Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

Комплексна бакалаврська дипломна робота на тему:
«Система тестування безпеки програмного застосунку. Частина 2. Підсистема динамічного тестування»

Виконав: студент 4 курсу групи 1БС-20б
спеціальності 125 Кібербезпека
_____ Дмитро ЖЕЛНИТСЬКИЙ

Керівник: к. т. н., доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
«14» червня 2024 р.

Рецензент: к. т. н. доцент каф. ПЗ
_____ Вікторія ВОЙТКО
«14» червня 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«14» червня 2024 р.

Вінниця ВНТУ – 2024 року

Міністерство освіти і науки України
Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Рівень вищої освіти І (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
Володимир ЛУЖЕЦЬКИЙ
«14» березня 2024 року

З А В Д А Н Н Я
НА КОМПЛЕКСНУ БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Дмитру ЖЕЛНИТСЬКОМУ

1. Тема роботи: «Система тестування безпеки програмного застосунку. Частина 2. Підсистема динамічного тестування»
   керівник роботи: Віталій ЛУКІЧОВ, к. т. н., доцент кафедри ЗІ,
   затверджені наказом ректора ВНТУ від 11 березня 2024 року за № 80.
2. Строк подання студентом роботи 14 червня 2024 р.
3. Вихідні дані до роботи:
   - Галузь застосування: Тестування безпеки веб-додатків.
   - Метод тестування: Динамічне тестування безпеки (DAST).
   - Середовище для реалізації підсистеми: хмарні технології, Amazon Web Services (AWS)
   - Метод для релізації підсистеми: Інфраструктура як код (IaC)
4. Зміст текстової частини: Вступ. Аналіз методів та інструментів динамічного тестування безпеки застосунків. Розробка підсистеми динамічного тестування безпеки застосунків. Реалізація підсистеми динамічного тестування безпеки застосунків. Висновки. Список використаних джерел. Додатки.
5. Перелік ілюстративного матеріалу: порівняння інструментів DAST (плакат А4), класифікація методів DAST (плакат А4), схема системи тестування безпеки (плакат А4), процеси підсистеми динамічного тестування безпеки (плакат А4), схема шаблону CloudFormation (плакат А4), схема інфраструктури (плакат А4).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	В. ЛУКІЧОВ, к.т.н., доц. каф.ЗІ	11.03.24	31.03.24
2	В. ЛУКІЧОВ, к.т.н., доц. каф.ЗІ	11.03.24	16.04.24
3	В. ЛУКІЧОВ, к.т.н., доц. каф.ЗІ	11.03.24	04.05.24

7. Дата видачі завдання 11 березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітк
1	Аналіз завдання. Вступ	12.03.24 – 14.03.24	
3	Аналіз інформаційних джерел за напрямком бакалаврської дипломної роботи	15.03.24 – 31.03.24	
4	Розробка рішень, моделей, алгоритмів	01.04.24 – 15.04.24	
5	Практична реалізація, моделювання, експериментування, результати	16.04.24 – 01.05.24	
6	Оформлення пояснювальної записки	02.05.24 – 10.05.24	
7	Попередній захист БДР	30.05.24 – 07.06.24	
8	Виправлення зауважень, підготовка ілюстративного матеріалу	10.06.24 – 12.06.24	
9	Перевірка на наявність текстових запозичень	12.06.24 – 13.06.24	
10	Представлення БДР до захисту, рецензування	13.06.24 – 17.06.24	
11	Захист БДР	18.06.24 – 21.06.24	

Студент _____ Дмитро ЖЕЛНИТСЬКИЙ

Керівник роботи _____ Віталій ЛУКІЧОВ

АНОТАЦІЯ

Комплексна бакалаврська дипломна робота складається з 56 сторінок формату А4, на яких є 17 рисунків, 6 таблиць, список використаних джерел містить 20 найменувань.

Бакалаврська робота присвячена розробці системи тестування безпеки застосунку, підсистеми динамічного тестування безпеки (DAST). Проведено аналіз існуючих методів і засобів для динамічного тестування безпеки, визначено їх основні переваги та недоліки. Розроблено підсистему, в основі якої інструменти та методики, що дозволяють ефективно та точно виявляти вразливості в застосунках під час їх виконання. Це дає змогу забезпечити високий рівень безпеки програмного забезпечення перед його впровадженням у виробниче середовище.

Ключові слова: DAST, безпека застосунків, тестування безпеки, вразливості, DevSecOps.

ABSTRACT

The comprehensive bachelor's thesis consists of 56 A4 pages, which include 17 figures, 6 tables, and a list of 20 references.

The bachelor's thesis is dedicated to the development of an application security testing system, specifically the dynamic application security testing (DAST) subsystem. An analysis of existing methods and tools for dynamic application security testing was conducted, identifying their main advantages and disadvantages. A system has been developed based on tools, practices, and methodologies that allow for the effective and accurate detection of vulnerabilities in applications during their execution. This ensures a high level of software security before its deployment in the production environment.

Keywords: DAST, application security, security testing, vulnerabilities, DevSecOps.

ЗМІСТ

ВСТУП

З урахуванням постійного розвитку технологій та загроз кібербезпеці, тестування безпеки застосунків стає необхідним етапом у життєвому циклі розробки програмного забезпечення. У сучасному світі, де кількість кібератак зростає з кожним днем, забезпечення безпеки програмного забезпечення стає критично важливим завданням. Вразливості в програмних застосунках можуть призвести до значних фінансових втрат, витоку конфіденційної інформації та втрати репутації компаній. Тому питання тестування безпеки програмного забезпечення набуває особливої важливості [1].

Динамічне тестування безпеки (Dynamic Application Security Testing, DAST) є одним із ключових методів виявлення вразливостей, оскільки дозволяє оцінювати безпеку застосунків у режимі реального часу, аналізуючи їх поведінку під час виконання. Це тестування дозволяє оцінити безпекові вразливості застосунків у реальних умовах, відтворюючи умови кібератак [2].

Метою цього дослідження є підвищення безпеки застосунків за допомогою підсистеми динамічного тестування. Об'єктом дослідження є процес тестування безпеки програмних застосунків, а предметом – методи та інструменти динамічного тестування безпеки програмних застосунків. Для досягнення поставленої мети необхідно виконати наступні завдання:

- Провести огляд і класифікацію методів динамічного тестування безпеки застосунків та здійснити аналіз існуючих інструментів для динамічного тестування безпеки.
- Розробити підсистему динамічного тестування застосунку.
- Оцінити ефективність та точність виявлення вразливостей обраним засобом динамічного тестування.
- Практично реалізувати підсистему динамічного тестування безпеки.
- Отримати і проаналізувати результат роботи підсистеми на практиці.

З огляду на постійно зростаючу кількість та складність кіберзагроз, дослідження ефективності засобів динамічного тестування має велике значення для

забезпечення безпеки програмного забезпечення. DAST допомагає виявляти вразливості безпеки у веб-додатках та надає рекомендації щодо їх усунення, що є важливим компонентом будь-якої комплексної системи перевірки безпеки додатків

1 АНАЛІЗ МЕТОДІВ ТА ІНСТРУМЕНТІВ ДИНАМІЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ ЗАСТОСУНКІВ

1.1 Загальний аналіз динамічного тестування безпеки

DAST (Dynamic Application Security Testing) — це метод тестування безпеки, відомий як "чорна скринька". Він дозволяє знаходити вразливості та слабкі місця в працюючих додатках шляхом ін'єкції шкідливих навантажень для виявлення потенційних недоліків, що можуть спричинити атаки, такі як SQL-ін'єкції або міжсайтовий скриптинг (XSS) тощо [2]. Інструменти DAST особливо корисні для виявлення таких проблем, як:

- Перевірка вхідних або вихідних даних.
- Проблеми з автентифікацією.
- Помилки в конфігурації сервера.

Інструменти DAST дозволяють виконувати складні сканування як на стороні клієнта, так і на стороні сервера без необхідності в доступі до вихідного коду або фреймворку, на якому побудовано додаток. Зазвичай вони вимагають мінімальної взаємодії з користувачем після налаштування і можуть запускатися як частина «нічного» сканування. До найважливіших інструментів DAST можна віднести наступні:

- Динамічні сканери безпеки.
- Фаззери.
- Інструменти проксі-атак.

Існує велика кількість як комерційних, так і відкритих інструментів цього типу, і всі вони мають свої сильні і слабкі сторони [2].

DAST пропонує кілька переваг, таких як виявлення вразливостей, які проявляються лише під час виконання, включаючи проблеми, пов'язані з автентифікацією, управлінням сесіями, обробкою даних тощо. Він тестує додаток у реальних умовах роботи та може виявляти помилки конфігурації та розгортання. Крім того, DAST не залежить від мови програмування, що робить його універсальним для тестування різноманітних додатків. Однак його обмеження

включають можливість великої кількості хибних спрацювань, неможливість надати інформацію про місцезнаходження вразливостей у базі коду, оскільки він працює без доступу до вихідного коду. Крім того, оскільки DAST тестує «живі» додатки, існує ризик порушення нормальної роботи, якщо не виконувати його обережно.

Переваги DAST:

- Може швидко виявляти вразливості в працюючих додатках без доступу до вихідного коду, що робить його цінним інструментом для тестування сторонніх додатків або застарілих систем, які можуть не підтримуватися активно [5].
- Метод імітує реальні атаки на додаток, надаючи більш точне уявлення про безпеку додатка, ніж інші методи тестування [5].
- Можна використовувати для тестування широкого спектру веб-додатків, включаючи ті, що побудовані з використанням різних мов програмування та розміщені в різних середовищах [5].
- Може виявляти вразливості, які можуть бути невидимими зсередини, такі як ті, що пов'язані з перевіркою введення та контролем доступу [5].
- DAST можна автоматизувати для економії часу та зусиль у процесі тестування [5].

Однак DAST може давати хибні спрацювання або пропускати певні вразливості, якщо інструмент налаштований неправильно або якщо тестове середовище не точно відтворює виробниче середовище.

- DAST може не виявляти вразливості, які невидимі ззовні, такі як ті, що пов'язані з кодом додатка [5].
- Може бути обмежений у здатності виявляти вразливості у додатках з архітектурами, такими як односторінкові додатки та ті, що використовують JavaScript фреймворки [5].
- Покладається на стандартні або застарілі методи веб-додатків, а не на динамічні додатки, які є нормою в наш час [5].
- DAST не надає повної картини безпеки додатка і повинен використовуватися разом з іншими методами тестування безпеки, такими як SAST та IAST [5].

1.2 Класифікація методів динамічного тестування безпеки

Методи DAST (Dynamic Application Security Testing) дозволяють проводити динамічне тестування безпеки додатків у режимі реального часу, коли вони працюють. Такий підхід дозволяє виявити вразливості, які можуть бути пропущені під час статичного аналізу вихідного коду. Щоб забезпечити повне розуміння різноманітних методів DAST, необхідно глибше зануритися в кожну з основних класифікацій. Загальне представлення можна побачити на рис. 1.1

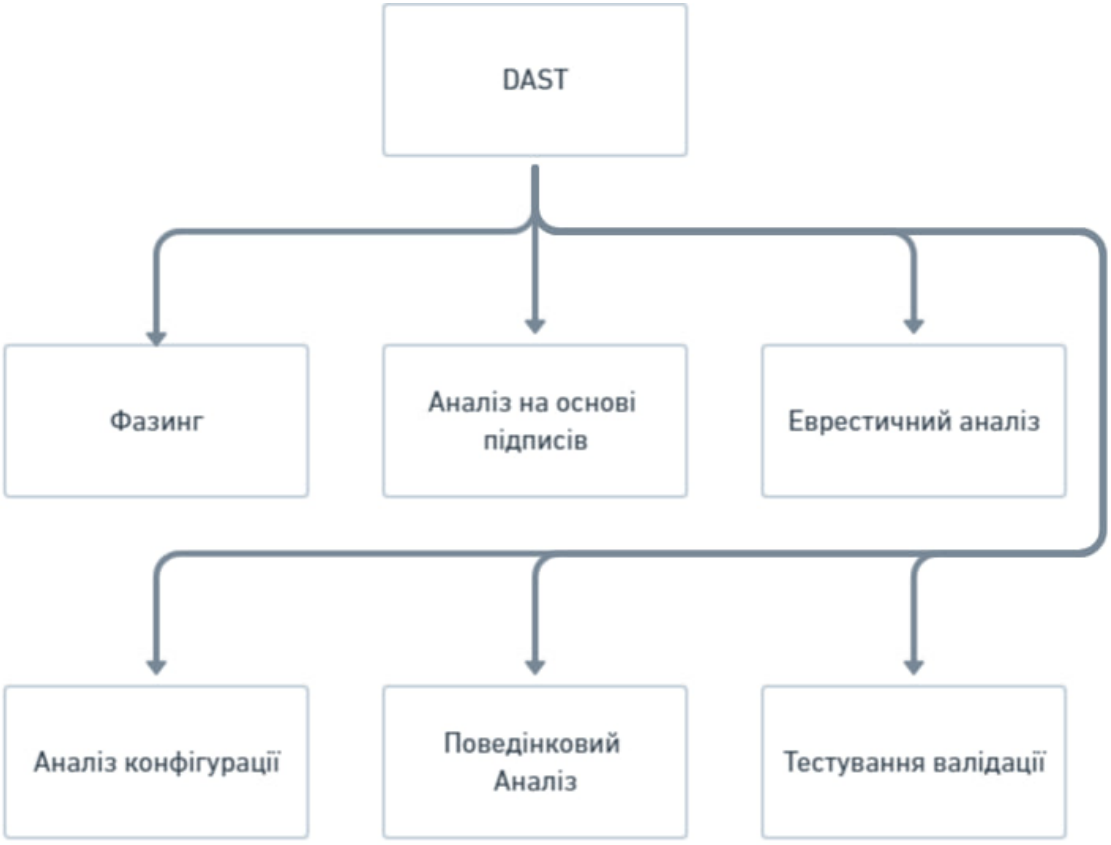


Рисунок 1.1 – Методи DAST

1.1.1 Фаззинг (Fuzzing)

Фаззинг — це метод тестування, який включає введення випадкових, некоректних або непередбачуваних даних у застосунок, щоб виявити потенційні вразливості. Мета фаззингу полягає в тому, щоб виявити слабкі місця, які можуть призвести до аварійного завершення роботи програми або до непередбачуваної поведінки. Використовується для виявлення переповнень буфера, коли дані, що

вводяться, перевищують розмір буфера, призначеного для їх зберігання, що може призвести до виконання довільного коду. Також застосовується для виявлення вразливостей, пов'язаних з ін'єкціями, таких як SQL-ін'єкції або командні ін'єкції, та аналізу стійкості до несподіваних вхідних даних, щоб визначити, як застосунок справляється з непередбачуваними або некоректними вхідними даними [4].

1.1.2 Аналіз на основі підписів (Signature-Based Analysis)

Цей метод використовує відомі шаблони або підписи вразливостей для виявлення проблем. Інструмент порівнює поведінку застосунку з базою даних відомих атак і вразливостей. Призначення методу — швидке виявлення відомих вразливостей, таких як SQL-ін'єкції, XSS, та інші поширені загрози. База даних підписів постійно оновлюється, що робить цей метод ефективним для виявлення нових загроз, що вже були задокументовані [11].

1.1.3 Евристичний аналіз (Heuristic Analysis)

Евристичний аналіз використовує евристичні методи для виявлення аномалій та підозрілих патернів у поведінці застосунку. Цей підхід базується на алгоритмах і правилах, які допомагають визначити потенційні вразливості за певними характеристиками. Застосування цього методу включає виявлення нових вразливостей, що раніше не були відомі, через аналіз підозрілої поведінки застосунку, а також визначення аномалій, які можуть вказувати на потенційні вразливості або атаки [11].

1.1.4 Поведінковий аналіз (Behavioral Analysis)

Цей метод спостерігає за поведінкою застосунку в різних умовах і з різними вхідними даними, щоб виявити відхилення від нормальної або очікуваної поведінки. Використовується для виявлення проблем з логікою застосунку, що включає аналіз логіки роботи застосунку для виявлення можливих вразливостей, а також для аналізу аутентифікації та управління сесіями, перевіряючи механізми аутентифікації та управління сесіями на наявність вразливостей [5].

