерфейсу користувача. Автоматизація генерації тестів з моделей або вихідного коду не так широко використовується, проте цей напрям активно розвивається.

Метою автоматизації безпеки є виявлення всіх потенційних дефектів безпеки перед випуском проекту шляхом застосування як інструментів безпеки з відкритим кодом, так і систем автоматизованого тестування. Проте автоматизація безпеки не означає повну заміну ручного тестування безпеки. Автоматизація безпеки має на меті зменшити кількість повторного ручного тестування та ефективніше збільшити охоплення тестуванням. Потенційні недоліки безпеки можуть існувати будь-де, від вихідного коду та сторонніх компонентів до незахищеної конфігурації чи вразливої інфраструктури.

Основними перевагами автоматизованого тестування є :

- Зменшення кількості людських помилок. Автоматизація покриває основні повторюванні завдання, які складні для постійного проведення тестувальником. Це дозволяє запобігти багатьох помилкових висновків та пропущених задач тестування;

- Раннє втручання в проект. Розташувавши автоматизованого тестування на самих ранніх етапах життєвого циклу веб-сайту дозволить виявити більшу кількість вразливостей системи та запобігти реалізації цих вразливостей після випуску проекту;

- Спрощене сортування та класифікація вразливостей. Звіти про автоматичне сканування можуть представити рівень загрози будь-якої вразливості, надати відповідну інформацію про неї та автоматично скласти класифікацію існуючих вразливостей та загроз. Це дозволить сконцентрувати ресурси на вирішення саме критичних проблем безпеки;

- Повторні перевірки безпеки. Будь-яке автоматизоване завдання має бути повторюваним, що означає, що після оновлення веб-сайту та його систем, легко можна провести аналогічний тест, щоб впевнитись що проблема з безпекою була

вирішена. Також автоматизований тест легко адаптувати під інший проект і використовувати повторно, не витрачаючи додаткові ресурси на його формування.

Сфера автоматизації включає перевірку коду під час тестування методом білої скриньки , модульне тестування, приймальне тестування, інтеграційне тестування, тестування API та тестування інтерфейсу користувача. З точки зору затрачених ресурсів для проведення, модульне тестування та перевірка білого ящика зазвичай вимагають найменших зусиль, тоді як тестування інтерфейсу користувача часто вимагає найбільших зусиль, особливо для того, щоб зрозуміти бізнес-потік інтерфейсу користувача [17].

Тому більшість випадків автоматизованого тестування виконується за допомогою модульного тестування або тестування на рівні API. Автоматизоване тестування інтерфейсу користувача може охоплювати лише сценарії з точки зору користувача, тоді як тестування API може охоплювати більше випадків використання бізнес-логіки або обробки винятків. Наступна діаграма ілюструє різні рівні автоматизованого тестування та скільки зусиль вони вимагають.

При проведені автоматизованого тестування, тестування безпеки може бути побудовано на його основі. Щоб тестування безпеки було інтегровано зі структурами звичайного автоматизованого тестування, потрібно враховувати вхідні дані для інструментів, підтримку API або CLI для запуску та виконання , бажаний формат звіту про тестування, наприклад JSON, XML або CSV.

Також існують фреймворки для адаптації автоматизованого тестування під уже існуючих компоненти СБ. До них можна віднести:

- Тестування WEB інтерфейсу (фреймворки Selenium та Robot Framework). Може бути використаний в наступних сценаріях [18]:

а) Реєстрації користувача в системі

б) Аутентифікація та авторизація

в) Скидання паролю

г) Проведення оплати на веб-сайті

- Тестування API (JMeter фреймворк). Може бути використаним :

а). RESTful API тестування.

б) Тестування інших типів API

- Фузз тестування (Radamsa фреймворк) . Використовується при:

а) Тестуванні переповнення буферу

б) Тестування різних корисних навантажень при зверненні до веб-сайт

Основними методами проведення автоматизованого тестування є написання спеціалізованих скриптів які будуть звертатися до веб-сайту або перевіряти вихідний код викликаючи певні його елементи під час модульного та інтеграційного тестування. Цей метод вимагає більшої затрати часу та ресурсів, проте він є самим гнучким.

Другий метод це використання спеціалізованого програмного забезпечення, такого як сканери безпеки, яке здатне виконувати частину задач тестування. Сканери вразливостей веб-сайтів – це автоматизовані інструменти, які сканують веб-додатки та веб-сайти, як правило, ззовні, для пошуку вразливостей безпеки, таких як міжкастовий скриптинг, SQL-ін’єкції, впровадження команд, обхід контролю доступу та небезпечна конфігурація сервера. Цю категорію інструментів часто називають інструментами динамічного тестування безпеки . Доступна велика кількість як комерційних, так і відкритих інструментів цього типу, і всі ці інструменти мають свої сильні та слабкі сторони.

Сканери веб-вразливостей працюють шляхом автоматизації кількох процесів. До них входять використання Spider або Crawler, виявлення стандартного та загального вмісту, а також пошук поширених уразливостей.

Існує два основних підходи до сканування вразливостей – пасивний і активний. Пасивне сканування виконує перевірки які не являються прямою атакою на веб-сайт, просто переглядаючи елементи, щоб визначити, чи є вони вразливими.

Активне сканування включає в себе проведення прямих взаємодій з компонентами веб-сайту та реалізації атак на них. До типових дій під час активного тестування можна віднести :

- Тестування аутентифікації — аналіз процесу аутентифікації на веб-сайті, а також спроби використання деякі обхідні шляхи, щоб перевірити, чи вона була реалізована належним чином . Наприклад, аналіз специфіка процесу аутентифікації користувача при початку нової сесії.

- Валідація вводу - це процес перевірки вхідних даних, отриманих програмою, на відповідність стандартам, визначеним у програмі. Це може бути як простим процесом, як чітке введення параметра, так і складним, як використання регулярних виразів або бізнес-логіки для перевірки введення. Існує два різні типи підходів перевірки вхідних даних: перевірка білого списку (іноді називається включенням або позитивною перевіркою) і перевірка чорного списку (іноді відома як виключення або негативна перевірка). Перевірка білого списку — це практика прийняття лише завідомо правильного набору даних. Це може включати перевірку відповідності очікуваному типу, довжині чи розміру, числовому діапазону або іншим стандартам формату перед прийняттям вхідних даних для подальшої обробки. Чорний список — це практика відхилення лише відомого, зловмисно вводу. Зазвичай це включає в себе відхилення вхідних даних, які містять вміст, про який відомо як зловмисний, шляхом перегляду вмісту на наявність кількох «відомо шкідливих» символів, рядків або шаблонів.

- Тестування безпеки API - тестування безпеки API допомагає переконатися, що базові вимоги безпеки виконані, включаючи умови доступу користувача, шифрування та проблеми автентифікації під час використання API. Ідея сканування API полягає в створенні вхідних даних для визначення помилок в їх опрацюванні і невизначеної поведінки API, по суті імітуючи дії та вектори атак потенційних зловмисників. Тестування безпеки API починається з визначення API, який потрібно перевірити. Ці дані надаються сканеру безпеки або визначають тестувальником перед проведенням тесту. Тестувальники надають інформацію про вхідні та вихідні дані API, використовуючи різні формати специфікацій, включаючи OpenAPI v2/v3, Postman Collections і файли HAR. Тести безпеки API використовують цю інформацію для створення нечітких вхідних даних,

адаптованих до вхідних даних, які очікує API. Результатом тестування безпеки API є звіт про будь-які вразливості чи помилки, виявлені під час обробки API.

- Тестування управління сесіями – процес тестування всього функціоналу пов’язаного зі створенням, ідентифікацією, підтримкою та закриттям сесій між користувачам та веб-сайтом. Одним з прикладів з атак на сесію це перехоплення сесії. Перехоплення сесії відбувається, коли ключ сеансу користувача було скомпрометовано. Компрометація ключа сеансу може статися різними способами, здебільшого, якщо файл cookie не має атрибута httponly, тоді до файлу cookie можна отримати доступ за допомогою JavaScript на стороні клієнта.

2.3 Функціонал та порівняння сканерів безпеки для Web

Для розгляду можна взяти три основні найпоширеніші рішення для автоматичного сканування — OWASP ZAP, BURP SUITE, Invicti.

ZAP (Zed Attack Proxy) — це безкоштовний багатофункціональний інструмент із відкритим кодом для тестування безпеки веб-додатків і веб-сайтів. Він відрізняється простотою встановлення та експлуатації, що робить його одним із найкращих варіантів для тих, хто новачок у цьому типі програмного забезпечення. OWASP ZAP доступний для Windows, Linux і Mac OS.

Burp Suite — це інтегрована платформа та графічний інструмент для тестування безпеки веб-додатків та веб-сайтів. Він підтримує весь процес тестування, від початкового відображення та аналізу веб-сайту до пошуку та використання вразливостей безпеки.

Invicti – це автоматизований сканер безпеки веб-додатків із можливістю повного налаштування, який дає змогу сканувати веб-сайти, веб-додатки та веб-сервіси та виявляти недоліки безпеки. Invicti може сканувати всі типи веб-додатків, незалежно від платформи чи мови, на якій вони створені.

Перш за все варто розглянути базовий функціонал сканерів безпеки. Сканери безпеки дозволяють проводити автоматичне або ручне тестування за

різними параметрами вказаними для тестування. Сканування можна поділити на актине та пасивне [19].

Пасивне сканування це сканування при якому сканер безпеки посилає звичайні запити (HTTP або HTTPS) та без додаткові модифікації переглядає та аналізує відповіді від веб-серверу. Пасивне сканування жодним чином не змінює ні запити, ні відповіді, тому безпечне у використанні. Деякі веб-сайти та веб-додатки агресивно реагують на атаки, припиняючи сеанс або блокуючи обліковий запис кожного разу, коли надходить явно зловмисний запит. У цій ситуації ви можете бути обмежені частковим ручним тестуванням, але ви все одно можете використовувати пасивне сканування, щоб виявити ряд проблем, не створюючи жодних затримок або термінуючи все тестування.

Активне сканування намагається знайти потенційні вразливості за допомогою відомих атак на вибрані цілі. Активне сканування – це атака на ці цілі, тому воно може модифікувати пейлоади та реквести, використовувати різні техніки для реалізації атака і збору даних після цього. Такий тип атак дозволяє детально доповнювати HTTP запити.

Всі три представлені рішення підтримують подібний тип сканування, проте в Invciti немає напряму вираженого пасивного та активного сканування, все проводиться налаштуваннями конкретного сканування. Також є підтримка різного типу парсингу, в тому числі API парсингу для веб-ресурсів. Особливістю OWASP ZAP виступає ручне тестування. Цей тип тестування дозволяє використовуючи веб-браузер тестового середовища проводити тестування конкретних елементів веб-сайту, ігноруючи непотрібний функціонал та сторінки. Хоча подібний тип сканування і є затратним по часу і ресурсам, якщо проводити повне тестування веб-сайту через нього, проте він надає гнучкості при виконання менших, конкретизованих задач.

Другою важливою частино функціоналу сканерів є Spider або Crawler. Це інструмент, який використовується для автоматичного виявлення нових ресурсів (URL-адрес) на певному сайті. Він починається зі списку URL-адрес, які потрібно

відвідати, які називаються початковими, які залежать від того, як запущено Spider. Потім Spider відвідує ці URL-адреси, він ідентифікує всі гіперпосилання на сторінці та додає їх до списку URL-адрес для відвідування, і процес продовжується рекурсивно, доки не будуть знайдені нові ресурси. Цей інструмент дозволяє швидка та ефективно створити карту веб-сайту та дає можливість не виконувати ручне додавання нових цілей [20].

Всі три сканера для порівняно підтримують цю функцію, хоча Burp Suite має більш широкі налаштування контексту для інструмента, що дає можливість чіткіше виставляти цілі для роботи з ним. В той же час Invicti надає можливість автоматичного додавання нових доменів до списку відкритих цілей. OWASP ZAP підтримує, як загальний функціонал, так і налаштування контексту. Також є можливість додавання обмеження до певного кластеру веб-сайту з якого і почалась робота інструменту. Це дозволяє гнучко підбирати необхідну частину веб-сайту для тестування [21].

Наступним ключовим елементом для роботи зі сканерами є їх proxy функціонал. Це інструмент, який використовується як посередник для перехоплення всього трафіку між клієнтами та веб-сервером. Проксі можна використовувати як механізм фільтрації вмісту в корпоративному середовищі, наприклад, для захисту конфіденційності. Їх також можна використовувати для налагодження. В сканерах цей інструмент використовується для перехоплення трафіку. Він працює як веб-проксі сервер між браузером і цільовими веб-сайтами та дозволяє перехоплювати, перевіряти та змінювати необроблений трафік, що проходить в обох напрямках. Це є важливою частиною ефективного сканування адже дозволяє глибоко аналізувати трафік та проводити його модифікацію для проведення певного виду атак або збору інформації про відповіді веб-сервера на певний вид трафіку. Хоча всі розглянуті рішення і мають цей функціонал в ньому є ще одна важлива особливість, яка може значно вплинути на якість тестування.

Цією особливістю виступає CA сертифікати для веб-серверу через при підключенні через проксі. Ці сертифікати виконують дві важливі функції при тестуванні веб-сайтів :

- робота з HTTPS запитами через проксі;
- перехоплення та аналіз зашифрованого трафіку.

Робота з HTTPS запитами дозволяє якомога більше наблизитись до реалій функціонування веб-сайту за межами тестового середовища, адже більшість сучасних веб-сайті підтримують цей протокол та мають встановлюй SSL або TSL сертифікат, а більшість сучасних браузерів мають посиленні засоби попередження проти незахищеного HTTP протоколу. Отже використання сертифікату є важливою частиною ефективного тестування.

Друга функція дозволяє створити проксі своє зашифроване з'єднання з клієнтом, яке можна використати для перехоплення та розшифрування трафіку з сервера. Таким чином успішно реалізується атака Man-in-the-Middle.

OWASP ZAP та Burp Suite дозволяють генерувати та завантажувати власні сертифікати для роботи з ними. Це надає можливість проводити всі тестування пов'язані з HTTPS на будь якому веб-сайті. В той же час Invicti значно обмежений в цьому плані адже він надає можливість лише для встановлення існуючих сертифікатів, якщо в цьому є необхідність [22].

