МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 Комп'ютерна і«нженерія»

Освітня програма: «Комп'ютерні системи та мережі»

Група: 4КС-56

Дипломний проект

здобувача освіти денної форми навчання
КС.56.ХХ.000.ДП

ДИМИТРЮКА
ГЛІБА ЮРІЙОВИЧА

м. Одеса
2023 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Комп'ютерна інженерія»**

Освітня програма: **«Комп'ютерні системи та мережі»**

Група: **4КС-56**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка системи безпеки додатків за допомогою цифрових методів контролю

Проектний матеріал складається з пояснювальної записки на 77 сторінках та графічного (презентаційного) матеріалу на 15 аркушах (слайдах).

Дипломник _____ (Димитрюк Г.Ю.)

Керівник _____ (Шевцов Ю.В.)

Консультанти

з економічної частини _____ (Копайгородська Т.Г.)

з охорони праці _____ (Чорновол Н.І.)

з дотримання вимог ЄСКД _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «21» червня 2023 р. Протокол ДКК № 3

Оцінка ДКК 5 (відмінно)

Секретар ДКК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

Спеціальність **123 «Комп'ютерна інженерія»**

Освітня програма **«Комп'ютерні системи та мережі»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР Беркань І.В.

"_____" _____ 2023 р.

ЗАВДАННЯ

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти Димитрюку Глібу Юрійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка системи безпеки додатків за допомогою цифрових методів контролю

затверджена наказом по коледжу від "17" жовтня 2022 р. № 235-А2-ОД

2. Термін здачі закінченого проекту (роботи) 16.06.2023р

3. Вихідні данні до проекту (роботи)

1. Ризики додатків, ризики безпеки додатків;
2. Підходи тестування додатків;
3. Перевірка додатку в безперервному тестуванні;
4. Інструменти тестування додатків;

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Вступ; Теоретичне проектування системи безпеки; Відкритий проект захисту додатків; Міжсайтинговий скритинг; Інструменти тестування безпеки додатків; Проектування системи безпеки додатків; Економічна частина; Охорона праці.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Приклад міжсайтового скриптінга (XSS)

Приклад підробки міжсайтових запитів (CSRF)

Схема CORS; Скіншот панелі приладів SonarQube

Скіншот панелі приладів New Relic;

Скіншот панелі приладів OWASP ZAP; Скіншот панелі приладів Fortify; Скріншот панелі приладів Snyk; Скріншот файла pre-commit.

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Технологічний	Шевцов Ю.В.		
Економічна частина	Копайгородська Т.Ю.		
Охорона праці	Чорновол Н.І.		
Нормоконтроль	Петрашова В.І.		
Старший консультант	Кривченко Ю.В.		

7. Дата видачі завдання 04.05.2023

Керівник Шевцов Ю.В. _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	Вступ. Постановка мети та задач проектування	5.05.2023	виконано
2	Комп'ютерні системи та їх взаємодії	7.05.2023	виконано
3	Аналіз найбільших ризиків для безпеки веб-додатків	9.05.2023	виконано
4	Міжсайтинговий скриптинг	11.05.2023	виконано
5	Підробка міжсайтингових запитів	13.05.2023	виконано
6	Спільне використання ресурсів з різних джерел	16.05.2023	виконано
7	Тестування програмного забезпечення	18.05.2023	виконано
8	Тестування безпеки додатків	20.05.2023	виконано
9	Інструмкнти тестування безпеки додатків	23.05.2023	виконано
10	Застосування цифрових методів контролю проектуємої програми	25.05.2023	виконано
11	Безперервне розгортання	27.05.2023	виконано
12	Динамічне тестування безпеки додатків	30.05.2023	виконано
13	Економічні розрахунки та питання з охорони праці	3.06.2023	виконано
14	Підготовка графічної частини проекту	6.06.2023	виконано
15	Підготовка проекту до захисту	8.06.2023	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КС 56. 07 000. 00 ДП	Дипломний проект		
А4			КС 56. 07 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КС 56. 07. 000. 00 ДП		
Розробив		Димитрюк Г.Ю.			Розробка безпеки додатків за допомогою цифрових методів контролю	Літ.	Аркуш 4 / Аркушів
Перевірив		Шевцов Ю..В				Н Д П	
Н. Контр.		Петрашова В.І..			ВСП "ОТФК ОНТУ" гр.КС-56		
Затверд.		Кривченко Ю.В.					

ЗМІСТ

ВСТУП

В сучасному світі, де все більше послуг і продуктів стають цифровими, забезпечення безпеки додатків стає критично важливим завданням. Існуючі методи захисту додатків можуть бути недостатніми або неефективними через постійно розвиваючіся загрози, в тому числі хакерські атаки, віруси та інші форми кіберзлочинності.

Тим не менш, багато сучасних систем безпеки додатків ще не впроваджують повноцінно цифрові методи контролю. Це може обмежувати їх здатність відстежувати, прогнозувати та запобігати потенційним загрозам, особливо в масштабних або складних додатках.

Таким чином, основною проблемою, яка буде вирішуватися в цій роботі, є розробка ефективної системи безпеки додатків, яка використовує цифрові методи контролю. Метою є покращення здатності до виявлення, прогнозування та реагування на загрози, тим самим зменшуючи ризик кібератак і збільшуючи надійність та безпеку додатків.

Система безпеки додатків повинна мати заходи для виявлення та захисту від вразливостей, атак та шкідливого коду. Це може включати в себе використання антивірусного програмного забезпечення, виявлення вторгнень, перевірку безпеки коду та інші техніки захисту.

Аудит та журналювання: Ведення аудиту та реєстрація активності в додатку допомагають виявити порушення безпеки, аналізувати події та відновлювати систему після інцидентів.

Одним з важливих аспектів системи безпеки додатків є навчання користувачів і персоналу щодо загроз безпеки, паролів, політик безпеки та найкращих практик. Свідомість користувачів є ключовим фактором у запобіганні соціальним інженерним атакам та іншим видам атак на безпеку додатків.

1. ТЕХНОЛОГІЧНИЙ РОЗДІЛ

1.1 Теоретичне проектування системи безпеки

Система безпеки додатків - це комплекс заходів, процедур, політик та технологій, що застосовуються для захисту додатків від потенційних загроз безпеці. Оскільки додатки використовуються для обробки, збереження та передачі чутливої інформації, важливо забезпечити їхню безпеку, щоб уникнути несанкціонованого доступу, втрати даних, порушень конфіденційності і цілісності.

Система безпеки додатків може включати такі складові:

Аутентифікація та авторизація: Для забезпечення безпеки додатків важливо перевіряти та підтверджувати ідентичність користувачів, що мають доступ до додатків, та надавати їм відповідні рівні дозволів і привілеїв.

Шифрування: Використання шифрування даних допомагає забезпечити конфіденційність інформації, що передається між додатком та іншими системами або користувачами. Шифрування може застосовуватись для захисту даних в спокої (зберігання) і під час передачі (протоколи забезпечення безпеки передачі даних).

Контроль доступу: Цей механізм визначає, хто має доступ до додатку і його функціональності. Це включає управління ролями, правилами доступу, аудитом та моніторингом активності користувачів.

Захист від зловмисних кодів: Система безпеки додатків повинна мати заходи для виявлення та захисту від вразливостей, атак та шкідливого коду. Це може включати в себе використання антивірусного програмного забезпечення, виявлення вторгнень, перевірку безпеки коду та інші техніки захисту.

Аудит та журналювання: Ведення аудиту та реєстрація активності в додатку допомагають виявити порушення безпеки, аналізувати події та відновлювати систему після інцидентів.

Оновлення та патчі: Система безпеки додатків включає процес

встановлення оновлень та патчів для виявлених вразливостей. Це допомагає забезпечити, що додатки залишаються захищеними від останніх загроз безпеки.

Навчання та свідомість: Одним з важливих аспектів системи безпеки додатків є навчання користувачів і персоналу щодо загроз безпеки, паролів, політик безпеки та найкращих практик. Свідомість користувачів є ключовим фактором у запобіганні соціальним інженерним атакам та іншим видам атак на безпеку додатків.

Всі ці складові спільно працюють, щоб забезпечити надійний рівень безпеки додатків та захистити їх від потенційних загроз та атак.

1.2 Опис компонентів системи, їх ролей і взаємодії

Розробка системи захисту додатків включає в себе впровадження різних заходів та практик, щоб забезпечити безпеку та захист додатка від несанкціонованого доступу, порушення даних та інших потенційних загроз. Ось загальна структура для розробки системи захисту додатків:

Моделювання загроз.

Почнемо з ідентифікації потенційних загроз та ризиків, з якими може зіткнутися наш додаток. Треба провесті ретельний аналіз архітектури, компонентів та взаємодій додатка, щоб виявити можливі вразливості.

Практики безпечного кодування.

Перевірка практики безпечного кодування, такі як перевірка введення, належна обробка помилок та безпечне зберігання даних. Дотримання стандартів і рекомендацій щодо безпеки кодування, таких як рекомендації OWASP (Open Web Application Security Project).

Аутентифікація та авторизація.

Впровадженя надійних механізмів аутентифікації, щоб перевірити ідентичність користувачів та забезпечити їм відповідні права доступу. Розглянути використання багатофакторної аутентифікації (MFA) для підвищення безпеки. Застосування принцип найменших привілеїв, надання

користувачам тільки необхідні дозволи для їх ролей.

Безпечна комунікація

Перевірка захісту даних під час передачі, використовуючи безпечні протоколи зв'язку, такі як HTTPS / TLS для веб-додатків. Впровадженя шифрування та перевірки цілісності, щоб забезпечити конфіденційність даних та запобігти їх зміни.

Валідація та очищення вводу.

Перевірка та очищеня всіх введень користувачів, щоб запобігти поширеним вразливостям, таким як SQL-ін'єкція, міжсайтовий скриптінг (XSS) та впровадження команд. Застосування строгіх правіл перевірки введення та використання параметризованих запитів або підготовлені оператори для зниження ризиків.

Обробка помилок та виключень.

Впровадження належних механізм обробки помилок та виключень, щоб запобігти розкриттю чутливої інформації зловмисникам. Уникання відображення деталей помилок користувачам в середовищі реальної експлуатації.

Управління сесіями.

Забезпеченя безпечного управління сесіями, впроваджуючи механізми для безпечного генерування та керування токенами сесій. Використання таймаутів сесій, перегенерація токенів сесій після аутентифікації та захищєння від викрадення сесій та атак фіксації сесій.

Безпечна конфігурація.

Налаштування додатоку безпечно, дотримуючись найкращих практик. Це включає безпечні конфігурації за замовчуванням, вимкнення непотрібних служб та функцій, а також регулярне оновлення та патчінг додатка та його залежностей.

Тестування безпеки. Регулярнє проводення тестування безпеки, включаючи пошук вразливостей, пенетраційне тестування та перевірка коду. Використання

автоматизованих інструментів та методик ручного тестування для виявлення та усунення вразливостей.

Журналювання та моніторинг.

Впровадження комплексних можливостєй журналювання та моніторингу для виявлення та реагування на інциденти безпеки. Слідкування за журналами додатка, мережевим трафіком та системними подіями для виявлення будь-якої підозрілої активності.

Регулярні оновлення та патчінг.

Контролюйтє останні патчі безпеки та оновлень для фреймворка, бібліотек та залежностей вашого додатка. Регулярно встановлюйте патчі для виправлення відомих вразливостей.

1.3 Відкритий проект захисту додаткив OWASP

Знання про OWASP Top 10 є важливим для розробки системи безпеки додатків з кількох причин. OWASP (Open Web Application Security Project) - це спільнота, що зосереджена на покращенні безпеки веб-додатків. OWASP Top 10 визначає десять найпоширеніших вразливостей веб-додатків, які можуть бути використані зловмисниками для атак.

Ось декілька причин, чому важливо знати про OWASP Top 10:

1.Стандарти безпеки: OWASP Top 10 є широко прийнятим стандартом в галузі безпеки веб-додатків. Знання про цей список допомагає розробникам та командам забезпечувати високий рівень безпеки, дотримуючись рекомендацій та найкращих практик.

2.Розпізнавання вразливостей: OWASP Top 10 надає вичерпний огляд найпоширеніших вразливостей, з якими можуть стикатися веб-додатки. Знання про ці вразливості допомагає розробникам визначати потенційні проблеми та недоліки в своєму коді та архітектурі додатків.

3.Зменшення ризиків: Ідентифікація і розуміння вразливостей з OWASP Top 10 дозволяє розробникам уникати спільних помилок та запобігати

використанню цих вразливостей зловмисниками. Це допомагає знизити ризики порушення безпеки та зберегти дані та функціональність додатка.

4.Освіта та навчання: OWASP Top 10 є важливим ресурсом для освіти та навчання розробників, тестувальників та інших зацікавлених сторін щодо безпеки веб-додатків. Цей список надає контекст і знання про актуальні загрози та рекомендації щодо їх усунення.

5.Вимоги замовника: Багато організацій та замовників ставлять вимоги до безпеки додатків, особливо якщо вони мають відношення до збереження конфіденційної інформації чи обробки особистих даних. Знання про OWASP Top 10 допомагає виконати такі вимоги та надати високий рівень безпеки для клієнтів.

Загалом, знання про OWASP Top 10 є важливою складовою розробки системи безпеки додатків, оскільки цей список надає цінну інформацію про поширені вразливості та рекомендації щодо їх усунення. Він допомагає забезпечити високий рівень безпеки додатків та зменшити ризики, пов'язані з атаками та витоками даних.

Топ-10 OWASP — це стандартний документ для розробників і безпеки веб-додатків. Він представляє широкий консенсус щодо найбільш критичних ризиків для безпеки веб-додатків.

У всьому світі визнано розробниками першим кроком до більш безпечного кодування.

Розробники повинні прийняти цей документ і почати процес забезпечення мінімізації цих ризиків у своїх веб-додатках.

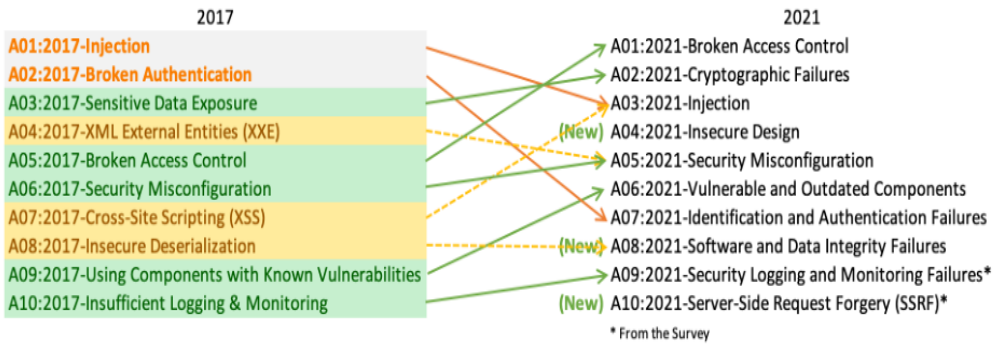

Рисунок 1.1. Топ-10 OWASP

1.4 Найбільші ризики для безпеки веб-додатків топ-10 OWASP

1.4.1 Порушений контроль доступу (A01:2021-Broken Access Control)

Контроль доступу забезпечує дотримання політики, щоб користувачі не могли діяти поза межами наданих їм дозволів. Збої зазвичай призводять до несанкціонованого розкриття інформації, модифікації чи знищення всіх даних або виконання бізнес-функцій за межами дозволених користувачеві повноважень. До поширених вразливостей контролю доступу належать. Порушення принципу найменших привілеїв або заборони за замовчуванням, коли доступ повинен надаватися лише для певних можливостей, ролей або користувачів, але доступний будь-кому. Обхід перевірок контролю доступу шляхом модифікації URL-адреси (підміна параметрів або примусовий перегляд), внутрішнього стану програми або HTML-сторінки, або за допомогою інструменту атаки, що модифікує запити API.

Дозвіл на перегляд або редагування чужого облікового запису шляхом надання його унікального ідентифікатора (небезпечні прямі посилання на об'єкти

Доступ до API з відсутніми елементами управління доступом для POST, PUT і DELETE.

Підвищення привілеїв. Дія від імені користувача без входу в систему або від імені адміністратора під час входу в систему як користувача. Маніпуляції з метаданими, такі як відтворення або підробка токену контролю доступу JSON Web Token (JWT), файлів cookie або прихованих полів для підвищення привілеїв або зловживання недійсністю JWT.

Неправильна конфігурація CORS дозволяє отримати доступ до API з несанкціонованих/ненадійних джерел. Примусовий перегляд автентифікованих сторінок від імені неавторизованого користувача або привілейованих сторінок від імені стандартного користувача. Як запобігти. Контроль доступу

ефективний лише в надійному серверному коді або безсерверному API, де зловмисник не може змінити перевірку контролю доступу або метадані. За винятком публічних ресурсів, забороняйте доступ за замовчуванням.

Реалізуємо механізми контролю доступу один раз і повторно будемо використовувати їх у всьому додатку, в тому числі мінімізуємо використання Cross-Origin Resource Sharing (CORS).

Модель управління доступом повинна забезпечувати право власності на записи, а не допускати, що користувач може створювати, читати, оновлювати або видаляти будь-який запис. Моделі доменів повинні забезпечувати дотримання вимог щодо обмежень для унікальних додатків.

Вимкнемо лістинг каталогів веб-серверів і переконаймося, що метадані файлів (наприклад, .git) та файли резервних копій не знаходяться в кореневих каталогах. Реєструємо збої контролю доступу, сповіщаємо адміністраторів, коли це доречно (наприклад, повторні збої). Обмежуємо доступ до API та контролерів, щоб мінімізувати шкоду від автоматизованого інструментарію атак.

Ідентифікатори сеансів зі станом повинні бути анульовані на сервері після виходу з системи. Токени JWT без статусу повинні бути скоріше короткоживучими, щоб звести до мінімуму вікно можливостей для зловмисника. Для довгоживучих JWT настійно рекомендується дотримуватися стандартів OAuth для відкликання доступу.

