МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАДИМА ГЕТЬМАНА»
навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
кафедра інформаційних систем в економіці

Освітньо-професійна програма

галузь знань 12 Інформаційні технології
спеціальність 122 Комп'ютерні науки
спеціалізація Інформаційні управляючі системи та технології

форма навчання: денна

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ БЕЗПЕКИ ВЕБ-САЙТІВ

здобувача Крошка Івана Андрійовича _____

Науковий керівник: к. е. н., доцент

_____ Гордієнко І. В.

Кваліфікаційна магістерська робота допущена до захисту в Екзаменаційній комісії з атестації здобувачів вищої освіти
в. о. завідувача кафедри: к. е. н., доцент

_____ Тішков Б.О.

Київ 2022

Міністерство освіти і науки України
Державний вищий навчальний заклад
«Київський національний економічний університет
імені Вадима Гетьмана»
Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»
кафедра інформаційних систем в економіці

галузь знань 12 «Інформаційні технології»
спеціальність 122 «Комп'ютерні науки»
спеціалізація «Інформаційні управляючі системи та технології»

Затверджую:
В. о. завідувача кафедри
_____ Тішков Б. О.
"____" _____2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
Крошка Івана Андрійовича
денної форми навчання
на підготовку кваліфікаційної магістерської роботи
на тему: ***«Інформаційна система моніторингу безпеки веб-сайтів»***
Тему затверджено наказом ректора від «21» грудня 2021 р. № 1917-ст.
Кваліфікаційна магістерська робота виконується на матеріалах

План кваліфікаційної магістерської роботи

Розділ І *Теоретичний розділ. Дослідження та аналіз підходів до створення інформаційних управляючих систем предметної області*

Розділ ІІ *Аналітичний розділ. Характеристика інформаційних управляючих систем та методи і моделі*

Розділ ІІІ *Конструктивний розділ. Розроблення проєктних рішень та їх реалізація*

Об'єкт дослідження *методи і засоби моніторингу безпеки веб-сайтів*

Предмет дослідження *процес розроблення інформаційної системи моніторингу безпеки веб-сайтів*

Мета кваліфікаційної магістерської роботи *набуття теоретичних знань та практичних навичок створення ІС і демонстрація рівня професійної підготовки магістра на прикладі розробки проєктних рішень зі створення ІС*

Конкретні завдання, які студент повинен виконати для досягнення поставленої мети:

У розділі І *Описати предметну галузь моніторингу безпеки веб-сайтів, підходи та інструменти протидії веб-атакам. Проаналізувати існуючі інформаційні системи і засоби захисту веб-додатків. Обґрунтувати вибір підходів і технологій для створення ІС моніторингу безпеки веб-сайтів та її компонентів*

У розділі ІІ *Визначити структуру і характеристики проєктованої системи та подати їх у формі діаграм і проєктної специфікації, розроблених з використанням CASE-засобів та мови SysML. Описати методи і моделі процесів і елементів інформаційної системи моніторингу безпеки веб-сайтів*

У розділі ІІІ *Спроєктувати базу даних інформаційної системи моніторингу безпеки веб-сайтів. Розробити інформаційне забезпечення ІС. Обгрунтувати склад комплексу технічних засобів інформаційної системи. Описати структуру та складові програмного забезпечення ІС. Описати результати реалізації ІС*

Завдання підготував
науковий керівник _____ **Гордієнко І. В.**
(підпис) “_____”_____
2022р.

Завдання одержав
студент _____ **Крошко І. А.**
(підпис)
“_____”_____
2022 р.

АНОТАЦІЯ

кваліфікаційної магістерської роботи

студента 6 курсу

Крошка Івана Андрійовича

виконана на тему:

«Інформаційна система моніторингу безпеки веб-сайтів»

Кваліфікаційна магістерська робота присвячена актуальній проблемі забезпечення інформаційної безпеки веб-сайтів. Дану проблему пропонується розв’язувати з використанням сучасних інформаційних технологій.

Кваліфікаційна магістерська робота складається з трьох розділів, логічно пов’язаних між собою.

В першому розділі дана характеристика предметної галузі й об’єкта дослідження, наведено аналіз задач, що розв’язуються, а також наводиться перелік існуючих програмних засобів даної предметної галузі.

Другий розділ є аналітичним. Він присвячений обґрунтуванню методу проектування системи, розробленню її архітектури, виконанню постановки та розробленню алгоритму розв’язання задач.

Третій розділ – конструктивний. Тут наведено інформаційне, технічне програмне та організаційне забезпечення для інформаційної системи онлайн-магазину. Висвітлені питання організації інформаційного забезпечення: розроблена структура інформаційного забезпечення, описані форми вхідних та вихідних документів. Обґрунтований комплекс технічних засобів, а також програмне забезпечення, що використовується для створення системи моніторингу безпеки веб-сайтів. Вказані структури інформаційних масивів, які використовуються під час вирішення задачі.

Висновки містять рекомендації щодо доцільності розробки та впровадження інформаційної системи моніторингу безпеки веб-сайтів.

РЕФЕРАТ

Робота обсягом 137 сторінок містить 22 ілюстрацій, 24 таблиці, 3 додатки та 31 літературне посилання.

«ІНФОРМАЦІЙНА СИСТЕМА МОНІТОРИНГУ БЕЗПЕКИ ВЕБ-САЙТІВ»

Метою даної кваліфікаційної роботи є проєктування та розробка інформаційної системи моніторингу безпеки веб-сайтів.

Об’єктом дослідження є методи і засоби моніторингу безпеки веб-сайтів.

Предметом дослідження є процес розробки інформаційної системи моніторингу безпеки веб-сайтів

Результати роботи викладені у вигляді таблиці та ілюстрацій, що демонструють ефективність розробленої інформаційної системи, а також можливість їх практичного застосування.

Результати роботи можуть бути використані для подальшої розробки інформаційних систем моніторингу безпеки веб-сайтів.

Ключові слова: веб-сайт, інформаційна безпека, вразливості, моніторинг.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

HTTP - широко поширений протокол передачі даних, спочатку призначений для передачі гіпертекстових документів (тобто документів, які можуть містити посилання, дозволяють організувати перехід до інших документів) (англ. HyperText Transfer Protocol).

HTTPS - це протокол, який забезпечує цілісність та конфіденційність даних при їх передачі між сайтом та пристроєм користувача(англ. HyperText Transfer Protocol Secure).

FTP - є одним із перших способів обміну даними між комп'ютерами по мережі TCP/IP. Дуже зручний для завантаження та завантаження файлів великого об'єму. Протокол працює у моделі клієнт-сервер, де FTP-сервер та FTP-клієнт виконують операції передачі даних(англ. File Transport Protocol).

SFTP - протокол прикладного рівня передачі файлів, що працює поверх безпечного каналу(англ. Secure File Transport Protocol).

SSL - криптографічний протокол, який забезпечує встановлення безпечного з'єднання між клієнтом і сервером (англ. Secure Socket Layer).

API - опис способів (набір класів , процедур , функцій), якими одна комп'ютерна програма може взаємодіяти з іншою програмою(англ. Application Programming Interface).

TCP/IP - мережна модель передачі даних , представлених у цифровому вигляді. Модель визначає спосіб передачі від джерела інформації до одержувачу. У моделі передбачається проходження інформації через чотири рівні, кожен із яких описується правилом (протоколом передачі). Набори правил, що вирішують задачу передачі даних, складають стек протоколів передачі даних , на яких базується Інтернет(англ. Transmission Control Protocol/Internet Protocol).

HTML - це код, який використовується для структурування та відображення веб-сторінки та її контенту(англ. HyperText Markup Language).

CVE - база даних загальновідомих вразливостей інформаційної безпеки(англ. Common Vulnerabilities and Exposures)

CVSS - відкритий стандарт для оцінки ступеня небезпеки вразливостей(англ. Common Vulnerability Scoring System).

SaaS - одна з форм хмарних обчислень, модель обслуговування, при якій передплатникам надається готове прикладне програмне забезпечення , що повністю обслуговується провайдером (англ. Software as a Service).

REST - це стиль архітектури програмного забезпечення для розподілених систем, таких як World Wide Web, який зазвичай використовується для побудови веб-служб(англ. Representational State Transfer).

ВСТУП

Веб-вразливості сьогодні перевершують за кількістю та пов'язаними ризиками будь-які інші проблеми інформаційної безпеки. Значна частина сучасних автоматизованих систем використовують веб-технології. Переважна більшість зовнішніх атак на корпоративні інформаційні системи націлені саме на вразливості веб-додатків, і з кратним зростанням ризиків суттєва увага стала приділятись виявленню та усуненню вразливостей саме в них. І якщо всередині мережі зловмиснику доступний Інтернет, його завдання спрощуються.

Щоб переконатися, що веб-програма є безпечною, необхідно визначити всі проблеми безпеки та вразливості в самій веб-програмі, перш ніж зловмисник ідентифікує та використає їх. Дуже важливо виконувати процес виявлення вразливостей веб-застосунків як протягом життєвого циклу розробки програмного забезпечення, так і в процесі його експлуатації.

Удосконалення методів захисту веб-ресурсів є також важливою задачею внаслідок зростаючих економічних, соціальних та політичних наслідків зловмисних дій.

Актуальність роботи. Зумовлюється тим, що на даний момент кількість веб-сайтів, а також їх цінність в рамках реалізації бізнес-процесів значно зростає. Управління інформаційною безпекою веб-ресурсів дозволяє забезпечити цілісність та конфіденційність чутливої інформації, а також попередити кіберзагрози більш масштабного рівня. Представлена робота пропонує реалізацію інформаційної системи моніторингу безпеки веб-сайтів.

Мета роботи. Проєктування та розробка інформаційної системи моніторингу безпеки веб-сайтів.

Об’єкт дослідження. Методи і засоби моніторингу безпеки веб-сайтів.

Предмет дослідження. Процес розроблення інформаційної системи моніторингу безпеки веб-сайтів.

Практичне значення. Результати роботи можуть бути використані для подальшої розробки інформаційних систем моніторингу безпеки веб-сайтів.

1. ТЕОРЕТИЧНИЙ РОЗДІЛ. ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ ІНФОРМАЦІЙНИХ УПРАВЛЯЮЧИХ СИСТЕМ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Аналіз існуючих управляючих систем предметної області

1.1.1. Дослідження предметної області

1.1.1.1. Веб-сайт як об’єкт захисту

Інтернет – глобальна система об'єднаних комп'ютерних мереж для зберігання та передачі даних. Сам дизайн та основи функціонування інтернету відчинили двері між «захищеними» корпоративними мережами та самоюм мережею. Це та сама мережа, побудована на TCP/IP. І керівники ІБ, і фахівці з мережевої безпеки добре знають, як зробити інтернет безпечним для своєї організації.

Всесвітня павутина (www), веб - це розподілена система зв'язку документів, розташованих на сотнях мільйонів веб-серверів, підключених до інтернету. Це давно вже не просто веб-сторінки та веб-сайти, а повноцінні інформаційно-комунікаційні системи з веб-інтерфейсом. Мінімальна конфігурація має веб-сервер (наприклад, Apache або IIS), операційну систему веб-сервера (Windows/Linux), сервер БД (MySQL/MS SQL) та мережну службу оновлення веб-сайту (FTP/SFTP). Всі ці компоненти повинні бути безпечними, оскільки у разі злому будь-якого з них зловмисники можуть отримати доступ до даних. Веб — лише транспорт, платформа, сервіс.

Вразливість (vulnerability) — помилка чи недолік у системі, використовуючи який можна навмисно порушити її конфіденційність, цілісність чи доступність. Йдеться не стільки про захист самого веб-додатку, а про те, що вразливості вебу дозволяють кіберзлочинцям отримувати доступ до мережевих пристроїв, серверів додатків, БД та ін. Деякі вразливості відомі тільки теоретично, інші ж можуть активно використовуватися за наявності експлойту (exploit) - програмного коду, що використовує вразливість у

програмному забезпеченні та застосовується для проведення атаки на систему. Вразливості веб-застосунків набагато простіше виявити, ефективніше використовувати і при цьому приховати свою присутність. Вразливості можуть експлуатуватися не лише з інтернету, а й із середини (внутрішній порушник).

Запропоновані розв'язання проблеми не завжди очевидні, зрозумілі та доступні — від суперечності один одному до втрати доступності та повного хаосу. Приклад популярного підходу:

1. Запровадити та підтримувати систему управління інформаційною безпекою, у тому числі забезпечити постійне інформування про нові проблеми безпеки (адже це основа, концепт, а не вирішення прикладного завдання);
2. Перевірити всі існуючі програми на предмет правильної розробки та забезпечення інформаційної безпеки (чи реально це зробити, не кажучи про вартість подібного);
3. Переконатися, що всі оновлення/патчі зроблені відразу, і регулярно перевіряти потребу в них (безперечно, але проблему не вирішує);
4. Довіряти веб-розробку/підтримку лише досвідченим та рекомендованим експертам програмістам (як це оцінити та перевірити);
5. Забезпечити ретельну перевірку їх роботи фахівцями з безпеки, у тому числі з використанням активного дослідження програмного коду (спеціалісти з безпеки не знають веб так, як програмісти);
6. Передати на кваліфіковану підтримку усі засоби захисту інформації;
7. Запровадити систему захисту від цільових атак (мова про масові атаки із простими сценаріями);
8. Провести оцінку всіх інформаційних систем щодо їх відповідності.

1.1.1.2. Теоретичні відомості про веб-атаки

На даний момент виділяють наступні атаки: mailbombing, переповнення буфера, використання спеціалізованих програм (вірусів, сніфферов,

троянських коней, поштових черв'яків, rootkit-ів і т.д.), мережева розвідка, IP-спуфинг, man-in-the-middle, ін'єкція (SQL-ін'єкція, PHP-ін'єкція, міжсайтовий скриптинг або XSS-атака, XPath ін'єкція), відмова в обслуговуванні (DoS- і DDoS- атаки), phishing-атаки [2].

1. Mailbombing;

Суть даної атаки полягає в тому, що на поштову скриньку користувача надсилається величезна кількість листів. Ця атака може викликати відмову роботи поштової скриньки або навіть цілого поштового сервера. Вона може проводитися будь-яким хоча б трохи підготовленим супротивником. Простим прикладом програми, за допомогою якої можна здійснити подібну атаку- The Unabomber. Досить знати адресу сервера, що дозволяє анонімно відправляти поштові повідомлення, і адресу користувача, якому ці повідомлення призначені. Кількість листів, яке можна відіслати для цієї програми дорівнює 12 розрядному числу. Зазвичай до таких атак досвідчені зловмисники вдаються вкрай рідко [3].

2. Переповнення буфера (buffer overflows);

Атака на переповнення буфера ґрунтується на пошуку програмних або системних вразливостей, здатних викликати порушення кордонів пам'яті та аварійно завершити роботу додатку або виконати довільний бінарний код від імені користувача, під яким працювала вразлива програма.

Таблиця 1.1

Класифікація атак переповнення буфера

Підготовка коду. Мета переповнення	**Впровадження коду**	**Впровадження параметрів**	**Не вимагається**
Спотворення адреси повернення з функції	Атака "зрив стека"	Атака "зрив стека" з параметризацією	Атака "зрив стека" з передачею керування
Спотворення покажчиків функцій	Атака на покажчики функцій	Атака на покажчики функцій з параметризацією	Атака на покажчики функцій з передачею керування
Спотворення таблиць переходів	Атака на таблиці переходів	Атака на таблиці переходів з параметризацією Атака на таблиці переходів з передачею керування	Атака на таблиці переходів з передачею керування
Спотворення покажчиків даних	Атака з спотворенням покажчиків даних	Атака з спотворенням покажчиків даних з параметризацією	Атака з спотворенням покажчиків даних з оригінальним кодом

Якщо програма працює під обліковим записом адміністратора, то дана атака може дозволити отримати повний контроль над комп'ютером, на якому

виконується дана програма [3]. Класифікація атак переповнення буфера представлена в табл.1.1. Вірусами називаються шкідливі програми, які впроваджуються в інші програми для виконання певної небажаної функції на робочій станції кінцевого користувача.

Як приклад можна привести вірус, який прописується у файлі command.com (головному інтерпретаторі систем Windows) і стирає інші файли, а також заражає всі знайдені ним версії command.com. "Троянський кінь" - це не програмна вставка, а справжня програма, яка виглядає як корисний додаток, а на ділі виконує шкідливу роль. Прикладом типового "троянського коня" є програма, яка виглядає, як проста гра для робочої станції користувача. Однак поки користувач грає в гру, програма відправляє свою копію електронною поштою кожному абоненту, занесеному в адресну книгу цього користувача. Всі абоненти отримують поштою гру, викликаючи її подальше поширення.

Сніфери пакетів є прикладною програму, яка використовує мережеву карту, що працює в режимі promiscuous mode (в цьому режимі всі пакети, отримані по фізичних каналах, мережевий адаптер відправляє додатком для обробки). При цьому сніфери перехоплюють всі мережеві пакети, які передаються через певний домен. В даний час сніфери працюють в мережах на цілком законній підставі. Вони використовуються для діагностики несправностей і аналізу трафіку. Однак з огляду на те, що деякі мережеві додатки передають дані в текстовому форматі (telnet, FTP, SMTP, POP3 і т.д.), за допомогою сніферу можна дізнатися корисну, а іноді і конфіденційну інформацію (наприклад, імена користувачів і паролі). Перехоплення імен і паролів створює велику небезпеку, так як користувачі часто застосовують один і той же логін і пароль для безлічі додатків і систем. Багато користувачів взагалі мають один пароль для доступу до всіх ресурсів і додатків. Якщо додаток працює в режимі клієнт / сервер, а аутентифікаційні дані передаються по мережі і читаються у текстовому форматі, цю інформацію з великою ймовірністю можна використовувати для доступу до інших корпоративних або зовнішніх ресурсів. Rootkit - програма або набір програм для приховування

слідів присутності зловмисника або шкідливої програми в системі. Більшість з реалізацій сучасних rootkit можуть ховати від користувача файли, папки і ключі реєстру, приховувати запущені програми, системні служби, драйвери і мережеві з'єднання. Тобто зловмисник має можливість створювати файли і ключі реєстру, запускати програми, працювати з мережею і ця активність не буде виявлена адміністратором. Крім того, rootkits можуть приховувати мережеву активність шляхом модифікації стека протоколів TCP / IP. Так, наприклад rootkit Hacker Defender перехоплює виклики Winsock і може обробляти мережевий трафік до того як він буде переданий додатком. Тобто якщо в системі встановлено Web сервер, і відповідно відкритий 80-й порт, rootkit може використовувати його для взаємодії з хакером, в той час як інші користувачі будуть без проблем працювати по протоколу HTTP.

3. Мережева розвідка;

Мережевою розвідкою називається збір інформації про мережу за допомогою загальнодоступних даних і додатків. При підготовці атаки проти будь-якої мережі зловмисник, як правило, намагається отримати про неї якомога більше інформації. Мережева розвідка проводиться у формі запитів DNS, сканування портів. Запити DNS допомагають зрозуміти, хто володіє тим чи іншим доменом і які адреси цього домену привласнені. Тестування (ping sweep) адрес, розкритих за допомогою DNS, дозволяє побачити, які хости реально працюють в даному середовищі. Отримавши список хостів, зловмисник використовує засоби сканування портів, щоб скласти повний список послуг, що надаються цими хостами. І, нарешті, зловмисник аналізує характеристики додатків, що працюють на хостах. В результаті видобувається інформація, яку можна використовувати для злому [4].

4. IP-спуфінг;

IP-спуфинг відбувається, коли зловмисник, що знаходиться всередині корпорації або поза нею видає себе за санкціонованого користувача. Це можна зробити двома способами. Зловмисник може скористатися IP-адресою, що знаходиться в межах діапазону санкціонованих IP-адрес, або вповноваженою зовнішньою адресою, якій дозволяється доступ до певних мережевих ресурсів.

Атаки IP-спуфінга часто є відправною точкою для інших атак. Класичний приклад - атака DoS, яка починається з чужої адреси, що приховує справжню особу зловмисника. Зазвичай IP-спуфінг обмежується вставкою помилкової інформації або шкідливих команд у звичайний потік даних, переданих між клієнтським і серверним додатком або по каналу зв'язку між одноранговими пристроями. Для двостороннього зв'язку зловмисник повинен змінити всі таблиці маршрутизації, щоб направити трафік на помилкову IP-адресу. Деякі зловмисники, однак, навіть не намагаються отримати відповідь від додатків. Якщо головне завдання полягає в отриманні від системи важливого файлу, відповіді додатків не мають значення. Якщо ж зловмиснику вдається поміняти таблиці маршрутизації і направити трафік на помилкову IP-адресу, зловмисник отримає всі пакети і зможе відповідати на них так, ніби він є санкціонованим користувачем.

5. Атака типу man-in-the-middle;

Для атаки типу Man-in-the-Middle зловмисникові потрібен доступ до пакетів, що передаються по мережі. Такий доступ до всіх пакетів, що передаються від провайдера в будь-яку іншу мережу, може, наприклад, отримати співробітник цього провайдера. Для атак цього типу часто використовуються сніфери пакетів, транспортні протоколи і протоколи маршрутизації. Атаки проводяться з метою крадіжки інформації, перехоплення поточної сесії і отримання доступу до приватних мережевих ресурсів, для аналізу трафіку і отримання інформації про мережу та її користувачів, для проведення атак типу DoS, спотворення переданих даних і введення несанкціонованої інформації в мережеві сесії [4].

6. Ін'єкція ;

SQL-ін'єкція - це атака, в ході якої змінюються параметри SQL-запитів до бази даних. В результаті запит набуває зовсім інший зміст, і в разі недостатньої фільтрації вхідних даних здатний не тільки зробити вибірку конфіденційної інформації, а й змінити / видалити дані. Схема атаки SQL-ін'єкції показана на рис.1.1.