Розглянувши самі базові функції можна перейти до більш ситуативного набору інструментів. Важливим для проведення тестування веб-сайтів є фаззинг. Фаз-тестування або фаззинг — це автоматизований метод тестування програмного забезпечення, який вводить у систему недійсні, неправильні або несподівані вхідні дані для виявлення дефектів і вразливостей програмного забезпечення. Інструмент фаззингу вводить ці вхідні дані в систему, а потім відстежує винятки, такі як збої або витік інформації. Простіше кажучи, фаззинг вводить несподівані вхідні дані в систему та спостерігає, чи система має негативні реакції на вхідні дані, які вказують на прогалини або проблеми з безпекою, продуктивністю або якістю.Основна передумова фазз-тестування полягає у введенні в систему

навмисно неправильних вхідних даних для виявлення збоїв. Багато вразливостей на основі введення, таких як впровадження SQL, міжкастовий сценарій і обхід шляху до файлу, можна виявити, надсилаючи різні тестові рядки в параметрах запиту та аналізуючи відповіді програми на наявність повідомлень про помилки та інших аномалій. Враховуючи розмір і складність сучасних програм, виконання цього тестування вручну є трудомістким і трудомістким процесом.

Фаззінг має три ключові компоненти: поет, який створює неправильні вхідні дані або тестові випадки, кур'єр, який доставляє тестові випадки до цільового програмного забезпечення, і оракул, який виявляє, чи стався збій у цільовому програмному забезпеченні. Процес починається з поета, який створює тестові приклади для випробування цільового програмного забезпечення. Тестові випадки можуть бути випадковими, еволюційними шаблонами або поколіннями. Випадковий фаззинг передбачає введення в систему випадкових даних. Еволюційний фаззинг шаблону вносить аномалії в дійсні вхідні дані, а потім отримує відгуки про поведінку системи під час початкових тестів, щоб зробити наступні тести більш ефективними та різноманітними. А генераційні тестові випадки базуються на розумінні протоколу, формату файлу чи API, що тестується, — тести знають правила системи. Через це фазз-тестування поколінь може систематично порушувати всі правила [23].

Далі кур'єр доставляє тестові кейси. Спосіб доставки значно відрізняється залежно від типу фаззингу, який потрібно виконати, але кінцева мета завжди одна: доставити тести до цілі.

Нарешті, оракул визначає, пройшов чи не пройшов тест. Оракул перевіряє цільову систему, щоб побачити, чи сталася будь-яка форма збою. Знати про помилку є критично важливим — без цієї інформації тестувальники не зможуть відтворити помилку, перевірити її та визначити її виправлення.

OWASP ZAP підтримує наступні можливості для фазингу:

- Вбудований набір корисних навантажень;
- Корисне навантаження, визначене додатковими додатками;

- Спеціальні скрипти.

Burp Suite такоє має вбудований функціонал для фаззингу проте він надає більшої варіативності та простоти в налаштуванні. Invicti також підтримає базовий функціонал для проведення фазінгу.

Ще одною цікавою особливістю виступає декодер. Цей функціонал можна знайти лише в OWASP ZAP та Burp Suite. Це простий інструмент для перетворення закодованих даних у канонічну форму або для перетворення необроблених даних у різні закодовані та хешовані форми. Він здатний інтелектуально розпізнавати декілька форматів кодування за допомогою евристичних методів.

ZAP дозволяє декодувати та кодувати: Base 64 URL, full URL, ASCII, hex encode, HTML, Javascript.

Burp Suite: HTML, Base64, ASCII hex, hex, octal, binary, gzip.

Це дозволяє отримати додаткові дані в ході проведення тестування безпеки.

Звітність в всіх рішеннях є автоматизованою. Є можливість імпорту всієї отриманої під час проведення сканування інформації в HTML, XML та PDF файл для подальшого використання.

Якщо брати покриття тестування, то OWASP ZAP здатен ефективно знаходити всі найпоширеніші загрози для web-сайтів. Тестування ним надає високі показники true positive спрацювання при виявленні різних типів ін’єкцій, і високі, понад 70 відсотків, під час виявлення міжсайтовго скриптингу [24].

Burp Suite ефективно працює з найпоширенішими типами загроз та має високі показники. Особливо добре він демонструє результати при знаходженні невірних конфігурацій безпеки та загроз пов’язаних з витоком конфіденційної інформації. Всі інші показники по основним загрозам з OWASP TOP 10. Серйозним недоліком можна назвати високий відсоток невірних спрацювань, 51%. Цей фактор хоч і не є критичним в цілому, проте додає роботи тестувальнику і підвищує затратність часу та ресурсів на проведення тестування. Ще одним значним його плюсом є широка база вразливостей які він здатен протестувати

Invicti серед всіх порівнюваних технологій має найнижчі показники по знаходженню загроз, проте 91% точності при їх визначенні. В загалом він також здатен знаходити основні загрози для веб-сайтів та є ефективним рішенням для автоматизації тестування.

Якщо розглядати ціну, то OWASP ZAP має відкритий код, тому може бути використаний або змінений, модифікований без обмеження. Це надає перевагу при обмеженості ресурсів для проведення тестування та сама технологія використовується в великій кількості проектів. Burp Suite має не комерційну версію з певними обмеженнями в функціоналі та повну версію , ціна якої залежить від кількості користувачів. Invicti хоча і має подібну до Burp Suite систему, проте мінімальна ціна цього рішення набагато вища, в зв'язку з тим що не має можливості його реєстрація на одного користувача.

Розглядаючи можливість розширення базового функціоналу рішень, можна відзначити , що OWASP ZAP через свою доступність та відкритий код має велику базу розширення. Розширення Zap – це пакети Java, які розширюють існуючі функції OWASP ZAP. Цю концепцію можна назвати «Механізмом розширення», який забезпечує стандартний спосіб створення функцій або API до Java-додатків. Розширення створені розробниками або користувачами можна знайти в спеціально створеному ZAP Marketplace. Це надає програмі адаптуватися до специфіки певного тесту або проекту,

Burp Suite також має функціонал по створенню розширень для додавання нового функціоналу, модифікації проведення тестів або конкретних дій під час тестування, проте кількість доступних розширень обмежена. Invicti не має функціоналу для створення розширень або інтеграції.

Документація по функціоналу та рекомендації щодо налаштування певних елементів сканера OWASP ZAP можна знайти, як в вигляді коротких статей від розробника, які описують загальний функціонал, налаштування та можливості сканера, так і в не офіційній документації по його використанню. Також існує

велика база інформації, щодо використання API та інтеграції технології з іншими рішеннями щодо тестування та безпеки.

Burp Suite та Invicti мають схожий підхід до формування документації, проте база знань про API або додатковий функціонал доволі обмежена.

Останнім пунктом для розгляду є інтеграція технологій з іншими рішеннями. Подібний функціонал дає гнучкості та адаптивності при проведенні тестування, а одже є важливим фактором при виборі рішення. Основний підхід до формування можливості інтеграції є використання API , звернення до якого через вихідний код іншого програмного продукти або скрипту є інтеграцією з цим продуктом.

OWASP ZAP має функціонал для надання своїх основних можливостей через API , яке можна використати при написанні будь якого коду на мові програмування Java. ZAP надає інтерфейс прикладного програмування (API), який дозволяє програмно взаємодіяти з ZAP. API доступний у форматах JSON, HTML і XML.Сканер дає використати та збирати данні зі своїх елементів .таких як Spider та збирати даних з них в форматі файлів, які згадано вище. За замовчуванням лише машина, на якій працює ZAP, має доступ до API, проте є можливість дозволити іншим машинам, які можуть використовувати ZAP як проксі, доступ до API.

Burp Suite Enterprise Edition надає два API, які можна використовувати для взаємодії з системою з іншого програмного забезпечення сторонніх виробників. GraphQL API є основним, в ньому зосереджений весь доступний функціонал. REST API обмежений в можливостях і використовується для надання спрощених можливостей по інтеграції. На відміну від OWASP ZAP, API має систему користувачів які можуть його використовувати. Кожний користувач має генерувати окремих ключ для його використання. Цей ключ виступає в ролі ідентифікатора [25]. Invicti API дозволяє клієнтським програмам переглядати та керувати завданнями сканування, переглядати проблеми, створювати агенти сканування. API виконано в форматі REST, його основний функціонал доступний через HTPP

запити. Також підтримується система користувачів для кожного з яких генерується спеціальний токен, який дозволяє проводити звернення до АР.

Таблиця 2.1.

Порівняння рішень для автоматизованого тестування безпеки

Технологія	OWASP ZAP	Burp Suite	Invicti
Сканування	Passive / Active	Passive / Active	Інтегрований підхід з налаштуваннями
Spider	Так	Так	Spider та парсинг
Проксі	Так	Так	Так
CA сертифікати	Можливість генерування	Можливість генерування	Без можливості генерування
Фаззінг	Так	Так	Так
Кодування та декодування	З підтримкою більшої кількості типів файлів	З підтримкою більшої кількості кодування	Ні
Звітність	Базовий функціонал	Базовий функціонал	Базовий функціонал
Покриття тестування	Покриває всі найпоширеніші загрози, показних точності вище середнього	Покриває всі найпоширеніші загрози, середній показник точності	Покриває всі найпоширеніші загрози, має високий показник точності
Знаходження вразливостей	Найпоширеніші вразливості	Широка база вразливостей	Найпоширеніші вразливості
Ціна	Open Source	Від $449.00	Від $4500.00
Розширення	Широкий вибір розширень	Базові розширення	Відсутні
Документація	Широка база документації	Документація від розробника	Документація від розробника
Інтеграція	Підтримує	Підтримує	Обмежено підтримує

Таким чином розглянувши технології для проведення автоматизованого сканування безпеки та провівши порівняно їх функціоналу та характеристик можна сказати, що Inicti хоч і підтримує всі необхідні функція для автоматизації тестування безпеки та має високі показники точності при знаходженні загроз значно поступається іншим представленим рішенням, особливо в плані інтеграції та підтримки розширення.

Burp Suite має найбільш широкий базовий функціонал, широку базу вразливостей, які він здатен перевірити. Також його розширення та інтеграція значно підвищує його ефективність при тестуванні. Основним недоліком цього рішення є середній показник точності при знаходженні загроз та певні обмеженості в документації. Також цінова модель направлена на оформлення підписки на користувача додає ресурсозатратності цьому рішенню.

OWASP ZAP має широкий базовий функціонал, значні можливості по інтеграції та розширенню, що дозволяє легко адаптувати його під специфіку певного проекту. Основними його недоліками є певна обмеженість в можливостях та базі загроз в порівнянні з іншими рішеннями, проте його відкритий вихідний код та користувацькі розширення здатні компенсувати цей недолік, а через свою доступність, відсутня додаткова ресурсозатратність при його використанні.

Висновки до розділу 2

Досліджено основі інструменти — програмні засоби та фреймворки для проведення тестування.

Проаналізовано процес автоматизованого тестування та порівняно його з ручним тестуванням. Виявлено, що більшість задач під час тестування можуть бути автоматизовані через використання скриптів та спеціалізованих сканерів вразливостей, або іншим схожих засобів.

Виокремлено рішення для проведення автоматизованого тестування та проведено їх порівняння за рядом параметрів.

Підкреслено можливості OWASP ZAP для адаптації до специфіку проекту або окремих тестів, через свої можливості для інтеграції, розширюваності та низьку ресурсозатратність.

3 ПРОЦЕС ТЕСТУВАННЯ НА ОСНОВІ OWASP ZAP

3.1 Процес проведення тестування

Перед початком тестування необхідно визначити ціль тестування та зробити першочергові налаштування. Ціллю даного тестування є пошук на веб-сайті основних вразливостей з актуальної версії OWASP TOP 10.

Переш за все, для реалізації повного функціоналу необхідне налаштування проксі-серверу. Це реалізується доволі просто через задання IP адреси та порту. Аналогічна конфігурація має бути додана на хості або лише для відповідного браузера. При конфігурація вибирається HTTPS порт 8080 (також є можливість задати будь-який інший порт при необхідності цього для конфігурації). Для даного тесту немає необхідності в конфігурації додаткових проксі, хоча ця можливість присутня в самому сканері безпеки. Оскільки сканер та тестове середовище розміщені на одній локальній машині, видається 127.0.0.1 адреса, яка дозволить звертатися до мережевого інтерфейс цієї ж машини.

Також даний інтерфейс дає можливість створення шаблонів налаштування проксі, які можуть бути використані в різних тестових середових або для одночасного використання декількома хостами. Хоча основна конфігурація зазвичай включає в себе лише loop-back адреси, є можливість конфігурації проксі для хостів які знаходяться в мережі, але не мають встановленого та налаштованого OWSAP ZAP.

Варто помітити, що конфігурація назначає вибраний порт як для HTTP, так і для HTTPS протоколу без можливості їх розділення.