1.4.2 Криптографічні збої (A02:2021-Cryptographic Failures)

Перш за все, необхідно визначити потреби в захисті даних під час передачі та в стані спокою. Наприклад, паролі, номери кредитних карток, медичні записи, особиста інформація та комерційні таємниці потребують додаткового захисту, особливо якщо ці дані підпадають під дію законів про конфіденційність, наприклад, Загального регламенту ЄС про захист даних (GDPR), або нормативних актів, наприклад, про захист фінансових даних, таких

як Стандарт безпеки даних PCI DSS. Для всіх таких даних:

Чи передаються дані у вигляді відкритого тексту? Це стосується таких протоколів, як HTTP, SMTP, FTP, а також з використанням оновлень TLS, таких як STARTTLS. Зовнішній інтернет-трафік є небезпечним. Перевіряємо весь внутрішній трафік, наприклад, між балансувальниками навантаження, веб-серверами або внутрішніми системами. А саме потрібно перевірити:

Чи використовуються старі або слабкі криптографічні алгоритми або протоколи за замовчуванням або в старому коді? Чи використовуються стандартні криптографічні ключі, чи генеруються або повторно використовуються слабкі криптографічні ключі, чи відсутнє належне управління ключами або їх ротація? Чи перевіряються криптографічні ключі у сховищах вихідного коду? Чи не забезпечується шифрування, наприклад, чи відсутні будь-які директиви або заголовки безпеки HTTP-заголовків (браузера)? Чи належним чином перевірено отриманий сертифікат сервера та ланцюжок довіри? Чи вектори ініціалізації ігноруються, повторно використовуються або генеруються недостатньо безпечно для криптографічного режиму роботи? Чи використовується незахищений режим роботи, такий як ECB? Чи використовується шифрування там, де більш доречним є шифрування з автентифікацією? Чи використовуються паролі як криптографічні ключі за відсутності функції отримання ключа на основі пароля? Чи використовується випадковість для криптографічних цілей, яка не була розроблена для задоволення криптографічних вимог?

Навіть якщо вибрано правильну функцію, чи потрібно її засіяти розробником, і якщо ні, то чи не переписав розробник вбудовану в неї потужну функцію засіву на засів, який не має достатньої ентропії/непередбачуваності? Чи використовуються застарілі хеш-функції, такі як MD5 або SHA1, або чи використовуються некриптографічні хеш-функції, коли потрібні криптографічні хеш-функції? Чи використовуються застарілі методи криптографічного заповнення, такі як PKCS number 1 v1.5? Чи можна використовувати криптографічні повідомлення про помилки або інформацію

побічних каналів, наприклад, у вигляді атак на оракул?

Як запобігти поперше необхідно класифікувати дані, що обробляються, зберігаються або передаються додатком. Визначте, які дані є конфіденційними відповідно до законів про конфіденційність, регуляторних вимог або бізнес-потреб.

Не зберігати конфіденційні дані без потреби. Видаляти їх якомога швидше або Використовуємо токенізацію чи навіть усічення, сумісні з PCI DSS. Дані, які не зберігаються, не можуть бути вкрадені. Переконуємося, що ви шифруєте всі конфіденційні дані у стані спокою. Використовуємо сучасні та надійні стандартні алгоритми, протоколи та ключі; Використовуємо належне управління ключами. Шифруйте всі дані під час передачі за допомогою безпечних протоколів, таких як TLS з прямим шифруванням (FS), визначенням пріоритетів шифрування на сервері та безпечними параметрами. Застосовуйте шифрування за допомогою таких директив, як HTTP Strict Transport Security (HSTS). Вимкніть кешування для відповідей, що містять конфіденційні дані. Застосовуйте необхідні засоби контролю безпеки відповідно до класифікації даних.

Не потрібно використовувати застарілі протоколи, такі як FTP і SMTP, для передачі конфіденційних даних. Зберігайте паролі за допомогою надійних адаптивних та солоних функцій хешування з робочим фактором (фактором затримки), таких як Argon2, scrypt, bcrypt або PBKDF2. Вектори ініціалізації повинні бути обрані відповідно до режиму роботи. Для багатьох режимів це означає використання CSPRNG (криптографічно захищеного генератора псевдовипадкових чисел).

Для режимів, які вимагають nonce, вектор ініціалізації (IV) не потребує ГПСЧ. У всіх випадках, IV ніколи не повинен використовуватися двічі для фіксованого ключа. Завжди використовуємо автентифіковане шифрування замість простого шифрування.

Ключі повинні генеруватися криптографічно випадковим чином і зберігатися в пам'яті у вигляді байтових масивів. Якщо використовується

пароль, він повинен бути перетворений на ключ за допомогою відповідної функції отримання базового ключа на основі пароля. Переконуємося, що криптографічна випадковість використовується там, де це доречно, і що вона не була засіяна передбачуваним чином або з низькою ентропією.

Більшість сучасних API не вимагають від розробника використовувати ГПСВЧ для забезпечення безпеки. Необхідно уникати застарілих криптографічних функцій і схем підстановки, таких як MD5, SHA1, PKCS number 1 v1.5. Самостійно перевіряти ефективність конфігурації та налаштувань.

1.4.3 Ін'єкція (A03:2021 – Injection

Додаток вразливий до атак, коли: Дані, що надаються користувачем, не перевіряються, не фільтруються та не очищуються програмою. Динамічні запити або непараметризовані виклики без контекстно-залежного екранування використовуються безпосередньо в інтерпретаторі.

Ворожі дані використовуються у параметров пошуку об'єктно-реляційного відображення (ORM) для вилучення додаткових, конфіденційних записів.

Шкідливі дані використовуються безпосередньо або конкатенуються. SQL або команда містить структуру і шкідливі дані в динамічних запитах, командах або збережених процедурах.

Деякі з найпоширеніших ін'єкцій - це SQL, NoSQL, команди ОС, об'єктно-реляційне відображення (ORM), LDAP, а також ін'єкції з використанням мови виразів (EL) або об'єктно-графічної навігаційної бібліотеки (OGNL). Концепція ідентична для всіх інтерпретаторів. Аналіз вихідного коду є найкращим методом виявлення вразливості додатків до ін'єкцій. Наполегливо рекомендується автоматизоване тестування всіх параметрів, заголовків, URL, файлів cookie, вхідних даних JSON, SOAP і XML. Організації можуть включити статичні (SAST), динамічні (DAST) та інтерактивні (IAST) інструменти тестування безпеки додатків в конвеєр CI/CD, щоб виявити вразливості до

ін'єкцій перед розгортанням у виробництво.

Як можна це запобігти. Запобігання ін'єкціям вимагає відокремлення даних від команд і запитів:

Кращим варіантом є використання безпечного API, який повністю уникає використання інтерпретатора, надає параметризований інтерфейс або мігрує до інструментів об'єктно-реляційного відображення (Object Relational Mapping Tools, ORM).

Примітка: Навіть параметризовані збережені процедури можуть спричинити SQL-ін'єкцію, якщо PL/SQL або T-SQL об'єднує запити і дані або виконує шкідливі дані за допомогою EXECUTE IMMEDIATE або exec().

Використовуємо позитивну перевірку вхідних даних на стороні сервера. Це не є повним захистом, оскільки багато додатків вимагають спеціальних символів, наприклад, текстові області або API для мобільних додатків.

Для будь-яких залишкових динамічних запитів екрануйте спеціальні символи, використовуючи спеціальний синтаксис екранування для цього інтерпретатора.

Примітка: Структури SQL, такі як назви таблиць, стовпців тощо, не можна екранувати, тому назви структур, введені користувачем, є небезпечними. Це поширена проблема у програмах для написання звітів.

Використовуємо LIMIT та інші елементи керування SQL у запитах, щоб запобігти масовому розкриттю записів у випадку SQL-ін'єкції.

1.4.4 Небезпечний дизайн (A04:2021-Insecure Design

Небезпечний дизайн - це широка категорія, що представляє різні недоліки, виражені як "відсутність або неефективність дизайну контролю". Небезпечний дизайн не є джерелом для всіх інших категорій ризиків з Топ-10. Існує різниця між ненадійним дизайном і ненадійною реалізацією. Ми розрізняємо недоліки дизайну та дефекти реалізації не просто так, вони мають різні першопричини та способи усунення.

Безпечний дизайн все ще може мати дефекти реалізації, що призводять до вразливостей, які можуть бути використані. Небезпечний дизайн не може бути виправлений досконалою реалізацією, оскільки за визначенням, необхідні засоби контролю безпеки ніколи не були створені для захисту від конкретних атак. Одним з факторів, що сприяють небезпечному проектуванню, є відсутність профілювання бізнес-ризиків, притаманних програмному забезпеченню або системі, що розробляється, і, таким чином, нездатність визначити, який рівень безпеки необхідний для проектування. Вимоги та управління ресурсами:

1. Збираємо і обговорюємо з бізнесом бізнес-вимоги до додатку, включаючи вимоги до захисту щодо конфіденційності, цілісності, доступності та автентичності всіх активів даних, а також очікувану бізнес-логіку. Візьмемо до уваги, наскільки вразливим буде ваш додаток і чи потрібен вам поділ орендарів (на додаток до контролю доступу).

2. Складємо технічні вимоги, включаючи функціональні та нефункціональні вимоги до безпеки.

3. Сплануємо та обговоримо бюджет, що охоплює всі етапи проектування, створення, тестування та експлуатації, включаючи заходи з безпеки.

4. Безпечний дизайн. Безпечне проектування - це культура і методологія, яка постійно оцінює загрози і гарантує, що код надійно спроектований і протестований для запобігання відомим методам атак.

Моделювання загроз має бути інтегроване в сеанси доопрацювання (або подібні заходи); зверніть увагу на зміни в потоках даних і контролі доступу або інших засобах контролю безпеки. При розробці історії користувача визначаємо правильні стани потоків і збоїв, переконаймося, що вони добре зрозумілі і узгоджені відповідальними сторонами і сторонами, на яких вони впливають.

Проаналізуємо припущення та умови для очікуваних потоків і станів відмов, Переконуємося, що вони все ще залишаються точними і бажаними. Визначте, як перевірити припущення і забезпечити виконання умов, необхідних для належної поведінки. Переконаймося, що результати задокументовані в

історії користувача.

Безпечний дизайн - це не надбудова чи інструмент, який можна додати до програмного забезпечення.

5. Безпечний життєвий цикл розробки.

Безпечне програмне забезпечення вимагає безпечного життєвого циклу розробки, певної форми безпечного шаблону проектування, чіткої методології, захищеної бібліотеки компонентів, інструментарію та моделювання загроз. Необхідно звернутися до фахівців з безпеки на початку програмного проекту, протягом усього проекту і підтримки програмного забезпечення. Розглянемо можливість використання Моделі зрілості забезпечення безпеки програмного забезпечення OWASP (SAMM), щоб допомогти структурувати ваші зусилля з розробки безпечного програмного забезпечення. Як запобігти Створіть і Використовуємо безпечний життєвий цикл розробки разом з професіоналами AppSec, щоб допомогти оцінити і спроектувати засоби контролю безпеки і конфіденційності. Використовуємо бібліотеку безпечних шаблонів проектування або готових до використання компонентів.

Використовуємо моделювання загроз для критично важливої автентифікації, контролю доступу, бізнес-логіки та потоків ключів Інтегрувати мову безпеки та засоби контролю в історії користувачів Інтегруйте перевірки правдоподібності на кожному рівні нашого додатку (від фронтенду до бекенду). Писати модульні та інтеграційні тести, щоб перевірити, що всі критичні потоки є стійкими до моделі загроз. Складіть сценарії використання та неправильного використання для кожного рівня вашого додатку. Розділіть рівні на системні та мережеві рівні в залежності від вразливості та потреб у захисті. Надійно розділяйте користувачів на всіх рівнях за допомогою дизайну.

1.4.5 Помилка конфігурації безпеки (A05:2021-Security Misconfiguration)

Додаток може бути вразливим: Відсутнє належне посилення безпеки в будь-якій частині стеку додатків або неправильно налаштовані дозволи на хмарні сервіси.

Увімкнені або встановлені непотрібні функції (наприклад, непотрібні порти, служби, сторінки, облікові записи або привілеї).

Облікові записи за замовчуванням та їхні паролі залишаються увімкненими та незмінними. Обробка помилок показує трасування стеку або інші надто інформативні повідомлення про помилки.

В оновлених системах найновіші функції безпеки вимкнені або налаштовані ненадійно.алаштування безпеки в серверах додатків, фреймворках додатків (наприклад, Struts, Spring, ASP.NET), бібліотеках, базах даних тощо не встановлені на безпечні значення. Сервер не надсилає заголовки або директиви безпеки, або вони не налаштовані на безпечні значення. Програмне забезпечення застаріле або вразливе (див. A06:2021 - Вразливі та застарілі компоненти). Без узгодженого, повторюваного процесу конфігурації безпеки додатків системи піддаються більшому ризику.

Для запобігання від цього необхідно впровадити безпечні процеси інсталяції, зокрема: Повторюваний процес загартовування дозволяє швидко і легко розгорнути інше середовище, яке належним чином заблоковано. Середовища розробки, контролю якості та виробництва повинні бути налаштовані однаково, з різними обліковими даними, що використовуються в кожному середовищі. Цей процес слід автоматизувати, щоб мінімізувати зусилля, необхідні для налаштування нового безпечного середовища. Мінімальна платформа без зайвих функцій, компонентів, документації та зразків. Видаляємо або не встановлюйте невикористовувані функції та фреймворки.

Переглядаємо та оновлюємо конфігурацій відповідно до всіх зауважень,

оновлень та виправлень безпеки в рамках процесу управління виправленнями (див. A06:2021 - Вразливі та застарілі компоненти). Переглянемо дозволи на хмарне сховище (наприклад, дозволи на S3 bucket). Сегментована архітектура додатків забезпечує ефективний і безпечний поділ між компонентами або орендарями за допомогою сегментації, контейнеризації або хмарних груп безпеки (ACL). Автоматизований процес перевірки ефективності конфігурацій і налаштувань у всіх середовищах.

1.4.6. Вразливі та застарілі компоненти (A06:2021-Vulnerable and Outdated Components)

Додаток вразливий якщо розробник не знаєте версій всіх використовуваних вами компонентів (як на стороні клієнта, так і на стороні сервера). Це стосується як компонентів, які ви використовуєте безпосередньо, так і вкладених залежностей. Програмне забезпечення вразливе, не підтримується або застаріле. Сюди входить операційна система, веб-сервер/сервер додатків, система управління базами даних (СУБД), додатки, API та всі компоненти, середовища виконання та бібліотеки. Не виконується регулярне сканування на наявність вразливостей і не підписуєтеся на бюлетені безпеки, пов'язані з компонентами, які ви використовуєте.

Розробник не виправляєте або своєчасно не оновлюєте базову платформу, фреймворки та залежності з урахуванням ризиків. Це часто трапляється в середовищах, де виправлення є щомісячним або щоквартальним завданням під контролем змін, що залишає організації відкритими для непотрібного впливу виправлених вразливостей на дні або місяці.

Розробники програмного забезпечення не перевіряють сумісність оновлених, модернізованих або виправлених бібліотек. Розробник не захищаєте конфігурації компонентів.

Має бути впроваджено процес керування виправленнями: Видалення невикористовуваних залежностей, непотрібних функцій, компонентів, файлів і документації.

Постійно інвентаризувати версії як клієнтських, так і серверних компонентів (наприклад, фреймворків, бібліотек) та їх залежностей за допомогою таких інструментів, як версії, OWASP Dependency Check, retire.js тощо. Постійно перевіряйте такі джерела, як Common Vulnerability and Exposures (CVE) та National Vulnerability Database (NVD) на наявність вразливостей у компонентах. Використовуємо інструменти аналізу складу програмного забезпечення для автоматизації цього процесу.

Отримання компонентів лише з офіційних джерел за захищеними посиланнями. Надавайте перевагу підписаним пакункам, щоб зменшити ймовірність включення модифікованого шкідливого компонента.

Відстеження бібліотек та компонентів, які не підтримуються або не створюють патчі безпеки для старих версій. Якщо встановлення патчів неможливе, розгляньте можливість розгортання віртуальних патчів для моніторингу, виявлення або захисту від виявленої проблеми.

Кожєн додаток повиннен бути забезпечиннен постійний план моніторингу, сортування та застосування оновлень або змін конфігурації протягом усього терміну служби програми або портфоліо.

1.4.7 Помилки ідентифікації та аутентифікації (A07:2021 – Identification and Authentication Failures)

Підтвердження особи користувача, автентифікація та керування сеансами є критично важливими для захисту від атак, пов'язаних з автентифікацією. У застосунку можуть бути вразливі місця в автентифікації:

- Дозволяє автоматизовані атаки, такі як підміна облікових даних, коли зловмисник має список дійсних імен користувачів і паролів.

- Дозволяє грубу силу або інші автоматизовані атаки.

- Дозволяє використовувати паролі за замовчуванням, слабкі або загальновідомі паролі, такі як "Password1" або "admin/admin".

- Використовує слабкі або неефективні процеси відновлення облікових даних і забутих паролів, такі як "відповіді на основі знань", які неможливо зробити безпечними.

- Використовує звичайний текст, зашифровані або слабо хешовані сховища даних паролів (див. A02:2021 - Криптографічні збої).

- Відсутня або неефективна багатофакторна автентифікація.

- Розкриває ідентифікатор сеансу в URL-адресі.

- Повторне використання ідентифікатора сеансу після успішного входу.

- Неправильно анулює ідентифікатори сеансів. Сеанси користувача або токени автентифікації (переважно токени єдиного входу (SSO)) не анулюються належним чином під час виходу з системи або періоду неактивності.

Кроки до запобігання цієй помилки:

- Впроваджуйте багатофакторну автентифікацію, щоб запобігти автоматизованому підбору облікових даних, атакам грубої сили та повторному використанню викрадених облікових даних.

- Не постачайте та не розгортайте систему з будь-якими обліковими даними за замовчуванням, особливо для користувачів з правами адміністратора.

- Впроваджуйте слабкі перевірки паролів, наприклад, перевіряйте нові або змінені паролі за допомогою списку 10 000 найгірших паролів.

- Узгодьте політику довжини, складності та ротації паролів з рекомендаціями Національного інституту стандартів і технологій (NIST) 800-63b (розділ 5.1.1 "Секрети, що запам'ятовуються") або іншими сучасними, науково обґрунтованими політиками використання паролів.