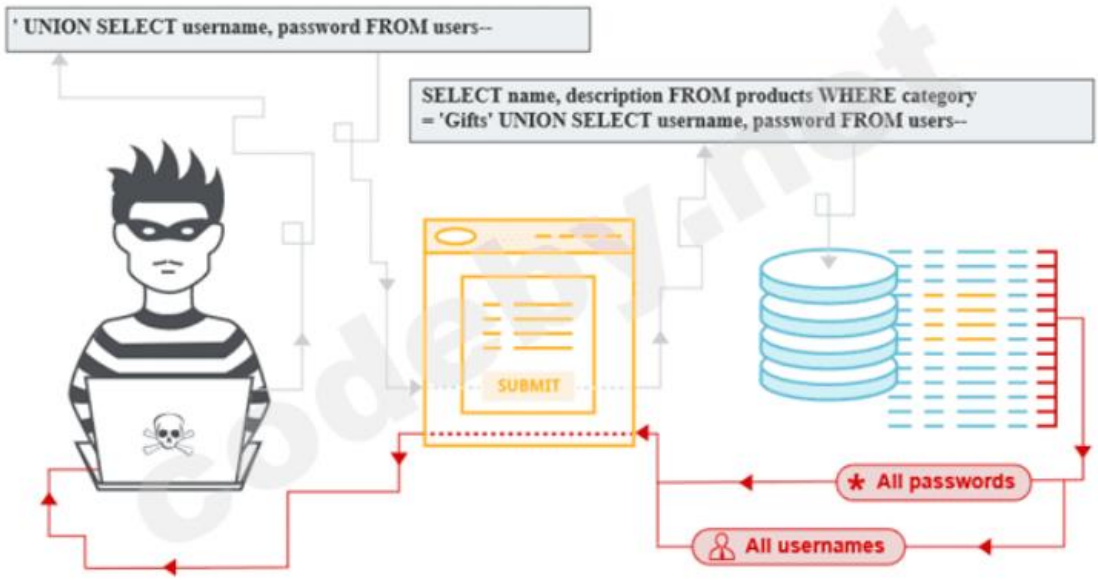


Рисунок 1.1 - Схема процесу SQL-ін'єкції

PHP-ін'єкція - один із способів злому веб-сайтів, що працюють на PHP. Він полягає в тому, щоб впровадити спеціально сформований зловмисний сценарій в код веб-додатку на серверній стороні сайту, що призводить до виконання довільних команд. Схема PHP-ін'єкції наведена нижче, на рис 1.2.

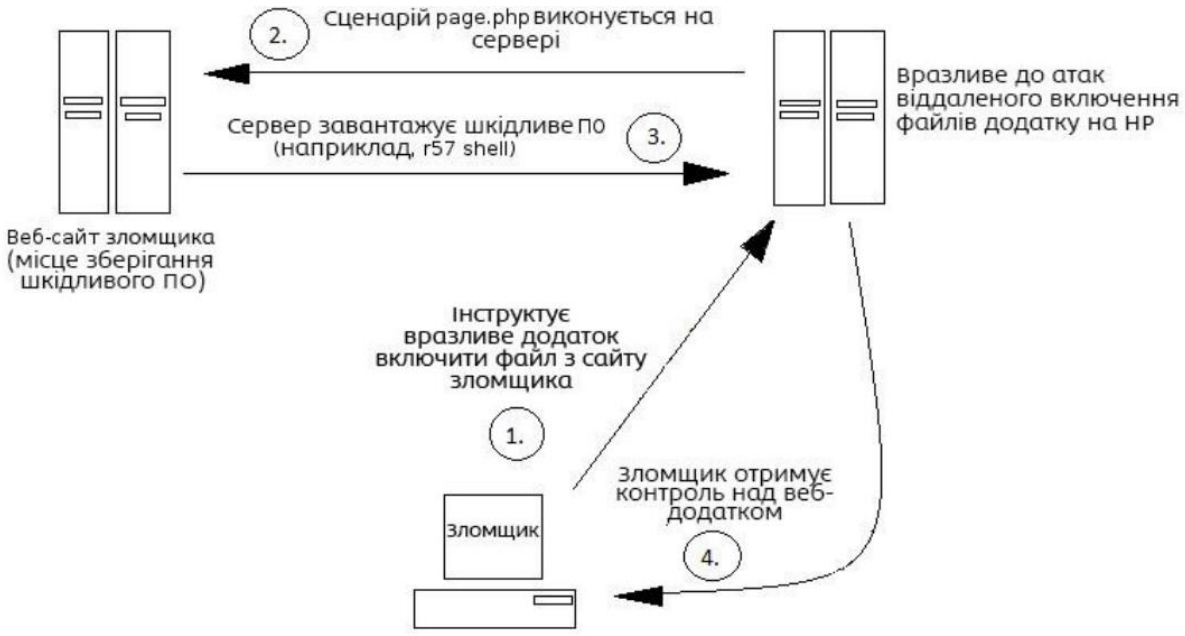


Рисунок 1.2 - Схема PHP-ін'єкції

7. XSS-атака;

XSS атака - це атака на вразливість, яка існує на сервері, що дозволяє впровадити в сторінку HTML, яка генерується сервером, якийсь довільний

код, в якому може бути взагалі все що завгодно і передавати цей код в якості значення змінної, фільтрація по якій не працює, тобто сервер не перевіряє дану змінну на наявність в ній заборонених знаків -, , ', ". Значення цієї змінної передається від HTML-сторінки що згенерувалася на сервері в скрипт, що її викликав шляхом відправки запиту. PHP-скрипт у відповідь на даний запит генерує HTML-сторінку, в якій відображаються значення потрібних зловмисникові змінних, і відправляє цю сторінку на браузер зловмисника. Тобто, кажучи простіше, XSS атака - це атака за допомогою вразливостей на сервері на комп'ютери клієнтів. XSS атака найчастіше використовується для крадіжки Cookies. У них зберігається інформація про сесії перебування користувача на сайтах, що і буває потрібним зловмисникам для перехоплення управління особистими даними користувача на сайті в межах, поки сесія не буде закрита сервером, на якому розміщений сайт. Крім цього в Cookies зберігається зашифрований пароль, під яким користувач входить на даний сайт, і при наявності необхідних утиліт і бажання, зловмисникам не дуже важко розшифрувати даний пароль.

Тепер опишемо інші можливості XSS атак (звичайно за умови їх успішного проведення).

- Можливість при відкритті сторінки ініціювати велику кількость непотрібних користувачеві вікон;
- Можливість переадресації на інший сайт (наприклад, на сайт конкурента);
- Можливість завантаження на комп'ютер користувача скрипта з довільним кодом (навіть шкідливого) шляхом впровадження посилання на виконуваний скрипт зі стороннього сервера;
- XSS атака може бути проведена не тільки через сайт, але і через вразливості в програмному забезпеченні (зокрема, через браузери). Тому рекомендується оновлювати програмне забезпечення що використовується. Також можливе проведення XSS атак через використання SQL-коду [5].

Виходячи з усього вищезазначеного, можливостей у XSS атак досить багато. Зловмисник може привласнити особисту інформацію, отримавши, навіть, паролі доступу до сайтів. До того ж XSS атака завдає шкоди виключно клієнтським машинам, залишаючи сервер в повністю робочому стані, і у адміністрації різних серверів часом мало стимулів встановлювати захист від цього виду атак. Розрізняють XSS атаки двох видів: активні і пасивні. При першому виді шкідливий скрипт зберігається на сервері і починає свою діяльність при завантаженні сторінки сайту в браузері клієнта. При другому виді атак скрипт не зберігається на сервері і шкідливий вплив починає виконуватися тільки в разі будь-якої дії користувача, наприклад, при натисканні на сформоване посилання [5].

8. XPath-ін'єкція;

XPath-ін'єкція - вид вразливостей, який полягає у впровадженні XPath-виразів в оригінальний запит до бази даних XML. XPath (XML Path Language) - це мова, яка призначена для довільного звернення до частин XML документа. XML (eXtensible Markup Language) - це відома мова розмітки, за допомогою якої створюються XML документи, що мають деревоподібну структуру.

9. Відмова в обслуговуванні (DoS- і DDoS- атаки);

DoS, поза всяким сумнівом, є найбільш відомою формою атак. Крім того, проти атак такого типу найважче створити стовідсотковий захист. Навіть серед зловмисників атаки DoS вважаються тривіальними, а їх застосування викликає зневажливі усмішки, тому що для організації DoS потрібно мінімум знань і вмінь. Проте, саме простота реалізації і величезна шкода яку завдають ці атаки є причиною пильної уваги адміністраторів, що відповідають за мережеву безпеку від DoS. Атаки DoS відрізняються від атак інших типів. Вони не націлені на отримання доступу до вашої мережі або на отримання з цієї мережі будь-якої інформації. Атака DoS робить вашу мережу недоступною для звичайного використання за рахунок перевищення допустимих меж функціонування мережі, операційної системи або програми. У разі використання деяких серверних додатків (таких як Web-сервер або FTP-сервер) атаки DoS можуть полягати в тому, щоб зайняти всі з'єднання,

доступні для цих додатків і тримати їх в зайнятому стані, не допускаючи обслуговування звичайних користувачів. В ході атак DoS можуть використовуватися звичайні інтернет протоколи, такі як TCP і ICMP (Internet Control Message Protocol). Більшість атак DoS спирається не на програмні помилки або проломи в системі безпеки, а на загальні слабкості системної архітектури. Деякі атаки зводять до нуля продуктивність мережі, переповняючи її небажаними і непотрібними пакетами або повідомляючи помилкову інформацію про поточний стан мережевих ресурсів. Цей тип атак важко попередити, так як для цього потрібна координація дій з провайдером. Якщо трафік, призначений для переповнення вашої мережі, не зупинений у провайдера, то на вході в мережу ви це зробити вже не зможете, тому що вся смуга пропускання буде зайнята. Коли атака цього типу проводиться одночасно через безліч пристроїв, ми говоримо про розподілеу атаки DoS (DDoS - distributed DoS) [6].

10. Phishing-атаки.

Phishing (фішінг) - процес введення в оману, або соціальна розробка клієнтів організацій для подальшого отриманнях їх ідентифікаційних даних та передачі їх конфіденційної інформації для злочинного використання[7]. Злочинці для свого нападу використовують spam або комп'ютери-боти. При цьому розмір компанії-жертви не має значення; якість особистої інформації, отриманої злочинцем в результаті нападу, має значення саме по собі.

1.1.1.3. Основні підходи та інструменти протидії веб-атакам

Використання інструментів для аналізу захищеності

Перевірку характеристик безпеки веб-сайту можна здійснювати за допомогою автоматизованих засобів. Такі засоби виконують тести на проникнення з використанням відомих шаблонізованих підходів.

До таких інструментів відносяться:

1. Програми та фреймворки
   - OpenVAS сканує вузли мережі на наявність вразливостей та дозволяє керувати вразливістю;

- OWASP Xenotix XSS Exploit Framework сканує ресурс на можливість експлуатації XSS-уразливостей.

2. Онлайн-сервіси
   - SecurityHeaders.io перевіряє на наявність та коректність заголовків відповіді сервера, що відповідають за безпеку веб-програми.
   - Observatory by Mozilla сканує ресурс на наявність проблем безпеки. Окрім своїх результатів, при виборі відповідної опції, збирає та додає до звіту аналітику зі сторонніх сервісів аналізу захищеності.
   - One button scan сканує на наявність вразливих компонентів ресурсу: DNS, HTTP-заголовки, SSL, чутливі дані тощо.
   - CSP Evaluator перевіряє правильність складання політики безпеки вмісту (CSP) та стійкість до XSS.
   - SSL Server Test виконує аналіз SSL конфігурації веб-сервера.
   - Snyk сканує JavaScript, Ruby та Java-програми на наявність вразливостей та, при необхідності, виправляє проблеми безпеки. Інтегрується з GitHub репозиторієм для проведення автоматичної перевірки та сповіщає про знайдені вразливості.

Перевірку також можна здійснювати і вручну, динамічно змінюючи значення HTTP запитів за допомогою налагоджувального проксі-сервера. (наприклад Fiddler).

Захист даних користувачів за допомогою HTTPS

HyperText Transfer Protocol Secure (HTTPS) — розширення HTTP, яке підтримує шифрування та захищає дані користувачів під час передачі в мережі Інтернет. HTTPS гарантує цілісність та конфіденційність взаємодії з сервером.

Використовувати HTTPS варто в тому випадку, якщо користувачі передають на сервер особисті дані: інформацію про кредитну картку, персональні дані та навіть адреси відвіданих сторінок. Якщо при надсиланні даних з форми авторизації встановлюються файли cookie, які потім відправляються при кожному запиті до сервера, зловмисник може отримати їх

і підробити запит до сервера. В результаті він перехопить сесію користувача. Щоб запобігти цьому, варто використовувати HTTPS на всіх сторінках сайту.

Якщо HTTPS вже налаштований, хорошою практикою є використання HTTP Strict Transport Security (HSTS) — заголовка відповіді сервера, який забороняє для домену використання незахищеного з'єднання.

Оновлення програмного забезпечення

Хакери регулярно виявляють і відразу застосовують нові вразливості операційних систем та іншого програмного забезпечення: HTTP-серверів або систем управління контентом (CMS).

Якщо ресурс розташований на сервері хостинг-провайдера, то встановлення оновлень для операційної системи входить до комплексу послуг. Інакше необхідно самостійно оновлювати операційну систему.

Якщо ресурс працює на базі платформи стороннього виробника (CMS), необхідно встановлювати оновлення безпеки одразу після випуску. Більшість розробників інформують про такі оновлення. Багато розробників використовують менеджери пакетів (наприклад, Composer, NPM або RubyGems) для встановлення залежних компонентів програм. У цих пакетах також виявляють уразливості, тому варто також слідкувати за їх оновленням. Щоб автоматично отримувати сповіщення про проблеми безпеки пакетів проекту, можна використовувати інструменти, такі як Gemnasium.

Запобігання SQL-інєкціям.

Простий запит на вибірку має вигляд:

```
SELECT * FROM WHERE column = 'parameter'
```

Якщо зловмисник змінить значення parameter на ' ' OR '1'='1, запит набуде наступного вигляду:

```
SELECT * FROM where column = '' OR '1' = '1'
```

Оскільки '1'дорівнює '1', атакуючий отримає доступ до всіх даних таблиці. Це дозволить виконати довільний запит, додавши до кінця виразу SQL. Вразливість запиту легко усунути за допомогою параметризації

[8]. Наприклад, для програми, написаної з використанням PHP та MySQL, він виглядає так:

```
$stmt = $pdo->prepare('SELECT * FROM table WHERE column = :value');
$stmt->execute(array('value' => $parameter));
```

Запобігання міжсайтовому скриптингу

Міжсайтовий скриптинг (XSS) - тип атаки на веб-ресурси, що полягає у введенні в сторінку сайту шкідливого коду, який виконується на комп'ютері користувача, змінює сторінку та передає вкрадену інформацію зловмиснику. Наприклад, якщо на сторінці коментарів немає перевірки вхідних даних, зловмисник впроваджує шкідливий код JavaScript. В результаті, у користувачів, які переглядають коментар, виконується код, і дані про авторизацію з cookies-файлів відправляються атакуючому.

Особливо схильні до цього виду атаки сучасні веб-додатки, де сторінки побудовані з контенту користувача, що інтерпретується фронтенд-фреймворками на кшталт Angular і Ember. У ці фреймворки вбудований захист від міжсайтового скриптингу, але змішане формування контенту на стороні сервера та клієнта створює нові комплексні атаки: використання директив Angular або хелперів Ember.

При перевірці варто зосередитись на контенті користувача, щоб уникнути некоректної інтерпретації браузером. Це схоже на захист від ін'єкцій SQL. При динамічній генерації HTML-коду потрібно використовувати спеціальні функції для зміни та отримання значень атрибутів (наприклад, element.setAttribute, element.textContent), а також шаблонізатори, які автоматично виконують екранізацію спеціальних символів.

Політика безпеки вмісту (CSP) – ще один інструмент захисту від XSS-атак. CSP — заголовки сервера, що визначають білий список джерел, звідки дозволено завантаження даних для різних типів ресурсів. Наприклад, заборона запуску скриптів із стороннього домену або вимкнення функції eval(). Завдяки політикам CSP навіть при впровадженні шкідливого коду до сторінки його виконання стає неможливим.

Перевірка та шифрування паролів

Зберігати паролі варто у вигляді хешів, причому краще використовувати алгоритми однобічного хешування, наприклад, SHA. Якщо зловмисник зламає ресурс і отримає хешовані паролі, збитки будуть знижені за рахунок того, що хеш має необоротну дію і отримати вихідні дані практично неможливо. Але хеші на популярні паролі легко перебираються за словником, тому використовуйте також «сіль», унікальну для кожного пароля. Тоді злам великої кількості паролів стає ще повільнішим і потребує великих обчислювальних витрат[10].

Що стосується валідації, варто встановити обмеження на мінімальну довжину пароля, а також здійснювати перевірку на збіг з логіном, e-mail та адресою сайту.

На щастя, більшість CMS надають інструменти управління політиками безпеки, але для використання «солі» або встановлення мінімальної складності пароля іноді потрібне додаткове налаштування або встановлення модуля. При використанні .NET варто застосувати провайдери членства, тому що в них закладено вбудовану систему безпеки з великою кількістю налаштувань та готовими елементами для автентифікації та зміни пароля.

Контроль процесу завантаження файлів

Завантажений файл, який на перший погляд виглядає нешкідливо, може містити скрипт і при виконанні на сервері відкриє зловмиснику доступ до сайту. Розширення або MIME-тип легко підробити, читання заголовка або використання функцій перевірки розміру зображення не дають 100% гарантії, більшість форматів зображень можливо впровадити код PHP, який буде виконаний на сервері[11].

Щоб запобігти цьому, необхідно заборонити виконання завантажуваних файлів користувачами.

Заходи захисту веб-застосунків для власників власних серверів[12]:

1. Налаштування міжмережевого екрану, у тому числі блокування портів, що не використовуються;

2. За наявності доступу до сервера з локальної мережі варто створити демілітаризовану зону (DMZ), відкривши доступ із зовнішнього світу лише до портів 80 та 443;
3. За відсутності доступу до сервера з локальної мережі необхідно використовувати захищені методи (SFTP, SSH та ін.) для передачі файлів та управління сервером ззовні;
4. Варто виділити окремий сервер для баз даних, який не буде доступний безпосередньо з зовнішнього світу;
5. Відмежування фізичного доступу до сервера.

Перевірка вхідних даних

Необхідно контролювати дані, отримані з веб-форм, як на стороні клієнта, так і на стороні сервера. У браузері перевіряються прості помилки на кшталт незаповненого обов'язкового поля або тексту, введеного в числове поле. Ці перевірки обходяться, тому контроль на сервері обов'язковий. Відсутність перевірки на сервері призводить до експлуатації зловмисником ін'єкцій та інших видів атак.

Розподіл прав доступу до файлів

Дозволи файлу (file permissions) визначають, хто і що може з ним робити[11]. У *nix системах для файлів існує 3 варіанти доступу, які подаються у вигляді цифр:

1. «Read» (4) - читання вмісту файлу;
2. «Write» (2) - зміна вмісту файлу;
3. «Execute» (1) - виконання програми або скрипта.

Щоб встановити множинні дозволи, достатньо скласти їх числові значення:

1. «Читання» (4) + «запис» (2) = 6;
2. «Читання» (4) + «запис» (2) + «виконання» (1) = 7.

При розподілі прав користувачі поділяються на 3 типи:

1. «Owner» (власник) - творець файлу (редагується, але може бути тільки один);
2. «Group» (група) - група користувачів, які отримують дозволи;
3. «Others» (інші) - інші користувачі.

При установці CMS дозволи, як правило, встановлюються коректно з точки зору безпеки. Проте в Інтернеті часто рекомендують вирішувати проблеми прав доступу установкою на всі файли значення 666 або 777. Ця порада допомагає вирішити проблему, але відкриває серйозну вразливість, тому що всім користувачам надається право змінити (вставити шкідливий код) або видалити файли на сервері. Варто розподілювати права доступу до файлів на сервері відповідно до завдань користувачів.

1.1.2. Дослідження ринку засобів захисту веб-додатків

Засоби захисту веб-додатків (програм) — програмні або апаратно-програмні комплекси, призначені для забезпечення захисту веб-програм та різних веб-сервісів. До засобів захисту веб-сайтів можна віднести:

1. Web Application Firewall (WAF);
2. Засоби аналізу веб-сайтів на наявність вірусів;
3. Балансувальники навантаження на веб-програми;
4. Засоби захисту від DDoS;
5. Сканери безпеки веб-застосунків (WASS).

Засоби захисту веб-сайтів забезпечують захист від таких типів загроз:

1. Загроза зміни вмісту веб-сайту;

Ця атака дозволяє зловмиснику розмістити на веб-сайті будь-яку інформацію, включаючи текст сторінок, а також дані платіжних реквізитів.

2. Загроза видалення даних;

Ця атака дозволяє зловмиснику видалити всі дані, включаючи інформацію про паролі, бази даних клієнтів, постачальників, товарів та інші.

3. Загроза застосування шкідливих програм;

Внаслідок такої атаки користувачі та власники сайту можуть не відразу помітити будь-які зміни. Тим не менш, шкідлива програма зможе робити шкідливі дії, включаючи переадресацію на шкідливий або шахрайський сайт, крадіжку персональних даних користувачів, зараження відвідувачів веб-сайту вірусами і так далі.

4. Загроза розсилки спаму;

В результаті цієї атаки веб-додаток використовуватиметься зловмисниками для розсилки небажаних повідомлень. Це загрожує занесенням веб-сайту в «спам-лісти», що не дозволить надалі надсилати санкціоновані повідомлення.

5. Загроза збільшення навантаження на ресурси або DDoS-атака.

Під цим передбачається надсилання на адресу веб-сервера спеціальних запитів, які призведуть до утруднення доступу користувачів до сайту або падіння прикладного та системного програмного забезпечення веб-сервера[14].

Щоб вибрати засіб захисту веб-додатку, слід звертати увагу на його можливості. До найбільш корисних функцій та властивостей засобу захисту веб-додатків можна віднести:

1. Швидкість обробки трафіку – засоби захисту не повинні впливати на роботу користувачів із веб-сайтом;
2. Блокування нелегітимних запитів та попередження атаки типу «Відмова в обслуговуванні»;
3. Робота з протоколами http/https і web-sockets;
4. Виявлення актуальних уразливостей та виявлення загроз інформаційній безпеці;
5. Блокування атак перебору паролів та збору інформації;
6. Актуальна база сигнатур атак, що постійно оновлюється, і IP-reputation (база даних, що зберігає інформацію про те, наскільки може бути небезпечний веб-сайт);
7. Віртуальний патчинг, який дозволяє закрити вразливість до того, як код веб-програми буде виправлено;
8. Мінімальний відсоток помилкових спрацьовувань.

Для адміністраторів засобів захисту важливою функцією в засобах захисту веб-сайтів будуть зручний інтерфейс, інформування про атаки (на електронну пошту, по SMS тощо) і можливість створювати персоналізовані звіти.

