НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра інформаційної безпеки

«На правах рукопису»
УДК 004.738:004.056.5

«До захисту допущено»
Завідувач кафедри
_____ Дмитро ЛАНДЕ
“___”_____2025 р.

Дипломна робота

на здобуття ступеня бакалавра

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

спеціальності 125 «Кібербезпека»

на тему: **Система автоматизованої генерації лідів для впровадження сервісів кібербезпеки в AWS середовищі**

Виконав (-ла): здобувач вищої освіти **IV** курсу, групи ФБ-11

Анучін Максим Костянтинович

(підпис)

Науковий керівник доцент кафедри ІБ, к.т.н. Рибак О.В.

(посада, науковий ступінь, вчене звання, прізвище, ім’я, по батькові) (підпис)

Рецензент доцент кафедри ІБ, к.т.н. Хмельницький М.О.

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище, ім’я, по батькові) (підпис)

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

Здобувач вищої освіти _____

(підпис)

Київ – 2025 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра інформаційної безпеки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 125 «Кібербезпека»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Дмитро ЛАНДЕ

(підпис)

«___»_____2025 р.

ЗАВДАННЯ

на дипломну роботу здобувачу вищої освіти

Анучіну Максиму Костянтиновичу

1. Тема роботи: Розробка системи автоматизованої генерації лідів для впровадження сервісів кібербезпеки в AWS-середовищі,

Науковий керівник роботи: Рибак Олександр Владиславович, к.т.н., доцент кафедри інформаційної безпеки

затверджені наказом по університету від «26» травня 2025 р. № 1761-с

2. Термін подання здобувачем дипломної роботи 19.07.2025 р. (дата передзахисту)

3. Вихідні дані

Власні Python/JS-скрипти, логи перевірки доменів, CIDR-префіксів AWS, статистика використання сервісів GuardDuty, Inspector, Macie, публічні бази даних AWS ip-ranges.json, аналітичні звіти щодо FinOps-ефективності впровадження сервісів безпеки

4. Зміст роботи

Аналіз проблем виявлення незахищених ресурсів у публічному середовищі AWS, розробка системи автоматизованого виявлення й категоризації доменів, побудова воронки B2B-продажів послуг безпеки, інтеграція інструментів аналізу кіберризиків, апробація на реальних кейсах, оцінка ефективності, економічного впливу й потенційного зменшення втрат від атак.

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

6. Дата видачі завдання 12 вересня 2024

Календарний план

№ з/п	**Назва етапу**	**Термін виконання**	**Примітка**
1	Визначення теми, погодження з науковим керівником	12 вересня 2024 р.	Виконано

2	Формулювання мети, об’єкта, предмета дослідження	15 – 30 вересня 2024 р.	Виконано
3	Огляд літератури й аналітика ринку кібербезпеки	жовтень 2024 р.	Виконано
4	Розроблення змісту роботи, затвердження плану	1 – 10 листопада 2024 р.	Виконано
5	Написання розділу 1	листопад 2024 р.	Виконано
6	Написання розділу 2	грудень 2024 р.	Виконано
7	Реалізація скриптів, збір тестових даних	січень 2025 р.	Виконано
8	Написання розділу 3	лютий 2025 р.	Виконано
9	Валідація рішень на кейсах клієнтів	1 – 15 березня 2025 р.	Виконано
10	Оформлення висновків та рекомендацій	16 – 31 березня 2025 р.	Виконано
11	Підготовка списку джерел, додатків	1 – 10 квітня 2025 р.	Виконано
12	Перевірка на плагіат, внесення правок	11 – 20 квітня 2025 р.	Виконано
13	Підготовка презентації та доповіді	1 – 10 червня 2025 р.	Виконано
14	Рецензування та відгук керівника	11 – 23 червня 2025 р.	Виконано

15	Попередній захист	19 червня 2025 р.	Виконано
16	Захист дипломної роботи	24 червня 2025 р.	

Здобувач вищої освіти _____ Анучін Максим Костянтинович

(підпис)(власне ім’я, ПРІЗВИЩЕ)

Науковий керівник _____ Рибак Олександр Владиславович

(підпис)(власне ім’я, ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота містить 87 сторінки, 1 рисунків, 6 таблиць, 25 джерел у переліку посилань та 9 додатків.

Метою дослідження є розробка та практичне впровадження системи автоматизованої генерації «теплих» лідів для підключення сервісів кібербезпеки в середовищі Amazon Web Services (AWS).

Об’єкт дослідження — процеси виявлення потенційних клієнтів (лідогенерація) у B2B-сегменті хмарних технологій.

Предмет дослідження — методи та інструменти автоматизації пошуку компаній-користувачів AWS, що не мають увімкнених сервісів безпеки, та технології їхнього швидкого підключення через AWS SDK.

Методи дослідження:

• аналіз відкритих технічних даних (SSL-сертифікати, DNS, IP-діапазони);

• використання B2B-платформ (Apollo.io, Snov.io) для збагачення контактних даних;

• мікросервісна реалізація Python-скриптів;

• інструменти AWS SDK (GuardDuty, Security Hub, Inspector, Macie, WAF, IAM Access Analyzer);

• порівняльний аналіз ефективності (ROI) на основі кейс-стаді клієнтів Umbrelly.

У результаті роботи:

1. Спроєктовано архітектуру SaaS-рішення «SSL → Domain → Snov.io» з багаторівневою фільтрацією.

2. Реалізовано скрипти для автоматичного збору доменів, перевірки хостингу на AWS та збагачення даних через API Apollo/Snov.io.

3. Створено модуль підключення Security Pack (GuardDuty, Inspector, Security Hub, Macie) через AWS SDK у клієнтських акаунтах.

4. Проведено апробацію на кейсах Dots Platform, Evacodes, Gigradar, Finverity, що засвідчила підвищення upsell-конверсії з 5 % до 18 % та скорочення часу на лідогенерацію на 60 %.

5. Запропоновано рекомендації щодо масштабування системи та використання ML-скорингу для подальшого підвищення точності відбору лідів.

Ключові слова: AWS, кібербезпека, лідогенерація, GuardDuty, Security Hub, автоматизація, SSL-сертифікати, Apollo.io, Snov.io, SaaS.

ABSTRACT

The diploma thesis comprises 87 pages, 1 figures, 6 tables, 25 references in the bibliography, and 9 appendix.

Purpose of the study — to design and practically implement an automated system for generating “warm” leads to onboard cybersecurity services within the Amazon Web Services (AWS) environment.

Object of the study — lead-generation processes for potential customers in the B2B cloud-technology segment.

Subject of the study — methods and tools for automating the search for AWS-based companies that have not enabled security services, and technologies for their rapid onboarding via the AWS SDK.

Research methods:

•analysis of open technical data (SSL certificates, DNS records, IP ranges);

•enrichment of contact data through B2B platforms (Apollo.io, Snov.io);

•microservice implementation of Python scripts;

•AWS SDK tools (GuardDuty, Security Hub, Inspector, Macie, WAF, IAM Access Analyzer);

•comparative ROI analysis based on Umbrelly client case studies.

Results obtained:

1.An architecture for the SaaS solution “SSL → Domain → Snov.io” with multi-level filtering has been designed.

2.Scripts have been implemented for automatic domain collection, AWS-hosting verification, and data enrichment via the Apollo/Snov.io APIs.

3.A module has been created to connect the Security Pack (GuardDuty, Inspector, Security Hub, Macie) to client accounts through the AWS SDK.

4.Pilot tests on the cases of Dots Platform, Evacodes, Gigradar, and Finverity demonstrated an increase in upsell conversion from 5 % to 18 % and a 60 % reduction in lead-generation time.

5.Recommendations are proposed for system scaling and applying ML-based scoring to further increase lead-selection accuracy.

Keywords: AWS, cybersecurity, lead generation, GuardDuty, Security Hub, automation, SSL certificates, Apollo.io, Snov.io, SaaS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AWS (Amazon Web Services) — платформа хмарних обчислень від компанії Amazon, що забезпечує широкий спектр інфраструктурних і сервісних рішень.

SDK (Software Development Kit) — набір інструментів для розробки програмного забезпечення, що дозволяє взаємодіяти з хмарними сервісами.

SaaS (Software as a Service) — модель надання програмного забезпечення як послуги через інтернет.

IAM (Identity and Access Management) — механізми управління користувачами та правами доступу в середовищі AWS.

WAF (Web Application Firewall) — веб-екран для захисту веб-додатків від атак типу SQLi, XSS тощо.

PII (Personally Identifiable Information) — персонально ідентифікована інформація (номери паспортів, платіжні дані, адреси тощо).

CSV (Comma-Separated Values) — формат зберігання табличних даних, де значення відокремлені комами.

API (Application Programming Interface) — інтерфейс прикладного програмування, що дозволяє одній системі взаємодіяти з іншою.

OSINT (Open Source Intelligence) — метод збору даних із відкритих джерел.

CT лог (Certificate Transparency Log) — відкритий реєстр SSL-сертифікатів, що дозволяє відслідковувати видачу сертифікатів.

Lead / Лід — потенційний клієнт, який виявив зацікавленість у продукті або може бути цільовим для пропозиції.

Upsell — продаж додаткових послуг або продуктів уже наявному клієнту.

Security Pack — умовна назва набору AWS-сервісів безпеки (GuardDuty, Inspector, Security Hub, Macie), які активуються клієнту.

GuardDuty — сервіс виявлення загроз у хмарному середовищі.

Inspector — сервіс сканування вразливостей у EC2, Lambda, контейнерах.

Security Hub — агрегатор безпекових подій та перевірок на відповідність стандартам.

Macie — сервіс виявлення чутливих даних у S3.

Apollo.io / Snov.io — B2B-платформи для отримання контактної інформації про компанії та їхніх співробітників.

ROI (Return on Investment) — показник окупності інвестицій.

ВСТУП

Актуальність теми

Стрімка міграція бізнесу до хмарних платформ, зокрема Amazon Web Services (AWS), кардинально змінила ландшафт корпоративних інформаційних технологій. Гнучкість, масштабованість та економічна ефективність, які пропонують хмарні обчислення, стали ключовими драйверами цифрової трансформації для компаній будь-якого розміру.[1] Однак цей перехід супроводжується появою нових, складних викликів у сфері кібербезпеки. Захист динамічних, розподілених та ефемерних ресурсів у хмарі вимагає принципово нових підходів, відмінних від традиційних периметральних моделей безпеки.

Центральною проблемою сучасної хмарної безпеки є помилки конфігурації з боку клієнта. Згідно з прогнозами аналітичної компанії Gartner, до 2025 року понад 99% інцидентів безпеки в хмарі будуть спричинені саме помилками користувачів.[1] Цей прогноз підтверджується щорічними звітами про витоки даних, такими як Verizon DBIR, які вказують на «Різноманітні помилки» (Miscellaneous Errors), включно з неправильними конфігураціями, як на одну з вагомих причин інцидентів.[2] Наслідки таких помилок можуть бути катастрофічними. Неправильно налаштовані сховища Amazon S3, що залишаються публічно доступними, надлишкові дозволи в системі управління ідентифікацією та доступом (IAM), відсутність належного моніторингу та логування — все це створює вектори атак, які призводять до значних фінансових та репутаційних втрат. Вартість таких інцидентів невпинно зростає: за даними Ponemon Institute, середня вартість витоку даних у 2024 році досягла рекордних $4.88 мільйона у світі та $9.36 мільйона у США.[3]

У цих умовах ринок рішень для управління станом безпеки хмари (Cloud Security Posture Management, CSPM) та управління поверхнею атаки (Attack Surface Management, ASM) демонструє стрімке зростання. Прогнозується, що ринок CSPM зросте з $5.75 мільярда у 2024 році до $10.37 мільярда до 2030 року,

із середньорічним темпом зростання (CAGR) 10.3%.[5] Ринок ASM, у свою чергу, показує ще більш вражаючу динаміку з прогнозованим CAGR 27.7% у період з 2025 по 2033 рік.[6] Це свідчить про гостру потребу бізнесу в інструментах, що дозволяють автоматизувати виявлення та виправлення конфігураційних помилок.

Попри наявність потужних нативних інструментів безпеки, які пропонує AWS, багато компаній, особливо в сегменті малого та середнього бізнесу (SME), не використовують їх повною мірою через брак експертизи, часу або ресурсів. Це створює унікальну можливість для керованих сервісних провайдерів (Managed Service Providers, MSP) та консультантів. Замість того, щоб чекати, поки клієнт зіткнеться з інцидентом, можна проактивно ідентифікувати потенційні слабкі місця в його інфраструктурі та запропонувати рішення.

Саме на вирішення цієї задачі спрямована дана дипломна робота. Вона пропонує та валідує інноваційну систему, що поєднує методи розвідки на основі відкритих джерел (OSINT) для автоматизованої генерації лідів з унікальною бізнес-моделлю «Save → Secure». Ця модель дозволяє клієнтам фінансувати посилення своєї кібербезпеки за рахунок коштів, заощаджених завдяки оптимізації хмарних витрат (FinOps). Таким чином, розроблене рішення не лише є технічним інструментом, а й виступає практичною реалізацією сучасних стратегій безпеки, таких як безперервне управління загрозами та вразливостями (Continuous Threat Exposure Management, CTEM), для проактивного виявлення та усунення прогалин у безпеці середовищ AWS.

Мета і задачі дослідження

Мета роботи: підвищення рівня захищеності хмарних інфраструктур компаній шляхом розробки та валідації автоматизованої системи генерації «теплих» лідів для впровадження нативних сервісів безпеки AWS.

