ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ

КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ

Пояснювальна записка

до магістерської роботи
на тему:

«ТЕХНОЛОГІЯ РОЗРОБКИ БЕЗПЕЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ»

Виконав студент 6 курсу, групи БСДМ-61
спеціальності 125 Кібербезпека
освітньо-професійної програми «Інформаційна та кібернетична безпека»
(шифр і назва спеціальності)

Поремський М.О.
(прізвище та ініціали)

Керівник Гахов С.О.
(прізвище та ініціали)

Рецензент
(прізвище та ініціали)

Нормоконтролер Чумак Н.С.
(прізвище та ініціали)

КИЇВ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Інститут ННІЗІ
Кафедра Інформаційної та кібернетичної безпеки
Ступінь вищої освіти Магістр
Спеціальність 125 Кібербезпека
Освітньо-професійна програма Інформаційна та кібернетична безпека

ЗАТВЕРДЖУЮ
Завідувач кафедри ІКБ
_____ Гайдур Г.І.
"___"_____2021 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Поремському Максиму Олександровичу
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи: «Технологія розробки безпечного програмного забезпечення для сучасних інформаційних систем»

керівник магістерської роботи Гахов Сергій Олександрович, к.військ.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від «___» _____ 2021 року №____.

2. Строк подання студентом магістерської роботи 15.12.2021 р.

3. Вихідні дані до магістерської роботи
корпоративна інформаційна система;
технологія розробки безпечного програмного забезпечення;
наукова та технічна література, експлуатаційна документація, нормативні документи, міжнародні стандарти.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Актуальність проблеми захисту процесу розробки безпечного програмного забезпечення.
2. Склад та умови функціонування корпоративної інформаційної системи.
3. Методи за засоби захисту життєвого циклу розробки.
4. Технологія безпечного процесу розробки програмного забезпечення
5. Перелік графічного матеріалу

1. Тема магістерської роботи.

2. Об’єкт, предмет, мета та наукові завдання дослідження.

3. Результати аналізу різних життєвих циклів розробки програмного забезпечення та їх вплив на кібербезпеку.

4. Результати аналізу процесів та методів забезпечення безпеки в DevOps моделі.

5. Методи за засоби захисту веб додатків.

6. Методи за засоби захисту хмарного середовища.

7. Методи за засоби захисту CI/CD.

8. Варіант технології забезпечення безпечного процесу розробки програмного забезпечення.

9. Рекомендації щодо застосування технології забезпечення розробки безпечного програмного забезпечення у сучасних інформаційних системах.

10. Висновки за результатами роботи.

6. Дата видачі завдання 27.09.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ зп	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1.	Визначення актуальності проблеми захисту життєвого циклу розробки.	27.09.2021 р.	
2.	Аналіз наукової та технічної літератури з питань теми магістерської роботи.	07.10.2021 р.	
3.	Аналіз процесів та методів забезпечення безпеки в DevOps моделі.	23.10.2021 р.	
4.	Розроблення варіанту технології для забезпечення розробки безпечного програмного забезпечення.	02.10.2021 р.	
5.	Розроблення рекомендацій щодо застосування технології безпечного процесу розробки програмного забезпечення.	15.11.2021 р.	
6.	Оформлення результатів дослідження. Проходження плагіату	26.11.2021 р.	
7.	Підготовка доповіді до захисту.	15.12.2021 р.	

Студент Поремський М.О.
(підпис) прізвище та ініціали

Керівник магістерської роботи Гахов С.О.
(підпис) прізвище та ініціали

ВІДГУК РЕЦЕНЗЕНТА
на магістерську роботу

студента Поремського Максима Олександровича

на тему: «Технологія розробки безпечного програмного забезпечення для сучасних інформаційних систем»

Актуальність:
Методології розробки програмного забезпечення для сучасних інформаційних систем змінились. Також змінилось і середовище та сервіси, які використовують розробники програмного забезпечення. Для того, щоб захистити підприємство та програмне забезпечення, треба підійти з новими підходами щодо захисту життєвого циклу розробки та підприємства. Реалізація технології та процесів захисту життєвого циклу розробки програмного забезпечення є важливою темою. Тому тема магістерської роботи є актуальною та своєчасною.

Позитивні сторони:
1. На основі проведеного аналізу, в роботі встановлено зміст проблеми забезпечення захисту життєвого циклу розробки програмного забезпечення, визначено мета та завдання захисту життєвого циклу розробки.
2. Досліджено методи та засоби життєвого циклу розробки програмного забезпечення та сучасної інфраструктури.
3. Запропоновано варіант технології захисту життєвого циклу розробки програмного забезпечення та рекомендації щодо її застосування.
4. Текст викладено достатньо грамотно, послідовно. Сформульовано чіткі та змістовні висновки. Графічний матеріал оформлено якісно. Список науково-технічної літератури свідчить про вміння користуватись матеріалами за темою магістерської роботи.

Недоліки:
1. У магістерській роботі бажано було б провести аналіз інструментів захисту хмарної інфраструктури.
2. Запропонований варіант захисту процесів розробки програмного забезпечення та хмарного середовища було би краще показати на прикладі існуючого підприємства.

Висновок: Враховуючи недоліки, магістерська робота заслуговує оцінку «**добре»**, а студент **Поремський Максим Олександрович** - присвоєння кваліфікації 2149.2 професіонал з організації інформаційної безпеки, викладач закладу вищої освіти.

Якість роботи	
Виконано на замовлення підприємства	
Виконано за тематикою НДР	
Виконано з макетом	
Виконано з застосуванням ЕОМ та МПТ	√
Має практичну цінність	√
Проект-частина комплексної теми	

Підпис рецензента (_____)

_____

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ МАГІСТЕРСЬКОЇ РОБОТИ

Направляється студент Поремський М.О. до захисту магістерської роботи
(прізвище та ініціали)
спеціальності 125 Кібербезпека
освітньо-професійної програми Інформаційна та кібернетична безпека
(шифр і назва спеціальності)
на тему: «Технологія розробки безпечного програмного забезпечення для сучасних інформаційних систем».

Магістерська робота і рецензія додаються.

Директор інституту _____ _____ Савченко В.А.
(підпис) (прізвище та ініціали)

Довідка про успішність

Поремський М.О. за період навчання в інституті
(прізвище та ініціали студента)
ННІЗІ з 2020 року по 2022 рік повністю виконав навчальний план за напрямом підготовки, спеціальністю з таким розподілом оцінок за:
національною шкалою: відмінно ____%, добре ____%, задовільно ____%;
шкалою ECTS: A ____%; B ____%; C ____%; D ____%; E ____%.

Секретар інституту _____ Черниш О.В.
(підпис) (прізвище та ініціали)

Висновок керівника магістерської роботи

Студент ***Поремський М.О.*** обрав тему роботи, метою якої було дослідити зміст технології розробки процесу розробки безпечного програмного забезпечення для сучасних інформаційних систем та розробити варіант захисту сучасного процесу розробки на підприємстві. Перелік використаних джерел свідчить про вміння магістром розбиратись в наукових питаннях та застосовувати їх при дослідженнях. Під час виконання магістерської роботи Поремський М.О. показав відмінну теоретичну та практичну підготовку, вміння самостійно вирішувати питання і робити висновки. Роботу виконував сумлінно, акуратно та вчасно за планом.
Все це дозволяє оцінити виконану магістерську роботу студента Поремського Максима Олександровича на оцінку **«добре»** та присвоїти йому кваліфікацію 2149.2 професіонал з організації інформаційної безпеки, викладач закладу вищої освіти.

Керівник магістерської роботи _____ Гахов С.О.
(підпис) (прізвище та ініціали)
“____” _____ 2021 року

Висновок кафедри про магістерську роботу

Магістерська робота розглянута. Студент Поремський М.О.
(прізвище та ініціали)
допускається до захисту даної магістерської роботи в Державній екзаменаційній комісії
Завідувач кафедри Інформаційної та кібернетичної безпеки
(назва)

_____ Гайдур Г.І.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи: 66 сторінки, 21 рисунків, 7 таблиці, 16 джерел.

Об'єкт дослідження – процес розробки безпечного програмного забезпечення для сучасних інформаційних систем.

Предмет дослідження – технологія розробки безпечного програмного забезпечення.

Мета роботи – розробити варіант захисту життєвого циклу розробки в корпоративній інформаційній системі та рекомендації щодо застосування технології захисту на підприємстві.

Методи дослідження – опрацювання літератури за даною темою, аналіз експлуатаційної документації, міжнародних стандартів та їх порівняння, моделювання процесу захисту життєвого циклу розробки в корпоративній інформаційній системі.

В роботі проведено аналіз проблеми забезпечення кібербезпеки життєвого циклу розробки програмного забезпечення у корпоративній інформаційній системі та визначено мета та завдання забезпечення захисту життєвого циклу розробки в корпоративній інформаційній системі. Проаналізовано існуючі технології захисту життєвого циклу розробки в корпоративній інформаційній системі.

Досліджено методи та засоби захисту життєвого циклу розробки в корпоративній інформаційній системі. Визначено процеси та методи для забезпечення безпеки в DevOps моделі веб додатків, хмарної інфраструктури та CI/CD.

На основі досліджень проведених в роботі розроблено варіант технології для забезпечення розробки безпечного програмного забезпечення в корпоративній інформаційній системі.

Галузь використання – кібербезпека корпоративної інформаційної системи.

КОРПОРАТИВНА ІНФОРМАЦІЙНА СИСТЕМА, КІБЕРБЕЗПЕКА, ЖИТТЄВИЙ ЦИКЛ РОЗРОБКИ, DEVOPS, ЗАХИСТ ВЕБ ДОДАТКІВ, ІНФРАСТРУКТУРА В ХМАРІ, CICD

ABSTRACT

Master's thesis: 66 pages, 21 figures, 7 tables, 16 sources.

Object of research – the process of developing secure software for modern information systems.

Subject of research – the technology of safe software development.

The aim of research – to develop a variant of protection of the development life cycle in the corporate information system and recommendations for the application of protection technology in the enterprise.

Research methods – elaboration of literature on this topic, analysis of operational documentation, international standards and their comparison, modeling of the process of protection of the development life cycle in the corporate information system.

The paper analyzes the problem of cybersecurity of the software development life cycle in the corporate information system and defines the purpose and objectives of ensuring the protection of the development life cycle in the corporate information system. The existing technologies of protection of a life cycle of development in the corporate information system are analyzed.

Methods and means of protection of the development life cycle in the corporate information system are studied. Processes and methods for security in DevOps model of web applications, cloud infrastructure and CI / CD are defined.

Based on the research conducted in the work, a variant of the technology was developed to ensure the development of secure software in the corporate information system.

Field of use – cybersecurity of corporate information system.

CORPORATE INFORMATION SYSTEM, CYBER SECURITY, DEVELOPMENT LIFECYCLE, DEVOPS, PROTECTION OF WEB APPLICATIONS, CLOUD INFRASTRUCTURE, CICD

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

TLS – Transport Layer Security
SSL – Secure Sockets Layer
HTTP – HyperText Transfer Protocol
HTTPS – Hypertext Transfer Protocol Secure
CSRF – Cross site request forgery
HTML – HyperText Markup Language
OWASP – Open Web Application Security Project
SAMM – Software Assurance Maturity Model
IDOR – Insecure direct object references
API – Application Programming Interface
AWS – Amazon Web Services
SDLC – Software Development Life Cycle
ZAP – Zed Attack Proxy

ВСТУП

Актуальність дослідження. Зараз інформаційні технології переживають час стрімкого розвитку, змінюються підходи до розробки програмного забезпечення та з'являються нові парадигми та методології. Наше життя стає все більш залежним від інформаційних технологій, росте їх значущість.

Цей швидкий темп змін та розробки програмного забезпечення потребує швидкого реагування з боку спеціалістів з кібербезпеки. Спеціалісти з кібербезпеки повинні не тільки вдосконалювати існуючі технології та методи захисту, але й аналізувати процеси розробки, які змінились, щоб почати розробляти нові підходи до взаємодії з іншими командами та захисту ресурсів підприємства.

Впровадження нових технологій та підходів до розробки, а саме DevOps, хмарна інфраструктура та мікросервісна архітектура, приводить до змін в ландшафті загроз та вимагає від спеціалістів з кібербезпеки адаптуватися та змінювати свої підходи до захисту інформаційних систем. Наразі не існує повного керівництва по тому, як треба адаптуватись під ці зміни. Саме тому аналіз проблеми реалізації технологій та процесів захисту життєвого циклу розробки програмного забезпечення є актуальною темою, бо з'ясувавши та описавши ці проблеми, можна перейти до побудови правильних технологій, підходів та процесів з кібербезпеки.

Ступінь наукової розробки – удосконалено підходи до захисту життєвого циклу розробки програмного забезпечення. Виявлено проблеми, які з'явились у команди розробки та команди безпеки при розробці програмного забезпечення за допомоги сучасних методологій розробки. На базі виявлених проблем, зроблено рекомендації щодо покращення рівня безпеки програмного забезпечення та інфраструктури підприємств. Через те, що в роботі було розглянуто та проведено аналіз старих методологій розробки, було указано на те, як команда безпеки може інтегрувати нові принципи щодо захисту інформації та ресурсів в сучасних середовищах розробки програмного забезпечення.

Практичне значення одержаних результатів полягає в тому, що отримані в роботі результати, а саме рекомендації щодо застосування технології захисту життєвого циклу розробки програмного забезпечення на підприємстві допоможе покращити рівень безпеки програмного забезпечення та цих підприємств.

1 АНАЛІЗ ПРОБЛЕМИ РОЗРОБКИ БЕЗПЕЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМИ

1.1. Основні поняття щодо розробки програмного забезпечення для сучасних інформаційних систем.

Інформаційні технології розвиваються дуже стрімко, змінюючи світ навколо нас, та безпосередньо впливають на наше життя. Разом з швидким розвитком інформаційних технологій, росте наша залежність від них. Від індустрії розваг до банківського та енергетичного секторів, інформаційні технології зайняли дуже важливе місце.

Перед тим як перейти до аналізу інформаційних систем, треба дати визначення для деяких термінів. Посилаючись на визначення Національного інституту стандартів та технологій США, можна сказати, що інформаційна система це дискретний набір інформаційних ресурсів, організованих для збору, обробки, підтримки, використання, обміну, поширення або розпорядження інформацією [1].

Ціллю та завданням розробки програмного забезпечення для інформаційних систем є вирішення конкретних проблем у реальному світі. Гарною ілюстрацією цього процесу є нинішній процес цифровізації багатьох послуг по всьому світу. Процес цифровізації торкається навіть державних послуг. Багато країн вже почало процес цифровізації державних послуг, серед них є і Україна. В Україні цим процесом займається міністерство цифрової трансформації України. Одним з їх проектів є створення додатку «Дія». «Дія» є мобільним застосунком та веб-сервісом, який допомагає зберігати електроні документи та виконувати деякі державні послуги.

Перед тим як перейти до аналізу сучасних проблем розробки програмного забезпечення, використовуючи приклади та аналізуючи статистику, треба розкрити

основні поняття щодо розробки програмного забезпечення для сучасних інформаційних систем.

По-перше, розробка програмного забезпечення є процесом дизайну, програмування, документування, тестування, розробки специфікації, створення прототипів, підтримки написаного програмного забезпечення та дослідження.

По-друге, через те що, процес розробки програмного забезпечення є комплексним завданням, люди створили спеціальні методології для менеджменту життєвого циклу розробки.

Етапи, які є частиною життєвого циклу розробки описані в стандарті ISO/IEC/IEEE 12207 Systems and software engineering – Software life cycle processes [2]. ISO/IEC/IEEE 12207 Systems and software engineering – Software life cycle processes є міжнародним стандартом, який описує життєві цикли розробки програмного забезпечення. Він був створений у 1995, наразі останньою версією є документ 2017 року. В ньому описані кожен етап процесу життєвого циклу розробки. Стандарт не наводить конкретну методологію розробки. Він вводить такі 2 поняття, як:

- Стадія;
- Етап.

Стандарт не описує ніяких конкретних стадій, він для цього використовує поняття процесу, бо розробники стандарту розуміють, стадії життєвого циклу розробки програмного забезпечення можуть відрізнятись.

Стандарт ділить процеси на чотири основні групи та багато підгруп, але вони не будуть наведені:

- Погодження;
- Організаційні проекти;
- Технічне управління;
- Технічні процеси.

Життєвий цикл розробки це певний набір кроків, які потрібні для декомпозиції всього процесу розробки. Ці кроки можуть бути різними, бо це

залежить від конкретної методології розробки.