1.1.2.1. WAF

Web Application Firewall (скорочено - WAF) - засоби фільтрації трафіку прикладного рівня, спеціально орієнтовані на веб-програми. Застосування Web Application Firewall традиційно вважається найефективнішим підходом захисту веб-ресурсів. WAF може бути реалізований як хмарний сервіс, агент на веб-сервері або спеціалізований апаратний або віртуальний пристрій.

Традиційно вважається, що прикладний рівень - це останній рівень моделі і вище за нього розташовуються тільки дані кінцевих додатків, які не можуть бути формалізовані і згруповані. Однак з розвитком стандартів представлення інформації прикладними сервісами вже можна говорити про те, що, частково, дані, якими оперують певні групи додатків, добре формалізуються, і правила їх подання, по суті, є деякими пропрієтарними протоколами або, спрощено, закономірностями.

Таким чином, можна говорити про появу нового рівня міжмережевої взаємодії, що прихована для класичних міжмережевих екранів прикладного рівня. Новий клас пристроїв – Web Application Firewall – характеризується здатністю розуміти групи протоколів та залежностей, властивих для веб-застосунків, які будуються над прикладними протоколами http/https.

Класичне розміщення WAF в мережі — в режимі зворотного проксі-сервера, перед веб-серверами, що захищаються. Залежно від виробника можуть підтримуватися й інші режими роботи - наприклад, прозорий проксі-сервер, міст або пасивний режим, коли продукт працює з реплікацією трафіку.

Після встановлення WAF і пуску продуктивного трафіку відразу ж починає роботу основний компонент захисту - машинне навчання, в ході якого складається еталонна модель комунікації з об'єктом захисту, і таким чином формується білий список допустимих ідентифікаторів доступу. На даний момент у веб-застосунках використовуються три типи ідентифікатора доступу: HTTP-параметри (в представленнях типу: Raw, XML, JSON), ідентифікатор ресурсу (URL, URN), ідентифікатор сесії (cookie). Завдання WAF полягає у визначенні допустимих значень ідентифікаторів для веб-

програми. З певних значень згодом складатиметься еталонна (позитивна) модель. Включення конкретних значень ідентифікатора в модель здійснюється на основі застосування математико-статистичного алгоритму, коли всі ресурси веб-застосунку додані в позитивну модель, адміністратор системи повинен переконатися у відсутності значної кількості помилково-позитивних спрацьовувань і переключити систему в режим блокування. Крім машинного навчання, у набір функцій WAF зазвичай входять такі типові механізми захисту:

1. валідація протоколу;
2. сигнатурний аналіз;
3. захист від ін'єкцій та XSS (часто пропрієтарний);
4. можливість створення власних правил захисту;
5. інтеграція з репутаційними та фрод-сервісами;
6. інтеграція з іншими пристроями у ландшафті ІБ компанії.

Пріоритетом виробника WAF є сфокусованість власних дослідницьких центрів на генерації оновлень політик безпеки для своїх пристроїв з урахуванням актуальних загроз веб-додаткам. Так з'являються, наприклад, сигнатури атак, притаманні конкретним веб-фреймворків та системам управління контентом або пропрієтарні механізми захисту від XSS та SQL-ін'єкцій.

1.1.2.2. Засоби аналізу веб-сайтів на наявність вірусів

Засоби перевірки веб-сайту на віруси — послуга, під час якої виконується пошук шкідливих програм у каталогах веб-сайту та його коді, внаслідок чого ідентифікуються можливі загрози з метою їхнього подальшого усунення[16]. Існує кілька способів та засобів перевірки веб-сайту на наявність шкідливих програм:

1. Сервіси хостинг-провайдера ;

Дозволяють перевірити файли та бази даних на наявність серверних шкідливих скриптів та ін'єкцій.

2. Онлайн-сервіси , що виконують статичний та динамічний аналіз коду веб-сторінки;

Дозволяють перевірити сторінки сайту на наявність шкідливих ін'єкцій. У такий спосіб можна перевірити будь-який веб-сайт в інтернеті. Для цього достатньо ввести адресу веб-сайту в спеціальний рядок для перевірки, при цьому не потрібно надавати доступ по SSH, FTP та ін. Такі сервіси дозволяють перевірити чи наявні на сайті небезпечні віджети та скрипти, спам-контент, а також чи спричинить перехід на даний веб-сайт перенаправлення на інший заражений веб-сайт (приховані редиректи).

3. Антивірусні програми ;

Дозволяють виконати локальну перевірку файлів веб-сервера на наявність шкідливого коду, а за наявності функції перевірки вихідного коду сторінок у процесі переходу на сайт здатні негайно сигналізувати про небезпеку.

4. Спеціальні плагіни для браузерів.

Призначені для звичайних користувачів. Такі плагіни перевіряють інтернет-сторінки та файли, які можуть бути завантажені в Інтернеті.

1.1.2.3. Балансувальники навантаження на веб-додатки

Балансування навантаження – це спосіб розподілу навантаження між кількома мережевими пристроями, такими як сервери, маршрутизатори, міжмережні екрани. Такий підхід забезпечує оптимальне використання потужностей, а також збільшення швидкості реагування серверів на запити та забезпечення відмовостійкості. Можна налаштувати балансування навантаження на багаторівневі комутатори, DNS-сервери, серверні кластери, проксі-сервери, веб-сервери, мережеві адаптери, віртуальні машини, програми та сервери інспектування вмісту.

Балансування навантаження може виконуватись на різних рівнях моделі OSI:

1. Мережевий;

Такий метод має на увазі використання однієї IP-адреси на різних мережевих пристроях, серверах. Існує кілька способів балансування навантаження на мережному рівні:

- DNS-балансування, коли одному домену надано кілька IP-адрес;
- NLB-кластер, або об'єднання кількох серверів у кластер;
- За територіальним ознакою, коли у різних регіонах налаштовуються однакові послуги, які мають однакові IP-адреси.

2. Транспортний;

Такий спосіб передбачає централізоване розподілення запитів до сервера між різними пристроями. При такому балансуванні навантаження можуть використовуватися різні алгоритми (наприклад, запит надсилатиметься найменш завантаженому серверу або серверу, який знаходиться наступним у черзі, так званий круговий перебір).

3. Прикладний.

При такому балансуванні навантаження існує певна програма, яка самостійно приймає рішення про перенаправлення клієнтського запиту на відповідний сервер залежно від того, до чого звертається клієнт. Цей спосіб застосовується при роботі з базами даних (додаток може перенаправляти запити на читання одного сервера, а запити на запис — до іншого).

Балансування навантаження широко використовується в корпоративних системах захисту, забезпечуючи рівномірне навантаження на сервери, для захисту веб-сервісів та сайтів, системах фільтрації веб-контенту тощо. Також існують різні онлайн-сервіси з організації балансування за певним методом.

1.1.2.4. Засоби захисту від DDoS

Засоби захисту від DDoS-атак – це спеціалізовані програмно-апаратні та програмні засоби, призначені для захисту вер-серверів (веб-сайтів) від розподілених атак типу «Відмова в обслуговуванні». Даний тип атак націлений на порушення доступності (повне або часткове) різних публічних сервісів, у тому числі веб-сайтів, баз даних та серверів додатків, при якому легітимні користувачі втрачають можливість отримання доступу. Принцип атаки полягає у генерації великої кількості паразитного трафіку, на обробку якого виділятиметься безліч обчислювальних ресурсів[19]. Внаслідок атаки організації можуть зазнати великих фінансових втрат через те, що їхні

замовники не зможуть скористатися сервісами компанії. Загалом захист від DDoS можна організувати двома способами:

1. Скориставшись послугами зовнішніх компаній за моделлю SaaS (Security as a Service);

Таке рішення дозволить уникнути глибокого занурення у технічну складову питань захисту від DDoS-атак , а також скоротить витрати на інформаційну безпеку.

2. Шляхом запровадження власних засобів захисту від атак типу «Відмова в обслуговуванні», а також застосування організаційно-технічних заходів, спрямованих на збільшення пропускної спроможності каналів зв'язку та використання механізмів балансування навантаження.

Засоби захисту від DDoS-атак on-premise(розміщені в локальній мережі) зазвичай розташовуються на межі мережі у розрив, для своєчасного виявлення та фільтрації шкідливого чи паразитного трафіку. Зазвичай представляють собою окреме обладнання, або розташовуються на віртуальній машині. Такі засоби вимагають постійного оновлення, оскільки сценарії DDoS-атак постійно змінюються. Ціна на спеціалізоване обладнання, як правило, залежить від заявленої швидкості обробки трафіку та пропускної спроможності. Враховуючи високу вартість засобів захисту, оптимальний комплекс може не впоратися з обробкою великої DDoS-атаки.

Щоб підвищити ефективність захисту, можуть використовуватися гібридні рішення, які крім вбудовування в інфраструктуру організації, здатні підключатися до центру фільрації трафіку, який знаходиться в хмарі. Такий спосіб дозволяє захиститися і від атак на інтернет-канал, однак у разі атаки потрібен час для перенаправлення трафіку на центр фільтрації. Існують також повністю хмарні рішення, які можуть пропускати трафік через хмару до об'єкта, що захищається в режимі on-demand (на вимогу), а також always-on (постійно).

Крім зазначених функцій, засоби захисту від DDoS-атак надають звіти за результатами атак, що здійснювалися на ресурси організації. Це дозволяє здійснити аналіз і в разі необхідності зробити модифікацію інфраструктури, а також оцінити достатність використовуваного засобу захисту.

1.1.2.5. Сканери безпеки веб-застосунків.

Аудит безпеки веб-сайту — це комплекс послуг та/або програмних засобів, які застосовуються для виявлення існуючих вразливостей веб-ресурсу. Аудит безпеки веб-сайтів необхідний, оскільки будь-яка вразливість може бути використана зловмисниками, що спричинить компрометацію організації та може завдати як фінансової, так і репутаційної шкоди.

Програмні або апаратні засоби для проведення аудиту, вони ж сканери вразливостей веб-застосунків, також дозволяють здійснювати пошук вразливостей в операційній системі сервера веб-додатків, системному та прикладному програмному забезпеченні, включаючи веб-середовище сервера та програмний код сайту. Сканери вразливостей, або сканери захищеності веб-застосунків мають різні абревіатури - WAS, або Web Application Scanning, WASVS, або Web Application Security Vulnerability Scanners, а також AST, що розшифровується як Application Security Testing. Аудит безпеки сайтів проводиться для того, щоб забезпечити стабільну роботу ресурсу та запобігти втраті даних. Аудит також дозволить уникнути наступних загроз безпеці:

1. Загроза отримання зловмисником несанкціонованого доступу до внутрішніх ресурсів підприємства;

Це може статися через те, що веб-сайти часто розміщуються на ресурсах організації. Як тільки зловмисник отримує доступ і права на управління веб-сайтом, висока ймовірність, що йому вдасться отримати доступ до інших критичних ресурсів організації.

2. Загроза порушення конфіденційності інформації, що належить компанії;

Ця загроза пов'язана з попередньою — якщо зловмисник зміг дістатися важливих інформаційних систем, йому не важко знайти конфіденційні дані.

3. Загроза отримання несанкціонованого доступу до внутрішньої системи платежів та переказів коштів;

У випадку, коли веб-сайт служить торговим майданчиком або надає можливість зовнішнім особам сплачувати за покупки, зловмисник зможе отримати доступ до такої системи та використовувати її в корисливих цілях.

4. Загрози витоку персональних даних;

Багато веб-ресурсів вимагають персональних даних під час реєстрації на сайті. Якщо вразливість дозволить зловмиснику отримати доступ до таких даних, він надалі зможе використовувати її для власної вигоди, продажу персональних даних або розсилки спаму.

5. Загроза порушення цілісності даних сайту.

Під цим передбачається зміна чи модифікація наявного контенту.

Аудит безпеки веб-сайтів дозволяє відобразити повну картину захищеності веб-ресурсу, виявити критичні вразливості та запобігти можливим атакам. Однією з функцій програмних засобів аудиту безпеки веб-сайтів є надання рекомендацій щодо усунення слабких місць у захисті. Загальний принцип роботи сканерів захищеності полягає у трьох кроках:

1. Визначення об'єкта сканування;
2. Здійснення сканування;

При цьому сканери дозволяють виявляти не тільки вразливість, але й помилки в коді, а також відповідність різним стандартам.

3. Надання звіту про сканування, який включає рекомендації для виправлення знайдених вразливостей.

1.2. Обґрунтування вибору підходів і технологій для створення інформаційних управляючих систем

Згідно з класифікацією засобів захисту веб-додатків, що представлена у розділі 1.2, пропоновану інформаційну систему можна віднести до динамічних сканерів вразливостей веб-додатків. Такий клас систем передбачає взаємодію з робочим додатком «ззовні», а тому існує ряд вимог до їх функціонування:

1. Здійснення постійного моніторингу;
2. Наявность засобів миттєвого інформування та реагування у випадку виявлення проблем;
3. Можливість швидкого оновлення та підключення нових правил перевірки, середовищ збору даних тощо;
4. Забезпечення високого рівня безпеки сховища даних;
5. Можливість інтеграції з іншими рішеннями забезпечення інформаційної безпеки.

В якості технологічної платформи було обрано створення саме веб-додатку, адже це надає ряд суттєвих переваг:

1. Забезпечення високої доступності системи;

Доступність системи контролює розробник централізовано;

2. Централізоване постачання оновлень;

У випадку випуску оновлень, вони стають доступними всім користувачам одночасно.

3. Загальне інформування щодо виявлення проблем;

Всі користувачі, котрі мають відношення до конкретного веб-сайту, отримують своєчасні сповіщення.

4. Стандартизація рівня безпеки сховища даних;

Політики безпеки застосовуються до всіх даних, що зберігаються в БД, адже сервер БД є частиною єдиного середовища.

5. Розширений спектр можливостей для інтеграції.

Веб-додаток розробляється у вигляді RESTful розподіленої системи, задовольняючи наступні критерії:

1. Клієнт-сервера архітектура;

Система має бути поділена на клієнти та на сервери. Поділ інтерфейсів означає, що, наприклад, клієнти не пов'язані зі зберіганням даних, яке залишається всередині кожного сервера, тому мобільність коду клієнта покращується. Сервери не пов'язані з інтерфейсом користувача або станом, так що сервери можуть бути простішими і масштабованішими. Сервери та клієнти

можуть бути замінені та розроблятися незалежно, доки інтерфейс не змінюється.

2. Відсутність стану;

Сервер не повинен зберігати будь-яку інформацію про клієнтів. У запиті повинна зберігатися вся необхідна інформація для обробки запиту та, якщо необхідно, ідентифікації клієнта.

3. Кешування;

Кожна відповідь повинна бути відмічена, чи є вона кешується чи ні, для запобігання повторному використанню клієнтами застарілих або некоректних даних у відповідь на подальші запити.

4. Єдиний інтерфейс.

Єдиний інтерфейс визначає інтерфейс між клієнтами та серверами. Це спрощує та відокремлює архітектуру, яка дозволяє кожній частині розвиватися самостійно.

Визначення основних джерел моніторингу та правил валідації отриманих даних здійснюється на основі реальних потенційних кроків кіберзлочинців на етапі розвідувальної діяльності:

1. Сканування портів сервера;
2. Перевірка версій сервісів;
3. Перевірка SSL сертифіката;
4. Перевірка заголовків HTTP пакетів;
5. Перевірка контенту, пов’язаного з цільовим доменом.

Підключення веб-сайту для моніторингу повинно здійснюватись лише після перевірки права власності на сайт, адже в руках кіберзлочинців отримана інформація може бути використана для подальших кібератак. Проектована система включатиме наступні способи:

1. Перевірка наявності відповідного мета-тегу на головній сторінці веб-сайту;
2. Перевірка інформації про реєстранта домену.

2. АНАЛІТИЧНИЙ РОЗДІЛ. ХАРАКТЕРИСТИКА ІС МОНІТОРИНГУ БЕЗПЕКИ ВЕБ-САЙТІВ

2.1. Структура і характеристика системи

Робота ІС моніторингу безпеки веб-сайтів полягає в скануванні веб-програми на наявність вразливостей і виглядає так:

1. вибір об'єкта сканування;
2. проведення сканування (оцінки безпеки) веб-додатка (може проводитись аналіз на наявність системних помилок та вразливостей, а також на предмет відповідності різним стандартам безпеки);
3. формування звітності (про знайдені системні помилки та уразливості, про відповідність стандартам безпеки) та рекомендації щодо усунення помилок, уразливостей та невідповідностей стандартам безпеки.

ІС моніторингу безпеки веб-сайтів може використовуватися на постійній основі, а також разово, коли моніторинг запускається лише адміністратором за необхідності проведення перевірки. У разі використання автоматизованого функціоналу, (сканування проводиться автоматично за заданим розкладом) доступна функція оперативного оповіщення відповідальних осіб про знайдені вразливості.

Продукти, що належать до даного класу систем, виконують аналіз та тестування додатків з використанням кількох методів:

1. Static AST (SAST): статичне сканування безпеки додатків, у якому здійснюється аналіз вихідних джерел веб-додатка (коду), зазвичай, на стадіях його програмування і тестування;
2. Dynamic AST (DAST): динамічне сканування безпеки програм, при якому здійснюється аналіз робочої програми;
3. Interactive AST (IAST): інтерактивне сканування безпеки програм, при якому здійснюється аналіз програм за допомогою комбінованого підходу (з використанням SAST та DAST).

В основу ІС, що описана в даній роботі, закладено метод для перевірки програм DAST. Для веб-застосунків він є переважним, тому що в основному аналіз захищеності проводиться для вже працюючих і запущених у виробничу експлуатацію веб-додатків (метод чорної скриньки).

2.1.1. Класи користувачів ІС

Дана система передбачає собою SaaS-сервіс, що надає механізм постійного моніторингу стану інформаційної безпеки веб-ресурсу, автономну ідентифікацію та інформування про визначений перелік потенційних загроз, постійне оновлення та актуалізацію інформації про вразливості як з внутрішніх так і з зовнішніх джерел.

Система передбачає наявність засобів побудови звітності та рекомендацій щодо вирішення проблем інформаційної безпеки веб-ресурсу.

Таблиця 2.1.

Класи користувачів ІС

№	Клас користувачів	Відмінні риси
1	Власник веб-ресурсу	Визначення переліку відповідальних осіб для отримання сповіщень щодо результатів моніторингу, аналіз звітності щодо стану функціонування ресурсу(ів)
2	Адміністратор безпеки	Реєстрація користувачів, отримання сповіщень щодо стану функціонування ресурсу(ів), налаштування параметрів моніторингу, підключення до моніторингу нових ресурсів
3	Технічний менеджмент	Формування та аналіз звітності щодо стану функціонування ресурсу(ів)

2.1.2. Опис варіантів використання ІС

Варіант використання «Підключення веб-ресурсу»:

Дійові особи: Адміністратор безпеки

Стислий опис: Адміністратор реєструє нове доменне ім’я в системі для подальшого моніторингу стану інформаційної безпеки.

Передумови:

1. Веб-ресурс є доступним для зовнішніх запитів;
2. Адміністратор має доступ до інтерфейсів управління веб-ресурсом.

Післяумови:

1. Дані про веб-ресурс зберігаються в системі.

Основний потік подій:

1. Адміністратор ініціює процес запитом на додавання нового доменного імені;
2. Адміністратор особисто обирає спосіб підтвердження права власності на веб-сайт(мета-тег на головній сторінці сайту, інформація про реєстранта домену);
3. Адміністратор виконує умови відповідного способу підтвердження або впевнюється в їх попередньому виконанні;
4. Адміністратор підтверджує завершення налаштувань;
5. Система здійснює перевірку права власності одним з визначених способів;
6. Система сповіщає про успішність перевірки права власності та ініціює збереження даних(А1);
7. Варіант використання завершено.

Альтернативні потоки подій:

А1. Право власності не підтверджено

1. Система надає детальну інформацію про помилку;
2. Система повертається в стан кроку №2 основного потоку подій.

Варіант використання «Налаштування моніторингу»:

Дійові особи: Адміністратор безпеки

Стислий опис: Адміністратору необхідно вказати ключові налаштування моніторингу конкретного веб-сайту.

Передумови:

1. Веб-сайт вже зареєстровано в системі.

Післяумови:

1. Моніторинг здійснюється відповідно до вказаних налаштувань.

Основний потік подій:

1. Адміністратор ініціює процес запитом переліку підключених веб-сайтів;
2. Адміністратор обирає необхідний сайт;
3. Адміністратор вказує частоту моніторингу, період збереження даних в БД сервісу, необхідність шифрування даних, а також налаштування сповіщень;
4. Адміністратор ініціює збереження даних;
5. Варіант використання завершено.

Варіант використання «Ініціювання моніторингу»:

Дійові особи: Адміністратор безпеки.

Стислий опис: Незалежно від автоматичного запуску моніторингу з визначеною періодичністю, адміністратор має змогу вручну запускати як повний цикл моніторингу, так і окремі його складові.

Основний потік подій:

1. Адміністратор безпеки ініціює процес запитом переліку підключених веб-сайтів;
2. Адміністратор безпеки обирає веб-сайт;
3. Адміністратор безпеки вказує, який саме етап моніторингу бажає ініціювати(або ж цикл загалом);
4. Адміністратор ініціює запуск вказаного етапу, або циклу загалом;
5. Варіант використання завершено.

Варіант використання «Отримання результатів про стан ІБ веб-сайту»:

Дійові особи: Технічний менеджмент.

Стислий опис: Технічний менеджмент здійснює перегляд історії циклів моніторингу для ознайомлення з результатами та переліком рекомендацій.

Передумови:

1. Наявність завершених циклів моніторингу

Основний потік подій:

1. Менеджер ініціює процес запитом переліку завершених циклів моніторингу;
2. Менеджер обирає необхідний пункт серед історії циклів моніторингу;
3. Система надає інформацію щодо результатів кожного етапу;
4. Менеджер здійснює запит на формування переліку рекомендацій;
5. Менеджер здійснює перегляд та аналіз рекомендацій;
6. Варіант використання завершено.

Варіант використання «Отримання звітності за період»:

Дійові особи: Власник веб-сайту.

Стислий опис: Власник веб-сайту отримує інформацію щодо динаміки рівня ІБ за певний період.

Передумови:

1. Наявність завершених циклів моніторингу.