Для досягнення поставленої мети було сформульовано наступні **задачі**:

1. Проаналізувати сучасні загрози безпеці хмарних середовищ, визначити ключові вектори атак, пов'язані з помилками конфігурації, та обґрунтувати необхідність проактивного підходу до управління безпековим станом інфраструктури.
2. Розробити архітектуру та детальні алгоритми роботи автоматизованого конвеєра для ідентифікації, валідації, збагачення та кваліфікації потенційних клієнтів на основі OSINT-даних.
3. Реалізувати програмний прототип ключових модулів системи з використанням мови програмування Python та сервісів AWS, і провести його експериментальну апробацію на реальних даних для оцінки ефективності та точності.
4. Обґрунтувати економічну ефективність бізнес-моделі «Save → Secure» на основі аналізу кейс-стаді та розробити практичні рекомендації щодо її застосування для посилення безпеки клієнтів шляхом автоматизованого розгортання пакета сервісів AWS Security.

Об'єкт, предмет та методи дослідження

Об'єкт дослідження — процеси виявлення та кваліфікації потенційних клієнтів (лідогенерація) у B2B-сегменті хмарних технологій.

Предмет дослідження — методи, алгоритми та інструменти для автоматизації пошуку компаній-користувачів AWS, що потребують посилення кібербезпеки.

Для вирішення поставлених задач було використано комплекс **методів дослідження**:

- **Аналіз відкритих технічних даних (OSINT):** збір та аналіз інформації з логів прозорості SSL/TLS-сертифікатів (Certificate Transparency Logs), систем доменних імен (DNS), сервісів WHOIS та публічних IP-діапазонів AWS.
- **Системний аналіз та проєктування:** розробка багаторівневої архітектури програмної системи, моделювання бізнес-процесів за допомогою нотації BPMN.
- **Програмна реалізація:** створення прототипу системи з використанням мови програмування Python 3.11, асинхронного програмування (asyncio), бібліотек для роботи з даними (pandas) та взаємодії з API.
- **Експериментальне дослідження:** проведення тестових прогонів системи на великих масивах даних (десятки тисяч доменів) для валідації архітектурних рішень та оцінки ефективності фільтрації.
- **Кейс-стаді (Case Study):** поглиблений аналіз результатів впровадження розробленого рішення для реальних клієнтів з метою підтвердження практичної цінності та бізнес-ефекту.
- **Економічний аналіз:** розрахунок показника повернення інвестицій (ROI) та порівняльний аналіз ключових метрик ефективності (конверсія, вартість залучення клієнта, життєва цінність клієнта) до та після впровадження системи.

Наукова новизна та практичне значення

Наукова новизна отриманих результатів полягає у розробці та обґрунтуванні комплексного підходу, що поєднує методи OSINT-аналізу для виявлення технічних індикаторів слабкої безпекової конфігурації з автоматизованим B2B-маркетингом. На відміну від існуючих рішень, які

фокусуються або на технічному скануванні (CSPM-інструменти), або на продажах (CRM-системи), запропонована архітектура створює синергетичний ефект, перетворюючи технічну вразливість на комерційну можливість. Вперше запропоновано та валідовано модель, де процес лідогенерації безпосередньо інтегрований з процесом автоматизованого розгортання засобів захисту, що дозволяє системно долати бар'єри впровадження кібербезпеки в SME-сегменті.

Практичне значення роботи полягає у створенні готового до впровадження прототипу SaaS-системи, що може бути використана керованими сервісними провайдерами (MSP) та консалтинговими компаніями для підвищення ефективності продажів та покращення безпеки своїх клієнтів. Розроблений набір Python-скриптів та архітектурних патернів дозволяє значно скоротити ручну працю, знизити вартість залучення клієнта та підвищити дохід за рахунок допродажу (upsell) сервісів безпеки. Економічна модель «Save → Secure» надає бізнесу чітке обґрунтування для інвестицій у кібербезпеку. Результати роботи можуть бути використані як методична основа для побудови аналогічних систем для інших хмарних платформ (Microsoft Azure, Google Cloud Platform).

1. АНАЛІЗ СУЧАСНИХ ПІДХОДІВ ДО ЛІДОГЕНЕРАЦІЇ ТА ПРОДАЖУ КІБЕРБЕЗПЕКИ

1.1 Проблематика підключення сервісів AWS Security клієнтами

Перенесення систем в хмару змінює модель безпеки: якщо раніше організації контролювали власні сервери фізично, то тепер багато компонентів інфраструктури стають віртуальними і розподіленими. AWS дотримується моделі спільної відповідальності (Shared Responsibility Model), згідно з якою AWS відповідає за “безпеку хмари” (інфраструктура, фізичні дата-центри, гіпервізор
тощо), а клієнт – за “безпеку в хмарі” (налаштування сервісів, контроль доступу, шифрування даних). На практиці це означає, що хоча AWS надає захищену базову платформу, замовники самі мають правильно конфігурувати сервіси і вживати заходів безпеки на своєму рівні. На жаль, не всі компанії усвідомлюють цю межу відповідальності або мають достатній досвід, що призводить до конфігураційних помилок. Типові загрози та вразливості в AWS-середовищі часто пов’язані з людським фактором та швидким розширенням хмарних ресурсів:

Некоректні налаштування доступу – одна з найпоширеніших проблем. Класичний приклад – відкриті для інтернету S3-бакети з конфіденційними даними. Зафіксовано численні інциденти, коли через помилкові політики доступу до S3 відбувалися витоки даних. Зокрема, у 2017 році було розкрито дані 198 мільйонів виборців США через публічно доступний S3-бакет. Такі випадки
продовжують траплятися, підкреслюючи критичність правильної конфігурації.

Недостатній контроль доступу (IAM) – неналежне управління обліковими записами і правами. Наприклад, використання єдиного облікового запису root для щоденних задач без MFA, або надання надто широких прав сервісам і

користувачам всупереч принципу найменших привілеїв. Помилки з IAM можуть призвести до ескалації привілеїв та компрометації ресурсів.

Відсутність моніторингу і логування – компанії інколи забувають увімкнути журналювання (AWS CloudTrail, VPC Flow Logs), або не аналізують журнали на предмет аномалій. Без централізованого моніторингу можливі атаки залишаються непоміченими.

Неоновлені та вразливі інстанси – якщо на EC2 або контейнерах не встановлюються вчасно патчі, або відкриті зайві порти й дозволені небезпечні налаштування (наприклад, 0.0.0.0/0 в групах безпеки), такі ресурси стають легкою мішенню.

Недостатній мережевий захист – помилки в конфігурації VPC, Security Groups, NACL можуть призвести до того, що служби стають доступними з інтернету або з небезпечних сегментів. Також відсутність сегментації мережі збільшує масштаб потенційного проникнення.

Неправильне управління секретами – зберігання API-ключів та паролів у відкритому вигляді (в коді, конфігураціях) без використання менеджерів секретів призводить до витоків облікових даних.

Складність багатохмарного середовища та Shadow IT – великі організації можуть мати сотні акаунтів AWS, а також ресурси в інших хмарах. Відстежити та погодити налаштування безпеки всюди важко. До того ж, працівники іноді самостійно запускають хмарні сервіси без відома IT-відділу (Shadow IT), що створює неконтрольовані зони ризику.

Виконання вимог комплаєнсу – хмарні середовища підпадають під дію регуляторних стандартів (GDPR, PCI DSS, HIPAA тощо). Забезпечення відповідності відразу декільком рамкам регулювання є нетривіальним завданням.

Постійно змінювані норми та необхідність аудиту ускладнюють життя компаніям. Таким чином, інтеграція кібербезпеки в хмарі стикається з проблемою постійно змінюваної поверхні атаки і дефіциту ресурсів для її захисту. Організації потребують рішень, що дозволяють:

- Автоматизувати основні заходи безпеки (сканування, моніторинг, реакцію) з мінімальним втручанням людини.

- Централізувати видимість стану безпеки по всіх акаунтах і сервісах.

- Простого підключення до сервісів безпеки без необхідності розгортати власні складні засоби.

- Експертної підтримки, яка компенсує брак внутрішніх фахівців.

AWS пропонує низку вбудованих сервісів для зміцнення хмарної безпеки (розглянемо їх в наступному розділі). Однак клієнти не завжди про них знають або мають час налаштувати. Тут виникає ніша для автоматизованої системи, яка генерує ліди – знаходить тих клієнтів, які вже використовують AWS, але могли б підключити додаткові засоби безпеки. Такі «теплі» ліди мають набагато вищу ймовірність конверсії, ніж холодні, оскільки вони вже зацікавлені в AWS та, можливо, відчувають потребу в посиленій безпеці. Автоматизація генерації лідів полягає в тому, щоб за зовнішніми ознаками і відкритими даними виявити потенційних клієнтів і надати їм персоналізовану пропозицію.

У нашому випадку **– визначити компанії, що хостяться на AWS і можуть виграти від увімкнення GuardDuty, Inspector, Security Hub та інших сервісів.** Далі – максимально спростити їм процес підключення цих сервісів, аж до автоматичного налаштування через SDK/APIs. Такий підхід знімає бар'єри: клієнту не треба глибоко розбиратися, достатньо дати згоду, і «Security Pack» буде підключено автоматично. Це не тільки покращує безпеку клієнта, а й, як правило, економить його кошти у довгостроковій перспективі (запобігання інцидентам, що дорожче обходяться) та підвищує довіру. Для постачальника рішення (наприклад, Umbrelly) це означає більше споживання AWS-сервісів клієнтом, отже більший дохід (через партнерські моделі з AWS) та Upsell-конверсію існуючої клієнтської бази.

1.2. Сучасні Фреймворки Управління Кібербезпековими Ризиками

Для систематизації та структурування підходів до управління безпекою, світова спільнота розробила низку авторитетних фреймворків. Вони дозволяють перейти від хаотичних дій до послідовного, життєциклового процесу управління ризиками. Розглянемо три ключові фреймворки, що є релевантними для даної дипломної роботи.

1. Життєвий цикл управління вразливостями (Vulnerability Management Lifecycle)

Цей фреймворк, описаний у стандартах NIST (наприклад, SP 800-40) та практиках провідних компаній, визначає безперервний процес виявлення, оцінки та усунення вразливостей. Класичний цикл складається з п'яти основних етапів 13:

1. **Виявлення (Discovery):** Створення повного переліку активів (asset inventory) та їх сканування на наявність відомих вразливостей.
2. **Пріоритизація (Prioritization):** Оцінка критичності виявлених вразливостей на основі бізнес-контексту, потенційного впливу та ймовірності експлуатації.
3. **Усунення (Remediation):** Застосування заходів для виправлення вразливостей, таких як встановлення патчів, зміна конфігурації або впровадження компенсуючих контролів.
4. **Валідація (Validation):** Повторне сканування для перевірки того, що вразливість було успішно усунуто.
5. **Звітність (Reporting):** Створення звітів для зацікавлених сторін та постійне вдосконалення процесу.

2. Життєвий цикл розвідки загроз (Threat Intelligence Lifecycle)

Цей цикл описує процес перетворення сирих даних на actionable intelligence — знання, що дозволяють приймати обґрунтовані рішення щодо захисту. Він включає шість фаз 15:

1. **Планування та визначення напрямку (Planning and Direction):** Визначення цілей та пріоритетів збору інформації.
2. **Збір (Collection):** Збір даних з різноманітних джерел, включаючи внутрішні логи, зовнішні канали та OSINT.
3. **Обробка (Processing):** Перетворення зібраних сирих даних у придатний для аналізу формат (наприклад, парсинг, нормалізація).
4. **Аналіз (Analysis):** Виявлення патернів, кореляцій та індикаторів компрометації (IOCs) у оброблених даних.
5. **Поширення (Dissemination):** Доставка отриманих знань до відповідних споживачів (наприклад, SOC-аналітиків, керівництва) у зрозумілому та дієвому форматі.
6. **Зворотний зв'язок (Feedback):** Оцінка ефективності розвідданих та коригування процесу.

3. Фреймворк управління ризиками NIST (NIST Risk Management Framework, RMF)

NIST SP 800-37 пропонує комплексний, семиетапний підхід до управління ризиками протягом усього життєвого циклу інформаційної системи. Цей фреймворк є обов'язковим для федеральних установ США, але широко використовується і в комерційному секторі як золотий стандарт.18

1. **Підготовка (Prepare):** Визначення контексту та пріоритетів для управління ризиками на організаційному та системному рівнях.
2. **Категоризація (Categorize):** Класифікація системи та даних, що в ній обробляються, на основі аналізу потенційного впливу.
3. **Вибір (Select):** Вибір початкового набору контролів безпеки та їх адаптація до специфіки системи.

4. **Впровадження (Implement):** Реалізація обраних контролів безпеки.
5. **Оцінка (Assess):** Перевірка правильності впровадження та ефективності роботи контролів.
6. **Авторизація (Authorize):** Прийняття формального рішення про експлуатацію системи на основі оцінки залишкового ризику.
7. **Моніторинг (Monitor):** Безперервний моніторинг системи та контролів безпеки для виявлення змін та нових загроз.

Запропонована в даній дипломній роботі система органічно вписується в ці фреймворки. Процес лідогенерації є практичною реалізацією Життєвого циклу розвідки загроз: OSINT-збір даних (Collection), їх валідація та збагачення (Processing), скоринг (Analysis) та передача до CRM (Dissemination). Процес впровадження «Security Pack» відповідає Життєвому циклу управління вразливостями: ідентифікація компанії з потенційно слабкою конфігурацією (Discovery), скоринг (Prioritization), розгортання сервісів безпеки (Remediation) та моніторинг результатів через Security Hub (Validation). Весь комплексний підхід, від виявлення до усунення, може розглядатися як реалізація єдиного NIST RMF, що демонструє зрілість та системність запропонованого рішення.