- Переконуємося, що реєстрація, відновлення облікових даних та шляхи доступу до API захищені від атак перебору облікових записів, використовуючи однакові повідомлення для всіх результатів.

- Обмежте або збільште час затримки невдалих спроб входу, але будьте обережні, щоб не створити сценарій відмови в обслуговуванні. Реєструйте всі збої та сповіщайте адміністраторів про виявлення підміни облікових даних, грубої сили або інших атак.

- Використовуємо безпечний вбудований менеджер сеансів на стороні сервера, який генерує новий випадковий ідентифікатор сеансу з високою ентропією після входу в систему. Ідентифікатор сеансу не повинен міститися в URL-адресі, повинен надійно зберігатися та анулюватися після виходу з системи, простою та абсолютних таймаутів.

1.4.8 Порушення цілісності програмного забезпечення та даних (A08:2021 – Software and Data Integrity Failures)

Порушення цілісності програмного забезпечення та даних пов'язані з кодом та інфраструктурою, які не захищають від порушень цілісності. Прикладом цього може бути ситуація, коли додаток покладається на плагіни, бібліотеки або модулі з ненадійних джерел, репозиторіїв та мереж доставки контенту (CDN). Незахищений конвеєр CI/CD може створити потенціал для несанкціонованого доступу, шкідливого коду або компрометації системи. Нарешті, багато додатків зараз включають функцію автоматичного оновлення, коли оновлення завантажуються без достатньої перевірки цілісності та застосовуються до раніше довірених додатків. Зловмисники можуть

завантажувати свої власні оновлення для розповсюдження та запуску на всіх інсталяціях.

Інший приклад - коли об'єкти або дані кодуються або серіалізуються в структуру, яку зловмисник може бачити і змінювати, і яка є вразливою до небезпечної десеріалізації.

Для запобігання цього необхідно зробити:

- Використовуємо цифрові підписи або подібні механізми для перевірки того, що програмне забезпечення або дані походять з очікуваного джерела і не були змінені.

- Переконуємося, що бібліотеки та залежності, такі як npm або Maven, використовують надійні сховища. Якщо ви маєте вищий профіль ризику, розгляньте можливість розміщення внутрішнього перевіреного репозиторію.

- Переконуємося, що для перевірки того, що компоненти не містять відомих вразливостей, використовується інструмент безпеки ланцюжка постачання програмного забезпечення, такий як OWASP Dependency Check або OWASP CycloneDX.

- Переконуємося, що існує процес перевірки змін коду та конфігурації, щоб мінімізувати ймовірність того, що шкідливий код або конфігурація можуть бути впроваджені у ваш програмний конвеєр.

- Переконуємося, що ваш конвеєр CI/CD має належне розділення, конфігурацію та контроль доступу, щоб забезпечити цілісність коду, який проходить через процеси збірки та розгортання.

- Переконуємося, що непідписані або незашифровані серіалізовані дані не надсилаються ненадійним клієнтам без певної форми перевірки цілісності або цифрового підпису для виявлення втручання або відтворення серіалізованих даних.

1.4.9 Журналювання та моніторинг збоїв безпеки (A09:2021 – Security Logging and Monitoring Failures)

Ця категорія допомагає виявляти, ескалувати та реагувати на активні порушення. Без реєстрації та моніторингу порушення неможливо виявити. Недостатня реєстрація, виявлення, моніторинг та активне реагування можуть виникнути в будь-який момент:

- Події, що підлягають аудиту, такі як входи, невдалі входи та транзакції з великою вартістю, не реєструються.
- Попередження та помилки не генерують жодних, неадекватних або незрозумілих повідомлень у журналі.
- Журнали програм та API не відстежуються на предмет підозрілої активності.
- Журнали зберігаються лише локально.
- Відповідні пороги сповіщення та процеси ескалації реагування відсутні або не є ефективними.

Тестування на проникнення та сканування за допомогою інструментів динамічного тестування безпеки додатків (DAST) (наприклад, OWASP ZAP) не викликають сповіщень.

Додаток не може виявити, ескалувати або попередити про активні атаки в режимі реального часу або в режимі, близькому до реального часу.

Додаток вразлив до витоку інформації, якщо робить події реєстрації та оповіщення видимими для користувача або зловмисника (див. A01:2021 - Порушення контролю доступу).

Щоб це запобігти розробники повинні впровадити деякі або всі наведені нижче засоби контролю, залежно від ризику додатку:

Переконуємося, що всі збої входу, контролю доступу та перевірки вхідних даних на стороні сервера можуть бути зареєстровані з достатнім контекстом користувача для виявлення підозрілих або зловмисних облікових записів і зберігатися протягом достатнього часу, щоб забезпечити можливість відкладеного криміналістичного аналізу.

Переконуємося, що журнали генеруються у форматі, який легко сприймається рішеннями для керування журналами.

Переконуємося, що дані журналів правильно закодовані, щоб запобігти ін'єкціям або атакам на системи реєстрації та моніторингу.

Переконуємося, що важливі транзакції мають аудиторський слід з контролем цілісності, щоб запобігти фальсифікації або видаленню, наприклад, таблиць баз даних, доступних лише для додавання, або подібних.

Команди DevSecOps повинні налагодити ефективний моніторинг та оповіщення, щоб швидко виявляти підозрілі дії та реагувати на них.

Створюємо або приймаємо план реагування на інциденти та відновлення, наприклад, Національний інститут стандартів і технологій (NIST) 800-61r2 або новішої версії.

Існують комерційні та відкриті системи захисту додатків, такі як OWASP ModSecurity Core Rule Set, а також програмне забезпечення з відкритим кодом для кореляції журналів, наприклад, стек Elasticsearch, Logstash, Kibana (ELK), які мають власні інформаційні панелі та сповіщення.

1.4.10 Підробка запитів на стороні серверу (A10:2021Server-Side Request Forgery)

Помилки SSRF виникають щоразу, коли веб-програма отримує віддалений ресурс без перевірки URL-адреси, наданої користувачем. Це дозволяє зловмиснику змусити додаток надіслати підроблений запит до неочікуваного місця призначення, навіть якщо він захищений брандмауером, VPN або іншим типом списків контролю доступу до мережі (ACL).

Оскільки сучасні веб-додатки надають кінцевим користувачам зручні функції, отримання URL-адреси стає поширеним сценарієм. Як наслідок, поширеність SSRF зростає. Крім того, серйозність SSRF стає вищою через хмарні сервіси та складність архітектури.

Розробники можуть запобігти SSRF, впровадивши деякі або всі наведені нижче засоби контролю глибинного захисту:

На мережевому рівні

- Сегментуємо функціонал віддаленого доступу до ресурсів в окремих мережах, щоб зменшити вплив SSRF

- Застосовуємо політики брандмауера "заборонити за замовчуванням" або правила контролю доступу до мережі, щоб заблокувати весь трафік інтрамережі, окрім необхідного.

- Встановіть право власності та життєвий цикл для правил брандмауера на основі додатків.

- Реєструйте всі прийняті та заблоковані мережеві потоки на брандмауерах (див. A09:2021 - Реєстрування та моніторинг збоїв у системі безпеки).

З прикладного рівня:

- Перевіряйте всі вхідні дані, що надаються клієнтом

- Застосуйте схему URL-адрес, порт і місце призначення за допомогою позитивного списку дозволів

- Не надсилайте необроблені відповіді клієнтам

- Вимкніть перенаправлення HTTP

- Слідкуйте за узгодженістю URL-адрес, щоб уникнути таких атак, як переприв'язка DNS і гонки "час перевірки, час використання" (TOCTOU)

Не зменшуйте вплив SSRF за допомогою списків заборон або регулярних виразів. Зловмисники мають списки корисного навантаження, інструменти та навички для обходу списків заборон.

Додаткові заходи, які слід врахувати:

Не розгортайте інші важливі для безпеки служби на підставних системах (наприклад, OpenID). Контролюйте локальний трафік на цих системах (наприклад, localhost)

Для інтерфейсів з виділеними та керованими групами користувачів використовуйте мережеве шифрування (наприклад, VPN) на незалежних системах, щоб врахувати дуже високі потреби у захисті.

1.5 Міжсайтовий скриптинг (XSS)

Міжсайтовий скриптинг (XSS) - це різновид ін'єкцій, коли шкідливі скрипти впроваджуються на безпечні та надійні веб-сайти. XSS-атаки відбуваються, коли зловмисник використовує веб-додаток для надсилання шкідливого коду, як правило, у вигляді сценарію браузера, іншому кінцевому користувачеві. Недоліки, які дозволяють цим атакам бути успішними, є досить поширеними і виникають скрізь, де веб-додаток використовує вхідні дані користувача у вихідних даних, які він генерує, без їх перевірки або кодування.

Зловмисник може використовувати XSS, щоб надіслати шкідливий скрипт користувачеві, який нічого не підозрює. Браузер кінцевого користувача не має можливості дізнатися, що скрипту не варто довіряти, і виконає його. Оскільки він вважає, що скрипт надійшов з надійного джерела, шкідливий скрипт може отримати доступ до будь-яких файлів cookie, токенів сеансу або іншої конфіденційної інформації, збереженої браузером і використовуваної на цьому сайті. Ці скрипти можуть навіть переписати вміст HTML-сторінки.

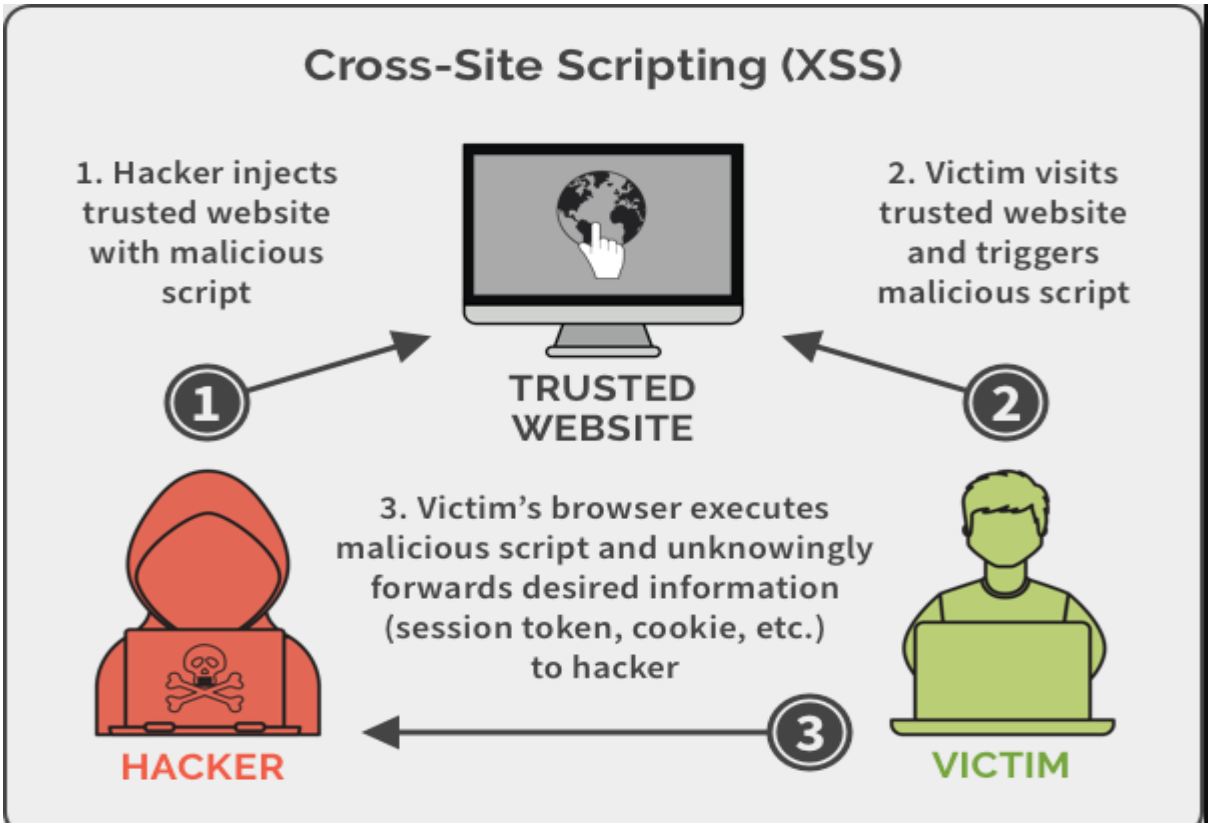

Рисунок 1.2. Приклад міжсайтового скриптінга (XSS)

Атаки на міжсайтовий скриптинг (XSS) відбуваються, коли:

- Дані потрапляють до веб-програми через ненадійне джерело, найчастіше через веб-запит.

- Дані включаються в динамічний вміст, який надсилається веб-користувачеві без перевірки на наявність шкідливого вмісту.

Шкідливий вміст, що надсилається веб-браузеру, часто має форму сегмента JavaScript, але може також включати HTML, Flash або будь-який інший тип коду, який може бути виконаний браузером. Різноманітність атак на основі XSS майже безмежна, але вони зазвичай включають передачу приватних даних, таких як файли cookie або іншу інформацію про сеанс, зловмиснику, перенаправлення жертви на веб-контент, контрольований зловмисником, або виконання інших шкідливих операцій на комп'ютері користувача під виглядом вразливого сайту.

Відображені та збережені XSS-атаки

XSS-атаки загалом можна розділити на дві категорії: відбиті та збережені. Існує третій, набагато менш відомий тип XSS-атак, який називається DOM Based XSS.

Відображені атаки - це атаки, в яких впроваджений скрипт відображається на веб-сервері, наприклад, у повідомленні про помилку, результатах пошуку або будь-якій іншій відповіді, яка включає в себе частину або весь вхідний код, відправлений на сервер як частина запиту. Відбиті атаки доставляються жертвам іншим шляхом, наприклад, в електронному повідомленні або на іншому веб-сайті. Коли користувача обманом змушують натиснути на шкідливе посилання, заповнити спеціально створену форму або навіть просто перейти на шкідливий сайт, впроваджений код потрапляє на вразливий веб-сайт, який відображає атаку назад у браузер користувача. Браузер виконує код, оскільки він надійшов з "довіреного" сервера. Відображену XSS також іноді називають непостійною або XSS типу I (атака здійснюється через один цикл запиту/відповіді).

Збережені атаки - це атаки, в яких впроваджений скрипт постійно зберігається на цільових серверах, наприклад, в базі даних, на форумі, в журналі відвідувачів, в полі для коментарів і т.д. Потім жертва отримує шкідливий скрипт з сервера, коли він запитує збережену інформацію. Збережені XSS також іноді називають персистентними або XSS типу II.

Сліпий міжсайтовий скриптинг - це різновид постійного XSS. Зазвичай він виникає, коли корисне навантаження зловмисника зберігається на сервері та відображається назад до жертви з внутрішньої програми. Наприклад, у формах зворотного зв'язку зловмисник може відправити шкідливий код за допомогою форми, і як тільки користувач/адміністратор додатку відкриє надіслану зловмисником форму через внутрішній додаток, код зловмисника буде виконано. Сліпий міжсайтовий скриптинг важко підтвердити в реальних умовах, але одним з найкращих інструментів для цього є XSS Hunter.

Наслідки XSS-атаки однакові, незалежно від того, чи є вона збереженою або відображеною (або заснованою на DOM). Різниця полягає в тому, як корисне навантаження потрапляє на сервер. Не варто думати, що сайт "тільки для читання" або "брошурне ПЗ" не вразливий до серйозних відображених XSS-атак. XSS може спричинити безліч проблем для кінцевого користувача, які варіюються за ступенем серйозності від роздратування до повної компрометації облікового запису. Найбільш серйозні XSS-атаки пов'язані з розкриттям сеансових файлів cookie користувача, що дозволяє зловмиснику перехопити сеанс користувача і заволодіти його обліковим записом. Інші шкідливі атаки включають розкриття файлів кінцевого користувача, встановлення троянських програм, перенаправлення користувача на іншу сторінку або сайт, або зміну представлення контенту. XSS-уразливість, що дозволяє зловмиснику змінити прес-реліз або новину, може вплинути на курс акцій компанії або знизити довіру споживачів. XSS уразливість на фармацевтичному сайті може дозволити зловмиснику змінити інформацію про дозування, що може призвести до передозування. Для отримання додаткової інформації про ці типи атак див. статтю Підміна контенту.

Як ми можемо визначити, чи додаток вразливий.

XSS помилки буває важко виявити та усунути з веб-додатку. Найкращий спосіб знайти уразливості - це виконати перевірку безпеки коду і знайти всі місця, де вхідні дані з HTTP-запиту можуть потрапити до HTML-виводу. Зверніть увагу, що для передачі шкідливого JavaScript можна використовувати безліч різних HTML-тегів. Nessus, Nikto та деякі інші доступні інструменти можуть допомогти просканувати веб-сайт на наявність цих недоліків, але вони можуть лише подряпати поверхню. Якщо одна частина веб-сайту є вразливою, існує висока ймовірність того, що існують й інші проблеми.

Для захищення себе від цього дуже важливо вимкнути підтримку HTTP TRACE на всіх веб-серверах. Зловмисник може викрасти дані cookie через Javascript, навіть якщо document.cookie відключений або не підтримується клієнтом. Ця атака проводиться, коли користувач публікує шкідливий скрипт на форумі, і коли інший користувач натискає на посилання, запускається асинхронний виклик HTTP Trace, який збирає інформацію про файли cookie користувача з сервера, а потім надсилає її на інший шкідливий сервер, який збирає інформацію про файли cookie, щоб зловмисник міг провести атаку перехоплення сеансу. Цьому можна легко запобігти, вимкнувши підтримку HTTP TRACE на всіх веб-серверах.

Кодування вихідних даних рекомендується, якщо вам потрібно безпечно відобразити дані саме в тому вигляді, в якому їх ввів користувач. Змінні не повинні інтерпретуватися як код замість тексту. У цьому розділі описано кожну форму кодування виводу, де її слід використовувати, а де слід уникати використання динамічних змінних взагалі.

Почніть з використання стандартного захисту кодування виводу вашого фреймворку, якщо ви бажаєте відображати дані у тому вигляді, в якому їх ввів користувач. Функції автоматичного кодування та екранування вбудовані у більшість фреймворків.