Основний потік подій:

1. Власник веб-сайту ініціює процес запитом переліку підключених доменних імен;
2. Власник обирає доменне ім’я, вказує дату початку та дату звершення періоду;
3. Власник обирає формат звіту (.html, .pdf, .xslx);
4. Власник ініціює завантаження файлу звіту;
5. Варіант використання завершено.

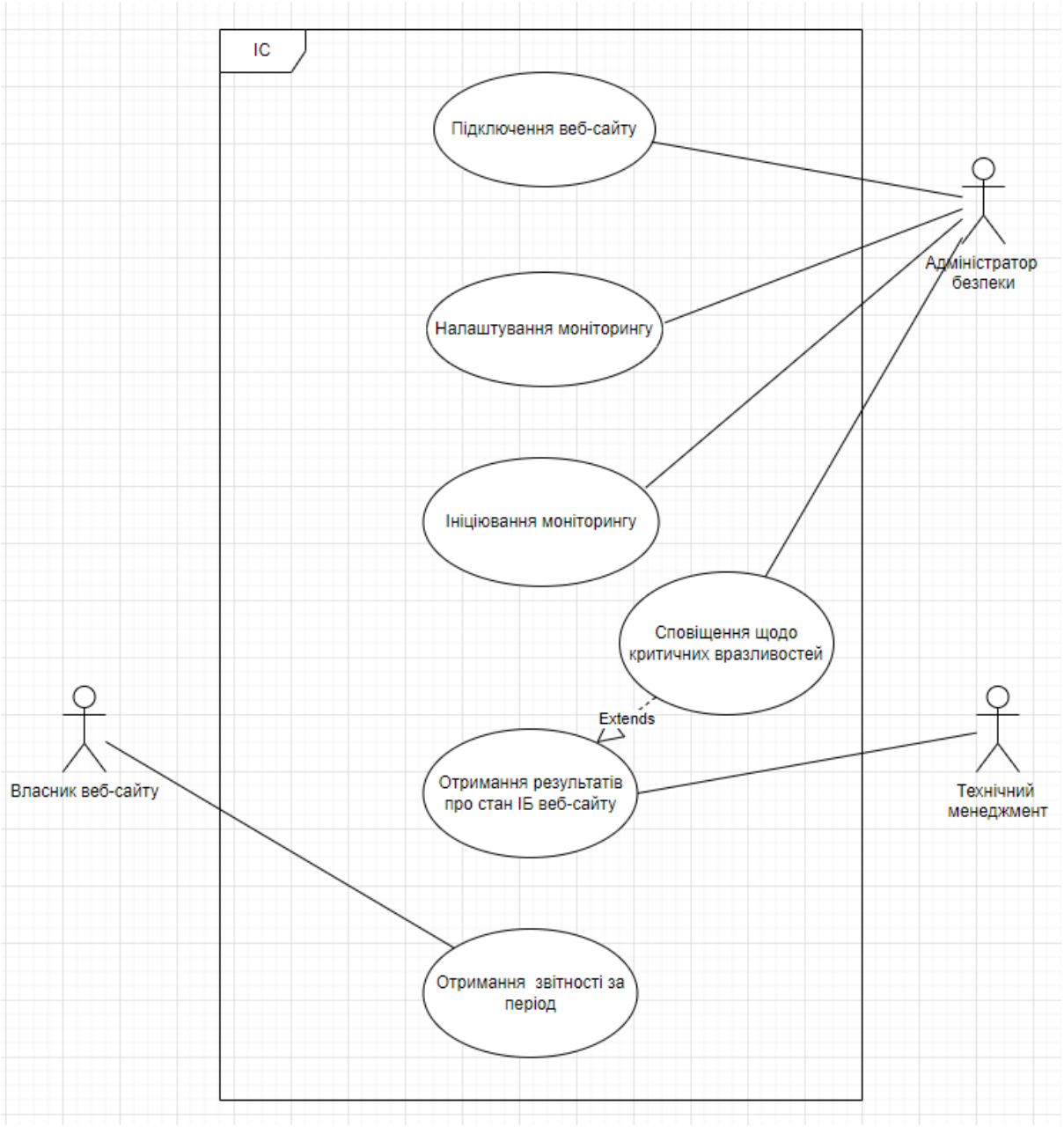


Рисунок 2.1 - Загальна діаграма варіантів використання системи

Опишемо кожен процес за допомогою діаграм послідовності.

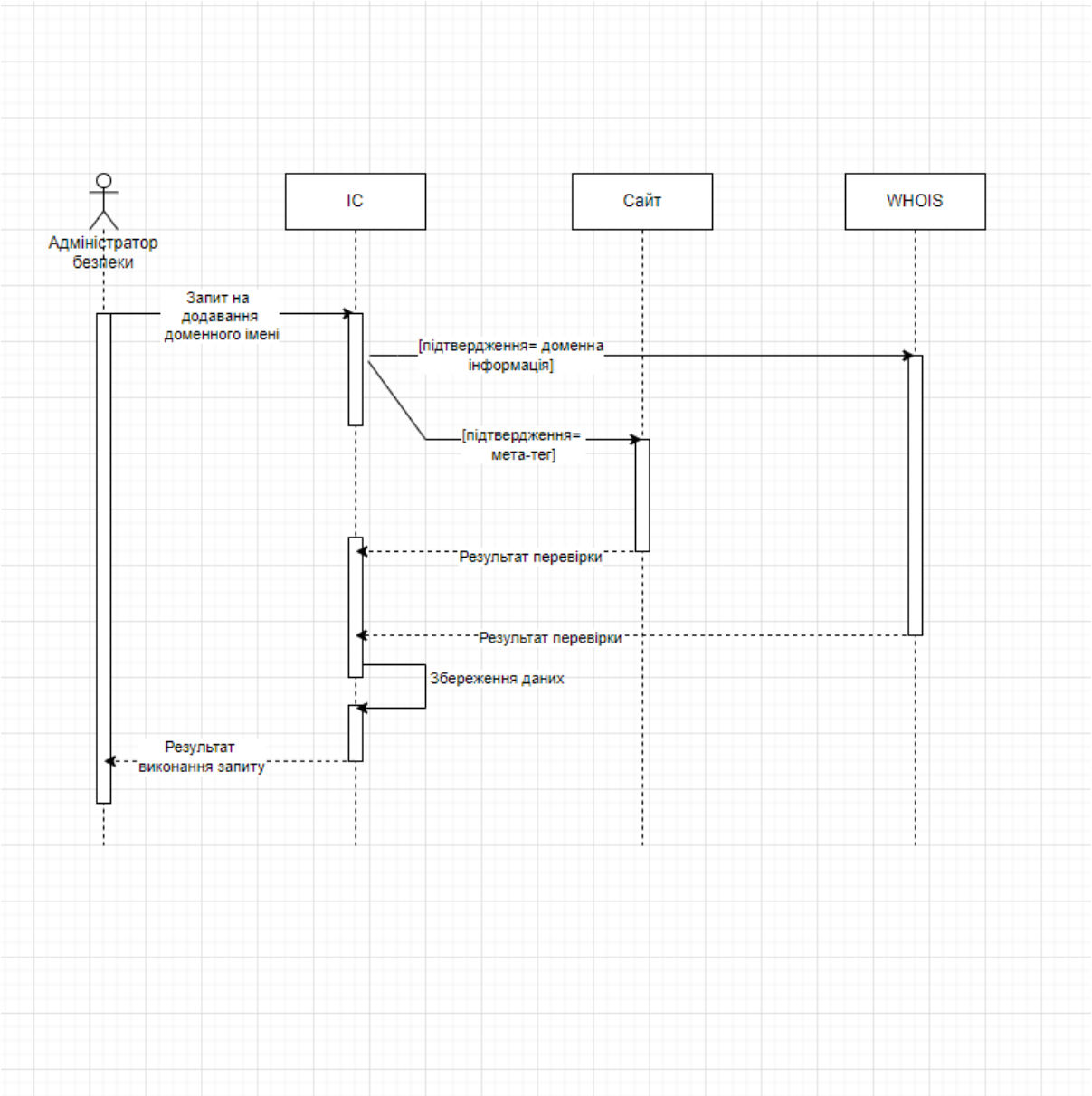


Рисунок 2.2 - Діаграма послідовності процесу підключення веб-сайту

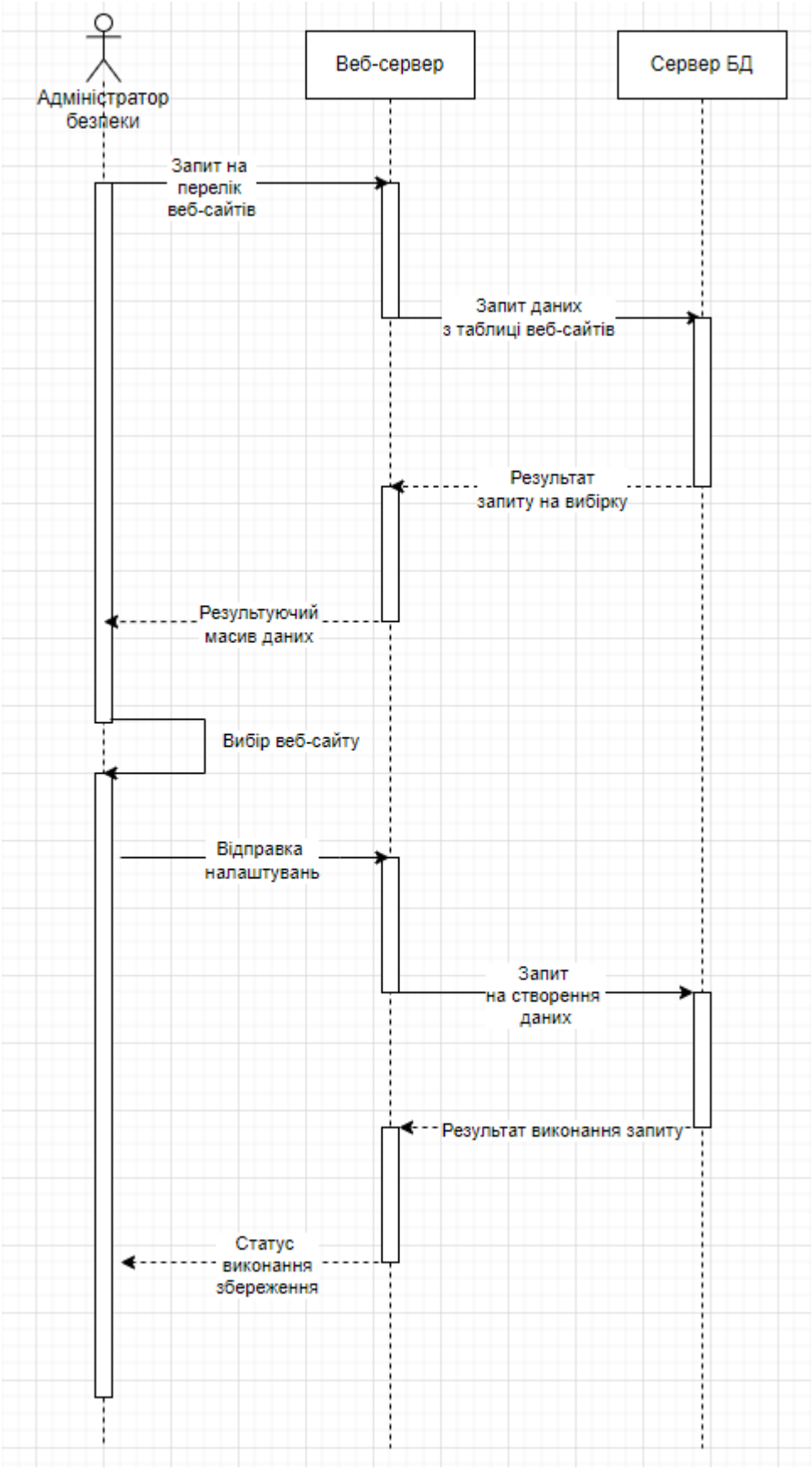


Рисунок 2.3 - Діаграма послідовності процесу налаштування моніторингу

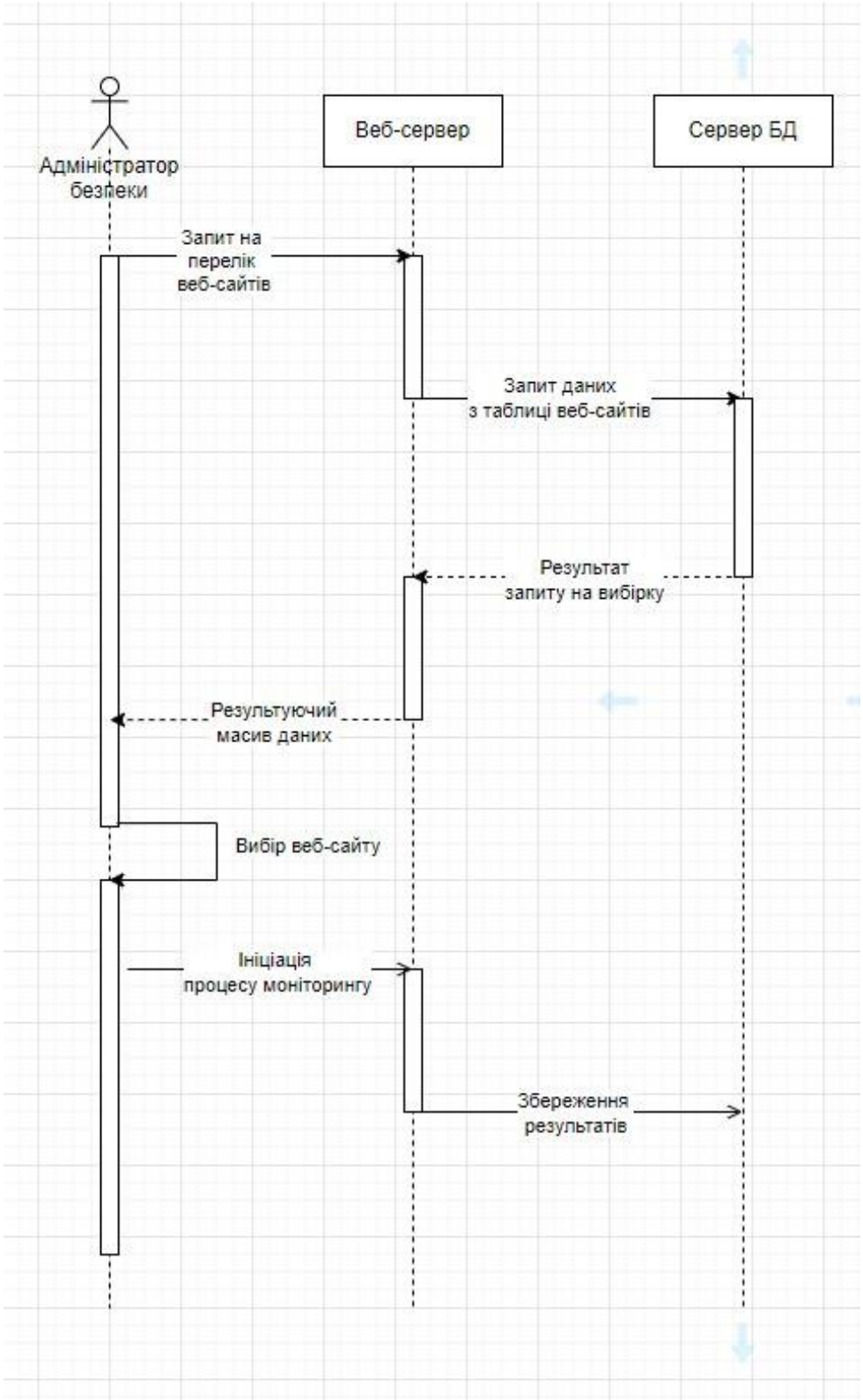


Рисунок 2.4 - Діаграма послідовності ініціації моніторингу

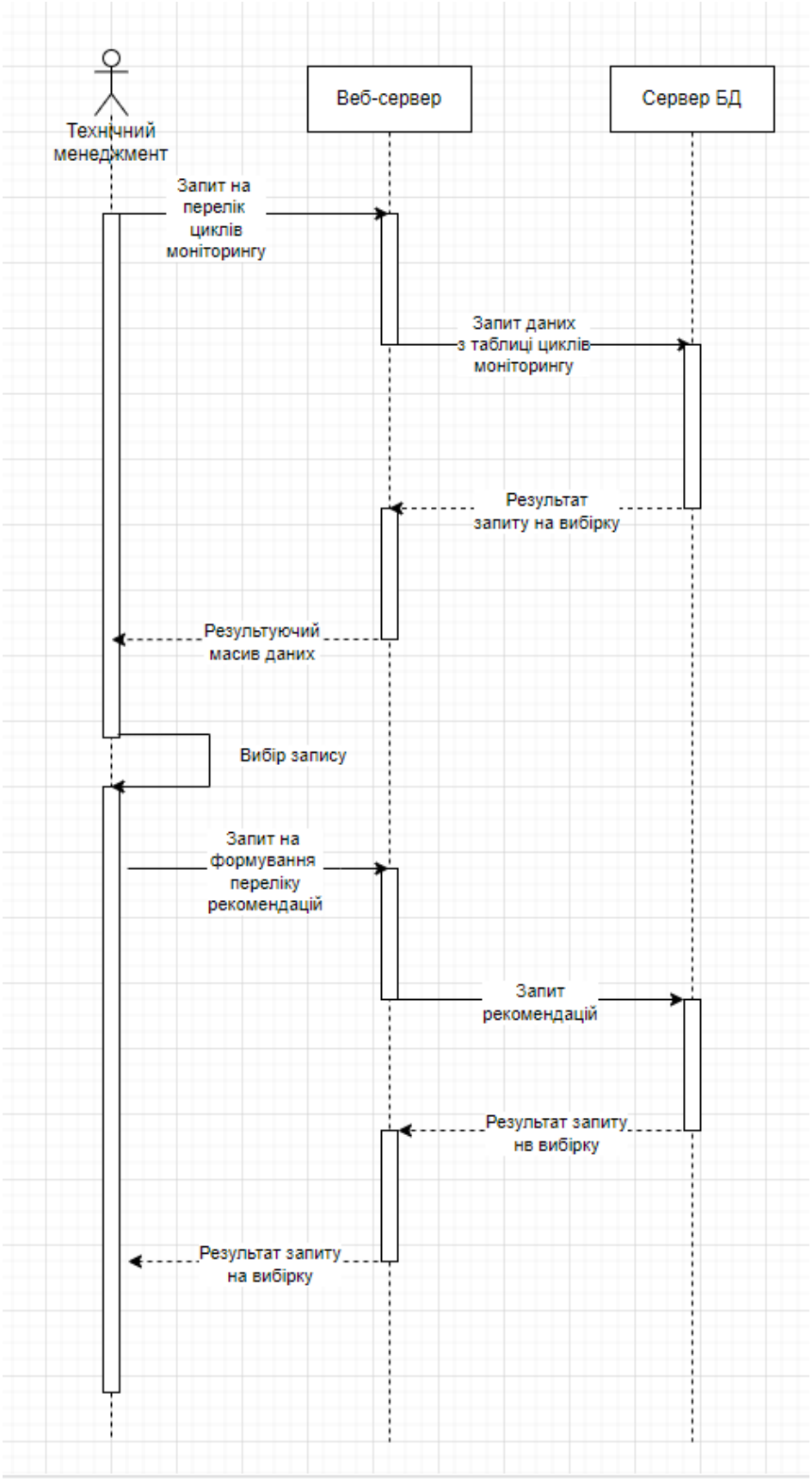


Рисунок 2.5 - Діаграма послідовності процесу отримання результатів моніторингу

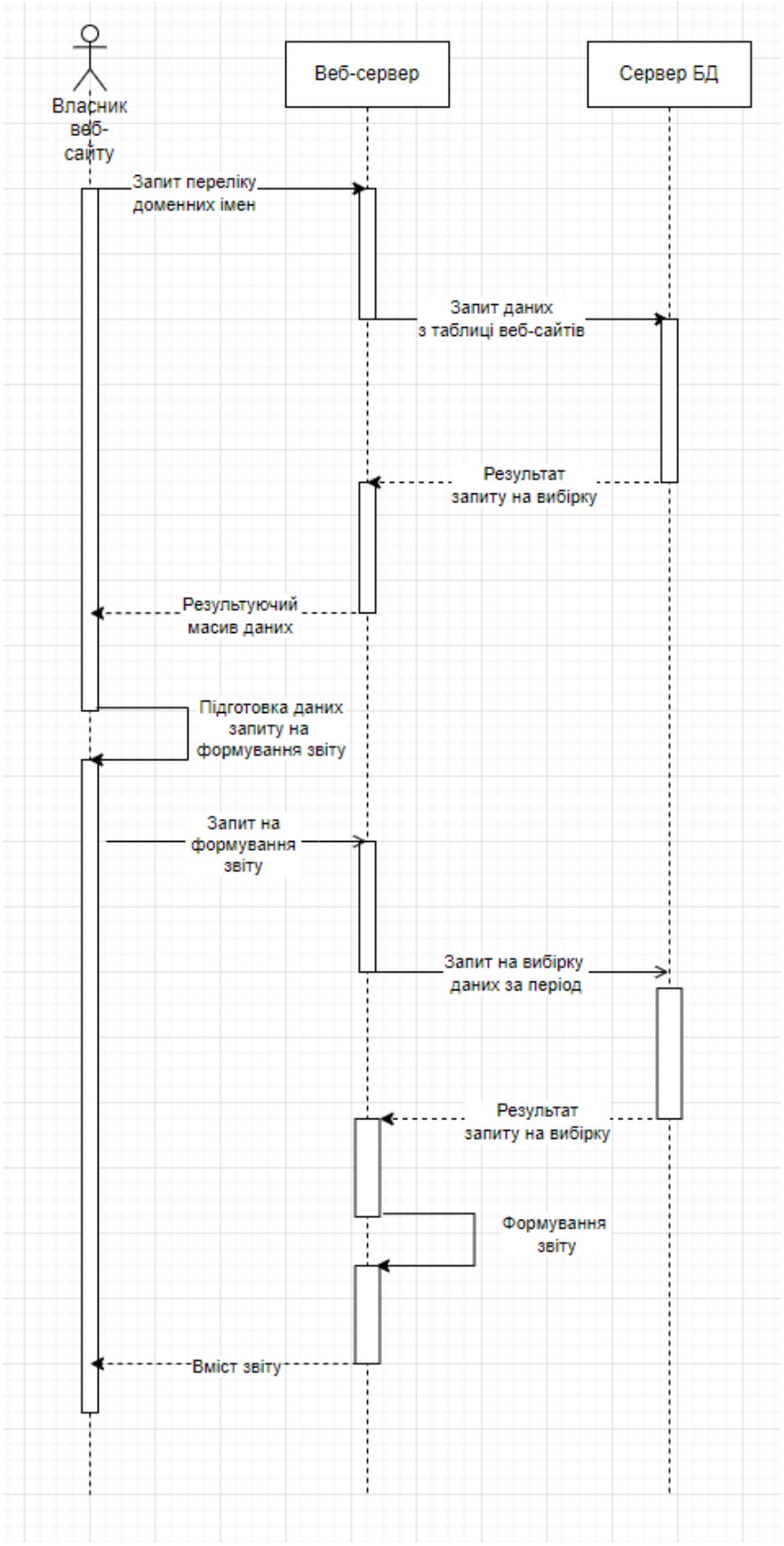


Рисунок 2.6 - Діаграма послідовності процесу формування звітності за період

2.1.3. Моделювання рівня бізнесу

Бізнес-рівень діаграми Archimate – рівень, що визначає послуги для зовнішніх клієнтів, процеси реалізації цих послуг, а також організаційну модель.