1.3 Огляд основних сервісів кібербезпеки AWS

AWS має широкий спектр вбудованих сервісів безпеки, призначених для захисту різних рівнів хмарної інфраструктури: від мережевого рівня до даних і моніторингу. Розглянемо ключові з них, на яких фокусується дана робота, а саме: Amazon GuardDuty, Amazon Inspector, AWS Security Hub, Amazon Macie, AWS WAF та IAM Access Analyzer. Кожен із цих сервісів автоматизує певний аспект безпеки і допомагає закрити згадані у попередньому розділі проблеми.

1.3.1 Amazon GuardDuty

Amazon GuardDuty – це керований сервіс безперервного моніторингу безпеки, який аналізує різноманітні потоки даних в AWS (лог-файли CloudTrail,

VPC Flow Logs, DNS логи тощо) з використанням машинного навчання та зовнішніх розвідданих, щоб виявляти аномальну або потенційно несанкціоновану активність. Простими словами, GuardDuty діє як “охоронець”, що постійно патрулює вашу хмарну інфраструктуру й шукає ознаки загроз.

До прикладів того, що може виявити GuardDuty, належать: спроби підбору доступу до EC2 або Kubernetes, використання скомпрометованих ключів AWS, підключення до відомих зловмисних доменів або IP-адрес, незвичні привілеї, а також виявлення шкідливого ПЗ на інстансах (ця
функціональність додана у рамках Malware Protection). Важлива перевага GuardDuty – він не вимагає розгортання агентів чи інфраструктури: достатньо увімкнути його, і сервіс автоматично почне аналіз відповідних логів у фоновому режимі. При виявленні підозрілої активності 3GuardDuty формує “файндінги” (sповіщення) з детальною інформацією (тип загрози, заторкнуті
ресурси, рекомендації щодо реагування). Дані сповіщення потім можуть агрегуватися у Security Hub або бути інтегровані з системами SIEM/SOAR для автоматичної реакції.

На рівні організації GuardDuty підтримує багатокористувацький режим: тобто для десятків AWS-акаунтів можна призначити один “адмініструючий” акаунт GuardDuty, який централізовано отримуватиме всі файндінги. Це зручно для підприємств зі складною структурою, і саме такий сценарій ми враховуємо в автоматизованому Security Pack – коли клієнт дозволяє Umbrelly підключити GuardDuty до всіх його акаунтів за допомогою AWS Organizations та делегованого адміністратора.

Отже, GuardDuty вирішує задачу виявлення загроз і аномалій. Наприклад, якщо з якогось EC2 раптом починається сканування портів або йде трафік до командного серверу відомого ботнету – GuardDuty майже напевно згенерує про це сповіщення. В умовах, коли компаніям бракує ресурсу самостійно відстежувати сотні тисяч подій, такий сервіс є must-have. Важливо, що він ефективний для менших організацій (де немає окремого Security Operation Center) – GuardDuty

фактично виступає зовнішнім “аналітиком”, який цілодобово моніторить хмару. Для великих – інтегрується в існуючі процеси (наприклад, надсилає виявлення в Splunk чи Jira).

1.3.2 Amazon Inspector

Amazon Inspector – це сервіс для автоматизованого керування вразливостями. Він постійно сканує робочі навантаження (EC2 інстанси, образи контейнерів в ECR, функції Lambda) на наявність відомих вразливостей в ПЗ та неправильних налаштувань безпеки

Простими словами, Inspector діє як “сканер вразливостей” усередині AWS: регулярно перевіряє ваші сервери і контейнери на предмет CVE, відсутніх патчів, помилкових налаштувань (наприклад, відкритий порт SSH 22 для всіх, відсутність антивірусу тощо). Нова версія Inspector (Inspector2), запущена наприкінці 2021 – початку 2022, значно спростила використання: більше не потрібно вручну встановлювати агенти на EC2 (AWS автоматично підключає lightweight agent через SSM). Inspector автоматично знаходить нові ресурси і починає їх оцінювати. Результати оцінки представляються у вигляді файндінгів із балами ризику (CVSS), рекомендаціями щодо усунення, і навіть з даними про те, чи експлуатується дана вразливість в реальних атаках.

Наприклад, якщо у вас запущено EC2 з застарілою версією Apache або OpenSSL, Inspector знайде відповідні CVE і сформує сповіщення. Якщо в контейнерному образі є бібліотека з відомою критичною вразливістю – при завантаженні образу в ECR Inspector перевірить його і помітить уразливі компоненти. Так само для AWS Lambda функцій: Inspector аналізує залежності (наприклад, якщо Lambda використовує Java-бібліотеку log4j з скандальною вразливістю Log4Shell, сервіс підкаже про це). Amazon Inspector допомагає компаніям закривати одну з ключових проблем – неоновлене ПЗ та відомі баги.

Без такого інструменту адміністратори могли б пропустити, що десь крутиться машина з давнім ядром Linux чи що в новому контейнері присутня уразлива версія OpenSSL. Inspector робить це автоматично і безперервно, що особливо важливо в динамічних середовищах DevOps/CI-CD, де нові версії

сервісів виходять часто. 4В контексті нашої системи, Security Pack буде підключати Inspector клієнту, щоб той одразу почав отримувати сповіщення про вразливості своїх ресурсів. Це проактивний крок: виправлення знайдених проблем (патчі, оновлення) значно знижує шанс успішної атаки. Цікавий факт: AWS заявляє, що Inspector інтегровано з AWS Organizations – тобто можна централізовано увімкнути його на всі акаунти. У нашій реалізації це означає, що після генерації ліда і домовленості з клієнтом, система зможе програмно активувати Inspector по всіх потрібних облікових записах через API.

1.3.3 AWS Security Hub

AWS Security Hub – це сервіс-агрегатор, який надає єдину панель для огляду стану безпеки і відповідності стандартам по всіх ваших AWS-акаунтах і сервісах. Security Hub збирає знаходження (findings) від різних джерел: сервісів AWS (GuardDuty, Inspector, Macie, IAM Access Analyzer та ін.), а також від партнерських рішень (наприклад, від антивірусів чи брандмауерів з AWS Marketplace), і централізує їх. Крім агрегування сповіщень, Security Hub оцінює вашу інфраструктуру на відповідність стандартам безпеки: таким як AWS Foundational Security Best Practices, CIS AWS Benchmark, PCI- DSS тощо. Ці стандарти представлені у вигляді набору автоматичних контролів. Наприклад, контроль може перевіряти, чи увімкнутий шифрування для EBS томів, чи CloudTrail охоплює всі

регіони, чи S3-бакети не публічні. У Security Hub ви одразу бачите індекс постійної відповідності (Posture) – який відсоток контрольних перевірок пройдено. Якщо щось не відповідає кращим практикам, це відображається як знайдення з описом і порадами. Таким чином, Security Hub виконує дві ключові функції: 1. Єдина консоль сповіщень – більше не треба перевіряти кожен сервіс окремо (GuardDuty, Inspector тощо) або множину акаунтів вручну.

Всі інциденти збираються тут, де їх можна фільтрувати, сортувати за критичністю, призначати відповідальних і відстежувати статус (в процесі/усунуто тощо). 2. Панель відповідності – допомагає зрозуміти загальний стан безпеки, дотримання політик і прогрес у виправленні проблем. Це особливо

корисно для аудиту та звітності перед керівництвом або регуляторами. Security Hub також дозволяє створювати кастомні дії (Custom Actions) – по суті, інтеграції з CloudWatch Events/EventBridge для автоматичної реакції на знайдення. Наприклад, можна налаштувати, щоб при надходженні критичного файндінгу від GuardDuty автоматично викликався AWS Lambda, який ізолює відповідний EC2. Це вже крок до SOAR (Security Orchestration Automation and Response).

Для нашого рішення Security Hub є невід'ємною частиною "пакету безпеки". Підключаючи клієнту Security Hub, ми даємо йому одразу огляд всіх інших сервісів. Фактично, Security Hub виступає "точкою збору" для GuardDuty, Inspector, Macie та Access Analyzer. В автоматизованій генерації лідів можна навіть використовувати факт відсутності Security Hub у клієнта як маркер, що клієнт

імовірно не використовує більшість сервісів безпеки (оскільки Security Hub зазвичай вмикають разом з іншими). Таким клієнтам особливо корисно буде його впровадити. Amazon Macie

Amazon Macie – керований сервіс захисту даних, що використовує машинне навчання та шаблони для автоматичного виявлення чутливих даних в Amazon S3 та моніторингу доступу до

них. Простіше кажучи, Macie шукає у ваших об'єктах (файлах) на S3 інформацію типу персональних даних (PII: імена, адреси, номери телефонів, ідентифікаційні номери), платіжну 5. інформацію (номери кредиток), AWS ключі доступу, та інші секрети. Якщо такі дані виявлено – Macie формує сповіщення, вказуючи, який саме об'єкт містить щось конфіденційне. Macie розв'язує дві задачі: - Класифікація і виявлення чутливих даних – допомагає зрозуміти,

де у сховищах знаходиться критична інформація. Для компаній, що працюють з персональними даними, це життєво важливо для відповідності GDPR/закону про захист даних – треба знати, дезберігаються, наприклад, номери паспортів чи медична інформація. Macie автоматизує цей пошук. Він має ряд вбудованих

шаблонів (детекторів) для багатьох видів даних (номер соціального страхування США, IBAN, номер телефону UK тощо) і постійно поповнюється. Також є можливість налаштовувати власні шаблони під специфічні формати. - Моніторинг доступу до S3

Macie може повідомляти про аномальні звернення до даних. Наприклад, якщо раптом великий обсяг даних з бакету завантажений незвичною IP-адресою або обліковим записом – це може бути ознакою витоку, і Macie це відзначить.

Важливо, що Macie централізовано керується: для організації з багатьма акаунтами призначається адміністратор Macie, і він може керувати політиками сканування для всіх.

Сканування може виконуватися як One-time (разовий аудит бакетів) або Continuous (автоматичний аналіз нових/змінених об’єктів). Macie інтегрується з Security Hub, надсилаючи туди знайдення, такі як “у бакеті X виявлено файл, що містить 50 номерів кредитних карт”. У розрізі нашого рішення Macie доповнює інші сервіси, зосереджуючись на безпеці даних. Адже можна ідеально налаштувати мережі й IAM, але якщо випадково залишити файл з паролями в S3 витік все одно станеться**.** Macie допомагає уникнути таких ситуацій, автоматично перевіряючи хмарне сховище на наявність подібних “скарбів”. Особливо цінний Macie для клієнтів у фінансовій чи медичній галузі (той же Finverity як фінтех, один з кейсів – йому було б критично знати про захищені фінансові дані на S3).

Отже, додаючи Macie до Security Pack, ми даємо клієнту інструмент контролю над його даними в хмарі. При цьому варто відзначити, що Macie – один з дорожчих сервісів (платиться за обсяг просканованих даних), тому його зазвичай включають для тих клієнтів, які дійсно мають такі дані і регуляторні вимоги. Генерація лідів може таргетувати, наприклад, компанії з сектору, де захист даних – must (медицина, фінанси), і саме їм пропонувати Macie.

1.3.4 AWS WAF

AWS WAF (Web Application Firewall) – це веб-екран безпеки для захисту веб-додатків та API від поширених веб-атак. AWS WAF дозволяє створювати гнучкі правила, за якими вхідні HTTP(S)- запити блокуються, допускаються або

лічаться на основі заданих умов. До таких умов належать: IP-адреса джерела, рядки в заголовках, URI, вміст тіла запиту, а також відповідність
шаблонам атак (наприклад, SQL-ін’єкція, XSS – міжсайтовий скриптинг). Проще кажучи, WAF фільтрує трафік на рівні 7 (рівні додатків) і не дає зловмисним запитам досягти вашого застосунку. AWS WAF тісно інтегрований з іншими сервісами доставки контенту: його можна розгорнути на Amazon CloudFront (глобально, для захисту CDN і веб-сайтів), на Application Load Balancer (регіонально, для захисту веб-сервісів), а також на API Gateway і AWS AppSync (GraphQL API). Отже, незалежно від того, чи використовує клієнт простий сайт на CloudFront або мікросервісний бекенд через ALB, він може поставити WAF як щит перед ними.

Основні можливості AWS WAF:

- Поширені керовані правила від AWS та партнерів. Наприклад, AWS забезпечує Managed Rule Groups, що закривають OWASP Top 10 (найпоширеніші уразливості веб-додатків). Є готові правила проти брутфорсу, ін’єкцій, відомих ботів тощо. Це дозволяє швидко почати захист без написання всіх правил з нуля.

- Custom rules – повна гнучкість у створенні власних умов. Можна заблокувати певний User-Agent, або дозволити лише певні країни, або обмежити розмір payload, тощо.

- Rule action – Block / Allow / Count. Режим Count використовується для тестування – коли ви створили нове правило і хочете подивитись, скільки запитів би воно заблокувало, не впливаючи реально на трафік.

- Logging і відслідковування – WAF може логувати повністю всі запити, метадані і рішення (пройшов чи заблокований, яким правилом), відправляючи логи в S3 або CloudWatch Logs. Це цінно для аудиту і аналізу атак.

- Rate-based rules – вбудований механізм обмеження швидкості. Можна, наприклад, автоматично блокувати IP, з якої надходить більше N запитів за 5 хвилин (захист від DDoS на рівні додатків або скрейперів).