Зазвичай життєвий цикл розробки можна розбити на 7 етапів:

1. Планування. На етапі планування створюються команди, розраховується ціна розробки, створюється бізнес план та визначається сфера та мета створення програмного забезпечення. Це потрібно для того, щоб чітко окреслити межі проекту та задати курс розробки.

2. Визначення вимог. Визначення вимог допомагає чітко окреслити конкретні функції, які потрібно розробити, щоб виконати поставлені задачі. Визначення вимог також допомагає зрозуміти, які ресурси можуть бути потрібні для виконання поставлених задач.

3. Дизайн та прототипування. На цьому етапі проектується архітектура системи. Визначаються бібліотеки та фреймворки, які можуть знадобитися, обираються протоколи та способи комунікації з іншими системами. Також на цьому етапі починають обирати механізми для забезпечення безпеки програмного забезпечення.

4. Розробка програмного забезпечення. На цьому етапі розробники пишуть код програмного забезпечення.

5. Тестування. На цьому етапі програмне забезпечення тестується, зазвичай тестування може виконуватись двома методами. Тестування це процес перевірки відповідності програмного забезпечення до поставлених вимог. Деякі з цих вимог можуть нести функцію забезпечення безпеки. Один з видів тестування, це автоматичне тестування, коли програмне забезпечення перевіряється за допомоги спеціальних скриптів. Також використовується мануальне тестування, коли людина перевіряє функціонал програмного забезпечення самостійно, запускаючи його та виконуючи певні тестові завдання.

6. Розгортання програмного забезпечення. Це може бути комплексним процесом, якщо програмне забезпечення складається з великої кількості залежностей та сервісів, які взаємодіють. Комплексна інфраструктура проекту може зробити цей процес більш швидким, якщо правильно використовувати

існуючи інструменти та абстракції.

7. Підтримка програмного забезпечення. Цей етап вже може не потребувати багато розробників, але він потрібен, бо після розробки програмного забезпечення можуть з'явитись нові помилки, які потрібно буде виправляти.

Зазвичай життєвий цикл розробки програмного забезпечення зображають простою схемою (Рис 1.1).

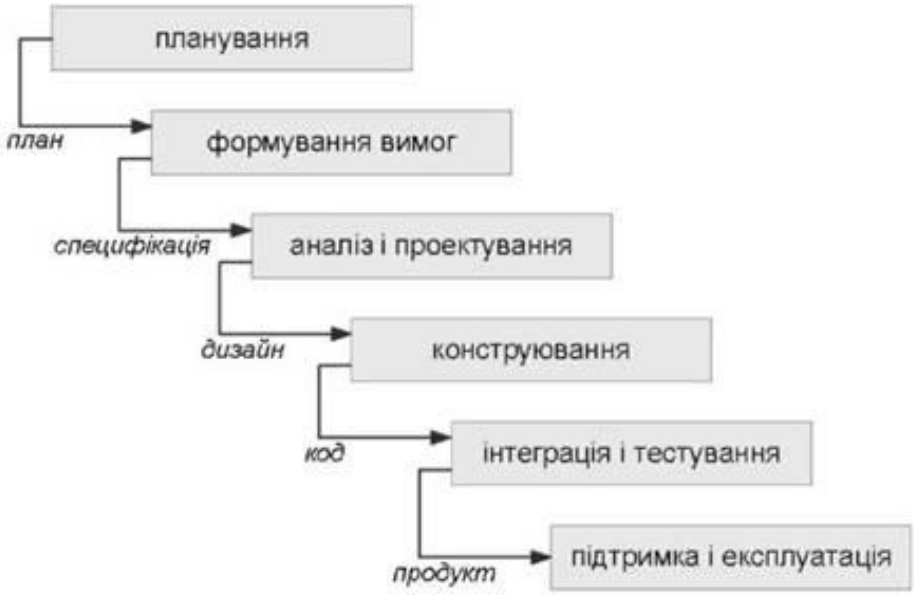

Рис. 1.1. Життєвий цикл розробки програмного забезпечення

Ці етапи розробки можуть відрізнятися, якщо використовується різна методологія розробки. Деякі з методологій можуть наслідувати інші, тим самим змінюючись під конкретні задачі. Серед найбільш значущих можна назвати такі, як:

- Waterfall модель;
- Agile модель;
- Spiral модель;
- DevOps модель.

Всі існуючі методології розробки програмного забезпечення мають свої переваги та недоліки. Кожну методологію треба обирати, опираючись на багато факторів, наприклад, тип сервісу, продукту, що розробляється, його аудиторія.

Модель розробки Waterfall є однією з найстаріших методологій розробки програмного забезпечення, що навіть в наш час користується деякою популярністю. Її суть полягає в тому, щоб розбити проект на декілька фаз, де кожна фаза залежить від результатів фази, що була до неї.

Модель розробки Waterfall характеризується, як одна з найбільш прямих методологій життєвого циклу розробки, бо прогрес йде зверху вниз, не повторюючи етапи, якщо проект змінюється та з'являються нові вхідні дані. Розробник моделі вчений Вінстон Ройс, отримавши відгуки від інших вчених , створив потім фінальну версію моделі, яка була більш гнучкою та дозволяла повертатися до етапів, які вже були пройдені (Рис1.2.).

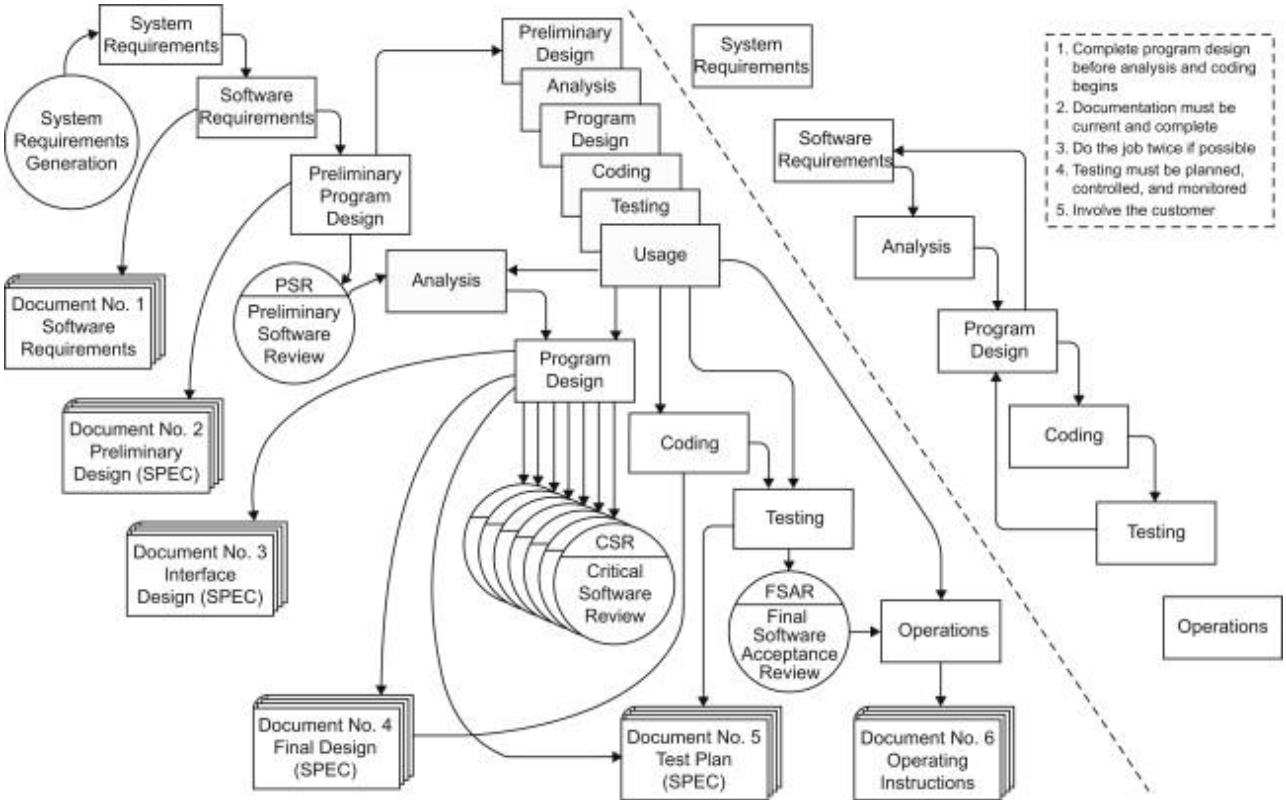

Рис. 1.2. Доповнена модель Ройса Вінстона

Доповнена модель Ройса вже почала використовувати щось схоже на ітеративний підхід у розробці, чим надихнула інших вчених на створення більш гнучких моделей [3].

Однією з перших ітеративних моделей стала Spiral модель. Spiral модель була створена вченим Баррі Боемом у 1988 в його роботі «A Spiral Model of Software Development and Enhancement». Spiral модель є ітеративною методологією життєвого циклу розробки програмного забезпечення. Ця модель поєднує методи управління ризиками та деякі елементи Waterfall моделі.

Зазвичай вона зображується, як спіраль (Рис. 1.3.). Кожен квадрант спіралі відповідає за певний етап розробки. А кожен етап розробки розбивається на менші під етапи.

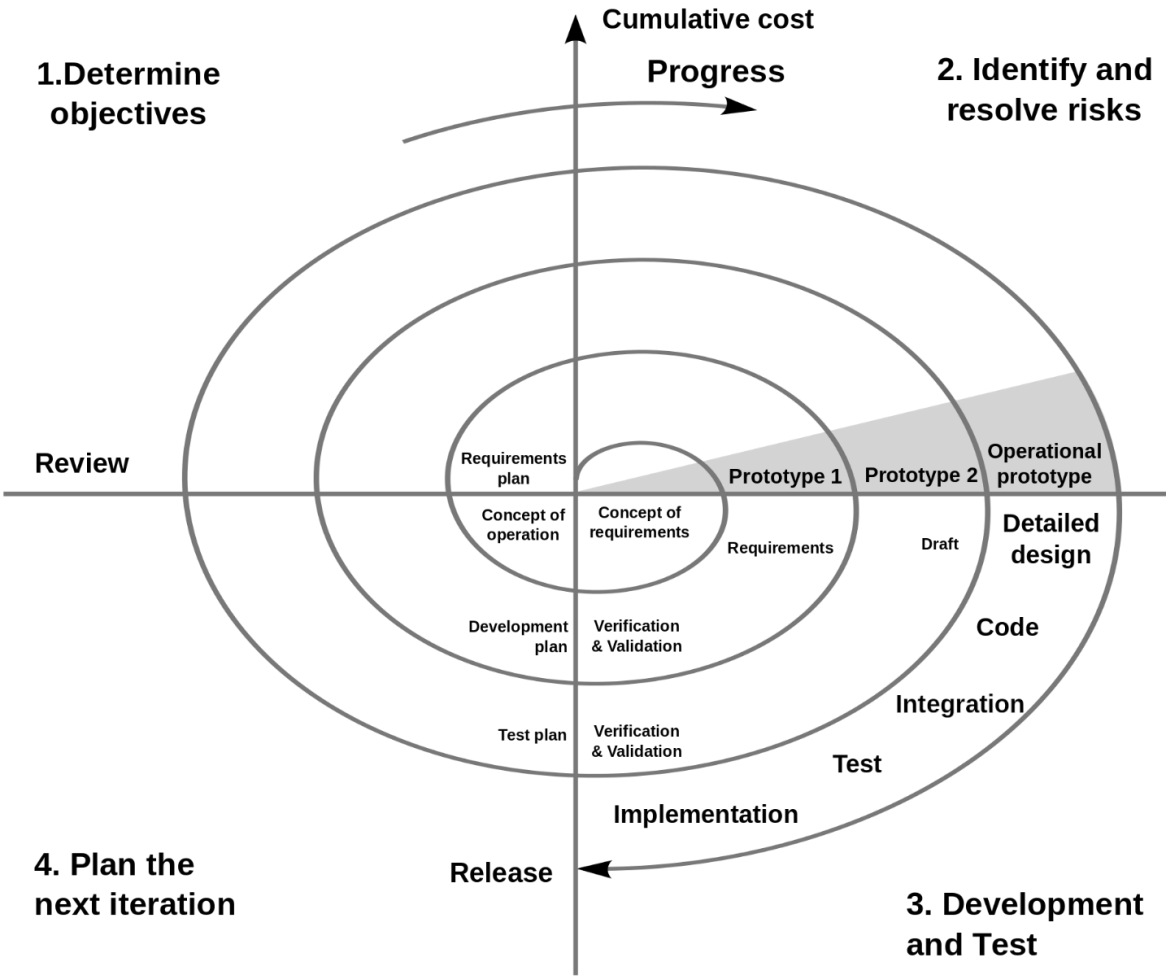

Рис. 1.3. Spiral модель

Кожний етап цієї моделі включає в себе аналіз ризиків, якщо ризиків немає, то серія етапів перетворюється в Waterfall модель.

Серед переваг цієї моделі можна назвати гнучкість, що вигідно її відрізняє від Waterfall моделі та спроможність зробити декомпозицію тяжких та великих проектів. Також Spiral модель включає оцінку ризиків на кожному із своїх етапів, тим самим покращуючи рівень безпеки проекту [4].

Серед недоліків цієї моделі, можна назвати:

1. Вона потребує багато ресурсів;
2. Є досить складною методологією, що не позволяє оцінити час проекту;

Поява Spiral моделі показала, що комплексні та високо ризикові проекти неможливо розробляти не ітеративно, не створюючи прототипи.

Разом зі зміною методологій розробки, змінювався і світ навколо. Бізнес не міг довго розробляти один продукт, використовуючи такі комплексні підходи, як Spiral модель, чи такі не гнучкі, як модель Waterfall. Змінилась аудиторія користувачів програмного забезпечення. Поширювались користувачі персональних комп'ютерів та виникала задача створення великої кількості програмного забезпечення, яке б вирішувало їх проблеми. Таким чином розробники перейшли від громіздких методологій розробки до більш простих та гнучких.

Такою методологією є Agile, який був створений у 2001 році, силами 17 розробників, які написали Agile маніфест. Agile об'єднав вже відомі підходи до ітеративної розробки та ідею організації функціональних, самостійних команд, які можуть швидко адаптуватись. В Agile маніфесті описали основні цінності розробки

- Індивіди та їх взаємодія важніше за процеси та інструменти;
- Працююче програмне забезпечення важніше за повну документацію;
- Співпраця з клієнтом важливіша за переговори щодо контракту;
- Реагування на зміни, а не сліпе виконування плану.

Agile модель базується на 12 принципах:

1. Задоволення клієнтів за рахунок ранньої та постійної доставки цінного програмного забезпечення;
2. Вітаємо зміни вимог, навіть на пізніх стадіях розробки;

3. Поставляйте працююче програмне забезпечення часто (тижні, а не місяці);

4. Тісна щоденна співпраця між бізнесменами та розробниками;

5. Проекти будуються навколо мотивованих людей, яким слід довіряти;

6. Розмова віч-на-віч – найкраща форма спілкування (сумісне розташування);

7. Працююче програмне забезпечення є основним показником прогресу;

8. Сталий розвиток, здатний підтримувати постійний темп;

9. Постійна увага до технічної досконалості та гарного дизайну;

10. Простота це мистецтво максимізації обсягу невиконаної роботи і це має важливе значення;

11. Найкращі архітектури, вимоги та дизайн виходять із самоорганізуючих команд;

12. Регулярно команда розмірковує, як стати ефективнішою, і відповідно коригує.

Agile модель стала однією з найбільш популярних у розробці программного забезпечення. Це можна побачити у досліджені StackOverFlow 2018 року [5].

Таблиця 1.1.

Популярність методологій у розробників

Методологія	%
Agile	85.4
Scrum	62.7
Kanban	35.2
Extreme programming (XP)	15.7
Waterfall	15.1
PRINCE2	1.5

Agile модель була створена, щоб перевести фокус розробників на потреби клієнтів. Agile дозволяє швидко розробляти новий фунціонал та реагувати на зміни, стрімко випускаючи нові цикли розробки.

Останньою моделюю є DevOps. Вона не протиставляється усім іншим, що були розібрані вище, а доповнює їх. DevOps модель це набір з процесів та практик, які використовуються розробниками та командою інфраструктури. Ця модель впливає на зв'язки та комунікації між різними командами, такими, як:

- Розробники;
- Тестувальники;
- Команда інфраструктури.

Основною роллю моделі DevOps є автоматизація життєвого циклу розробки. Це впливає на швидкість розробки та виходу сервісу до поточної аудиторії.