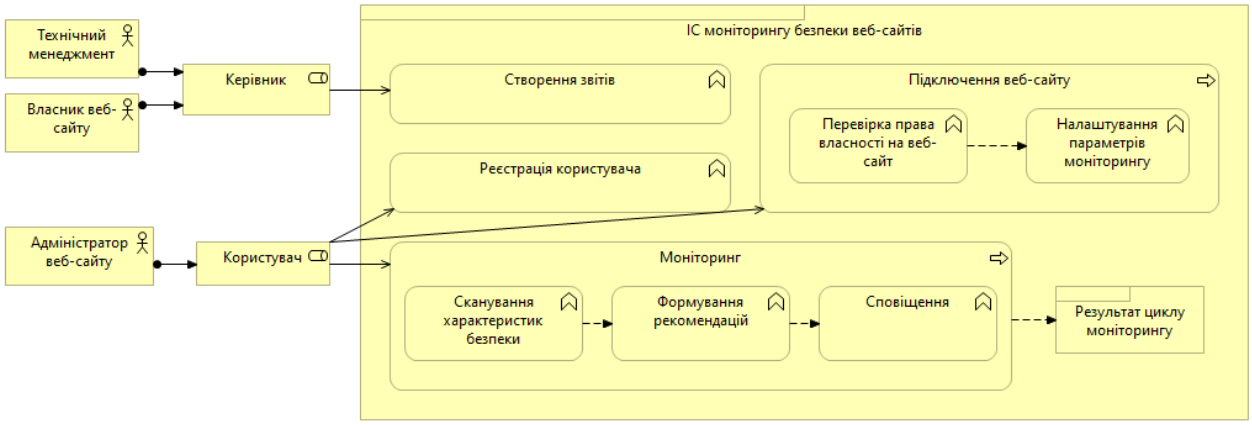


Рисунок 2.7 - Діаграма бізнес-рівня ArchiMate

Складові моделі:

1. Бізнес-виконавець;
    - Технічний менеджмент
    - Власник веб-сайту
    - Адміністратор веб-сайту
2. Бізнес-роль;
    - Керівник
    - Користувач
3. Бізнес-процес;
    - Підключення веб-сайту
    - Моніторинг
4. Бізнес-функція;
    - Створення звітів;
    - Реєстрація користувача
    - Перевірка права власності на веб-сайт

- Налаштування параметрів моніторингу
- Сканування характеристик безпеки
- Формування рекомендацій
- Сповіщення

5. Продукт.
    - ІС моніторингу безпеки веб-сайтів
    - Результат циклу моніторингу

2.1.4. Моделювання рівня додатку

Рівень додатків діаграми Archimate - описує підтримку бізнес рівня IT-додатками та основні види даних.

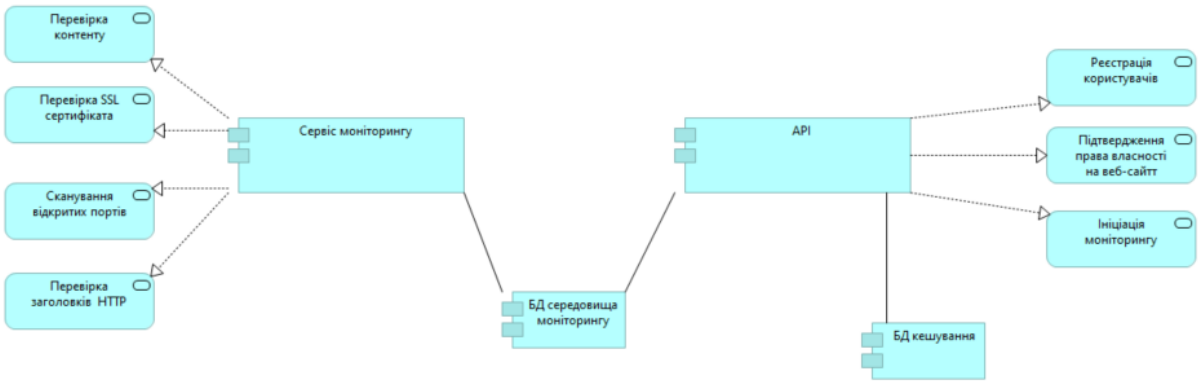


Рисунок. 2.8 - Діаграма рівня додатку ArchiMate

Складові моделі:

1. Програмний компонент;
    - Сервіс моніторингу
    - API
    - БД середовища моніторингу
    - БД кешування
2. Програмний сервіс.
    - Сканування відкритих портів
    - Перевірка заголовків HTTP
    - Перевірка SSL сертифіката

- Перевірка контенту
- Реєстрація користувачів
- Підтвердження права власності на веб-сайт
- Ініціація моніторингу

2.2. Методи та моделі в інформаційно управляючих системах

2.2.1. Перевірка права власності на веб-сайт

ІСПР підтримує декілька способів підтвердження. Перелік способів відображено в таблиці 2.2.

Таблиця 2.2.

Способи підтвердження права власності на веб-сайт

Спосіб	**Характеристика**
Розміщення HTML-файлу на сервері	Цей спосіб відносно простий, але використовувати його можна тільки в тому випадку, якщо користувач має можливість завантажити файл і опублікувати його за певним URL.
HTML-тег	Цей спосіб відносно простий, але необхідно, щоб ви мали можливість редагувати вихідний HTML-код головної сторінки сайту.

Продовження таблиці 2.2.

Провайдер доменних імен	Цей спосіб складніший, проте при роботі з доменними ресурсами підходить тільки він. Перевага доменних ресурсів полягає в тому, що вони містять дані про всі протоколи (HTTP і HTTPS) та варіанти субдоменних імен сайту клієнта.

2.2.2. Сканування відкритих портів

Сканування портів — це метод визначення відкритих портів у мережі, які можуть приймати або надсилати дані. Крім того, це процес відправлення пакетів на певні порти на хості та аналізу відповідей для виявлення вразливостей.

Метою сканування портів та мережі є визначення організації системи IP-адрес, хостів та портів, щоб належним чином визначити місце розташування відкритих або вразливих серверів та рівень безпеки. Сканування як мережі, так і портів виявляє наявність безпеки, наприклад брандмауера між сервером і пристроєм користувача.

Найбільш поширеною утилітою для сканування портів є Nmap.

2.2.2.1. Методи сканування

1. TCP connect() (TCP з’єднання).

Це тип TCP сканування, що використовується за замовчуванням, коли недоступне SYN сканування. Це відбувається у випадку, коли користувач не має привілеїв для використання сирих пакетів або при скануванні IPv6 мереж. Замість використання сирих пакетів, як це відбувається при більшості інших типів сканування, Nmap "просить" операційну систему встановити з'єднання з цільовою машиною по зазначеному порту шляхом системного виклику connect. Це такий самий високорівневий системний виклик, який

використовується браузерами, P2P клієнтами та іншими програмами для встановлення з'єднання. Цей виклик є частиною програмованого інтерфейсу, відомий як Berkeley Sockets API. Замість того, щоб зчитувати відповіді у формі сирих пакетів, Nmap використовує цей API для отримання інформації про статус кожної спроби з'єднання.

2. TCP SYN.

Використовується за замовчуванням і є найбільш популярним типом сканування. На це є кілька причин. Він може бути швидко запущений, він здатний сканувати тисячі портів за секунду при швидкому з'єднанні, його роботі не перешкоджають бранмауери, що обмежують. Цей тип сканування є ненав'язливим і непомітним, адже при такому скануванні TCP з'єднання ніколи не встановлюється до кінця. Він працює з будь-яким TCP стеком, не залежно від особливостей специфічної платформи, як це відбувається при скануваннях типу FIN/NULL/Xmas, Maimon та idle скануванні. Він також надає ясну і достовірну диференціацію між станами відкритий, закритий та фільруеться.

Цю техніку часто називають скануванням з використанням напіввідкритих з'єднань, тому що повне TCP з'єднання не відкривається. Відповіді SYN/ACK вказують на те, що порт прослуховується (відкритий), а RST (скидання) на те, що не прослуховується. Якщо після декількох запитів не надходить жодної відповіді, порт позначається як фільтрований. Порт також позначається як фільтрований, якщо у відповідь надходить ICMP повідомлення про помилку недосяжності (тип 3, 1,2, 3, 9, 10 або 13).

Така техніка сканування передбачає потенційні затримки при обробці пакетів. Наприклад, якщо маршрутизатор в якості політики черги використовує «перший зайшов, перший вийшов», а N- кількість пакетів то затримка буде розраховуватись за формулою:

$$\frac{(N * R)}{H}, \tag{2.1}$$

де N – кількість пакетів;

R – розмір пакету;

H – пропускна здатність маршрутизатора.

3. Stealth FIN (Приховане FIN), Xmas Tree (Різдвяне дерево), Null scan (нульове сканування).

Ці три типи сканування використовують непомітну лазівку TCP RFC , щоб розділяти порти на відкриті та закриті. Коли сканується система, що відповідає вимогам RFC, будь-який пакет, що не містить встановленого біта SYN, RST або ACK, спричинить за собою відправку RST у відповідь у випадку, якщо порт закритий, або не спричинить жодної відповіді, якщо порт відкритий. Оскільки жоден з цих бітів не встановлений, то будь-яка комбінація трьох, що залишилися (FIN, PSH і URG) буде правильною. Nmap використовує це у трьох типах сканування:

- Null сканування (-sN). Не встановлюються ніякі біти (Прапорів у заголовку TCP 0)
- FIN сканування (-sF). Встановлюється лише TCP FIN біт.
- Xmas сканування (-sX). Встановлюються FIN, PSH та URG прапори.

Ці три типи сканування працюють за однією схемою, відмінності тільки в прапорах TCP встановлених у пакетах запитів. Якщо у відповідь приходить RST пакет, порт вважається закрытым, відсутність відповіді означає, що порт відкритий або фільтрується. Порт позначається як той, що фільтрується, якщо у відповідь приходить ICMP помилка про недосяжність (тип 3, код 1, 2, 3, 9, 10 або 13).

Ключовою особливістю цих типів сканування є їх здатність непомітно обійти деякі брандмауери, що не враховують стан (non-stateful) і роутери з функцією пакетної фільтрації. Ще однією перевагою є те, що вони навіть трохи непомітніші, ніж SYN сканування. Все ж таки не треба на це покладатися - більшість сучасних IDS можуть бути налаштовані на їх виявлення. Великим недоліком є те, що не всі системи наслідують RFC 793. Деякі системи надсилають RST відповіді на запити незалежно від того, відкритий порт або закритий. Це призводить до того, що всі порти позначаються як закриті.

Основними системами, що поводяться подібним чином, є Microsoft Windows, багато пристроїв Cisco, BSDI і IBM OS/400. Таке сканування застосовується до більшості систем, що базуються на Unix. Ще одним недоліком цих видів сканування є їхня нездатність розділяти порти на відкриті, та ті, що фільтруються.

4. Ping-сканування.

Найпростіше сканування портів називається ping-скануванням. Перевірка зв'язку (ping) використовується в мережі, щоб з'ясувати, чи можливе надсилання мережних пакетів даних на IP-адресу без помилок. Ping-сканування — це запити протоколу міжмережевих повідомлень (ICMP) з автоматичним надсиланням декількох запитів ICMP на різні сервери, щоб спровокувати відповідь. IT-адміністратори можуть використовувати цей метод для усунення несправностей або відключити відповідь на ping-запити за допомогою брандмауера, що не дозволить зловмисникам знайти мережу за допомогою перевірки зв'язку.

5. UDP-сканування.

У той час як більшість інтернет-сервісів використовують TCP протокол, UDP служби також широко поширені. Найбільш популярними є DNS, SNMP та DHCP (використовують порти 53, 161/162 та 67/68). Оскільки UDP сканування в загальному випадку повільніше і складніше TCP, багато фахівців з безпеки ігнорують ці порти. Це помилка, адже. є UDP служби, які використовуються для атак. На щастя, Nmap дозволяє інвентаризувати порти UDP.

UDP сканування запускається опцією -sU. Воно може бути скомбіноване з будь-яким типом сканування TCP, наприклад SYN сканування (-sS), щоб використовувати обидва протоколи за один прохід.

UDP сканування працює шляхом надсилання порожнього (без даних) UDP заголовка на кожен цільовий порт. Якщо відповідь приходить ICMP помилка про недосяжність порту (тип 3, код 3), значить порт закритий. Інші ICMP помилки недосяжності (тип 3, коди 1, 2, 9, 10 або 13) вказують на те, що порт фільтрується. Іноді служба буде відповідати UDP пакетом, вказуючи на

те, що порт є відкритим. Якщо після кількох спроб не було отримано відповіді, то порт класифікується як відкритий|фільтрується. Це означає, що порт може бути відкритий або, можливо, пакетний фільтр блокує його. Функція визначення версії (-sV) може бути корисною для диференціації дійсно відкритих портів і фільтрованих.

Великою проблемою при скануванні UDP є його повільна швидкість роботи. Відкриті та фільтровані порти рідко посилають будь-які відповіді, змушуючи Nmap відправляти повторні запити, якщо пакети були втрачені. Закриті порти часто виявляються ще більшою проблемою. Зазвичай вони у відповідь повертають помилку ICMP про недосяжність порту. Але на відміну від RST пакетів, що відсилаються закритими TCP портами у відповідь на SYN або сканування з установкою з'єднання, багато хостів обмежують ліміт ICMP повідомлень про недосяжність порту за замовчуванням.

Nmap виявляє такі обмеження і відповідно скорочує кількість запитів, щоб не забивати мережу марними пакетами, які все одно будуть відкинуті цільовою машиною. На жаль, при обмеженні Linux (один пакет за секунду) сканування 65,536 портів займе понад 18 годин. До способів збільшення швидкості UDP сканування відносяться:

- паралельне сканування декількох хостів;
- сканування в першу чергу тільки найбільш популярних портів;
- сканування через брандмауер і використання --host-timeout для пропуску повільних хостів.

6. Сканування IP-протоколів.

Сканування такого типу дозволяє визначити, які протоколи IP (TCP, ICMP, IGMP тощо) підтримуються цільовими машинами. Технічно таке сканування перестає бути різновидом сканування портів, оскільки воно циклічно перебирає номери IP протоколів замість номерів TCP чи UDP портів. Хоча тут все ж таки використовується опція -p для вибору номерів протоколів для сканування, результати видаються у форматі таблиці портів, і навіть використовується той же механізм сканування, що і при різних

варіантах сканування портів. Тому він досить близький до сканування портів та описується тут.

Спосіб роботи цього типу сканування дуже схожий на реалізований в скануванні UDP. Замість того, щоб змінювати в UDP пакеті поле, що містить номер порту, надсилаються заголовки IP пакета, і змінюється 8-бітове поле IP протоколу. Винятками є TCP, UDP та ICMP. Включення правильного заголовка цих протоколів необхідно, адже деякі системи не будуть їх відсилати, та й у Nmap є всі необхідні функції для їх створення. Замість очікування ICMP повідомлення про недосяжність порту, цей тип сканування очікує ICMP повідомлення про недосяжність протоколу . Якщо Nmap отримує будь-яку відповідь за будь-яким протоколом, то протокол позначається як відкритий. ICMP помилка про недосяжність протоколу (тип 3, код 2) позначає протокол як закритий. Інші помилки ICMP недосяжності (тип 3, код 1, 3, 9, 10 або 13) позначають протокол як той, що фільтрується(у той же час вони показують, що протокол ICMP открыт). Якщо не надходить жодної відповіді після кількох запитів, то протокол позначається як відкритий або фільтрується.

7. Idlescan.

Цей просунутий метод сканування дозволяє здійснити непомітне TCP сканування портів мети (мається на увазі, що ніякі пакети не відсилаються на цільову машину з реальної IP адерси). Замість цього, на «зовнішній» машині Є використовується передбачувана послідовність генерації ID IP фрагментів для збору інформації про відкриті порти мети[20]. Системи IDS будуть вважати, що сканування проводиться із заданої «зовнішньої» машини (яка повинна працювати та задовольняти певним критеріям).

Крім його непомітності, цей тип сканування також дозволяє визначати засновані на IP «довірчі» відносини між машинами. Список відкритих портів показує відкриті порти з точки зору машини. Тому можна спробувати просканувати ціль, використовуючи різні зомбі машини, яким, можливо, будуть «довіряти» (за допомогою правил роутера/пакетного фільтра).

8. ACK-сканування.

Цей тип сканування дуже відрізняється від інших тим, що він не здатний визначити відкритий порт (або навіть відкритий або фільтрований). Він використовуються для виявлення правил брандмауерів, визначення враховують вони стан чи ні, а також для визначення портів, що фільтруються ними.

Пакет запиту при такому типі сканування містить встановлений тільки ACK прапор (якщо не використовується --scanflags). При скануванні нефільтрованих систем, відкриті або закриті порти повертатимуть у відповідь пакет RST. Nmap позначає їх як нефільтровані, маючи на увазі, що вони доступні для ACK пакетів, але невідомо відкриті вони чи закриті. Порти, які не відповідають або надсилають у відповідь ICMP повідомлення про помилку (тип 3, код 1, 2, 3, 9, 10 або 13), позначаються як фільтровані.

9. Віконне сканування.

Цей тип сканування практично те ж саме, що і ACK сканування, за винятком того, що він використовує особливості реалізації різних систем для поділу портів на відкриті та закриті, замість того, щоб завжди при отриманні пакета RST виводити нефільтрований. Це здійснюється шляхом аналізу TCP Window поля отриманого у відповідь пакета RST. У деяких системах відкриті порти використовують позитивне значення цього поля (навіть у пакетах RST), а закриті - нульове. Тому замість того, що весь час при отриманні RST пакета у відповідь помічати порти як нефільтровані, при Window скануванні порти позначаються як відкриті або закриті, якщо значення поля TCP Window позитивне або дорівнює нулю відповідно.

Цей тип сканування ґрунтується на особливостях реалізації меншості систем в Інтернеті, тому не варто на нього спиратись. Загалом у системах, які мають такі особливості, всі порти позначатимуться як закриті. Звичайно, можливо, що машина дійсно не має відкритих портів. Якщо більшість просканованих портів закриті, і лише кілька поширених портів (таких як 22, 25, 53) фільтруются, то, швидше за все, результатам сканування можна довіряти. Іноді системи будуть поводитися прямо протилежним чином. Якщо

в результаті сканування буде знайдено 1000 відкритих портів і 3 закритих або фільтрованих, то ці 3 можуть виявитися дійсно відкритими.

10.RPC-сканування.

Цей метод використовується спільно з іншими методами сканування та дозволяє визначити програму, яка обслуговує RPC-порт, та номер її версії. Для цього на всі відкриті TCP/UDP-порти хоста відправляються запити з NULL-командами оболонки SunRPC, після чого визначаються RPC-порти та закріплені за ними програми. Таким чином, формується інформація, яку можна отримати за допомогою команди rpcinfo -p , навіть якщо portmapper сканованого хоста закритий брандмауером.

11. FTP bounce.

Цікавою можливістю FTP протоколу є підтримка проксі FTP з'єднань. Це дозволяє користувачеві підключитися до одного сервера FTP, а потім попросити його передати файли іншому. Це є грубим порушенням, тому багато серверів припинили підтримувати цю функцію. Використовуючи цю функцію, можна здійснити за допомогою цього FTP сервера сканування портів інших хостів. Необхідно надіслати FTP серверу команду переслати файл на кожен порт цільової машини по черзі. Повідомлення про помилку вкаже: відкритий порт чи ні. Це добрий спосіб обходу брандмауерів, адже FTP сервери зазвичай мають більше доступу внутрішніх хостів, ніж будь-які інші машини.

2.2.3. Перевірка SSL-сертифіката

Перевірка автентичності SSL-сертифікату здійснюється для кожної IP-адреси, в разі його наявності. Аналізується не лише його взаємозв’язок з доменом цільової системи, а й повний шлях сертифікації.

Алгоритм включає в себе перелік наступних функцій:

1. Перевірка відповідності домену;
2. Перевірка терміну життя сертифіката;
3. Перевірка шляху сертифікації;
4. Перевірка OCSP.

Рисунок 2.9 - Алгоритм перевірки SSL-сертифіката

2.2.4. Перевірка заголовків безпеки HTTP-пакетів

Частина серверних заголовків HTTP призначена для підвищення безпеки сесій під час роботи браузера з веб-сайтами. Ці заголовки називаються "заголовками безпеки". Історично, вони вводилися у вживання поступово, як відповідь на нові напрями атак. Заголовки безпеки дозволяють адміністратору веб-ресурсу передати браузеру відомості, які описують рекомендовані адміністратором правила та політики роботи клієнта (браузера) із веб-

сайтом. Підтримка та інтерпретація заголовків різняться в деталях від браузера до браузеру (а також у різних версіях браузерів однієї лінійки), але базові принципи у всіх сучасних браузерах загальні.