Якщо ви не використовуєте фреймворк або вам потрібно закрити прогалини у фреймворку, вам слід скористатися бібліотекою кодування виводу. Кожна

змінна, що використовується в інтерфейсі користувача, повинна передаватися через функцію кодування виводу. Список бібліотек кодування виводу наведено у додатку.

Існує багато різних методів кодування виводу, оскільки браузери по-різному розбирають HTML, JS, URL і CSS. Використання неправильного методу кодування може призвести до вразливостей або зашкодити функціональності вашої програми.

Проект OWASP ESAPI створив набір багаторазових компонентів безпеки кількома мовами, включаючи процедури перевірки та екранування для запобігання підміни параметрів та XSS-атак.

1.6 Підробка міжсайтових запитів (CSRF)

Підробка міжсайтових запитів (CSRF) - це атака, яка змушує кінцевого користувача виконати небажані дії у веб-додатку, в якому він наразі автентифікований. За допомогою невеликої допомоги соціальної інженерії (наприклад, надсилання посилання електронною поштою або в чаті) зловмисник може обманом змусити користувачів веб-додатку виконати дії на власний розсуд. Якщо жертвою є звичайний користувач, успішна CSRF-атака може змусити його виконати запити на зміну стану, наприклад, переказ коштів, зміну адреси електронної пошти тощо. Якщо жертвою є адміністративний обліковий запис, CSRF може скомпрометувати весь веб-додаток.

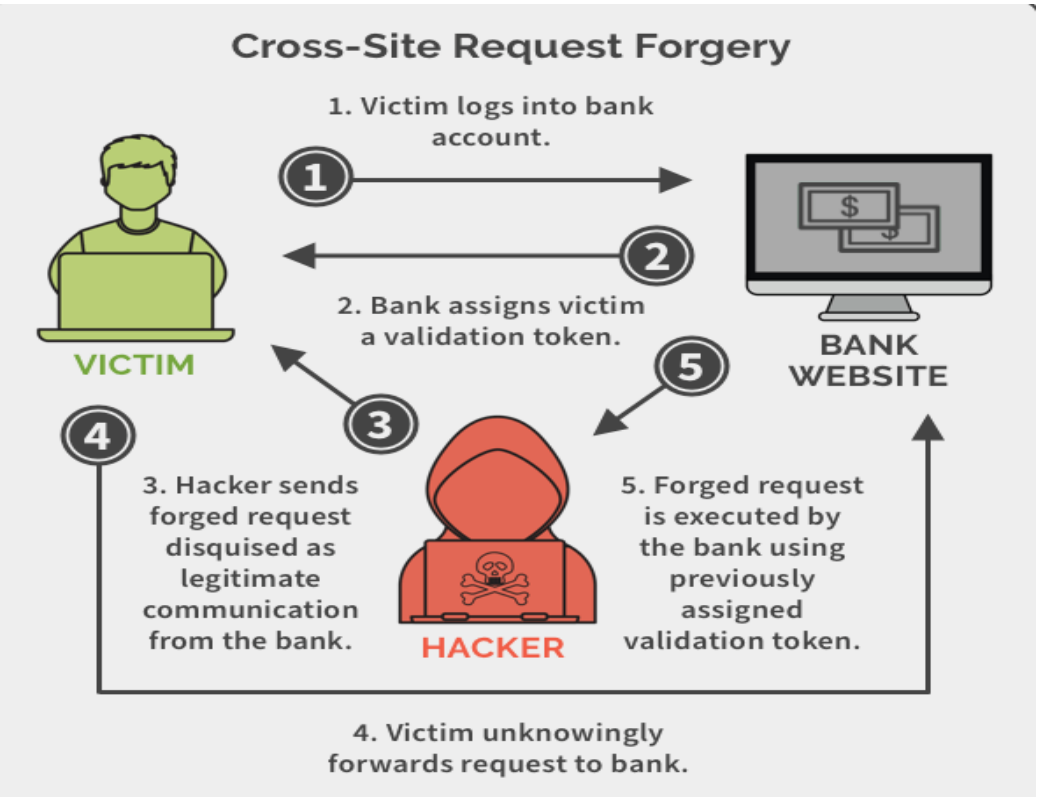

Рисунок 1.3. Приклад підробки міжсайтових запитів (CSRF)

CSRF - це атака, яка обманом змушує жертву надіслати шкідливий запит. Вона успадковує особистість і привілеї жертви, щоб виконати небажану функцію від її імені (хоча зауважте, що це не стосується CSRF для входу в систему, особливої форми атаки, описаної нижче). Для більшості сайтів запити браузера автоматично включають будь-які облікові дані, пов'язані з сайтом, такі як сесійний файл cookie користувача, IP-адреса, облікові дані домену Windows і так далі. Тому, якщо користувач в даний момент аутентифікований на сайті, сайт не зможе відрізнити підроблений запит, надісланий жертвою, від легітимного запиту, надісланого жертвою.

CSRF атакує цільову функціональність, яка викликає зміну стану на сервері, наприклад, зміну адреси електронної пошти або пароля жертви, або купівлю чогось. Змушуючи жертву отримати дані, зловмисник не отримує вигоди, оскільки відповідь отримує не зловмисник, а жертва. Таким чином, CSRF-атаки націлені на запити, що змінюють стан.

Зловмисник може використовувати CSRF для отримання приватних даних жертви за допомогою спеціальної форми атаки, відомої як login CSRF.

Зловмисник змушує неавторизованого користувача увійти в обліковий запис, який контролює зловмисник. Якщо жертва не усвідомлює цього, вона може додати до облікового запису особисті дані, наприклад, інформацію про кредитну картку. Потім зловмисник може знову увійти в обліковий запис, щоб переглянути ці дані, а також історію активності жертви у веб-додатку.

Іноді існує можливість зберігати CSRF-атаку на самому вразливому сайті. Такі уразливості називаються "збереженими CSRF-помилками". Це може бути досягнуто простим збереженням тегу IMG або IFRAME в полі, яке приймає HTML, або більш складною міжсайтовою скриптовою атакою. Якщо атака може зберігати CSRF-атаку на сайті, серйозність атаки посилюється. Зокрема, ймовірність збільшується через те, що жертва з більшою ймовірністю перегляне сторінку, яка містить атаку, ніж якусь випадкову сторінку в Інтернеті. Ймовірність також збільшується, оскільки жертва вже напевно пройшла автентифікацію на сайті.

Для захисту від CSRF слід дотримуватися наступних принципів:

- Перевірте, чи має ваш фреймворк вбудований захист від CSRF і використовуйте його

- Якщо фреймворк не має вбудованого захисту від CSRF, додайте токени CSRF до всіх запитів, що змінюють стан (запити, які викликають дії на сайті), і валідуйте їх на бекенді

- Для додатків з підтримкою станів використовуйте шаблон токенів синхронізатора

- Для додатків, керованих API, які не використовують теги <form>, розгляньте можливість використання кастомних заголовків запитів

- Використовуйте атрибут SameSite Cookie Attribute для сесійних файлів cookie, але будьте обережні, щоб НЕ встановлювати файл cookie спеціально для домену, оскільки це може призвести до вразливості безпеки, оскільки всі субдомени цього домену будуть використовувати цей файл cookie спільно. Це особливо актуально, коли субдомен має CNAME до доменів, які ви не контролюєте.

- Розглянемо можливість впровадження захисту на основі взаємодії з користувачем для високочутливих операцій

Подумаємо про перевірку походження за допомогою стандартних заголовків

Пам'ятаємо, що будь-який міжсайтовий скриптинг (XSS) може бути використаний для подолання всіх методів запобігання CSRF!

За потребую дивимося шпаргалку OWASP XSS Prevention Cheat Sheet для отримання детальної інформації про те, як запобігти XSS-атакам.

Не використовуйте GET-запити для операцій зміни стану.

1.7 Спільне використання ресурсів з різних джерел (CORS)

CORS - це механізм, який дозволяє багатьом ресурсам (наприклад, шрифтам, JavaScript тощо) на веб-сторінці бути запитаними з іншого домену, який не є доменом, з якого походить ресурс. CORS важливий, оскільки він надає безпечний спосіб дозволити одному сайту (сайту походження) викликати API на іншому сайту. Це стандарт, який всі сучасні веб-браузери виконують, щоб захистити конфіденційність користувачів та забезпечити їх дані. Без CORS веб-сайт може потенційно отримати доступ до конфіденційної інформації з іншого сайту, зробивши до нього запит. За допомогою CORS сервери можуть вказувати, хто може отримати доступ до їх активів, а браузер може потім запобігти будь-яким скриптам, які не походять від довіреного джерела, робити запит.

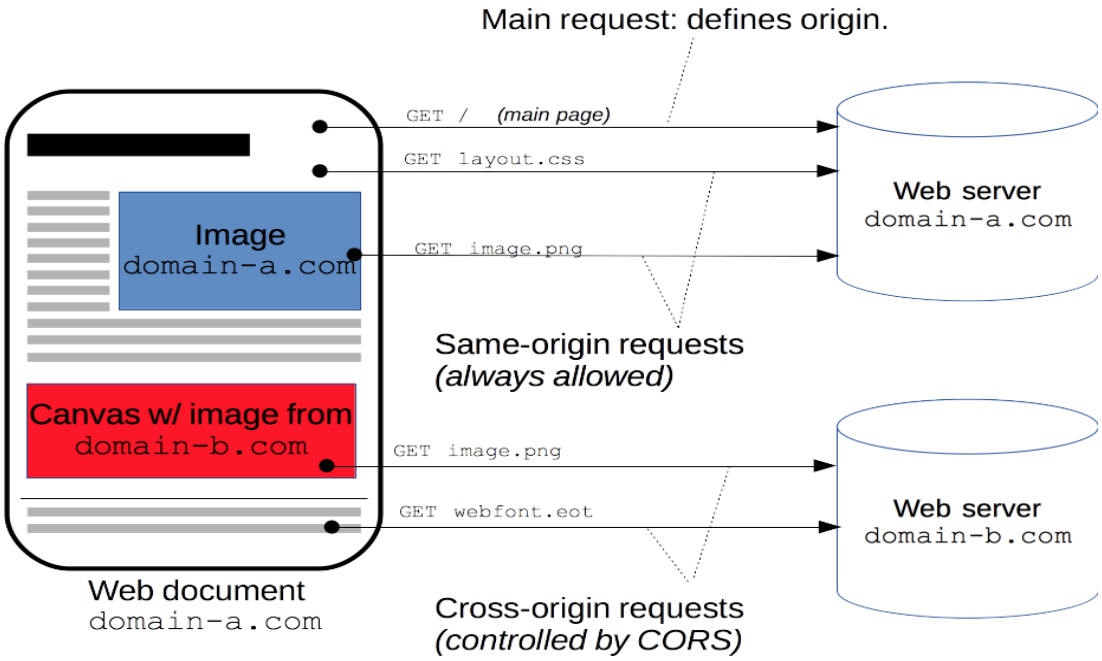

Рисунок 1.4. Схема CORS

Cross Origin Resource Sharing (CORS) - це механізм, який дозволяє веб-браузеру виконувати міждоменні запити за допомогою API XMLHttpRequest (XHR) Level 2 (L2) в контрольованому режимі. Раніше API XHR L1 дозволяв надсилати запити тільки в межах одного походження, оскільки це було обмежено політикою одного походження (SOP).

Запити перехресного походження мають заголовок Origin, який ідентифікує домен, що ініціює запит, і завжди надсилається на сервер. CORS визначає протокол, який використовується між веб-браузером і сервером, щоб визначити, чи дозволений перехресний запит. Для цього використовуються заголовки HTTP.

Специфікація W3C CORS вимагає, щоб для складних запитів, таких як запити, відмінні від GET або POST, або запити, що використовують облікові дані, заздалегідь надсилався попередній запит OPTIONS, щоб перевірити, чи не матиме тип запиту поганого впливу на дані. Попередній запит перевіряє методи і заголовки, дозволені сервером, і чи дозволені облікові дані. На основі результату запиту OPTIONS браузер вирішує, чи дозволений запит чи ні.

Походження та контроль доступу - Дозволити походження

Заголовок запиту Origin завжди надсилається браузером у CORS-запиті і вказує на походження запиту. Заголовок Origin не може бути змінений з JavaScript, оскільки браузер (користувацький агент) блокує його модифікацію; однак покладатися на цей заголовок для перевірки контролю доступу - не найкраща ідея, оскільки він може бути підроблений за межами браузера, наприклад, за допомогою проксі-сервера, тому вам все одно потрібно перевірити, чи використовуються протоколи на рівні додатків для захисту конфіденційних даних.

Access-Control-Allow-Origin - це заголовок відповіді, який використовується сервером, щоб вказати, яким доменам дозволено читати відповідь. Відповідно до специфікації CORS W3, клієнт сам визначає і застосовує обмеження на доступ до даних відповіді на основі цього заголовка.

Метод запиту контролю доступу та метод дозволу контролю доступу

Заголовок Access-Control-Request-Method використовується, коли браузер виконує попередній запит OPTIONS і дозволяє клієнту вказати метод остаточного запиту. З іншого боку, Access-Control-Allow-Method - це заголовок відповіді, який використовується сервером для опису методів, які дозволено використовувати клієнтам.

Заголовки *Access-Control-Request-Headers* та *Access-Control-Allow-Headers*

Ці два заголовки використовуються між браузером і сервером, щоб визначити, які заголовки можуть бути використані для виконання перехресного запиту.

Заголовок *Access-Control-Allow-Credentials* відповіді дозволяє браузерам читати відповідь, коли передаються облікові дані. Коли цей заголовок надсилається, веб-програма повинна встановити походження на значення заголовка Access-*Control-Allow-Origin*. Заголовок *Access-Control-Allow-Credentials* не можна використовувати разом із заголовком Access-Control-Allow-Origin, значенням якого є символ підстановки *, як показано нижче:

Access-Control-Allow-Origin: *

Access-Control-Allow-Credentials: true

XHR L2 впроваджує можливість створення міждоменних запитів за допомогою XHR API для забезпечення зворотної сумісності. Це може призвести до появи уразливостей в безпеці, яких не було в XHR L1. Цікавими місцями коду, які можуть бути використані, є URL-адреси, які передаються в XMLHttpRequest без перевірки, особливо якщо дозволені абсолютні URL-адреси, оскільки це може призвести до ін'єкції коду. Аналогічно, інша частина програми може бути використана, якщо дані у відповідь не екрануються, і ми можемо контролювати їх, надаючи користувачеві вхідні дані.

Інші заголовки:

Є й інші заголовки, такі як *Access-Control-Max-Age*, який визначає час, протягом якого передпольотний запит може кешуватися в браузері, або *Access-Control-Expose-Headers*, який вказує, які заголовки безпечно передавати до API специфікації CORS API.

Прості запити:

Деякі запити не викликають попереднього обльоту CORS. У застарілій специфікації CORS вони називаються простими запитами, хоча у специфікації Fetch (яка зараз визначає CORS) цей термін не використовується.

Мотивація полягає в тому, що елемент <form> з HTML 4.0 (який передує міжсайтовим XMLHttpRequest і fetch) може надсилати прості запити до будь-якого джерела, тому будь-хто, хто пише сервер, повинен вже передбачити захист від міжсайтового підроблення запитів (CSRF). Згідно з цим припущенням, серверу не потрібно погоджуватися (відповідаючи на попередній запит) на отримання будь-якого запиту, який виглядає як відправлення форми, оскільки загроза CSRF не є гіршою, ніж загроза відправлення форми. Однак, сервер все одно повинен погодитися за допомогою Access-Control-Allow-Origin, щоб поділитися відповіддю зі скриптом.

1.8 Тестування додатків

1.8.1 Тестування програмного забезпечення

Тестування програмного забезпечення - це критично важливий процес, який визначає, наскільки надійною, ефективною та безпечною є ваша програма або додаток. Існує кілька типів тестування програмного забезпечення:

Функціональне тестування: Це тестування спрямоване на перевірку функціональності програмного забезпечення, тобто чи працює воно відповідно до вимог специфікації.

Тестування продуктивності: Цей тип тестування вимірює продуктивність та відгук системи під різними обсягами та швидкостями роботи.

Тестування безпеки: Це тестування зосереджується на виявленні будь-яких потенційних вразливостей або дір у безпеці програмного забезпечення.

Тестування користувацького інтерфейсу (UI): Це тестування перевіряє, наскільки зручно та інтуїтивно зрозуміло користувачам взаємодіяти з програмним забезпеченням.

Тестування сумісності: Це тестування перевіряє, як програмне забезпечення взаємодіє з іншими системами, програмним забезпеченням або апаратним забезпеченням.

Тестування програмного забезпечення важливе через декілька причин:

Забезпечення якості продукту: Тестування допомагає забезпечити, що програмне забезпечення виконує свої функції належним чином.

Підвищення задоволеності користувача: Якісне програмне забезпечення, яке було належно протестовано, може підвищити задоволеність користувачів та залученість їх до продукту.

Виявлення та виправлення помилок: Тестування дозволяє виявити та виправити помилки перед тим, як продукт потрапить до кінцевих користувачів.

Запобігання втратам: Непротестоване програмне забезпечення може призвести до серйозних проблем, включаючи втрату даних, часу та, в кінцевому підсумку, коштів.

1.8.2 Тестування безпеки додатків

Тестування безпеки - це процес, спрямований на виявлення недоліків у механізмах безпеки інформаційної системи, які захищають дані та забезпечують функціонування відповідно до призначення. Це важливо, оскільки допомагає виявити потенційні вразливості в системі та гарантує, що дані в системі захищені від можливих атак. Ось деякі види тестування безпеки:

- Сканування на вразливість: це виконується за допомогою автоматизованого програмного забезпечення для сканування системи на відомі сигнатури вразливостей.
- Сканування безпеки: воно включає в себе виявлення слабких місць мережі та системи, а потім надає рішення для зменшення цих ризиків. Це сканування можна виконувати як вручну, так і автоматично.
- Тестування на проникнення: це тестування включає в себе виявлення вразливостей в конкретній системі та спроби їх використати для проникнення в систему.
- Оцінка ризику: це тестування включає аналіз спостережуваних ризиків безпеки в організації. Ризики класифікуються як низькі, середні та високі. Це тестування рекомендує контрольні заходи та заходи для зменшення ризику.
- Аудит безпеки: це внутрішній огляд додатків та операційних систем на наявність недоліків безпеки. Аудит також може бути проведений шляхом перевірки коду построчно.
- Етичний хакінг: це хакінг програмних систем організації. На відміну від шкідливих хакерів, які крадуть для власного збагачення, метою є виявлення недоліків безпеки в системі.
- Оцінка стану: це поєднує сканування безпеки, етичний хакінг та оцінку ризиків для показу загального стану безпеки організації.