Розібравши всі ці моделі, можна простежити, що вони розвивались, опираючись на багато факторів. Одним за найважливіших є конкуренція бізнес продуктів. Зараз від розробки програмного забезпечення очікують, що процес буде гнучким, швидким та може швидко змінюватись, доповнюватись та тестуватись під потреби клієнтів та бізнесу. Тому зараз більшість компаній починають застосовувати Agile модель.

На популярність DevOps моделі впливають ще такі речі, як:

- Еволюція в хмарових продуктах. Використання провайдерів хмарових послуг сильно впливає на поширення DevOps практик;
- Потреба бізнесу швидко випускати продукт;
- Поширення мікро сервісної архітектури.

1.2. Аналіз проблеми забезпечення захисту програмного забезпечення для сучасних інформаційних системи.

Разом з еволюцією підходів та методологій життєвого циклу розробки, змінювався підхід до безпеки програмного забезпечення, що розроблялося. Для того, щоб відповісти на питання, як забезпечити захист програмного забезпечення для сучасних інформаційних систем, треба спочатку провести аналіз того, як розвилася індустрія, інструменти і підходи до безпеки.

Сьогодні суспільство та світ розраховує та залежить від програмного забезпечення. Але разом з ростом залежності від програмного забезпечення, виріс і рівень проблем з кібербезпекою.

В українському законодавстві під кібербезпекою розуміють захищеність життєво важливих інтересів людини і громадянина, суспільства та держави під час використання кіберпростору, за якої забезпечуються сталий розвиток інформаційного суспільства та цифрового комунікативного середовища, своєчасне виявлення, запобігання і нейтралізація реальних і потенційних загроз національній безпеці України у кіберпросторі [6].

Internet Crime Complaint Center це американський державний сервіс, який має статистику кіберзлочинів. З 2000 по 2020 роки на Internet Crime Complaint Center надходили скарги на інтернет-злочини (Рис. 1.4). Можна побачити тренд зростання надходження цих скарг.

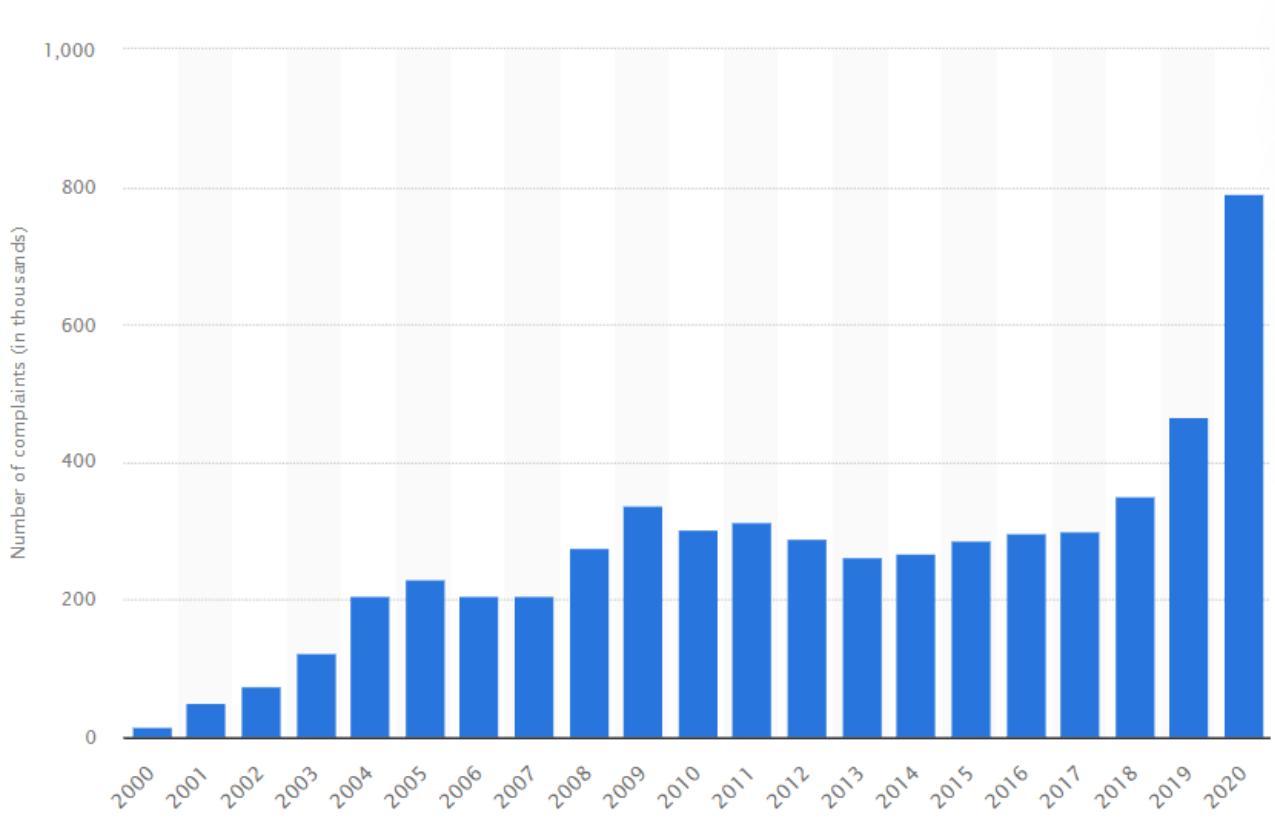

Рис. 1.4. Скарги на інтернет-злочини

Також є статистика на тему - сума грошової шкоди, завданої кіберзлочинами з 2001 по 2020 роки (Рис. 1.5.). Ця статистика також зібрана Internet Crime Complaint Center. Цей тренд також зростаючий [7]. На діаграмі можна побачити, що у 2020 році, сума грошової шкоди від кіберзлочинів взяла свій рекорд у чотири трильйони доларів.

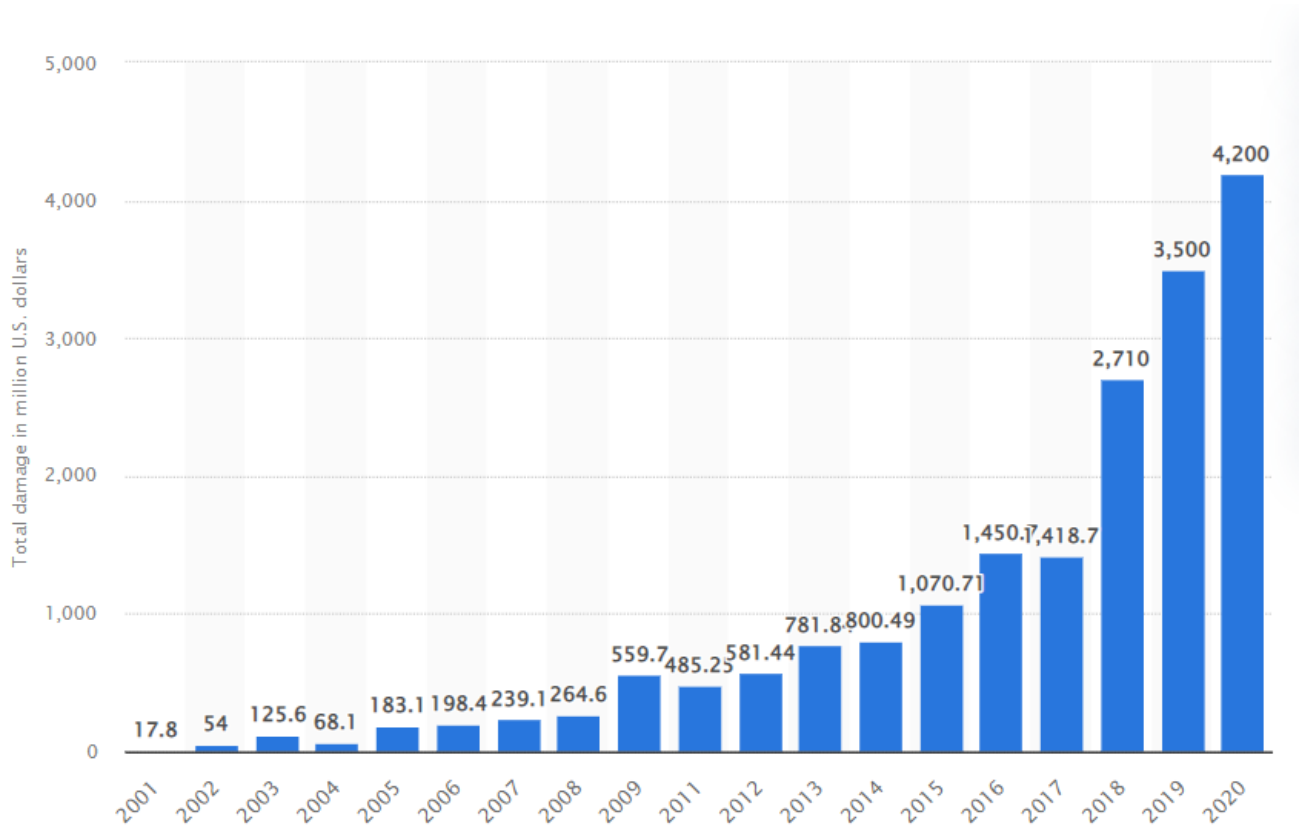

Рис. 1.5. Сума грошової шкоди

Останньою важливою статистикою, яку потрібно привести, є річна кількість порушень даних і викритих записів у Сполучених Штатах з 2005 по 2020 рік (Рис. 1.6.). Дивлячись на статистику, можна сказати, що присутній тренд зростання кількості порушень даних та викритих записів. Порушення безпеки даних – це порушення безпеки, при якому конфіденційні, захищені або конфіденційні дані копіюються, передаються, переглядаються, викрадаються або використовуються особою, яка не має на це права.

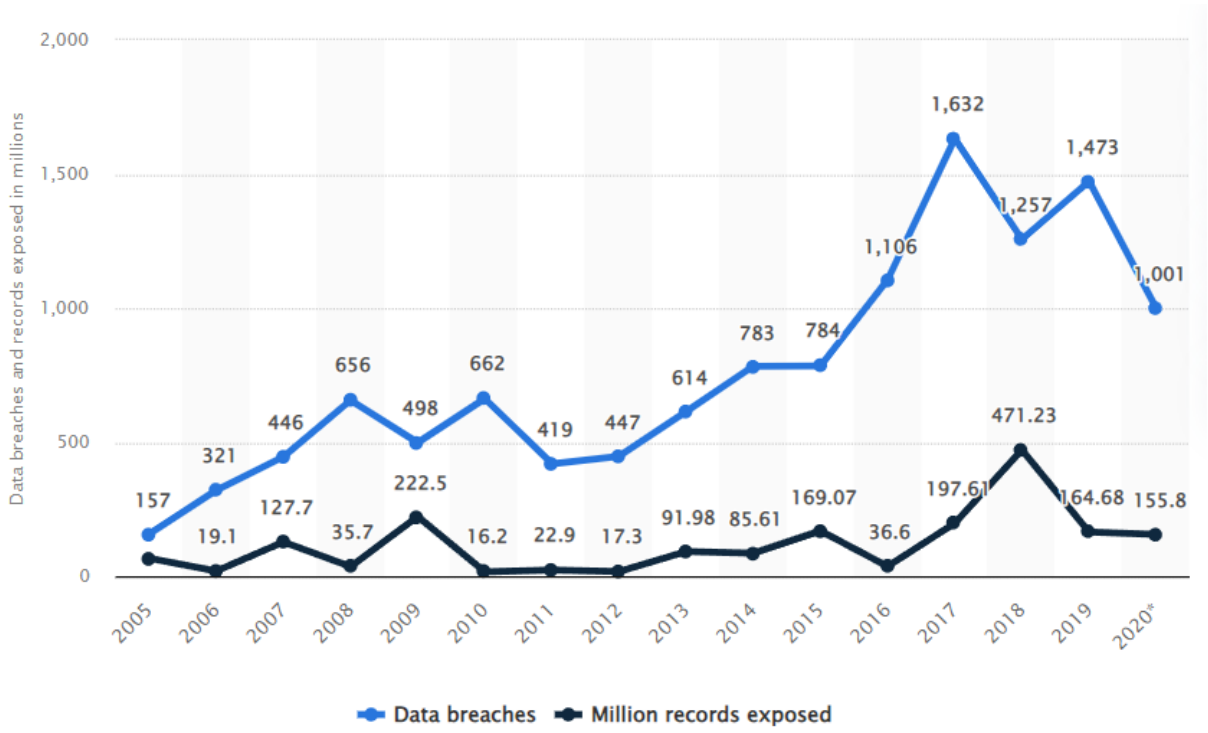

Рис. 1.6. Річна кількість порушень даних і викритих записів у США

Проаналізувавши статистику та зростаючу тенденцію кіберзлочинів, можна зробити висновок, що питання кібербезпеки є актуальною проблемою сьогодення. Також може з'явитись питання, чому в нас зараз так багато проблем з кібербезпекою. Для цього треба проаналізувати історію розвитку.

У 1975 році Джеррі Сальцер та Майкл Шродер публікують свою працю The Protection of Information in Computer Systems. Вони розкривають такі поняття, як створення безпечного механізму аутентифікації та механізмів доступу. Формують базові принципи, які потрібні для забезпечення безпеки програмного забезпечення.

Роберт Моріс та Кен Томпсон у 1978 році створюють політику менеджменту паролів, яка все ще впливає на політики менеджменту паролів сьогодення.

Джеймс Андерсон у 1980 році пише наукову роботу Computer Security Threat Modeling and Surveillance. В ній він розкриває поняття логування для спеціалістів з кібербезпеки.

У 1985 році Міністерство оборони США публікує «Помаранчеву книгу», де описано різні рівні безпеки та критерії кожного.

Дороті Денінг та Пітер Нейман у 1986 році починають розробку першої системи виявлення вторгнень, яка працює на базі аналізу аномальних явищ.

У 1989 році державний сектор США почав розробку системи Haystack. Це була удосконалена система виявлення вторгнень, яка працювала не лише на рівні хосту, а й на мережевому.

У 1987 році з'являється мова розробки Perl. Вона була розроблена Ларі Волом та на довгий час стала однією з найпопулярніших мов системних адміністраторів.

У 1988 році з'являться хробак Морріса, який є одним за найперших мережевих хробаків. Це вплинуло на системний дизайн Linux та появлення пауз при невдалій аутентифікації. Історія з Моррісом вплинула на розробку нових механізмів забезпечення безпеки та створення нових інструментів для цього.

У 1989-1992 роках з'являється багато профінансованих урядом проектів у сфері кібербезпеки, наприклад, MIDAS, NADIR і ASIM. Починають з'являтися комерційні пакети програмного забезпечення, що пропонують автоматичне скорочення слідів аудиту, аналіз і виявлення аномалій, наприклад, ISOA , Clyde Vax Audit, Stalker, CMDS і NetRanger. Перший комерційний пакет брандмауера, написаний Маркусом Ранумом, був випущений як DEC SEAL у 1990 році.

Тім Бернерс-Лі 6 серпня 1991 року публікує документ, де описує проект всесвітньої павутини. Разом з ним анонсується протокол HTTP, який є протоколом, що працює над TCP. І хоча протокол TCP зберігає свій стан, HTTP ні. Це стало проблемою безпеки, яку навіть зараз ще не вирішили. Прикладом можуть послугувати нескінчені проблеми безпеки веб додатків, що пов'язані з контролем сесії.

Філ Цімерман у 1992 році випускає PGP (Pretty Good Privacy). Pretty Good Privacy (PGP) це програма шифрування, яка забезпечує криптографічну конфіденційність та аутентифікацію для передачі даних. PGP використовується для підписання, шифрування та дешифрування текстів, електронних листів, файлів,

каталогів і цілих розділів диска, а також для підвищення безпеки зв'язку електронною поштою.

У 1993 році Марк Андерсон випускає перший графічний веб браузер Mosaic. Компанії почали розміщувати рекламу своїх товарів та послуг у Інтернеті. Ще не виникли інструменти та підходи для розміщення в Інтернеті цілих бізнес процесів, але перехід вже почався.

1993–1995: Роб МакКул пише і випускає веб-сервер NCSA httpd; До httpd додано функціональність CGI (Common Gateway Interface). Вільна група Apache Group випускає веб-сервер Apache (який використовує httpd 1.3 NCSA як основу). У специфікації HTML 2.0.23 з’являються перші форми для вводу даних. Більш-менш стандартизований спосіб обслуговування динамічного контенту з’явився.