1. X-Frame-Options;

Цей заголовок визначає те, як браузер повинен показувати відповідну сторінку в контексті відображення, що вбудовується. Наприклад, у HTML існує поняття "фрейму" (теги <frame> та <iframe>). "Фрейм" дозволяє прозоро вбудувати веб-сторінку, яка відповідає одному серверу, в певну область видимості всередині сторінки, що відноситься до іншого сервера. Так показується реклама на веб-сайтах: на основній сторінці виділяється візуальний блок, - зазвичай, фіксованого розміру, - і всередині цього блоку браузер показує контент, отриманий з іншого сайту (і сервера). Однак, за допомогою тієї ж конструкції з "фреймами", контент деякого веб-сайту може бути вбудований у сторінку іншого, всупереч бажанню власника першого сайту. Це не просто дозволяє зловмисникам видати чужий контент за свій, прозоро підмінивши видиму адресу сторінки, але й ввести користувача в оману щодо того, на якому сайті насправді виконуються ті чи інші дії – наприклад, користувач натискає на якесь зображення. Заголовок X-Frame-Options, якщо він підтримується браузером, протидіє всім цим неприємностям.

2. X-XSS-Protection;

Цей заголовок управляє включенням фільтра "крос-сайтового скриптингу" на стороні браузера (заголовок вважається застарілим, але все ще рекомендований до використання, оскільки дозволяє захистити застарілі браузери). "Крос-сайтовий скриптинг" (cross-site scripting - XSS) - це поширений інструмент для побудови атак, що базуються на впровадженні додаткового коду та нав'язуванні спеціально згенерованих HTTP-запитів у браузерні сесії, що стосуються інших веб-сайтів, а не сайту -Джерело. Базова логіка атак XSS у тому, що нав'язаний скрипт чи запит виконується у іншій сесії. Так, користувач відкриває в браузері веб-сторінку деякого сайту P, яка містить спеціально підготовлений скрипт, що змушує браузер зробити, в контексті збереженої сесії, запит на сайт Q; оскільки збережена сесія може

містити активну авторизацію для сайту Q - запит буде виконано з відповідними правами на сайті Q, якби сам користувач відкрив цей сайт у браузері і, наприклад, відправив якесь повідомлення. Заголовок X-XSS-Protection підтримується не всіма сучасними браузерами.

3. Content-Security-Policy (CSP);

Найнасиченіший директивами та опціями заголовок безпеки. Він дозволяє адміністратору сервера задати політики, що визначають завантаження браузером різних ресурсів та об'єктів, що належать до цієї адреси (або сторінки). Наприклад, політики, що задаються Content-Security-Policy, визначають як і з яких серверів браузер може завантажувати скрипти, файли зображень, додаткові шрифти, плагіни та інші елементи сторінки. Але CSP не обмежується цим – опцій та директив дуже багато. Вони поділяються на п'ять наборів: завантажувальні директиви, документальні директиви, навігаційні директиви, директиви, що інформують, та інші директиви.

4. Referrer-Policy;

Відноситься до іншого типу заголовків безпеки: цей заголовок визначає, які дані браузер може передати в заголовку HTTP Referer, який використовується, наприклад, при обробці браузером "переходу за посиланням". Так, Referer може містити адресу попереднього ресурсу, з якого користувач перейшов на цю. Тобто сервер, який обслуговує цільовий ресурс, бачить, звідки прийшов користувач. Щоб зменшити ймовірність витоку інформації, що відбувається через цей заголовок, перелік даних, які можуть передаватися, регулюється заголовком безпеки Referrer-Policy. Наприклад, "Referrer-Policy: origin" - визначає, що у полі Referer браузер передає лише частину повної адреси (URL), що визначає протокол та ім'я сервера: під час переходу зі сторінки https://test.ru/from.html у Referer браузер запише лише https: // Test.ru/. (У браузерах є інші механізми, що обмежують Referer.)

5. X-Content-Type-Options ;

Наказує браузеру суворо дотримуватися переліку ідентифікаторів типів контенту, вказаних у полі заголовка Content-Type. Передбачається, що тип даних (або контенту), що передається сервером у відповіді HTTP, коректно

позначений в HTTP-заголовку. Однак у спірних випадках браузери можуть використовувати власні гнучкі алгоритми визначення типу контенту на підставі даних. Це допомагає, наприклад, коректно обробити файли зображень, які помилково передаються з невідповідним типом. Заголовок X-Content-Type-Options дозволяє заборонити браузеру використання цих алгоритмів. Заголовок може блокувати атаки з заміною даних, при яких браузер через помилкову інтерпретацію типу виконує шкідливий код.

6. Strict-Transport-Security;

Він повідомляє браузеру, що при з'єднанні з даним веб-ресурсом слід використовувати лише безпечний протокол HTTPS. Цей заголовок протидіє атакам, які використовують зниження рівня захисту, саме заміну зазначеного на адресу ресурсу протоколу з https:// на http://. Таку заміну можна зробити різними способами, у тому числі прямим втручанням у незахищений HTTP-трафік. При цьому користувач може не помітити різниці в типі з'єднання, а відсутність TLS дозволяє здійснити і заміну адреси, зберігши знайоме доменне ім'я в адресному рядку браузера.

7. Public-Key-Pins ;

Це механізм, що дозволяє передати клієнту відбитки серверних криптографічних ключів TLS, які є допустимими для цього сервера. Це відомий за іншими безпечними протоколами (наприклад, SSH) метод запам'ятовування ключів при першій зустрічі. Метод дозволяє в подальшому виявити заміну, навіть якщо така заміна проводиться за участю довіреного центру. Цей заголовок раніше повністю підтримувався поширеними браузерами, зокрема Mozilla Firefox, але, на жаль, зараз практично виведений з підтримки (в актуальних версіях Firefox потрібно включати підтримку вручну, а коректна робота не гарантується).

2.2.5. Перевірка контенту

Алгоритм аналізу метаданих передбачає обробку документів та зображень з розширенням .pdf, .docx, .xls, xslx, .png, .jpg. Такі документи

найчастіше публікуються на рекламних сторінках компаній, сторінках партнерів, загальнодоступних файлових серверах, тощо(рис. 2.10).

Рисунок 2.10 - Збір та виявлення характеристик метаданих

Визначення автора документа здійснюється в контексті виявлення існування облікового запису в цільовій системі. На рисунку 2.3 представлено процес валідації e-mail адреси.

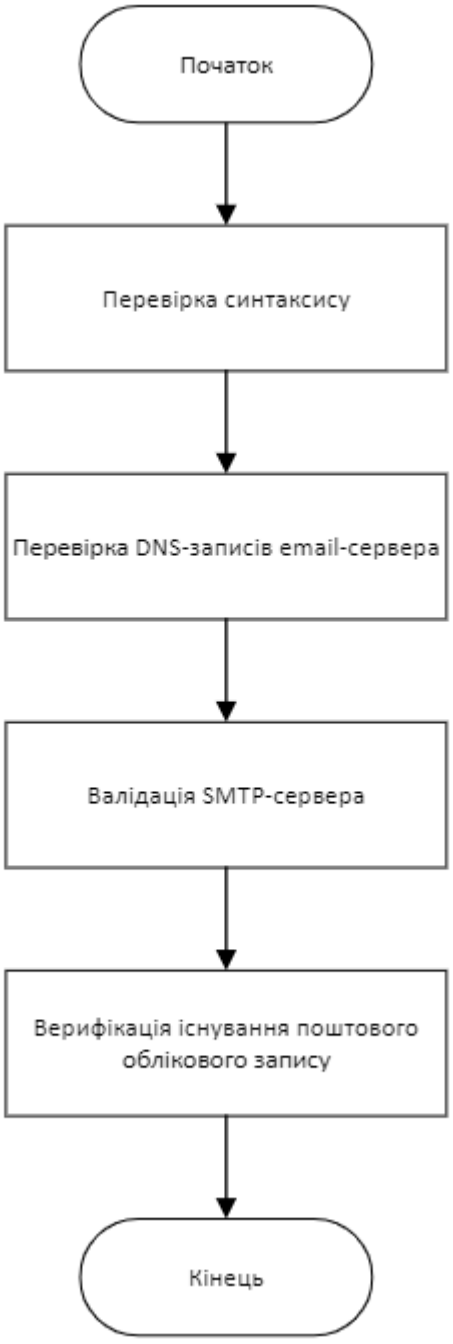


Рисунок 2.11 - Перевірка автентичності e-mail адреси

3. КОНСТУКТИВНИЙ РОЗДІЛ. РОЗРОБКА ПРОЄКТНИХ РІШЕНЬ

3.1. Проєктування бази даних для інформаційної системи

3.1.1. Інформаційне забезпечення та основні джерела даних

Представлена ІС включає в себе набір наступних джерел вхідної інформації:

1. Конфігурація фізичного сервера(серверів), мережі;

Сканування хосту(хостів) на предмет наявності відкритих портів, визначення сервісів, їх версій.

2. Конфігурація протоколів прикладного рівня;
3. Відкриті бази даних вразливостей;

Відкриті бази даних вразливостей є ще одним джерелом даних, які були визнані корисними і також використовувалися у відповідних дослідженнях. До даного переліку входять NIST (NVD), ExploitDB і vulnDB. Ці бази даних зазвичай складаються з спільно накопичених звітів про уразливість, а також деяких додаткових дескрпиторів, таких як цільова платформа, тип експлойта, CVE і бал CVSS. На додаток до них зазвичай також додається код або console/io, що відтворює вразливість.

4. Дані з відкритих джерел;

Документи, файли, конфігураційні файли, які можуть включати в себе інформацію про мережеві налаштування, використовуване програмне та технічне забезпечення тощо.

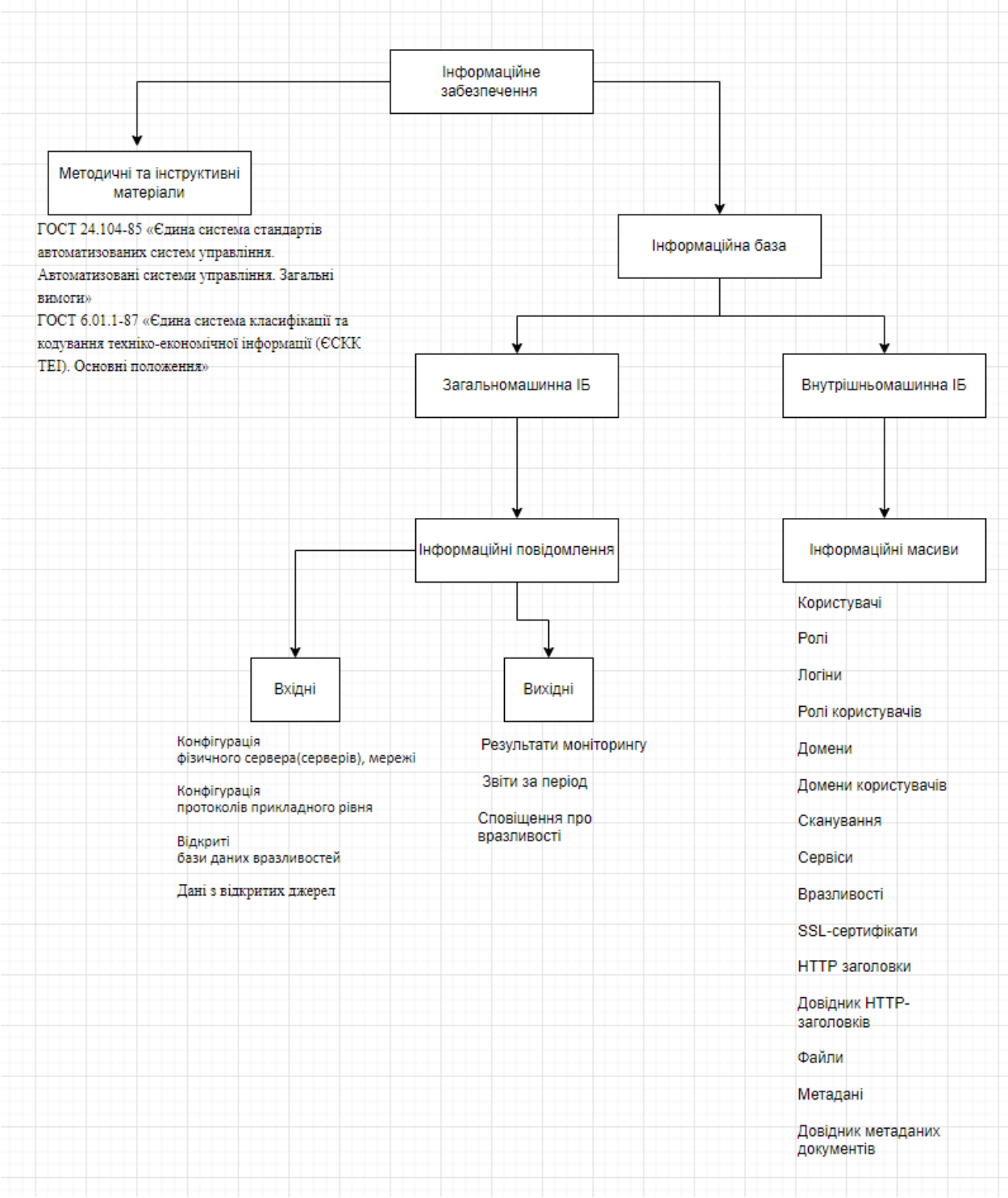


Рисунок 3.1 - Структура інформаційного забезпечення

3.1.2. Системний аналіз та словесний опис інформаційних об'єктів

Загалом інформацію, що зберігається в БД розроблюваної інформаційної системи, можна віднести до трьох основних доменів:

1. Користувачі;
2. Веб-сайти;

3. Моніторинг.

До домену «Користувачі» відносяться такі об’єкти:

1. Користувач;

Описуються основні характеристики та ідентифікатори в системі, а також показники зміни стану даного об’єкта відносно життєвого циклу системи.

2. Роль;

Описуються характеристики ролей в системі.

3. Логін.

Описується історія автентифікації користувачів в системі.

4. Ролі користувачів;

Описується взаємозв’язок між ролями та користувачами.

До домену «Веб-сайти» відносяться такі об’єкти:

1. Домен;

Описуються характеристики підключених веб-сайтів, налаштовані способи підтвердження права власності.

2. Домени користувачів;

Описується взаємозв’язок між підключеними веб-сайтами та користувачами.

До домену «Моніторинг» відносяться такі об’єкти:

1. Сканування.

Описується зв’язок одиниці моніторингу з його фактичними результатами.

2. Сервіс;

Описує характеристики служб та встановлено ПЗ цільового веб-ресурсу, що було ідентифіковано внаслідок сканування.

3. Вразливість;

Описуються вразливості ідентифікованих сервісів, відповідно до даних, що отримуються з бази даних CVE(ідентифікатор, адреса).

4. SSL-сертифікат;

Описуються основні характеристики SSL сертифікатів підключених веб-сайтів у розрізі сканувань.

5. HTTP заголовок;

Описується інформація щодо заголовків, що були отримані, їх властивостей, а також результат перевірки значень.

6. Довідник HTTP заголовків;

Описується інформація щодо заголовків, що мають бути присутні в тілі запиту, а також їх необхідні значення.

7. Файл;

Описуються файли та документи, що відносяться до цільового веб-сайту та були знайдені серед відкритих джерел.

8. Метадані;

Описуються метадані, що було отримано з файлів та документів, знайдених серед відкритих джерел.

9. Довідник метаданих.

Описуються ключі метаданих та характеристика інформації, котра зберігається в якості значень

3.1.3. Логічне проєктування БД

Схему БД розроблюваної системи представлено наступними відношеннями:

Таблиця 3.1

Відношення сутності «Користувачі» AspNetUsers

Ім’я стовпця	Тип	Довжина	Призначення
Id	Рядок	36	Унікальний ідентифікатор(GUID), первинний ключ
UserName	Рядок	256	Ім’я користувача
NormalizedUserName	Рядок	256	Нормалізоване ім’я користувача
Email	Рядок	256	E-mail
NormalizedEmail	Рядок	256	Нормалізований E-mail

Продовження таблиці 3.1

EmailConfirmed	Булевий	1	E-mail підтверджено, обов’язкове
PasswordHash	Рядок	-	Хеш паролю
SecurityStamp	Рядок	-	Випадковий ідентифікатор(GUID), ідентифікуючий зміну даних користувача,
ConcurrencyStamp	Рядок	-	Випадковий ідентифікатор(GUID), ідентифікуючий ідемпотентну операцію
PhoneNumber	Рядок	-	Номер телефону
PhoneNumberConfirmed	булевий	1	Номер телефону підтверджено
TwoFactorEnabled	Булевий	1	Активовано двохфакторну автентифікацію, обов’язкове
LockoutEnd	Дата і час	-	Дата та час завершення блокування користувача
LockoutEnabled	Булевий	1	Активовано блокування користувача, обов’язкове

Таблиця 3.2

Відношення сутності «Ролі» AspNetRoles

Ім’я стовпця	Тип	Довжина	Призначення
Id	Рядок	36	Унікальний ідентифікатор(GUID), первинний ключ
Name	Рядок	256	Найменування ролі
NormalizedName	Рядок	256	Нормалізоване найменування ролі
ConcurrencyStamp	Рядок	-	Випадковий ідентифікатор(GUID), ідентифікуючий ідемпотентну операцію

Таблиця 3.3

Відношення сутності «Ідентифікатори безпеки ролей» AspNetRoleClaims

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Roleid	Рядок	-	Зовнішній ключ, що посилається на первинний ключ відношення AspNetRoles, обов'язкове
ClaimType	Рядок	-	Тип ідентифікатора безпеки
ClaimValue	Рядок	-	Значення ідентифікатора безпеки

Таблиця 3.4

Відношення сутності «Ролі користувачів» AspNetUserRoles

Ім’я стовпця	Тип	Довжина	Призначення
UserId	Рядок	36	Зовнішній ключ, що посилається на первинний ключ відношення AspNetUsers, обов'язкове
RoleId	Рядок	36	Зовнішній ключ, що посилається на первинний ключ відношення AspNetRoles, обов'язкове

Таблиця 3.5

Відношення сутності «Токени користувачів» AspNetUserTokens

Ім’я стовпця	Тип	Довжина	Призначення
UserId	Рядок	36	Зовнішній ключ, що посилається на первинний ключ відношення AspNetUsers, обов'язкове
LoginProvider	Рядок	-	Провайдер авторизації Первинний ключ, обов'язкове
Name	Рядок	-	Первинний ключ, обов'язкове
Value	Рядок	-	Значення

Таблиця 3.6

Відношення сутності «Логіни користувачів» AspNetUserLogins

Ім’я стовпця	Тип	Довжина	Призначення
LoginProvider	Рядок	-	Провайдер авторизації Первинний ключ, обов'язкове
ProviderKey	Рядок	-	Ідентифікатор провайдера авторизації Первинний ключ, обов'язкове
ProviderDisplayName	Рядок	-	Найменування провайдера
UserId	Рядок	36	Зовнішній ключ, що посилається на первинний ключ відношення AspNetUsers, обов'язкове

Таблиця 3.7

Відношення сутності «Ідентифікатори безпеки користувачів» AspNetUserClaims

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	-	Унікальний ідентифікатор, первинний ключ
UserId	Рядок	-	Зовнішній ключ, що посилається на первинний ключ відношення AspNetUsers, обов'язкове
ClaimType	Рядок	-	Тип ідентифікатора безпеки
ClaimValue	Рядок	-	Значення ідентифікатора безпеки

Таблиця 3.8

Відношення сутності «Веб-сайти» Domains

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Url	Рядок	255	Url-адреса, обов'язкове
VerificationType	Ціле число	10	Тип підтвердження права власності на веб-сайт. Можливі значення: metatag = 0, domainInfo = 1
LastVerification	Дата і час(UTC)	-	Дата та час останньої перевірки права власності

Таблиця 3.9

Відношення сутності «Веб-сайти користувачів» AspNetUserDomains

Ім’я стовпця	Тип	Довжина	Призначення
UserId	Рядок	36	Зовнішній ключ, що посилається на первинний ключ відношення AspNetUsers, обов'язкове
DomainId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Domains, обов'язкове

Таблиця 3.10

Відношення сутності «Сканування» Scans

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
TimeStamp	Дата і час(UTC)	-	Мітка часу сканування, обов'язкове
DomainId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Domains, обов'язкове

Таблиця 3.11

Відношення сутності «Дані SSL сертифікату» SslInfos

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
ExpirationDate	Дата і час(UTC)	-	Дата завершення дії сертифіката, обов'язкове
SslPolicyError	Ціле число	-	Ідентифікатор помилки сертифіката, обов'язкове. Можливі значення: None = 0x0, RemoteCertificateNotAvailable = 0x1, RemoteCertificateNameMismatch = 0x2, RemoteCertificateChainErrors = 0x4
ScanId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Scans, обов'язкове

Таблиця 3.12

Відношення сутності «Сервіси» Services

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Port	Ціле число	-	Номер порту, обов'язкове
Name	Рядок	500	Найменування сервісу, обов'язкове
Version	Рядок	100	Версія, обов'язкове
State	Ціле число	10	Стан сервісу, обов'язкове. Можливі значення: Undefined = 0, Open = 1, Closed = 2, Filtered = 3, Unfiltered = 4, OpenFiltered = 5, ClosedFiltered = 6
ScanId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Scans, обов'язкове

Таблиця 3.13

Відношення сутності «Вразливості» Vulnerabilities

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Code	Рядок	50	Ідентифікатор вразливості, обов'язкове
Url	Рядок	500	Посилання, обов'язкове

Таблиця 3.14

Відношення сутності «HTTP заголовки» HttpHeaders

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
ParentId	Ціле число	10	Ідентифікатор батьківського заголовка, обов'язкове
Url	Рядок	500	Посилання, обов'язкове
Name	Рядок	200	Найменування заголовка, обов'язкове
Value	Рядок	200	Значення заголовка, обов'язкове
State	Ціле число	10	Стан значення заголовка, обов'язкове. Можливі значення: Ok = 0, Missed = 1, ValidationFailed = 2
ScanId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Scans, обов'язкове

Таблиця 3.15

Відношення сутності «Документи» DorkFiles

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
FileName	Рядок	255	Назва файла, обов'язкове
FileUrl	Рядок	-	Посилання на файл, обов'язкове
ScanId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення Scans, обов'язкове

Таблиця 3.16

Відношення сутності «Метадані» Metadata

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Name	Рядок	128	Найменування, обов'язкове
Value	Рядок	500	Значення, обов'язкове
Name	Рядок	200	Найменування, обов'язкове
State	Ціле число	10	Стан значення метаданих, обов'язкове. Можливі значення: Ok = 0, Missed = 1, ValidationFailed = 2

Таблиця 3.17

Відношення сутності «Довідник HTTP заголовків» HbHttpHeaders

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
ParentId	Ціле число	10	Ідентифікатор батьківського заголовка, обов'язкове
Name	Рядок	200	Найменування заголовка, обов'язкове

Таблиця 3.18

Відношення сутності «Локалізація довідника HTTP заголовків» HbHttpHeaderLocs

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Locale	Рядок	5	Ідентифікатор локалі, обов'язкове
Description	Рядок	-	Опис заголовка, обов'язкове
HbHttpHeaderId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення HbHttpHeader, обов'язкове

Таблиця 3.19

Відношення сутності «Довідник метаданих» HbFileMetadata

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Name	Рядок	200	Найменування значення метаданих, обов'язкове

Таблиця 3.20

Відношення сутності «Локалізація довідника метаданих» HbFileMetadataLocs

Ім’я стовпця	Тип	Довжина	Призначення
Id	Ціле число	10	Унікальний ідентифікатор, первинний ключ
Locale	Рядок	5	Ідентифікатор локалі, обов'язкове
Description	Рядок	-	Опис значення метаданих, обов'язкове
HbFileMetadataId	Ціле число	10	Зовнішній ключ, що посилається на первинний ключ відношення HbFileMetadata, обов'язкове

Інфологічну модель представлено в додатку 1.