Тестування безпеки - це найважливіше тестування для додатка і перевіряє, чи залишається конфіденційна інформація конфіденційною. У цьому типі тестування тестер виконує роль нападника і грає навколо системи, щоб знайти

помилки, пов'язані з безпекою. Тестування безпеки дуже важливе в інженерії програмного забезпечення для захисту даних всіма можливими способами.

1.9 Вибір інструменту тестування безпеки додатків

SonarQube - відкрита платформа, розроблена SonarSource для безперервного контролю якості коду. Він автоматично проводить огляди зі статичним аналізом коду для виявлення помилок, "запахів" коду та вразливостей безпеки. Він підтримує понад 20 мов програмування.

Ось деякі ключові особливості SonarQube:

1. Реальний зворотний зв'язок: SonarQube надає реальний зворотний зв'язок про якість коду, як тільки він написаний, що допомагає розробникам вчитися та вдосконалювати свої навички кодування.
2. Аналіз безпеки: SonarQube надає всеохоплюючий аналіз безпеки, включаючи виявлення вразливостей та гарячих точок. Він охоплює OWASP Top 10 та CWE Top 25, і використовує аналіз забруднення для відстеження ненадійного введення користувача через потік виконання.
3. Інтеграція з IDE: SonarQube може бути інтегрований у вашу IDE для надання зворотного зв'язку в реальному часі, що допомагає розробникам знаходити та виправляти проблеми, перш ніж вони будуть закомічені.
4. Quality Gate: SonarQube надає Quality Gate, який гарантує, що доставляється тільки чистий, протестований код.

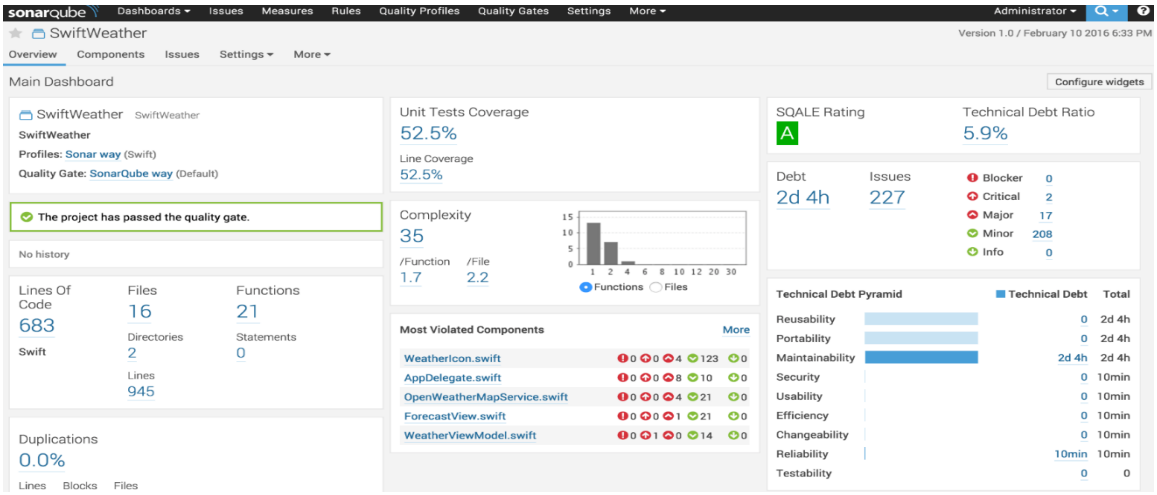

Рисунок 1.5. Скіншот панелі приладів SonarQube.

- Підтримка багатьох мов: SonarQube підтримує понад 30 популярних мов, що робить його універсальним інструментом для різноманітних команд розробників.

- Масштабування для підприємств: SonarQube може бути масштабований для відповіді на потреби великих підприємств, і він пропонує спеціалізовані звіти про безпеку для відстеження відповідності безпеки коду на рівні підприємства.

- **New Relic**- платформа спостереження, створена для того, щоб інженери створювали більш досконале програмне забезпечення. Він надає дані в реальному часі про продуктивність ваших веб-додатків та інфраструктури. Хоча він не спеціалізується на тестуванні безпеки, він може допомогти виявити проблеми з продуктивністю, які можуть вказувати на проблему безпеки.

- режимі реального часу за кілька хвилин, а не днів, та визначати пріоритети потенційних експлуатацій у допоміжних бібліотеках для більш безпечного життєвого циклу розробки.

- Безпека для всіх: New Relic має на меті поліпшити співпрацю команди навколо безпеки, об'єднуючи інформацію про безпеку для команд DevOps, SREs та InfoSec.

Робочі процеси безпеки: З New Relic ви можете синхронізувати команди розробки, операцій та безпеки, визначати пріоритети, відстежувати та звітувати про вразливості на рівні організації, команди або компонента

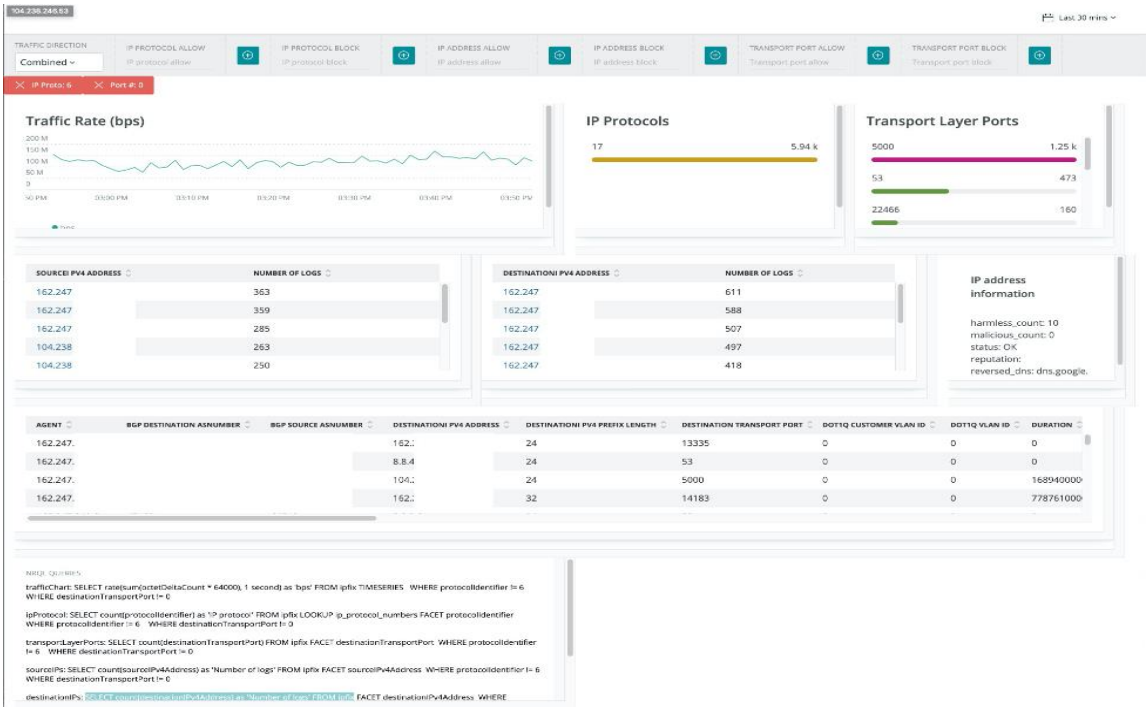

Рисунок 1.6. Скіншот панелі приладів New Relic.

Fortify - це інструмент, розроблений Micro Focus, який використовується для статичного тестування безпеки додатків

Він пропонує комплексні рішення з безпеки додатків з можливістю тестування на місці та за запитом для покриття всього життєвого циклу розробки програмного забезпечення. Fortify надає статичний аналізатор коду, динамічний аналіз та управління безпекою програмного забезпечення. Він призначений для того, щоб допомогти швидко створювати безпечне програмне забезпечення, виявляючи проблеми безпеки на ранніх стадіях розробки з високою точністю.

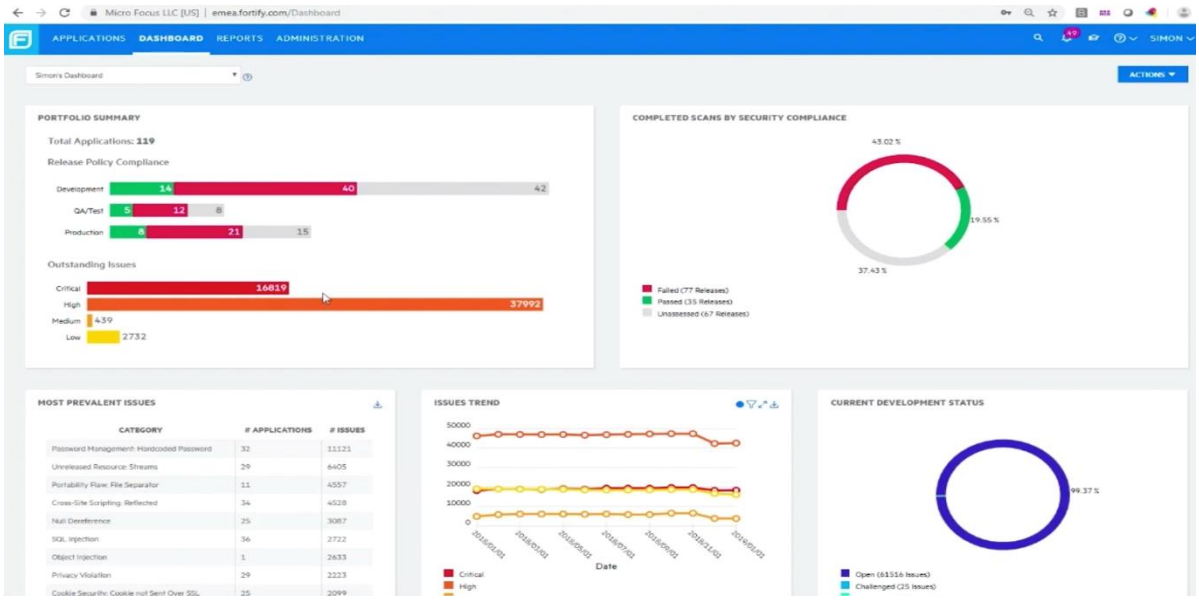

Рисунок 1.7 Скіншот панелі приладів Fortify.

Ключові особливості Fortify:

- Провідний в галузі SAST: Fortify забезпечує швидкий, безтривожний статичний й аналіз без жертвування якістю, охоплюючи понад 30 мов та фреймворків.

- Автоматизація: він дозволяє автоматизувати безпеку в CI/CD конвеєрі з робастною екосистемою інтеграцій та інструментів аналізу компонентів з відкритим вихідним кодом.

- Гнучке розгортання: він доступний на місці, у хмарі або як AppSec-як-сервіс.

- Забезпечення безпеки хмарних додатків: він забезпечує всеохоплюючу безпеку зсуву вліво для хмарних додатків, від інфраструктури як код (IaC) до безсерверних в єдиному рішенні.

- Масштабування підприємства: він можединамічно масштабувати ваші скани SAST вгору або вниз, щоб задовольнити змінювані вимоги CI/CD конвеєра.

Snyk - зручна для розробників платформа безпеки з аналізом семантичного коду в реальному часі.

Snyk - це платформа безпеки, орієнтована на розробників, яка допомагає організаціям використовувати відкрите програмне забезпечення та залишатися в безпеці. Вона розроблена для використання в життєвому циклі розробки програмного забезпечення (SDLC), безшовно інтегруючись в існуючі процеси кодування та розгортання.

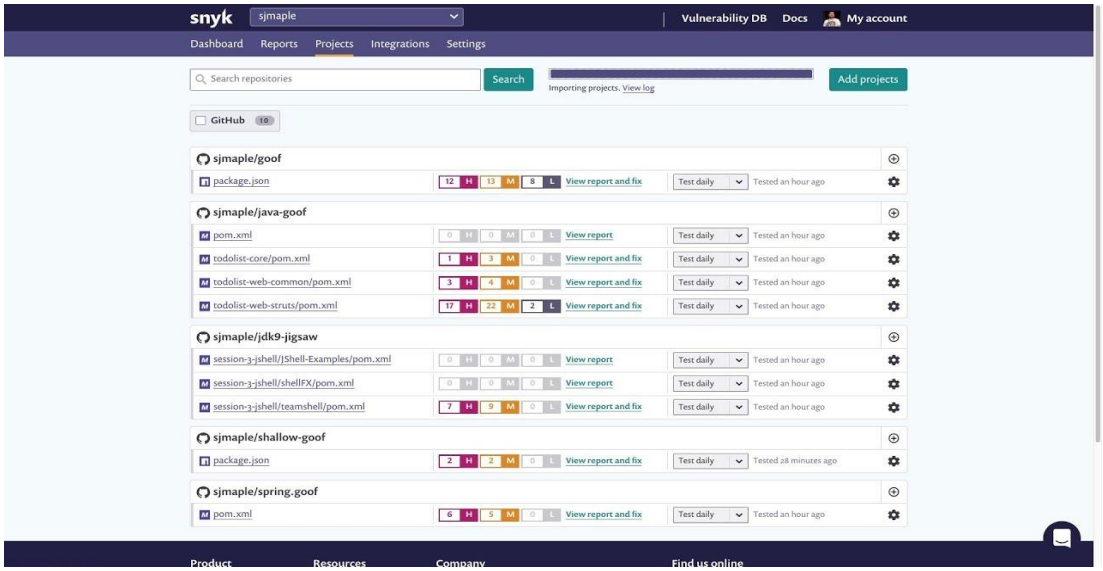

Рисунок 1.8. Скріншот панелі приладів Snyk.

Ключові особливості Snyk:

- Snyk Open Source: цей інструмент допомагає виявляти та виправляти вразливості в залежностях відкритого коду. Він надає детальну інформацію про проблеми та їх виправлення.

- Snyk Container: він допомагає знаходити та виправляти вразливості в образах контейнерів та додатках Kubernetes.

- Snyk Code: це інструмент для статичного тестування безпеки додатків (SAST), який допомагає знаходити та виправляти проблеми безпеки в коді.

- Snyk Infrastructure as Code: цей інструмент допомагає знаходити та виправляти помилки конфігурації безпеки в файлах інфраструктури як код (IaC).

- Snyk Intel Vulnerability DB: це всеохоплююча, кураторська база даних про вразливості JavaScript та Ruby.

Snyk - потужний інструмент для розробників та команд безпеки, який надає широкий спектр можливостей для забезпечення безпеки додатків від стадії кодування до розгортання.

OWASP ZAP (Zed Attack Proxy) - безкоштовний сканер безпеки веб-додатків з відкритим вихідним кодом. Це найбільш широко використовуваний у світі сканер веб-додатків, який активно підтримується міжнародною командою волонтерів.

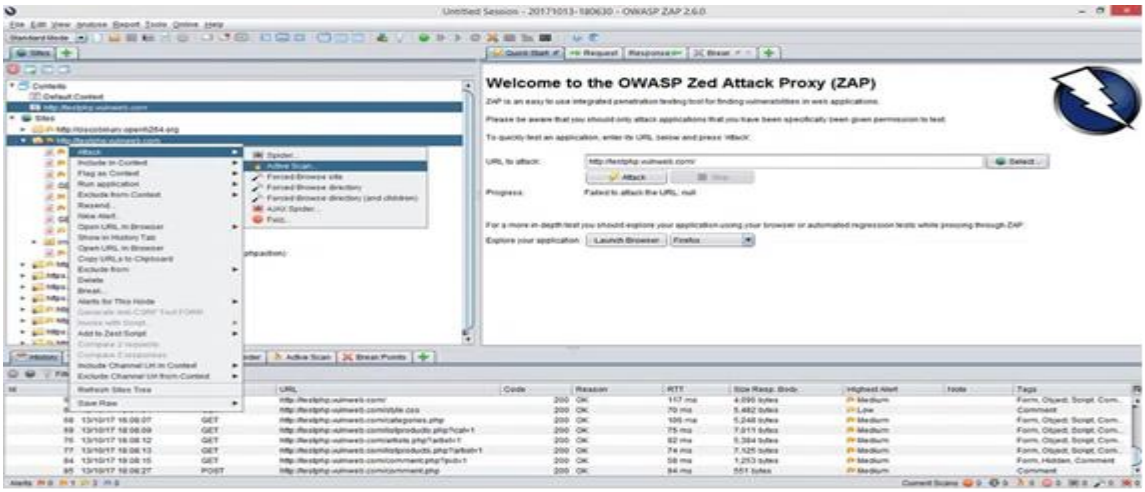

Рисунок 1.9 Скіншот панелі приладів OWASP ZAP.

Ось деякі ключові особливості ZAP:

Зручність використання: Якщо ви новачок у тестуванні безпеки, ZAP розроблений з урахуванням ваших потреб. Він надає ряд ресурсів, щоб допомогти вам почати роботу, включаючи серію відео "ZAP in Ten".

Автоматизація: ZAP надає ряд опцій для автоматизації безпеки. Це дозволяє інтегрувати ZAP у ваші процеси розробки та тестування, роблячи безперервне тестування безпеки більш ефективним.

Розширюваність: Маркетплейс ZAP містить додатки, які були внесені спільнотою. Це означає, що ви можете розширити функціональність ZAP, щоб задовольнити ваші конкретні потреби.

Активна спільнота: Будучи проектом з відкритим вихідним кодом, ZAP має велику спільноту користувачів та учасників. Це гарантує, що інструмент постійно оновлюється та вдосконалюється.