У 1995 Sun Microsystems випускає вихідний код Java (версія 1.0a2, «повна загальнодоступна альфа-версія») в Інтернет. Десятки тисяч копій завантажуються розробниками за кілька місяців. Технологія Java вплетена в новий потужний веб-браузер Netscape Navigator.

Ден Фармер і Вітсе Венема у 1995 написали та випустили суперечливий сканер безпеки SATAN. Ефективний у пошуку відкритих дір у безпеці, і водночас дуже портативний і з інтерфейсом веб браузера, SATAN був свого роду точкою опори. Він об’єднав елементи, які до цього були різнорідними: безпека мережі, вразливості системного програмного забезпечення, корпоративне прикладне програмне забезпечення та використання нових гіперпортативних веб-платформ (PERL, CGI та HTML).

Гарі Макгроу та Едвард Фелтон у 1996 написали і випустили першу книгу з безпеки Java для користувачів, щоб створити комплексну стратегію використання Java, яка враховує ризики програмного забезпечення.

Гордон Ліон у 1997 випускає NMAP, сканер мережевої безпеки, який використовує виявлення хостів і сканування портів для пошуку вразливостей мережі. Цей сканер використовується навіть по сей день.

У 1998 Мартін Реш випускає SNORT, програму перевірку пакетів, яка перетворилася на систему запобігання вторгненню в мережу, а Ренауд Дерайсон випускає Nessus як безкоштовний віддалений сканер безпеки.

Проаналізувавши історію розвитку індустрії, можна сказати, що розробка програмного забезпечення, Інтернет, протоколи та стандарти, інструменти кібербезпеки розвивались поступово.

Через бум дот нетів, швидкий розвиток Інтернету, інфраструктура та стандарти не поспішали за питаннями безпеки. Тому, наприклад, можна сказати про протокол HTTP, який не зберігає стан сесії, цим самим створюючи багато проблем з безпекою, які вже потім вирішувались створенням нових підходів, інструментів та протоколів, наприклад, створення cookies у браузері, які схильні до великої кількості атак типу CSRF. Наступним можна згадати HTTPS, який тільки зараз стає популярнішим за HTTP. Захищений протокол передачі гіпертексту (HTTPS) є розширенням протоколу передачі гіпертексту (HTTP). Він використовується для безпечного зв'язку через комп'ютерну мережу, і широко використовується в Інтернеті. У HTTPS протокол зв’язку шифрується за допомогою безпеки транспортного рівня (TLS) або, раніше, за допомоги SSL.

Те саме з інструментами забезпечення кібербезпеки по типу систем виявлення вторгнень та брандмауерів. Вони працюють та еволюціонують реактивно та відстають від розвитку індустрії, бо спочатку змінюється розробка та індустрія, а потом з’являються проблеми кібербезпеки і останніми з’являються рішення та інструменти для вирішення цих проблем.

Зараз, коли, найбільш популярними методологіями розробки є Agile та DevOps, спеціалісти з кібербезпеки повинні впроваджувати нові процеси та інструменти до життєвого циклу розробки, який є основним процесом в розробці програмного забезпечення.

Багато спеціалістів з кібербезпеки не знайомі з цими підходами Agile та DevOps, це призводить до того, що команди з кібербезпеки у компаніях не працюють разом з командами розробників, тестувальників та командою

інфраструктури разом. Розділяючи ці команди ми втрачаємо певні переваги роботи разом.

Можна назвати такі проблеми, як:

- Розробники фокусуються на нових функціях програмного забезпечення, а не проблемах безпеки;
- Індустрія не вчиться на своїх помилках;
- Розробники не мають достатніх знань в сфері кібербезпеки;
- Комплексна архітектура та багато слоїв абстракцій у нинішньому програмному забезпечені. Прикладом можуть послугувати мікро сервіси та Kubernetes;
- Програмне забезпечення дуже взаємопов'язане [8]. Розробники за замовчуванням довірять багатьом речам.

Прикладом проблем можна назвати довіру розробників до відкритого програмного забезпечення. У 2021 році сталась така проблема, як зараження одного з пакетів розробки JS вірусом. Цю бібліотеку використовували мільйони розробників, за неділю проходило зазвичай 6-7 мільйонів скачувань бібліотеки. Використовування цієї бібліотеки скомпрометувало мільйони розробників та їх продуктів [9].

Прикладом того, що індустрія не вчиться на своїх помилках, можна назвати дослідження Джеймса Кетла о вразливості HTTP Smuggling. Smuggling HTTP-запитів це метод перешкоджання тому, як веб-сайт обробляє послідовності запитів HTTP, отримані від одного або кількох користувачів. Ця вразливість часто має критичний характер, дозволяючи зловмиснику обійти контроль безпеки, отримати несанкціонований доступ до конфіденційних даних і безпосередньо скомпрометувати інших користувачів програми. Ця вразливість вже була досліджена та розібрана до нього, але через те що, розробники та спеціалісти з кібербезпеки не співпрацюють достатньо тісно, при еволюції архітектури веб додатків, ніхто не звернув уваги на цю проблему. Широке використання серій різних проксі у архітектурі, посприяло поширенню цієї вразливості.

1.3. Аналіз існуючих підходів до захисту процесу розробки та самої інформаційної системи.

Вже давно спеціалісти з кібербезпеки вирішують проблеми безпеки, завдяки брандмауерам, системам виявлення вторгнень, SIEM, сканерам безпеки та багатьом іншим інструментам та процесам. Це все не є поганим, але не адресує усі проблеми, що були перелічені. Для того, щоб ефективно захищати нинішні компанії, що займаються розробкою програмного забезпечення, треба, щоб відбувалась більш тісна співпраця між командою безпеки та розробниками, адміністраторами та інженерами.

Для того, щоб адресувати деякі з проблем, що були описані в попередньому підрозділі, а точніше:

- Поганий стан знань розробників щодо кібербезпеки;
- Комплексна архітектура та багато слоїв абстракцій у нинішньому програмному забезпечені.

Було розроблено такі речі, як OWASP SAMM та OWASP Top 10. Хоча OWASP Top 10 адресований до проблем безпеки веб додатків, але концептуально ці проблеми поширюються і на інші платформи. Також існують схожі документи для розробки на мобільних платформах типу IOS, Android.

OWASP Top 10 це документ для розробників та спеціалістів з кібербезпеки веб-додатків. Він являє собою консенсус щодо найбільш критичних ризиків безпеки для веб-додатків [10]. Серед цих ризиків можна назвати такі:

1. Порушений контроль доступу;
2. Проблеми з використанням криптографії;
3. Ін’єкції;
4. Небезпечний дизайн;
5. Неправильна конфігурація безпеки;
6. Уразливі та застарілі компоненти;

7. Помилки ідентифікації та аутентифікації;
8. Збій цілісності програмного забезпечення та даних;
9. Проблеми журналювання та моніторингу;
10. Server-Side Request Forgery (SSRF).

Ці ризики можуть змінюватись періодично так як і їх популярність та місце в топ десять. Їх роль не тільки принести до розробників поняття про найпопулярніші проблеми в їх сфері з боку безпеки, але й допомогти організаціям поставити пріоритети до потенційно проблемних місць у програмного забезпечені.

OWASP SAMM це методологія OWASP, яка допомагає організаціям оцінити, сформулювати та впровадити стратегію безпеки програмного забезпечення, яку можна інтегрувати в їхній існуючий життєвий цикл розробки програмного забезпечення . OWASP SAMM підходить для більшості проектів, незалежно від того, яку модель розробки використовує організація, метод Agile, Waterfall або DevOps.

OWASP SAMM складається з п’ятнадцяти практик та методів по забезпеченню кібербезпеки в організації. Усі ці практики та методи поділені на п’ять різних категорій. Кожен метод чи практика містить три різних рівня зрілості. Дії на нижчому рівні зрілості, як правило, легші у виконанні та потребують меншої формалізації, їх треба виконувати спочатку. Ріні, що мають більш високий рівень зрілості більш тяжкі у виконані та впроваджені.

Документ OWASP SAMM створений спеціально так, щоб допомогти з такими речами:

- Оцінка конкретного рівня захисту програмного забезпечення у організації;
- Визначення стратегії, яку організація має реалізувати;
- Формулювання дорожньої карти впровадження, як туди дістатися;
- Рекомендаційні поради щодо того, як виконувати певні дії.

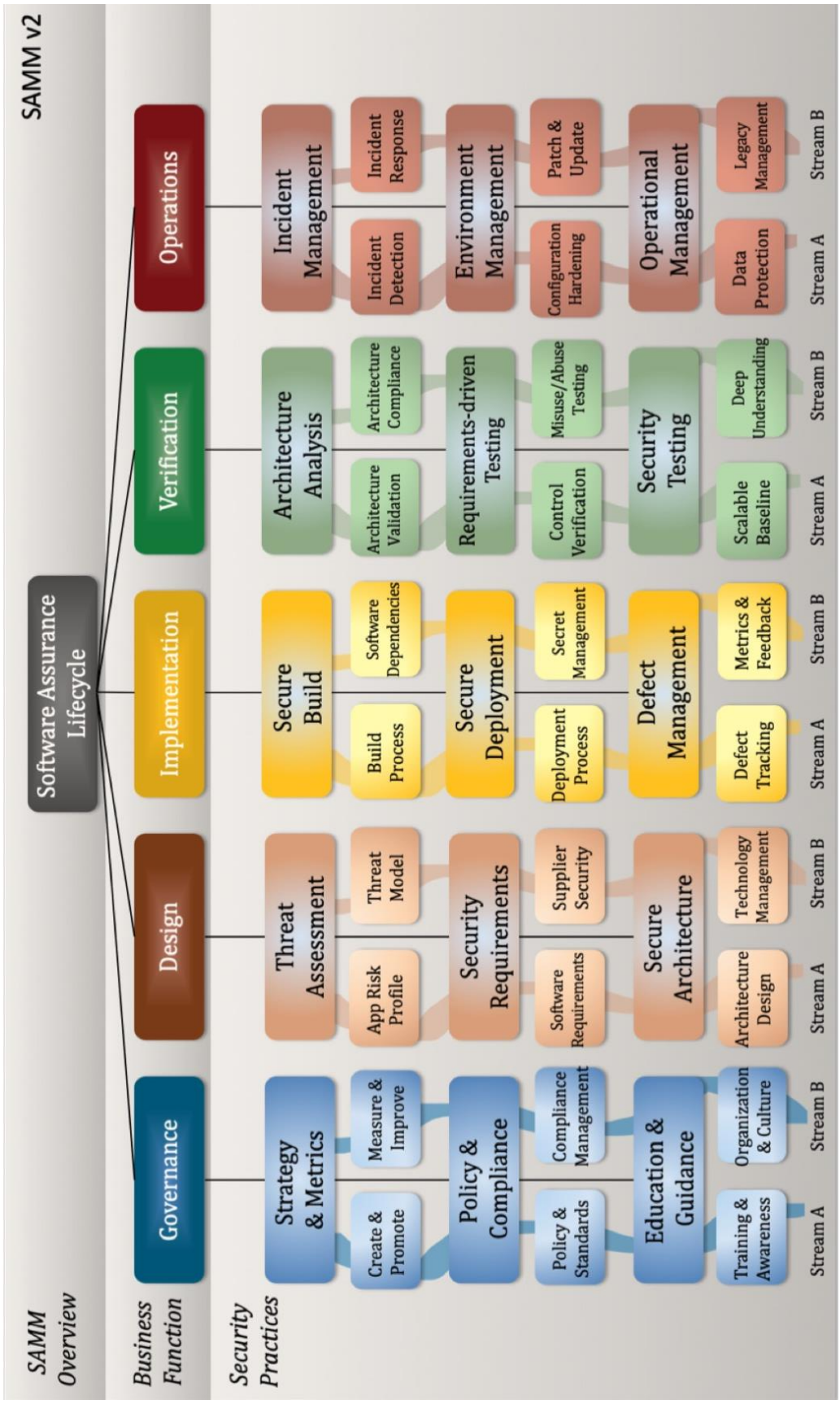

Рис. 1.7. OWASP SAMM модель

Задля впровадження практик з OWASP SAMM, треба налагодити співпрацю між командою розробки та безпеки. Треба зрозуміти, що безпека це не продукт, а процес, який має бути безперервним та повинен поступово покращуватись, відповідаючи на останні проблеми та ризики.

Повертаючись до теми існуючих технологій та методів захисту процесу розробки та інформаційної системи, треба сказати, що у сучасних компаніях, які займаються розробкою, безпека повинна пронизувати усі етапи життєвого циклу розробки програмного забезпечення. Через те, що наразі популярні DevOps та Agile підходи, безпека повинна вміти бути включена в ці моделі та користуватися їх перевагами, а саме гнучкістю та автоматизацією багатьох задач. Далі я розповім про процеси та методи розробки безпечного програмного забезпечення для сучасних інформаційних систем

2 АНАЛІЗ ПРОЦЕСУ ТА МЕТОДІВ РОЗРОБКИ БЕЗПЕЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМ

2.1. Методи та засоби захисту веб додатків в DevOps середовищі

Безпека веб додатків є цілою спеціалізацію та окремою сферою розвитку спеціаліста з кібербезпеки. Спеціалісти з безпеки веб додатків займаються пошуком вразливостей в компонентах веб додатку та роблять рекомендації по їх усуненню.

Зазвичай веб додаток складається з декількох компонентів:

- Frontend частини, що написана на JavaScript;
- Backend частини, що може бути написана на великій кількості мов програмування, зазвичай це Python, Java, PHP, Go, NodeJS;
- Database частини.

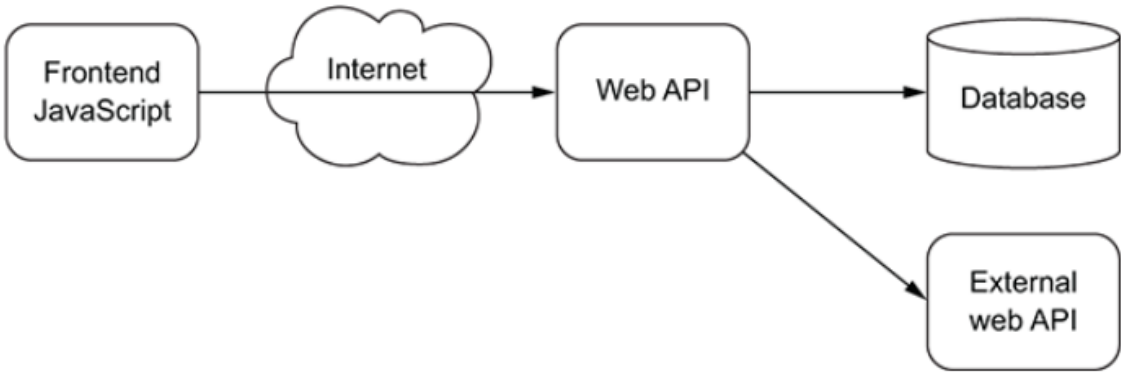

Рис. 2.1. Спрощена модель архітектури веб додатку

Кожна з цих частин може мати свої специфічні вразливості. Зазвичай команди спеціалістів використовують такі сканери, як Nessus, Qualys та Netsparker для пошуку вразливостей в веб додатках. Вони запускають ці сканери своїми руками, з визначеним інтервалом, чи при релізі програмного забезпечення.

При використані DevOps моделі цей процес можна покращити, зробивши його автоматичним та постійним. Тим самим покращити покриття та безпеку розроблюваного сервісу.

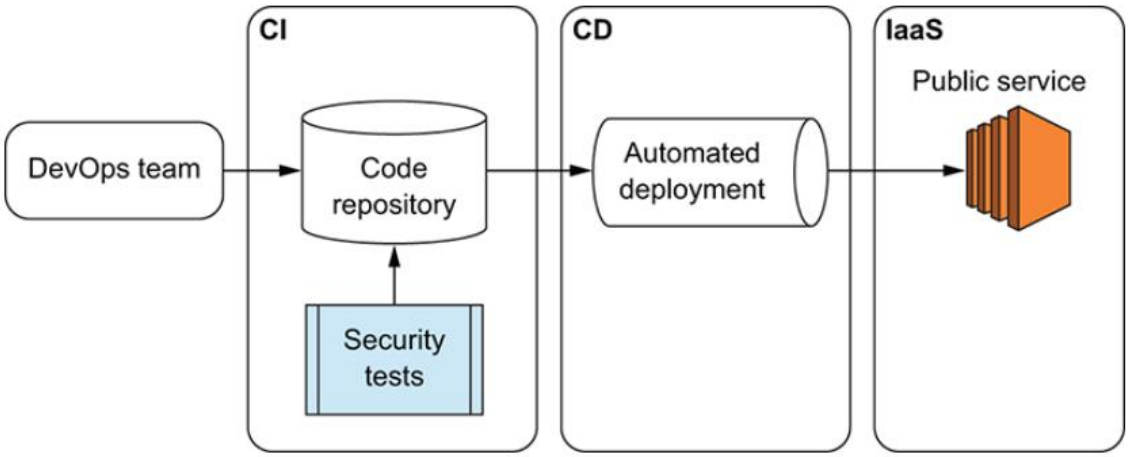

Рис. 2.2. Схема роботи сканеру у DevOps моделі

На етапі CI тести можуть бути такі:

- Статичне тестування безпеки додатків (SAST), або статичний аналіз це методологія тестування, яка аналізує вихідний код для пошуку вразливостей безпеки, які роблять програми вашої організації сприйнятливими до атак. SAST сканує програму перед компіляцією коду.
- Динамічне тестування безпеки програми (DAST) це метод тестування, коли сканер відправляє до програми шкідливі запити під час її роботи, не знаючи внутрішніх взаємодій або дизайну програми.