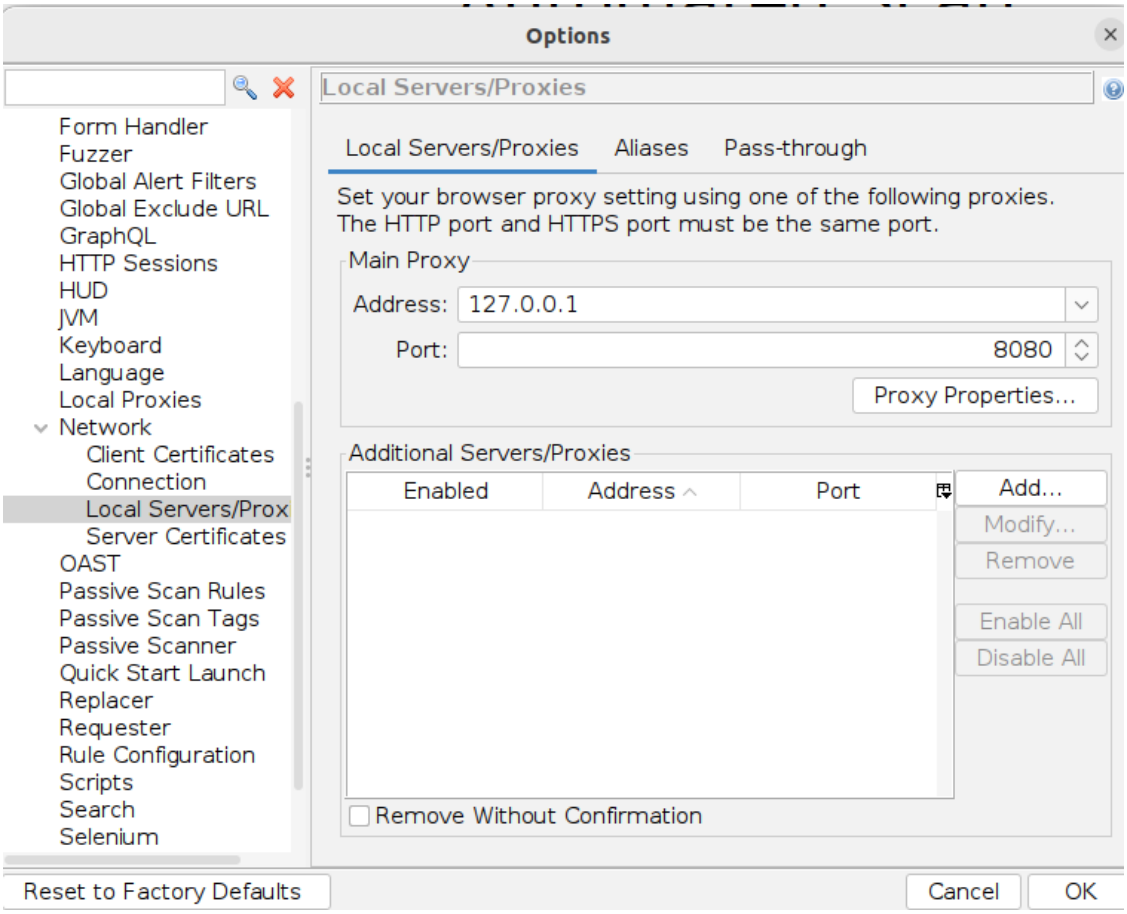


Рис. 3.1. Налаштування проксі-сервісу

Після цього проводиться за необхідністю додаткові налаштування. Тут є можливість вибрати протоколи для SSL. Серед цих налаштування вибираємо останні версії TSL, для більшого покриття окрім 1.3 додаємо попередню 1.2. За рахунок цього наше проксі-з’єдання зможе підтримувати CA сертифікати і безпечне з’єднання яке формується через них. Серед інших налаштувань вимикається опція «Behind NAT», оскільки тестове середовище не налаштоване з його використанням. Інші налаштування залишаються за замовчуванням, для оптимальної підтримки роботи проксі.

Наступним кроком є генерація та експорт CA сертифікату для встановлення htpps з'єднання з OWASP ZAP під час тестування. В даному випадку буде використано сертифікат з 365 днями валідності, з можливістю його використання в наступних тестуваннях без зміни сертифікату. Цей сертифікат має бути

встановленим на рівні операційної системи хосту або в самому браузері для його використання.

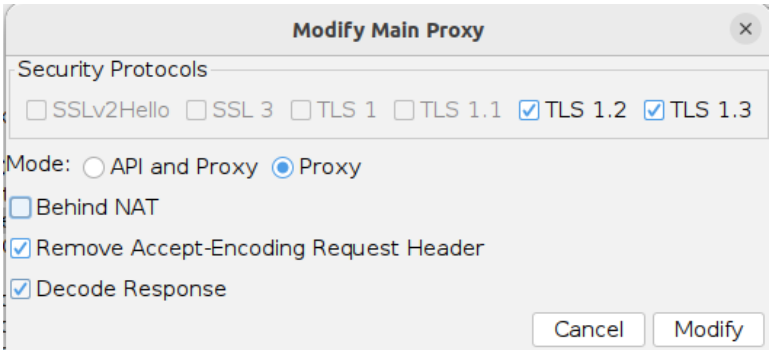


Рис.3.2. Додаткові налаштування проксі

Як додаткове налаштування, цей сертифікат буде використаний як для веб-сайтів, так і для ідентифікації електронної пошти. Це дозволить розширити його можливості в майбутніх тестуваннях. Як можна помітити з екрану налаштувань, є можливість переглянути текстовий варіант сертифікату перед його імпортом. Також є опція перегляду вже згенерованих сертифікатів. Якщо тестування не потребує використання CA сертифікатів для успішного проведення, або немає прямої необхідності в використанні проксі, OWASP ZAP з певними обмеженнями в можливості отримання тестових результатів , може бути використаний.

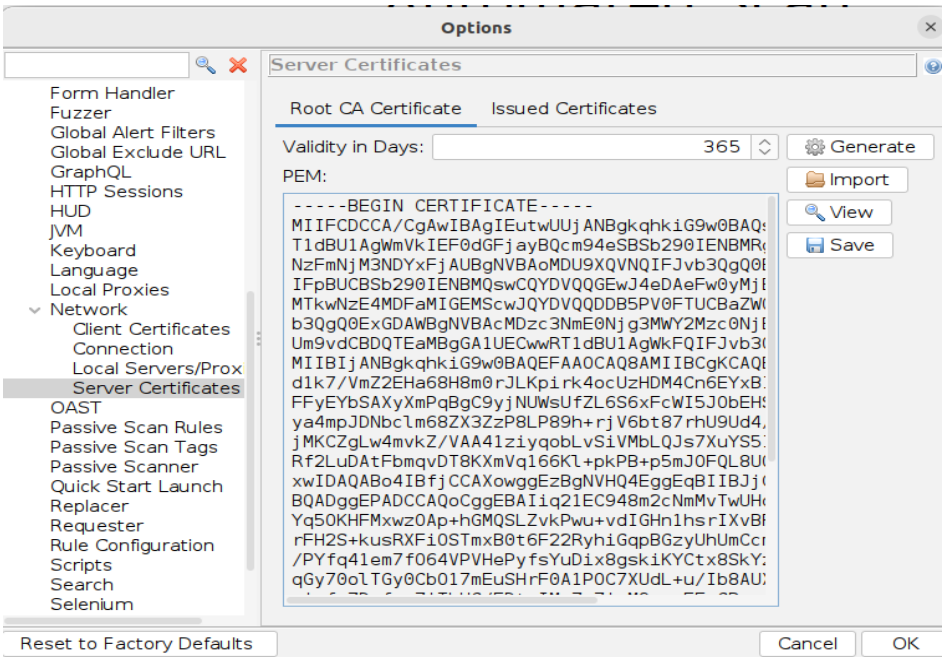


Рис. 3.3. Генерація сертифікату

Після налаштування сертифікату його необхідно завантажити в форматі .cer файлу, який підтримується всіма сучасними операційний системами та браузерами. Інформація про згенерований сертифікат зберігається в сканері, проте його можна зробити не валідним до закінчення терміну придатності просто виконавши генерацію нового сертифікату.

Варто помітити що використання подібного сертифікату і його встановлення поза межами тестового середовища несе певні загрози безпеці, а одже варто запобігати виконанню налаштувань CA сертифікату на активних хостах КМ. Також є можливість видалення вже встановленого сертифікату для запобігання компрометації тестового середовища після завершення тестування.

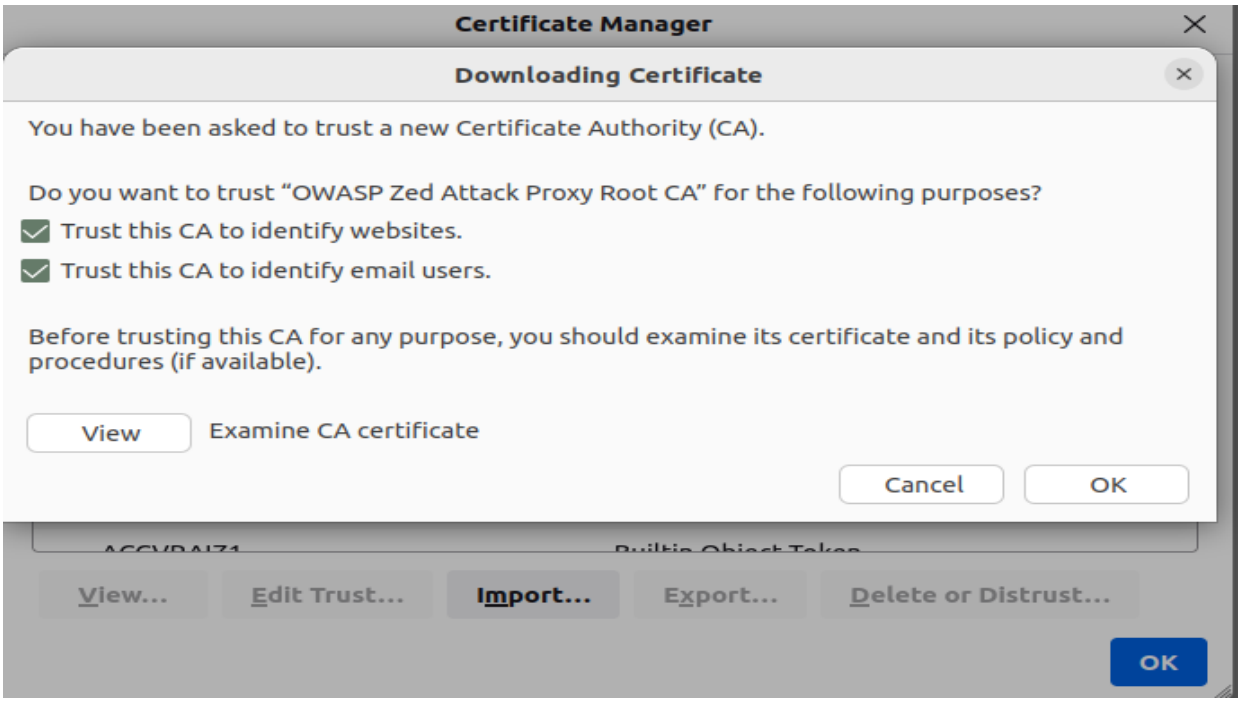


Рис. 3.4. Додаткові налаштування та завантаження сертифікату

Є можливість прямого перегляду даних, які додані до сертифікату при необхідності в їх використанні. Як можна помітити сертифікат є само підписаним корінним сертифікатом, тому його використання можливе лише при прямому встановленні. В його даних не вказано конфіденційної інформації організації, замість цього всі поля заповнені автоматично генерованими деталями з вказанням OWASP та OWASP ZAP ROOT CA.

В цьому інтерфейсі можна переглянути ключі для сертифікату та його основні параметри. Сертифікат має статус «Critical», не має обмежень по шляху який може бути проведений до нього проте сам OWASP ZAP не підриває функцію під створенню та підписанню інших сертифікатів для формування їх мережі. Сам ключ сертифікату по своїм вказаним характеристикам може бути використаним як електронний цифровий підпис, для шифрування ключів і даних, а також по розширеним параметрам для аутентифікації клієнту та серверу.

Ці данні неможливо редагувати в базовому функціоналі OWASP ZAP, єдиний параметр який піддається прямій зміні це термін дії сертифікату, та можливість пере генерування його з новими ключами.

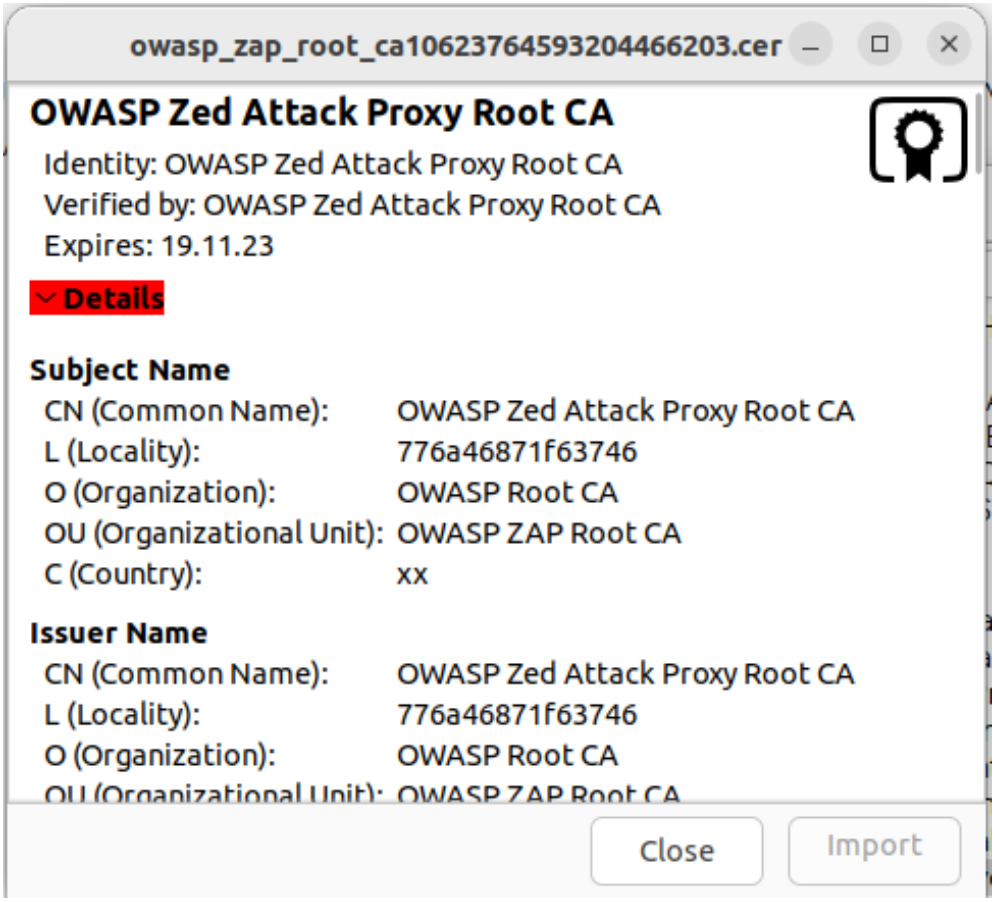


Рис. 3.5. Перегляд сертифікату

Після завантаження файлу сертифіката він має бути встановленим для його подальшого використання.

Після цього можна перейти до налаштування Spider. Встановлення глибини сканування до 10, підвищується кількість потенційних сторінок які можуть бути знайдені. Для цієї ж цілі підвищується кількість похідних сторінок до 5. Подібна конфігурація дозволить зібрати достатньо даних без перенавантаження ресурсів.