1.1.5 Тестування валідації вхідних даних (Input Validation Testing)

Зосереджений на перевірці того, як застосунок обробляє вхідні дані, щоб забезпечити правильну валідацію та очищення даних. Застосовується для запобігання атакам ін'єкцій, перевіряючи механізми валідації даних для запобігання SQL-ін'єкціям, XSS та іншим типам ін'єкцій, а також для перевірки санітарії введених даних, забезпечуючи, що всі вхідні дані проходять належну валідацію та очищення перед обробкою [5].

1.1.6 Аналіз конфігурації (Configuration Analysis)

Перевіряє конфігураційні налаштування застосунку, включаючи налаштування безпеки, середовище розгортання та налаштування сервера. Застосовується для виявлення неправильних конфігурацій, які можуть призвести до порушень безпеки, таких як слабкі налаштування шифрування або неправильні налаштування доступу, а також для перевірки середовища розгортання на наявність потенційних вразливостей у налаштуваннях сервера та інших компонентах інфраструктури [5].

1.3 Загальний аналіз інструментів динамічного тестування безпеки

Існує велика кількість інструментів для тестування на безпеку [6], але ми звернемо свою увагу головним чином на наступні три популярні інструменти:

1.2.1 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) є одним з найбільш популярних інструментів для динамічного тестування безпеки веб-додатків. Цей відкритий та безкоштовний інструмент надає широкий набір функцій для автоматичного сканування та інтерактивного аналізу безпеки. OWASP ZAP дозволяє виявляти різноманітні вразливості, такі як SQL-ін'єкції, XSS (міжсайтовий скриптинг), CSRF (підробка міжсайтових запитів) та інші [7].

Основні функції OWASP ZAP:

- Автоматичне сканування: дозволяє виявляти найпоширеніші вразливості у веб-додатках [7].

- Інтерактивний аналіз: можливість вручну перевіряти та маніпулювати HTTP/HTTPS запитами для більш детального аналізу [7].
- Підтримка плагінів: розширює можливості ZAP шляхом додавання нових функцій та інтеграцій [7].
- Репортинг: генерація звітів про знайдені вразливості, що допомагає розробникам швидко реагувати на проблеми [7].

Тестер використовує OWASP ZAP для сканування веб-додатку, отримуючи звіт про знайдені вразливості, такі як відкриті порти, небезпечні заголовки HTTP та можливі SQL-ін'єкції. Потім він використовує інтерактивний режим для більш детального аналізу певних частин додатку, наприклад, тестування форми входу на предмет XSS [7].

1.2.2 Burp Suite

Burp Suite є потужним комерційним інструментом для тестування безпеки веб-додатків, який надає широкий набір функцій для інтерактивного аналізу та автоматичного сканування. Burp Suite дозволяє тестерам проводити складні атаки на веб-додатки, маніпулювати HTTP/HTTPS трафіком та генерувати звіти про знайдені вразливості. Серед основних можливостей Burp Suite варто відзначити підтримку плагінів, інтеграцію з іншими інструментами та зручний інтерфейс.

Основні функції Burp Suite:

- Intercepting Proxy: дозволяє перехоплювати та змінювати HTTP/HTTPS трафік між браузером та сервером [8].
- Scanner: автоматичне сканування веб-додатків на наявність вразливостей [8].
- Intruder: інструмент для автоматизації складних атак, таких як brute-force атаки [8].
- Repeater: дозволяє вручну відправляти модифіковані запити та аналізувати відповіді [8].
- Extender: підтримка плагінів для розширення функціональності Burp Suite.

Тестувальник використовує Burp Suite для перехоплення та аналізу трафіку між браузером та веб-додатком, ідентифікує потенційні вразливості, такі як небезпечні

cookie або вразливості CSRF. Використовуючи Intruder, він автоматизує атаку для підбору паролів до облікового запису адміністратора, а за допомогою «Scanner» автоматично перевіряє наявність відомих вразливостей [8].

1.2.3 Acunetix

Acunetix є комерційним інструментом для автоматизованого сканування веб-додатків та аналізу їх безпеки. Acunetix надає можливості для виявлення широкого спектру вразливостей, включаючи SQL-ін'єкції, XSS, XXE (XML External Entity) та інші. Однією з переваг Acunetix є його інтеграція з системами CI/CD, що дозволяє автоматизувати процес тестування безпеки на всіх етапах розробки програмного забезпечення.

Основні функції Acunetix:

- Автоматичне сканування: виявлення широкого спектру вразливостей у веб-додатках [9].
- Інтеграція з CI/CD: автоматизація процесу тестування безпеки на всіх етапах розробки [9].
- Генерація звітів: створення детальних звітів про знайдені вразливості та рекомендацій щодо їх усунення [9].

Команда розробників інтегрує Acunetix з системою CI/CD (наприклад, Jenkins), що дозволяє автоматично запускати сканування безпеки після кожного зміненого коду застосунку. Це забезпечує постійний моніторинг безпеки веб-додатку та своєчасне виявлення нових вразливостей. Після сканування генеруються звіти, які надають детальну інформацію про знайдені проблеми та способи їх усунення [9].

1.4 Класифікація інструментів DAST

Інструменти для динамічного тестування безпеки можна класифікувати за декількома критеріями:

- Open-source та Commercial: відкриті інструменти (наприклад, OWASP ZAP) зазвичай безкоштовні та дозволяють користувачам змінювати вихідний код,

тоді як комерційні інструменти (наприклад, Burp Suite, Acunetix) надають розширені можливості та підтримку, але вимагають оплати за ліцензію [10].

- Standalone та Integrated: деякі інструменти працюють як автономні застосунки, тоді як інші інтегруються з IDE (інтегрованими середовищами розробки) або CI/CD системами [10].
- Автоматичні та Інтерактивні: автоматичні інструменти (наприклад, Acunetix) дозволяють швидко сканувати додатки на наявність вразливостей, тоді як інтерактивні інструменти (наприклад, Burp Suite) дозволяють більш детально аналізувати та маніпулювати трафіком [10].
- За цільовою платформою: інструменти DAST, орієнтовані на веб-додатки, спеціалізуються на виявленні вразливостей, властивих веб-технологіям. Інструменти DAST для мобільних додатків аналізують як код мобільного додатка, так і його взаємодію з бекенд-сервісами [5]. Інструменти, спрямовані на аналіз безпеки мережевих сервісів, серверів, баз даних та інших інфраструктурних компонентів, перевіряють налаштування серверів, мережеві протоколи, механізми аутентифікації та інші аспекти інфраструктури.

1.5 Порівняння інструментів DAST

Кожен з розглянутих інструментів має свої переваги та недоліки, детальніше порівняння наведено у таблиці 1.1.

Таблиця 1.1 - Порівняння інструментів

Критерій	OWASP ZAP	Burp Suite	Acunetix
Точність виявлення	Відома своєю підтримкою з боку open-source спільноти. Хороша виявлення базових вразливостей, таких як XSS та SQL і. Залежить від конфігурації та плагінів.	Професійна версія високо оцінюється за точність і розширені функції, включаючи розширені методи сканування та інструменти ручного тестування.	Відзначається високою точністю виявлення та всеохоплюючою базою даних вразливостей.

Продовження таблиці 1.1

Критерій	**OWASP ZAP**	**Burp Suite**	**Acunetix**
Підтримка векторів введення	Підтримує різні вектори введення, включаючи GET та POST, з додатковими плагінами для JSON та XML.	Пропонує широку підтримку різних векторів введення. Потрібні розширення для повної підтримки деяких векторів, таких як AMF та GWT.	Забезпечує надійну підтримку різноманітних векторів введення, що робить його ефективним у різних сценаріях тестування.
Продуктивність та швидкість	Ефективний, але може бути повільнішим у порівнянні з комерційними інструментами, особливо при масштабних скануваннях.	Відомий ефективною продуктивністю, особливо у Pro-версії, яка включає функції для оптимізації швидкості сканування.	Загалом пропонує швидкі можливості сканування з оптимізаціями для швидкості та точності виявлення вразливостей.
Інтерфейс користувача та зручність використання	Інтерфейс може бути менш інтуїтивним для новачків, але пропонує широку документацію та підтримку спільноти.	Пропонує досконалий і зручний інтерфейс з обширною документацією та великою спільнотою користувачів.	Має зручний інтерфейс з інтуїтивною навігацією та комплексними ресурсами підтримки.
Розширені функції	Включає функції, такі як автоматизовані сканери, плагіни та підтримка скриптів для налаштування сканів.	Забезпечує розширені функції, такі як Intruder, Repeater та обширні розширення для користувацької функціональності.	Пропонує розширені можливості сканування, інтеграцію з іншими інструментами та детальні звіти.

Вибір між OWASP ZAP, Burp Suite та Acunetix залежить від конкретних потреб і контексту використання. OWASP ZAP є відмінним безкоштовним open-source інструментом з широким спектром функцій, підходить для різних завдань тестування безпеки. Burp Suite Pro ідеально підходить для професіоналів, які шукають розширені функції та високу точність виявлення. Acunetix підходить для організацій, яким потрібне всеохоплююче, швидке та точне рішення для сканування, інтегроване в їхні процеси розробки.

1.6 Висновки до розділу

У цьому розділі було розглянуто основні методи та інструменти динамічного тестування безпеки програмних застосунків, а також принципи їх функціонування та можливості. Динамічне тестування безпеки є важливим елементом забезпечення безпеки програмного забезпечення, оскільки дозволяє виявляти вразливості у режимі реального часу. В ході роботи було досягнуто наступних висновків:

Методи динамічного тестування безпеки можна класифікувати за методами, такі як: фаззин, метод передбачає введення випадкових, некоректних або непередбачуваних даних у застосунок з метою виявлення потенційних вразливостей, таких як переповнення буфера або ін'єкції. Також було проаналізовано методи, що базуються на використанні підписів вразливостей, які швидко виявляють відомі загрози за допомогою постійно оновлюваної бази даних шаблонів. Евристичний аналіз, з іншого боку, застосовує алгоритми і правила для виявлення аномалій та підозрілих патернів у поведінці застосунку, що дозволяє ідентифікувати нові, раніше невідомі вразливості. Поведінковий аналіз фокусується на спостереженні за поведінкою застосунку в різних умовах, щоб виявити відхилення від очікуваної поведінки, а тестування валідації вхідних даних зосереджується на перевірці правильності обробки вхідних даних, запобігаючи атакам ін'єкцій. Окрім цього, було розглянуто аналіз конфігурацій, який перевіряє конфігураційні налаштування застосунку та середовища розгортання для виявлення потенційних вразливостей. Цей підхід допомагає виявити неправильні налаштування, які можуть призвести до порушень безпеки. Всі ці методи разом створюють комплексний підхід до тестування безпеки застосунків, забезпечуючи виявлення і усунення широкого спектра вразливостей.

Серед численних інструментів для динамічного тестування безпеки виділяються OWASP ZAP, Burp Suite та Acunetix. OWASP ZAP є відкритим та безкоштовним інструментом, що надає широкі можливості для автоматичного сканування та інтерактивного аналізу. Burp Suite пропонує потужні функції для інтерактивного аналізу та автоматичного сканування, але є комерційним рішенням.

Acunetix забезпечує високу автоматизацію процесу тестування та інтеграцію з CI/CD системами, що робить його ефективним для постійного моніторингу безпеки.

Вибір відповідних методів та інструментів залежить від конкретних потреб та умов, однак використання сучасних технологій DAST дозволяє значно підвищити рівень безпеки програмного забезпечення.

2. РОЗРОБКА ПІДСИСТЕМИ ДИНАМІЧОГО ТЕСТУВАННЯ БЕЗПЕКИ ЗАСТОСУНКІВ

2.1 Огляд системи тестування безпеки

Сучасні компанії-виробник програмного забезпечення створюють велику кількість програмної продукції, яка постійно потребують перевірки на вразливості, особливо якщо мова йде про власноруч розроблене програмне забезпечення. Існує кілька методів тестування безпеки додатків серед них основними є, такі як статичне (SAST) та динамічне (DAST) тестування безпеки. Кожен з цих методів зазвичай підтримується комерційними продуктами для захисту програмного забезпечення. Однак, багато компаній або ігнорують заходи безпеки, або використовують лише один метод тестування, що не забезпечує повний спектр заходів для перевірки безпеки продукту [12].

Попри те, що більшість компаній зосереджуються на статичному аналізі, динамічний аналіз може виявити типи вразливостей, які важко знайти за допомогою статичного аналізу. Включення динамічного тестування безпеки додатків дозволяє виявити більше вразливостей. Компанії, що поєднують динамічне сканування зі статичним, швидше закривають вразливості та відновлюють роботу своїх продуктів після інциденту на 24,5 дні швидше [12], ніж інші. Крім того, поєднання SAST та DAST забезпечує всебічний підхід до безпеки, що дозволяє виявити як структурні недоліки у коді, так і поведінкові вразливості. Цей комбінований підхід значно підвищує рівень захищеності програмного забезпечення, знижує ризики та підвищує довіру користувачів до продукту.

Тому для забезпечення максимальної безпеки пропонується використовувати систему, яка комбінує SAST та DAST. Такий підхід дозволяє компаніям виявляти і усувати вразливості на різних етапах життєвого циклу програмного забезпечення, забезпечуючи надійний захист від сучасних загроз. Пропонована система тестування безпеки має вирішувати наступні завдання:

- Ідентифікація вразливостей: виявлення як відомих, так і нових вразливостей у програмному забезпеченні.
- Аналіз та оцінка ризиків: оцінка виявлених вразливостей з точки зору їх потенційного впливу на безпеку системи.
- Автоматизація тестування: застосування автоматизованих інструментів для проведення регулярних та детальних тестів.
- Інтеграція з CI/CD: вбудовування в процеси безперервної інтеграції та доставки для забезпечення постійного моніторингу безпеки.
- Звітність: генерація звітів.

Очікується, що система буде відповідати наступним вимогам:

- Висока точність та ефективність: система повинна виявляти максимальну кількість вразливостей з мінімальним числом хибно-позитивних результатів.
- Масштабованість: здатність ефективно працювати з великою кількістю додатків та збільшенням обсягу тестування.
- Підтримка хмарної інфраструктури: можливість реалізації системи на хмарних платформах для забезпечення гнучкості та масштабованості.

Система повинна бути підходящою для опису за допомогою IaC (інфраструктура як код), що дозволить її реалізацію на хмарній інфраструктурі автоматизовано, з високою точністю та відтворюваністю. Це забезпечить легке налаштування, масштабування та управління ресурсами, дозволяючи швидко розгортати і оновлювати середовища розробки, тестування та продуктивного використання. Завдяки IaC, всі конфігурації зберігатимуться у вигляді коду, що дозволить відстежувати зміни, легко впроваджувати нові функції та забезпечувати відповідність політикам безпеки.

В свою чергу відповідно де теми даної роботи, потрібно детальніше саме розглянути підсистему динамічного тестування безпеки та її модулі. На рис. 2.1 представлена розроблена система тестування безпеки та виділена підсистема DAST

(Dynamic Application Security Testing), яка складається з двох модулів: модуль розгортання додатку та модуль DAST. Розглянемо кожен з них детальніше.