Підсумовуючи, всі ці інструменти мають свої переваги і використовуються на різних стадіях життєвого циклу розробки програмного забезпечення. SonarQube та Fortify більше зосереджені на статичному аналізі коду, Snyk великий для безпеки відкритого коду та дотримання ліцензій, OWASP ZAP - це всеохоплюючий інструмент для динамічного тестування безпеки додатків, а New Relic забезпечує моніторинг продуктивності, який може опосередковано допомогти у виявленні проблем з безпекою.

1.11 Проектування системи безпеки додатків

Проектування системи безпеки - це складний процес, який вимагає глибокого розуміння потреб організації, потенційних загроз та найкращих:

Аналіз ризиків: Перший крок полягає в ідентифікації потенційних загроз та оцінці ризиків. Це включає в себе визначення активів, які потребують захисту, потенційних загроз для цих активів та можливих наслідків, якщо ці загрози реалізуються.

Визначення вимог до безпеки: На основі аналізу ризиків визначаються вимоги до безпеки. Це можуть бути вимоги до конфіденційності, цілісності, доступності, відповідності та інших аспектів безпеки.

Вибір контрольних заходів: На основі вимог до безпеки вибираються відповідні контрольні заходи. Це можуть бути технічні заходи (наприклад, шифрування, аутентифікація), фізичні заходи (наприклад, контроль доступу до приміщень) або адміністративні заходи (наприклад, політики безпеки).

Проектування архітектури системи безпеки: На цьому етапі розробляється загальна структура системи безпеки. Це включає в себе визначення компонентів системи, їх ролей та взаємодії.

Реалізація системи безпеки: Після проектування системи безпеки вона реалізується. Це може включати встановлення та налаштування

обладнання та програмного забезпечення, розробку та впровадження політик безпеки та навчання користувачів.

Тестування та оцінка системи безпеки: Після реалізації систему безпеки необхідно протестувати, щоб переконатися, що вона працює належним

Моніторинг та постійне вдосконалення: Систему безпеки необхідно постійно моніторити та оновлювати, щоб вона залишалася ефективною в умовах зміни загроз та технологій.

Розглянемо ключові компоненти системи безпеки та їх роль:

Межі безпеки: Межі безпеки використовуються для захисту активів від зовнішніх загроз. Це може включати в себе мережеві межі, такі як файрволи, або фізичні межі, такі як контроль доступу до приміщень.

Механізми аутентифікації та авторизації: Ці механізми використовуються для перевірки того, хто має доступ до активів, та контролю того, що вони можуть робити з цими активами.

Захист даних: Захист даних включає в себе заходи, такі як шифрування та управління ключами, які використовуються для захисту даних від несанкціонованого доступу.

Моніторинг безпеки: Моніторинг безпеки включає в себе збір та аналіз даних про безпеку, таких як журнали подій та сповіщення, для виявлення та реагування на інциденти безпеки.

Відновлення після аварії: Системи відновлення після аварії використовуються для відновлення активів та служб після інциденту безпеки або іншої аварії.

Кожен з цих компонентів взаємодіє з іншими, щоб забезпечити цілісну систему безпеки. Наприклад, механізми аутентифікації та авторизації

використовуються разом з межами безпеки для контролю доступу до активів. Захист даних використовується для захисту цих активів після того, як вони були доступні. Моніторинг безпеки використовується для виявлення та реагування на будь-які спроби порушити ці заходи безпеки. Нарешті, системи відновлення після аварії використовуються для відновлення активів та служб у випадку, якщо ці заходи безпеки не вдаються.

1.12 Застосування цифрових методів контролю проектуємої програми

Заглибимося глибше в нашу програму, для якої ми збираємося розробити комплексний цифровий метод керування. Ця програма, розроблена з використанням універсального середовища виконання NodeJS, є надійним веб-сервером із набором із 10 різноманітних кінцевих точок, які взаємодіють із різними функціями.

Пропонована нами цифрова система керування структурована з трьох окремих, але взаємопов’язаних частин, кожна з яких призначена для забезпечення певного рівня перевірки та операційного контролю. Ця багаторівнева стратегія безпеки розроблена, щоб гарантувати, що програма залишається безпечною та стійкою, зміцнюючи довіру серед її користувачів.

Початковий і, мабуть, найважливіший рівень входить в сферу відповідальності розробника. Це передбачає впровадження суворого контролю в коді програми. Він включає такі практики, як дотримання стандартів безпечного кодування, впровадження обробки помилок і винятків, перевірка введених даних тощо. Очікується, що розробники внесуть ці зміни та вдосконалення безпосередньо в код. Але перед тим, як цей модифікований код буде закріплено в репозиторії Git, його критично перевіряють, щоб переконатися, що впроваджені зміни відповідають попередньо визначеним стандартам безпеки.

Другий рівень перевірки закріплено в конвеєрі безперервної інтеграції/безперервного розгортання (CI/CD). Цей етап контролю є автоматизованим і гарантує, що код добре інтегрується з існуючою системою, а будь-які вразливості виявляються на ранніх стадіях циклу розробки. На цьому етапі виконуються сценарії автоматичного тестування, перевірки безпеки та перевірки якості коду. Крім того, якщо на цьому рівні виявлено будь-які проблеми, розгортання припиняється, гарантуючи, що потенційно дефектний код не досягне виробничого середовища.

Нарешті, третій рівень передбачає постійний моніторинг і пильність програми в режимі реального часу на наявність потенційних вразливостей, що є важливим для підтримки стабільно безпечного середовища програми. Цей процес включає постійне сканування на наявність аномалій, виявлення вторгнень і впровадження необхідних патчів безпеки. Цей рівень не тільки пильно стежить за наявною програмою, але й активно вчиться на цих висновках, щоб передбачити потенційні загрози безпеці, таким чином уможливлюючи проактивні захисні заходи.

Цей комплексний трирівневий цифровий метод контролю забезпечує безпеку програми від початкового етапу кодування до її розгортання та поточних операцій. Ця стратегія поглибленого контролю забезпечує інтеграцію безпеки на кожній фазі життєвого циклу програми.

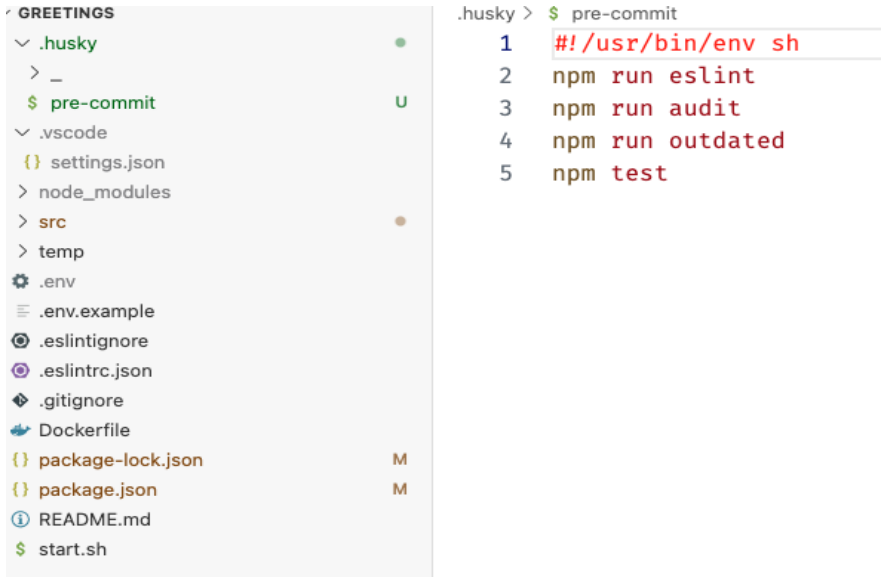

Рисунок 1.10 Скріншот файла pre-commit

Скрипт pre-commit, який використовується Husky. Husky - це інструмент для створення git-хуків, тобто скриптів, які Git виконує до або після таких подій, як коміт і пуш. Ці хуки використовуються для автоматизації завдань і допомагають забезпечити якість коду.

Ось покрокове пояснення того, що відбувається, коли ви намагаєтесь виконати комміт з цим хуком Husky pre-commit:

Коли ми запускаєте git commit, Husky перехоплює цю команду і запускає скрипт до того, як процес фіксації фактично почнеться.

Спочатку Husky виконає команду npm run eslint. Ця команда запускає інструмент ESLint у кодовій базі. ESLint - це інструмент статичного аналізу коду для виявлення проблемних патернів у JavaScript-коді. Він розроблений, щоб допомогти писати кращий і більш узгоджений код, дотримуючись правил і стандартів кодування. Якщо ESLint знайде у коді проблеми, які порушують правила, встановлені у вашій конфігурації ESLint, він видасть помилку і зупинить процес фіксації.

Якщо код пройшов перевірку ESLint, Husky переходить до запуску команди npm run audit. Ця команда запускає інструмент аудиту npm. npm audit перевіряє ваш додаток Node.js на наявність відомих вразливостей у пакунках, які ви використовуєте. Для цього він аналізує ваш файл package-lock.json і порівнює конкретні версії кожного пакета, який ви використовуєте, з базою даних відомих уразливостей JavaScript. Якщо буде знайдено будь-які відомі вразливості, npm-аудит видасть помилку і зупинить процес комміту

Якщо npm-аудит не виявив жодних відомих вразливостей, Husky виконує команду npm run outdated. Ця команда запускає npm outdated для перевірки наявності застарілих пакунків у вашій програмі. Вона порівнює поточну версію ваших пакунків з останньою доступною версією. Якщо буде знайдено застарілі пакунки, вона видасть помилку і зупинить процес фіксації.

Нарешті, якщо застарілих пакунків не знайдено, Husky запускає npm-test. Ця команда запускає будь-які тестові скрипти, які ви визначили у файлі

package.json. Якщо всі тести пройшли успішно, коммітування буде виконано. Якщо тести не пройдуть, буде видано помилку і процес комміту буде зупинено.

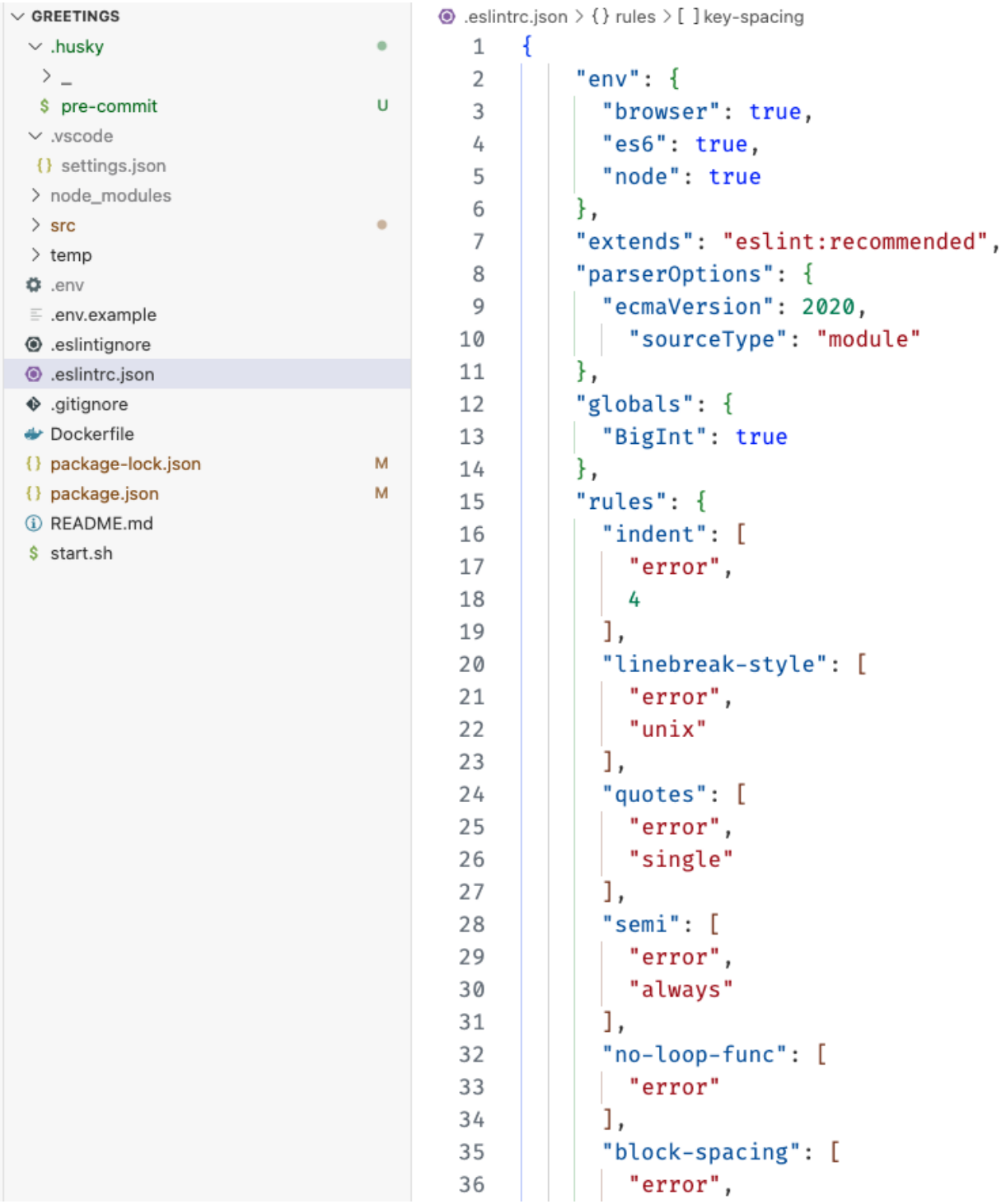

Рисунок 1.11. Скрін конфігураційного файлу ESLint

Таким чином, цей хук перед коммітом гарантує, що ваш код відповідає стандартам кодування, не має відомих вразливостей, використовує найновіші пакунки і проходить всі визначені тести перед тим, як зміни будуть зафіксовані у вашому Git-репозиторії. Це гарантує, що будь-які потенційні проблеми будуть

виявлені та вирішені на ранній стадії процесу розробки, підтримуючи загальну якість і безпеку кодової бази.

1.13 Безперервне розгортання (CI/CD)

Безперервна інтеграція/безперервна доставка (CI/CD) - це метод частої доставки додатку клієнтам шляхом впровадження автоматизації на етапах розробки додатків. Основними поняттями CI/CD є безперервна інтеграція, безперервна доставка та безперервне розгортання. CI/CD запроваджує постійну автоматизацію та безперервний моніторинг протягом усього життєвого циклу додатків, від етапів інтеграції та тестування до доставки та розгортання.

Jenkins - це сервер автоматизації з відкритим вихідним кодом, який дозволяє розробникам по всьому світу надійно створювати, тестувати та розгортати своє програмне забезпечення. Він зазвичай використовується для реалізації робочих процесів CI/CD, також відомих як конвеєри.

Ось спрощена схема того, як конвеєр CI/CD зазвичай працює з Jenkins:

Інтеграція системи контролю версій: По-перше, Jenkins потрібно підключити до системи контролю версій, такої як Git, SVN тощо, де розміщується код розробників.

Запуск збірки: Коли розробники вносять зміни до вихідного коду і фіксують ці зміни у сховищі, Jenkins можна налаштувати на автоматичне виявлення цих змін. Це можна зробити за допомогою опитування або за допомогою веб-хуків. Як тільки зміни буде виявлено, Jenkins витягне найновіший код зі сховища і запустить нову збірку.

Створення проекту: Дженкінс починає будувати проект. Це може означати різні речі залежно від типу проекту - компіляція вихідного коду, пакування скомпільованого коду у виконувані файли або образи Docker тощо.

Запуск тестів: Після збірки Дженкінс може запустити серію тестів, щоб переконатися, що зміни в коді нічого не зламали. Це можуть бути модульні тести, інтеграційні тести, функціональні тести тощо.

Звітування: Якщо збірка або тести не вдаються, Дженкінс може повідомити відповідальних розробників про проблему. Цей цикл зворотного зв'язку є одним з ключових аспектів CI/CD.

Розгортання: Якщо збірка і тести пройшли успішно, Дженкінс може автоматично розгорнути додаток в середовище для подальшого тестування, і якщо все виглядає добре, він може бути розгорнутий у виробництво.

Моніторинг: Після розгортання Дженкінс також може допомогти з моніторингом програми, запускаючи сповіщення, якщо щось піде не так.

Всі ці етапи можна налаштовувати, і ви можете додавати або видаляти етапи відповідно до потреб вашого проекту. Ці конвеєри визначаються в Jenkinsfile, який є текстовим файлом, що містить визначення конвеєра Дженкінса і перевіряється в контролі вихідного коду.

Автоматизуючи ці процеси, Jenkins допомагає зменшити кількість ручних помилок, прискорює процес розробки програмного забезпечення і гарантує, що продукт завжди знаходиться в стані, в якому його можна випустити для користувачів, таким чином досягаючи CI/CD.