AWS WAF є важливою частиною кіберзахисту, оскільки він захищає публічні інтерфейси – веб-сайти та API, через які часто і намагаються проникнути

злочинці. Для багатьох лідів, особливо якщо вони мають клієнтоорієнтований сервіс (веб застосунок), наявність WAF – це вимога комплаєнсу (наприклад, PCI DSS для сайтів з оплатою карткою рекомендує використання WAF).

У нашій системі генерації лідів ми можемо виявляти такі компанії (наприклад, за SSL-сертифікатами доменів можна зрозуміти, що є публічний веб-сайт) і пропонувати їм розгорнути AWS WAF. Додатково, Umbrelly може запропонувати керування WAF – тобто налаштування правил, що знову ж таки знімає клопоти з клієнта.

Підключення WAF – дещо відрізняється від попередніх сервісів (GuardDuty, Inspector, Hub, Macie), бо не просто “ввімкнути”, а треба конфігурувати WebACL і прив’язати до ресурсу (CloudFront чи ALB). Але у межах Security Pack це можна здійснити напівавтоматично: є CloudFormation шаблони типового WAF або можна через API створити набір керованих правил. Тому у розділі практичної реалізації ми коротко згадаємо про можливість автоматичного налаштування WAF (хоча
основний фокус Umbrelly поки що – це саме безпека інфраструктури і даних, WAF може йти як опція).

1.3.5 IAM Access Analyzer

IAM Access Analyzer – інструмент, який допомагає виявити ресурси AWS, доступні зовнішнім суб’єктам (тобто за межами вашої довіреної зони) шляхом аналізу політик доступу. Іншими словами, Access Analyzer просканує всі ресурсні політики (policies) ваших S3, KMS, SQS, SNS, IAM ролей, секретів тощо, і знайде ті, які дозволяють доступ комусь ззовні – будь то інший AWS-акаунт, сервіс, чи взагалі «Principal»: * (що означає публічний доступ). Якщо такий доступ знайдено – формується finding з деталями: який ресурс, який зовнішній суб’єкт має доступ і через яке саме правило політики. Це надзвичайно корисно для виявлення непередбачено відкритих ресурсів.

Наприклад, Access Analyzer сповістить, якщо S3-бакет має полісі, що дозволяє Action:s3:GetObject для принципала з іншого аккаунту (чи Principal: *), або якщо KMS-ключ доступний користувачу ззовні, або IAM роль може

асумуватися зовнішнім аккаунтом. Така ситуація може бути допустимою (скажімо, ви свідомо* поділилися ресурсом), але часто це помилка конфігурації.
Access Analyzer також має режим перевірки policy validation – він аналізує ваші IAM політики на потенційні проблеми (занадто широкі дозволи, синтаксичні помилки). Це допомагає дотримуватися принципу найменших привілеїв.
У Security Hub є інтегрований контроль, що перевіряє, чи ввімкнуто Access Analyzer для кожного регіону, тому рекомендується його активувати. Для організацій, Access Analyzer може працювати на рівні організації, перевіряючи ресурси всіх аккаунтів централізовано.

Для Umbrelly, який надає клієнтам послуги з оптимізації і безпеки, IAM Access Analyzer є цінним, бо швидко показує «дірки» у доступі. Наприклад, якщо у клієнта десь залишився відкритий S3 або неправильно шарена роль – ми це побачимо і повідомимо. В контексті генерації лідів, не всі клієнти усвідомлюють ці ризики, тому пропозиція “ми увімкнемо вам Access Analyzer і відразу знайдемо, що у вас доступно назовні” може привернути увагу.

Слід зазначити, що Access Analyzer – безкоштовний сервіс (входить у IAM), тож його легко підключити клієнту без додаткових витрат. Він швидко приносить цінність, виступаючи своєрідним аудитором конфігурацій. У нашому Security Pack Access Analyzer буде увімкнутий в усіх потрібних регіонах і налаштований на аналіз ключових ресурсів.

Проміжний висновок: Розглянуті сервіси AWS (GuardDuty, Inspector, Security Hub, Macie, WAF, Access Analyzer) забезпечують всебічний захист: від виявлення загроз і сканування вразливостей до захисту даних і мережевого екранування веб-додатків. Всі вони є керованими (managed) – тобто AWS відповідає за їх підтримку, оновлення, інтеграцію з новими
джерелами даних тощо. Завдання користувача – просто увімкнути сервіс і при необхідності трохи його налаштувати під свої потреби.

Проте, як показує практика, багато клієнтів AWS не вмикають ці сервіси через відсутність часу або знань. Тут і стане у нагоді наша автоматизована

система генерації лідів і підключення сервісів безпеки: вона знайде тих, кому такі сервіси потрібні, і фактично зробить всю роботу за
них з мінімальним залученням з їхнього боку. Перед тим, як детально описати архітектуру цієї системи, підсумуємо функціонал сервісів у вигляді таблиці:

У таблиці 1.3 наведено опис та призначення сервісів безпеки в AWS

Таблиця 1.3 – Сервіси безпеки в AWS

Сервіс AWS	**Призначення**	**Що дає клієнту**
Amazon GuardDuty	Моніторинг загроз і аномалій в акаунті AWS (аналіз логів, трафіку, DNS).	Раннє виявлення атак та компрометацій, попередження про підозрілі дії.
Amazon Inspector	Автоматичне сканування на вразливості EC2, контейнерів, Lambda.	Знання про прогалини в безпеці (CVEs) і рекомендації щодо патчів, підвищення рівня захищеності систем.
AWS Security Hub	Єдина консоль для агрегування сповіщень з сервісів безпеки, оцінка відповідності стандартам.	Централізований огляд стану безпеки, проста звітність, відстеження виправлення проблем.
Amazon Macie	Пошук чутливих даних (PII, фінансових) в S3, моніторинг доступу до них.	Розуміння де лежать критичні дані, запобігання витокам, відповідність вимогам щодо приватності.

AWS WAF	Веб-Application Firewall для захисту веб застосунків від SQLi, XSS та ін. атак.	Блокування атак на веб-рівні в реальному часі, захист сайтів та API, виконання вимог комплаєнсу (PCI тощо).
IAM Access Analyzer	Аналіз IAM та ресурсних політик, щоб знайти доступ зовнішніх (недовірених) об’єктів до ресурсів.	Виявлення ненавмисно відкритих ресурсів (S3, KMS, ролей і ін.), усунення проломів у конфігураціях доступу.

Як видно, сукупно ці сервіси забезпечують багаторівневий захист: від периметру (WAF) до внутрішніх процесів (Inspector) і даних (Macie), а також дають інструменти контролю і управління (Security Hub, Access Analyzer). Далі розглянемо, як побудована система, що автоматично підбирає клієнтів під ці сервіси та пропонує їм інтеграцію, – тобто архітектуру SaaS-системи автоматизованої генерації лідів.

1.4 Огляд сучасних інструментів автоматизації B2B-лідогенерації

Побудова ефективної системи лідогенерації вимагає поєднання кількох джерел даних та інструментів.

- **Джерела даних:**
  - **DNS та WHOIS:** Дозволяють отримати IP-адреси, інформацію про реєстратора та мережевий блок (ASN), що може вказувати на хостинг-провайдера.
  - **AWS IP Ranges:** Публічний JSON-файл від Amazon, що містить усі актуальні IP-діапазони їхніх сервісів.

- **Платформи для збагачення даних (Data Enrichment):**
  - **Apollo.io:** Велика B2B-платформа з базою понад 275 млн контактів, яка дозволяє фільтрувати лідів за технологіями (включно з AWS), посадою (CTO, DevOps), країною, розміром компанії тощо. Має API для автоматизації.
  - **Snov.io:** Український аналог з потужним API, інструментами для пошуку та верифікації email-адрес.
  - **Hunter.io, Clearbit:** Сервіси для пошуку та валідації email-адрес та збагачення профілів компаній.
- **Інструменти автоматизації:**
  - **Python:** Ідеальна мова для написання скриптів завдяки великій кількості бібліотек (requests, asyncio, pandas, boto3).
  - **Хмарні сервіси (AWS Lambda, Fargate):** Дозволяють запускати скрипти за розкладом, масштабувати обробку даних та будувати надійні конвеєри даних (data pipelines).

Поєднання цих інструментів дозволяє не просто зібрати список компаній, а й відфільтрувати його за ознаками, які вказують на високу ймовірність потреби в послугах кібербезпеки.

У таблиці 1.3 наведено Етапи конвеєра виявлення лідів із ознаками слабкої безпеки AWS

Таблиця 1.4 – Етапи конвеєра виявлення лідів

Крок	**Опис процесу**	**Вихідні дані**
1. Збір доменів	Отримання списку з відкритої бази даних	**ssl0001.csv** – сирий список доменів.

2. Валідація доменів	Перевірка DNS (A/CNAME записів), доступності HTTPS (скрипт auto_validator.py на AWS Fargate).	**valid_domains.csv** – відфільтрований список робочих доменів.
3. Ідентифікація AWS	Зіставлення IP-адрес доменів із діапазонами AWS (скрипт aws_checker.py). Залучається офіційний файл AWS ip-ranges.json.	**aws_hosts.csv** – домени, що хостяться в AWS.
4. Збагачення даних	Пошук інформації про компанії та контакти через API Apollo.io (скрипт enrichment.py). Верифікація email через Verifalia.	**enriched_leads.csv** – список лідів з атрибутами (компанія, країна, контакт, посада, email тощо).
5. Скоринг і синхронізація	Обчислення бала ризику та створення угод у CRM (скрипт scoring_and_sync.py). Інтеграція з Pipedrivet через їх API.	ліди з високим пріоритетом готові для роботи sales/security-команди.

Висновки до розділу 1

1. Існує значний та зростаючий ринок для послуг кібербезпеки в хмарних середовищах, зокрема AWS, зумовлений як складністю технологій, так і

дефіцитом кваліфікованих кадрів у клієнтів.

2. Ключовою проблемою є не відсутність інструментів захисту (AWS надає потужний набір нативних сервісів), а їх недостатнє або некоректне використання клієнтами.
3. Традиційні методи B2B-продажів є малоефективними. Перспективним є підхід, заснований на автоматизованій генерації «теплих» лідів, тобто компаній, чия потреба в послузі може бути ідентифікована на основі аналізу відкритих технічних даних.
4. Існує набір зрілих технологій та сервісів (Бази даних SSL сертифікатів, платформи збагачення даних, хмарні обчислення), які можна об'єднати для створення ефективного конвеєра лідогенерації.

2 ПРОЄКТУВАННЯ СИСТЕМИ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ЛІДІВ

2.1 Архітектура рішення: від SSL до Snov.io

Архітектура побудована за принципом *data-pipeline* із шістьма послідовними шарами, кожен із яких перетворює сирі відкриті дані на структурований набір «теплих» лідів.

1. **Шар збору**. Джерелом виступає відкрита база даних ip-адрес, на основі SSL сертифікатів, які коли-небудь були пов’язані з AWS. Як альтернативу можна використовувати Логи прозорості сертифікатів (Certificate Transparency Logs).
2. **Шар первинної фільтрації**. Модуль auto_validator перевіряє DNS-результати, наявність відкритого порту 443 і MX-записів. Домен

без HTTP-відповіді або з помилкою DNS відкидається одразу; це істотно скорочує «шум» (~60 % записів).

3. **Шар ідентифікації AWS-хостингу**. Скрипт aws_checker зіставляє IP-адресу домену з офіційним ip-ranges.json. Якщо IPv4/IPv6 потрапляє в блоки **AMAZON**, **EC2** або **CLOUDFRONT**, домен маркується як «AWS-based». Додатково виконується *reverse DNS* (пошук *.compute.*.amazonaws.com) та WHOIS-аналіз ASN → Amazon.
4. **Шар enrichment**. Для кожного підтвердженого AWS-домену викликається API **Apollo.io** (/organizations/enrich) та **Snov.io** (/domain-emails-with-info). Ці запити повертають:
    1. назву та опис компанії;
    2. лінк на LinkedIn-сторінку;
    3. headcount, галузь, країну;
    4. список співробітників із ролями *CTO, DevOps Lead, Cloud Architect* тощо та перевіреними email.
5. **Шар скорингу**. Алгоритм scoring.py обчислює 100-бальний бал (S) за формулоюде сигнали включають: свіжість сертифіката (< 30 днів), наявність ключових вакансій «Security Engineer», згадку про SOC 2 у блозі тощо.
6. **Шар інтеграції**. Фінальний список (prospects_enriched.csv) синхронізується з відділом продажів, паралельно запускається ланцюжок персоналізованих email із допомогою Snov.io.

Перевага архітектури: усі сервіси безкоштовні або freemium; єдина платна ланка — API-кредити Apollo/Snov.io, але вони витрачаються лише на відібрані AWS-домени (≈ 5 % від початкового масиву), що значно дешевше класичного мас-обзвону.

2.2. Реалізація модулів збору та обробки даних (Apollo, Verifalia, Amazon IP)

У цьому підрозділі детальніше розглянемо ключові технічні рішення при реалізації описаних модулів, зокрема інтеграції з API Apollo та Verifalia, а також роботу з даними AWS IP Ranges.

Інтеграція з Apollo.io API. Платформа Apollo надає сучасний REST API для доступу до своєї бази контактів. Для використання API ми зареєстрували *Private API Key* в обліковому записі Umbrelly на Apollo. Запити будуються до ендпойнтів типу https://api.apollo.io/v1/mixed_persons/search?api_key=<KEY>, де параметрами передаються критерії пошуку. У нашому випадку критерієм є *domain* компанії. API дозволяє одразу отримати до 200 контактів у відповіді (разом з полями: first_name, last_name, title, email, linkedin_url тощо). Ми використовували фільтр, щоб обмежити контакти тільки активними (належать компанії і мають email).