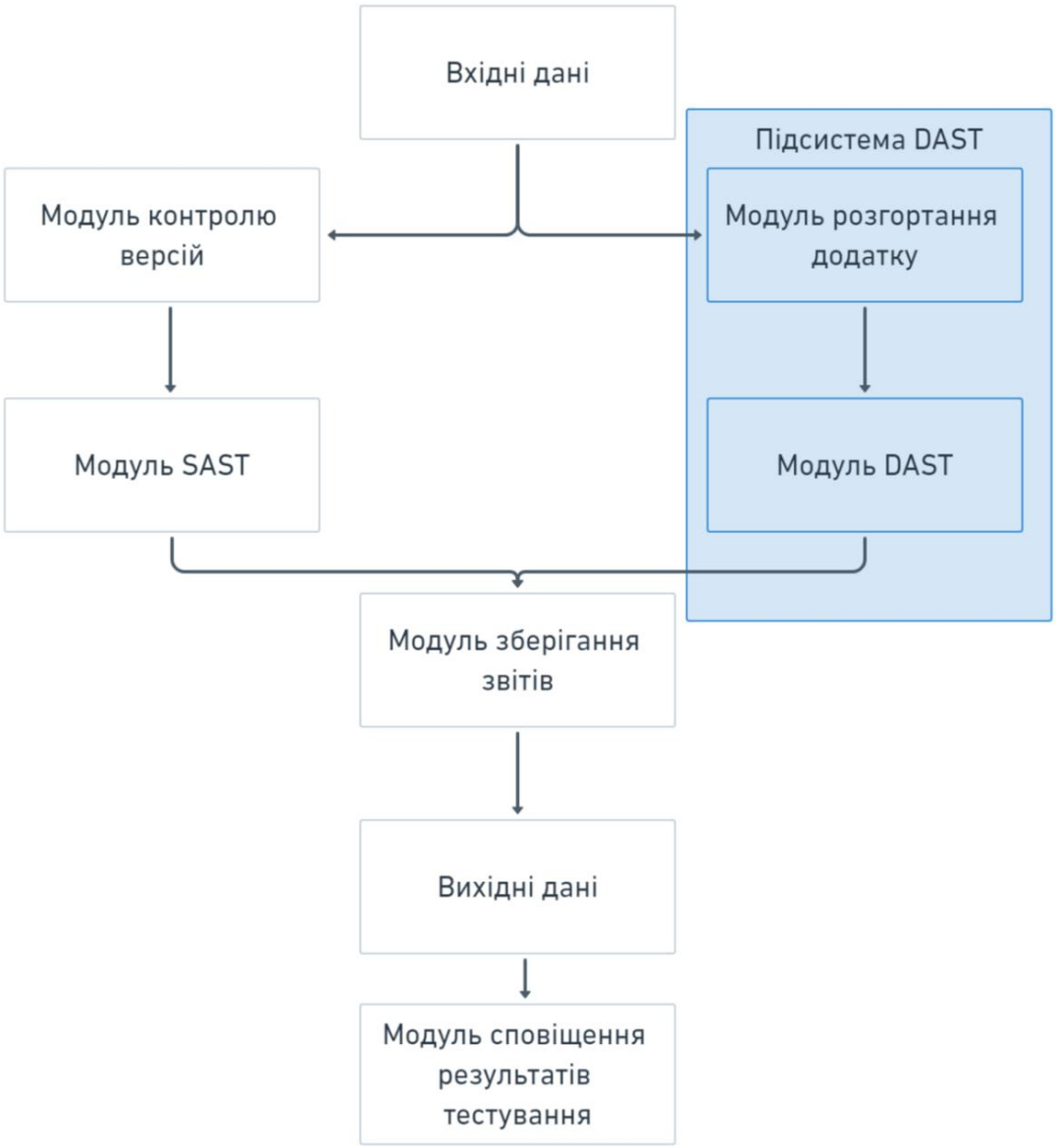


Рисунок 2.1 – Схема системи тестування безпеки застосунків

Модуль розгортання додатку відповідає за автоматизацію процесу розгортання веб-застосунку у відповідному середовищі. Цей модуль забезпечує виконання таких основних завдань:

- Ініціалізація середовища: створення та налаштування необхідної інфраструктури для запуску веб-застосунку, включаючи налаштування серверів, баз даних та мережевих компонентів.

- Розгортання застосунку: автоматичне завантаження та встановлення веб-застосунку на підготовлену інфраструктуру. Це може включати копіювання файлів, встановлення залежностей та запуск необхідних сервісів.
- Конфігурація застосунку: налаштування конфігураційних файлів та параметрів застосунку для забезпечення його коректної роботи у розгорнутому середовищі.

Модуль DAST призначений для виконання динамічного тестування безпеки веб-застосунку у розгорнутому середовищі. Основні завдання цього модуля включають:

- Проведення тестів на вразливості: виконання серії автоматизованих тестів для виявлення вразливостей у веб-застосунку.
- Аналіз результатів: збір та аналіз результатів тестування, визначення ступеня критичності виявлених вразливостей та їхнього впливу на безпеку застосунку.
- Генерація звітів: автоматичне створення звітів про результати тестування, що містять детальний опис виявлених вразливостей, рекомендації щодо їх усунення та інші важливі дані для розробників та адміністраторів.

Обидва модулі разом забезпечують повний цикл автоматизованого динамічного тестування безпеки веб-застосунку, від його розгортання до детального аналізу та звітування про вразливості, що значно підвищує рівень безпеки та стабільності програмного забезпечення.

2.2 Огляд процесу модуля динамічного тестування безпеки

Динамічне тестування здійснює аналіз HTTP/HTTPS запитів та відповідей та представляє з себе автоматизований елемент системи, який включає у собі підготовку інструмент тестування безпеки та передачі результатів тестування у місце зберігання та повідомлення про знайдені вразливості. Модуль динамічного тестування генерує запити до веб-додатку та на основі отриманих відповідей

робить висновки про наявність вразливостей, це можна представити у вигляді наступного рис. 2.2:

Рисунок 2.2 – Процес динамічного тестування безпеки

Огляд процесів представлних у підсистемі:

- Підготовка додатка. Першим кроком є підготовка до тестування. Це включає ідентифікацію компонентів додатка, таких як веб-сторінки, форми та поля введення, а також налаштування інструменту для доступу та тестування цих компонентів.

- Конфігурація інструменту. Інструмент тестування налаштовується для імітації атак на додаток. Це включає встановлення таких параметрів, як кількість запитів, типи атак, які потрібно виконати, та дані, які потрібно вводити.
- Імітація атак. Інструмент DAST надсилає запити до веб-додатка, імітуючи різні типи атак, такі як SQL-ін'єкції, XSS і міжсайтові підробки запитів (CSRF). Наприклад, у разі SQL-ін'єкції інструмент може спробувати додати до запиту спеціальні символи, які змінюють структуру SQL-запиту, і на основі отриманих результатів визначити, чи є додаток вразливим. Аналіз міжсайтового скриптингу (XSS) передбачає відправлення запитів із спеціальними скриптами та перевірку, чи виконується введений код на стороні клієнта. Також можна тестувати безпеку аутентифікації, перевіряючи механізми входу на предмет наявності слабких місць, таких як підбір паролів або уразливості у механізмах відновлення пароля.
- Аналіз результатів. Інструмент аналізує реакцію додатка на імітовані атаки та виявляє вразливості, такі як небезпечні механізми автентифікації, вразливості SQL-ін'єкцій або небезпечне управління сесіями.
- Звітування. Останнім кроком є створення звіту про вразливості, виявлені інструментом DAST. Цей звіт містить інформацію про вразливості та їх потенційний вплив на додаток, а також пропонує можливі дії з усунення. Після завершення створення звіту, він перенаправляється у інструмент управління вразливостями.

Також слід зазначити, що розглянутий процес є автоматичним скануванням. Автоматичне сканування дозволяє швидко перевірити додаток на наявність загальновідомих вразливостей, тоді як ручне тестування дозволяє детально аналізувати специфічні аспекти безпеки. Методи виявлення вразливостей включають автоматичне сканування, яке використовує бази даних відомих вразливостей для швидкого виявлення потенційних проблем, ручне тестування, що передбачає детальний аналіз певних аспектів безпеки шляхом проведення спеціалізованих атак або аналізу реакцій додатку на неочікувані дії. Комбінація

автоматичного та ручного тестування дозволяє отримати більш комплексну картину безпеки.

2.3. Ключові фактори вибору оптимального DAST інструменту

Задля визначення оптимального інструменту для використання у підсистемі динамічного тестування безпеки необхідно виконати експеремент з дослідженням інструментів DAST. Метою даного експерименту є проведення детального та всебічного порівняльного аналізу ефективності різних засобів динамічного тестування безпеки для виявлення вразливостей у веб-додатках. Завдання експерименту включають наступні ключові аспекти:

Оцінка точності виявлення вразливостей. Точність виявлення вразливостей є одним з основних критеріїв ефективності DAST інструментів. Важливо, щоб інструмент міг точно ідентифікувати існуючі загрози, без пропуску критичних вразливостей, які можуть бути використані зловмисниками для компрометації системи. Точність виявлення включає не тільки здатність знаходити відомі вразливості, але й ефективність у виявленні нових та невідомих загроз. В цьому контексті важливо оцінити, як кожен з обраних інструментів справляється з цією задачею і наскільки результати сканування відповідають реальним загрозам.

Аналіз продуктивності інструментів. Продуктивність інструментів динамічного тестування є важливим аспектом, який впливає на загальну ефективність процесу тестування. Це включає швидкість сканування, здатність обробляти великі обсяги даних і стабільність роботи під навантаженням. Аналіз продуктивності допомагає зрозуміти, наскільки швидко інструмент може завершити сканування веб-додатку і які ресурси для цього потрібні. Це особливо важливо у випадках, коли час тестування є критичним фактором, наприклад, у великих компаніях або при тестуванні в рамках CI/CD процесів, де тривалість тестів може впливати на загальний цикл розробки.

Виявлення сильних та слабких сторін кожного інструменту. Кожен інструмент має свої унікальні особливості, які можуть бути корисними у різних

сценаріях. Важливо визначити сильні сторони кожного інструменту, наприклад, високу точність виявлення певних типів вразливостей, зручність використання або інтеграцію з іншими інструментами та процесами. Також необхідно виявити слабкі сторони, такі як високий рівень хибно позитивних результатів, потреба у значних системних ресурсах або складність налаштування. Розуміння цих аспектів допоможе зробити обґрунтований вибір інструментів для конкретних потреб та умов використання.

В результаті проведеного експерименту очікується отримати комплексне уявлення про ефективність кожного з розглянутих інструментів та отримати загальне уявлення про стан ефективності сучасних інструментів безпеки.

2.4 Обґрунтування бенчмарку тестування

У цій роботі для порівняння ефективності інструментів тестування безпеки веб-додатків обирається Owasp benchmark. Цей вибір ґрунтується на результатах досліджень, а також на перевагах, які OWASP пропонує у порівнянні з іншими бенчмарками.

Аргументація на користь OWASP:

- Функціональність та надійність. OWASP - це повнофункціональний веб-застосунок з відкритим кодом. Він пропонує широкий спектр можливостей для оцінки інструментів тестування, включаючи точність, охоплення та швидкість.
- Візуалізація результатів. OWASP включає декілька генераторів таблиць оцінювання, які візуально ілюструють результати тестування. Це значно полегшує порівняння та аналіз даних.
- Дослідження Балуме Мбурано та Вейшенга Сі [13] ретельно порівнює два основних бенчмарки. WAVSEP та OWASP. Результати свідчать про значну перевагу OWASP у плані складності та реалістичності тестових сценаріїв.

Owasp benchmark містить експлуатовані тестові сценарії, які можуть бути проаналізовані як SAST, так і DAST інструментами. OWASP обирається як

бенчмарк для порівняння інструментів тестування безпеки веб-додатків завдяки своїй комплексності, функціональності, надійності та відповідності цілям роботи.

2.5. Обґрунтування вибору інструментів тестування

Для проведення тестування були обрані наступні інструменти [6]:

- OWASP ZAP. Відкритий інструмент для автоматизованого сканування та аналізу безпеки веб-додатків. Відомий за хорошу підтримку, обширний функціонал та ефективність.
- Wapiti. Відкритий інструмент для сканування веб-додатків на вразливості. Відзначається своєю здатністю знаходити широкий спектр уразливостей. Використовує консольний інтрефейс. Інструмент також має гнучкі та просто у виконанні налаштування.
- Arachni. Відкритий інструмент, відомий своєю потужністю та гнучкістю у виявленні вразливостей. Варто зазначити, що цей інструмент був обраний для тестування, через те, що у 2022 році підтримка його розробки була припинена. На його прикладі можна буде оцінити, як впливає припинення підтримки на ефективність інструмента.

Головним чином, яким обиралися інструменти це критерій бути open-source засобом.

2.6. Методологія проведення експерименту

2.6.1. Налаштування середовища тестування

Налаштування середовища тестування є важливим етапом у проведенні експерименту. Перш за все, потрібно було встановити новий образ операційної системи Kali Linux. Kali Linux є спеціалізованою версією Linux, яка призначена для тестування безпеки та містить велику кількість інструментів для проведення різноманітних тестувань. Після завантаження і встановлення образу Kali Linux, потрібно було перейти до налаштування самого середовища для подальшої роботи.

Наступним кроком було встановлення бенчмарку OWASP з GitHub [14]. Завантаження бенчмарку з офіційного репозиторію на GitHub вимагало налаштування Git на Kali Linux і клонування відповідного репозиторію.

Після успішного клонування бенчмарку OWASP, потрібно було перейти до інсталяції та конфігурації інструментів тестування. Для цього спочатку варто було переконатися, що всі необхідні залежності та пакети встановлені на систему. Деякі інструменти мали специфічні вимоги щодо середовища виконання, тому потрібно було встановити додаткові пакети (рис. 2.3).

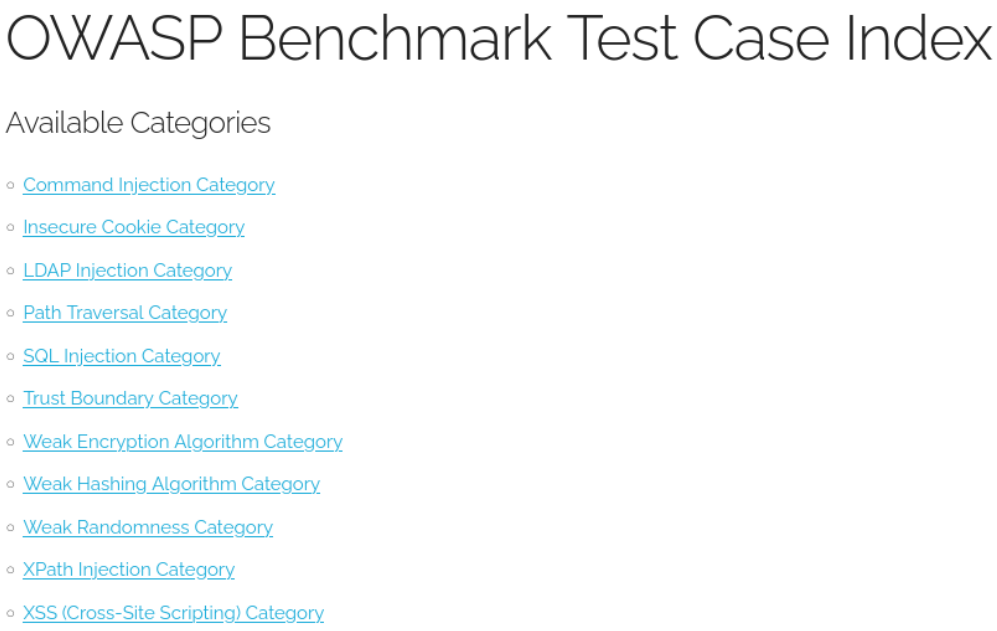


Рисунок 2.3 – Головна сторінка бенчмарку

Інсталяція інструментів тестування включала завантаження та налаштування DAST інструментів. Першим кроком було завантаження інструментів з офіційних вебсайтів, забезпечуючи використання найновіших та найбільш стабільних версій. Це гарантувало отримання усіх останніх оновлень безпеки та функціональності. Після завантаження, наступним етапом було встановлення інструментів. Цей процес потребував дотримання інструкцій, наданих розробниками, включаючи встановлення необхідних залежностей та бібліотек, що є обов'язковими для роботи інструментів. Наприклад, для OWASP ZAP це включає встановлення Java, а для

Wapiti — Python. Особливу увагу було приділено додаванню сертифікатів безпеки. Це забезпечувало можливість коректного аналізу HTTPS-з'єднань та уникнення помилок при роботі з захищеними ресурсами. Для цього використовувалися власні або довірені сертифікати, які необхідно було імпортувати до інструменту, що дозволило йому здійснювати перевірку захищених з'єднань без перешкод (рис. 2.4).