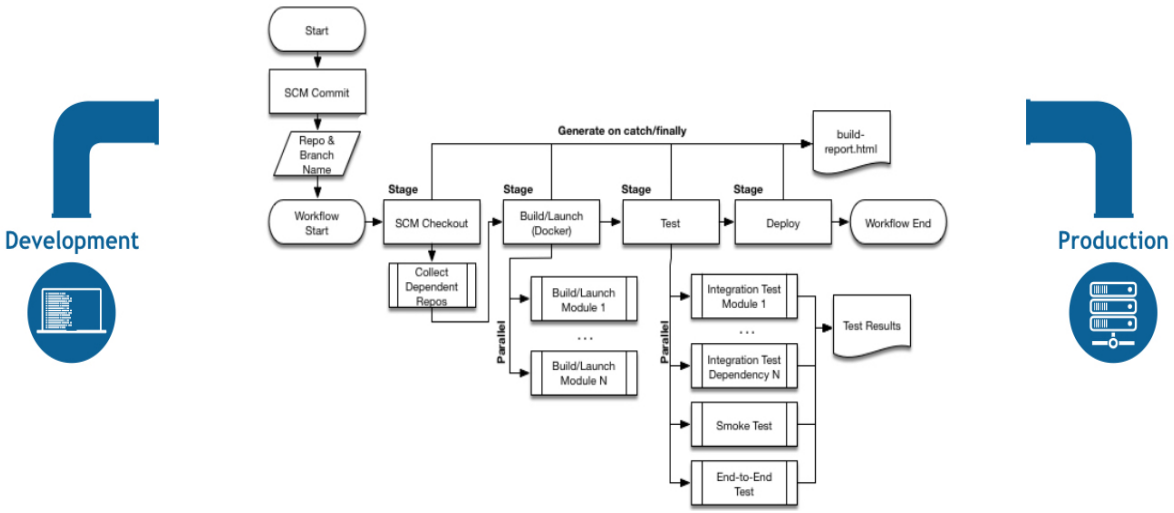

Рисунок 1.12 Блок-схема з сценаріем одного

Синтаксис конвеєра в Jenkins Pipeline:

```
pipeline {
    agent any // Виконати цей Pipeline на будь-якому доступному агенті.
    stages {
        stage('Build') { // Визначає етап "Збірка".
            steps {
```

```
                // Виконати деякі кроки, пов'язані з етапом "Збірка".
            }
        }
        stage('Test') { // Визначає етап "Тестування".
            steps {
                // Виконання деяких кроків, пов'язаних з етапом "Тестування".
            }
        }
        stage('Deploy') { // Визначає етап "Розгортання".
            steps {
                // Виконання деяких кроків, пов'язаних з етапом "Розгортання".
            }
        }
    }
}
```

Налаштування Jenkins для додатку:

Встановимо необхідні плагіни в Jenkins. Як мінімум, вам знадобиться наступне:

Плагін Git (для керування вихідним кодом)

Плагін AWS Steps (для взаємодії з AWS)

Плагін SonarQube Scanner (для перевірки якості та безпеки)

Створення конвеєра Дженкінса:

Створюємо нове завдання конвеєра в Jenkins, даємо йому назву, а потім налаштовуєте його. Ось короткий опис того, як може виглядати ваш файл Jenkins:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git branch: 'main', credentialsId: 'GIT_REPO', url:
'https://github.com/devopshint/sonarqube.git'
            }
        }

        stage('Test') {
            steps {
                    sh 'npm ci'
             sh ' npm run eslint'
```

```
                sh ' npm run audit'
                sh ' npm run outdated'
                sh ' npm test'
            }
        }

        stage('SonarQube Analysis') {
            steps {
      def scannerHome = tool 'SonarQube'
       withSonarQubeEnv('SonarQube') {
        sh
"""/var/lib/jenkins/tools/hudson.plugins.sonar.SonarRunnerInstallation/SonarQube/b
in/sonar-scanner \
      -D sonar.projectVersion=1.0-SNAPSHOT \
      -D sonar.login=admin \
      -D sonar.password=admin \
      -D sonar.projectBaseDir=/var/lib/jenkins/workspace/jenkins-sonar/ \
      -D sonar.projectKey=my-app1 \
      -D sonar.sourceEncoding=UTF-8 \
      -D sonar.language=java \
      -D sonar.sources=my-app/src/main \
      -D sonar.tests=my-app/src/test \
      -D sonar.host.url=http://3.143.3.6:9000/"""
        }            }
            }
        }

        stage('Build') {
            steps {
                sh 'npm run build'
            }
        }

        stage('Deploy to AWS EC2') {
            steps {
                // The details here depend on your deployment strategy,
                // but may involve pushing a Docker image and then
                // sending commands via ssh to the EC2 instance to
                // pull the new image and restart the service.
```

```
                // Be sure to use appropriate credentials and secure methods.
                sh 'aws ec2 ...'
            }
        }
    }
}
```

1.14 Динамічне тестування безпеки інструментом OWASP ZAP

OWASP ZAP (Zed Attack Proxy) - це безкоштовний інструмент тестування на проникнення з відкритим вихідним кодом, який допомагає знаходити вразливості у веб-додатках. Він широко використовується під час розробки та тестування для забезпечення безпеки додатків.

Кроки для встановлення та налаштування OWASP ZAP для перевірки вразливостей безпеки у додатку Node.js, з конвеєром CI/CD в Jenkins, що працює в контейнері AWS EC2.

Крок 1: Встановіть OWASP ZAP

Завантажимо та встановимо OWASP ZAP на свій екземпляр EC2. Він доступний у вигляді пакету для різних дистрибутивів ОС.

Крок 2: Налаштування OWASP ZAP
Після того, як ви встановили OWASP ZAP, вам потрібно його налаштувати.

Запускаємо програму ZAP.

Треба перейти до Інструменти -> Параметри.

Налаштувати потрібно параметри для проксі, активного сканування, пасивного сканування тощо.

Збережіть сеанс, якщо ви хочете використовувати ці налаштування повторно.

Пам'ятайте, що OWASP ZAP може діяти як проксі-сервер між вашим додатком і Інтернетом. Пропускаючи ваш HTTP/HTTPS трафік через ZAP, ви можете виявити потенційні вразливості.

Крок 3: Докеризація вашого Node.js додатку

Ваш Node.js-додаток має бути докеризований для запуску в контейнері. Це передбачає створення Docker-файлу, який повинен містити інструкції для створення образу вашого додатку.

Зробимо Docker-файл:

FROM node:14

WORKDIR /usr/src

COPY package.json ./*

RUN npm install

COPY . .
EXPOSE 3000
CMD ["node", "server.js"]

Для запуску додатку треба виконати команду
docker build -t nodejs-app .

Крок 4: Налаштування конвеєра Дженкінса

Щоб інтегрувати OWASP ZAP у ваш конвеєр CI/CD, вам слід створити завдання Дженкінса і налаштувати його так, щоб воно містило етап для сканування ZAP.

Ось приклад файлу Jenkins з етапом для ZAP-сканування:

```
pipeline {
    agent any
    stages {
        stage('Checkout') {
            steps {
                git 'YOUR_GIT_REPO_URL'
            }
        }

        stage('Test') {
            steps {
                sh 'npm ci'
                sh 'npm test'
            }
        }
```

```
        stage('Build Docker Image') {
            steps {
                sh 'docker build -t nodejs-app .'
            }
        }
        stage('ZAP Scan') {
            steps {
                sh 'zap-cli start --start-options \'-config api.disablekey=true\"
                sh 'zap-cli open-url http://localhost:8080'
                sh 'zap-cli spider http://localhost:8080'
                sh 'zap-cli active-scan --scanners \'all\' http://localhost:8080'
                sh 'zap-cli alerts'
                sh 'zap-cli shutdown'
            }
        }