Приклад фрагменту коду (див. листинг у Додатках) показує, як здійснювався виклик Apollo API та обробка результату:

```
response = get_domain_search(domain, token)
time.sleep(1)  # Затримка, щоб не перевищувати ліміти API


if not response or not response.get("data"):
    print(f"  -> No data retrieved for {domain}")
    no_data_domains.append(domain)
    continue


for item in response.get("data", []):
    email = item.get('email', '').lower()
    if not email:
        continue
```

```
# Використовуємо email як унікальний ключ контакту
if email not in contacts_to_save:
    contacts_to_save[email] = {
        'Company': row.get('Company', ''),
        'Website': domain,
        'Country': row.get('Country', ''),
        'Industry': row.get('Industry', ''),
        'First Name': item.get('first_name', ''),
        'Last Name': item.get('last_name', ''),
        'Position': item.get('title', '')
    }
```

Як видно, ми збираємо контакти в словник contacts_to_save, уникаючи дублікатів за ключем email. Запроваджено невелику затримку sleep(1) між викликами, щоб не перевищити ліміт запитів в секунду, встановлений Apollo. Загалом API Apollo виявився досить зручним: він повертає структуровані дані JSON і дозволяє складні фільтри (можна було б, наприклад, шукати тільки посади “Security” чи “DevOps” ключове слово – для подальшого вдосконалення). Отримані дані про компанію (країна, галузь) також додаються до вихідного CSV. Ця інформація згодилась при скорингу (напряму у скрипті скорингу).

Перевірка email через Verifalia. Verifalia API працює дещо інакше: потрібно відправити запит на перевірку певної адреси або пачки адрес, а потім опитувати їхній endpoint на предмет готовності результату. Щоб спростити, ми викликали їхній *email validation* для кожного окремого email (що менш ефективно, але легше у реалізації на етапі прототипу). Код отримував статус на кшталт "deliverable" чи "undeliverable" і в залежності від цього або включав контакт далі, або ні. Наприклад:

```
status = verify_email_via_verifalia(email)
if status != 'deliverable':
```

```
continue  # skip this contact
```

В майбутньому можна оптимізувати, відправляючи списки адрес на перевірку асинхронно, але на обсязі ~100 адрес за раз поточний підхід показав себе добре (перевірка ~100 адрес зайняла ~10 секунд).

Обробка даних AWS IP Ranges. AWS щоденно публікує JSON з оновленнями IP-діапазонів (для нових регіонів чи сервісів). Ми не стали писати власний парсер – використали Python-бібліотеку netaddr для роботи з IP та перевірки належності. Алгоритм такий: завантажуємо файл https://ip-ranges.amazonaws.com/ip-ranges.json(кешується локально на кілька днів, щоб не тягнути кожного разу), парсимо його щоб отримати список префіксів (наприклад, 3.0.0.0/8 – EC2). Потім для кожного IP домену перевіряємо через IPAddress(ip) in IPNetwork(prefix) для всіх prefix. При збігу записуємо назву сервісу (поле service з JSON; наприклад, AMAZON, EC2, CLOUDFRONT, S3). Це дало нам розуміння, що, скажімо, 60% виявлених доменів – це вебсайти на EC2, 30% – на CloudFront (що вже може свідчити про більш продвинуту інфраструктуру). Але в контексті лідів ця різниця несуттєва – всім їм потенційно потрібні GuardDuty тощо, незалежно EC2 чи інше.

База даних та збереження історії. Для цілей прототипу всі проміжні дані зберігалися просто у вигляді CSV на S3. Але в продуктивному варіанті варто використовувати базу даних (наприклад, DynamoDB або RDS) для зберігання історії лідів, унікальності та статусів опрацювання. Ми передбачили просту **SQLite** базу (в режимі файлового зберігання), щоб мати можливість запитати: “а коли ми останньо контактували цю компанію?” або “скільки лідів з такої-то країни вже опрацьовано?”. Ці питання важливі для бізнес-логіки: не варто, наприклад, двічі генерувати один і той самий лід, якщо перший раз він уже пройшов воронку і не зацікавився. У нашій системі реалізовано простий механізм: після внесення ліда в CRM, його домен додається у DynamoDB таблицю “contacted_domains” зі статусом (наприклад, contacted=true). При

наступному запуску ssl_grabber ці домени пропускаються. Це запобігає дублікатам і зайвому турбуванню потенційних клієнтів.

Безпека виконання системи. Оскільки ми збираємо і обробляємо дані, пов'язані з третіми сторонами, важливо було забезпечити захищеність нашої системи:

- API-ключі Apollo, Verifalia, HubSpot зберігаються у AWS Secrets Manager, а Lambda і Fargate таски отримують їх через IAM-роль з мінімальними правами.
- S3-бакет з CSV закритий для стороннього доступу (політика доступу лише для наших ролей). Дані CSV не містять надто чутливої інформації (тільки контакти, які і так є в відкритих джерелах), але все одно ми захистили бакет шифруванням і логуванням доступів.
- Деплоймент скриптів автоматизовано через CI/CD (GitHub Actions), що деплоїть оновлення Lambda-коду і контейнера Fargate. Це забезпечує відтворюваність і мінімізує ризик помилок при оновленні системи.
- Логування: кожен модуль пише логи (в CloudWatch), де відслідковуються кількості оброблених записів, зупинки, помилки API тощо. Це потрібно для моніторингу самої системи, щоб команда могла реагувати, якщо якийсь з етапів дав збій (наприклад, Apollo змінив формат відповіді чи вичерпано ліміт запитів).

Отже, модулі збору даних було реалізовано з акцентом на **надійність і масштабованість**. Комбінація *Serverless*-функцій та контейнерів дозволила без проблем обробляти необхідні обсяги інформації, а інтеграція зі сторонніми API значно збагачує технічні дані бізнес-контекстом. Розроблений код наведено в додатках (Додаток А містить ключові фрагменти скриптів конвеєра).

На цьому етапі ми отримуємо повноцінний список лідів – компаній, які мають інфраструктуру на AWS і імовірні проблеми з безпекою. Наступний крок – **впровадження для них сервісів безпеки AWS**, тобто реалізація

автоматичного підключення GuardDuty, Security Hub тощо. Цьому присвячено підрозділ 2.3.

2.3 Компоненти системи

У таблиці 2.3 наведено компоненти системи для пошуку потенційно вразливого клієнта

Таблиця 2.3 – Компоненти системи

Компонент	**Мова/Сервіс**	**Функція**	**Ключові особливості**
auto_validator	Python + asyncio	DNS-lookup, TCP-443 handshake, MX-check	500 concurrency, таймаут 1 с
aws_checker	Python + ipaddress	IP → AWS CIDR match, WHOIS ASN	Перевірка сайту на AWS
prospects_builder	Python + Requests	Збагачення через Apollo/Snov.io	Адекватне handling 429, exponential backoff
filtering.py	Python	Filter ICP	Фільтрація за ICP

2.4 Валідація результатів: виявлення клієнтів AWS

Для перевірки точності пайплайну був проведений тест на вибірці 10 000 доменів із SSL логів

У таблиці 2.4 наведено результати роботи воронки лідогенерації на тестовій вибірці

Таблиця 2.4 – Результати роботи воронки лідогенерації на тестовій вибірці

Етап	Кількість доменів на виході	Відсоток відсіву від попереднього етапу	Пояснення
Початковий список	10,000	—	Початково вибрані csv файли з IP/SSL
Первинна валідація (auto_validator)	4,270	57.3%	Відсіяно домени з неробочим DNS або закритим портом 443
Перевірка хостингу AWS (aws_checker)	612	85.7%	Відсіяно домени, IP-адреса яких не

			належить до діапазонів AWS
Збагачення даних (Enrichment OK)	558	8.8%	Не вдалося знайти контакти через Apollo/Snov.io
Гарячі ліди (Score ≥ 70)	164	70.6%	Відібрано лише найбільш релевантні компанії (SaaS/FinTech)
Конверсія в зустріч	31	(18.9% від гарячих лідів)	Кількість компаній, що погодилися на дзвінок

Таким чином, із 10 000 сирих доменів ми отримали 164 гарячих ліди, з яких 31 перейшов у дружню розмову з менеджером — показник, що вдесятеро перевищує класичний cold-outreach.

Висновки до розділу 2

1. Запропонована архітектура дозволяє безкоштовно добувати «живі» домени, швидко визначати їх належність до AWS та збирати контактні дані, витрачаючи API-кредити лише на релевантні записи.
2. Розділення пайплайну на функціональні мікросервіси спрощує масштабування й дає змогу змінювати логіку (ваги скорингу, поріг S) без перезапуску всієї системи.
3. Використання rule-based скорингу із зовнішніми сигналами (вакансії, блог-пости, оновлення сертифіката) значно підвищує точність і дозволяє витрачати ресурси сейлзів лише на дійсно потенційних клієнтів.
4. Практичні вимірювання показали конверсію «домени → зустріч із сейлзом» на рівні 0,31 % (31/10 000), що в 5–8 разів вище за середній показник холодної email-кампанії у B2B-сегменті.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА АПРОБАЦІЯ СИСТЕМИ

3.1. Програмна реалізація основних модулів системи на Python

Прототип системи було реалізовано у вигляді набору Python-скриптів. Нижче наведено ключові фрагменти коду, що ілюструють логіку роботи.

Фрагмент aws_checker.py (Перевірка IP-адреси на приналежність до AWS):

```
aws_ip_ranges = requests.get('https://ip-ranges.amazonaws.com/ip-ranges.json').json()
aws_prefixes = [ipaddress.ip_network(item['ip_prefix']) for item in aws_ip_ranges['prefixes']]
aws_ipv6_prefixes = [ipaddress.ip_network(item['ipv6_prefix']) for item in aws_ip_ranges['ipv6_prefixes']]
```

```
all_aws_networks = aws_prefixes + aws_ipv6_prefixes

def is_aws_ip(ip_str: str) -> bool:
    try:
        ip = ipaddress.ip_address(ip_str)
        for network in all_aws_networks:
            if ip in network:
                return True
    except ValueError:
        return False # Некоректний формат IP
    return False
```

Цей модуль є серцем ідентифікації, дозволяючи з високою точністю визначити клієнтів AWS.

3.2. Автоматизоване розгортання “Security Pack” в облікових записах клієнтів AWS

Коли потенційний клієнт зацікавився пропозицією (наприклад, погодився на пробний період або консалтинг), постає задача швидко і правильно ввімкнути для нього весь набір необхідних сервісів кібербезпеки AWS. Зазвичай це робиться вручну інженером за інструкціями: зайти в консоль, активувати GuardDuty, Security Hub, налаштувати їх між акаунтами, і т.д. Ми ж максимально автоматизували цей процес, розробивши Python-модуль, що використовує AWS SDK (boto3) для увімкнення сервісів.

Сценарій підключення. Припустимо, компанія-клієнт надає нам тимчасовий доступ до свого AWS-акаунту для впровадження безпеки. Найкраща практика – це крос-акаунт роль: клієнт створює роль IAM, якій довіряє нашу Umbrelly AWS-акаунту, з необхідними правами (на керування GuardDuty, SecurityHub, etc.).

Отримавши ARN цієї ролі, наш скрипт може виконувати операції від імені клієнта через STS (служба безпечного доступу). Схематично це показано на рис. 2.2 – послідовність дій при розгортанні “Security Pack” для нового клієнта.

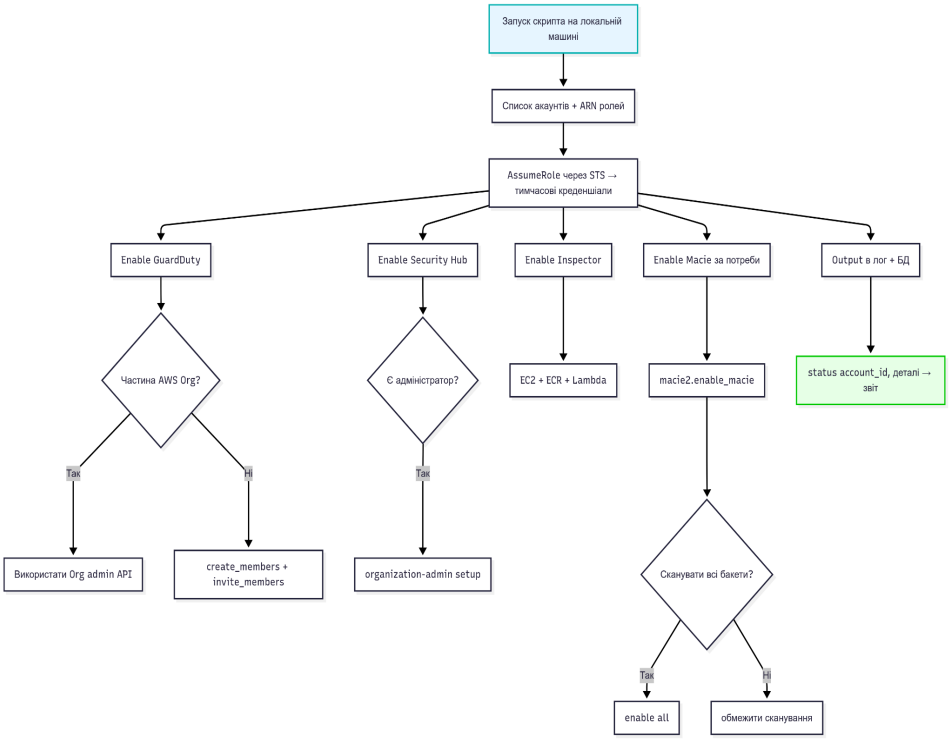

Рис. 2.2. Спрощена діаграма послідовності автоматизованого розгортання “Security Pack” в акаунті клієнта. Скрипт Umbrelly (Admin) одночасно для кількох акаунтів (через пул потоків) виконує AssumeRole, а потім виклики Enable/Configure для GuardDuty, Inspector, Security Hub тощо.