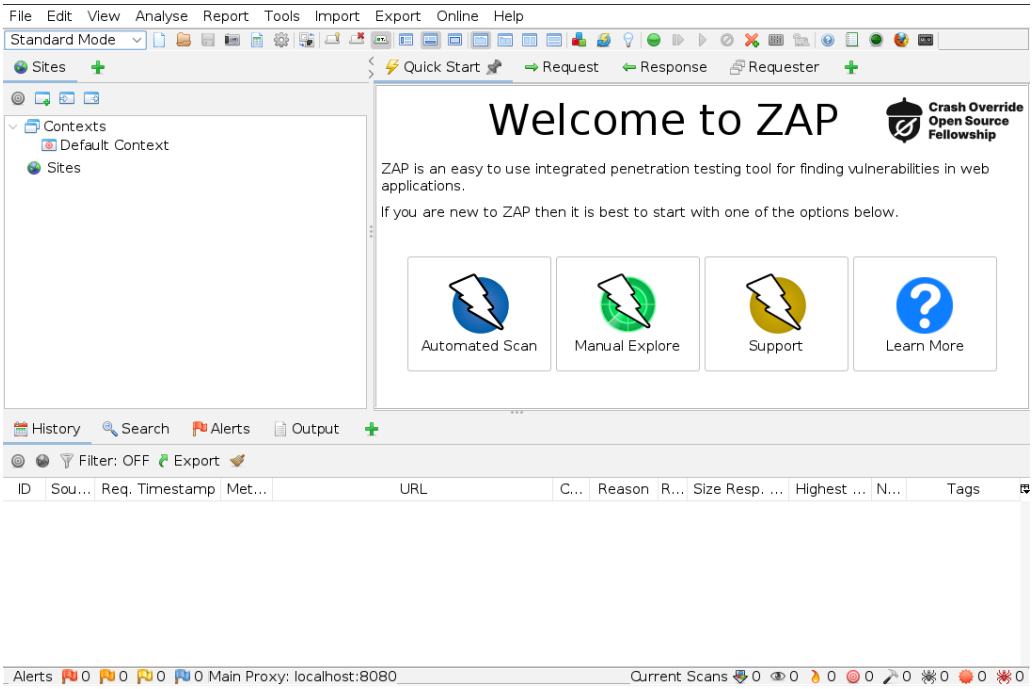


Рисунок 2.4 Головне вікно OWASP ZAP

Після завершення всіх цих кроків, середовище тестування було готове до проведення експериментів з DAST інструментами, що дало змогу перейти до наступного етапу дослідження та оцінки ефективності цих інструментів у виявленні вразливостей у реальних веб-додатках.

2.6.2. Процедура проведення тестування

Процедура проведення тестування включає:

- Попереднє налаштування інструментів(зміна параметрів, додавання скриптів-сценаріїв)
- Запуск автоматизованих тестів кожного інструменту.
- Збір та аналіз результатів тестування.

Після того, як інструменти DAST, ефективність яких буде досліджуватися, вже обрано, встановлено та налаштовано, необхідно розпочати тестування на безпеку. Після завершення сканування інструменти, які його виконували, завантажують документ з кінцевими результатами у форматі xml. Згенеровані файли розміщуються в директорії OWASP Benchmark, і починається генерація оціночних карток (scorecards) для підбиття підсумків і розрахунку необхідних метрик для подальшого аналізу.

2.7 Аналіз результатів тестування

Основні критерії включають [15]:

- Тип інтерфейсу. Цей критерій аналізує, чи має інструмент графічний інтерфейс користувача (GUI) або інтерфейс командного рядка (CLI). GUI зручний у використанні і легший для початківців, тоді як CLI може запропонувати більше можливостей для скриптування і гнучкість для досвідчених користувачів.
- Час, необхідний для сканування. Цей параметр вимірює час, необхідний інструменту для завершення сканування. Швидкість сканування може змінюватися в залежності від складності та розміру програми, що тестується. Більш швидкий час сканування зазвичай кращий, але не повинен компрометувати ретельність аналізу.
- Кількість виявлених попереджень про безпеку. Цей параметр підраховує загальну кількість попереджень або вразливостей, виявлених під час сканування. Більша кількість виявлених попереджень вказує на ретельність інструменту, але важливо збалансувати це з точністю, щоб уникнути хибно-позитивних результатів.
- Загальна кількість тестових випадків. Цей критерій відображає кількість тестових випадків або сценаріїв, які використовує інструмент під час аналізу.

Більша кількість тестових випадків може означати більш повне охоплення потенційних вразливостей.

- Істинно-позитивні (TP), хибно-позитивні (FP), істинно-негативні (TN), хибно-негативні (FN). Ці метрики є важливими для оцінки точності інструменту:

  а) Істинно-позитивні (TP). Реальні вразливості, правильно ідентифіковані інструментом.

  б) Хибно-позитивні (FP). Нереальні вразливості, помилково позначені як присутні інструментом.

  в) Істинно-негативні (TN). Нереальні вразливості, правильно ідентифіковані як відсутні інструментом.

  г) Хибно-негативні (FN). Реальні вразливості, помилково ідентифіковані як відсутні інструментом.

- Істинно-позитивний показник (TPR) і хибно-позитивний показник (FPR) Ці показники допомагають кількісно оцінити точність інструменту:

  а) Істинно-позитивний показник (TPR). Також відомий як recall, це пропорція реальних позитивів, правильно ідентифікованих інструментом (TP / (TP + FN)).

  б) Хибно-позитивний показник (FPR). Пропорція реальних негативів, помилково ідентифікованих як позитиви (FP / (FP + TN)).

- Оцінка. Оцінка зазвичай розраховується як різниця між TPR і FPR (Оцінка = TPR - FPR). Вища оцінка вказує на кращу продуктивність, збалансовуючи здатність виявляти істинно-позитивні результати і мінімізувати хибно-позитивні.

Першим застосунок на тестування є OWASP ZAP, після встановлення та налаштування і успішного тесту, представлені в таб. 2.1.

Таблиця 2.1 – Результати OWASP ZAP

№	Критерій	Результати
1	Тип інтерфейсу	Графічний інтерфейс користувача (GUI)
2	Час, необхідний для сканування (хв)	Близько 45 хвилин
3	Загальна кількість тестових випадків	2740
4	Кількість виявлених попереджень про безпеку	600 попереджень про безпеку
5	Істинно-позитивні, хибно-позитивні, істинно-негативні, хибно-негативні	Істинно-позитивні (TP) = 86, Хибно-позитивні (FP) = 2, Істинно-негативні (TN) = 1320, Хибно-негативні (FN) = 1332
6	Істинно-позитивний та хибно-позитивний показники (%)	Істинно-позитивний показник (TPR) = 6,06%, Хибно-позитивний показник (FPR) = 0,15%
7	Оцінка (%)	Оцінка = TPR - FPR = 5,91%

Другим обраним додатком був Wapiti. Так, як інтерфейс представляє з себе тільки командний рядок, початок роботи зайняв більше часу, результати представлені в таб. 2.2.

Таблиця 2.2 – Результати Wapiti

№	Критерій	Результати
1	Тип інтерфейсу	Інтерфейс командного рядка(CLI)
2	Час, необхідний для сканування (хв)	Приблизно 45 хвилин
3	Загальна кількість тестових випадків	2740
4	Кількість виявлених попереджень про безпеку	480 попереджень про безпеку
5	Істинно-позитивні, хибно-позитивні, істинно-негативні, хибно-негативні	Істинно-позитивні (TP) = 70, Хибно-позитивні (FP) = 5, Істинно-негативні (TN) = 1280, Хибно-негативні (FN) = 1385
6	Істинно-позитивний та хибно-позитивний показники (%)	Істинно-позитивний показник (TPR) = 5,06%, Хибно-позитивний показник (FPR) = 0,39%
7	Оцінка (%)	Оцінка = TPR - FPR = 4,67%

Останій іструмент, і головним чином показовий по рівню не ефективності, викликав багато проблем, через великий рівень використання ресурсів ПК та дуже великий час на проведення тестування (таб. 2.3).

Таблиця 2.3 – Результати Arachni

№	**Критерій**	**Результати**
1	Тип інтерфейсу	Інтерфейс командного рядка
2	Час, необхідний для сканування (хв)	Потребує близько 20 годин на повне сканування
3	Загальна кількість тестових випадків	2740
4	Кількість виявлених попереджень про безпеку	71 попереджень про безпеку
5	Істинно-позитивні, хибно-позитивні, істинно-негативні, хибно-негативні	Істинно-позитивні (TP) = 43, Хибно-позитивні (FP) = 0, Істинно-негативні (TN) = 1325, Хибно-негативні (FN) = 1372
6	Істинно-позитивний та хибно-позитивний показники (%)	Істинно-позитивний показник (TPR) = 3,10%, Хибно-позитивний показник (FPR) = 0%
7	Оцінка (%)	Оцінка = TPR- FPR = 3.10 %

2.8. Висновки до розділу

При проведенні аналізу результатів сканування за допомогою трьох різних інструментів DAST (OWASP ZAP, Wapiti, Arachni), було отримано наступні дані щодо часу сканування та загальної оцінки ефективності кожного інструменту.

OWASP ZAP має час сканування близько 45 хвилин, що робить його досить швидким інструментом для проведення тестування. Загальна оцінка ефективності для OWASP ZAP становить 8,32%, що є найвищою оцінкою серед усіх трьох інструментів. Це вказує на високу ефективність OWASP ZAP у виявленні істинно-позитивних випадків та мінімізації хибно-позитивних результатів, що робить його найкращим вибором для швидкого та точного сканування.

Wapiti також показує час сканування приблизно 45 хвилин, що є порівняльним з OWASP ZAP. Проте, загальна оцінка ефективності для Wapiti становить 4,67%, що є нижчою, ніж у OWASP ZAP. Це вказує на те, що Wapiti, хоча і є ефективним інструментом, поступається OWASP ZAP у точності виявлення загроз.

Arachni, з іншого боку, має значно довший час сканування, іноді до 20 годин, що може бути неефективним для швидкого тестування. Загальна оцінка ефективності для Arachni становить лише 3,10%, що є найнижчою серед усіх трьох інструментів. Це робить Arachni менш привабливим для використання в порівнянні з іншими інструментами через його довгий час сканування та нижчу точність.

Отже, виходячи з отриманих результатів, можна зробити висновок, що OWASP ZAP є найкращим вибором для проведення динамічного аналізу безпеки додатків завдяки своїй високій ефективності та швидкості сканування. Wapiti також є гарним вибором, але поступається OWASP ZAP у точності. Arachni, через свій довгий час сканування та нижчу ефективність, може бути менш ефективним інструментом для швидкого тестування.

3. РЕАЛІЗАЦІЯ ПІДСИСТЕМИ ДИНАМІЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ ЗАСТОСУНКІВ

3.1 Обґрунтування вибору середовища для реалізації

Задля реалізації системи та власне самої підсистеми, потрібно визначитися з тим, як та де вона буде реалізована. Враховуючи попередньо визначені задачі та вимоги, обраним рішенням є платформа AWS (Amazon Web Services). AWS пропонує широкий спектр сервісів для побудови, розгортання та масштабування додатків, що робить її ідеальною для розробки складних систем, таких як система тестування безпеки додатків.

Перед тим, як повноцінно розглянути чим є платформа AWS, потрібно визначитися, що таке хмарна інфраструктура

Хмарна інфраструктура - це сукупність апаратних та програмних ресурсів, які надаються через Інтернет, забезпечуючи користувачам доступ до обчислювальних потужностей, зберігання даних та різноманітних IT-сервісів без необхідності купувати та обслуговувати фізичне обладнання. В основі хмарної інфраструктури лежать великі центри обробки даних, що складаються з серверів, систем зберігання даних та мережевих компонентів, які об'єднані в єдину систему. Користувачі можуть орендувати ці ресурси на вимогу, сплачуючи лише за ті обчислювальні потужності та сервіси, які вони використовують [16]. Це дозволяє значно знизити витрати на IT-інфраструктуру та забезпечити гнучкість у масштабуванні ресурсів відповідно до потреб бізнесу. Основними компонентами хмарної інфраструктури є інфраструктура як сервіс (IaaS), платформа як сервіс (PaaS) та програмне забезпечення як сервіс (SaaS). IaaS надає базові обчислювальні ресурси, такі як віртуальні машини та сховища даних. PaaS забезпечує платформу для розробки, тестування та розгортання програм. SaaS пропонує готові до використання програмні додатки через Інтернет. Хмарна інфраструктура також включає в себе засоби для забезпечення безпеки, управління та моніторингу ресурсів, що гарантує надійність та ефективність її роботи [16].

AWS – це хмарна платформа, яка надає понад 200 повнофункціональних сервісів з центрів обробки даних по всьому світу. Вона забезпечує високий рівень доступності, безпеки та масштабованості, що дозволяє створювати надійні та безпечні рішення. Однією з ключових переваг AWS є її гнучкість і можливість адаптації під будь-які потреби бізнесу [17].

Основні причини вибору AWS:

- Широкий набір сервісів. AWS пропонує різноманітні сервіси для обробки, зберігання, управління даними, аналітики, штучного інтелекту, машинного навчання, IoT та багато іншого [17].
- Масштабованість. Інфраструктура AWS дозволяє швидко масштабувати ресурси вгору або вниз відповідно до потреб проєкту [17].
- Безпека. AWS забезпечує комплексні заходи безпеки, включаючи шифрування даних, контроль доступу та відповідність стандартам безпеки [17].
- Економія. Модель оплати за використання дозволяє знизити витрати, оскільки ви платите лише за ті ресурси, які використовуєте [17].
- Глобальна присутність. AWS має глобальну мережу дата-центрів, що забезпечує низькі затримки та високу доступність для користувачів у різних регіонах [17].

Ще однією важливою вимогою до системи є можливість опису її у вигляді коду за допомогою принципу IaC (Infrastructure as Code).

IaC – це підхід до управління та розгортання інфраструктури через файли конфігурації, які можуть бути версійовані, протестовані та автоматизовані [18]. Це дозволяє забезпечити:

- Автоматизацію. Зменшення кількості ручних дій і підвищення продуктивності за рахунок автоматизації розгортання інфраструктури [18].
- Консистентність. Забезпечення однакових середовищ для розробки, тестування та виробництва, що зменшує ризики помилок і несподіваних проблем [18].

- Відтворюваність. Легке відтворення та розгортання інфраструктури в нових середовищах або регіонах [18].
- Відслідковуваність. Здатність відслідковувати зміни в конфігураціях і версійність інфраструктури [18].

Для опису інфраструктури був обраний нативний сервіс AWS CloudFormation.

AWS CloudFormation – це сервіс, який дозволяє описувати та управляти пов'язаними ресурсами AWS, записуючи їх як шаблони у вигляді текстових файлів JSON або YAML [19]. Основні переваги CloudFormation включають:

- Інфраструктура як Код. Можливість визначення всієї інфраструктури у вигляді коду, що спрощує управління та автоматизацію [19].
- Повторюваність. Шаблони CloudFormation можна використовувати багаторазово для створення однакових середовищ [19].
- Автоматизація. Автоматизація процесу розгортання, оновлення та видалення ресурсів AWS [19].
- Інтеграція. Глибока інтеграція з іншими сервісами AWS, такими як IAM, S3, EC2, RDS, що спрощує управління всією інфраструктурою [19].

Для реалізації системи було створено схему інфраструктури, що включає головним чином наступні компоненти:

- EventBridge Event Rule. Правило подій EventBridge для відстеження стану CodeBuild.
- InitPipeline. Компонент ініціалізації конвеєра, включає ролі CloudFormation та CodePipeline.
- SASTScan. Компонент для запуску SAST сканування.
- DASTScan. Компонент для запуску DAST сканування.
- S3 Bucket. Сховище для зберігання результатів сканування.

Ця схема інфраструктури дозволяє забезпечити надійність, масштабованість і безпеку системи, використовуючи сучасні практики DevSecOps та інструменти AWS. На рис. 3.1 представлено шаблон взаємодії між різними сервісами AWS, забезпечуючи повний цикл автоматизації тестування безпеки.

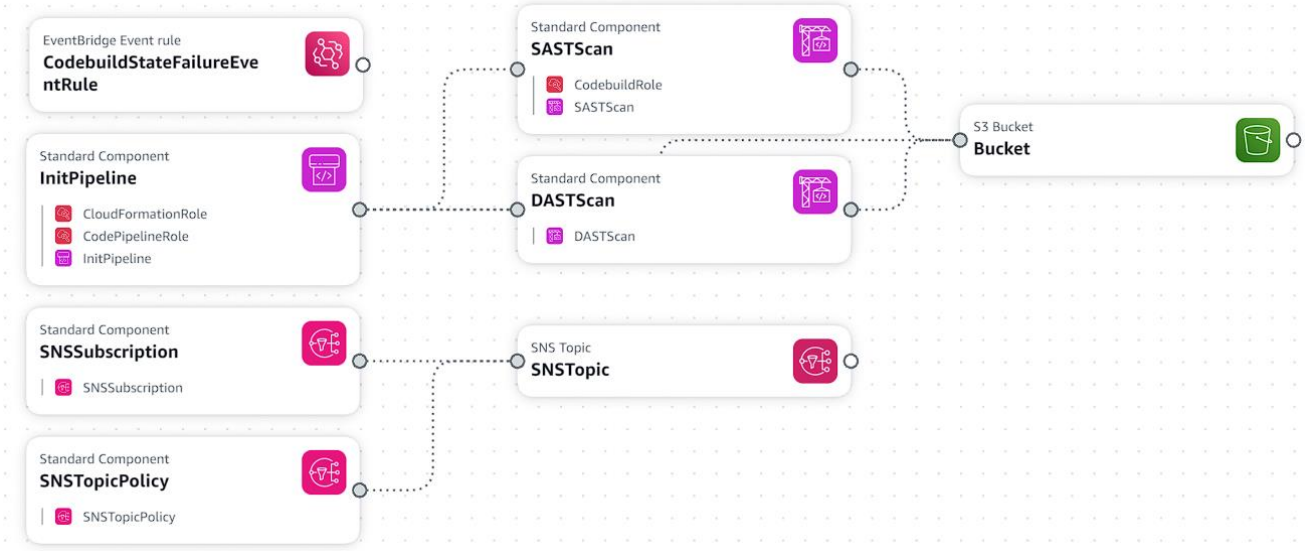


Рисунок 3.1 – Схема шаблону CloudFormation

3.2 Реалізація за допомогою IaC підсистеми динамічного тестування безпеки

Для налаштування автоматизованого динамічного сканування безпеки додатків (DAST) у CI/CD пайплайні, використовується інструмент AWS CodeBuild разом із AWS CloudFormation. Це забезпечує автоматичне виконання перевірок безпеки кожного разу, коли відбувається оновлення коду.