Параметр «Maximum Duration» встановлюється на 0 для запобігання ситуацій невірного формування картини веб-сайту через повільні відповіді віб веб-сервера або значне навантаження на одній з веб-сторінок.

Параметр «Maximum Parse Size» обмежує розмір веб-сторінки та її елементів над якими буде проводиться парсинг. Всі згадані параметри можуть сильно підвищити час та ресурсозатратність проведення тестування, проте це надасть найбільш повну картину веб-сайту, а специфіка даного тесту немає встановлених обмеження на ресурси. Якщо подібні обмеження існують, налаштування Spider можна використати для протилежної цілі — оптимізації проведення тесту. Якщо відома технічна специфіка веб-сайту і є інформація про максимальний розмір файлів на веб-сторінках, а також

Серед інших налаштувань вибираються всі можливі пункти. Особливу увагу необхідно приділити cookie-файлам, Файл cookie — це частина даних веб-сайту, яка зберігається у веб-браузері, яку веб-сайт може використати в подальших сесіях. Файли cookie використовуються, щоб повідомити серверу, що користувачі повернулися на певний веб-сайт. Коли користувач повертається на веб-сайт, файл cookie надає інформацію та дозволяє сайту відображати вибрані налаштування та цільовий вміст. Наданню Spider параметру, який буде вказувати на прийняття cookie-файлів дозволить оптимально та в повній мірі переглядати веб-сторінки. Якщо не надати це налаштування, певні елементи об’єкту тестування можуть бути некоректно оброблені або надані від веб-серверу не в повній мірі.

Ще одним важливим пунктом є robots.txt парсинг. Файл robots.txt — це лише текстовий файл без коду розмітки HTML (звідси розширення .txt). Файл robots.txt розміщується на веб-сервері, як і будь-який інший файл на веб-сайті. Зазвичай цей файл доступний із корінної директорії веб-сайту. Файл не містить жодних

посилань на сайті, тому користувачі навряд чи натраплять на нього, але більшість Spider або Crawler програмних засобів шукатимуть цей файл, перш ніж сканувати решту сайту. Файл містить інструкції по подальшому скануванню веб-сайту та може вказати певні обмеження. Хоча файл robots.txt містить інструкції для ботів, насправді він не може забезпечити їх виконання.

Зазвичай бот, наприклад веб-сканер або бот стрічки новин, спробує спочатку відвідати файл robots.txt перед переглядом будь-яких інших сторінок у домені та виконуватиме вказівки, проте деякі боти ігнорують файл robots.txt, або обробляють його для знаходження заборонених веб-сторінок. Робот веб-сканера слідуватиме найдеталізованішому набору інструкцій у файлі robots.txt. Якщо у файлі є суперечливі команди, бот виконуватиме більш детальну команду. Важливо зауважити, що для всіх субдоменів потрібен власний файл robots.txt.

Spider OWASP ZAP обробить цей файл, при умові його існування на домені або субдомені, щоб отримати інформації про інструкції які веб-сайт хоче, щоб були виконані при його обробці та для знаходження заборонених веб-сторінок. Сам Spider не буде слідувати інструкціям з файлу, адже це лише обмежить результати сканування.

Наступним пунктом який буде корисним для сканування є обробка sitemap.xml файлу. XML Sitemap — це спеціальний документ, який містить перелік усіх сторінок веб-сайту, щоб надати пошуковим системам огляд усього доступного вмісту. Як і в прикладі з robots.txt цей файл не буде виступати як повна інструкція для Spider по скануванню веб-сайту, проте він дозволить отримати додаткову інформацію про структуру веб-сайту та прискорить роботу самого Spider.

Заголовок HTTP-запиту Referer містить абсолютну або часткову адресу, з якої на ресурс зроблений запит. Заголовок Referer дозволяє серверу ідентифікувати сторінки переходу, яка була відвідана до отримання реквесту, або де використовуються запитувані ресурси. Ці дані можна використовувати для аналітики, журналювання, оптимізованого кешування тощо для самого веб-

сервера. В сценарії сканування через Spider ці дані надають можливість оптимізувати передачу даних від сканера до сервера та отримати більше інформації про структуру веб-сайту.

Інші використані параметри також нададуть додаткові данні про веб-сайт проте вони не є настільки важливими. Захоплення форм може додати декілька веб-сторінок до результатів сканування, проте результати можуть бути не релевантними для тесту в цілому. Метадані та коментарі в HTML лише виводять інформацію створену розробником до результатів тесту.

Наступною важливою групою налаштувань є налаштування політик сканування. Основними параметрами тут виступає поріг виділення загрози та сила сканування. Для максимального отримання результатів , виставляється низький поріг. Це дозволить виділяти загрози, які сам сканер відніс до категорії менш важливих або критично, проте подібні данні можуть бути корисні для тесту в цілому. Сила або інтенсивність атаки виділяє ресурси, які сканер буде приділити на певний тип атаки. Всього виділяється п’ять категорій, проте через специфіку задачі тестування параметри будуть додані до всіх типів.

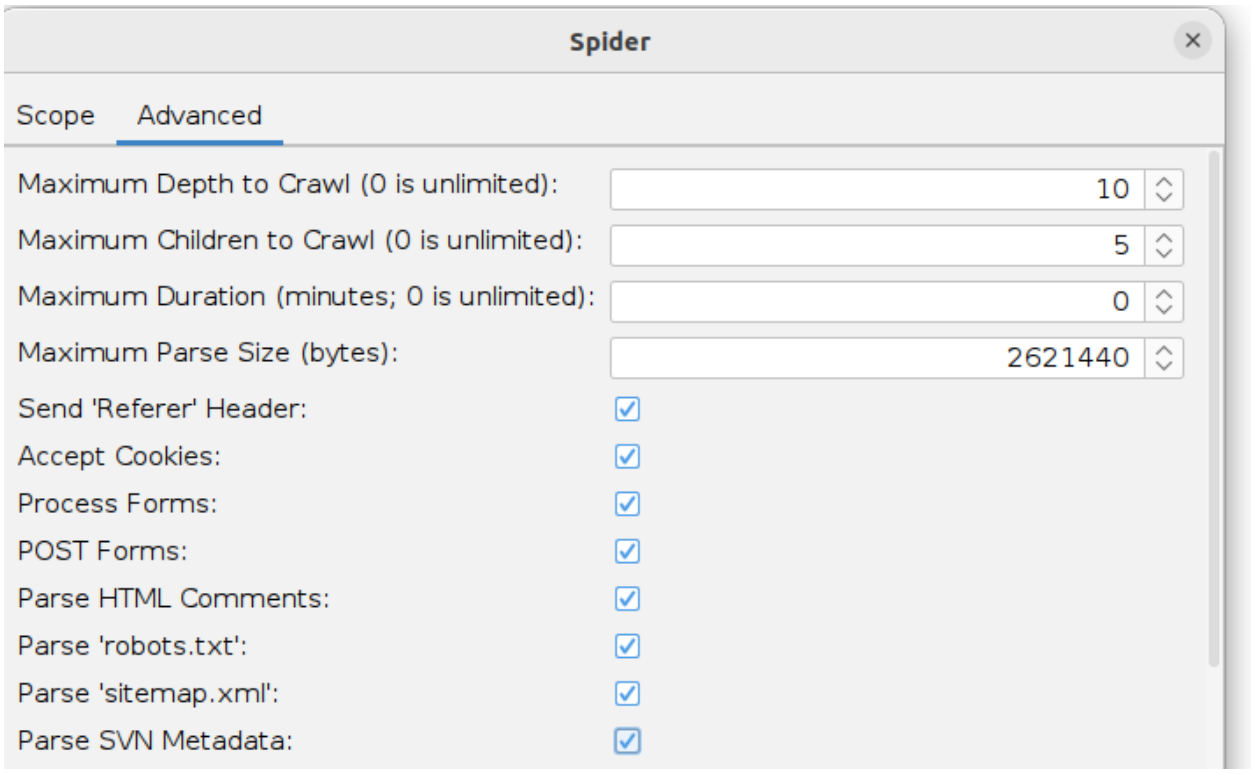


Рис. 3.6. Налаштування Spider

Загалом сканер дозволяє налаштовувати та підривати декілька політик одночасно і підбирати оптимальну для конкретного тесту, який буде проводитися. Це дозволить проводити ряд тестів без постійної зміни конфігурації та підтримувати певну стандартизацію процесу тестування веб-сайту.

При необхідності доповнення або конкретизації політики, є можливість її імпорту та експорту. Це дозволяє запобігати варті створених конфігурацій для відповідних політик. Також подібна функція надає можливість передачі сконфігурованих політик між декількома тестовими середовищами, без необхідності повторного налаштування.

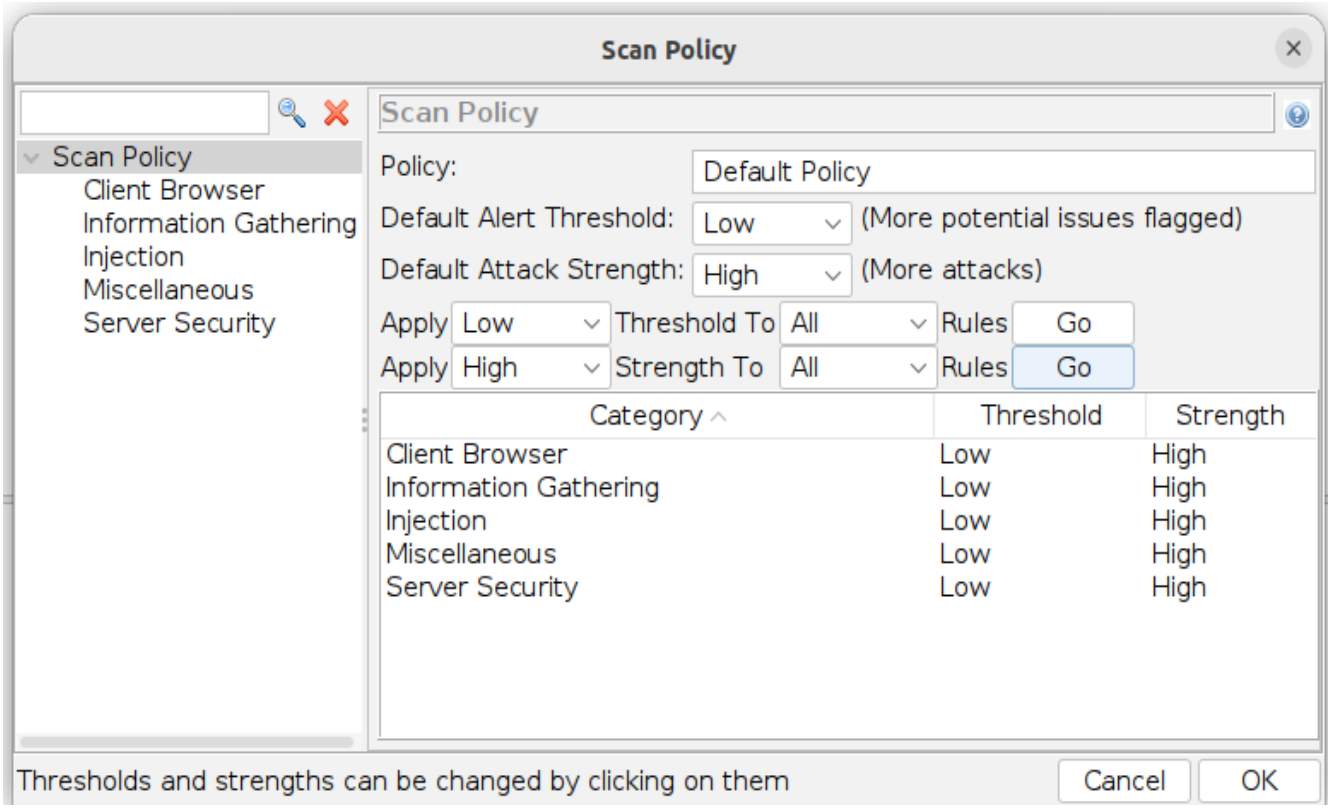


Рис. 3.7. Налаштування політик сканування

Додатково є можливість налаштування ще одного Spider – AJAX Spider. Розширення AJAX Spider інтегрує в ZAP сканер сайтів із технологіями AJAX під назвою Crawljax. Його можна використовувати його для ідентифікації сторінок цільового сайту. Ви можете поєднати його з звичайним Spider для кращих результатів.

Його налаштування подібні до налаштування Spider, проте є певні додаткові параметри, такі як кількість вікон, що дозволяє прискорити його роботу, проте викличе більші затрати ресурсів. Також в AJAX налаштуваннях можна визначити параметри очікування для завантаження веб-сторінок.

Додатково можна надати параметри для інтеракції з веб-сторінками. Для цього тесту будуть вибрані використання випадкових значення для заповнення форм та вимкнене написання лише звичайних елементів. Замість цього вибрані всі доступні елементи веб-сайту для отримання найбільш повних результатів тестування.