Серед найбільш популярних вразливостей з останнього документу OWASP Top 10, можна назвати такі:

- Порушений контроль доступу;
- Проблеми з використанням криптографії;
- Ін’єкції.

Контроль доступу відповідає за певний набір дозволів, які доступні для користувача. Якщо його дії виходять за цей набір дозволів, це стає порушенням

контролю доступу. Згідно з OWASP Top 10, це найбільш популярна вразливість зараз. Зазвичай проблеми з контролем доступу включають до себе:

- Порушення принципу мінімальних привілеїв;
- Обхід контролів за допомоги зміни параметрів;
- IDOR
- Відсутність керування доступом до різних HTTP методів;
- Підвищення привілеїв;
- Зловживання проблемами у JWT дизайні;
- Погана конфігурація CORS;

Наступною популярною проблемою є проблеми при використанні криптографії. Є певний перелік інформації, яка повинна бути захищена більш ретельно. Це, наприклад, фінансові дані. Інформація про це написана в стандарті Стандарт безпеки даних PCI (PCI DSS). Зазвичай ці проблеми включають до себе такі речі:

- Передача незашифрованої інформації через протокол HTTP;
- Використання алгоритмів криптографії з відомими вразливостями;
- Поганий процес управління секретами та ключами;
- Погана перевірка сертифікатів;
- Зберігання ключів шифрування в небезпечному місці;
- Погані практики при використанні криптографічних алгоритмів та примітивів;
- Доступ до сторонніх каналів про інформацію щодо процесу шифрування чи дешифрування.

Останньою проблемою є ін’єкції. Ін’єкція це процес, коли вхідні дані користувача не перевіряються, не фільтруються та попадають в певний контекст, де можуть вплинути на роботу додатку. Деякі з найпоширеніших ін’єкцій це SQL, NoSQL, команди ОС, ORM, LDAP, OGNL. Перевірка коду веб додатку та fuzzing є способами перевірки веб додатку на ін’єкції. Цей тип вразливості раніше був

найбільш популярним, але його тепер легко знаходити, завдяки автоматичним сканерам, які можуть бути використані при розробці, як частина CI\CD процесу.

Окрім того, щоб тестувати веб додаток на вразливості, треба перевіряти усі залежності, бібліотеки, які можуть використовувати розробники у продукті. За цю задачу відповідають 2 процеси:

- Vendoring;
- Перевірка залежностей на відомі вразливості.

Vendoring це процес, який потрібен, щоб захисти проект від двох речей:

- Втрата доступності. Наприклад, коли місце, де розробник залежності її зберігав стає недоступним.
- Втрата цілісності. Трапляється, коли вихідний код можна замінити чимось шкідливим.

Для того, щоб захиститись від цих проблем, розробники використовують спеціальні репозиторії, де зберігають залежності. Наприклад Apache Maven. Apache Maven це інструмент для керування проектами програмного забезпечення. На основі концепції об'єктної моделі проекту (POM), Maven може керувати збіркою проекту, звітами та документацією з центральної частини інформації. Maven зберігає потрібні залежності в локальному репозиторії.

Наступним інструментом, який допомагає розробникам з менеджментом залежностей є SCA. SCA це інструмент для аналізу версій програмного забезпечення та перевірки на наявність у ньому відомих вразливостей.

Згідно з дослідженням The Unfortunate Reality of Insecure Libraries [11], зробленим Contrast Security:

- 25% організацій продовжують працювати без політики відкритого коду;
- З тих організацій, які мають політику щодо відкритого коду, лише 68% її дотримуються;
- Лише 22% організацій заборонили залежність, коли дізналися про проблему в ній;
- 75% не мають значущого контролю над тим, які компоненти є в їхніх

програмах;

Резюмуючи в моделі DevOps, себе добре показують такі контролі безпеки, як динамічні та статичні сканери безпеки та SCA для контролю вразливостей у залежностях. Вони запускаються автоматично, кожен раз, коли розробник додає зміни до репозиторія, як один з етапів життєвого циклу розробку.

2.2. Методи та засоби захисту інфраструктури хмари

Іншою великою темою та спеціалізацією в сфері кібербезпеки є забезпечення безпеки хмари та інфраструктури в цій хмарі. Зараз існує декілька великих провайдерів послуг хмарної інфраструктури, а саме AWS, Azure та GCP. Всі вони мають схожі абстракції та концепти на яких побудували свої сервіси. Для того, щоб розповісти про захист інфраструктури хмари, буде достатньо розібрати лише одного провайдера. На прикладі AWS можна описати всі процеси та методи по захисту інфраструктури хмари.

Краще всього це структурувати через концепцію багаторівневої безпеки, розбивши захист інфраструктури в хмарі на декілька доменів:

- Захист VPC;
- Управління доступом;
- Логування.

Перед тим як перейти до аналізи цих сфер, треба визначити, які загрози є в хмарному середовищі AWS. Далі наведена таблиця з сервісами чи компонентами AWS та загрозами, які їм притаманні.

Таблиця 2.1.

Загрози в AWS

Компонент	Загроза
IAM	IAM політика описує надмірно доступів. Доступ до KMS не заборонений. Дозволений Інтернет трафік до ELK. Доступ до політик безпеки та таблиць маршрутизації незаборонений. Використання застарілих AWS написаних політик.
MFA	MFA не активована.
Root акаунт	Не використовується електроні листи дистрибуції. MFA не активована.
Секрети	Ключі доступу не відповідають стандартам. Використання секретів, що видані за замовчуванням. Доступ до секретів погано налаштований. Секрети повторюються в декількох сервісах.
S3	Увімкнена глобальна функція видалення в S3. Увімкнена глобальна функція читання в S3. Доступ дозволений усім користувачам. Доступ дозволений усім. Відсутність реплікації даних. Дані не захищені від перезапису.
EBS	Відсутність політики бекапів. Увімкнений публічний доступ.
WAF	WAF не покриває усі потрібні сервіси. WAF має погано написані політики. WAF не блокує шкідливі запити. Відсутнє логування.

Продовження таблиці 2.1.

Загрози в AWS

Компонент	Загроза
DDoS	Відсутність захисту від DDoS.
Шифрування	Політики дозволяють HTTP трафік до CloudFront. Відсутність шифрування S3. Відсутність шифрування EBS. Політики дозволяють HTTP трафік між сервісами. Відсутність шифрування RDS.
Відповідність міжнародним стандартам	Відсутність перевірки PCI процесів. Відсутність перевірки відповідності стандартам.
Логування	Вимкнений CloudTrail. Вимкнений збір подій S3. Відсутність централізованої системи логування. Відсутність подій VPC. Відсутність маскування конфіденціальної інформації.
Групи безпеки	Відсутність заборон на трафік. Відсутність груп за замовчуванням. Дозвіл RDP/SSH публічним IP.
Реагування на інциденти	Відсутність сповіщень. Відсутність правил реагування на інциденти.
Управління вразливостями	Використовування небезпечних образів віртуальних машин. Відсутність процесу внесення змін в віртуальні машини. Відсутність процесу управління вразливостями в додатках та сервісах.

VPC це віртуальна приватна хмара, виділена для конкретного облікового запису. Вона логічно ізольована від інших віртуальних мереж у хмарі AWS. За безпеку VPC відповідають:

- Групи безпеки. Група безпеки діє як віртуальний брандмауер для вашої віртуальної машини для контролю вхідного та вихідного трафіку. Можна назначити до п’яти груп до одної віртуальної машини. Групи безпеки діють на рівні віртуальної машини, а не на рівні підмережі.
- Список контролю доступу до мережі. Список контролю доступу до мережі працює як брандмауер для контролю трафіка в підмережах.

Порівняння цих двох варіантів наведено в таблиці.

Таблиця 2.2.

Порівняння груп безпеки зі списками контролю

	Список контролю	Групи безпеки
Сфера роботи	Підмережа	Віртуальна машина
Тип правил	Дозвіл та відказ	Дозвіл
Сесія	Stateless	Stateful
Алгоритм роботи правил	Правила працюють згідно до номеру	Перевіряються всі правила перед тим як дати дозвіл
Правила	Правила додаються на кожну віртуальну машину	Правила додаються на кожну віртуальну машину у підмережі
Ціль	Щоб заборонити конкретний трафік	Щоб дозволити конкретний трафік

Схему де зображено, як працює списки контролю з групами безпеки можна побачити на рисунку. Трафік спочатку попадає до інтернет-шюзу, а потім направляється до потрібної мережі, взявши потрібну інформацію в таблиці маршрутизації. Потім правила списків контролю мережевого доступу перевіряють

чи може запит йти далі, якщо так, то потім його перевіряють групи безпеки, щоб допустити до віртуальної машини.

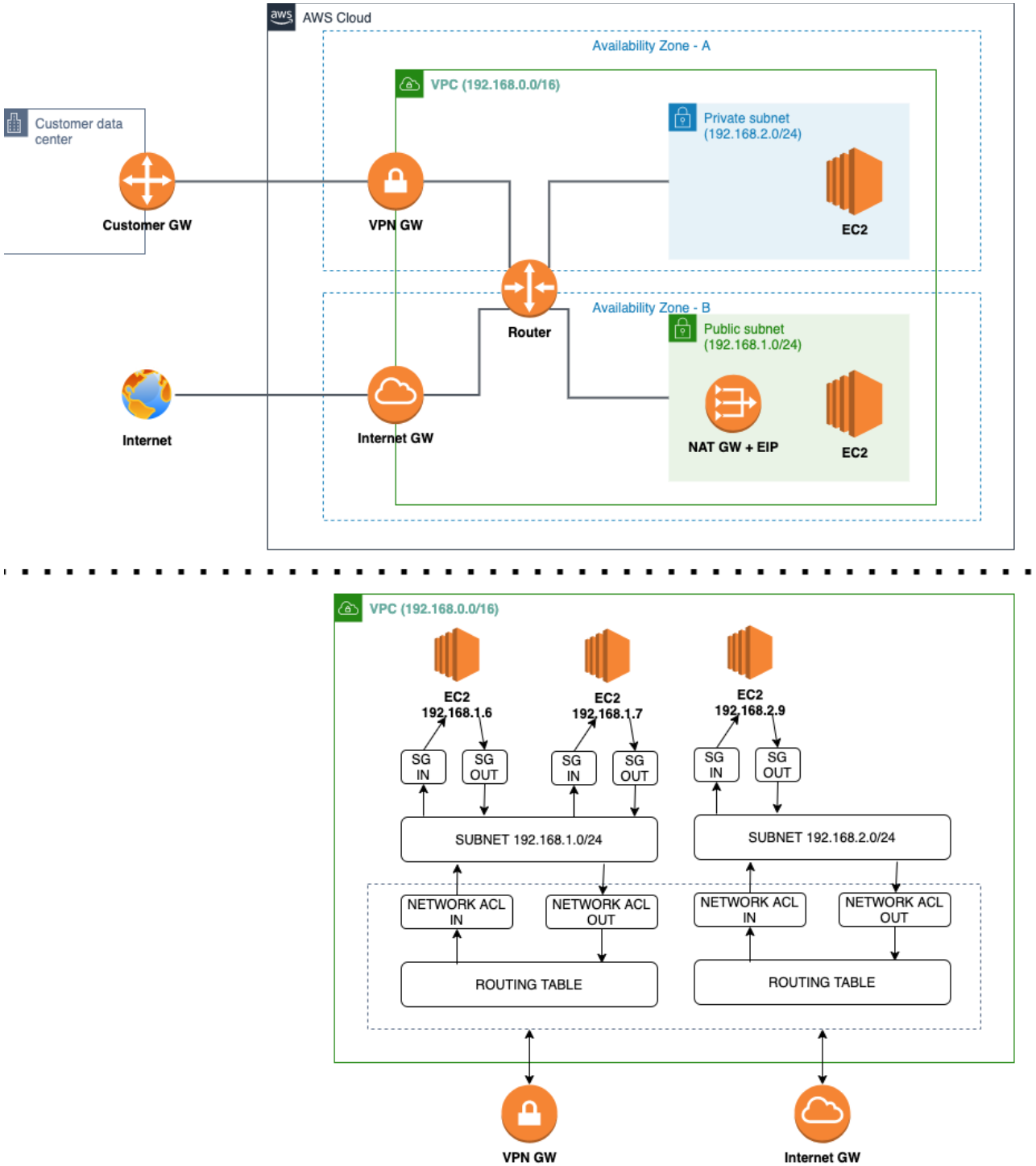

Рис. 2.3. Схема роботи брандмауера в VPC

Наступною темою є контроль доступу до ресурсів. В AWS він реалізований завдяки IAM. IAM (Управління ідентифікацією та доступом) це сервіс, що

відповідає за контроль доступу до ресурсів, він дозволяє регулювати, хто може пройти аутентифікацію та авторизацію.

Для того, щоб розібратися з роботою IAM, треба розкрити декілька термінів, які він використовує:

- IAM ресурси. Це можуть бути об'єкти користувача, групи, ролі, політики чи провайдера акаунтів.
- IAM особистості. Це об'єкти, які він використовує, щоб ідентифікувати та групувати.
- IAM сутності. Це IAM об'єкти ресурсів, які AWS використовує для аутентифікації. Він включає до себе IAM користувачів та ролі.
- Директори. Користувач або програма, яка використовує root-користувача облікового запису AWS, користувача IAM або роль IAM для входу та надсилання запитів до AWS. Можуть бути перманентними чи змінними.

Аутентифікація проходить у декілька способів. Їх можна побачити на таблиці нижче:

Таблиця 2.3.

Види аутентифікації

Тип аутентифікації	Опис
Логін\пароль	Користувачі
Ключ доступу	Додаток
Ключ доступу з токеном сесії	Процес, що виконується під назначеною роллю

Наступною частиною IAM є авторизація. Доступи до ресурсів пишуться в спеціальному документі в форматі JSON, цей документ є політикою і він і контролює авторизацію до ресурсу. Доступ в AWS керується, створюючи політики та приєднуючи їх до ідентифікаційних даних IAM (користувачів, груп користувачів або ролей) або ресурсів AWS. Політика це об'єкт в AWS, який, коли пов'язаний із ідентифікатором або ресурсом, визначає їхні дозволи. AWS оцінює ці політики, коли користувач або роль робить запит. Дозволи в політиках визначають, дозволено чи відхилено запит.