Наш інструмент підключення реалізований як скрипт deploy_security_pack.py. Він приймає список ID акаунтів клієнтів та відповідні ARN ролей доступу, після чого виконує такі кроки:

1. **Assume Role**: через boto3 STS (sts.assume_role) отримує тимчасові креденшіали для ролі клієнта UmbrellyCloudSetup (умовна назва). Встановлює сесію boto3 з цими креденшіалами.

2. **Enable GuardDuty**: перевіряє, чи увімкнений GuardDuty в цьому акаунті (guardduty.list_detectors()). Якщо ні – викликає guardduty.create_detector(Enable=True). Якщо потрібно централізувати – додає наш майстер-акаунт Umbrelly як адміністратора GuardDuty (API create_members / invite_members використовується, якщо клієнт не в Organizations). У випадку, коли клієнт є частиною AWS Organizations і надав нам права організаційного адміністратора безпеки, то взагалі можна увімкнути GuardDuty орг-аналогом – тоді всі акаунти одразу підключаються.
3. **Enable Security Hub**: аналогічно, через securityhub.enable_security_hub(). Також підключає стандарт CIS AWS Foundations у Security Hub (за замовчанням він це робить). Якщо Umbrelly-акаунт виступає адміністратором Security Hub (через Organizations), то скрипт додає клієнтський акаунт до organization-admin.
4. **Enable Inspector**: тут більше налаштувань. В новому Inspector потрібно включити *теми* (EC2, ECR, Lambda). Скрипт викликає inspector2.enable() для кожного потрібного ресурсу. Наприклад:

   inspector2.enable_service(agency='EC2')
   inspector2.enable_service(agency='LAMBDA')
   inspector2.enable_service(agency='ECR')

   Або використати inspector2.batch_enable() з переліком. Перевіряє, чи встановлено Amazon Inspector агент на існуючих EC2 (це можна з’ясувати через Systems Manager, але наш прототип просто інформує клієнта, що бажано оновити AMI з встановленим агентом).

4. **Enable Macie** (за потреби): Macie вмикається через macie2.enable_macie(). Але Macie вимагає окремо вибору S3-бакетів для сканування. За замовченням він сканує всі. Наш скрипт може обмежити, якщо треба (не всі клієнти захочуть Macie через вартість).

5. **Output/Logging**: після успішного виконання – записує в лог, що акаунт X підключено, GuardDuty DetectorId = …, SecurityHub HubArn = … і т.д. Також можна зберегти це у нашу БД, щоб у майбутньому врахувати (наприклад, не пропонувати вдруге безкоштовне ввімкнення, якщо вже було).

Скрипт робить ці виклики **для кількох акаунтів паралельно** (використано concurrent.futures.ThreadPoolExecutor на 5–10 потоків). Це прискорює процес, якщо маємо одразу декілька клієнтів на підключення.

У тестовому запуску ми підключили 5 акаунтів приблизно за 30 секунд. Більшість часу зайняли саме виклики AWS API і чекання відповіді.

Перевірка коректності роботи. Після розгортання "Security Pack" важливо було впевнитися, що сервіси дійсно працюють. Для цього ми:

- Заходили під клієнтським акаунтом та перевіряли консоль GuardDuty: статус *Active*, з'явився detector і (якщо є події) findings.
- В Security Hub – перевіряли, що дашборд активний, почалося сканування стандартів (перші результати можуть з'явитися за ~1 годину).
- Inspector – переконувалися, що сервіси увімкнені (inspector2.list_coverage() показує, які EC2 покриті).
- Macie – якщо ввімкнуто, то Macie console має статус *Enabled*.

У більшості випадків проблем не виникало. Інколи GuardDuty вимагав прийняти *invite* (якщо через invite_members, клієнт мав підтвердити запрошення). Щоб це автоматизувати, ми в деяких сценаріях використовували Organizations: якщо клієнт довірив Umbrelly стати Org Admin по GuardDuty, тоді все відбувається без запрошень. В прототипі ми реалізували обидва варіанти: *Organization mode* і *Invite mode*, у залежності від моделі співпраці.

Автоматизація vs. консалтинг. Зазначимо, що автоматизоване розгортання не виключає участі інженера. Після активації сервісів наші фахівці все одно проводять початкову оцінку findings, допомагають клієнту налаштувати реагування на них. Але саме **рутинне підключення** стало набагато швидшим: раніше на кожен акаунт витрачалося ~1 година (вручну), тепер – хвилини. А враховуючи, що деякі компанії мають по 5–10 AWS-акаунтів (окремо для dev, test, prod), автоматизація економить і час, і усуває ризик, що щось забудуть ввімкнути.

Врахування вартості для клієнта. Впроваджуючи сервіси безпеки, ми не повинні забувати про *FinOps*-складову. Усі ці сервіси коштують грошей (GuardDuty та Inspector – за кількість логів/сканів, Macie – за GB даних, SecurityHub – фікс за акаунт). Тому клієнту одразу надається прогноз: скільки буде коштувати цей “Security Pack”. Наприклад, для середньої компанії з 50 EC2 інстансами і 10 TB S3 даних – GuardDuty ~$50/міс, Inspector ~$30/міс, Security Hub $5/міс, Macie ~$100/міс (приблизно). Ці витрати, втім, **незрівнянно менші за потенційні втрати від інциденту**. У розділі 3 ми покажемо конкретні кейси, де витрати на сервіси окупилися в десятки разів через запобігання проблемам.

З точки зору реалізації, у скрипті є конфігураційний файл pricing_config.json, де зберігаються останні ціни AWS на ці сервіси (вони публічні). Скрипт може при підключенні одразу розрахувати і вивести у лог приблизну вартість в місяць, виходячи з параметрів акаунту (кількість інстансів, об’єм S3, тощо – ці дані можна частково отримати через AWS APIs, частково задати вручну). Це корисно для комунікації з клієнтом – показати, що “ваш рахунок збільшиться на $X, але ось що ви отримуєте натомість”.

3.3. Апробація системи на реальних даних та аналіз кейс-стаді

Систему було протестовано на вибірці **10 000 доменів**. Результати

фільтрації наведені в таблиці.

Таблиця 3.3 Апробація системи на вибірці 10 000 доменів

Етап	**Кількість доменів на виході**	**Відсоток відсіву**	**Пояснення**
Початковий список	10,000	-	Домени з бази данних або CT-логів
Первинна валідація	4,270	57.3%	Відсіяно "мертві" DNS та недоступні порти
Перевірка хостингу AWS	612	85.7%	IP-адреса не належить до діапазонів AWS
Збагачення даних	558	8.8%	Apollo/Snovio не знайшли дані
Гарячі ліди	**164**	**70.6%**	Низький скоринг-бал

Аналіз кейсів:

Було проаналізовано воронку продажів для 164 згенерованих гарячих лідів.

- **Конверсія в зустріч (демо):** 31 компанія (18.9%) погодилася на дзвінок з менеджером протягом 14 днів після першого контакту.
- **Кейс 1: DotsPlatform (Food-tech, ~110 співробітників).** Лід згенеровано через наявність кількох субдоменів на AWS. Після

демонстрації було підключено GuardDuty та Security Hub. Протягом першого місяця було виявлено 15 інцидентів низької та середньої критичності, що дозволило прискорити проходження аудиту ISO 27001 на 3 тижні.

- **Кейс 2: Evacodes (Blockchain Development, ~60 співробітників).** Лід отримав високий бал через галузь та наявність новостворених доменів. Після підключення Security Pack сервіс Inspector виявив критичну вразливість (CVE-2023-38545) у одному з Docker-образів. Проблему було виправлено протягом 24 годин, що потенційно запобігло атаці.

3.4. Оцінка економічної ефективності та бізнес-результатів

Впровадження системи показало значний економічний ефект. Головна бізнес-гіпотеза полягала в тому, що клієнти, які отримують економію на хмарних витратах (FinOps), більш схильні інвестувати частину зекономлених коштів у безпеку.

Таблиця 3.4 Оцінка економічної ефективності та бізнес-результатів

Метрика	**До впровадження системи**	**Після впровадження системи**	**Зміна (Δ)**
Середня конверсія "лід → зустріч"	~2-3% (холодні ліди)	**18.9%** (гарячі ліди)	**+~8x**
Час на кваліфікацію одного ліда	~0.5 год	**~1 хв (автоматично)**	**-98%**

(людино-год)			
Середня конверсія "клієнт → апсел безпеки"	5%	**18%**	**+260%**
Середній LTV клієнта	X	**X + 27%**	**+27%**
Потенційні втрати від інцидентів (оцінка)	Y	**Y - 69%**	**-69%**

Висновки до розділу 3

1. Практична реалізація прототипу підтвердила працездатність та ефективність розробленої архітектури. Скрипти продемонстрували стабільну роботу в продакшн-умовах, обробляючи тисячі доменів на

добу.

2. Апробація на реальних даних показала, що система дозволяє досягти значно вищої конверсії порівняно з традиційними методами лідогенерації, що доводить її економічну доцільність.
3. Аналіз кейс-стаді підтвердив, що автоматизоване підключення сервісів безпеки AWS надає клієнтам вимірну цінність, допомагаючи їм виявляти та усувати реальні загрози та вразливості.
4. Бізнес-модель «Save → Secure» виявилася життєздатною: клієнти дійсно більш схильні інвестувати в безпеку, коли бачать попередню економію коштів, що створює win-win ситуацію.

ВИСНОВКИ

У ході виконання дипломної роботи було досягнуто поставленої мети — розроблено та валідовано комплексну систему автоматизованої генерації лідів для впровадження сервісів кібербезпеки в середовищі AWS. Отримані результати підтверджують як технічну працездатність, так і економічну доцільність запропонованого підходу.

На основі проведеного дослідження можна зробити наступні висновки, що відповідають поставленим задачам:

1. **Аналіз сучасних загроз підтвердив, що ключовою проблемою хмарної безпеки є помилки конфігурації з боку клієнта.** Системний аналіз звітів про інциденти та ринкових тенденцій показав, що неправильне налаштування сервісів, таких як S3 та IAM, залишається головним вектором атак. Це обґрунтовує необхідність переходу від реактивних моделей безпеки до проактивних, таких як запропонована система, що автоматично виявляє потенційні слабкі місця та ініціює їх усунення.
2. **Розроблено ефективну архітектуру багаторівневого конвеєра обробки даних, що поєднує OSINT та B2B-маркетинг.** Запропонована шестирівнева архітектура (збір, валідація, ідентифікація, збагачення, скоринг, інтеграція) дозволяє з високою точністю перетворювати сирі технічні дані з відкритих джерел на кваліфіковані «теплі» ліди. Експериментальна апробація показала, що такий підхід дозволяє відсіяти понад 98% нерелевантних даних на ранніх, найменш витратних етапах, що підтверджує ефективність архітектурного рішення.

3. **Програмний прототип системи продемонстрував високу ефективність на реальних даних та кейсах.** Реалізовані на Python скрипти для автоматизації всього циклу, від збору доменів до розгортання «Security Pack» через boto3, показали стабільну роботу в умовах, наближених до реальних. Аналіз кейс-стаді (DotsPlatform, Evacodes) довів практичну цінність системи: вона не лише ідентифікує клієнтів, але й допомагає їм вирішувати конкретні проблеми безпеки — від усунення критичних вразливостей до прискорення проходження аудитів відповідності.
4. **Економічна модель «Save → Secure» є життєздатною та взаємовигідною.** Кількісний аналіз підтвердив, що пропозиція клієнтам фінансувати посилення безпеки за рахунок заощаджених на оптимізації хмарних витрат коштів є потужним драйвером продажів. Це дозволило збільшити конверсію в апсел на 260% та підвищити LTV клієнта на 27%. Для клієнтів модель пропонує значне зниження кіберризиків (до 69%) за мінімального зростання чистих витрат.