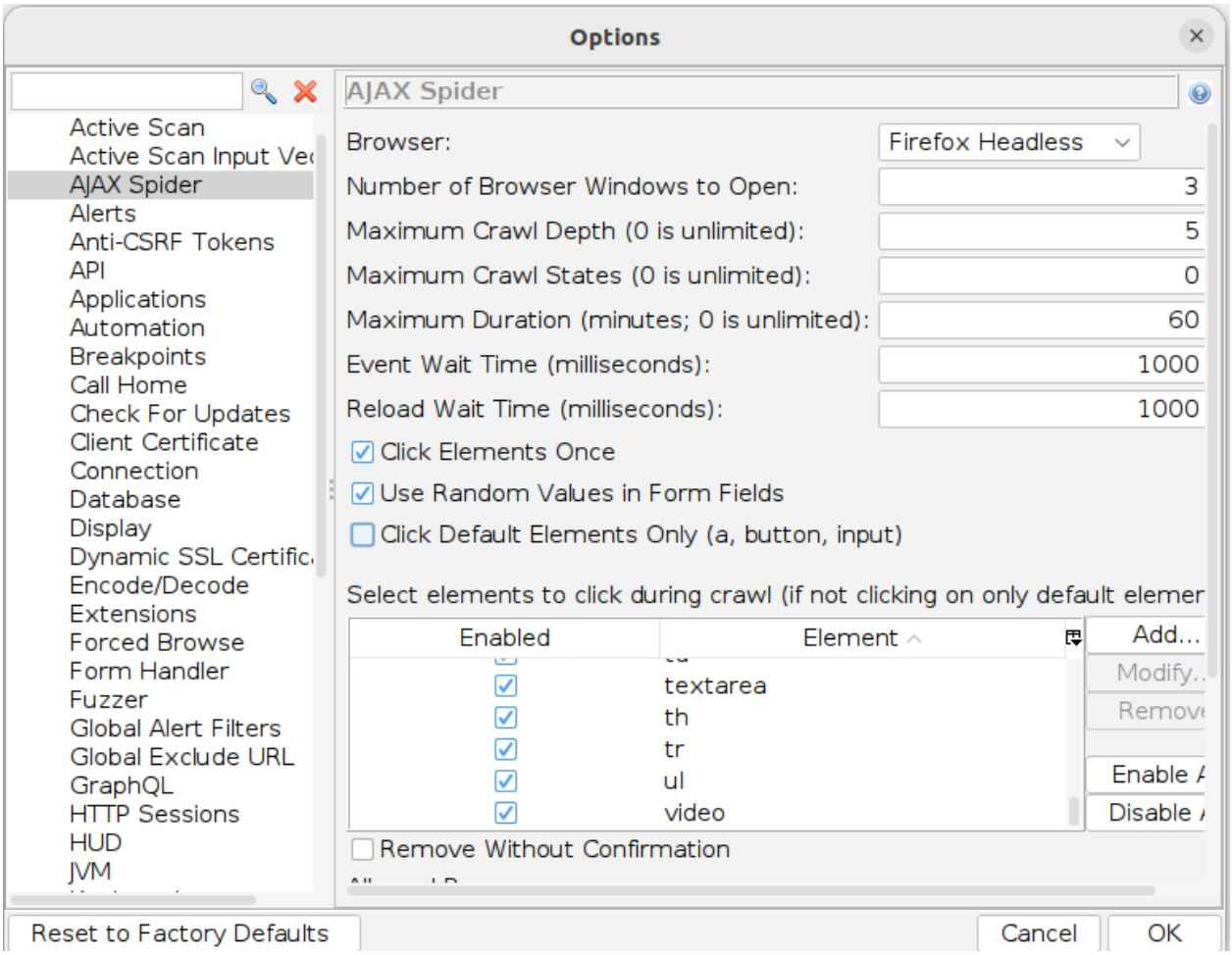


Рис. 3.8. Налаштування AJAX Spider

Наступним кроком є вибір цілей для сканування та задання параметрів сканування. На даному етапі вказується ціль сканування в вигляді посилання. Підтримуються як доменні імена , так і IP адреси. Також є можливість виділення

конкретної директорії, як точки початку атаки. В даному випадку для тестування вибрана конкретна частина веб-сайту з якої буде починатися атака і сканування через Spider.

Також ця функція дозволяє проводити повторні атаки, після збору даних, конкретизуючи директорії, веб-додатки тощо. Для початку сканування також є можливість вибору ручного режиму. Ця функція дозволить точно, використовуючи інтерфейс веб-браузера вибрати елементи для сканування, проте подібний функціонал не входить до автоматизованого сканування.

Важливим налаштуванням є Spider. Тут конкретизується чи потрібно функціоналу Spider використовувати один з наших дійсно встановлених веб-браузерів, або надати налаштування для емуляції діяльності цього веб-браузера.

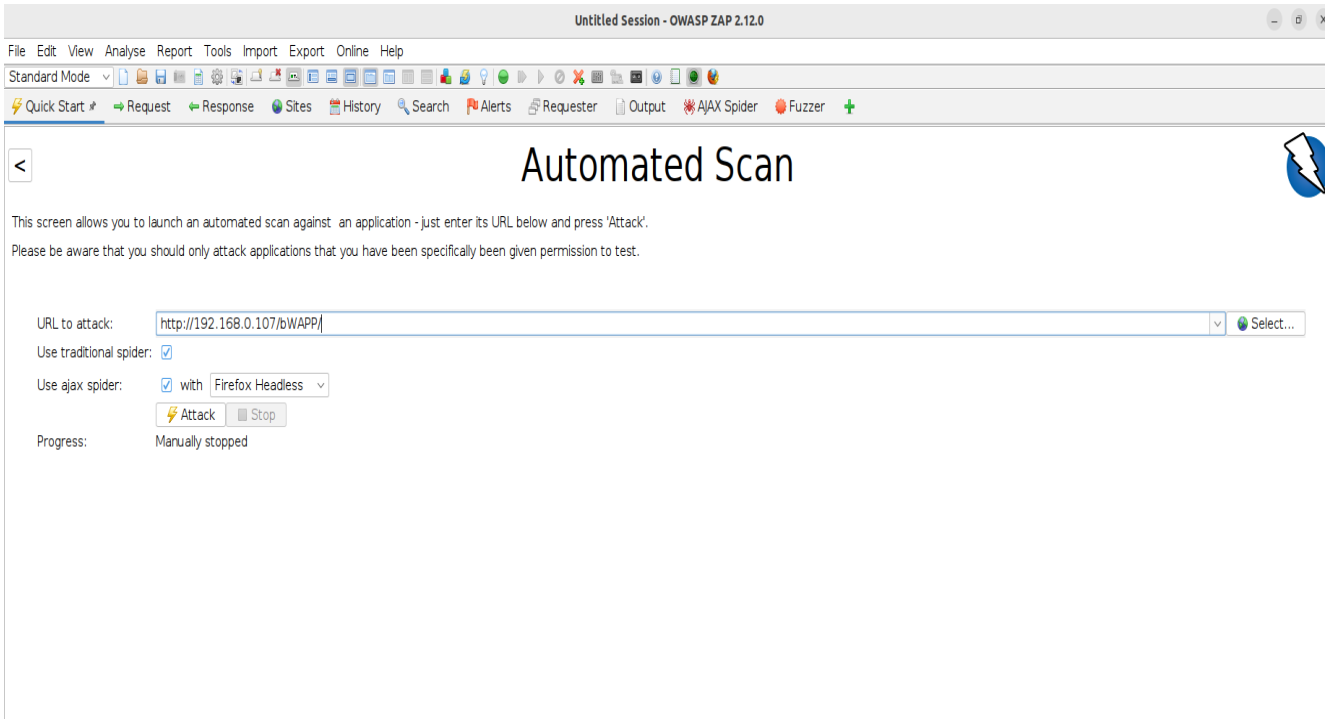


Рис. 3.9. Початок сканування

Таким чином отримується інформація про перші знайдені вразливості в веб-сайті. Основне джерело інформації для цього — інтерфейс «Alerts». Тут сканер виділяє критичність загрози, дає її загальний опис та класифікацію. Також тут

знаходиться CWE ID, яке дозволить детальніше проаналізувати виявлену вразливість.

Якщо брати детальну інформацію, то можна знайти серію запитів яка була виконана при знаходженні даної вразливості. Як можна помітити з даної вразливості , їй був наданий середній рівень критичності, вона посилається на CWE 1021 і присутнє вказання директорії в якій була знайдена ця вразливість.

Дана вразливість — невірне встановлення обмежень на обробку рівнів інтерфейсу або кадрів. Веб-сайт не обмежує або неправильно обмежує об’єкти фрейму або рівні інтерфейсу користувача, які належать іншій програмі чи домену, що може призвести до плутанини користувача щодо того, з яким інтерфейсом він взаємодіє. Це може бути використано зловмисником для проведення атак на користувачів веб-сайту.

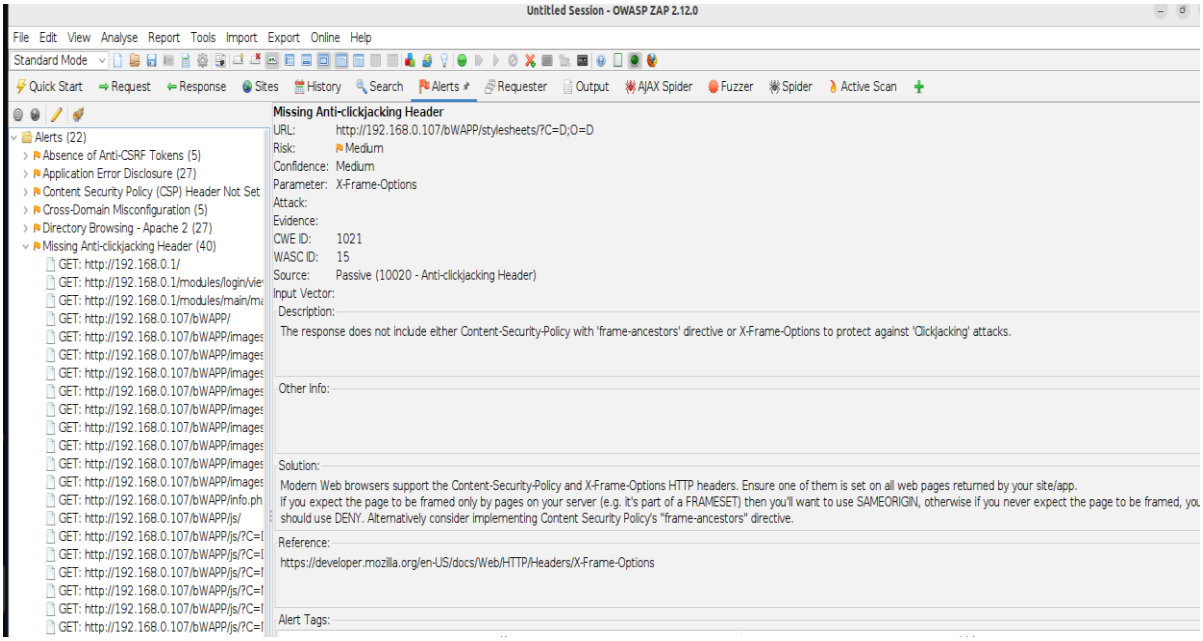
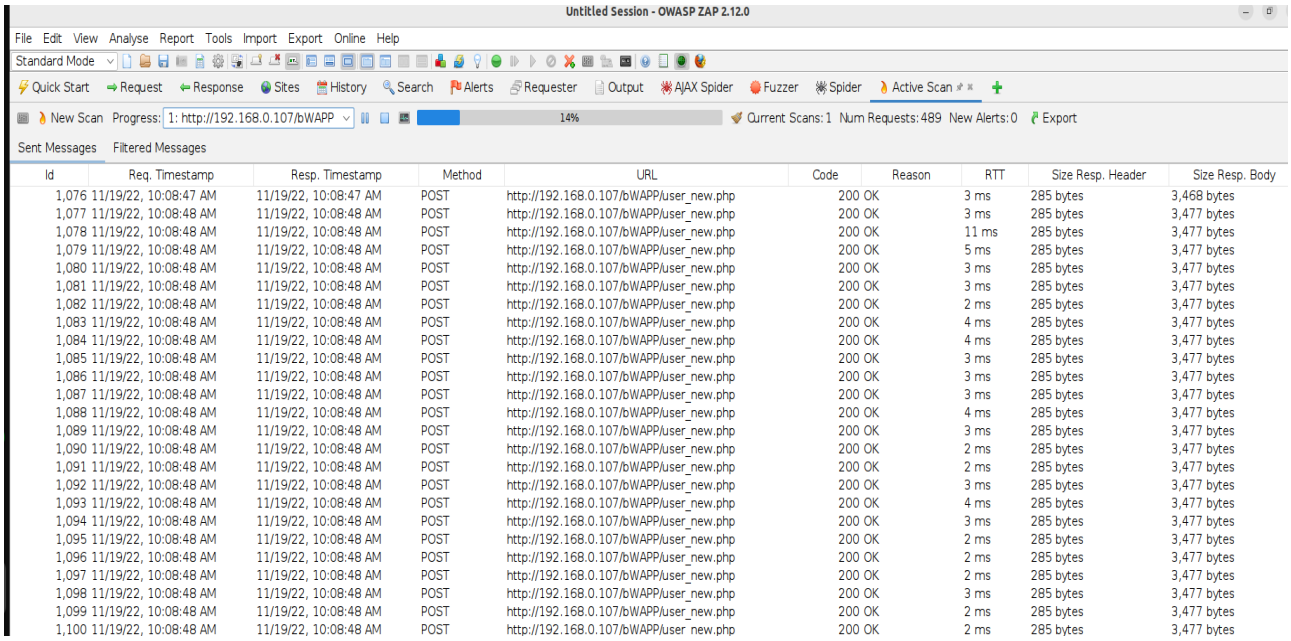


Рис. 3.10. Огляд знайдених вразливостей

Також є можливі відслідковування активного сканування в прогресії та проаналізувати лог виконаних сканером дій. Сам сканер виділяє декілька характеристик кожної дії. Перш за все для логування виділяється унікальний числовий ідентифікатор кожної виконаної сканером дії і час виконання даної дії.

Другою важливою характеристикою виступає метод, який був використаний. Це може бути один із методів, які використовую протокол HTTP для своїх запитів. Наступним полем виступає URL — повне посилання на директорію або файл з яким виконуються певні дії. Як можна помітити з даного фрагменту проведення тестування, сканер може повторно робити запити до одного файлу під час своєї роботи для отримання необхідних результатів.

Одною з характеристик кожної дії є HTTP код, який був повернутий після виконання запитів. Якщо запит був успішним всі коди будуть починатися з 20, останнє число буде характеризувати формат виконання запиту. В даному прикладі всі запити мають код 200 — успішне виконання запиту.



Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
1,076	11/19/22, 10:08:47 AM	11/19/22, 10:08:47 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,468 bytes
1,077	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,078	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	11 ms	285 bytes	3,477 bytes
1,079	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	5 ms	285 bytes	3,477 bytes
1,080	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,081	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,082	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,083	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	4 ms	285 bytes	3,477 bytes
1,084	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	4 ms	285 bytes	3,477 bytes
1,085	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,086	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,087	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,088	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	4 ms	285 bytes	3,477 bytes
1,089	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,090	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,091	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,092	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,093	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	4 ms	285 bytes	3,477 bytes
1,094	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,095	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,096	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,097	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,098	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	3 ms	285 bytes	3,477 bytes
1,099	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes
1,100	11/19/22, 10:08:48 AM	11/19/22, 10:08:48 AM	POST	http://192.168.0.107/bWAPP/user_new.php	200	OK	2 ms	285 bytes	3,477 bytes

Рис.3.11. Дії сканера під час тестування

Також є можливість відслідкувано активності Spider і результатів виконання його роботи. Інтерфейс подібний до минулого, проте тут в першу чергу виділяється метод, який був використаний під час роботи та успішність виконання запиту. На відміну від відслідкувано дій сканеру, успішність викання позначається простим індикатором, без специфікації коду, який повернув запит. З цього інтерфейсу можна відслідковувати прогрес роботи Spider та приблизно аналізувати структуру веб-сайту, проте більшість знайдених даних будуть

автоматично додані до цілей тестування і загалом немає особливої потреби в їх зборі через цей інтерфейс.

Деякі з результатів мають мітку «Out of scope». Це позначення того, що знайдене посилання відноситься до іншого веб-сайту, веб-додатку або веб-ресурсу і оскільки він не включений в цілі тестування, подальші запити та сканування не буде проведене. Це дозволяє запобігти проведенню кібератак на зовнішні ресурси , але при цьому зберегти дані про наявність зовнішніх посилань. Якщо при ручному аналізі результатів знайдене посилання буде класифіковане, як те, що повинно відноситись до меж тестування, воно може бути додане до цілей тестування.

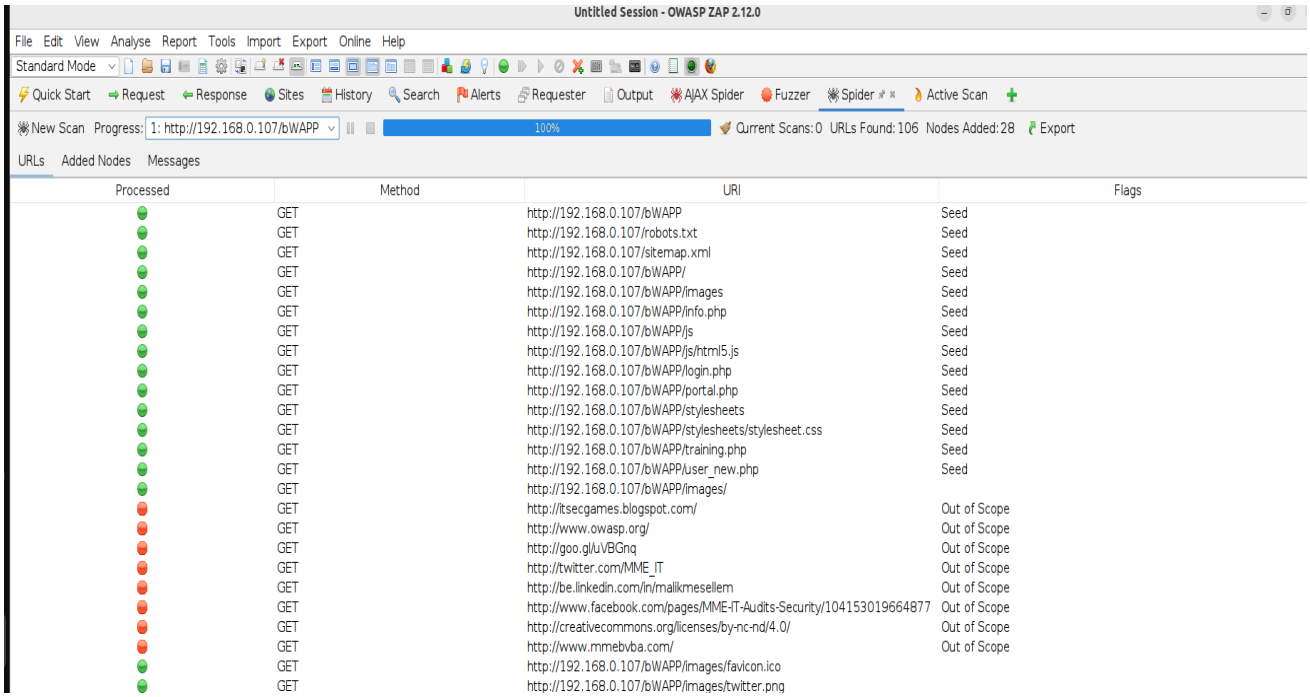


Рис. 3.12. Дії Spider

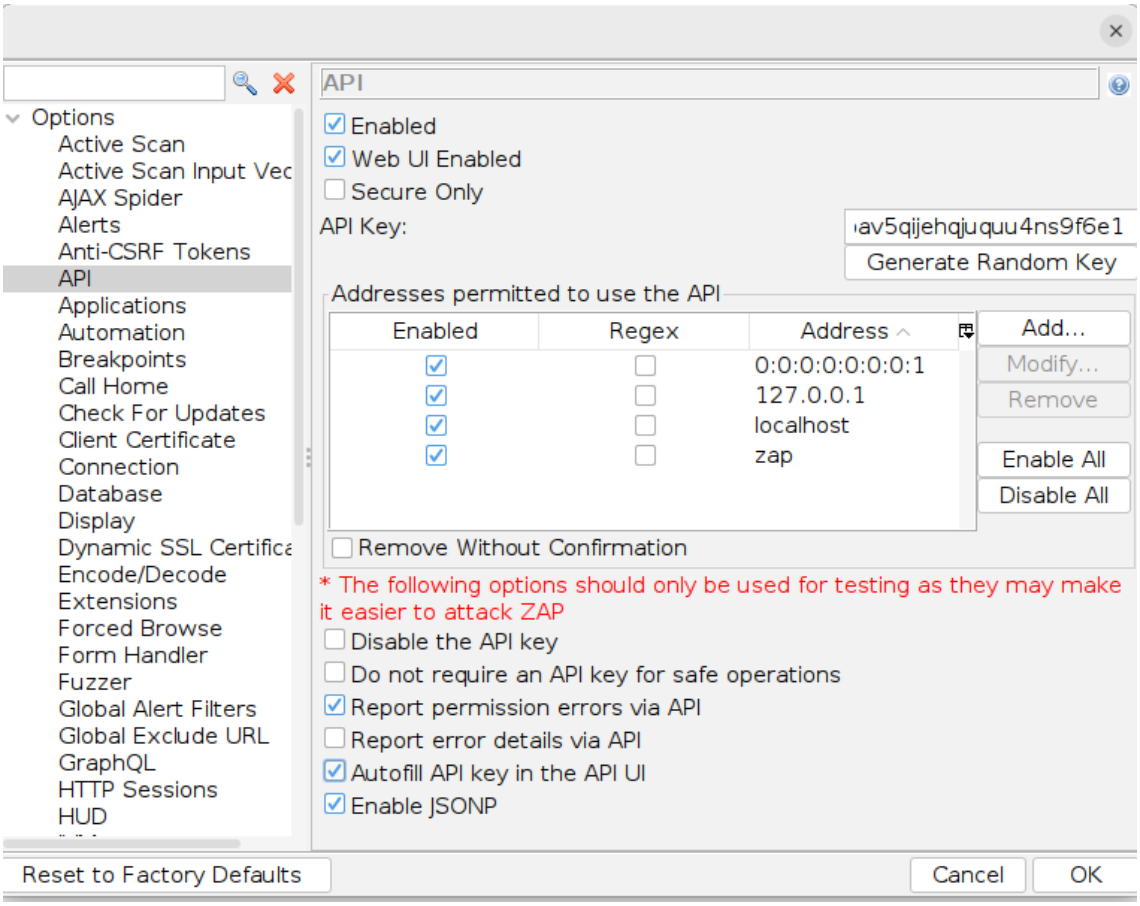


Рис. 3.13. Налаштування API

Для забезпечення оптимального функціонування вибирається підтримка JSON файлового типу, адже він широко використовується при взаємодії з API. Також для забезпечення безпеки, всі помилки з доступом до API будуть звітуватися. Для зручності додається автозаповнення ключів API в веб-інтерфейсі сканеру.

ZAP API, який дозволяє програмно взаємодіяти з ZAP. API доступний у форматах JSON, HTML і XML. Простий веб-інтерфейс користувача, який дозволяє досліджувати та використовувати API, доступний за URL-адресою http://zap/, коли використовуєте проксі через ZAP, або через хост і порт, які ZAP прослуховує, наприклад, http://localhost:8080/. За замовчуванням лише машина, на якій працює ZAP, має доступ до API.

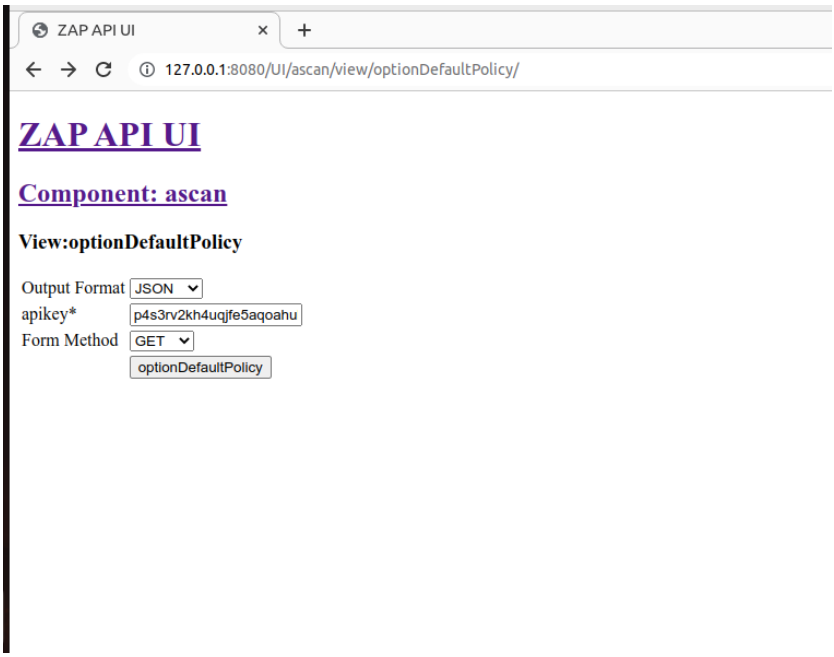


Рис. 3.14. Використання API

Для використання API через веб-інтерфейс необхідно перейти за однією з відповідних адрес та вибрати компонент, який необхідно налаштувати. Всі функції доступні за відповідним посиланням, або через графічний інтерфейс. Для використання компонента необхідно вибрати метод з яким буде робитись запит.

Також, оскільки налаштований доступ через API-ключ, його необхідно використати в відповідному полі. Для отримання результатів, також необхідно вибрати тип файлу в якому вона буде наданий результат звернення (якщо виконана дія має надати відповідний результат.

Нотація об’єктів JavaScript (JSON) — це стандартний текстовий формат для представлення структурованих даних на основі синтаксису об’єктів JavaScript. Він зазвичай використовується для передачі даних у веб-додатках (наприклад, надсилання деяких даних із сервера до клієнта, щоб їх можна було відобразити на веб-сторінці, або навпаки). Цей формат файлу оптимальний для використання як людиною, так і для обробку програмним забезпеченням.

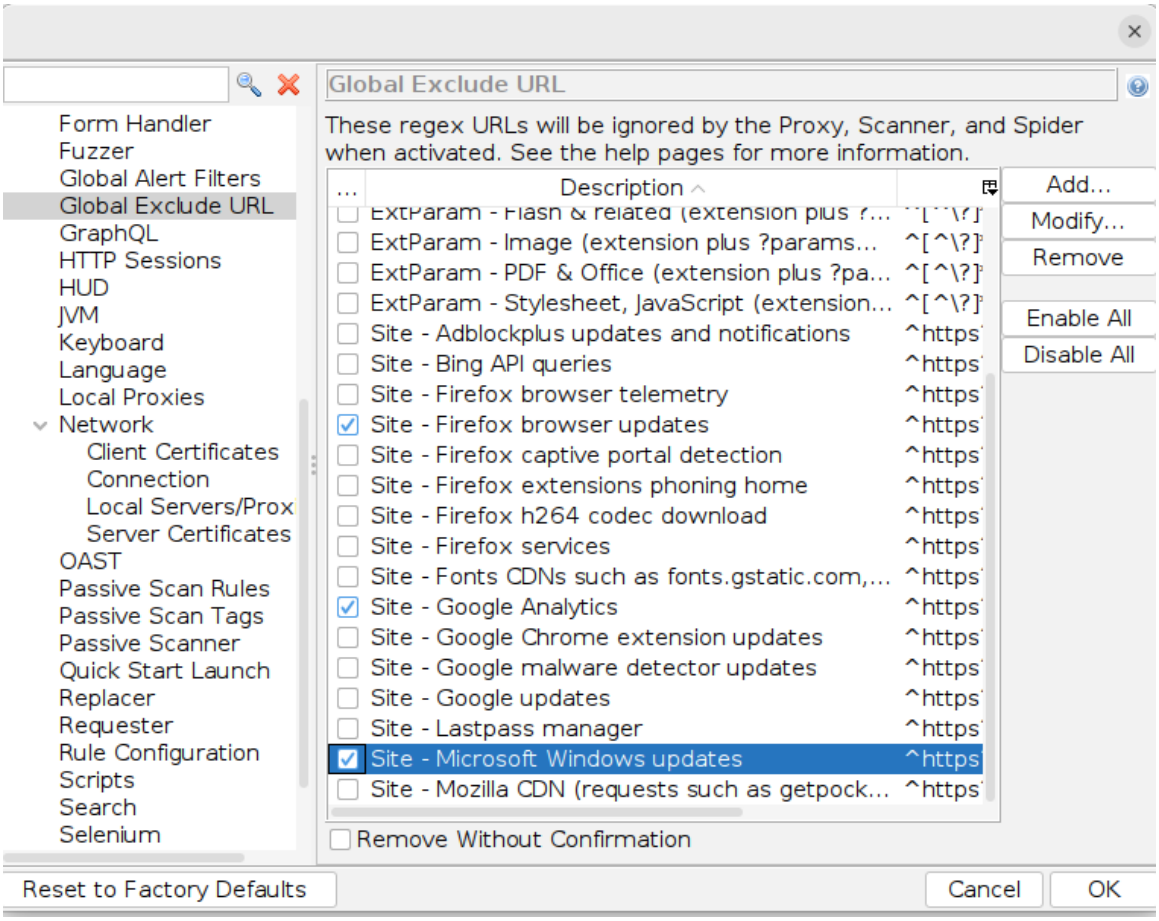


Рис. 3.15. Налаштування виключенньь

Для запобігання додавання в тестування непотрібних посилань, можливо виключити певні їх типи через спеціальний інтерфейс налаштування. В ньому виділяються стандартні типи URL. Серед них вибирається типові для вибраного середовища.