Приклад цього документу з політикою можна побачити на рисунку нижче. Ця політика дозволяє створення користувача, але лише з певними характеристиками.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TagUsersWithOnlyTheseTags",
            "Effect": "Allow",
            "Action": [
                "iam:CreateUser",
                "iam:TagUser"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIfExists": {
                    "aws:RequestTag/Department": [
                        "Development",
                        "QualityAssurance"
                    ],
                    "aws:RequestTag/JobFunction": "Employee"
                },
                "ForAllValues:StringEquals": {
                    "aws:TagKeys": [
                        "Department",
                        "JobFunction"
                    ]
                }
            }
        }
    ]
}
```

Рис. 2.4. Приклад політики доступу

Для того, щоб безпечно розробляти політики доступу, треба розуміти алгоритм їх роботи. Далі на рис. 2.5. можна побачити алгоритм роботи авторизації в AWS.

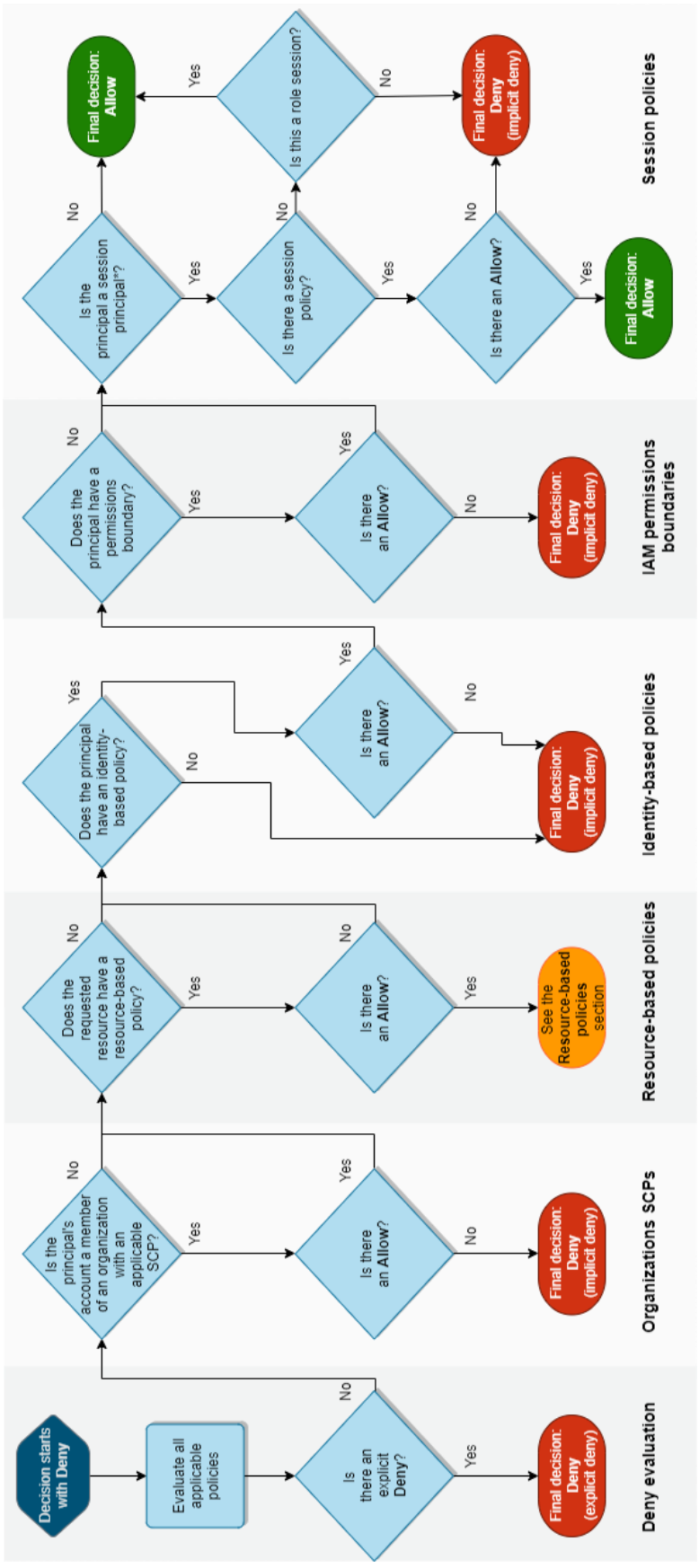

Рис. 2.5. Алгоритм роботи авторизації

AWS підтримує шість типів політик:

- політики на основі ідентифікації;
- політики на основі ресурсів;
- межі дозволів;
- SCP організацій;
- списки керування доступом;
- політики сеансів.

Останньою темою є логування та моніторинг. AWS має велику кількість сервісів, що зберігають логи. В таблиці нижче наведено їх опис та затримка, яка виникає з моменту від скоєння події до її появи в логах.

Таблиця 2.4.

Види сервісів та опис їх логування

Сервіс	Опис	Затримка	Місце зберіганння
CloudTrail	Користувачі	15 хвилин	S3
CloudWatch	Виклики API AWS	0	CloudWatch\SES
VPC	Мережа	15 хвилин	CloudWatch
S3	Доступ до S3	1 година	S3
ELB	Веб запити	5 хвилин	S3
CloudFront	Кеш запити	До 24 годин	S3

Логування є важливою темою у захисті інфраструктури та веб додатків. Воно дозволяє команді спеціалістів з кібербезпеки реагувати на інциденти та робити аналіз інцидентів, що вже відбулись.

Через велику кількість різних файлів, що потрібно зберігати та великої кількості подій, з’являються нові підходи до побудови логування в компаніях. Це достатньо великий процес, який поділений на декілька етапів, кожний з яких виконує певну функцію. На рис. 2.6. можна побачити етапи логування в сучасних середовищах.

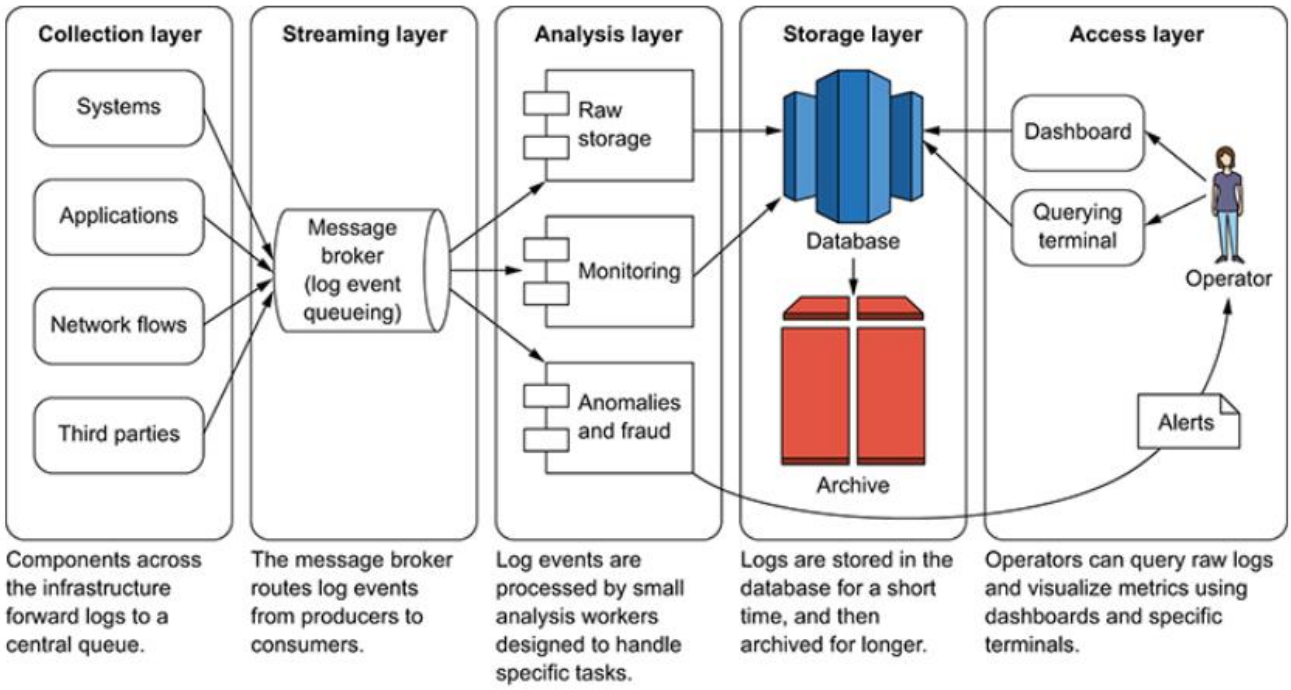

Рис. 2.6. Етапи логування

Можна розбити увесь цей процес на 5 етапів:

- Етап збору. Етап збору подій з різних додатків, систем, мережевого обладнання, сервісів.
- Етап потокової передачі; Данні передаються до брокеру повідомлень, який виступає, як черга до передачі їх до аналізу.
- Етап аналізу; Данні оброблюються та аналізуються спеціальними правилами, щоб знайти аномалії, певні моделі та шаблони.
- Етап зберігання; Данні зберігаються в базах даних на протязі короткого часу, а потім архівуються.
- Етап доступу. Спеціальні програми використовуються для доступу до даних та їх візуалізації.

Існує декілька сервісів, що виконують функції етапів логування, серед них найбільш популярним є ELK. ELK це абревіатура трьох проектів з відкритим кодом: Elasticsearch, Logstash і Kibana.

Elasticsearch це пошукова система на основі бібліотеки Lucene. Він надає розподілену повнотекстову пошукову систему з підтримкою кількох клієнтів із

веб-інтерфейсом HTTP та документами JSON. Він працює на етапі зберігання, аналізу та доступу.

Logstash це легкий конвеєр обробки даних із відкритим вихідним кодом на стороні сервера, який дозволяє збирати дані з різних джерел, перетворювати їх на льоту та надсилати до потрібного місця призначення.

Kibana це інструмент візуалізації та дослідження даних, який використовується для аналітики журналів, моніторингу додатків і при операційній розвідці.

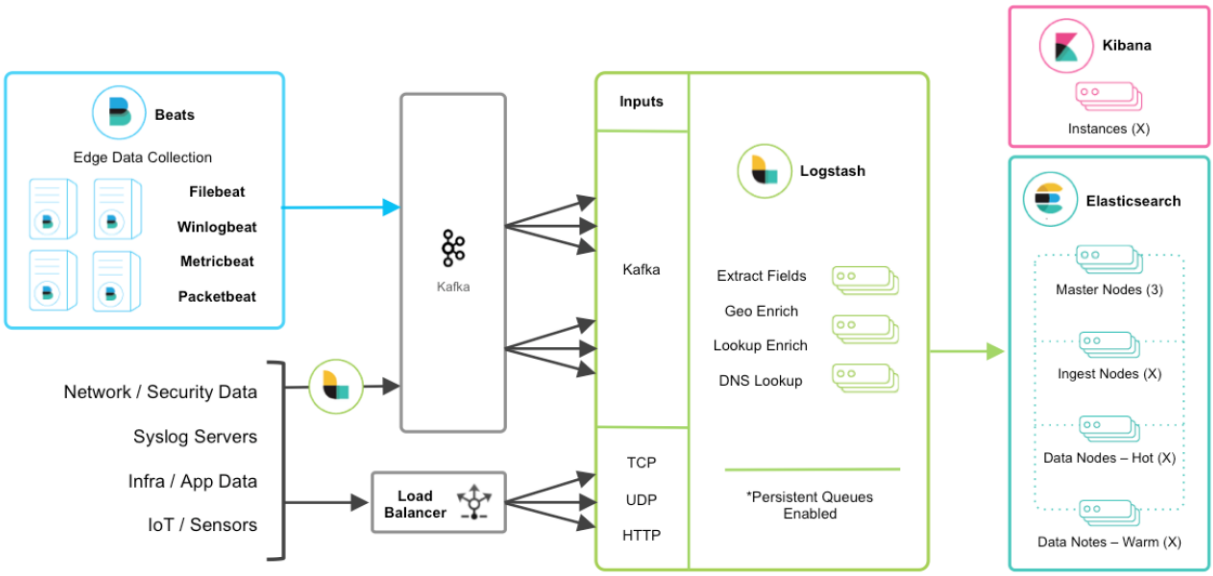

Рис. 2.7. Архітектура логування в ELK [12]

Також ще використовують Beats. Beats це легкі агенти, що відправляють дані з систем, де встановлені до Elasticsearch чи Logstash.

Є 7 типів Beats агентів:

- Filebeat
- Metricbeat
- Packetbeat
- Winlogbeat
- Auditbeat
- Heartbeat

- Functionbeat

Окрім збору та аналізу даних, треба ще зробити систему та процес реагування на інциденти. AWS для цього має плагін Alerting у їх версії ELK та сервіс SNS. Amazon Simple Notification Service це один з сервісів, який відповідає за відправлення повідомлень. Клієнти можуть підписатися на сервіс, що буде відправляти дані в потрібне місце, приклад роботи є на рис. нижче.

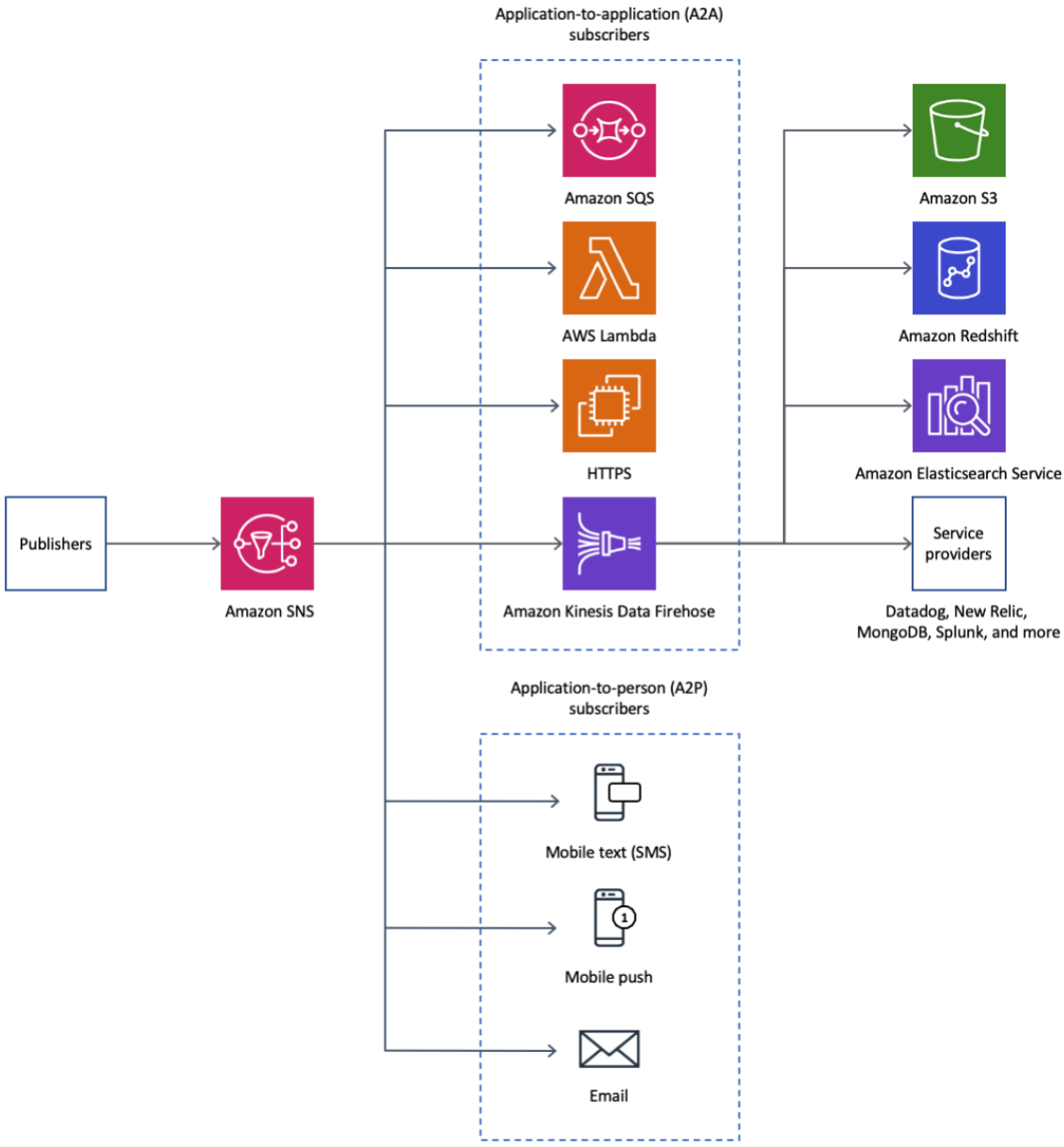

Рис. 2.8. SNS сервіс [12]

Щоб додати SNS сервіс до функціоналу Alerting плагіну, треба зробити такі кроки:

1. Обрати пункт Alerting в ELK;
2. Додати пункт призначення відправки повідомлень;
3. Додати унікальне ім’я;
4. Для типу відправки обрати SNS;
5. Додати ARN для сповіщення;
6. Додати ARN для IAM ролі;

Далі наведено приклади доданих записів:

```
{
  "Version": "2021-11-07",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "es.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }]
}
{
  "Version": "2021-11-07",
  "Statement": [{
    "Effect": "Allow",
    "Action": "sns:Publish",
    "Resource": "sns-topic-arn"
  }]
}
```

2.3. Методи та засоби захисту комунікацій

Захист каналу комунікацій досягається при дотримані трьох речей, які повинні бути притаманні інформації, що передається. Цими речами є такі 3 характеристики, як:

- Конфіденційність;
- Цілісність;
- Автентичність.

Конфіденційність відповідає за гарантію того, що лише дозволені люди мають змогу отримати доступ до інформації, що передається. Цілісність відповідає за те, що повідомлення не можуть бути змінені при передачі від одної людині іншій. Автентичність відповідає за гарантію того, що той, хто надсилає повідомлення, може підтвердити, що сам він його відіслав.

Усі ці характеристики можуть бути забезпечені при використані HTTPS протоколу. HTTPS це захищена версія HTTP, який є основним протоколом, який використовується для передачі даних в Інтернеті. HTTPS використовує TLS або SSL для шифрування даних, що передаються.

Для розуміння того, як TLS допомагає захистити дані, треба зрозуміти метод роботи діалогу між обома сторонами, що хочуть заключити безпечне з'єднання. TLS потрібен для того, щоб вирішити, як клієнт та сервер будуть розмовляти. Це відбувається через вибір алгоритмів з'єднання, які ще називають набором шифрів. Є багато наборів шифрів, деякі з них старіють та стають небезпечними для використовування, бо в імплементації алгоритмів шифрування знаходять вразливості.

Спрощена схема домовленості в TLS зображена на малюнку нижче.

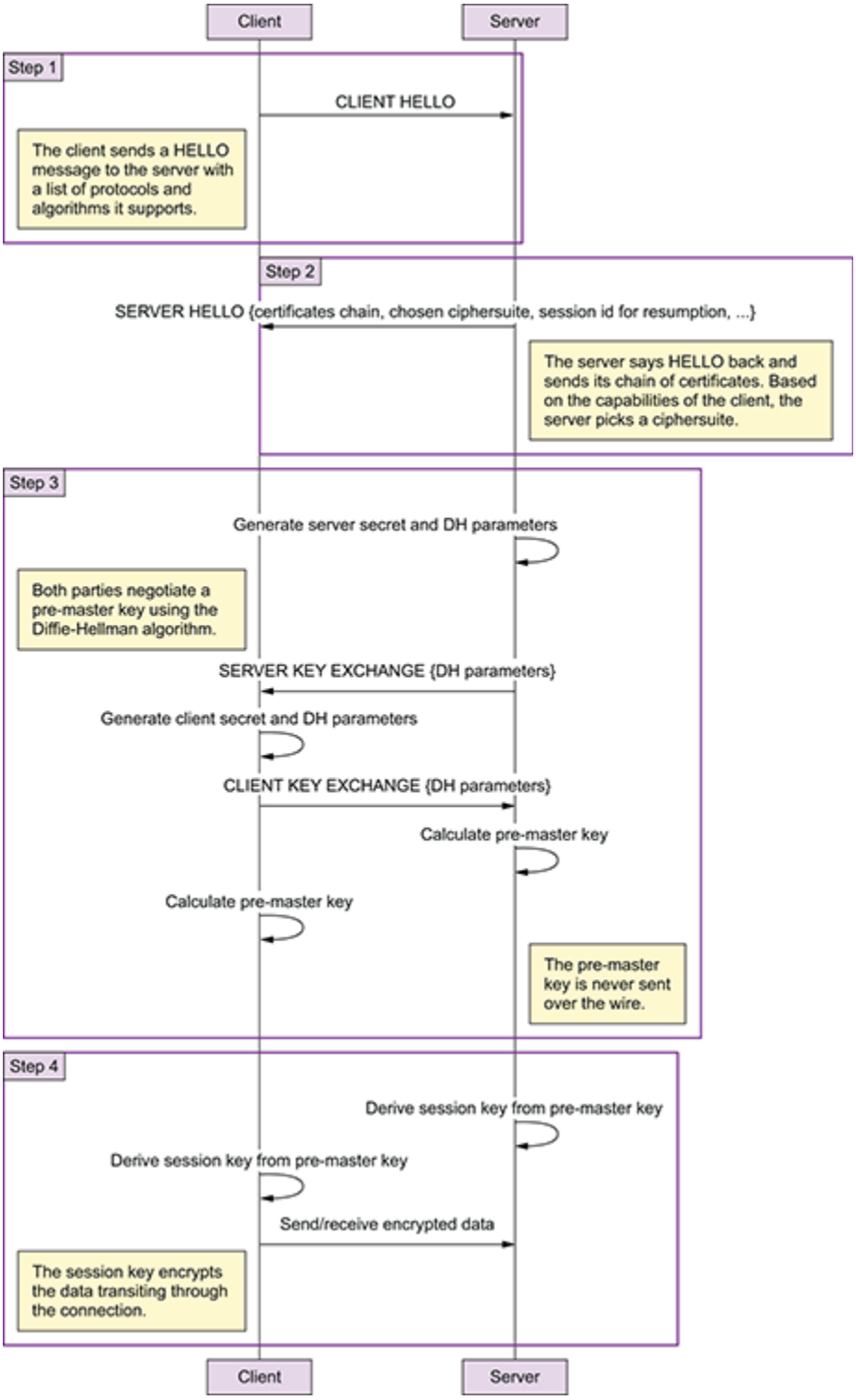

Рис. 2.9. Процес роботи TLS

Кроки роботи TLS:

1. Клієнт надсилає на сервер повідомлення HELLO зі списком протоколів і алгоритмів, які він підтримує.
2. Сервер каже HELLO і надсилає свій ланцюжок сертифікатів. Виходячи з

можливостей клієнта, сервер вибирає набір шифрів.

3. Якщо набір шифрів підтримує ефемерний обмін ключами, як це робить ECDHE, сервер і клієнт узгоджують попередній ключ за допомогою алгоритму Діффі-Хеллмана. Головний ключ ніколи не передається по мережі.
4. Клієнт і сервер створюють ключ сеансу, який буде використовуватися для шифрування даних, що передаються через з'єднання.

Після конфігурування TLS, треба зробити процес тестування шифрів та версій TLS, які підтримуються, бо недостатньо лише його зробити через ризик знаходження нових вразливостей в алгоритмах шифрування.

Окрім конфігурування TLS, можна включити таку функцію, як HTTP Strict-Transport-Security та HSTS [14]. Strict-Transport-Security це заголовок, який відправляє веб сервер браузеру, щоб сказати, що підключення до сайту повинно використовувати лише HTTPS протокол.

2.4. Методи та засоби захисту CI/CD

Як було вже проаналізовано та розглянуто у першому розділі, з’явились нові моделі розробки. Ці моделі розробки використовують нові інструменти розробки, які впливають на загальний стан безпеки компанії. Щоб покращити рівень кібербезпеки компанії треба розібратись в цих нових інструментах та процесах.

У розробників з’явилось багато нової інфраструктури, яку потрібно захищати, а наприклад репозиторії для проектів, типу Github, Gitlab. Контейнери Docker, які широко використовують при розробці та CI/CD сервіси.

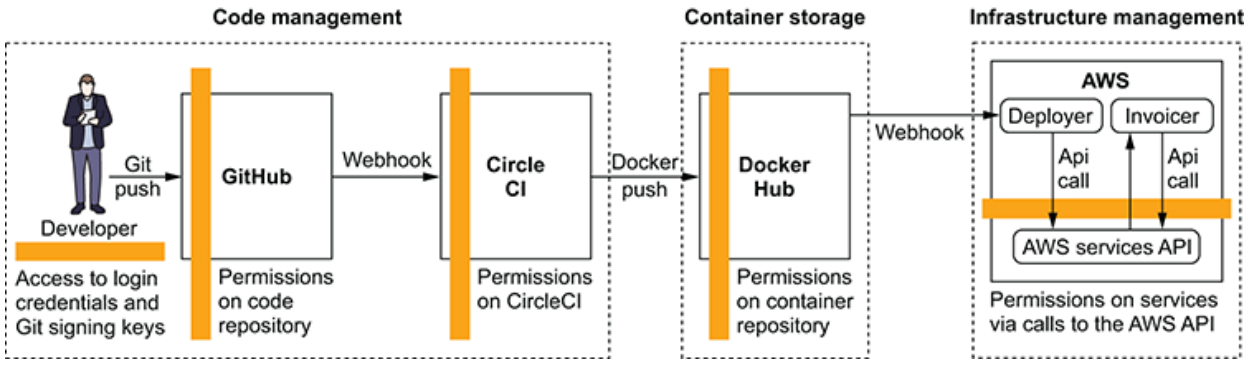

Рис. 2.9. Приклад нових сервісів розробки

Основні сфери, куди треба приділити увагу це:

- Дозволи користувачів та сервісів;
- Доступ до репозиторію;
- Доступ до інфраструктури побудови коду;
- Відсутність політики менеджменту секретів.

Опираючись на приклад вище, щоб забезпечити захист CI/CD треба:

- Захистити доступ до репозиторія. Для цього треба включити MFA, перевірити access token, які можуть бути у публічних репозиторіях. Також треба перевірити доступ інших сервісів до репозиторія.
- При підключені сервісів, як Circle CI, треба перевіряти, які доступи вони потребують при OAuth аутентифікації. Треба створити звичайний спеціальний акаунт з обмеженим доступом до Github ресурсів, щоб використовувати його з цими сервісами.
- Docker Hub репозиторій повинен бути захищеним, щоб атакуючий не міг підмінити зображення контейнера, де буде запускатися програмне забезпечення. Для цього достатньо використовувати механізм Docker Content Trust, щоб перевіряти підписи контейнерів.
- В AWS потрібно створити спеціальну IAM роль, яка буде мати мінімально потрібні рівні доступу, щоб використовувати потрібні сервіси та ресурси.
- Останньою річчю є менеджмент секретів та паролів. Для цього є декілька підходів. Один з них Hashicorp Vault, це сервіс який дозволяє підняти сервер менеджменту паролів та секретів.

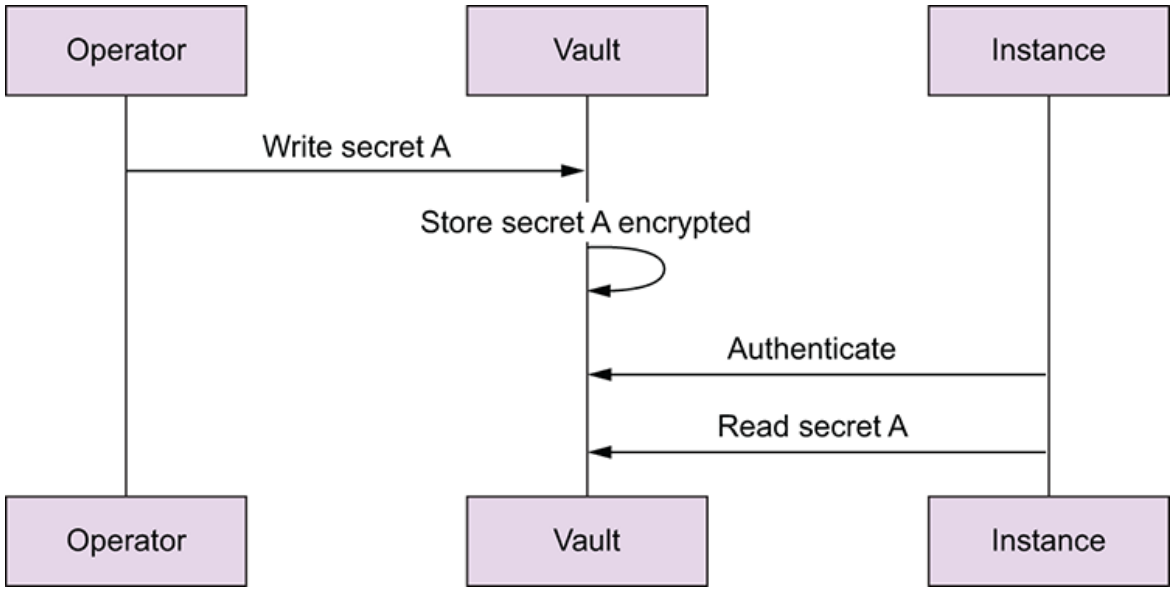

Рис. 2.10. Схема роботи Vault

Резюмуючи тему захисту CI/CD треба сказати, що більшість проблем пов’язані з доступом та дозволами. Якщо підійти відповідально до створення акаунтів з мінімальними правами та IAM політик, можна забезпечити непоганий рівень безпеки CI/CD.

3 РОЗРОБЛЕННЯ ВАРІАНТА ТЕХНОЛОГІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ РОЗРОБКИ БЕЗПЕЧНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ У СУЧАСНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

3.1. Організація процесу захисту розробки програмного забезпечення у сучасних інформаційних системах

Сучасні команди розробки використовують такі методології розробки, як Agile та DevOps. Це одні з найновіших та найпопулярніших методологій розробки програмного забезпечення. Перша робить процес розробки швидким та ітеративним. Друга додає в цей процес ще багато питань з управління інфраструктурою програмного забезпечення.

Для того, щоб організувати процес захисту розробки програмного забезпечення у сучасних інформаційних системах, треба застосувати багато процесів, методів, підходів та інструментів. Вони повинні бути спроможні інтегруватися в методології DevOps та Agile.

Ці підходи до захисту можна зобразити на таблиці.

Таблиця 3.1.

Підходи захисту розробки у SDLC

Підхід	Опис
Навчання розробників	Розробників треба навчати проблемам з кібербезпеки. Це можна робити за допомогою ресурсів OWASP Top 10, курсів та спеціальних сервісів.
Статичний аналіз	Використовується як один із етапів SDLC. Перевіряє код на вразливості.

Продовження таблиці 3.1.

Підходи захисту розробки у SDLC

Моделювання загроз	Є великою, але дуже актуальною темою. Існує багато методологій моделювання загроз, деякі з них добре додаються в процес розробки програмного забезпечення.
Огляди дизайну	Є одним із процесів розробки програмного забезпечення. Може застосовуватись для огляду дизайну з боку безпеки.
Динамічне сканування	Використовується як один із етапів SDLC. Перевіряє програмне забезпечення на вразливості. Для цього він його запускає та сканує.
Перевірка на проникнення	Один із процесів перевірки безпеки. Може бути трьох типів: blackbox, whitebox, greybox.

Наступною темою захисту в сучасних компаніях є хмарна інфраструктура. Для того, щоб забезпечити захист хмарної інфраструктури, треба описати проблеми, які їй притаманні та поставити пріоритети того, що буде виправлятись у першу чергу.

Таблиця 3.2.

Killchains у хмарному середовищі [15]

Killchain	Вплив
Публічний доступ до облікових даних API, що дозволяють вкрасти акаунт	Високий

Продовження таблиці 3.2.

Killchains у хмарному середовищі

Killchain	Вплив
Відкриті порти для віддаленого доступу	Високий
Публічний доступ до баз даних	Високий
Публічний доступ S3	Високий
Підробка запитів серверу	Середній
Криптомайнінг	Високий
Атаки на мережу	Середній
Скомпрометовані секрети	Високий
Subdomain Takeover	Середній

Приклад проблем у таблиці можна вирішити, якщо побудувати правильну стратегію захисту хмарної інфраструктури. Для цього потрібно обрати основні функції та процеси, що потрібно зробити та домени, де потрібно їх застосовувати. Для цього я скористаюсь NIST Cybersecurity Framework [16]. За допомогою фреймворку NIST, я виділяю 5 основних функцій:

1. Ідентифікація.
2. Захист.
3. Виявлення.
4. Реагування.
5. Відновлення.

Серед доменів можна назвати такі:

1. Політики та стандарти. Вони надають інформацію щодо найкращих методів забезпечення хмарної безпеки.
2. Архітектура. Архітектура мережі, управління доступом, класифікація даних та управління секретами.
3. Перевірка відповідності. Постійна перевірка відповідності політикам и стандартам.
4. Безпека ланцюга поставок.
   - Забезпечення безпеки зображень контейнерів;
   - Сканування Docker, Kubernetes конфігурацій;
   - Захист цілісності ланцюга поставок;

- Правила обмеження доступів.

5. Моніторинг та сповіщення. Імплементація журналювання подій, щоб мати ретроспективну видимість інцидентів та також реагування на них у реальному часі.
6. Інциденти. Створення процесів по автоматичному виправленню інцидентів безпеки.
7. Безперервність бізнесу. Підготовка контрзаходів для несподіваних інцидентів або катастроф.

Наступним кроком буде опис конкретних задач, що потрібно зробити, щоб забезпечити потрібний рівень безпеки. Також треба описати рівні безпеки, як це зроблено в OWASP SAMM, де перший рівень відповідає найбільш пріоритетним та мінімальним стандартам, наприклад перший рівень захисту хмарної інфраструктури буде включати такі напрямки та домени:

- Політики та стандарти. Визначаються найбільш важливі політики, наприклад політика безпеки в хмарі.
- Архітектура. Створюється архітектура мережі та її сегрегація. Створюється система керування доступом та секретами.
- Перевірка відповідності. Робиться інвентаризація ресурсів та їх автоматичне сканування.
- Безпека ланцюга поставок. Схеми контролю доступів для користувачів у хмарі, сканування Docker зображень.
- Моніторинг та сповіщення. Визначається стратегія логування подій в хмарному середовищі та будується архітектура рішення.

Після досягнення першого рівня безпеки, треба переходити до другого, виконуючи поставлені завдання. Ці рівні ще називають рівнями зрілості.

- Політики та стандарти. Розробляються нові стандарті, які охоплюють інші тема, наприклад контроль за відкритою інфраструктурою ключів.
- Архітектура. Після конфігурування найбільш важливих IAM переходять до створення ролей та груп для користувачів.

- Перевірка відповідності. Починається використання автоматичних сканерів конфігурацій, що перевіряють конфігурацію на відповідність написаним політикам та стандартам. Схожим займається сканер CIS Benchmark.
- Безпека ланцюга поставок. Створення безпечних Docker контейнерів, підписання їх, як один з етапів життєвого циклу розробки програмного забезпечення.
- Моніторинг та сповіщення. Покращення рішення, що було зроблено на першому рівні зрілості. Конфігурування сповіщень на індикатори компрометації.

На третьому рівні зрілості треба перейти до наступних кроків:

- Політики та стандарти. Написання нових політик, наприклад політика контроля та управління доступом.
- Перевірка відповідності. Сповіщення про зміни в конфігураціях важливих сервісів, які було вже конфігуровано згідно до політик та стандартів. В цьому може допомогти AWS Security Hub сервіс, який може відправляти сповіщення в SIEM.
- Безпека ланцюга поставок. Перевірка конфігурації Kubernetes кластерів. Сканування їх на відомі вразливості. Перевірка архітектури Kubernetes кластерів.
- Моніторинг та сповіщення. Створення графічних візуалізацій та панелей подій, що допомагають швидко розібратися в стані організації та помітити аномалії чи інциденти.
- Інциденти. Створення процесів автоматизації найбільш типових інцидентів.

На четвертому треба зробити такі кроки:

- Безперервність бізнесу. Початок вирішування питань безперервності бізнесу. План відновлення після інцидентів повинен бути написаний тут.
- Моніторинг та сповіщення. Використання IDS інструментів в

інфраструктурі, створення автоматичних сповіщень про зміни в конфігураціях.

- Інциденти. Створення документів, де чітко описані процеси, які потрібно почати при виявлені інциденту
- Безпека ланцюга поставок. Використання спеціальних методологій перевірки цілісності в життєвому циклі розробки.

На останньому, найвищому 5 рівні зрілості такі кроки:

- Моніторинг та сповіщення. Використання інструментів, що можуть допомогти помітити складні види атак та експлуатації, наприклад передача даних за допомоги DNS запитів.
- Інциденти. Створення автоматичних механізмів збору даних після інциденту, які можуть допомогти при подальшому розслідуванні. Наприклад створення копій диску, ізолювання віртуальної машини від мережі.
- Безперервність бізнесу. Перевірка усіх процесів та інструментів, що було використані в організації для забезпечення її безпеки, щоб перевірити, що вони всі працюють.

3.2. Розроблення рекомендацій щодо застосування технології забезпечення розробки безпечного програмного забезпечення у сучасних інформаційних системах

Так як тема захисту життєвого циклу розробки та хмарної інфраструктури є досить великою. В роботі наведено фокус на технології забезпечення безпечного процесу розробки програмного забезпечення за допомоги імплементації сканування DAST та SAST в процесі CI/CD.

У ролі DAST виступає OWASP ZAP, а CI/CD процес буде реалізований на базі Gitlab. OWASP ZAP сканує додаток та генерує знайдені вразливості. Знайдені

вразливості порівнюються з тими, що були знайдені до цього і нові додаються в запит злиття.

OWASP ZAP має 3 види сканування:

1. Базове сканування – обмежене за часом, не користується AJAX.
2. Повне сканування – використовує AJA, щоб знаходити нові URL, для сканування.
3. API Scan – повне сканування API, визначеного за допомогою OpenAPI / Swagger або GraphQL.

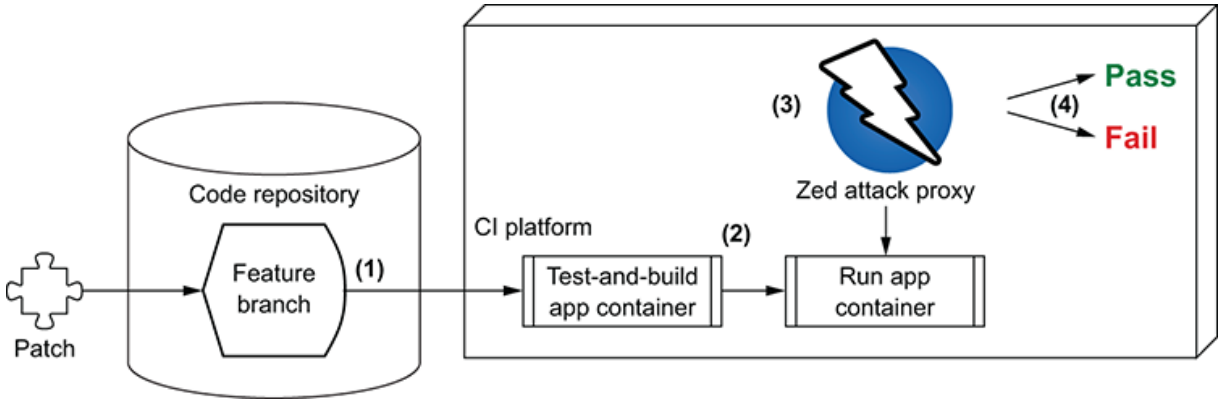

Рис. 3.1. Схема роботи DAST ZAP

Так як багато компаній зараз використовує Docker зображення. Тут розглянутий варіант, де DAST сканує код, що вже зібраний в Docker зображення. Для цього потрібно створити файл, де описаний процес сканування OWASP ZAP.

Цей файл є конфігурацією сканеру та того, як він буде працювати. OWASP ZAP може також проводити сканування конкретних частин веб додатків. Це може бути зручним, коли увесь додаток дуже великий, щоб бути повністю просканованним при виконані одного з етапів життєвого циклу розробки.