        stage('Deploy to AWS EC2') {
            steps {
                sh 'aws ecr get-login-password --region region | docker login --username
AWS --password-stdin <your-repo-url>'
                sh 'docker tag my-nodejs-app:latest <your-repo-url>/my-nodejs-app:latest'
                sh 'docker push <your-repo-url>/my-nodejs-app:latest'
                // Continue with your AWS deployment...
            }
        }
    }
}
```

У цьому конвеєрі запускається ZAP, відкривається URL-адреса програми, запускається павук для виявлення вмісту сайту, виконується активне сканування, друкуються будь-які попередження і, нарешті, ZAP завершується.

Крок 5: Налаштуйте і запустіть ваше завдання Jenkins

Після того, як файл Jenkinsfile буде перевірено у вашому контролі джерел, ви можете створити нове завдання Дженкінса типу "Конвеєр" і вказати його на ваше сховище. Запускаємо завдання, і воно має виконати сканування OWASP ZAP як частину вашого процесу CI/CD.

Цей файл визначає конвеєр, який перевіряє ваш код, запускає тести, аналізує ваш код за допомогою SonarQube, збирає ваш проект, а потім розгортає його в AWS EC2.

Налаштування контролера якості SonarQube:

Ворота якості - це набір умов, яким повинен відповідати проект, перш ніж він зможе претендувати на випуск у виробництво. По суті, це рішення "йти/не йти". Щоб створити ворота якості:

1. Перейдимо на сервер SonarQube і увійдіть в систему.
2. Натискаємо на "Ворота якості".
3. Натискаємо кнопку "Створити" і назвіть його відповідно до ваших потреб.
4. Встановлюємо умови у вашому шлюзі якості (наприклад, Покриття коду, Нові помилки, Вразливості тощо).
5. .Натисніть на кнопку "Зберегти".
6. Тепер ви успішно створили ворота якості в SonarQube. Не забудьте налаштувати ваш проект на використання нових воріт якості.

Налаштування AWS EC2:

1. Переконуємося, що ваш екземпляр EC2 готовий до розгортання. Це може бути налаштування Docker або будь-який інший метод, за допомогою якого ви плануєте розгортати свій додаток. AWS CLI повинен бути правильно налаштований в Jenkins для взаємодії з вашим обліковим записом AWS.

2.Наведена вище конфігурація конвеєра є базовою відправною точкою. Залежно від специфіки вашого додатку, вам може знадобитися додати або змінити кроки відповідно до ваших потреб. Також не забудьте захистити конфіденційні дані, такі як облікові дані AWS або SonarQube, за допомогою управління секретами. Для цього в Jenkins передбачена функція "Облікові дані". Обов'язково дотримуйтесь найкращих практик безпеки, щоб убезпечити ваш процес CI/CD.

1. ЕКОНОМІЧНА ЧАСТИНА

2.1 Резюме

В даному дипломному проекті спроектована система безпеки для додатків. Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

2.2. Визначення трудомісткості розробки програмного забезпечення

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога.

Таблиця 2.1. -Каталог аналогів

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800

У таблиці 2.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця.2.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,8 = 183,2 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \quad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \quad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \quad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.3.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5.).

Таблиця 2.3.Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПО	1,75 – 1,2
Б	ПО – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПО маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПО типовими програмами, %	
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Та*L_1*К_н=183,2 \ (*0,12*0,7= 15,38 \text{ (люд/годин)} \quad (2.4)$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Та*L_2*К_н=183,2 \ (*0,11*0,7 = 14,11 \text{ (люд/годин)} \quad (2.5)$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Та*L_3*К_н*К_т=183,2 \ (*0,61*0,7*0,8= 62,58 \text{ (люд/годин)} \quad (2.6)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$=3 (стр), розробка ТП $N_{тп}$=19(стр), розробка робочого проекту $N_{рп}$=14 (стр), пояснювальна записка відповідно $N_{пз}$=36 (стр)

Розрахунок зведений у таблицю 2.6.

Найменування·етапів¤	Розрахунок,·годин.¤		
1.ТЗ¤	$T_{Pтз}$=15,38¤	$T_{кк}$=0,7*$N_{тз}$=·0,7*3=2,1¤	$T_{нк}$=0,15*$N_{тз}$=0,15*3=0,45¤
2.Розробка·ТП¤	$T_{Pтп}$=14,11¤	$T_{кк}$=0,7*$N_{тп}$=0,7*19=13,3¤	$T_{нк}$=0,15*$N_{тп}$=0,15*19=2,85¤
3.Розробка·РП¤	$T_{Pрп}$=·62,58··¤	$T_{кк}$=0,7*$N_{рп}$=0,7*14=9,8¤	$T_{нк}$=0,15*$N_{рп}$=0,15*14=2,1¤
4.Розробка·ПЗ¤	$T_{пз}$=1,5**$N_{пз}$=¶ 1,5*36·=54¤	$T_{кк}$=0,7*$N_{тз}$=0,7*36=25,2¤	$T_{нк}$=0,15*$N_{пз}$=0,15*36·=5,4¤
Усього,·в·т.ч.:¤	207,6¤	¤	¤
-·на·розробку¤	$\Sigma T_{р}$=146.1¤	¤	¤
-·контроль¶ ·керівника¤	¤	$\Sigma T_{кк}$=50,7¶ ¤	¤
-·нормоконтроль	¤	¤	$\Sigma T_{нк}$=10,8¤

2.3. Розрахунок ціни програмного продукту

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 2.7. Відповідно до статті 8 «Закону про Державний бюджет України на 2021» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6700 гривень; мінімальну погодинну тарифну ставку – 40,46 грн.

Таблиця 2.7. - Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн	Розрахунок
1.Розробка ПП	146,1	40,46	5911,21
2.Контроль керівника	50,7	80,00	4056,00
3.Нормоконтроль	10,8	70,00	756,00
Усього	-	-	ΣЗо= 10723,21

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П=(С_п*Р)/100=(18910,81*15)/100=2836,62$ грн (2.4)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о = С_п + П = 18910,81 + 2836,62 = 21747,43$ грн; (2.5)

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$Ц_р = Ц_о + ПДВ = 21747,43 + 21747,43 * 0.2 = = 26096,92$ грн (2.6)

3. ОХОРОНА ПРАЦІ

Сучасний розвиток технічного та технологічного стану виробництва тягне за собою постійну автоматизацію та оптимізацію виробничих процесів.

Сьогодні, мабуть, важко уявити собі підприємство, в якому господарська діяльність провадиться без використання комп'ютерних технологій. Важко уявити підприємство, у якому господарську діяльність здійснюється без використання комп'ютерних технологій. Тому що.

через характер роботи, яку виконують співробітники з використанням комп'ютерів,

Українське законодавство чітко визначає стандарти та вимоги до використання комп'ютерної техніки на підприємствах, а також прямі та охорону праці під час використання комп'ютерів на підприємствах.

У розділі "Охорона праці" дипломної роботи розглядаються основні шкідливі та небезпечні виробничі чинники, що виникають у процесі праці.

Питання охорони праці має вирішуватися всіх етапах робочого процесу, незалежно від виду професійної діяльності. До розгляду беремо робоче місце програміста ПК

3.1 Аналіз небезпечних та шкідливих факторів, що впливають на програмістів під час роботи

Виявлення та вивчення факторів, що впливають на функціональний стан користувачів комп'ютерів, дозволить визначити основні причини напруги, втоми та стресових станів та вжити відповідних профілактичних заходів. Зміни в людини можуть бути викликані різними чинниками однакової складності. Це можуть бути фактори робочого середовища, надмірні фізичні чи розумові навантаження, нервовий чи емоційний стрес, а також різні комбінації цих причин. Робота з комп'ютером, як і будь-яка інша робота, має професійні ризики, які враховуються при розробці відповідних правил і норм експлуатації та управління.

Трудова діяльність користувачів комп'ютерів відбувається у специфічному робочому середовищі, що впливає на їх функціональний стан. До найважливіших фізичних факторів робочого середовища відносяться електромагнітне випромінювання в різних діапазонах частот, електростатичні поля, шум, мікрокліматичні параметри та багато параметрів освітлення. Хімічні чинники робочого середовища, особливо біологічні, значно менше впливають на користувачів комп'ютерів.

Робочий процес значно впливає на психофізіологічні можливості користувачів комп'ютерів. Це пов'язано з тим, що їхня діяльність характеризується великими статичними навантаженнями на організм, недостатньою руховою активністю, напругою органів чуття та вищих нервових центрів, що забезпечують функції уваги, мислення та рухової регуляції. З іншого боку, робочі процеси користувачів комп'ютерів характеризуються великим інформаційним навантаженням.

3.2 Виробниче приміщення

Площа приміщення, в якому встановлено ПК, визначається відповідно до Державних санітарних правил і норм. Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2-007-98) норм із розрахунку на одне робоче місце з ПК.

Вони становлять:

- площа приміщення – не менше 6,0 кв. м,

- об'єм приміщення – не менше 20,0 кубічних метрів, з урахуванням максимальної кількості одночасно працюючих людей протягом однієї зміни.

- Робочі місця повинні розташовуватись на відстані не менше 1 м від стіни з вікном,

- відстань між бічними сторонами комп'ютера має бути не менше 1,2 м;

- Відстань між задньою частиною комп'ютера та екраном іншого комп'ютера має бути не менше 2,5 м.

- Відстань між рядами робочих місць має бути не менше ніж 1 м.

Робочі місця з ПК не повинні розташовуватись у підвалі або на цокольному поверсі будівлі. Небезпечні хімічні речовини не повинні бути присутніми. Підлоги повинні бути матовими, з плоскою поверхнею, нековзними та антистатичними. Заземлені конструкції у приміщенні (наприклад, радіатори, водопровідні труби, кабелі з відкритими заземленими екранами) повинні бути надійно захищені від випадкового дотику діелектричними екранами або сітками. Особливу увагу слід приділити колірній гармонії офісних приміщень.

Колір - це засіб забезпечення психологічного комфорту та підвищення продуктивності праці. Найбільш сприятливими для нервової системи є світлі пастельні кольори, такі як синьо-зелений, світло-сірий та золотий. Яскраві та контрастні поєднання (синій та помаранчевий, червоний та фіолетовий) можуть викликати роздратування. Вікна цієї кімнати виходять на південь. Тому переважаючий колір інтер'єру – світло-блакитні стіни та зелена підлога.

У кімнатах з комп'ютерами необхідно щодня проводити вологе прибирання, щоб підлога та меблі не забруднювалися пилом. Ці приміщення також повинні бути обладнані аптечками першої допомоги та автоматичною пожежною сигналізацією з датчиками диму та двома переносними вуглекислотними вогнегасниками на кожні 20 кв. м майдану. Доступ до вогнегасників має бути забезпечений у будь-який час.

3.3 Вимоги до робочого середовища ПК

Робоче середовище посідає важливе місце серед причин виникнення професійних захворювань у користувачів ВДТ. Основними є ті, які виникають під впливом випромінювання від ВДТ, блискавки, шуму, вмісту шкідливих речовин у повітрі робочого місця, іонного складу повітря, електростатичних полів і т.д.

3.4 Формулювання заходів з охорони праці

Робота з комп'ютером характеризується значною розумовою напругою та нервово-емоційним навантаженням на оператора, високим зоровим навантаженням і, при використанні клавіатури, досить високим навантаженням на м'язи рук.

Використання комп'ютера. Раціональна конструкція та розташування компонентів робочого місця важливі для підтримки оптимальної робочої пози людини-оператора.

3.4.1 Ергономічна конструкція робочого місця

Розташування робочих місць має бути засноване на наступному.

Правильне розташування робочого місця у виробничому приміщенні, вибір виробничих меблів, раціональне розміщення комп'ютерного обладнання на робочому місці та облік характеру та особливостей трудової діяльності. Робоче місце користувача має такі основні елементи: сидіння, спинка та підлокітники є знімними. Робоче сидіння є поворотним, а висота, кути нахилу сидіння та спинки регулюються. Сидіння плоске та має закруглений передній край.

Конструкція робочого місця забезпечує оптимальну робочу позу.

Оптимальну робочу позу забезпечують такі ергономічні характеристики

- Ноги на підлозі або підставці для ніг, стегна горизонтально, передпліччя вертикально, лікті зігнуті під кутом 70°-90° до вертикальної площини, зап'ястя зігнуті під кутом не більше 20° до горизонтальної площини, Нахил голови - від 15° до 20° відносно вертикальної площини. Обладнання розміщується на основному робочому столі з лівого боку. Стіл має висоту робочої поверхні 680-800 мм і ширину, що дозволяє виконувати роботи в межах досяжності моторного поля. Розміри столу: висота - 725 мм, ширина - від 600 до 1400 мм, глибина - від 800 мм до 1000 мм Робочий стіл оснащений регульованою по

висоті підставкою для ніг шириною 400 мм. Основа рифлена і має бортик висотою 10 мм по передньому краю.

3.4.2 Освітлення робочого місця

Правильно спроектоване і впроваджене промислове освітлення покращує візуальні умови праці, зменшує втому, підвищує продуктивність, сприятливо впливає на виробниче середовище, має позитивний психологічний вплив на працівників, підвищує безпеку праці і знижує травматизм.

Недостатнє освітлення викликає зорову втому, знижує пильність і призводить до швидкої стомлюваності. Надмірно яскраве освітлення викликає відблиски, подразнення і колючий біль в очах.

Освітлення на робочому місці створює різкі тіні, відблиски і збиває з пантелику працівників. Всі ці причини можуть призвести до нещасних випадків і професійних захворювань, тому дуже важливо, щоб освітлення було правильно розраховане.

Приміщення, де встановлюються персональні комп'ютери, повинні бути забезпечені природним і штучним освітленням відповідно до ДБН В.2.5-28:2018 “Природне і штучне освітлення”.

Природне освітлення повинно забезпечуватися переважно через світлові прорізи, розташовані з північної або північно-східної сторони, з коефіцієнтом природного освітлення (КПО) не більше 0,5%.

Штучне освітлення в приміщеннях з робочими місцями забезпечується загальними системами рівномірного освітлення.

При великому обсязі роботи з документами може застосовуватися комбінована система освітлення (до системи загального освітлення встановлюються додаткові світильники).

(встановлюються додаткові лампи місцевого освітлення). Освітленість поверхні робочого столу, на якому розміщуються документи, повинна бути від 300 до 500 лк, а освітленість екрану

не повинна перевищувати 300 люкс.

Для штучного освітлення джерелом світла є переважно люмінесцентні лампи типу ЛБ. Також для відбивного освітлення в приміщеннях, де переважно працюють з документами, використовуються метало галогенні лампи потужністю 250 Вт. Для місцевого освітлення можуть використовуватися лампи розжарювання. Система загального освітлення повинна складатися з безперервної або переривчастої лінії світильників, розташованих з боків робочого місця, переважно з лівого боку від робочого місця, паралельно лінії зору працівника.

Використання світильників без розсіювачів або захисних решіток заборонено. Яскравість світильників загального освітлення в зоні з кутом випромінювання від 50° до 90° з вертикальною і горизонтальною площинами не повинна перевищувати 200 кд/м2.

3.4.3 Мікроклімат

Робочі місця, де використовуються персональні комп'ютери, повинні бути обладнані системами опалення, кондиціонування повітря та припливно-витяжної вентиляції. На робочих місцях повинні бути забезпечені оптимальні значення параметрів мікроклімату, такі як температура, відносна вологість і рухливість повітря. Відповідно до гігієнічних норм ДСН 3.3.6.042-99 "Мікроклімат виробничих приміщень", температура повітря в офісних приміщеннях повинна становити 22-25°С, відносна вологість 40-60%, а швидкість руху повітря не повинна перевищувати 0,1 м/с.

Параметри нормуються залежно від пори року та категорії важкості робіт. Оптимальні параметри мікроклімату встановлені для постійних робочих місць, якими є робочі місця операторів ПК.

У таблиці 3.1. наведені оптимальні параметри мікроклімату для приміщень, де виконуються роботи операторського типу.

Таблиця 3.1 - Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С 40...60% 0,1...0,2 м/с

Для нормалізації параметрів мікроклімату слід звернути увагу на наступне.

Використовувати кондиціонування повітря в приміщеннях або подавати свіже повітря через систему вентиляції. Для підтримання прийнятних показників мікроклімату та концентрації позитивних і негативних іонів передбачені системи або пристрої зволоження та/або штучної іонізації та кондиціонування повітря.

3.4.4 Шум.

Джерелами шуму при використанні комп'ютера є жорсткі диски, вентилятори блоку живлення, вентилятори процесора, високошвидкісні CD-ROM, механічні сканери та рухомі механічні частини принтерів.

Під час роботи матричного голчастого принтера шум генерується рухом головки принтера та ударами голки по паперу. Аеродинамічний шум генерується системою вентиляції ПК, встановленою на задній панелі, яка забезпечує оптимальні температурні умови для електронних компонентів ПК. Інші шуми надходять від зовнішніх джерел, які не мають нічого спільного з роботою комп'ютера. В офісах і приміщеннях з персональними комп'ютерами та бізнес-обладнанням допустимий рівень шуму становить 50 дБА протягом 24 годин з максимальним рівнем шуму 65 дБА.

3.4.5 Електробезпека

Персональні комп'ютери, периферійне обладнання, інше обладнання (органи управління, контрольно-вимірювальні прилади, світильники), проводи та кабелі повинні мати відповідну конструкцію та клас захисту і бути обладнані пристроями аварійного захисту, наприклад, від струмів короткого замикання. Лінії електроживлення персональних комп'ютерів і периферійного обладнання повинні бути виконані як окрема групова 3-провідна мережа, з прокладанням фазних, нульових і захисних провідників.

провідники. Нульовий захисний провідник використовується для заземлення споживачів електроенергії (занулення). Персональні комп'ютери та периферійні пристрої повинні підключатися до електромережі тільки за допомогою ремонтопридатних штепсельних з'єднань і штепсельних розеток заводського виготовлення.

Якщо в одному приміщенні встановлено до п'яти комп'ютерів і периферійних пристроїв, можна використовувати трипровідну систему електропроводки. Дроти або кабелі, захищені оболонкою з негорючого або важкозаймистого матеріалу по периметру приміщення, без металевих труб або гнучких металевих рукавів.

3.5 Робочий час і час відпочинку

Глава 5 ДСанПіН 3.3.2-007-98 встановлює вимоги до режимів праці та відпочинку при роботі з ВДТ і персональними комп'ютерами.

При роботі з персональними комп'ютерами в робочий час повинні бути передбачені перерви для відпочинку з метою охорони здоров'я працівників, запобігання професійним захворюванням і збереження працездатності.

З метою охорони здоров'я працівників, запобігання професійним захворюванням і збереження працездатності повинні бути визначені перерви

протягом зміни. Внутрішньо змінний режим праці та відпочинку повинен передбачати додаткові короткочасні перерви в період до появи ознак об'єктивної та суб'єктивної втоми або зниження працездатності.

Зниження працездатності Основним завданням використання персонального комп'ютера слід вважати роботу з ним, яка займає не менше 50% часу протягом робочої зміни. З метою збереження здоров'я працівника, профілактики професійних захворювань і підтримки працездатності рекомендується передбачати встановлені періоди відпочинку (короткі перерви) протягом робочої зміни:

- Перерва для відпочинку і харчування (обідня перерва) - тривалість обідньої перерви визначається чинним трудовим законодавством та правилами внутрішнього трудового розпорядку підприємства (організації, установи);

- перерви для відпочинку та особистих потреб (згідно з правилами внутрішнього трудового розпорядку);

- додаткові перерви для відпочинку, що встановлюються для окремих професій з урахуванням специфіки трудової діяльності.

3.6 Пожежна безпека при використанні ВДТ

Пожежна безпека забезпечується системами запобігання та захисту від пожежі. Всі офіси повинні мати план евакуації на випадок пожежі, який визначає дії працівників у разі виникнення пожежі та вказує на місцезнаходження протипожежного обладнання. Пожежа є особливо небезпечною, оскільки тягне за собою значні матеріальні втрати.

Сучасні комп'ютери мають дуже високу щільність елементів електронних схем. З'єднувальні дроти і кабелі розташовані в безпосередній близькості один від одного. Коли через них протікає струм, виділяється значна кількість тепла. Це може призвести до розплавлення ізоляції. Відведення тепла.

Надлишок тепла в ПК відводиться системами вентиляції та кондиціонування повітря. Якщо ці системи працюють постійно, вони створюють додаткову пожежну небезпеку.

Горючі компоненти включають будівельні матеріали та перегородки для акустики та естетики приміщення.

До горючих компонентів відносяться будівельні матеріали для акустичної та естетичної обробки приміщень, перегородки, двері, підлоги, перфокарти і перфострічки, ізоляція кабелів тощо.

Джерелами займання можуть бути електронні схеми в ПК та побутовій техніці, джерела живлення та кондиціонери, які використовуються для обслуговування, що може призвести до перегріву елементів, електричних іскор та дуг в результаті різних несправностей, які можуть запалити горючі матеріали.

До засобів пожежогасіння належать внутрішні протипожежні водопроводи (ПК), вогнегасники (вуглекислотні та порошкові) і сухий пісок.

У будівлях пожежні крани розташовані в коридорах і на сходових клітинах. Кожен кран оснащений пожежним рукавом і розміщений у відповідному ящику на висоті 1,35 м над підлогою.

Вогнегасники слід розміщувати на видному місці, на висоті не більше 1,5 м над підлогою. Вогнегасники слід встановлювати на вертикальних перегородках або стінах, на спеціальних кронштейнах або в пожежних шафах. Інструкція та правила користування, надруковані на корпусі вогнегасника, повинні бути звернені назовні і бути добре видимими в надзвичайній ситуації. На механізм приведення в дію та дверцята пожежної шафи повинні бути нанесені захисні пломби.

Виробниче приміщення повинно бути забезпечене евакуаційними виходами. Такі двері повинні бути

освітлені та позначені написом "аварійний вихід". План евакуації повинен бути вивішений на видному місці біля головних виходів з приміщення.

План евакуації повинен бути вивішений на видному місці біля головного виходу з приміщення.

ВИСНОВКИ

У ході виконання цієї роботи було проведено дослідження та аналіз методів та інструментів для розробки безпечних додатків за допомогою цифрових методів контролю. Визначено, що ключовими етапами в розробці безпечних додатків є ідентифікація потенційних загроз, визначення вимог до безпеки, реалізація системи безпеки, тестування та оцінка системи безпеки, а також постійний моніторинг та вдосконалення.

Було виявлено, що впровадження суворого контролю в коді програми, включаючи дотримання стандартів безпечного кодування, обробку помилок і винятків, перевірку введених даних, є критично важливим на початковому етапі розробки. Це включає в себе використання інструментів, таких як Husky, для створення git-хуків, що допомагають автоматизувати завдання та забезпечити якість коду.

Було встановлено, що на другому рівні перевірки, в рамках конвеєра безперервної інтеграції/безперервного розгортання (CI/CD), використовуються автоматизовані тести, перевірки безпеки та перевірки якості коду. Це допомагає забезпечити, що код добре інтегрується з існуючою системою, а будь-які вразливості виявляються на ранніх стадіях циклу розробки.

Третій рівень передбачає постійний моніторинг і пильність програми в режимі реального часу на наявність потенційних вразливостей, що є важливим для підтримки стабільно безпечного середовища програми.

Використання інструментів, таких як Jenkins та OWASP ZAP, було визнано ефективним для автоматизації процесів розробки, тестування та моніторингу безпеки додатків.

Висновок: Впровадження цифрових методів контролю в процес розробки додатків є критично важливим для забезпечення їх безпеки. Це включає в себе використання стандартів безпечного кодування, автоматизованого тестування та моніторингу, а також постійного вдосконалення системи безпеки. Ці методи допомагають ідентифікувати та усунути потенційні вразливості на ранніх стадіях розробки, а також забезпечують постійний контроль за безпекою

додатку після його розгортання. Використання інструментів, таких як Husky, Jenkins та OWASP ZAP, може значно підвищити ефективність цих процесів. Отже, розробка системи безпеки додатків за допомогою цифрових методів контролю є важливим аспектом сучасної розробки програмного забезпечення.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бішоп, Метт. Вступ до комп'ютерної безпеки. - Видавництво Pearson, 2004.

2. Ховард, Майкл та Ліпнер, Стів. The Security Development Lifecycle. - Microsoft Press, 2006.

3. Шварц, Етієн. Secure Programming with Static Analysis. - Addison-Wesley Professional, 2007.

4. Мак-Гроу, Джейсон. Software Security: Building Security In. - Addison-Wesley Professional, 2006.

5. Node.js Security Working Group: https://github.com/nodejs/security-wg

6. OWASP ZAP User Guide: https://www.zaproxy.org/docs/

7. Husky Documentation: https://typicode.github.io/husky/#/

8. Jenkins User Documentation: https://www.jenkins.io/doc/

9. Node.js Best Practices: https://github.com/goldbergyoni/nodebestpractices

10. https://snyk.io/blog/ten-node-js-security-best-practices/

11. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

12. Software Security: https://www.coursera.org/learn/software-security

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Димитрука Глеба Юрійовича

(прізвище, ім`я та по батькові)

Спеціальність: 123 "Комп'ютерна інженерія"

Освітня програма: «Обслуговування комп'ютерних систем і мереж»

Тема дипломного проекту: Розробка системи безпекі додатків за допомогою цифрових методів контролю

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка містить 77 сторінка. У пояснювальній записці виконано опис етапів розробки системи безпеки додатків за допомогою цифрових методів, а також його програмне забезпечення. Графічна частина складається з 14 слайдів мультимедійної презентації, які також містять креслення, передбачені технічним завданням. Якість виконання пояснювальної записки та графічної частини добра, розробку виконано в повному обсязі.

б) самостійність роботи над проектом: Протягом всього строку дипломного проектування та переддипломної практики здобувач освіти Димитрук Г.Ю. поступово та послідовно виконував всі етапи розробки. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника

в) теоретична підготовка випускника (випускниці): Здобувач освіти Димитрук Г.Ю. під час роботи над дипломним проектом вивчив достатню кількість літературних джерел та матеріалів за даною тематикою.

Вважаю, що теоретична підготовка дипломника добра і він готовий до захисту дипломного проекту

г) вміння розв'язувати виробничі та конструкторські питання

Під час дипломного проектування здобувач освіти Димитрук Г.Ю. мав змогу самостійно приймати окремі рішення з реалізації системи безпеки додатківта показав вміння організовано працювати над поставленим завданням, складати блок-схему та програмні коди за допомогою сучасних комп'ютерних програмних засобів та мов програмування.

Оцінка розрахункової частини Відмінно

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові керівника дипломного проекту

Шевцов Юрій

Місце роботи і посада керівника дипломного проекту

ВСП "Одеський технічний фаховий коледж ОНТУ", викладач спецдисциплін комісії комп'ютерних технологій та програмної інженерії,

Підпис

«16» 06 2023 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект (роботу) здобувача (здобувачки) освіти

відділення комп'ютерних систем

Димитрюка Глеба Юрійовича

(прізвище, ім`я та по батькові)

Спеціальність **123 Комп'ютерна інженерія**

Освітня програма **«Обслуговування комп'ютерних систем і мереж»**

Керівник дипломного проекту (роботи) к.т.н.,Кунуп Т.В.

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) **Розробка системи безпеки додатків за допомогою цифрових методів контролю**

Обсяг розрахунково-пояснювальної записки 77 сторінок

Обсяг графічної (презентаційної) частини 12 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлений на рецензію робота відповідає затверджений темі та виконаний відповідно технічному завданню. Дипломний проект є актуальним з погляду безпеки додатків.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Пояснювальна записка складається з технологічної частини, розробки структури системи, опису і сучасних систем безпеки додатків, економічної частини, розділу охорони праці та додатку. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано. Розділ охорони праці містить загальну інформацію та вимоги до техніки безпеки оператора ЕОТ. Економічна частина проекту містить розрахунок затрат на виконання та реалізацію проекту

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи)

Графічна частина складається з 15 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять ілюстративні схеми, блок-схеми алгоритмів, скріншоти роботи програмних застосунків, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання графічної частини проекту та пояснювальної записки висока, розробку виконано у повному обсязі

г) перелік позитивних якостей дипломного проекту (роботи)

У роботі досить обґрунтовано досліджено основні характеристики принципів робіт сучасних систем безпеки додатків, що використовуються повсюдно.

д) основні недоліки дипломного проекту (роботи)

1. Занадто великий обсяг аналітичного матеріалу.

2. Немає посилання на використану літературу.

Оцінка розрахункової частини 4(добре)

Оцінка графічної частини 4(добре)

Загальна оцінка 4(добре)

Прізвище, ім'я, по батькові рецензента Кривенко Юрій Вікторович

Місце роботи і посада рецензента ВСП "ФФК ОНТУ", голова ЦК КТ та ПІ

Підпис:

«16» червня 2023 р.

UNICHECK
by Turnitin

Ім'я користувача:
Наталія Вікторівна Копусь

ID перевірки:
1015661292

Дата перевірки:
20.06.2023 21:58:30 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
20.06.2023 21:59:55 EEST

ID користувача:
100011688

Назва документа: **4КС-56 Димитрюк Г.Ю**

Кількість сторінок: **88** Кількість слів: **15527** Кількість символів: **114951** Розмір файлу: **1.78 MB** ID файлу: **1015306181**

5.97%
Схожість

Найбільша схожість: **0.43%** з Інтернет-джерелом (https://er.nau.edu.ua/handle/NAU/45189)

5.97% Джерела з Інтернету	890	Сторінка 90

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	14

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Димитрюк Гліб Юрійович,
здобувач освіти гр. 4КС-56, та

Шевцов Юрій Сергійович,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи молодшого спеціаліста на тему:

«Розробка системи безпеки додатків за допомогою цифрових методів» (автор роботи – Димитрюк Г.Ю., керівник роботи – Шевцов Ю.С.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2023 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи, і даємо згоду на обробку персональних даних.

Виконавець _____ / Димитрюк Г.Ю./

Керівник _____ / Шевцов Ю.С./

«12» 06 2023 р.