Елемент шаблону CloudFormation, що описує конфігурацію DAST скану в пайплайні:

```
DASTScan:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:
        Type: S3
        Location: !Ref Bucket
        Name: DASTCheck
        Packaging: ZIP
      BadgeEnabled: false
      Name: DASTCheck
      Description: Запуск DAST сканування
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:2.0
```

```
        Type: LINUX_CONTAINER
        EnvironmentVariables:
          - Name: APP_URL
            Value: http://example.com
      ServiceRole: !Ref CodebuildRole
      Source:
        Type: CODECOMMIT
        Location: !GetAtt CodeCommitRepository.CloneUrlHttp
        BuildSpec: DASTconfig.yaml
```

Цей елемент шаблону CloudFormation описує конфігурацію проекту CodeBuild для виконання DAST-скану в пайплайні. Він містить усі необхідні налаштування для збереження артефактів, налаштування середовища виконання, а також параметри вихідного коду та ролі сервісу.

Наступний блок конфігурації — це файл DASTconfig.yaml, який використовується для налаштування процесу сканування безпеки. Цей файл визначає фази встановлення, збірки та пост-збірки, забезпечуючи встановлення необхідних залежностей, запуск сканування та обробку результатів.

```
phases:
  install:
    runtime-versions:
      java: corretto11
    commands:
      - echo "Installing jq and necessary dependencies"
      - yum install jq -y
      - echo "Downloading and unpacking OWASP ZAP"
      - wget
https://github.com/zaproxy/zaproxy/releases/download/v2.15.0/ZAP_2.1
5.0_Linux.tar.gz
      - tar -xf ZAP_2.15.0_Linux.tar.gz
      - rm ZAP_2.15.0_Linux.tar.gz
      - export PATH=$PWD/ZAP_2.15.0:$PATH
      - zap.sh -cmd -version
      - echo "Ensure all necessary ZAP add-ons are installed and up-
to-date"
      - zap.sh -cmd -addonupdate

  build:
    commands:
      - echo "Starting OWASP ZAP scan"
      - zap.sh -cmd -quickurl $APP_URL -quickout $HOME/report.html

  post_build:
    commands:
      - echo "Checking generated report"
      - ls -l $HOME/report.html
      - cat $HOME/report.html
```

```
artifacts:
  files:
    - $HOME/report.html
  discard-paths: yes
```

У фазі встановлення визначаються версії середовища виконання, зокрема використовується Java версії corretto11. Спочатку виконується команда, яка встановлює jq та інші необхідні залежності через пакетний менеджер yum. Далі завантажується та розпаковується архів OWASP ZAP, після чого виконується команда, яка додає шлях до OWASP ZAP у змінну середовища PATH. Це дозволяє використовувати команди OWASP ZAP без необхідності вказувати повний шлях до їх виконуваних файлів. Для перевірки коректності встановлення виконується команда zap.sh -cmd -version, яка виводить версію встановленого OWASP ZAP. Крім того, здійснюється оновлення всіх необхідних додатків OWASP ZAP за допомогою команди zap.sh -cmd -addonupdate, що забезпечує актуальність та повноту функціоналу інструменту.

Після встановлення залежностей настає фаза збірки, де запускається саме сканування за допомогою OWASP ZAP. Використовується команда zap.sh -cmd -quickurl, яка здійснює швидке сканування вказаного URL (в даному випадку URL додатку задається змінною середовища $APP_URL). Результати сканування зберігаються у файл report.html, розташований в домашній директорії користувача.

На завершальному етапі, фазі пост-збірки, здійснюється перевірка результатів сканування. Спочатку перевіряється наявність файлу звіту за допомогою команди ls -l $HOME/report.html, яка виводить список файлів у директорії. Далі виконується команда cat $HOME/report.html, яка виводить вміст файлу звіту на консоль для подальшого аналізу.

Після завершення всіх фаз процесу, файл конфігурації визначає артефакти, які будуть збережені. У даному випадку це файл report.html, що містить результати сканування. Вказується, що шляхи до файлів будуть відкинуті, залишаючи лише сам файл звіту, що спрощує доступ до нього.

Ця конфігурація забезпечує автоматизацію процесу сканування безпеки, встановлюючи необхідні інструменти, виконуючи сканування та зберігаючи

результати для подальшого аналізу, що дозволяє ефективно контролювати безпеку додатку на всіх етапах розробки.

3.3 Обраний об'єкт тестування

Перш за все, необхідно обрати об'єкт тестування – веб-застосунок, який буде розгорнутий та протестований у процесі виконання динамічного тестування безпеки. В результаті попереднього відбору серед кількох open-source застосунків, написаних різними мовами програмування, був обраний OWASP Juice Shop [20].

OWASP Juice Shop є спеціально створеним веб-застосунком для експериментів із засобами тестування та перевірки власних навичок у тестуванні на вразливості. Цей застосунок широко використовується у навчальних цілях завдяки його відкритому коду та наявності багатьох вразливостей, що дозволяє тестувальникам безпечно тренувати свої навички у виявленні та усуненні проблем безпеки.

Основною причиною вибору OWASP Juice Shop є його добре документована кількість вразливостей, яка на даний момент складає 107 [20]. Це дозволяє проводити всебічне тестування різних аспектів безпеки. Вразливості відповідно до документації Juice Shop [20] поділяються на наступні класи (таб. 3.1).

Таблиця 3.1 – Класи вразливостей

Категорія вразлисовостей	**Опис класу**	**Кількість вразливостей**
Broken Access Control	Вразливості, які дозволяють неавторизованим користувачам отримати доступ до ресурсів або функціональностей, що мають бути обмежені.	11
Broken Anti Automation	Вразливості, що дозволяють автоматизованим ботам або скриптам взаємодіяти з веб-застосунком у спосіб, який мав би бути заблокований.	4
Broken Authentication	Недоліки в процесах аутентифікації, які дозволяють зловмисникам видавати себе за інших користувачів.	9

Продовження таблиці 3.1

Категорія вразлисовостей	**Опис класу**	**Кількість вразливостей**
Cryptographic Issues	Проблеми, пов'язані з використанням слабких або неправильно реалізованих криптографічних алгоритмів, що призводить до компрометації даних.	5
Improper Input Validation	Вразливості, які виникають через недостатню або некоректну перевірку вхідних даних, що дозволяє виконувати шкідливі операції.	12
Injection	Вразливості, що дозволяють впроваджувати та виконувати небажані команди або коди через вхідні дані, наприклад, SQL ін'єкції.	11
Insecure Deserialization	Вразливості, які дозволяють виконувати небезпечні операції під час десеріалізації даних, що може призвести до виконання довільного коду.	2
Miscellaneous	Різні вразливості, які не підпадають під інші категорії, але все ж можуть мати негативний вплив на безпеку.	7
Security Misconfiguration	Неправильні налаштування безпеки, які можуть призвести до вразливостей, такі як залишення облікових записів за замовчуванням або увімкнення непотрібних функцій.	4
Security through Obscurity	Практики, які покладаються на приховування інформації для забезпечення безпеки, що може бути недостатньо ефективним захистом.	3
Sensitive Data Exposure	Витоки чутливої інформації через неправильну обробку або недостатній захист даних.	17
Unvalidated Redirects	Вразливості, які дозволяють перенаправляти користувачів на небажані або шкідливі веб-сайти через неконтрольовані або невалідувані перенаправлення.	2
Vulnerable Components	Використання бібліотек або компонентів з відомими вразливостями, що можуть бути використані для атаки.	9
XSS	Вразливості, які дозволяють впроваджувати та виконувати шкідливий скрипт у веб-сторінці, що взаємодіє з користувачем.	9
XXE	Вразливості, що дозволяють зловмисникам втручатися в обробку XML-даних і виконувати шкідливі операції через зовнішні сутності в XML.	2

Вибір OWASP Juice Shop як об'єкта тестування надає можливість перевірити ефективність та можливості інструментів динамічного тестування безпеки у реальних умовах, і в особливості, дозволить цілісно зробити аналіз ефективності проведеного тестування для цієї роботи.

3.4 Демонстрація роботи підсистеми динамічного тестування безпеки

Процес динамчіного тестування починається, з того, що AWS CodePipeline переходить до етапу створення середовища для розгортання застосунка. Це середовище створюється за допомогою CloudFormation, який генерує та виконує зміни на основі визначених шаблонів. У даній демонстрації середовище використовується для розміщення вище згаданого веб-додатку, який вже збережений у репозиторії AWS CodeCommit. Додаток розгортається за допомогою CloudFormation, що автоматизує процес розгортання та забезпечує консистентність середовища. Вигляд створених ресурсів (рис. 3.3)

Resources (16)

Search resources

Logical ID	Type	Status
ALBListener	AWS::ElasticLoadBalancingV2::Listener	CREATE_COMPLETE
ALBTargetGroup	AWS::ElasticLoadBalancingV2::TargetGroup	CREATE_COMPLETE
ApplicationLoadBalancer	AWS::ElasticLoadBalancingV2::LoadBalancer	CREATE_COMPLETE
AttachGateway	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE
ELBSecurityGroup	AWS::EC2::SecurityGroup	CREATE_COMPLETE
InstanceSecurityGroup	AWS::EC2::SecurityGroup	CREATE_COMPLETE
InternetGateway	AWS::EC2::InternetGateway	CREATE_COMPLETE
LaunchConfig	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
PublicRoute	AWS::EC2::Route	CREATE_COMPLETE
PublicRouteTable	AWS::EC2::RouteTable	CREATE_COMPLETE
PublicSubnet1RouteTableAssociation	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
PublicSubnet2RouteTableAssociation	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
ServerGroup	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE
myVPC	AWS::EC2::VPC	CREATE_COMPLETE
subnet1	AWS::EC2::Subnet	CREATE_COMPLETE
subnet2	AWS::EC2::Subnet	CREATE_COMPLETE

Рисунок 3.3 – Створені ресурси для застосунку

Для забезпечення доступу до додатку через інтернет використовується AWS Application Load Balancer (ALB). ALB розподіляє вхідний трафік на кілька екземплярів додатку, що працюють у групі авто масштабування, забезпечуючи високу доступність та масштабованість.

URL-адреса ALB (рис 3.4) передається до наступного етапу в AWS CodeBuild, до створеної віртуальної машини, після автоматичного налаштування виконується динамічне тестування безпеки додатку.

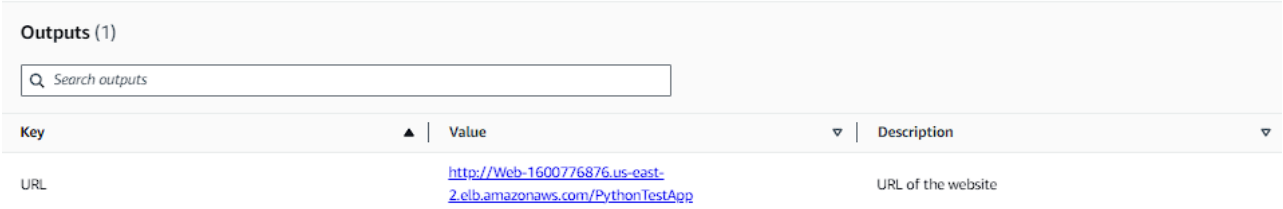


Рисунок 3.4 – Вихідні дані. Посилання на веб-застосунок

Етап DAST включає виконання сканування безпеки на вказаній URL-адресі додатку. Це сканування виявляє потенційні вразливості та генерує звіт про результати. Коли виявляються вразливості, вони позначаються у звіті для подальшого аналізу та усунення. Якщо етап DAST завершується успішно, процес переходить до етапу затвердження для продукційного середовища. Цей етап вимагає ручного схвалення, що дозволяє команді забезпечення якості та безпеки переглянути результати сканування, оцінити ризики, при необхідності внести виправлення та прийняти рішення про розгортання додатку у продукційне середовище. Нище наведений приклад, логу процесу підготовки та проведення тестування, який можна переглядати у режимі реального часу (рис. 3.5)

```
[Container] 2024/06/09 23:12:57.253653 Running on CodeBuild On-demand
[Container] 2024/06/09 23:12:57.253754 Waiting for agent ping
[Container] 2024/06/09 23:12:58.458553 Waiting for DOWNLOAD_SOURCE
[Container] 2024/06/09 23:12:58.664841 Phase is DOWNLOAD_SOURCE
[Container] 2024/06/09 23:12:58.665739 CODEBUILD_SRC_DIR=/codebuild/output/src1925766347/src
[Container] 2024/06/09 23:12:58.666296 YAML location is /codebuild/output/src1925766347/src/DASTBuildspec.yaml
[Container] 2024/06/09 23:12:58.667983 Setting HTTP client timeout to higher timeout for S3 source
[Container] 2024/06/09 23:12:58.668155 Processing environment variables
[Container] 2024/06/09 23:12:58.715606 Selecting 'java' runtime version 'corretto11' based on manual selections...
[Container] 2024/06/09 23:12:58.716376 Running command echo "Installing corretto(OpenJDK) version 11 ..."
Installing corretto(OpenJDK) version 11 ...

[Container] 2024/06/09 23:12:58.722756 Running command export JAVA_HOME="$JAVA_11_HOME"

[Container] 2024/06/09 23:12:58.728832 Running command export JRE_HOME="$JRE_11_HOME"

[Container] 2024/06/09 23:12:58.734280 Running command export JDK_HOME="$JDK_11_HOME"

[Container] 2024/06/09 23:12:58.739697 Running command for tool_path in "$JAVA_HOME"/bin/*;
 do tool=`basename "$tool_path"`;
  if [ $tool != 'java-rmi.cgi' ];
  then
   rm -f /usr/bin/$tool /var/lib/alternatives/$tool \
    &amp;#38;&amp;#38; update-alternatives --install /usr/bin/$tool $tool $tool_path 20000;
  fi;
done

[Container] 2024/06/09 23:12:58.833377 Moving to directory /codebuild/output/src1925766347/src
[Container] 2024/06/09 23:12:58.836430 Unable to initialize cache download: no paths specified to be cached
[Container] 2024/06/09 23:12:58.836483 Configuring ssm agent with target id: codebuild:e9ef55ac-df4e-4bed-bbed-e13b07e059c4
[Container] 2024/06/09 23:12:58.836757 Successfully updated ssm agent configuration
[Container] 2024/06/09 23:12:58.837027 Registering with agent
[Container] 2024/06/09 23:12:58.868522 Phases found in YAML: 3
[Container] 2024/06/09 23:12:58.868640  POST_BUILD: 3 commands
[Container] 2024/06/09 23:12:58.868650  INSTALL: 10 commands
[Container] 2024/06/09 23:12:58.868708  BUILD: 2 commands
[Container] 2024/06/09 23:12:58.869007 Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
[Container] 2024/06/09 23:12:58.869019 Phase context status code:  Message:
[Container] 2024/06/09 23:12:58.929856 Entering phase INSTALL
[Container] 2024/06/09 23:12:58.930358 Running command echo "Installing jq and necessary dependencies"
Installing jq and necessary dependencies

[Container] 2024/06/09 23:12:58.936828 Running command yum install jq -y
Loaded plugins: ovl, priorities
230 packages excluded due to repository priority protections
Package jq-1.5-1.amzn2.0.2.x86_64 already installed and latest version
Nothing to do
```

Рисунок 3.5 – Лог роботи інструменту

Після завершення динамічного тестування безпеки веб-застосунку за допомогою OWASP ZAP, ми отримали детальний звіт, який автоматично зберігається у сховищі S3 (рис 3.5).