Оскільки використовується браузер Firefox, виключаються посилання на його оновлення щоб запобігти проведенню тестування по цілям які не повинні входити до тесту вибраного веб-сайту. Аналогічно проводиться виключення Microsoft Windows Updates.

Серед інших виключень вибирається сервіс Google Analytics, який часто використовується в сучасних веб-сайтах, проте проведення його тестування може порушити умови проведення тесту.

Інші параметри, такі як файлу різного типу залишаються для тестування, адже їх аналіз може надати релевантні тестові данні і при проведенні тестування цих елементів не будуть порушені умови тестування. Відповідно не виключаються посилання пов’язані з іншими браузерами та CDN сервісів, які відсутні в даному об’єкті тестування.

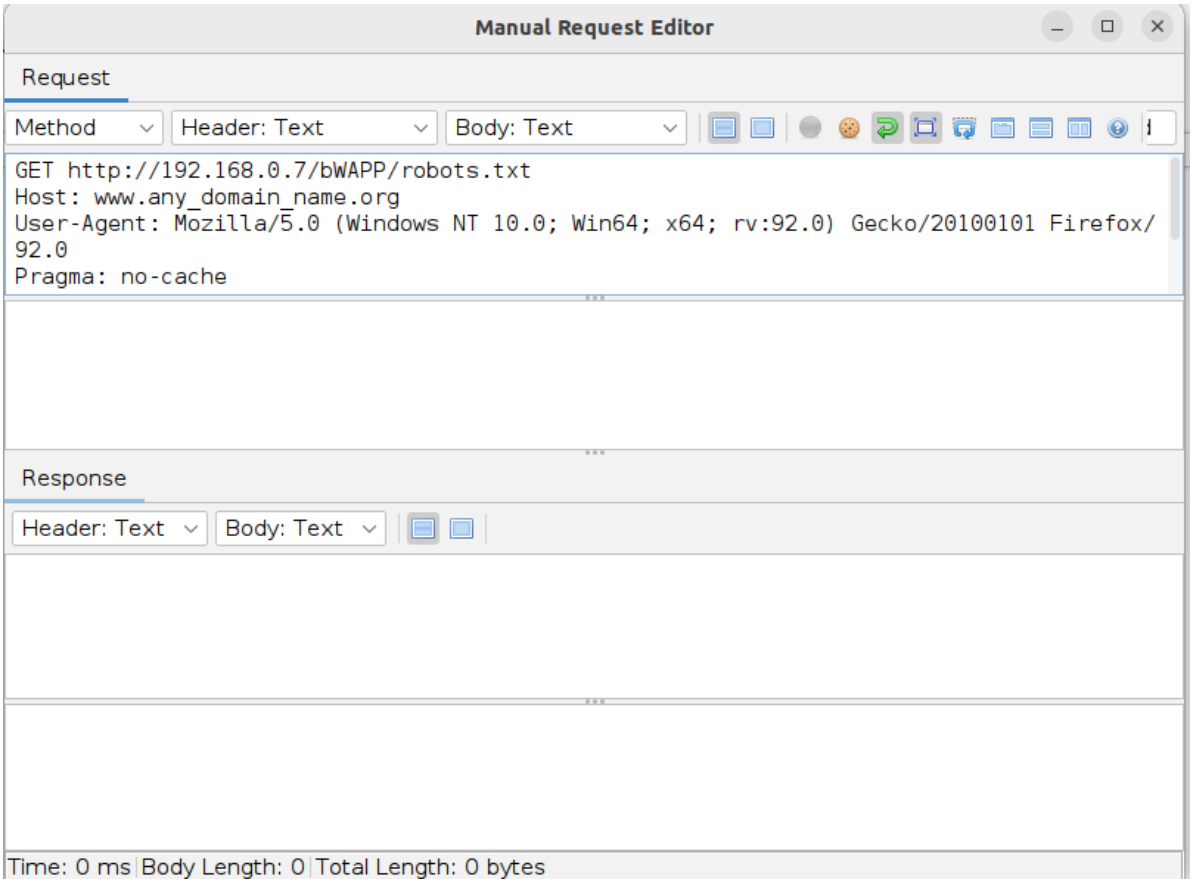


Рис. 3.16. Ручні запити

Додатковим функціоналом який може доповнити тестування є ручне формування запитів. Він підтримує основний набір функцій з аналогічних автоматизованого тестування, проте дає більше можливостей для детального налаштування.

Перш за все вибирається метод запиту. Набір доступних методів відповідає всім основним методам HTTP протоколу. Далі формується ціль запиту, яке може бути подана в вигляді повного доменного імені або IP-адреси. Також є можливість в специфікації директорі, або файлу.

Далі можна додати інформацію про хост з якого робиться запит. Для цього надається інформація про веб-браузер, операційну систему та інші дані. Вся надана інформація може не відповідати характеристикам та програмному забезпеченню, яке налаштоване на машині з якої проводиться тестування. Додатково підтримується можливість переведння запиту в різний формат презентації данних, такий як HEX. В цьому ж вікні можна ввімкнути або відключити прийняття cookie-файлів, якщо цього потребує специфіка запиту.

В нижній частині інтерфейсу буде відображено відповідь від об’єкту тестування.

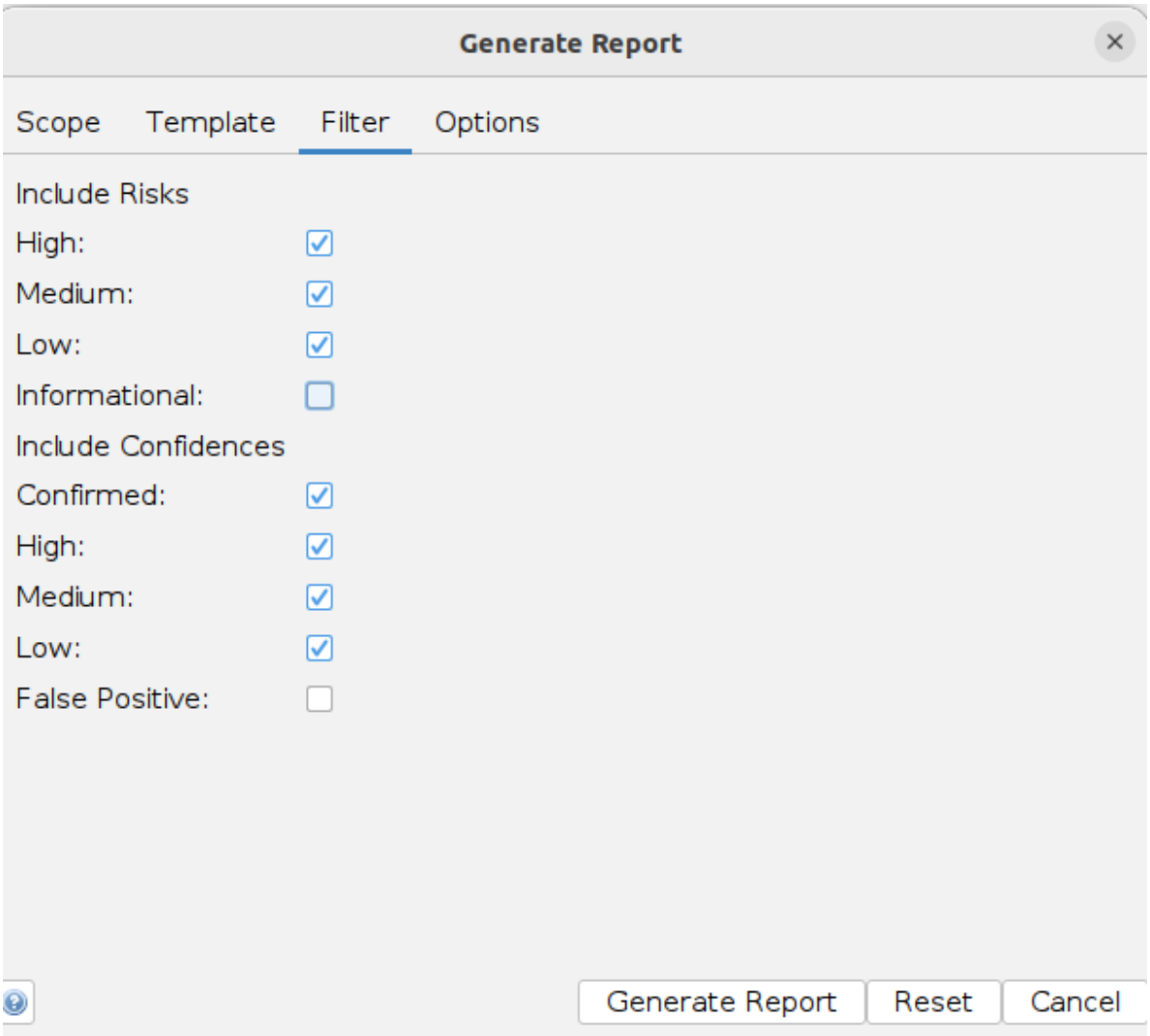


Рис. 3.17. Налаштування звіту по тестуванню

Останнім важливим кроком роботи є налаштування генерації звітів. Вибираються всі опції окрім інформаційних даних які не несуть в собі інформації про загрози в веб-сайті та невірних спрацювань. Ця інформація може бути корисною в цілому, але не відповідає поставленій цілі тестування.

Налаштування «Include Confidences» виділяє загрози , які будуть включені в звіт. Параметр Confidence вказує рівень впевненості сканера в те , що загроза не є невірним спрацюванням. Рівень може бути характеризований число від 0 до 4 , де 0 — це невірне спрацювання.

Оскільки в даному тесті збирається інформація про всі потенційні загрози , виключається лише всі знайдені вразливості з рівнем 0, а всі інші залишаються для подальшого розгляду. Цей параметр дає можливість виділити лише загрози , які мають високі шанси на існування в веб-сайті, для їх подальшого розгляду і створення тест-кейсів.

Наступним налаштуванням яке може бути доданим до звітів є шаблон звіту. Шаблон звіту являє собою загальну форму, дизайн та структуру в якій буде поданий звіт. Це налаштування повністю опирається на стандарти проведення звітування в організації, проте для даного тесту буде налаштований звіт, який буде зручно аналізувати для подальшого складання тестів.

Перш за все буде додана структура звіту на початку та загальний опис і параметри звіту. Всі знайдені вразливості та загрози будуть додані з додатковим вказанням їх числа знайденого за типом та рівнем впевненості. Також будуть додані запити за якими були знайдені вказані вразливості для їх подальшого аналізу.

Також в звіт буде включений додаток з інформацією і опис знайдених вразливостей і додаткові дані по типу вразливостей. Це дозволить використовувати звіт, як джерело інформації по існуючим загрозам без використання інших ресурсів.