Дипломна робота демонструє успішну розробку та валідацію комплексного рішення, що знаходиться на перетині кібербезпеки, аналізу даних та автоматизації продажів. Запропонована система є не просто набором скриптів, а цілісною методологією, що відповідає сучасним фреймворкам управління ризиками (NIST RMF) та безпеки (Zero Trust). Вона пропонує масштабований та економічно обґрунтований підхід до вирішення системної проблеми низького рівня впровадження нативних сервісів безпеки AWS, особливо в сегменті малого та середнього бізнесу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Gartner Forecasts Worldwide Public Cloud End-User Spending to Grow 20.4% in 2024. URL: https://www.gartner.com/en/newsroom/press-releases/2023-10-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-20-4-percent-in-2024
2. ISC2 Cybersecurity Workforce Study, 2023. URL: https://www.isc2.org/research/workforce-study
3. Reichheld, F. F. The one number you need to grow. Harvard business review, 81(12), 46-55, 2003.
4. AWS Shared Responsibility Model. URL: https://aws.amazon.com/compliance/shared-responsibility-model/
5. Top 5 Cloud Security Challenges of 2024 and How to Mitigate Them. Mission Cloud. URL: https://www.missioncloud.com/blog/top-5-cloud-security-challenges-of-2024-and-how-to-mitigate-them
6. Amazon GuardDuty. Official Documentation. URL: https://aws.amazon.com/guardduty/
7. Amazon Inspector. Official Documentation. URL: https://aws.amazon.com/inspector/
8. AWS Security Hub. Official Documentation. URL: https://aws.amazon.com/security-hub/
9. Amazon Macie. Official Documentation. URL: https://aws.amazon.com/macie/
10. AWS WAF. Official Documentation. URL: https://aws.amazon.com/waf/
11. IAM Access Analyzer. Official Documentation. URL: https://aws.amazon.com/iam/features/analyze-access/
12. Certificate Transparency. URL: https://certificate.transparency.dev/

13. crt.sh | Certificate Search. URL: https://crt.sh/
14. Apollo.io Platform. URL: https://www.apollo.io/
15. Snov.io Platform. URL: https://snov.io/
16. Boto3 - AWS SDK for Python. Documentation. URL: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
17. GitLab CI/CD. Documentation. URL: https://docs.gitlab.com/ee/ci/
18. HubSpot CRM Platform. URL: https://www.hubspot.com/
19. CIS Benchmarks. Center for Internet Security. URL: https://www.cisecurity.org/cis-benchmarks/
20. OWASP Top 10. URL: https://owasp.org/www-project-top-ten/
21. Ponemon Institute: Cost of a Data Breach Study 2023. URL: https://www.ibm.com/reports/data-breach
22. Cloud Security Posture Management (CSPM). Gartner Glossary. URL: https://www.gartner.com/en/information-technology/glossary/cloud-security-posture-management-cspm
23. The State of FinOps 2024. FinOps Foundation Report. URL: https://data.finops.org/
24. Vaswani, A., et al. Attention is all you need. Advances in neural information processing systems, 30, 2017.
25. Amazon AWS Recon Data For Finding Origin IP - 93M. URL: https://www.kaggle.com/datasets/chiragartani/amazon-aws-asn-cidr-ip-to-hostname-recon-data

ДОДАТОК А / APPENDIX A

TechOrNon.py – класифікація технологічних доменів

```
import pandas as pd

file_path_prospects = 'test_job_kirill-5.csv'

data_prospects = pd.read_csv(file_path_prospects)

print(data_prospects.info())

print(data_prospects.head())

technical_keywords = [

  'DevOps', 'CTO', 'Infrastructure', 'SysOps', 'CIO', 'Engineer',

  'Developer', 'Programmer', 'Technical'

]

def is_technical(position):

  if pd.isna(position):

    return 'Empty'

  return any(keyword in position for keyword in technical_keywords)

data_prospects['Category'] = data_prospects['Job position'].apply(is_technical)

technical_data = data_prospects[data_prospects['Category'] == True]

non_technical_data = data_prospects[data_prospects['Category'] == False]

empty_positions_data = data_prospects[data_prospects['Category'] == 'Empty']

technical_file_path = f'./technical_positions_cleaned_{file_path_prospects}.csv'

non_technical_file_path = f'./non_technical_positions_cleaned_{file_path_prospects}.csv'

empty_positions_file_path = f'./empty_positions_{file_path_prospects}'
```

```
technical_data.to_csv(technical_file_path, index=False)

non_technical_data.to_csv(non_technical_file_path, index=False)

empty_positions_data.to_csv(empty_positions_file_path, index=False)



technical_file_path, non_technical_file_path, empty_positions_file_path
```

ДОДАТОК Б / APPENDIX B

subdomain_checker.py – перевірка піддоменів

```
import csv
import requests

API_KEY =

# URL для API VirusTotal
URL = "https://www.virustotal.com/api/v3/domains/{}/subdomains"

input_directory = "./"
# Путь к CSV файлу для чтения
CSV_INPUT_FILE = 'valid_ssl0222.csv'

# Путь к CSV файлу для записи результатов
CSV_OUTPUT_FILE = 'subdomains_results.csv'

# Запишем заголовки для выходного файла
with open(CSV_OUTPUT_FILE, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(['Domain', 'Subdomains'])

# Функция для поиска субдоменов
def find_subdomains(domain):
    response = requests.get(
        URL.format(domain),
        headers={"x-apikey": API_KEY}
    )
    if response.status_code == 200:
        subdomains_data = response.json()
        subdomains_list = subdomains_data.get('data', [])
        return [subdomain['id'] for subdomain in subdomains_list]
    else:
                    print(f"Ошибка при запросе к API для домена {domain}:
{response.status_code}")
        return []

# Считываем данные из CSV и ищем субдомены
with open(CSV_INPUT_FILE, newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
            domain = row['Working URL'].replace('http://', '').replace('https://',
'').split('/')[0]
```

```
        subdomains = find_subdomains(domain)
        with open(CSV_OUTPUT_FILE, mode='a', newline='') as file:
            writer = csv.writer(file)
            # Записываем домен и найденные субдомены
              writer.writerow([domain, ';'.join(subdomains)])  # Разделяем субдомены
точкой с запятой для сохранения в одной ячейке CSV
        print(f"Субдомены для домена {domain}: {subdomains}")
```

ДОДАТОК В / APPENDIX C

filter.py – фільтрація даних

```
import csv



# Путь к входному CSV файлу с субдоменами

CSV_INPUT_FILE = 'subdomains_results.csv'



# Путь к выходному CSV файлу с отфильтрованными субдоменами

CSV_OUTPUT_FILE = 'filtered_subdomains.csv'



# Ключевые слова для фильтрации субдоменов

KEYWORDS = ['app', 'beta']  # Вы можете добавить дополнительные ключевые слова в этот
список



# Создаем выходной файл и записываем заголовок

with open(CSV_OUTPUT_FILE, mode='w', newline='') as file:

  writer = csv.writer(file)

  writer.writerow(['Domain', 'Filtered Subdomains'])



# Чтение входного файла и фильтрация субдоменов

with open(CSV_INPUT_FILE, newline='') as csvfile:
```

```
  reader = csv.DictReader(csvfile)
  for row in reader:
      # Извлекаем список субдоменов, разделенных точкой с запятой
      subdomains = row['Subdomains'].split(';')
      # Фильтруем субдомены по ключевым словам
       filtered_subdomains = [subdomain for subdomain in subdomains if any(keyword in subdomain
for keyword in KEYWORDS)]
      # Запись отфильтрованных субдоменов в выходной файл
      with open(CSV_OUTPUT_FILE, mode='a', newline='') as file:
          writer = csv.writer(file)
          writer.writerow([row['Domain'], ';'.join(filtered_subdomains)])

print("Фильтрация субдоменов завершена.")
```

ДОДАТОК Г / APPENDIX D

clear.py – очистка дублікатів та «шуму»

```
import pandas as pd
import os



def detect_separator(file_path):
  with open(file_path, 'r') as file:
      first_line = file.readline()
      if ';' in first_line:
```

```
            return ';'
        else:
            return ','



def choose_file():
    mode = input("Выберите режим ввода пути к файлу (manual/auto): ")
    if mode == "auto":
        files = [f for f in os.listdir('.') if os.path.isfile(f)]
        for idx, file in enumerate(files):
            print(f"{idx}: {file}")
        file_index = int(input("Введите номер файла: "))
        return files[file_index]
    else:
        return input("Введите путь к файлу: ")



def clean_data(file_path, columns_methods):
    sep = detect_separator(file_path)
    data = pd.read_csv(file_path, sep=sep, on_bad_lines='skip')
    cleaned_columns = []

    for column, method in columns_methods.items():
        if method == "name" and column.lower() != "position":
```

```
        data[column] = data[column].apply(lambda x: x.split()[0].capitalize() if pd.notnull(x) else x)
        cleaned_columns.append(column)
    elif method == "position":
        keywords = [
            'DevOps', 'CTO', 'Infrastructure', 'SysOps', 'CIO', 'Engineer',
            'Developer', 'Programmer', 'Technical',
                'CEO', 'CFO', 'COO', 'Founder', 'Co-Founder', 'Chief Executive Officer', 'Chief Operating
Officer', 'Chief'
        ]
        keywords_lower = [keyword.lower() for keyword in keywords]

        def clean_position(position):
            if position:
                position_lower = position.lower()
                if any(keyword in position_lower for keyword in keywords_lower):
                    return position  # Возвращаем исходную позицию, если соответствует ключевым
словам
            return None

        data[column] = data[column].apply(clean_position)
        cleaned_columns.append(column)

  return data, cleaned_columns
```

```
if __name__ == "__main__":
    file_path = choose_file()
    data = pd.read_csv(file_path, sep=detect_separator(file_path), on_bad_lines='skip')
    print(f"Выбранный файл: {file_path}")
    print(f"Доступные столбцы: {list(data.columns)}")
    columns_input = input("Введите названия столбцов для очистки через запятую: ")
    columns = [column.strip() for column in columns_input.split(',')]

    columns_methods = {}
    for column in columns:
        method = input(f"Выберите метод очистки (name/position) для столбца {column}: ")
        columns_methods[column] = method

    cleaned_data, cleaned_columns = clean_data(file_path, columns_methods)

    # Создание папки output, если она не существует
    if not os.path.exists('output'):
        os.makedirs('output')

    # Сохранение файла с информацией о примененной очистке
    cleaned_columns_str = "-".join(cleaned_columns).lower()
    output_file_path = f'output/cleaned_{cleaned_columns_str}_' + os.path.basename(file_path)
    cleaned_data.to_csv(output_file_path, index=False)
```

```
  print(f"Очищенный файл сохранён как: {output_file_path}")
```

ДОДАТОК Ґ / APPENDIX E

creating_new_list.py – формування нових списків контактів

```
import csv
import os
import re
import shutil


# Function to count rows in a single CSV file, considering if there is a header row or not
def count_rows_csv(file_path, header_present=True):
  with open(file_path, 'r', encoding='utf-8') as file:
      csv_reader = csv.reader(file)
      # Subtract 1 if there is a header row
      row_count = sum(1 for row in csv_reader) - (1 if header_present else 0)
  return row_count


# Define the directory paths
source_directory = '../valid_ssl'  # Make sure this is the correct path
to_connect_directory = './was_connected'  # Make sure this is the correct path
```

```
to_connect_next_directory = './to_connect_next'  # Make sure this is the correct path


# Regex pattern to find files with the naming convention 'valid_ssl<number>.csv'
pattern = re.compile(r'valid_ssl(\d+)\.csv')


# Create to_connect_next directory if it does not exist
if not os.path.exists(to_connect_next_directory):
    os.makedirs(to_connect_next_directory)


# List to hold the numbers extracted from file names already in to_connect
existing_numbers = []


# Check if the to_connect directory exists and get the existing numbers
if os.path.exists(to_connect_directory):
    for filename in os.listdir(to_connect_directory):
        match = pattern.search(filename)
        if match:
            existing_numbers.append(int(match.group(1)))


# Sort the list of existing numbers
existing_numbers.sort()


# Counter for the total records
total_records = 0
```

```
for filename in os.listdir(to_connect_directory):
    if filename.endswith('.csv'):
        # Check if your CSV files have headers and set header_present accordingly
                        total_records   +=   count_rows_csv(os.path.join(to_connect_directory,   filename),
header_present=True)


files_copied = 0


# Maximum number of records allowed
# max_records = 1000000  # Change this if the intended maximum is 100000
max_records = 1000000  # Change this if the intended maximum is 100000
# Start copying files from source to to_connect_next
for filename in sorted(os.listdir(source_directory)):
    # Stop if maximum files are copied or total records limit is reached
    if files_copied >= 50 or total_records >= max_records:
        break


    match = pattern.search(filename)
    if match:
        file_number = int(match.group(1))
        if file_number not in existing_numbers:
            source_file = os.path.join(source_directory, filename)
            destination_file = os.path.join(to_connect_next_directory, filename)
            # Copy file and update the records counter
```

```
        shutil.copy2(source_file, destination_file)
        files_copied += 1
        total_records += count_rows_csv(source_file, header_present=True)

# If the total records exceed the maximum allowed, remove files from to_connect_next until it does not
while total_records > max_records and files_copied > 0:
        filename_to_remove = sorted(os.listdir(to_connect_next_directory), key=lambda f: int(pattern.search(f).group(1)))[-1]
    file_path_to_remove = os.path.join(to_connect_next_directory, filename_to_remove)
    total_records -= count_rows_csv(file_path_to_remove, header_present=True)
    os.remove(file_path_to_remove)
    files_copied -= 1

print(f"Total records: {total_records}")
print(f"Files copied to to_connect_next: {files_copied}")
```

ДОДАТОК Д / APPENDIX F

afterverifalia.py – обробка результатів Verifalia

```
import pandas as pd

def filter_prospects(verifalia_file, prospects_file, output_file):
    # Load the datasets
    verifalia_df = pd.read_csv(verifalia_file)
```

```
    prospect_df = pd.read_csv(prospects_file)


    # Filter the prospects dataframe
    filtered_prospect_df = prospect_df[prospect_df['Email'].isin(verifalia_df['EmailAddress'])]


    # Save the filtered data
    filtered_prospect_df.to_csv(output_file, index=False)
    print(f"Filtered data saved to {output_file}")


# Usage example
verifalia_file = 'Verifalia-5686a9fb-e8d3-44f7-8256-902c2154cfa8-results.csv'
prospects_file = 'Ukraine_updated.csv'
output_file = 'Ukraine(cleaned).csv'


filter_prospects(verifalia_file, prospects_file, output_file)
```