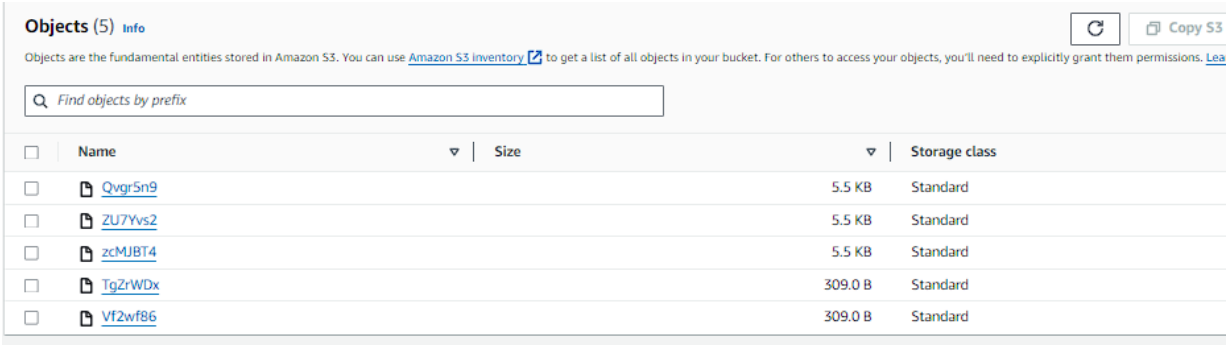


Рисунок 3.5 – Сховище DAST звітів

Звіт містить загальний огляд перевірених сайтів, дату і час генерації, а також версію використаного інструменту ZAP. Загальний вигляд усієї підсистеми у пайплайні (рис. 3.6)

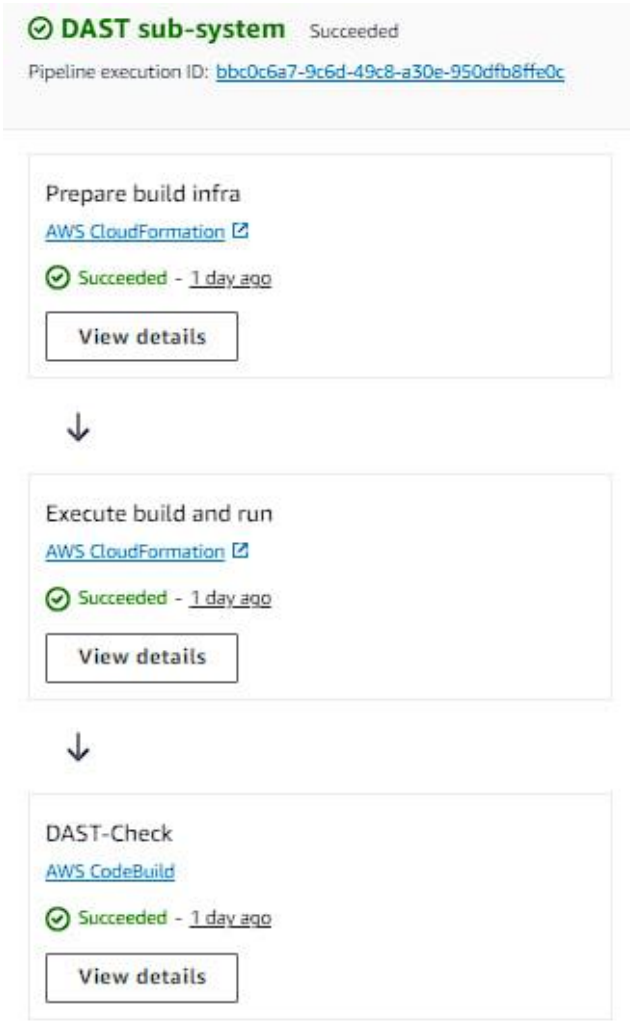


Рисунок 3.6 – Успішно виповнені етапи підсистеми

Крім того, у звіті представлено узагальнені дані про виявлені вразливості, класифіковані за рівнями ризику: високий, середній, низький та інформаційний (рис. 3.7).

Sites: http://Web-2140011250.us-east-2.elb.amazonaws.com https://cdnjs.cloudflare.com https://juice-shop.herokuapp.com

Generated on Tue, 11 Jun 2024 13:05:15

ZAP Version: 2.15.0

ZAP is supported by the Crash Override Open Source Fellowship

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	5
Low	5
Informational	4

Рисунок 3.7 – Заголовок звіту

Однією з виявлених вразливостей високого рівня є "Cloud Metadata Potentially Exposed". Ця вразливість полягає у можливості зловмисника отримати доступ до метаданих хмарних сервісів через некоректно налаштований сервер NGINX. Зловмисник може використати внутрішню IP-адресу у заголовку Host, щоб отримати доступ до метаданих, які можуть містити критично важливу інформацію для компрометації системи (рис. 3.8).

High	Cloud Metadata Potentially Exposed
Description	The Cloud Metadata Attack attempts to abuse a misconfigured NGINX server in order to access the instance metadata maintained by cloud service providers such as AWS, GCP and Azure. All of these providers provide metadata via an internal unroutable IP address '169.254.169.254' - this can be exposed by incorrectly configured NGINX servers and accessed by using this IP address in the Host header field.
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/latest/meta-data/
Method	GET
Parameter	
Attack	169.254.169.254
Evidence	
Other Info	Based on the successful response status code cloud metadata may have been returned in the response. Check the response data to see if any cloud metadata has been returned. The meta data returned can include information that would allow an attacker to completely compromise the system.
Instances	1
Solution	Do not trust any user data in NGINX configs. In this case it is probably the use of the $host variable which is set from the 'Host' header and can be controlled by an attacker.
Reference	https://www.nginx.com/blog/trust-no-one-perils-of-trusting-user-input/
CWE Id	
WASC Id	
Plugin Id	90034

Рисунок 3.8 – Елемент звіту: інформація про вразливість

Для вирішення цієї проблеми рекомендується уникати використання даних користувача у конфігураціях NGINX. Зокрема, необхідно уникати використання змінної $host, яка задається з заголовка Host і може бути контрольована атакуючим. Дотримання цих рекомендацій допоможе запобігти можливим атакам та забезпечить безпеку веб-застосунку.

Звіт OWASP ZAP надає цінну інформацію для розробників та системних адміністраторів, дозволяючи виявляти та усувати критичні вразливості. Виконання рекомендацій зі звіту сприяє підвищенню загального рівня безпеки і захисту веб-застосунків від потенційних загроз.

3.4 Аналіз проведеного тестування

У результаті проведеного динамічного тестування, аналіз звіту продемонструвати, що було виявлено 11 унікальних вразливостей, та 4 примітки до безпеки (таб. 3.2).

Таблиця 3.2 – Віднайдені вразливості

Назва вразливості	**Клас вразливості**	**Рівнь ризику**
Cloud Metadata Potentially Exposed	Security Misconfiguration	Високий
Content Security Policy (CSP) Header Not Set	Security Misconfiguration	Середній
Cross-Domain Misconfiguration	Security Misconfiguration	Середній
Missing Anti-clickjacking Header	Security Misconfiguration	Середній
Session ID in URL Rewrite	Broken Authentication	Середній
Vulnerable JS Library	Vulnerable Components	Середній
Cross-Domain JavaScript Source File Inclusion	Cross-Site Scripting (XSS)	Низький
Private IP Disclosure	Information Disclosure	Низький
Strict-Transport-Security Header Not Set	Security Misconfiguration	Низький
Timestamp Disclosure - Unix	Information Disclosure	Низький
X-Content-Type-Options Header Missing	Security Misconfiguration	Низький
Information Disclosure - Suspicious Comments	Information Disclosure	Інформаційний
Modern Web Application	Miscellaneous	Інформаційний
Re-examine Cache-control Directives	Security Misconfiguration	Інформаційний
Retrieved from Cache	Security Misconfiguration	Інформаційний

Наступна діаграма відображає кількість віднайдених вразливостей по класу (рис. 3.9)

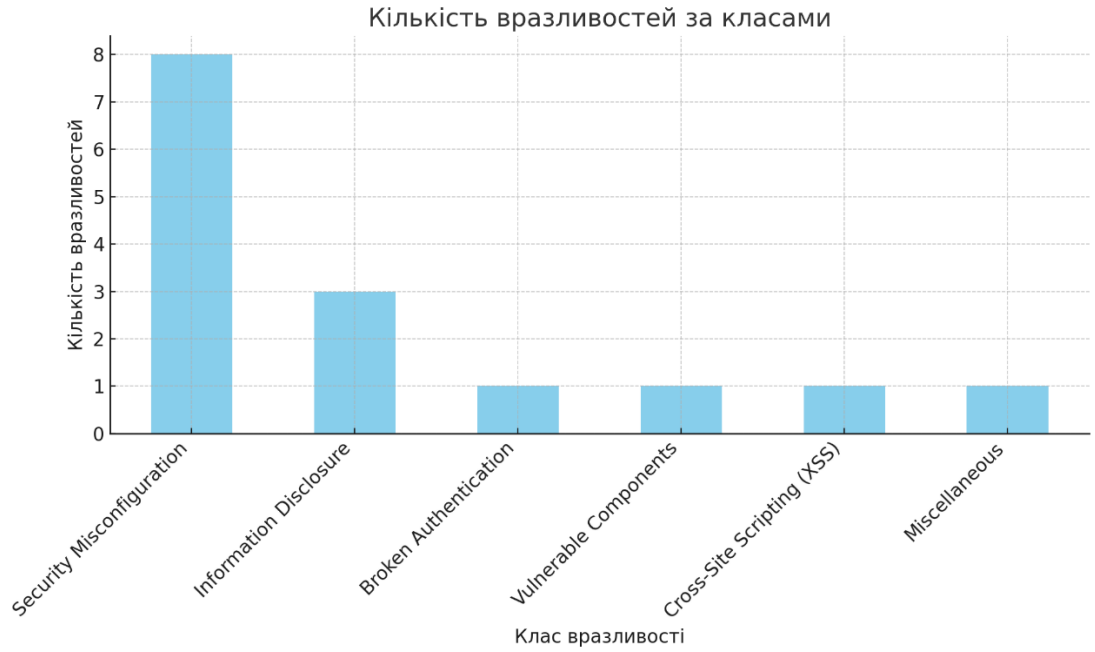


Рисунок 3.9 – Діграма кількості вразливостей за класом

Таким чином відповідно до раніше згаданого числа у загальному 107 з унікальних вразливостей у веб-застосунку, після динамічного сканування було віднайдено 11 унікальних вразливостей, не рахуючи інформаційні примітки, що становить приблизно 10.28% від усіх, без урахування не відомих хибно-позитивних результатів, що приблизно відповідає ефективності OWASP ZAP 5.91%, раніше визначених використовуючи бенчмарк. Потрібно розуміти, що виявлені таким чином, попередньо вразливості у процесі розробки ПЗ дозволяє зменшити ймовірність великих збитків у майбутньому, зменшити кількість необхідної роботи для спеціаліста зі знаходження вразливостей, адже він зможе сконцетруватися на мануальному пошуку, найбільш важко видимих вразливостей, які потенційно можуть бути особливо небезпечними.

3.5 Висновки до розділу

У процесі роботи над розділом було виконано низку важливих завдань, спрямованих на створення та впровадження підсистеми динамічного тестування безпеки застосунків. Перш за все, було здійснено обґрунтування вибору

середовища для реалізації цієї підсистеми. Враховуючи всі вимоги та можливості, було вирішено використовувати AWS як основну платформу для розгортання та тестування.

Зокрема, реалізація підсистеми динамічного тестування безпеки була здійснена за допомогою принципу «інфраструктура як код» (Infrastructure as Code). Це дозволило автоматизувати процеси налаштування, розгортання та управління ресурсами, що значно підвищило ефективність та зменшило ймовірність людських помилок. Було створено модулі підсистеми, які включають всі необхідні компоненти для проведення динамічного тестування безпеки.

Для подальшого ефективного аналізу результатів тестування було обрано об'єкт тестування – веб-застосунок OWASP Juice Shop. Цей застосунок є ідеальним вибором, оскільки містить безліч відомих вразливостей, що дозволяє всебічно перевірити ефективність підсистеми тестування безпеки.

У ході роботи було продемонстровано роботу підсистеми тестування безпеки у розгорнутому середовищі AWS. Було налаштовано та автоматизовано всі етапи тестування, від розгортання веб-застосунку до проведення динамічного тестування за допомогою інструментів, таких як OWASP ZAP. Це дозволило отримати реальні дані про вразливості та перевірити працездатність підсистеми у бойових умовах.

Після проведення тестування було здійснено детальний аналіз отриманих результатів. На основі звітів OWASP ZAP було виявлено та класифіковано знайдені вразливості.

Таким чином, виконана робота показала практичність та ефективність створеної підсистеми динамічного тестування безпеки застосунків, забезпечила перевірку обраного об'єкту тестування та надала цінну інформацію для покращення безпеки в реальному середовищі.

ВИСНОВКИ

У ході роботи було виконано комплексний аналіз методів та інструментів динамічного тестування безпеки застосунків, реалізовано підсистему для динамічного тестування на базі AWS, та проведено її практичну апробацію. Розглянуто основні методи динамічного тестування, такі як фаззинг, аналіз на основі підписів, евристичний аналіз, поведінковий аналіз, тестування валідації вхідних даних та аналіз конфігурацій. Ці методи створюють комплексний підхід, що забезпечує виявлення і усунення широкого спектра вразливостей.

Було здійснено аналіз різних інструментів з відкритими кодом для динамічного тестування, включаючи OWASP ZAP, Wapiti та Arachni. Кожен з них має свої переваги та недоліки, але найвищі результати за швидкістю та ефективністю показав OWASP ZAP, що робить його оптимальним вибором для багатьох завдань.

Реалізація підсистеми динамічного тестування безпеки на базі AWS з використанням принципу «інфраструктура як код» дозволила автоматизувати процеси налаштування, розгортання та управління ресурсами. Було вибрано об'єкт тестування – веб-застосунок OWASP Juice Shop, який містить безліч відомих вразливостей і добре підходить для проведення аналізу ефективності роботи підсистеми.

Аналіз результатів тестування показав, що підсистема ефективно виявляє вразливості, що підтверджено детальними звітами OWASP ZAP. Ці результати підтвердили практичність та ефективність створеної підсистеми динамічного тестування безпеки застосунків.

Загалом, виконана робота підтвердила важливість використання сучасних технологій динамічного тестування для забезпечення високого рівня безпеки програмного забезпечення. Отримані результати та рекомендації сприятимуть підвищенню рівня безпеки та надійності веб-застосунків, зменшенню ризиків, пов'язаних з кіберзагрозами, та покращенню загальної стійкості програмного забезпечення до атак.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Желнитський Д. Ю., Лукічов В. В. Принцип роботи та методи інструментів динамічного тестування безпеки. Матеріали Всеукраїнської науково-практичної інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)», Вінниця, 11-20 травня 2024 р. 2024. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/21589
2. OWASP Foundation. Dynamic Application Security Testing (DAST). URL: https://owasp.org/www-project-devsecops-guideline/latest/02b-Dynamic-Application-Security-Testing (accessed: 06.05.2024).
3. OWASP Foundation. OWASP Top Ten. URL:https://owasp.org/www-project-developer-guide/draft/foundations/owasp_top_ten/ (accessed: 08.05.2024).
4. Trail of Bits. Fuzzing. URL: https://appsec.guide/docs/fuzzing/ (accessed: 10.05.2024).
5. Verma V. S. Implementing DevSecOps Practices. Birmingham-Mumbai : Packt, 2023. 258 p.
6. OWASP Foundation. Vulnerability Scanning Tools. URL: https://owasp.org/www-community/Vulnerability_Scanning_Tools (accessed: 12.05.2024).
7. The Crash Override Open Source Fellowship. Zed Attack Proxy (ZAP) - Getting Started. URL: https://www.zaproxy.org/getting-started/ (accessed: 14.05.2024).
8. PortSwigger. Burp Suite documentation. URL: https://portswigger.net/burp/documentation/contents (accessed: 16.05.2024).
9. Acunetix. Introduction to Acunetix. URL: https://www.acunetix.com/support/docs/introduction/ (accessed: 18.05.2024).
10. Chen S. Security Tools Benchmarking. 2018. URL: https://sectooladdict.blogspot.com/ (accessed: 20.05.2024).