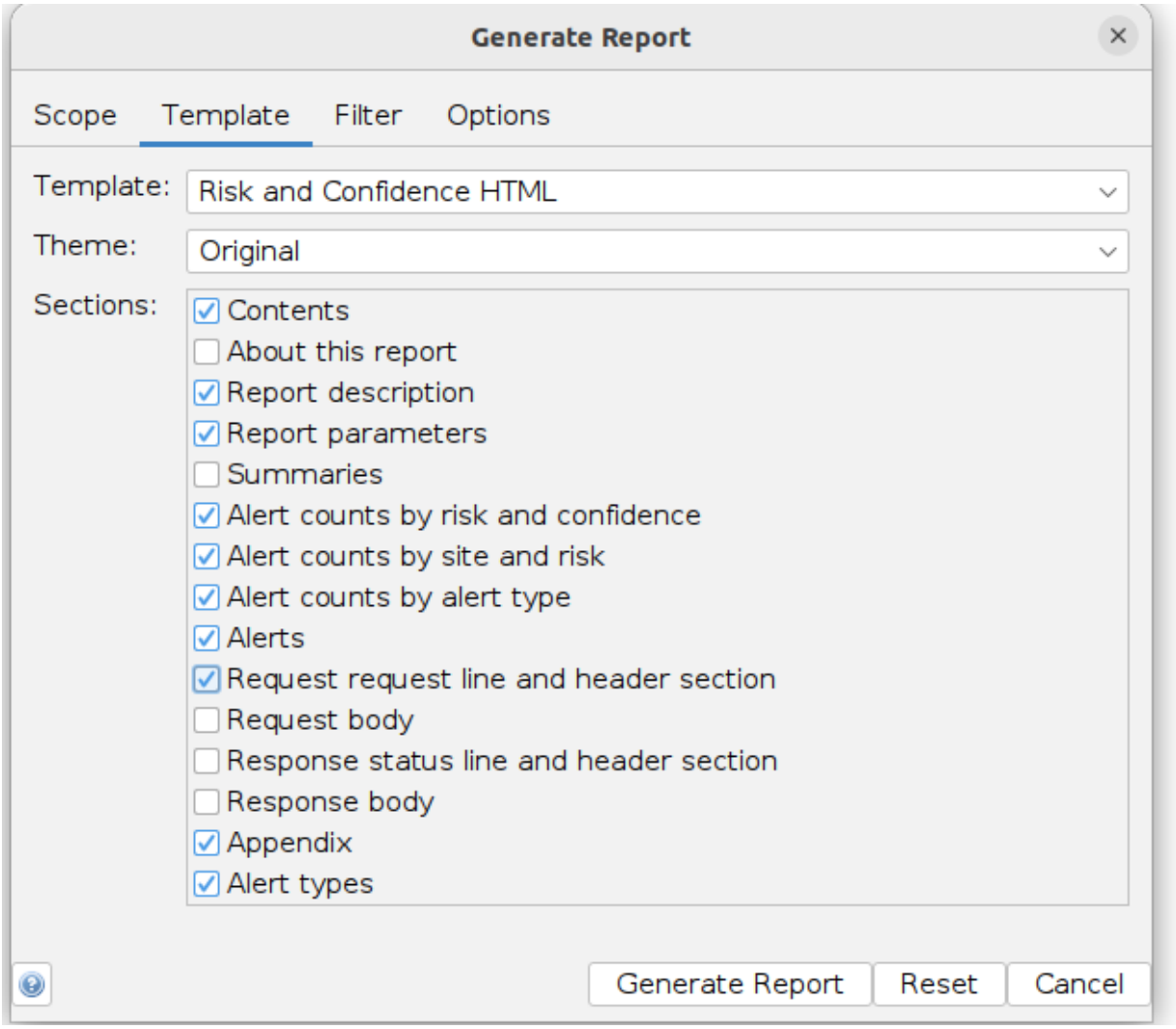
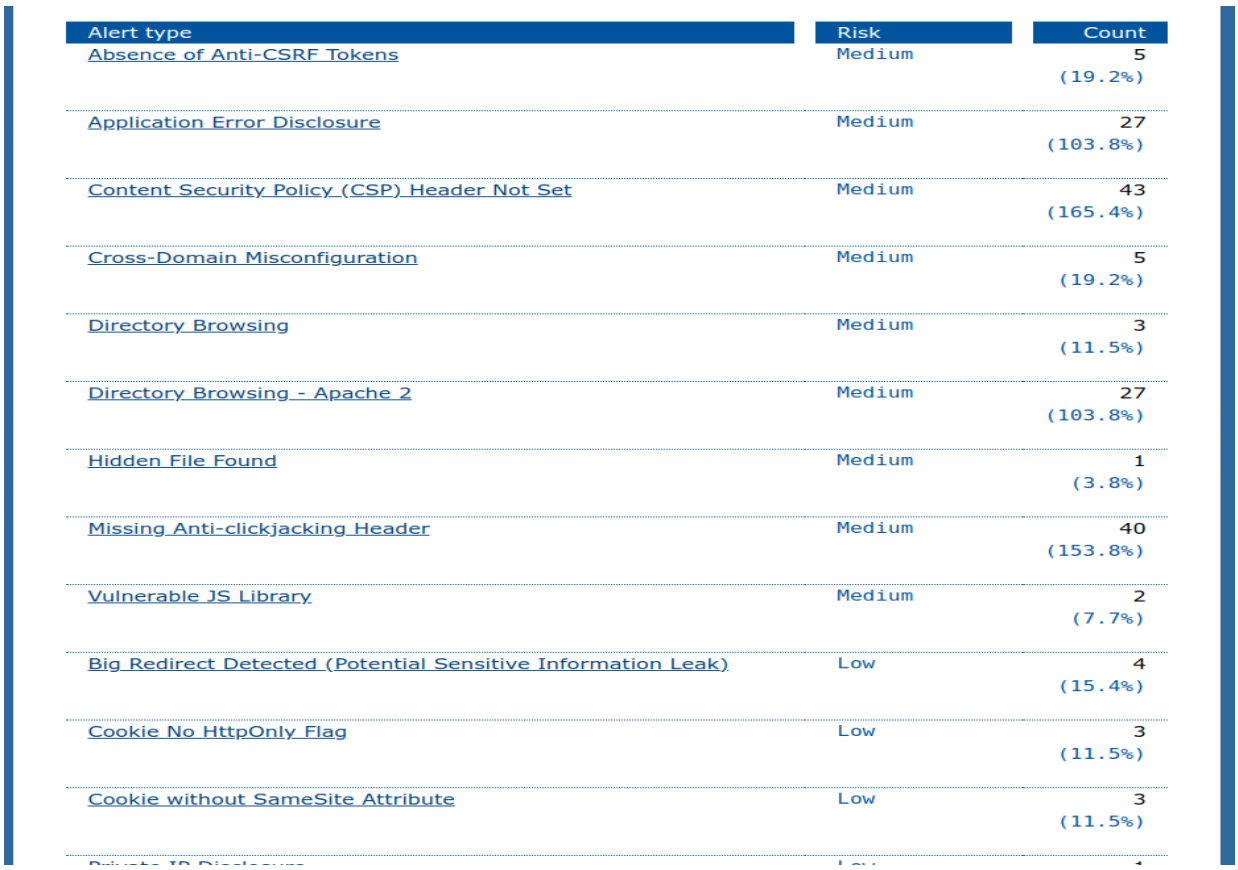


Рис. 3.18. Налаштування шаблонів для звітів

Звіт має наступний формат доступний для перегляду і аналізу. Оскільки для формату була вибрана HTML-сторінка OWASP ZAP генерував зручний для використання звіт з гіперпосиланнями на додаткову інформацію. В ньому можна зробити загальний огляд результатів тестування, переглянути статистику, загальна й опис загроз та вразливостей, які були знайдені в веб-сайті. Інформація зі звіту може бути використана, як відділами спрямованими на забезпечення технічної забезпеченості веб-сайту та безпеки, так і відділом по оцінці ризиків та фінансуванням.

Склад звіту та його формат мають відповідати тому, хто буде з ними ознайомлюватись в ході подальшої роботи над тестуванням.



Alert type	Risk	Count
Absence of Anti-CSRF Tokens	Medium	5 (19.2%)
Application Error Disclosure	Medium	27 (103.8%)
Content Security Policy (CSP) Header Not Set	Medium	43 (165.4%)
Cross-Domain Misconfiguration	Medium	5 (19.2%)
Directory Browsing	Medium	3 (11.5%)
Directory Browsing - Apache 2	Medium	27 (103.8%)
Hidden File Found	Medium	1 (3.8%)
Missing Anti-clickjacking Header	Medium	40 (153.8%)
Vulnerable JS Library	Medium	2 (7.7%)
Big Redirect Detected (Potential Sensitive Information Leak)	Low	4 (15.4%)
Cookie No HttpOnly Flag	Low	3 (11.5%)
Cookie without SameSite Attribute	Low	3 (11.5%)

Рис. 3.19. Звіт по тестуванню

Для формування формального та стандартизованого тестування можливо сформувати тест-кейс. Тест кейс — це набір вхідних значень, попередніх умов виконання, очікуваних результатів і постумов виконання, розроблених для конкретної цілі або умови тестування, наприклад, для виконання певного програмного шляху або для перевірки відповідності певній вимозі.

Для формування проведеного тестування в тест кейс необхідно надати йому відповідний ID, назву, передумови, пункти для виконання тесту та очікувані результати. В даному випадку кейс може мати умовний ID 1 та назву «Тестування на визначення вразливостей OWASP TOP 10».

Передумовами виступатимуть наявність тестового середовища, проведення налаштування сканера для оптимального виконання тесту. Всі виконані пункти налаштування можна описати як частину передумов.

Оскільки це автоматизоване сканування, пунктами для виконання виступають лише вибір цілі та початок сканування. Ручне сканування потребувало би точного описання запитів та дій які необхідно провести для тестування.

Очікуваним результатом буде виступати відсутність в веб-сайті відповідних вразливостей, або всі спрацювання сканеру на них не будуть відповідати дійсності.

3.2 Рекомендація щодо проведення тестування на основі OWASP ZAP

Для проведення ефективного тестування безпеки на основі сканера веб-безпеки OWASP ZAP необхідно точно визначити цілі тестування, ресурси які можуть бути затрачені на тестування та час тестування. При цьому для формування згаданих вимог до тесту можливе використання інших інструментів, таких як сканер безпеки Nikto або сканер мережі NMAP. Ці інструменти дозволять дізнатися як виглядає мережа в якій знаходиться наш веб-сайт зі сторони зловмисник, а також виділити всі доступні з зовні данні про сервіси в ній. При цьому додаткові інструменти можуть бути використані і під час проведення тестів як додаткове джерело інформації. Маючи всі можливі інформації про об’єкт тестування, можна отримати точну інформацію і використати її в подальшому.

Важливим кроком в проведенні тестування також виступає налаштування самого OWASP ZAP. Перш за все необхідно налаштувати проксі-сервер, який дозволить сканеру зібрати більше інформації про рівень безпеки веб-сайту. Проксі налаштування потрібно також додати на рівні операційної системи, мережевого адаптеру або веб-браузера в залежності від специфіки тестового середовища. Так додавання налаштування на рівні веб-браузера дозволить мінімізувати вплив роботи сканера на інші компоненти тестового середовища, в той час як реалізація цього налаштування на рівні операційної системи дозволить додати додаткові данні для сканеру. Важливим етапом в налаштування проксі є генерація SSL-сертифікату для використання під час тестування. Це дозволить сканеру

встановити зашифроване з’єднання з хостом від якого іде сканування та аналізувати зашифрований трафік. При цьому сам сертифікат встановлюється на рівні операційної системи або веб-браузера. В його налаштуваннях можна вказати термін валідності. При планування великої кількості тестів, які будуть проводитись через наш сканер, термін валідності краще підняти до одного року, щоб не витрачати час на постійне переналаштування сканера та тестового середовища.

Важливим налаштуванням є використання сертифікату. Тут є можливість додати або виключити два налаштування – використання сертифікату для з’єднання з веб-сайтами та використання його для електронної пошти. Самим необхідним пунктом виступає використання його для з’єднання з веб-сайтами, проте додатковий функціонал для електронної пошти також розширить спектр можливостей сканеру.

Ще одним налаштування яке необхідно провести перед початком тестування є налаштування політики сканування. Це дозволить сканеру ігнорувати загрози певного рівня, для пониження кількості непотрібної в звіті по тестуванню, або в інтерфейсі самого сканеру інформації. Таке налаштування особливо важливе, якщо проводиться пошук лише критичних загроз і данні з категорії «Низький рівень загрози» або «Інформація» не мають цінності для даного тесту. Також є можливість встановлення сили самого тестування, що вкаже сканеру з якою інтенсивністю необхідно проводити атаку. Якщо цього потребує тестування, то є можливість створення подібних налаштування для певного типу атаки.

Щодо налаштування додаткових інструментів, то необхідно налаштувати Spider. Це надасть можливість надати необхідну ресурсозатратність та не додавати до цілей тесту зайвих веб-сторінок. Базовою конфігурацією є 5, проте при детальному розгляді об’єкту тестування краще зробити підвищення до 5-8.

При розгляді самого тестування в процесі є можливість перегляду великої кількості даних з різних інтерфейсів, проте найкраще та найбільш детальними є «Alerts». Саме тут зібрані деталізовані звіти по знайденим вразливостям, опис

загрози, надана оцінка її критичності та посилання на додаткові інформаційні ресурси. Опираючись на данні отримані звідси можна продовжити проводити окремі тестування по тим елементам де були знайдені критичні вразливості.

Ще одним потенційним використанням інформації звідси є формування загальної представлення про рівень безпеки веб-сайту. Оскільки інформація подана в розділеному по категоріям вигляду, її легко обробляти для експертної оцінки. При потребі в більших деталях про виконанні дії або конкретизації інформації про атаку краще використати відповідні інтерфейси.

При генерації звітів варто вказати цілі, про які цей звіт буде створено, вказати саме тестування та налаштувати яка саме інформація повинна бути включена в звіт. Це дозволить не перенавантажувати звіт непотрібною інформацією. При цьому, перед формуванням самого звіту варто ознайомитись з даними тестування та зокрема виділити пункти відповідаючи цілі тестування для подальшого включення його в звіт.

Щодо розробки самих тестів, оптимальним рішенням є використанням в тестуваннях безпеки стандартів тестування програмного забезпечення. Це є загально використаною та перевіреною практикою, яка може бути налаштована для відповідного проекту та його специфіки. Ця практика включає формування тест кейсів з відповідними параметрами. Звичайний тест-кейс має включати наступне:

- ID тест кейсу;
- Цілі тест-кейсу;
- Умови для виконання;
- Очікуваний результат виконання тест-кейсу;
- Додаткова інформація або інструментарій.

Варто помітити, що використання цієї структури є неоптимальним, якщо тестування проводиться інструментами з широким функціоналом, такими як OWASP ZAP. В цьому випадку краще замінити «Очікуваний результат» на напрями тестування, які будуть виконані та який рівень безпеки можна прийняти

для тестованих елементів, які вразливості є критичними і їх знаходження варто відмітити. OWASP ZAP з його функціоналом по генерації звітів легко використовувати з подібною структурую тест-кейсу.

Висновки до розділу 3

Досліджено процес проведення тестування безпеки веб-сайту на основі рішення OWASP ZAP. Проведено тестування на виявлення вразливостей в спеціалізованому тестовому середовищі для пошуку вразливостей в ньому.

Проведено налаштування програмного засобу для відповідності цілі тестування та його ресурсам. Налаштовано генерацію звітів по тестування та переглянуто його основні пункти.

Проаналізовано можливості отримання тестових даних з різних інтерфейсів OWASP ZAP та можливість їх використання в ході наступних тестування. Виявлено основні напрямки функціонування компонентів автоматизованого сканеру.

Розроблено рекомендації, щодо налаштування OWASP ZAP під специфіку та цілі проектів та окремих тестів. Надано рекомендації, щодо проведення безпечного тестування.

Виокремлено можливість використання загальних практик тестування програмного забезпечення для проведення тестування безпеки веб-сайтів.

ВИСНОВКИ

В магістерській роботі отримано наступні наукові та науково-практичні результати:

1. Проаналізовано та досліджено основні загрози для веб-сайтів та веб-додатків, а також динаміку їх зміни.

2. Наведено аналітичні дані по основним сучасним принципам та методам проведення тестування безпеки.

3 Виокремлено сучасні інструменти проведення тестування безпеки та тестові данні, які можуть бути отримані з їх використанням.

4. Виконано порівняльний аналіз трьох рішень для автоматизації тестування безпеки : OWASP ZAP, Burp Suite та Invciti .

5. Виокремлено переваги та недоліки у функціонуванні між OWASP ZAP, Burp Suite та Invciti.

6. Досліджено основні функції та компоненти OWASP ZAP та їх використання під час тестування безпеки.

7. Виконано налаштування OWASP ZAP та проведено автоматизоване тестування безпеки з його використанням для визначення вразливостей вибраного тестового середовища.

8. Виконано налаштування звітування в OWASP ZAP відповідно до поставлених задач тестування.

9. Розроблено рекомендації щодо налаштування та використання під час тестування безпеки технології OWASP ZAP.

10. Розроблено рекомендації, щодо використання практик тестування програмного забезпечення для тестування безпеки