```yaml
stages:
  - build
  - test
  - deploy
  - dast

include:
  - template: DAST.gitlab-ci.yml

deploy:
  services:
  - name: docker:dind
    alias: dind
  image: docker:19.03.5
  stage: build
  script:
    - docker login -u gitlab-ci-token -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker pull $CI_REGISTRY_IMAGE:latest || true
    - docker build --tag $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA --tag $CI_REGISTRY_IMAGE:latest .
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    - docker push $CI_REGISTRY_IMAGE:latest

services:
  - name: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA
    alias: yourapp

variables:
  DAST_FULL_SCAN_ENABLED: "true"
  DAST_ZAP_USE_AJAX_SPIDER: "true"
  DAST_INCLUDE_ALPHA_VULNERABILITIES: "true"
```

Рис. 3.2. Конфігурація DAST ZAP

Для того, щоб сканування DAST виконувалось автоматично, треба додати його конфігурацію в .gitlab-ci.yml файл.

Сканування методом SAST конфігурується схожим способом. Треба зробити конфігурацію, де описано поведінку SAST сканеру.

```yaml
sonarqube-check:
  image:
    name: sonarsource/sonar-scanner-cli:latest
    entrypoint: [""]
  variables:
    SONAR_USER_HOME: "${CI_PROJECT_DIR}/.sonar"
    GIT_DEPTH: "0"
  cache:
    key: "${CI_JOB_NAME}"
    paths:
      - .sonar/cache
  script:
    - sonar-scanner -Dsonar.qualitygate.wait=true
  allow_failure: true
  only:
    - merge_requests
    - master
    - develop
```

Рис. 3.3. Конфігурація SAST

Таким чином, разом з працюючими DAST та SAST, які сканують код, що додається до репозиторія, можна значно покращити рівень безпеки програмного забезпечення. Використовування DevOps підходів та автоматизації полегшують життя спеціалістів з кібербезпеки.

Розробники можуть виправляти вразливості ще до потрапляння програмного забезпечення в публічний доступ. Все це є частиною концепції безперервної безпеки. Це є ідеальною ситуацією, коли вразливості не доходять до користувачів та виправляються ще до релізу.

Звичайно, що не всі вразливості будуть знайдені за допомогою SAST та DAST сканування, але це є гарним прикладом того, що можна додати в життєвий цикл розробки, щоб покращити рівень безпеки продукту.

Окрім автоматичного сканування є ще такі методи захисту в ітеративній концепції покращення безпеки програмного забезпечення:

1. Оцінка ризиків;
2. Мануальне тестування;

3. Red team;
4. Bug bounty.

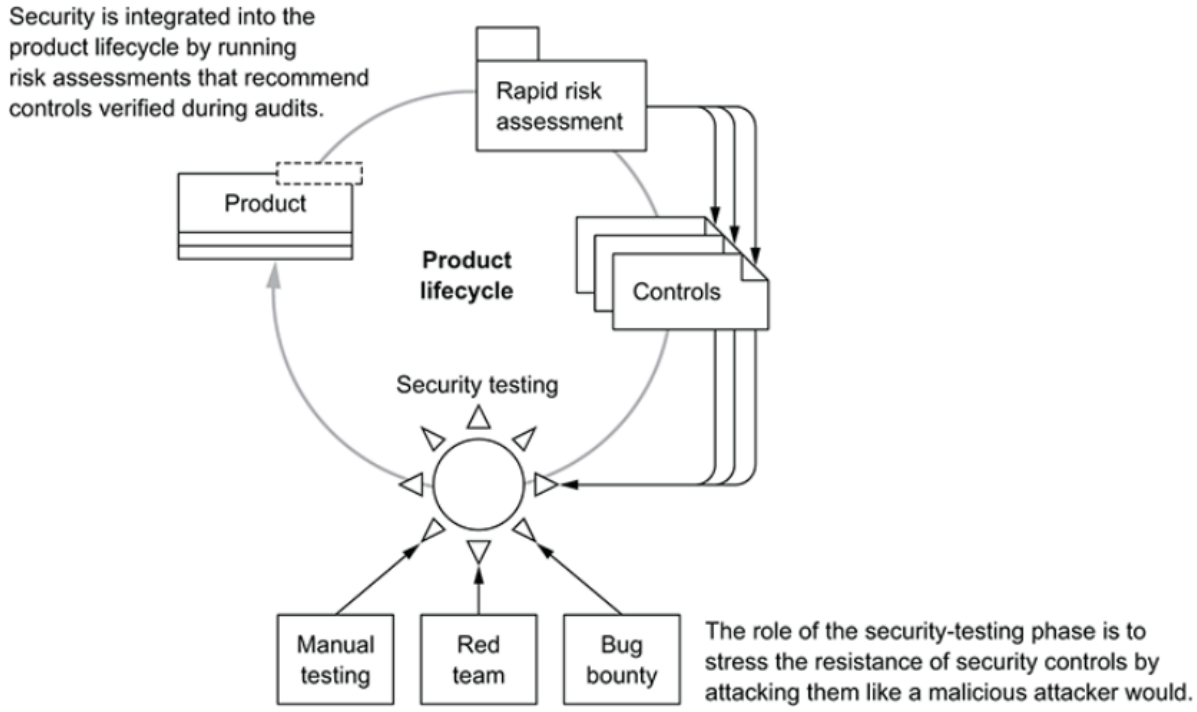

Рис. 3.4. Життєвий цикл продуктової безпеки

Забезпечення безпеки це не продукт, а процес, тому він включає до себе багато різних задач. Ці задачі вирішуються великою кількістю інструментів та методів. В роботі надані рекомендації щодо покращення безпеки життєвого циклу розробки програмного забезпечення підприємстві.

ВИСНОВКИ

В результаті виконання цього дипломного проекту було аналіз проблеми забезпечення кібербезпеки життєвого циклу розробки програмного забезпечення у корпоративній інформаційній системі та визначено мета та завдання забезпечення захисту життєвого циклу розробки в корпоративній інформаційній системі.

У ході розв'язання поставлених задач були отримані наступні результати:

1. Було проаналізовано існуючі технології захисту життєвого циклу розробки в корпоративній інформаційній системі.

Для застосовування технологій захисту життєвого циклу розробки, треба було проаналізувати еволюцію змін життєвих циклів розробки. Причини та наслідки змін методологій та їх вплив на безпеку.

2. Досліджено методи та засоби захисту життєвого циклу розробки в корпоративній інформаційній системі.

Для цього в дипломній роботі було розглянуто, як виглядає нинішній найбільш популярний процес життєвого циклу розробки. Базуючись на цих знаннях було розкладено на різні сфери конкретні елементи життєвого циклу розробки, а саме захист веб додатків, захист хмарної інфраструктури, захист комунікацій та захист CI/CD.

3. Визначено процеси та методи для забезпечення безпеки в DevOps моделі веб додатків, хмарної інфраструктури та CI/CD.

Були розглянуті та проаналізовані рекомендації до кожної зі сфер, а саме веб додатків, інфраструктури хмари та CI/CD. Для CI/CD процесу було розглянуто рекомендації по застосуванню сканування SAST та DAST, як частину концепції безперервної безпеки. Був наведений приклад створення рекомендацій по захисту хмарної інфраструктури, який побудований на концепції схожою на Software Assurance Maturity Mode, де різні рівні зрілості, мають різні вимоги до безпеки.

Отримані результати можуть бути використані для розробки технології

забезпечення безпечного процесу розробки програмного забезпечення у сучасних корпоративних системах.

ПЕРЕЛІК ПОСИЛАНЬ

1. Лабораторія інформаційних технологій \ Глосарій [Електронний ресурс] – Режим доступу World Wide Web – URL - https://csrc.nist.gov/glossary/term/information_system
2. ISO/IEC/IEEE 12207:2017 Systems and software engineering — Software life cycle processes.
3. ICSE '87: Proceedings of the 9th international conference on Software EngineeringMarch 1987 Pages 328–338
4. Barry W. Boehm ComputerVolume 21Issue 5May 1988 pp 61–72
5. Developer Survey Results 2018. [Електронний ресурс] – Режим доступу World Wide Web – URL - https://insights.stackoverflow.com/survey/2018
6. Про основні засади забезпечення кібербезпеки України [Текст] : Закон України №2163-VIII від 8 липня 2017 р. / Верховна Рада України // Відомості Верховної Ради України. – 2017. – №45. – Ст. 403.
7. Amount of monetary damage caused by reported cyber crime to the IC3 from 2001 to 2020 [Електронний ресурс] – Режим доступу World Wide Web – URL - https://www.statista.com/statistics/267132/total-damage-caused-by-by-cyber-crime-in-the-us/
8. Software Security: The Trinity of Trouble [Електронний ресурс] – Режим доступу World Wide Web – URL – https://freedom-to-tinker.com/2006/02/15/software-security-trinity-trouble/
9. Malware found in npm package with millions of weekly downloads [Електронний ресурс] – Режим доступу World Wide Web – URL – https://therecord.media/malware-found-in-npm-package-with-millions-of-weekly-downloads/
10. OWASP Top Ten [Електронний ресурс] – Режим доступу World Wide Web – URL – https://owasp.org/www-project-top-ten/
11. The Unfortunate Reality of Insecure Libraries [Електронний ресурс] – Режим доступу World Wide Web – URL – https://cdn2.hubspot.net/hub/203759/file-1100864196-pdf/docs/Contrast_-_Insecure_Libraries_2014.pdf
12. ELK Stack + Kafka End to End Practice¶ [Електронний ресурс] – Режим доступу World Wide Web – URL – https://elastic-stack.readthedocs.io/en/latest/e2e_kafkapractices.html

13. What is Amazon SNS? [Електронний ресурс] – Режим доступу World Wide Web – URL – https://docs.aws.amazon.com/sns/latest/dg/welcome.html
14. Internet Engineering Task Force (IETF) Request for Comments: 6797 [Електронний ресурс] – Режим доступу World Wide Web – URL – https://datatracker.ietf.org/doc/html/rfc6797
15. Stop Today's Top 10 Cloud Attack Killchains [Електронний ресурс] – Режим доступу World Wide Web – URL – https://disruptops.com/stop-todays-top-10-cloud-attack-killchains/
16. Framework for Improving Critical Infrastructure Cybersecurity [Електронний ресурс] – Режим доступу World Wide Web – URL – https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)