11. Invicti. Advantages of Signature and Heuristic-Based Web Vulnerability Scanners. URL: https://www.invicti.com/blog/web-security/advantages-signature-heuristic-web-vulnerability-scanner/ (accessed: 22.05.2024).

12. Veracode. The State of Software Security. 2020. (accessed: 24.05.2024).

13. Mburano, B., & Si, W. Evaluation of web vulnerability scanners based on owasp benchmark. 2018 26th International Conference on Systems Engineering (ICSEng), 1– 6. https://doi.org/10.1109/ICSENG.2018.8638176 (accessed: 24.05.2024).

14. OWASP Benchmark. Benchmark for Java. URL: https://github.com/OWASP-Benchmark/BenchmarkJava (accessed: 26.05.2024).

15. OWASP Foundation. OWASP Benchmark Project. URL: https://owasp.org/www-project-benchmark/ (accessed: 28.05.2024).

16. Amazon Web Services. What is Cloud Computing? URL: https://aws.amazon.com/what-is-cloud-computing/ (accessed: 30.05.2024).

17. Amazon Web Services. Benefits of Application Hosting on AWS. URL: https://aws.amazon.com/application-hosting/benefits/ (accessed: 01.06.2024).

18. Amazon Web Services. What is IaC? URL: https://aws.amazon.com/what-is/iac/ (accessed: 03.06.2024).

19. Amazon Web Services. AWS CloudFormation Documentation. URL: https://docs.aws.amazon.com/cloudformation/ (accessed: 05.06.2024).

20. OWASP Foundation. OWASP Juice Shop Project. URL: https://owasp.org/www-project-juice-shop/ (accessed: 07.06.2024).

ДОДАТКИ

Додаток А

Текст коду конфігурації. Ініціалізація ресурсів для тестованого додатку.

```
AWSTemplateFormatVersion: 2010-09-09
Parameters:
  Environment:
    Type: String
    Description: S3Bucket Path with temp
  AmazonLinux2ImageID:
   Description: "Amazon Linux 2 Latest AMI ID"
   Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
Resources:
  myVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 10.10.0.0/16
      InstanceTenancy: default
      EnableDnsSupport: 'true'
      EnableDnsHostnames: 'true'
  subnet1:
    Type: 'AWS::EC2::Subnet'
    Properties:
      CidrBlock: 10.10.10.0/24
      VpcId: !Ref myVPC
      AvailabilityZone: !Select
        - 0
        - !GetAZs
          Ref: 'AWS::Region'
  subnet2:
    Type: 'AWS::EC2::Subnet'
    Properties:
      CidrBlock: 10.10.20.0/24
      VpcId: !Ref myVPC
      AvailabilityZone: !Select
```

```
      - 1
      - !GetAZs
        Ref: 'AWS::Region'
InternetGateway:
  Type: 'AWS::EC2::InternetGateway'
  DependsOn: myVPC
AttachGateway:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
    VpcId: !Ref myVPC
    InternetGatewayId: !Ref InternetGateway
PublicRouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref myVPC
    Tags:
      - Key: Name
        Value: Public
PublicRoute:
  Type: 'AWS::EC2::Route'
  DependsOn: AttachGateway
  Properties:
    RouteTableId: !Ref PublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId: !Ref InternetGateway
PublicSubnet1RouteTableAssociation:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref subnet1
    RouteTableId: !Ref PublicRouteTable
PublicSubnet2RouteTableAssociation:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref subnet2
    RouteTableId: !Ref PublicRouteTable
```

```
ServerGroup:
  Type: 'AWS::AutoScaling::AutoScalingGroup'
  CreationPolicy:
    ResourceSignal:
      Timeout: PT15M
      Count: '1'
  UpdatePolicy:
    AutoScalingRollingUpdate:
      MaxBatchSize: '1'
      MinInstancesInService: '1'
      PauseTime: PT15M
      WaitOnResourceSignals: 'true'
  Properties:
    VPCZoneIdentifier:
      - !Ref subnet1
      - !Ref subnet2
    LaunchConfigurationName: !Ref LaunchConfig
    MinSize: '1'
    MaxSize: '2'
    TargetGroupARNs:
      - !Ref ALBTargetGroup
    Tags:
      - Key: Name
        Value: !Sub "ServerGroup-${Environment}-EC2"
        PropagateAtLaunch: true
LaunchConfig:
  Type: 'AWS::AutoScaling::LaunchConfiguration'
  Properties:
    ImageId: !Ref 'AmazonLinux2ImageID'
    InstanceType: t2.micro
    AssociatePublicIpAddress: 'true'
    SecurityGroups:
      - !Ref InstanceSecurityGroup
    UserData:
      "Fn::Base64":
```

```
          !Sub |
            #!/bin/bash -xe
            yum update -y aws-cfn-bootstrap
            exec > >(tee /var/log/user-data.log|logger -t user-data -s
2>/dev/console) 2>&1
            amazon-linux-extras install java-openjdk11 -y
            amazon-linux-extras install tomcat8.5 -y
            wget https://github.com/bkimminich/juice-
shop/releases/download/v12.7.1/juice-shop-12.7.1.war -P
/usr/share/tomcat/webapps
            mv /usr/share/tomcat/webapps/juice-shop-12.7.1.war
/usr/share/tomcat/webapps/ROOT.war
            systemctl start tomcat
            systemctl enable tomcat
            /opt/aws/bin/cfn-signal -e $? --stack ${AWS::StackId} --
resource ServerGroup --region ${AWS::Region}
  ApplicationLoadBalancer:
    Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
    Properties:
      Name: !Sub WebELB-${Environment}
      Subnets:
        - !Ref subnet1
        - !Ref subnet2
      SecurityGroups:
        - !Ref ELBSecurityGroup
    DependsOn: AttachGateway
  ALBListener:
    Type: 'AWS::ElasticLoadBalancingV2::Listener'
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref ALBTargetGroup
      LoadBalancerArn: !Ref ApplicationLoadBalancer
      Port: '80'
      Protocol: HTTP
  ALBTargetGroup:
    Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
```

```
  Properties:
    HealthCheckIntervalSeconds: 30
    HealthCheckPath: /
    HealthCheckTimeoutSeconds: 5
    HealthyThresholdCount: 3
    Port: 8080
    Protocol: HTTP
    Matcher:
      HttpCode: 200
    UnhealthyThresholdCount: 5
    VpcId: !Ref myVPC

InstanceSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Enable port 8080 only from ELB SG.
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '8080'
        ToPort: '8080'
        SourceSecurityGroupId: !GetAtt ELBSecurityGroup.GroupId
    VpcId: !Ref myVPC
    Tags:
      - Key: Name
        Value: !Sub "InstanceSecurityGroup-${Environment}-SG"

ELBSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Enable HTTP access to the ELB
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: 0.0.0.0/0
```

```
      VpcId: !Ref myVPC
      Tags:
        - Key: Name
          Value: !Sub "ELBSecurityGroup-${Environment}-SG"

Outputs:
  URL:
    Description: URL of the Juice Shop application
    Value: !Join
      - ''
      - - 'http://'
        - !GetAtt
          - ApplicationLoadBalancer
          - DNSName
```

Додаток Б

Текст коду конфігурації. ініціалізація DAST сканування.

```
DASTScan:
    Type: AWS::CodeBuild::Project
    Properties:
      Artifacts:D
        Type: S3
        Location: !Ref Bucket
        Name: DASTCheck
        Packaging: ZIP
      BadgeEnabled: false
      Name: DASTCheck
      Description: Запуск DAST сканування
      Environment:
        ComputeType: BUILD_GENERAL1_SMALL
        Image: aws/codebuild/amazonlinux2-x86_64-standard:2.0
        Type: LINUX_CONTAINER
        EnvironmentVariables:
          - Name: APP_URL
            Value: http://example.com
      ServiceRole: !Ref CodebuildRole
      Source:
        Type: CODECOMMIT
        Location: !GetAtt CodeCommitRepository.CloneUrlHttp
        BuildSpec: DASTconfig.yaml
```

Додаток В

Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

Назва роботи: Система тестування безпеки програмного застосунку. Частина 2. Підсистема динамічного тестування

Автор роботи: Желнитський Дмитро Юрійович

Тип роботи: бакалаврська дипломна робота.

Підрозділ кафедра захисту інформації ФІТКІ. (кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність – 98,7%. Схожість – 1,3%.

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

☐ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку (підпис) Валентина КАПЛУН (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи (підпис) Дмитро ЖЕЛНИТСЬКИЙ (прізвище, ініціали)

Керівник роботи (підпис) Віталій ЛУКІЧОВ (прізвище, ініціали)

Додаток Г

ІЛЮСТРАТИВНА ЧАСТИНА

СИСТЕМА ТЕСТУВАННЯ БЕЗПЕКИ ПРОГРАМНОГО ЗАСТОСУНКУ.
ЧАСТИНА 2. ПІДСИСТЕМА ДИНАМІЧНОГО ТЕСТУВАННЯ

Виконав: студент 4 курсу групи 1БС-20 б
спеціальності 125 Кібербезпека
_____ Дмитро ЖЕЛНИТСЬКИЙ

Керівник: к. т. н., доцент, доцент каф. ЗІ
_____ Віталій ЛУКІЧОВ
«14» червня 2024 р.

ПОРІВНЯННЯ ІНСТРУМЕНТІВ DAST

Критерій	OWASP ZAP	Burp Suite	Acunetix
Точність виявлення	Відома своєю підтримкою з боку open-source спільноти. Хороша виявлення базових вразливостей, таких як XSS та SQL і. Залежить від конфігурації та плагінів.	Професійна версія високо оцінюється за точність і розширені функції, включаючи розширені методи сканування та інструменти ручного тестування.	Відзначається високою точністю виявлення та всеохоплюючою базою даних вразливостей.
Підтримка векторів введення	Підтримує різні вектори введення, включаючи GET та POST, з додатковими плагінами для JSON та XML.	Пропонує широку підтримку різних векторів введення. Потрібні розширення для повної підтримки деяких векторів, таких як AMF та GWT.	Забезпечує надійну підтримку різноманітних векторів введення, що робить його ефективним у різних сценаріях тестування.
Продуктивність та швидкість	Ефективний, але може бути повільнішим у порівнянні з комерційними інструментами, особливо при масштабних скануваннях.	Відомий ефективною продуктивністю, особливо у Pro-версії, яка включає функції для оптимізації швидкості сканування.	Загалом пропонує швидкі можливості сканування з оптимізаціями для швидкості та точності виявлення вразливостей.
Інтерфейс користувача та зручність використання	Інтерфейс може бути менш інтуїтивним для новачків, але пропонує широку документацію та підтримку спільноти.	Пропонує досконалий і зручний інтерфейс з обширною документацією та великою спільнотою користувачів.	Має зручний інтерфейс з інтуїтивною навігацією та комплексними ресурсами підтримки.
Розширені функції	Включає функції, такі як автоматизовані сканери, плагіни та підтримка скриптів для налаштування сканів.	Забезпечує розширені функції, такі як Intruder, Repeater та обширні розширення для користувацької функціональності.	Пропонує розширені можливості сканування, інтеграцію з іншими інструментами та детальні звіти.

КЛАСИФІКАЦІЯ МЕТОДІВ DAST

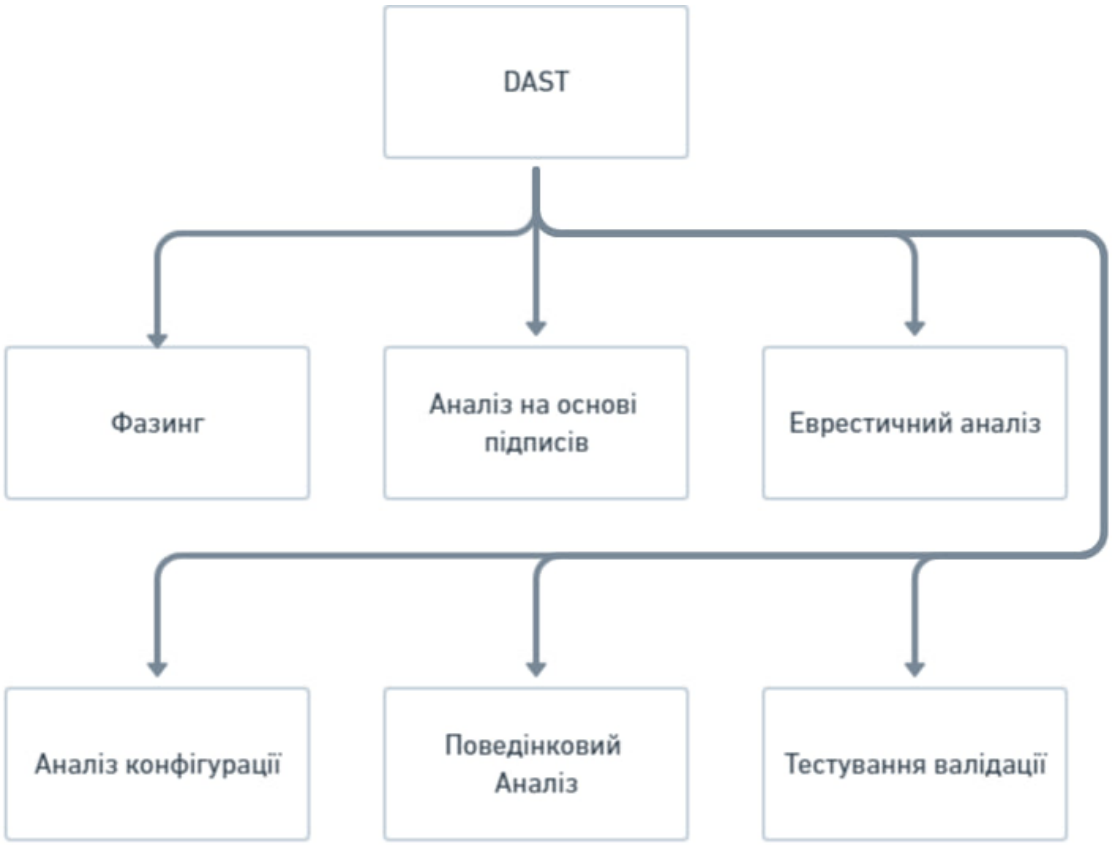
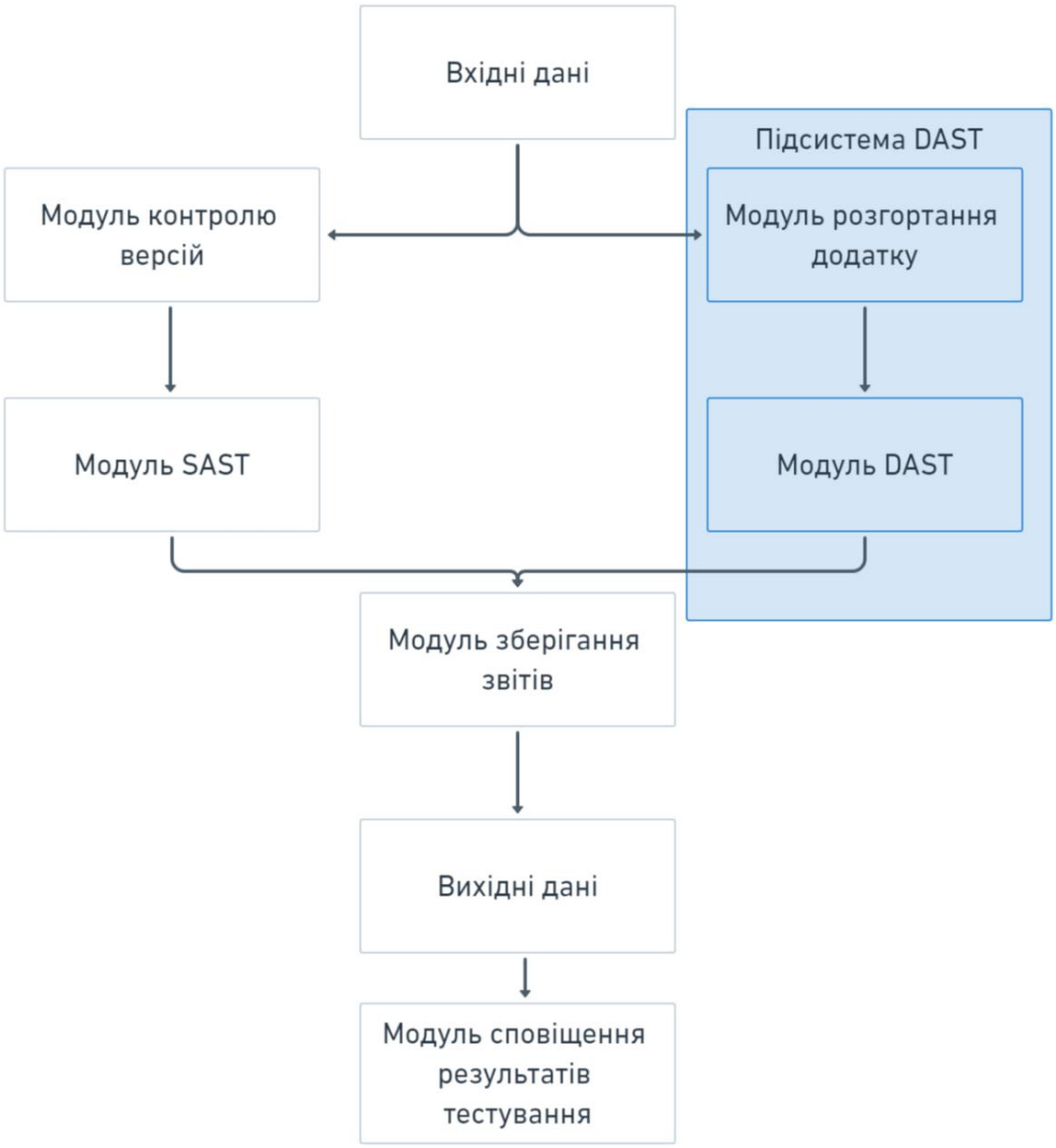