3.1.4. Обгрунтування вибору СКБД та фізична модель

Для розроблюваної інформаційної системи було вибрано Postgresql. PostgreSQL це потужна об'єктно-реляційна система управління базами даних із відкритим вихідним кодом. Вона запускається на всіх основних платформах, включаючи Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS

X, Solaris, Tru64) та Windows. Вона повністю відповідає ACID, має повну підтримку ключів, об'єднань, тригерів, та збережених процедур (різними мовами). Вона включає більшість типів даних SQL92 і SQL99, включаючи integer, numeric, boolean, char, varchar, date, interval, та timestamp. Вона також підтримує зберігання великих двійкових об'єктів (BLOB's), включаючи картинки, звук або відео. Вона має API для C/C++, Java, Perl, Python, Ruby, Tcl, ODBC та ін.

Будучи СКБД класу підприємства, PostgreSQL надає такі особливості як Multi-Version Concurrency Control (MVCC), відновлення по точці в часі, табличний простір, асинхронна реплікація, вкладені транзакції (точки збереження), гаряче резервування, планувальник/оптимізатор запитів, та попередження відмов. Він підтримує міжнародні кодування, у тому числі багатобайтові, при використанні різних кодувань можна використовувати сортування і повнотекстовий пошук, розрізняти регістр. Доступна велика кількість підконтрольних даних і велика кількість користувачів, що одночасно працюють, проте не сильно впливає на масштабованість системи. Є діючі PostgreSQL системи, які управляють більш ніж 4 терабайтами даних.

Реалізація SQL у PostgreSQL відповідає ANSI-SQL 92/99 стандартам. Він має повну підтримку вкладених запитів (включаючи вибірку з FROM), рівень читання тільки зафіксованих даних та транзакції, що серіалізуються. І тому що PostgreSQL має повністю реляційний системний каталог, що підтримує безліч схем баз даних, його каталог також доступний за допомогою інформаційної схеми відповідно до стандарту SQL.

Засоби забезпечення цілісності даних включають складені первинні ключі, зовнішні ключі з підтримкою заборони та каскадування змін/видалень, перевірку обмежень (constraints), обмеження унікальності та обмеження на непусті значення.

PostgreSQL також включає набір розширень та покращень. Серед зручностей є автоінкрементні поля на основі послідовностей, і конструкції LIMIT/OFFSET, що дозволяють повертати результуючий набір лише

частково. PostgreSQL підтримує складені, унікальні та функціональні індекси, для побудови (зберігання) яких можуть використовуватись методи B-дерева, R-дерева, перемішування, або GiST.

Інші поліпшення включають табличне успадкування, систему правил і події бази даних. Табличне успадкування надає створенню таблиць об'єктно-орієнтований ухил, що дозволяє при створенні бази даних успадковувати нові таблиці від старих, розглядаючи як базові класи. Більше того, PostgeSQL підтримує як одиночне, так і множинне спадкування.

PostgreSQL може виконувати процедури, що зберігаються, написані на різних мовах програмування, включаючи Java, Perl, Python, Ruby, Tcl, C/C++, і власному PL/pgSQL, аналогічному Oracle's PL/SQL. У стандартну бібліотеку функцій включені сотні вбудованих функцій – від базових математичних та рядкових операцій до криптографічних функцій та функцій, що забезпечують сумісність із Oracle. Тригери і процедури, що зберігаються, можуть бути написані на Сі і завантажені в базу даних в якості бібліотеки, дозволяючи тим самим розширювати її можливості. Також PostgreSQL включає засоби розробки, що дозволяють створювати типи даних користувача разом з функціями і операторами, що описують їх поведінку.

Поряд з безліччю мов, які можуть використовуватися для написання процедур, що зберігаються, існує і безліч інтерфейсних бібліотек, що дозволяють як інтерпретованим, так і компілюваним мовам взаємодіяти з PostgreSQL. Це інтерфейси Java (JDBC), ODBC, Perl, Python, Ruby, C, C++, PHP, Lisp, Scheme, Qt та інших.

Крім того, вихідний код PostgreSQL доступний під найбільш ліберальною з відкритих ліцензій - ліцензією BSD,

В рамках інформаційної системи моніторингу безпеки веб-сайтів, така СКБД є дуже важливою перевагою, адже життєвий цикл такого класу систем передбачає постійне розширення джерел даних, зміну типів даних тощо.

Фізичну модель представлено у додатку 2. Надано SQL-скрипт для створення БД включаючи дані контрольного прикладу, котрий буде описано в наступному розділі.

3.1.5. Контрольний приклад та розрахунок прогнозних обсягів БД

Розрахунок прогнозних обясгів БД здійснюється за настпуним алгоритмом:

1. Визначення обсягу пам’яті, котрий займає кожен тип даних обраної СКБД;
2. Підрахунок необхідного обсягу пам’яті для одного запису кожної таблиці;
3. Визначення зростання обсягів даних за одиницю часу;
4. Підрахунок загального обсягу даних за визначену одиницю часу.

Таблиця 3.21

Застосовані типи даних Postgresql та обсяги пам’яті

Тип даних	**Кількість байт**
integer	4 байти
text	1 байт на символ
character varying(n)	1 байт на символ
timestamp with time zone	8 байт
boolean	1 байт

Обсяг одного запису кожної таблиці:

1. AspNetUsers

```
"Id","UserName","NormalizedUserName","Email","NormalizedEmail","EmailConfirme
d","PasswordHash","SecurityStamp","ConcurrencyStamp","PhoneNumber","PhoneNumb
erConfirmed","TwoFactorEnabled","LockoutEnd","LockoutEnabled","AccessFailedCo
unt" "eb4cf9e3-5c95-4895-942f
8d5fe1113e85","180bmx360@gmail.com","180BMX360@GMAIL.COM","180bmx360@gmail.co
m","180BMX360@GMAIL.COM",False,"AQAAAAEAACcQAAAAEJ2hi4mpcEwPXTncHbGyguoRwJzlF
y34S5+s7AqD3P/oP2MJLhn5EuvRWSf+zNkQ/w==","LKXNJJ2COL6L5BXPJKV2TUU6IKMVNQVM","
c450eac3-cfd3-4c0f-aef8-47ec09e6b58c",NULL,False,False,NULL,True,0
```

Необхідний обсяг пам’яті: 2772 байти.

2. AspNetRoles

```
"Id","Name","NormalizedName","ConcurrencyStamp"
"5ce57b55-2a76-43aa-924f-4e1fba1f3099","Admin","Admin","stamp"
```

Необхідний обсяг пам’яті: 328 байт.

3. AspNetUserRoles

```
"UserId","RoleId"
"eb4cf9e3-5c95-4895-942f-8d5fe1113e85","5ce57b55-2a76-43aa-924f-4e1fba1f3099"
```

Необхідний обсяг пам’яті: 72 байти.

4. AspNetUserLogins

```
"LoginProvider","ProviderKey","ProviderDisplayName","UserId"
"b6f7ee33-eb1a-4cda-9341-78614bc332d7","882578fd-374d-499b-9cfe-
23b891ceff11","AuthToken","eb4cf9e3-5c95-4895-942f-8d5fe1113e85"
```

Необхідний обсяг пам’яті: 117 байт.

5. AspNetUserClaims

```
"Id","UserId","ClaimType","ClaimValue"
1,"eb4cf9e3-5c95-4895-942f-8d5fe1113e85","Edition","Full"
```

Необхідний обсяг пам’яті: 49 байт.

6. AspNetUserTokens

```
"UserId","LoginProvider","Name","Value"
"eb4cf9e3-5c95-4895-942f-8d5fe1113e85","b6f7ee33-eb1a-4cda-9341-
78614bc332d7","AuthToken","e76567e6-9471-46a2-995c-b1454262c813"
```

Необхідний обсяг пам’яті: 117 байт.

7. AspNetRoleClaims

```
"Id","RoleId","ClaimType","ClaimValue"
1,"5ce57b55-2a76-43aa-924f-4e1fba1f3099","Primary","True"
```

Необхідний обсяг пам’яті: 51 байт.

8. Domains

```
"Id","Url","VerificationType","LastVerification"
1,"https://altstudy.com.ua/",0,"2022-01-30 17:07:43.957063+02"
```

Необхідний обсяг пам’яті: 308 байт.

9. AspNetUserDomains

```
"UserId","DomainId"
```

```
"eb4cf9e3-5c95-4895-942f-8d5fe1113e85",1
```

Необхідний обсяг пам’яті: 40 байт.

10. Scans

```
"Id","TimeStamp","DomainId"
6,"2022-05-01 18:15:17.913384+03",1
```

Необхідний обсяг пам’яті: 51 байт.

11. Services

```
"Id","Port","Name","Version","State","ScanId"
18,22,"OpenSSH","6.6.1p1 Ubuntu 2ubuntu2.13",1,6
```

Необхідний обсяг пам’яті: 166 байт.

12.SslInfos

```
"Id","EffectiveDate","ExpirationDate","SslPolicyError","ScanId"
3,"-infinity","-infinity",1,6
```

Необхідний обсяг пам’яті: 28 байт.

13.HttpHeaders

```
"Id","ParentId","Name","Value","State","ScanId"
10,0,"X-Frame-Options","",1,6
```

Необхідний обсяг пам’яті: 224 байти.

14. HbHttpHeaders

```
"Id","ParentId","Name"
1,0,"X-Frame-Options"
```

Необхідний обсяг пам’яті: 208 байт.

15. HbHttpHeaderLocs

```
"Id","Locale","Description","HbHttpHeaderId"
1,"uk-UA","Дозволяє або забороняє відображати сторінку в якості (i)frame,
embed, object; Допустимі значення: DENY, SAMEORIGIN, ALLOW-FROM <url>.",1
```

Необхідний обсяг пам’яті: 513 байт.

16. DorkFiles

```
"Id","FileName","FileUrl","ScanId"
1,"Test.doc","https://test.com",6
```

Необхідний обсяг пам’яті: 313 байт.

17. Metadata

```
"Id","Name","Value","State","DorkFileId"
1,"SUBJECT","test@test.com",1,1
```

Необхідний обсяг пам’яті: 33 байта.

18. HbFileMetadata

```
"Id","Name"
1,"SUBJECT"
```

Необхідний обсяг пам’яті: 14 байт.

19.hbFileMetadataLocs

```
"Id","Locale","Description","HbFileMetadataId"
1,"uk-UA","Власник документа",1
```

Необхідний обсяг пам’яті: 43 байти.

Прогнозований приріст користувачів щомісячно – 20.

Кількість щомісячних сканувань одного веб-сайту– 150.

Прогнозоване зростання об’єму необхідної пам’яті щомісячно(Без урахування індексів):

1. Користувачі – 69.2 кб;
2. Сканування – 357 кб;

3.2. Програмне забезпечення

3.2.1. Вибір інструментальних засобів розробки

Основою для розробки інформаційної системи було вибрано мову програмування C#, а саме веб-фреймворк ASP.NET CORE(технологія WEB API), адже система являє собою веб-додаток.

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різноманітних веб-додатків: від невеликих веб-сайтів до великих веб-порталів та веб-сервісів.

ASP.NET Core може працювати поверх крос-платформного середовища .NET Core, яке може бути розгорнуто на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core можна створювати крос-платформні програми. І хоча Windows як середовище для розробки та розгортання програми досі переважає, але тепер розробники не обмежені тільки цією операційною системою. Тобто запускати веб-програми можна не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-програми можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.
Розгортання ІС в локальному і промисловому середовищах здійснюється за допомогою платформи Docker. Docker – це програмна платформа для швидкої розробки, тестування та розгортання програм. Docker упаковує програмне забезпечення в стандартизовані блоки, які називаються контейнерами . Кожен контейнер включає все необхідне роботи програми: бібліотеки, системні інструменти, код і середовище виконання. Завдяки Docker можна швидко розгортати та масштабувати програми в будь-якому середовищі та зберігати впевненість у тому, що код працюватиме.
В основі роботи Docker є стандартизований спосіб виконання коду. Docker – це операційна система контейнерів. Подібно до того, як віртуальна машина створює віртуальне уявлення апаратного забезпечення сервера (тобто усуває необхідність безпосередньо керувати таким), контейнери створюють віртуальне представлення серверної операційної системи. Після встановлення

на кожен сервер Docker надає доступ до простих команд, необхідних для збирання, запуску або зупинки контейнерів.

Використання Docker дозволяє швидше і ефективніше доставляти або переміщувати код, стандартизує операції, що виконуються додатками, і в цілому економить засоби, оптимізуючи використання ресурсів. Завдяки Docker користувачі одержують об'єкт, який з високою надійністю можна запускати на будь-якій платформі. Простий і зрозумілий синтаксис Docker забезпечує повний контроль над операціями, що виконуються. Повсюдне впровадження контейнерів має на увазі доступ до різноманітних інструментів та готових програм, які можна використовувати з Docker.

Для зручності розробки та підтримки цілісності даних було застосовано систему контролю версій Git. Git – розподілена система контролю версій, яка дає можливість розробникам відстежувати зміни у файлах та працювати над одним проектом разом із колегами. Git відомий своєю швидкістю, простим дизайном, підтримкою нелінійної розробки, повною децентралізацією та можливістю ефективно працювати з великими проектами.

В якості середовища розробки було застосовано наступні інструменти:

1. Microsoft Visual Studio 2022(Windows);

   Одне з найбільш популярних інтегрованих середовищ розробки на платформі .NET.

2. Microsoft Visual Studio Code(Windows, Linux, Mac);

   Кросплатформний редактор коду з відкритим вихідним кодом.

3. Postman.

   Інструмент для тестування API.

3.2.2. Архітектура ПЗ

До складових компонентів системи відносяться:

1. Компонент керування;

Представлений у вигляді програмного інтерфейсу, що реалізує функціональність управління в контексті користувача та пов’язанго

підприємства. Здійснюється отримання результатів моніторингу, та встановлення налаштувань. Також здійснюється кешування результатів моніторингу для швидкої обробки запитів.

2. Компонент моніторингу;

Представлений у вигляді автономної веб служби. Здійснює збір даних з визначеною періодичністю, формалізацію та збереження даних в БД.

3. БД середовища моніторингу(Postgresql);

Основне сховище даних. Зберігаються дані про користувачів, результати моніторингу, додаткові метадані. Будь-яка інформація, що стосується інфраструктури користувача є зашифрованою.

4. Redis БД(БД для кешування);

Окреме NoSql сховище даних для кешування відповіді сервера про стан системи. Використовується для швидкого доступу до даних.

5. HAProxy для балансування навантаження[29].

Розподіл навантаження та створення відмовостійкості на серверах.

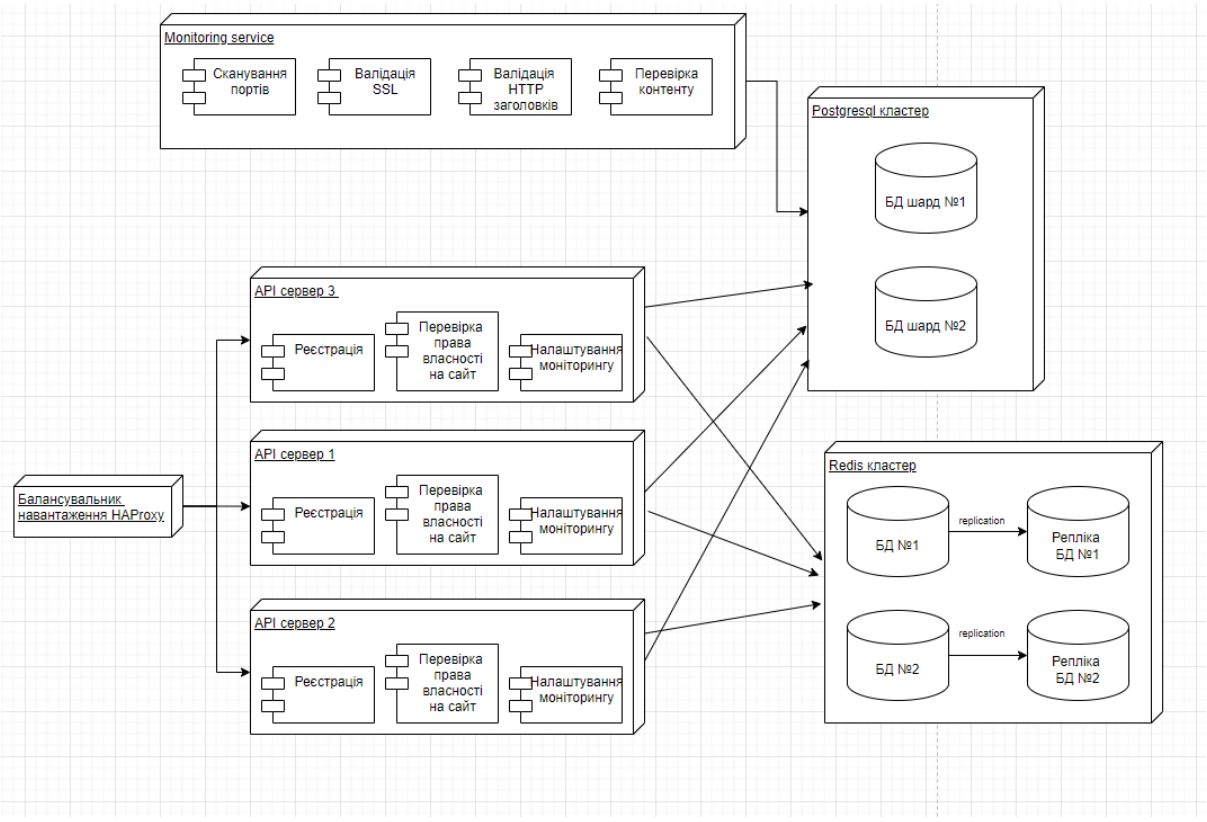


Рисунок. 3.2. Діграма розгортання в нотації UML

Для зменшення ймовірності блокування сервісу внаслідок функціонування алгоритмів моніторингу можна використовувати проксі-сервер або “ланцюг” проксі-серверів.

3.3. Технічне забезпечення

Кожен елемент описаного ПЗ має відповідні рекомендовані системні вимоги для стабільної роботи(один сервер):

1. ASP .NET CORE;
   - Процесор: 4 ядра;
   - Оперативна пам’ять: 8 GB;
   - Дисковий простір: 3 GB.
2. PostgreSQL;
   - Процесор: 1Ghz;
   - Оперативна пам’ять: 2GB;
   - Дисковий простір: 512 MB.
3. Redis;
   - Прцоесор: 8 ядер;
   - Оперативна пам’ять: 30 GB;
   - Дисковий простір: 10 GB.
4. HAProxy.
   - Процесор: 2 ядра;
   - Оперативна пам’ять: 2GB;
   - Дисковий простір.

Для ефективного функціонування системи, розглядається розгортання додатку на основі PaaS-моделі надання хмарних обчислень. В якості хостинг провайдера було обрано Heroku — хмарну платформу, що заснована на керованій контейнерній системі з інтегрованими службами передачі даних та розвиненою екосистемою для розгортання та запуску додатків. За безпеку,

архітектуру та налаштування серверів відповідають спеціалісти платформи, а тому це значно оптимізує витрати на обслуговування системи.

3.4. Результати реалізації інформаційної системи

Запропонована реалізація інформаційної системи[31] представляє собою мінімальну життєздатну версію проєктованої, що є готовою для подальших пілотних впроваджень або апробації, та включає в себе наступні складові:

1. Реєстрація веб-сайту та перевірка права власності;
2. Сканування портів сервера;
3. Перевірка сервісів на наявність вразливостей;
4. Перевірка SSL-сертифікату;
5. Перевірка заголовків HTTP пакетів.

Варто зазначити, що сканування портів без офіційної згоди власника цільового серверу є незаконним, а тому тестування системи здійснювалось використовуючи наступний перелік веб-сайтів:

1. scanme.nmap.org;

Офіційний веб-сайт розробника утиліти Nmap, призначений для ознайомлення з функціональними можливостями;

2. altstudy.com.ua.

Особистий веб-ресурс.

На відміну від більшості відомих засобів захисту веб-ресурсів, дана реалізація системи не потребує повноцінної інтеграції з цільовим веб-сайтом, що значно спрощує процес попереднього налаштування та підключення.

Дана інформаційна система є модульною, а тому кількість джерел моніторингу може бути необмеженим.

Розширення функціональних можливостей передбачає:

1. Пошук витоків корпоративної інформації серед джерел «глибинного вебу»;

2. Розробка та підключення алгоритмів динамічної побудови критеріїв валідації інформації з джерел моніторингу на основі даних про вразливості з відкритих джерел;
3. Розробка та підключення алгоритмів кількісної оцінки ризиків ІБ.

3.4.1. Результати тестування

Процес тестування показав, що результати циклу моніторингу відповідають поточним налаштування цільового веб-сайту.