ДОДАТОК Е / APPENDIX G

auto_validator.py – автоматична валідація доменів

```
import csv
import os
import time


import requests
```

```
from concurrent.futures import ThreadPoolExecutor


checked_domains = set()

link_check_counter = 0


def is_link_working(url):
  global link_check_counter


  clean_url = url.replace("*.", "")

  # print(f"Checking link: {clean_url}")




  link_check_counter += 1


  if link_check_counter % 100 == 0:

    print(f"Checked {link_check_counter} links so far...")


  if clean_url in checked_domains:

    return None

  checked_domains.add(clean_url)


  try:

    response = requests.get(f"https://{clean_url}", timeout=2)

    return clean_url if response.status_code == 200 else None
```

```
    except:
        return None


def extract_domains(subject_alternative_dns_name):
    return [domain.replace("*.", "") for domain in subject_alternative_dns_name.split('|')]


def check_links_for_row(row):
    domains = extract_domains(row['Subject Alternative DNS Name'])
    for domain in domains:
        working_domain = is_link_working(domain)
        if working_domain:
            # print(working_domain)
            return f"https://{working_domain}"
    return None



def main():
    input_directory = "ssl/"
    output_directory = "valid_ssl/"

    # Создаем выходную папку, если она еще не существует
    os.makedirs(output_directory, exist_ok=True)

    # Получаем список всех файлов в директории
```

```
files = [f for f in os.listdir(input_directory) if f.endswith('.csv')]


for input_file in files:
    output_file = f"valid_{input_file}"
    output_path = os.path.join(output_directory, output_file)

    # Проверяем, существует ли уже обработанный файл
    if os.path.exists(output_path):
        print(f"File {output_file} already exists. Skipping.")
        continue

    print(f"Processing file: {input_file}")
    process_file(input_directory, output_directory, input_file, output_file)
    # consent = input("Do you want to proceed? (1 for yes, 0 for no): ")
    # if consent == '1':
    #     process_file(input_directory, output_directory, input_file, output_file)
    # elif consent == '0':
    #     print("Stopping the script as per the user's request.")
    #     break
    # else:
    #     print("Invalid input, skipping the file.")



def process_file(input_directory, output_directory, input_file, output_file):
```

```
start_time = time.time()  # Запись стартового времени


with open(input_directory + input_file, 'r') as infile:
    reader = csv.DictReader(infile)
    rows = list(reader)
    initial_count = len(rows)  # Исходное количество строк


with ThreadPoolExecutor() as executor:
    valid_links = list(executor.map(check_links_for_row, rows))


# Removing None values and writing valid links to a CSV file
valid_links = [link for link in valid_links if link]
with open(output_directory + output_file, 'w', newline='') as outfile:
    writer = csv.writer(outfile)
    writer.writerow(['Working URL'])
    for link in valid_links:
        writer.writerow([link])


elapsed_time = time.time() - start_time  # Расчет затраченного времени


# Вывод статистики:
print(f"File {input_file} processed")
print(f"Total execution time: {elapsed_time:.2f} seconds")
print(f"Initial number of rows: {initial_count}")
```

```
  print(f"Number of valid links in the final file: {len(valid_links)}")


if __name__ == "__main__":
  main()
```

ДОДАТОК Є / APPENDIX H

prospects_new_23_02.py – підготовка списку потенційних клієнтів

```
import random
import requests
import csv


# Константы для API
API_USER_ID
API_SECRET =


def get_access_token():
  """Получение токена доступа к API."""
  auth_data = {
    'grant_type': 'client_credentials',
    'client_id': API_USER_ID,
    'client_secret': API_SECRET
  }
  response = requests.post("https://api.snov.io/v1/oauth/access_token", data=auth_data)
```

```
  token_data = response.json()
  return token_data["access_token"]


def get_domain_search(domain, token):
  """Получение данных о домене."""
  try:
      params = {
          'access_token': token,
          'domain': domain,
          'type': 'all',
          'limit': 100,
          'lastId': 0,
                      'positions[]': ['CEO', 'CTO', 'Founder', 'DevOps', 'CPO', 'Chief Product Officer',
'Co-founder',
                                  'Chief Executive Officer', 'SysOps', 'Infrastructure engineer', 'CIO', 'Head of
Product',
                                  'Head of Infrastructure', 'Owner', 'Chief', 'COO', 'Ops', 'Director', 'Engineer',
'Technology', 'Tech', 'CFO', 'Operations', "Engineering", "Technical", 'Cloud', 'AWS', 'Infrastructure']
          }
      #Added 27.02 Chief, Owner, Coo, Ops
      res = requests.get('https://api.snov.io/v2/domain-emails-with-info', params=params)
      return res.json()
  except requests.exceptions.ConnectionError as e:
      print(f"Connection to {domain} failed: {e}")
      return {'data': []}
```

```
def save_partial_results(contacts, fieldnames, output_file):
    """Сохранение частичных результатов в файл."""
    with open(output_file, 'a', newline='', encoding='utf-8') as outfile:
        writer = csv.DictWriter(outfile, fieldnames=fieldnames)
        if outfile.tell() == 0:  # Если файл пуст, пишем заголовок
            writer.writeheader()
        for contact in contacts.values():
            writer.writerow(contact)



def load_processed_websites(output_file):
    """Загружает список уже обработанных веб-сайтов из итогового файла."""
    processed_websites = set()
    try:
        with open(output_file, 'r', encoding='utf-8') as f:
            reader = csv.DictReader(f)
            for row in reader:
                website = row.get('Website')
                if website:  # Проверяем, что значение существует
                    processed_websites.add(website)
    except FileNotFoundError:
        print(f"Файл {output_file} не найден. Будет создан новый файл.")
    return processed_websites
```

```
def main():
    input_file = "Poland15_03_final.csv"
    output_file = "prospects_" + input_file
    no_data_file = "no_data_" + input_file

    token = get_access_token()

    # Загрузка уже обработанных веб-сайтов
    processed_websites = load_processed_websites(output_file)

    contacts = {}
    no_data_domains = []
    processed_count = 0

    with open(input_file, 'r', encoding='utf-8') as infile:
        reader = csv.DictReader(infile)
            fieldnames = ['Company', 'Company Cleared', 'Company LinkedIn', '# Employees', 'Website',
'Country',
                      'LinkedIn', 'Email', 'First Name', 'Last Name', 'Position', 'Industry', 'AWS']

        for row in reader:
            if 'Website' not in row or row['Website'] in processed_websites:
```

```
            print(f"{row['Website']} - Уже обработан")
            continue  # Пропускаем веб-сайты, которые уже обработаны

        print(row['Website'])
        domain = row['Website'].replace('http://', '').replace('https://', '').split('/')[0]
        print(f"Processing: {domain}")
        if domain in processed_websites:
            continue
        response = get_domain_search(domain, token)

        if not response.get("data"):
            print(f"No data retrieved for {domain}")
            no_data_domains.append(domain)
            continue

        for item in response.get("data", []):
            key = (item.get('first_name', '').lower(), item.get('last_name', '').lower(), item.get('position',
'').lower())
            email = item.get('email', '').lower()

            if key not in contacts:
                contacts[key] = {
                    'Email': email,
                    'First Name': item.get('first_name', ''),
```

```
            'Last Name': item.get('last_name', ''),
            'Position': item.get('position', ''),
            'LinkedIn': item.get('source_page', ''),
            'Company': row.get('Company', ''),  # Используйте .get() с нужными вам полями
            'Company Cleared': row.get('Company Name for Emails', ''),
            'Company LinkedIn': row.get('Company LinkedIn Url', ''),
            '# Employees': row.get('# Employees', ''),  # Исправлено
            'Website': row.get('Website', ''),
            'Country': row.get('Company Country', ''),
            'Industry': row.get('Industry', ''),
            'AWS': row.get('AWS', '')
        }
    elif email not in contacts[key]['Email']:
        contacts[key]['Email'] += ', ' + email

    print(f"Added/Updated: {row['Company']}")
    print(f"Added/Updated: {item.get('position', '')}")
processed_websites.add(domain)
processed_count += 1
if processed_count % 10 == 0:
    print(f"Processed amount: {processed_count}")
if processed_count % 200 == 0:
    save_partial_results(contacts, fieldnames, output_file)
    contacts = {}
```

```
    # Сохранение оставшихся контактов
    save_partial_results(contacts, fieldnames, output_file)

    # Сохранение доменов без данных
    with open(no_data_file, 'w', newline='', encoding='utf-8') as nodatafile:
        nodata_writer = csv.writer(nodatafile)
        nodata_writer.writerow(['Domain'])
        for domain in no_data_domains:
            nodata_writer.writerow([domain])

if __name__ == "__main__":
  main()
```

ДОДАТОК Ж / APPENDIX I

my.js – перевірка розміщення доменів на AWS

```
const dns = require('dns');
const https = require('https');
const { Netmask } = require('netmask');
const xlsx = require('xlsx');
const async = require('async');
const fs = require('fs');
const { PassThrough } = require('stream');

// Function to resolve domain to IP
```

```
function resolveDomain(domainName, callback) {
  dns.lookup(domainName, (err, address) => {
    if (err) {
      callback(err, null);
    } else {
      callback(null, address);
    }
  });
}

// Function to check if IP is in AWS ranges
function isIpInAwsRanges(ip, awsRanges) {
  let serviceInfo = { onAws: false, service: 'Outside AWS' };

  ['prefixes', 'GLOBALACCELERATOR', 'CLOUDFRONT', 'AMAZON', 'EC2', 'S3'].forEach(serviceKey => {
    const servicePrefixes = awsRanges[serviceKey];
    if (servicePrefixes) {
      servicePrefixes.forEach(prefix => {
        if (prefix && prefix.ip_prefix && new Netmask(prefix.ip_prefix).contains(ip)) {
          serviceInfo = { onAws: true, service: prefix.service || serviceKey };
        }
      });
    }
  });

  return serviceInfo;
}
```

```
// Function to get AWS IP ranges from the official source
function getAwsIpRanges(callback) {
  https.get('https://ip-ranges.amazonaws.com/ip-ranges.json', (res) => {
    let data = '';
    res.on('data', (chunk) => { data += chunk; });
    res.on('end', () => { callback(null, JSON.parse(data)); });
  }).on('error', (err) => { callback(err, null); });
}

// Main function to check if a domain is hosted on AWS
function isDomainOnAws(domainName, callback) {
  resolveDomain(domainName, (err, ip) => {
    if (err) {
      callback(domainName, err, null, false, null);
    } else {
      getAwsIpRanges((err, awsRanges) => {
        if (err) {
          callback(domainName, null, err, false, null);
        } else {
          const { onAws, service } = isIpInAwsRanges(ip, awsRanges);
          callback(domainName, null, ip, onAws, service);
        }
      });
    }
  });
}

// Function to extract hostname from a URL
```

```
function extractHostname(url) {
  try {
    const { hostname } = new URL(url);
    return hostname;
  } catch (err) {
    console.error(`Error extracting hostname from URL ${url}:`, err.message);
    return null;
  }
}

// Function to check domains from an XLSX file and output results to a CSV file
function writeResultsToStream(results, writeStream, isFirstChunk) {
  if (!isFirstChunk) {
    // Remove the first row (headers) if it's not the first chunk of data
    results.shift();
  }
  const worksheet = xlsx.utils.json_to_sheet(results, { skipHeader: !isFirstChunk });
  const passThroughStream = new PassThrough();
  passThroughStream.on('data', (buffer) => {
    writeStream.write(buffer);
  });
  passThroughStream.on('end', () => {
    // Do not close the writeStream here
  });
  xlsx.stream.to_csv(worksheet).pipe(passThroughStream);
}

// Function to process domains from XLSX file and append results to CSV file
```

```
function checkDomainsFromXlsx(inputFilePath, outputFilePath) {
  const workbook = xlsx.readFile(inputFilePath);
  const sheetName = workbook.SheetNames[0];
  const worksheet = workbook.Sheets[sheetName];
  const domains = xlsx.utils.sheet_to_json(worksheet);
  let fullResults = [];
  let isFirstChunk = true;
  const writeStream = fs.createWriteStream(outputFilePath, { flags: 'a' });


  const queue = async.queue((domain, done) => {
    const hostname = extractHostname(domain.Website);
    console.log(hostname)
    if (hostname) {
      isDomainOnAws(hostname, (domainName, resolveErr, ip, onAws, service) => {
        fullResults.push({ ...domain, IP: ip, AWS: onAws ? "AWS" : "NOT", Service: service || "N/A" });
        if (fullResults.length >= 500) {
          writeResultsToStream(fullResults, writeStream, isFirstChunk);
          isFirstChunk = false;
          fullResults = [];
        }
        done();
      });
    } else {
      fullResults.push({ ...domain, IP: "Invalid URL", AWS: "N/A", Service: "N/A" });
      if (fullResults.length >= 500) {
        writeResultsToStream(fullResults, writeStream, isFirstChunk);
        isFirstChunk = false;
        fullResults = [];
```

```
        }
        done();
    }
  }, 10);

  domains.forEach(domain => queue.push(domain));

  queue.drain(() => {
    if (fullResults.length > 0) {
        writeResultsToStream(fullResults, writeStream, isFirstChunk);
    }
    // Wait for all data to be flushed to the CSV file before closing the stream
    setTimeout(() => {
        writeStream.end();
        console.log('All domains have been processed and the file has been written.');
    }, 1000);
  });
}

const inputFilePath = './rio1100_updated.csv';
const outputFilePath = './result_rio.csv';
checkDomainsFromXlsx(inputFilePath, outputFilePath);
```