СХЕМА СИСТЕМИ ТЕСТУВАННЯ БЕЗПЕКИ



ПРОЦЕСИ ПІДСИСТЕМИ ДИНАМІЧНОГО ТЕСТУВАННЯ

СХЕМА ШАБЛОНУ CLOUDFORMATION

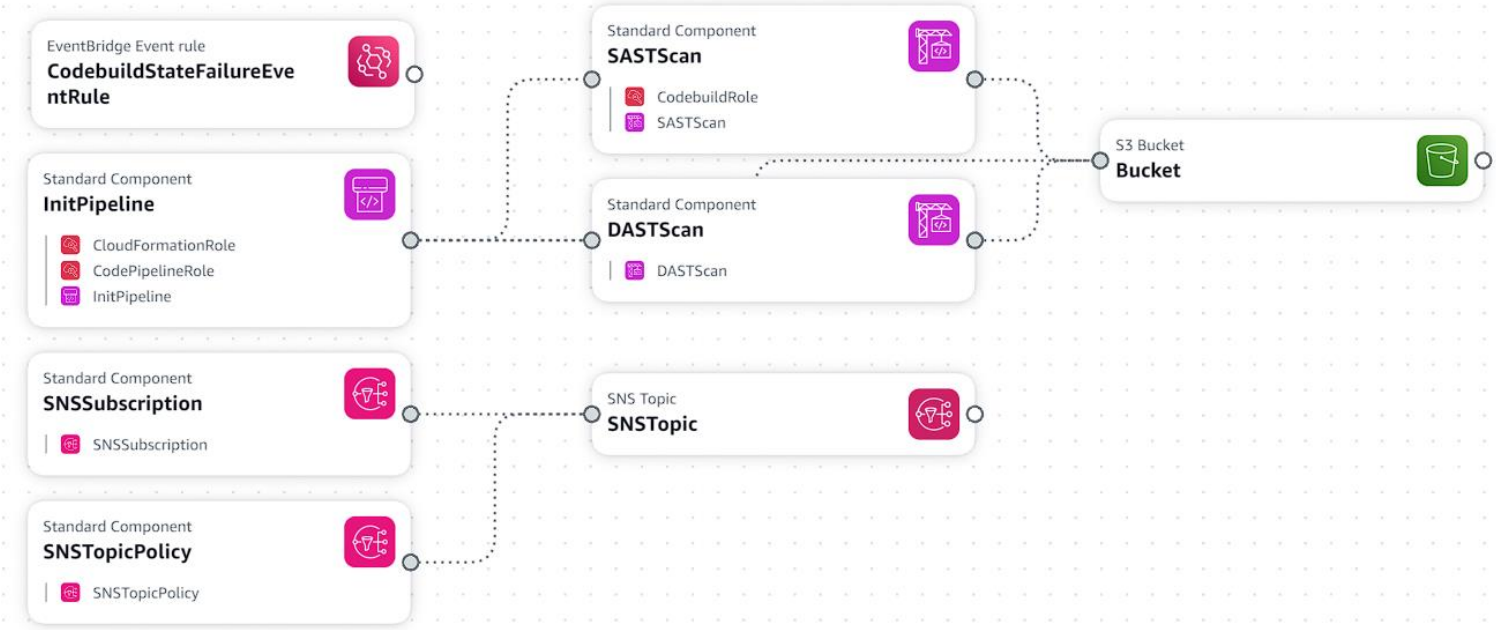
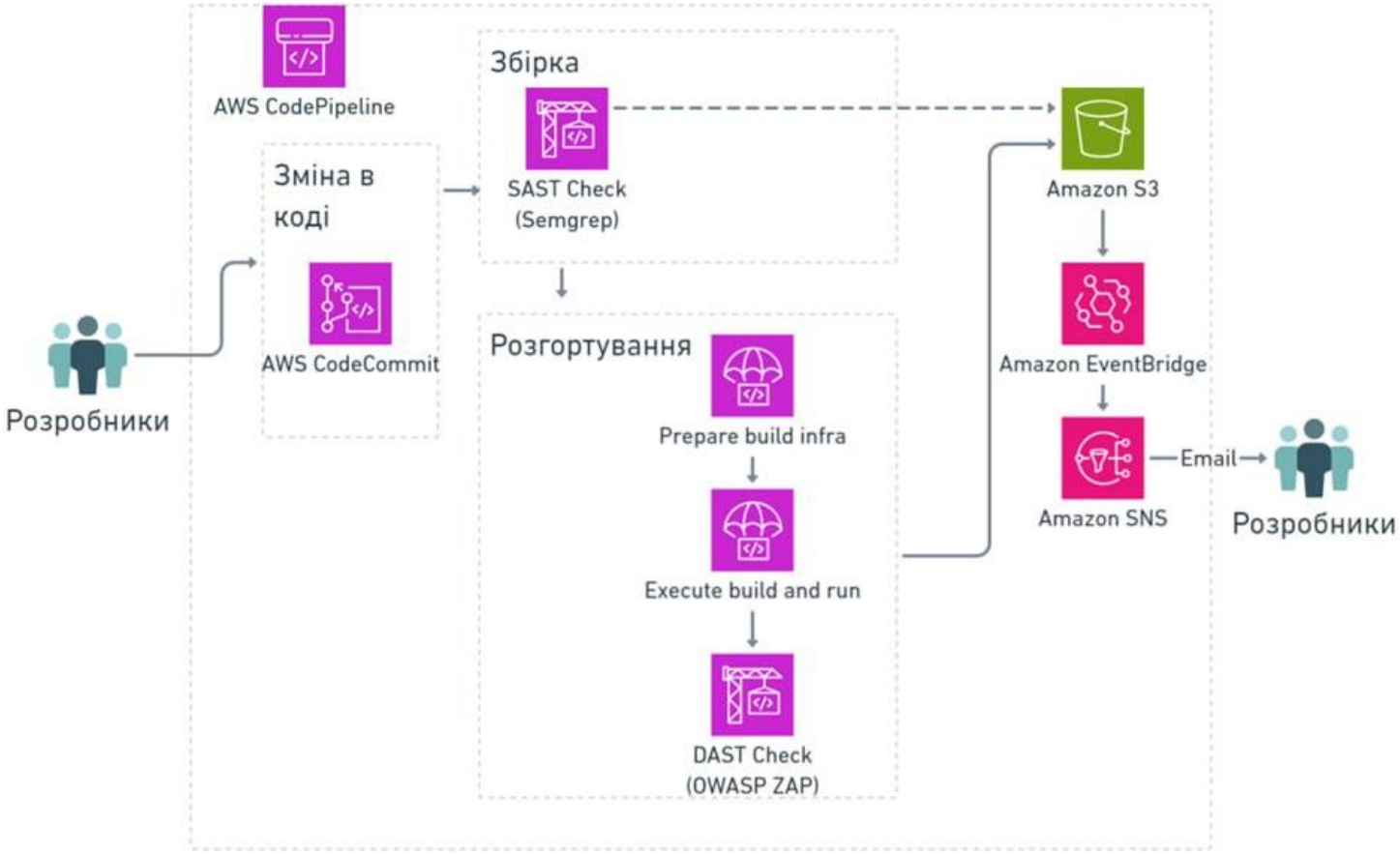


СХЕМА ІНФРАСТРУКТУРИ



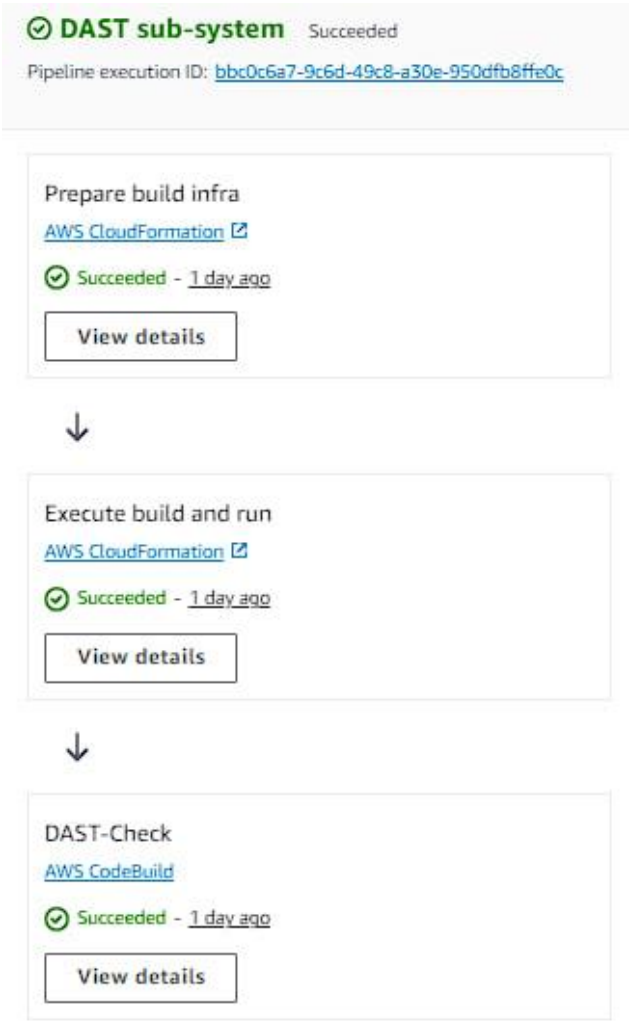
ІЛЮСТРАЦІЇ ПРОЦЕСУ ДИНАМІЧНОГО ТЕСТУВАННЯ

Resources (16)

Search resources

Logical ID	Type	Status
ALBListener	AWS::ElasticLoadBalancingV2::Listener	CREATE_COMPLETE
ALBTargetGroup	AWS::ElasticLoadBalancingV2::TargetGroup	CREATE_COMPLETE
ApplicationLoadBalancer	AWS::ElasticLoadBalancingV2::LoadBalancer	CREATE_COMPLETE
AttachGateway	AWS::EC2::VPCGatewayAttachment	CREATE_COMPLETE
ELBSecurityGroup	AWS::EC2::SecurityGroup	CREATE_COMPLETE
InstanceSecurityGroup	AWS::EC2::SecurityGroup	CREATE_COMPLETE
InternetGateway	AWS::EC2::InternetGateway	CREATE_COMPLETE
LaunchConfig	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE
PublicRoute	AWS::EC2::Route	CREATE_COMPLETE
PublicRouteTable	AWS::EC2::RouteTable	CREATE_COMPLETE
PublicSubnet1RouteTableAssociation	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
PublicSubnet2RouteTableAssociation	AWS::EC2::SubnetRouteTableAssociation	CREATE_COMPLETE
ServerGroup	AWS::AutoScaling::AutoScalingGroup	CREATE_COMPLETE
myVPC	AWS::EC2::VPC	CREATE_COMPLETE
subnet1	AWS::EC2::Subnet	CREATE_COMPLETE
subnet2	AWS::EC2::Subnet	CREATE_COMPLETE

Розгорнуті ресурси інфрастуктури



Успішно виконані етапи підсистеми

ЗВІТ ДИНАМІЧНОГО ТЕСТУВАННЯ БЕЗПЕКИ

ZAP Version: 2.15.0

ZAP is supported by the Crash Override Open Source Fellowship

Summary of Alerts

Risk Level	Number of Alerts
High	1
Medium	5
Low	5
Informational	4

Заголовок звіту

Alerts

Name	Risk Level
Cloud Metadata Potentially Exposed	High
Content Security Policy (CSP) Header Not Set	Medium
Cross-Domain Misconfiguration	Medium
Missing Anti-clickjacking Header	Medium
Session ID in URL Rewrite	Medium
Vulnerable JS Library	Medium
Cross-Domain JavaScript Source File Inclusion	Low
Private IP Disclosure	Low
Strict-Transport-Security Header Not Set	Low
Timestamp Disclosure - Unix	Low
X-Content-Type-Options Header Missing	Low
Information Disclosure - Suspicious Comments	Informational
Modern Web Application	Informational
Re-examine Cache-control Directives	Informational
Retrieved from Cache	Informational

Огляд усіх вразливостей

High	Cloud Metadata Potentially Exposed
Description	The Cloud Metadata Attack attempts to abuse a misconfigured NGINX server in order to access the instance metadata maintained by cloud service providers such as AWS, GCP and Azure. All of these providers provide metadata via an internal unroutable IP address '169.254.169.254' - this can be exposed by incorrectly configured NGINX servers and accessed by using this IP address in the Host header field.
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/latest/meta-data/
Method	GET
Parameter	
Attack	169.254.169.254
Evidence	
Other Info	Based on the successful response status code cloud metadata may have been returned in the response. Check the response data to see if any cloud metadata has been returned. The meta data returned can include information that would allow an attacker to completely compromise the system.
Instances	1
Solution	Do not trust any user data in NGINX configs. In this case it is probably the use of the $host variable which is set from the 'Host' header and can be controlled by an attacker.
Reference	https://www.nginx.com/blog/trust-no-one-perils-of-trusting-user-input/
CWE Id	
WASC Id	
Plugin Id	90034

Приклад вразливості високого рівня

Medium	Vulnerable JS Library
Description	The identified library jquery, version 2.2.4 is vulnerable.
URL	https://cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Method	GET
Parameter	
Attack	
Evidence	/2.2.4/jquery.min.js
Other Info	CVE-2020-11023 CVE-2020-11022 CVE-2015-9251 CVE-2019-11358
Instances	1
Solution	Please upgrade to the latest version of jquery.
Reference	https://github.com/jquery/jquery/issues/2432 http://blog.jquery.com/2016/01/08/jquery-2-2-and-1-12-released/ http://research.insecurelabs.org/jquery/test/ https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 https://github.com/advisories/GHSA-rmxg-73gg-4p98 https://nvd.nist.gov/vuln/detail/CVE-2015-9251 https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b https://github.com/jquery/jquery.com/issues/162 https://bugs.jquery.com/ticket/11974 https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/
CWE Id	829
WASC Id	
Plugin Id	10003

Приклад вразливості середнього рівня

Low	Cross-Domain JavaScript Source File Inclusion
Description	The page includes one or more script files from a third-party domain.
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
Other Info	
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
Other Info	
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/juice-shop/build/routes/assets/public/assets/public/favicon_js.ico
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
Other Info	
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/juice-shop/build/routes/assets/public/assets/public/favicon_js.ico
Method	GET
Parameter	//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js
Attack	
Evidence	<script src="//cdnjs.cloudflare.com/ajax/libs/jquery/2.2.4/jquery.min.js"></script>
Other Info	
URL	http://Web-2140011250.us-east-2.elb.amazonaws.com/juice-shop/build/routes/assets/public/favicon_js.ico

Приклад вразливості низького рівня