Результати тестування використовуючи сайт scanme.nmap.org:

1. Сканування портів сервера;


```
"services": [
    {
        "id": 24,
        "port": 21,
        "name": "",
        "version": "",
        "state": 2,
        "vulnerabilities": null
    },
    {
        "id": 25,
        "port": 22,
        "name": "OpenSSH",
        "version": "6.6.1p1 Ubuntu 2ubuntu2.13",
        "state": 1,
        "vulnerabilities": []
    },
    {
        "id": 26,
        "port": 25,
        "name": "",
        "version": "",
        "state": 3,
        "vulnerabilities": null
    },
    {
        "id": 27,
        "port": 53,
        "name": "",
        "version": "",
        "state": 2,
        "vulnerabilities": null
    },
    {
        "id": 28,
        "port": 80,
        "name": "Apache httpd",
        "version": "2.4.7",
        "state": 1,
        "vulnerabilities": [
            {
                "id": 4,
                "code": "CVE-2021-44224",
                "url": "http://httpd.apache.org/security/vulnerabilities_24.html"
            }
        ]
    },
    {
        "id": 29,
        "port": 443,
        "name": "",
        "version": "",
        "state": 2,
        "vulnerabilities": null
    },
    {
        "id": 30,
        "port": 465,
        "name": "",
        "version": "",
        "state": 2,
        "vulnerabilities": null
    }
],
```


Рисунок 3.3 - Результат сканування портів за допомогою ІС

```
PS C:\Users\Ivan Kroshko> nmap --privileged  -sV  -O  -p 21,22,25,53,465,80,443 scanme.nmap.org
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-03 12:42 FLE Summer Time
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.18s latency).

PORT    STATE    SERVICE VERSION
21/tcp  closed   ftp
22/tcp  open     ssh     OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
25/tcp  filtered smtp
53/tcp  closed   domain
80/tcp  open     http    Apache httpd 2.4.7 ((Ubuntu))
443/tcp closed   https
465/tcp closed   smtps
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 12 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 12.49 seconds
PS C:\Users\Ivan Kroshko>
```

Рисунок 3.4 - Результат сканування портів scanme.nmap.org за допомогою утиліти Nmap

2. Перевірка сервісів на наявність вразливстей;


```
{
    "id": 35,
    "port": 80,
    "name": "Apache httpd",
    "version": "2.4.7",
    "state": 1,
    "vulnerabilities": [
        {
            "id": 5,
            "code": "CVE-2021-44224",
            "url": "http://httpd.apache.org/security/vulnerabilities_24.html"
        }
    ]
},
```


Рисунок 3.5 - Результат перевірки на наявність вразливостей Apache HTTP server 2.4.7 засобами IC

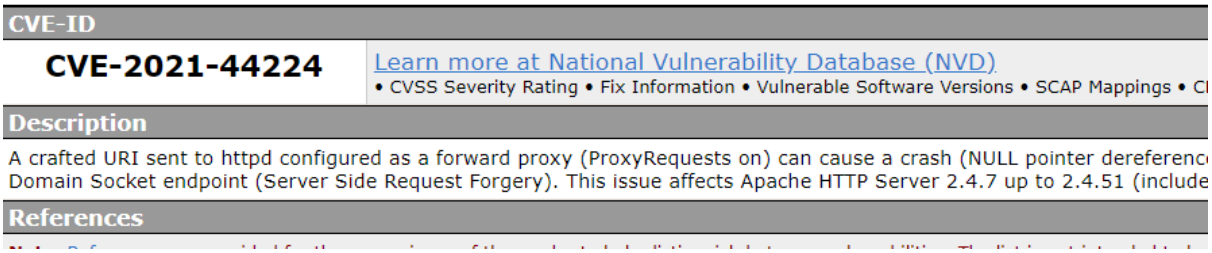


Рисунок 3.6 - Результат пошуку вразливостей Apache HTTP server 2.4.7 серед відкритої бази даних вразливостей CVE.

3. Перевірка SSL-сертифікату;


```
"sslInfo": {
    "id": 4,
    "effectiveDate": "0001-01-01T00:00:00",
    "expirationDate": "0001-01-01T00:00:00",
    "sslPolicyError": 1,
    "scanId": 7
},
```

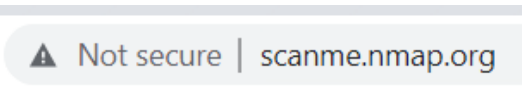



Рисунок 3.7 - Відсутність захищеного з’єднання з веб-сайтом scanme.nmap.org

4. Перевірка заголовків HTTP-пакетів.


```
"headers": [
    {
        "id": 14,
        "parentId": 0,
        "name": "X-Frame-Options",
        "value": "",
        "state": 1
    },
    {
        "id": 15,
        "parentId": 0,
        "name": "X-Content-Type-Options",
        "value": "",
        "state": 1
    }
```


Рисунок 3.8 - Результат аналізу наявності заголовків безпеки scanme.nmap.org засобами IC.

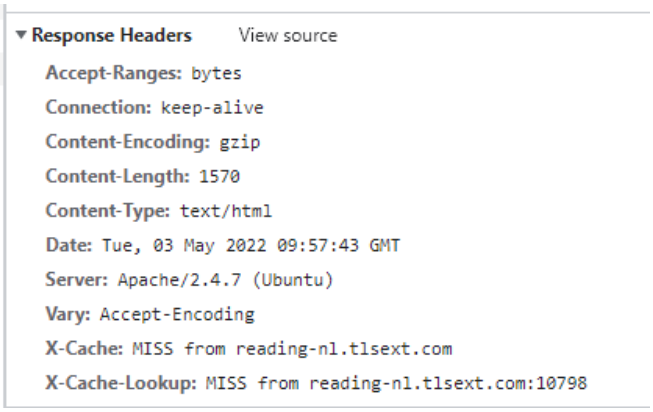


Рисунок 3.9 - Фактичні заголовки відповіді scanme.nmap.org

ВИСНОВКИ

В результаті виконання роботи було спроєктовано та розроблено інформаційну систему моніторингу безпеки веб-сайтів. Аналіз предметної області показав, що для розробки сучасних веб-додатків використовуються різноманітні засоби та мови програмування. Зокрема, широко розповсюджені динамічні мови програмування, такі як PHP, Pyhton тощо. Тим не менш, враховуючи найбільш розповсюджені вразливості, а також алгоритми здійснення кібератак, спрямованих на веб-додатки, було визначено перелік основних джерел моніторингу, котрі є достатніми для виявлення критичних недоліків конфігурації веб-сайтів:

1. Відкриті порти сервера;
2. Версії ПЗ та їх вразливості;
3. Наявність SSL-сертифікату та його актуальність;
4. Наявність відповідних заголовків безпеки HTTP-пакетів;
5. Наявність документів та файлів серед відкритих джерел, котрі можуть містити в собі «чутливу» інформацію.

У зв’язку з постійним оновленням ПЗ, а також виявленням нових вразливостей, моніторинг має здійснюватись регулярно, щоб своєчасно попереджувати власника або адміністратора веб-сайту про необхідність виправлення існуючих проблем.

Тестування реалізованих алгоритмів показало, що інформація, отримана в результаті моніторингу, може бути корисною не тільки для попередження кібрезагроз, а й для їх ескалації. Саме тому, дуже важливою складовою інформаційної системи моніторингу безпеки веб-сайтів є перевірка права власності на цільовий веб-ресурс, щоб виключити можливість потрапляння результатів до рук кіберзлочинців.

Перспективи застосування представленої інформаційної системи грунтуються на необхідності постійного контролю компаніями стану інформаційної безпеки власних веб-ресурсів. Такий підхід дозволяє своєчасно

реагувати на виявлені проблеми, здійснювати їх аналіз, а також розширювати перелік джерел моніторингу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Об’єкти захисту інформації та технічні канали її витоку [Електронний ресурс]. – Режим доступу: World Wide Web. – URL: https://infopedia.su/1x978f.html
2. Мережеві атаки, можливості та недоліки мережевих екранів [Електронний ресурс]. – Режим доступу: World Wide Web. – URL https://ukrbukva.net/page,2,91957-Setevye-ataki-vozmozhnosti-i-nedostatki-setevyhekranov.html
3. Типи мережевих атак, їх опису, засоби боротьби [Електронний ресурс]. – Режим доступу: World Wide Web. – URL https://moluch.ru/conf/tech/archive/5/1115/
4. Мережеві атаки, можливості та недоліки мережевих екранів [Електронний ресурс]. –Режим доступу: World Wide Web. – URL https://ukrbukva.net/page,6,91957-Setevye-ataki-vozmozhnosti-i-nedostatki-setevyhekranov.html
5. XSS-атаки [Електронний ресурс]. – Режим доступу: http://www.univd.edu.ua/science-issue/issue/3345
6. IP-спуфинг [Електронний ресурс]. – Режим доступу: http://um.co.ua/11/11-3/11-30168.html
7. Khandelwal, Shekhar & Das, Rik. (2022). Phishing and Cybersecurity. 10.1201/9781003217381-1.
8. Gawali, Prof & Shaikh, Sajid & Pawar, Pranav & Thakur, Utkarsh. (2022). SQL Injection. International Journal of Advanced Research in Science, Communication and Technology. 463-466. 10.48175/IJARSCT-3292.
9. Брутфорс [Електронний ресурс]. – Режим доступу: https://www.ptsecurity.com/ru-ru/research/analytics/web-application-attacks-2018/
10. Аналіз існуючих інформаційних атак на веб-ресурси та методів і засобів захисту від них [Електронний ресурс]. – Режим доступу:

http://refua.in.ua/analiz-isnuyuchih-informacijnih-atak-na-veb-resursi-ta-metodiv.html

11. Аналіз сучасних Web-вразливостей [Електронний ресурс]. – Режим доступу: http://www.rusnauka.com/pdf/255343.pdf
12. Мережеві атаки, можливості та недоліки мережевих екранів[Електронний ресурс]. – Режим доступу: https://ukrbukva.net/page,5,91957-Setevyeataki-vozmozhnosti-i-nedostatki-setevyh-ekranov.html
13. Безпека і переповнення буфера [Електронний ресурс]. – Режим доступу: http://pautina34.ru/p=167/
14. Безпека комп'ютерних мереж [Електронний ресурс]. – Режим доступу: https://svitppt.com.ua/informatika/bezpeka-kompyuternih-merezh.html
15. Захист баз даних [Електронний ресурс]. – Режим доступу: http://ua.waykun.com/articles/zahist-baz-danih-2-stattja-storinka-7.php
16. Актуальні питання протидії кіберзлочинності та торгівлі людьми. [Електронний ресурс]. – Режим доступу: http://www.univd.edu.ua/scienceissue/issue/3345
17. Усунення небезпеки XPath-впровадження [Електронний ресурс]. – Режим доступу:http://easy-code.com.ua/2012/09/usunennya-nebezpeki-xpathvprovadzhennya-isxodniki-rizne-programuvannya-statti/
18. Оцінка стійкості роботи комп'ютерної інформаційної системи в умовах дії загрозливих чинників НС [Електронний ресурс]. – Режим доступу:https://studwood.ru/2388680/informatika/otsinka_stiykosti_roboti_kompyuternoyi_inform atsiynoyi_sistemi_umovah_zagrozlivih_chinnikiv
19. Як захистити веб-додатки: основні поради, інструменти, корисні посилання [Електронний ресурс]. – Режим доступу: https://echo.lviv.ua/dev/6231
20. Hwang J., Kim M. Effective Detecting Method of Nmap Idle Scan. JOURNAL OF ADVANCED INFORMATION TECHNOLOGY AND CONVERGENCE. 2019. Vol. 9, no. 1. P. 1–10. URL: https://doi.org/10.14801/jaitc.2019.9.1.1.
21. OWASP: Available at http://www.owasp.org/index.php/ Category: OWASP Top Ten Project, 2017. [6] Bozic, Josip, and Franz Wotawa. "PURITY: a

Planning-based security testing tool." Software Quality, Reliability and Security-Companion (QRS-C), 2015 IEEE International Conference on. IEEE, 2015.

22. Goel, Jai Narayan, and B. M. Mehtre. "Vulnerability assessment & penetration testing as a cyber defence technology." Procedia Computer Science 57 (2015): 710-715

23. M.F Jnena 2013. Modern Approach for WEB Applications Vulnerability Analysis retrieve on 27/8/2015 from http://library.iugaza.edu.ps/thesis/1 09553 .pdf 229

24. X. Zhang, and Z. Wang. 2010. Notice of Retraction A Static Analysis Scanner for Detecting Web Application Injection Vulnerabilities for ASP Program. Paper presented at the eBusiness and Information System Security (EBISS), 2010 2nd International Conference on

25. Mihai, Ioan-Cosmin & Genoe, Ray & PRUNĂ, Ștefan. (2015). Strategies for Monitoring Website Security against Cyber-Attacks. International Journal of Information Security and Cybercrime. 4. 9-14. 10.19107/IJISC.2015.01.01.

26. Doupé, Adam, et al. "Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner." USENIX Security Symposium. Vol. 14. 2012.

27. F. Duchene, S. Rawat, J.-L. Richier, and R. Groz, “Kameleon- Fuzz: Evolutionary Fuzzing for Black-Box XSS Detection,” in CODASPY. ACM, 2014, pp. 37–48

28. B. Garn, I. Kapsalis, D. E. Simos, and S. Winkle8, “On the applicability of combinatorial testing to web application security testing: A case study,” in Proceedings of the 2nd International Workshop on Joining AcadeMiA and Industry Contributions to Testing Automation (JAMAICA’14). ACM, 2014

29. Мікула М., Коцюк Ю., Мікула О. Організація баз даних та знань. Вид-во Нац. ун-ту «Острозь-ка акад.», 2021. URL: https://doi.org/10.25264/978-617-8041-08-3.

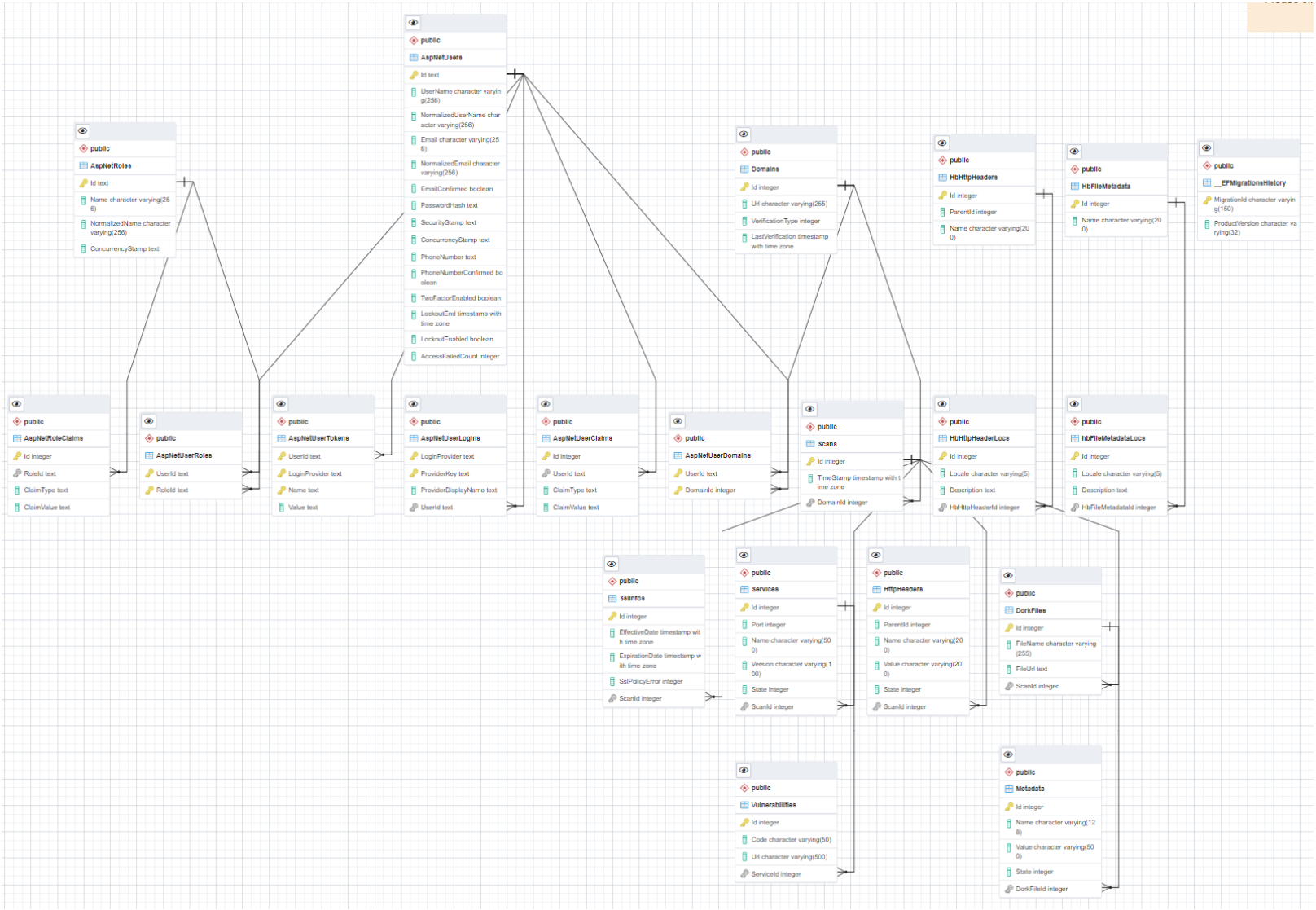
30. Щербаков Є. В., Щербакова М. Є. Особливості масштабування веб-додатків. ВІСНИК СХІДНОУКРАЇНСЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ імені Володимира Даля. 2021. № 8(264). С. 15–19. URL: https://doi.org/10.33216/1998-7927-2020-264-8-15-19.

31. Крошко І.А. Гордієнко І. В. Інформаційна система моніторингу безпеки веб-сайтів // Інформаційні технології в соціокультурній сфері, освіті та економіці: матеріали Міжнародної наук.-практ. конф. здобувачів вищої освіти і молодих учених 19 - 20 квітня 2022 р. / М-во освіти і науки України; Київ. нац. ун-т культури і мистецтв. Київ, Видавничий центр КНУКіМ, 2022. – Подано на конференцію. *URL:* http://knukim.edu.ua/naukova-robota/naukovi-konferentsiyi/

ДОДАТКИ

Додаток 1

Інфологічна модель проєктованої БД



Фізична модель проєктованої БД

```
CREATE TABLE public."AspNetRoleClaims" (
    "Id" integer NOT NULL,
    "RoleId" text NOT NULL,
    "ClaimType" text,
    "ClaimValue" text
);


ALTER TABLE public."AspNetRoleClaims" OWNER TO postgres;

--
-- TOC entry 214 (class 1259 OID 16513)
-- Name: AspNetRoleClaims_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."AspNetRoleClaims" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."AspNetRoleClaims_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 210 (class 1259 OID 16493)
-- Name: AspNetRoles; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetRoles" (
    "Id" text NOT NULL,
    "Name" character varying(256),
    "NormalizedName" character varying(256),
    "ConcurrencyStamp" text
);


ALTER TABLE public."AspNetRoles" OWNER TO postgres;

--
-- TOC entry 217 (class 1259 OID 16527)
-- Name: AspNetUserClaims; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUserClaims" (
    "Id" integer NOT NULL,
    "UserId" text NOT NULL,
    "ClaimType" text,
    "ClaimValue" text
);


ALTER TABLE public."AspNetUserClaims" OWNER TO postgres;

--
-- TOC entry 216 (class 1259 OID 16526)
```

```
-- Name: AspNetUserClaims_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."AspNetUserClaims" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."AspNetUserClaims_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 221 (class 1259 OID 16580)
-- Name: AspNetUserDomains; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUserDomains" (
    "UserId" text NOT NULL,
    "DomainId" integer NOT NULL
);


ALTER TABLE public."AspNetUserDomains" OWNER TO postgres;

--
-- TOC entry 218 (class 1259 OID 16539)
-- Name: AspNetUserLogins; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUserLogins" (
    "LoginProvider" text NOT NULL,
    "ProviderKey" text NOT NULL,
    "ProviderDisplayName" text,
    "UserId" text NOT NULL
);


ALTER TABLE public."AspNetUserLogins" OWNER TO postgres;

--
-- TOC entry 219 (class 1259 OID 16551)
-- Name: AspNetUserRoles; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUserRoles" (
    "UserId" text NOT NULL,
    "RoleId" text NOT NULL
);


ALTER TABLE public."AspNetUserRoles" OWNER TO postgres;

--
-- TOC entry 220 (class 1259 OID 16568)
-- Name: AspNetUserTokens; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUserTokens" (
    "UserId" text NOT NULL,
    "LoginProvider" text NOT NULL,
    "Name" text NOT NULL,
```

```
    "Value" text
);


ALTER TABLE public."AspNetUserTokens" OWNER TO postgres;

--
-- TOC entry 211 (class 1259 OID 16500)
-- Name: AspNetUsers; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."AspNetUsers" (
    "Id" text NOT NULL,
    "UserName" character varying(256),
    "NormalizedUserName" character varying(256),
    "Email" character varying(256),
    "NormalizedEmail" character varying(256),
    "EmailConfirmed" boolean NOT NULL,
    "PasswordHash" text,
    "SecurityStamp" text,
    "ConcurrencyStamp" text,
    "PhoneNumber" text,
    "PhoneNumberConfirmed" boolean NOT NULL,
    "TwoFactorEnabled" boolean NOT NULL,
    "LockoutEnd" timestamp with time zone,
    "LockoutEnabled" boolean NOT NULL,
    "AccessFailedCount" integer NOT NULL
);


ALTER TABLE public."AspNetUsers" OWNER TO postgres;

--
-- TOC entry 213 (class 1259 OID 16508)
-- Name: Domains; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Domains" (
    "Id" integer NOT NULL,
    "Url" character varying(255) NOT NULL,
    "VerificationType" integer NOT NULL,
    "LastVerification" timestamp with time zone NOT NULL
);


ALTER TABLE public."Domains" OWNER TO postgres;

--
-- TOC entry 212 (class 1259 OID 16507)
-- Name: Domains_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."Domains" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."Domains_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 237 (class 1259 OID 24818)
```

```
-- Name: DorkFiles; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."DorkFiles" (
    "Id" integer NOT NULL,
    "FileName" character varying(255) NOT NULL,
    "FileUrl" text NOT NULL,
    "ScanId" integer
);


ALTER TABLE public."DorkFiles" OWNER TO postgres;

--
-- TOC entry 236 (class 1259 OID 24817)
-- Name: DorkFiles_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."DorkFiles" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."DorkFiles_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 241 (class 1259 OID 24840)
-- Name: HbFileMetadata; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."HbFileMetadata" (
    "Id" integer NOT NULL,
    "Name" character varying(200) NOT NULL
);


ALTER TABLE public."HbFileMetadata" OWNER TO postgres;

--
-- TOC entry 240 (class 1259 OID 24839)
-- Name: HbFileMetadata_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."HbFileMetadata" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."HbFileMetadata_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 235 (class 1259 OID 16666)
-- Name: HbHttpHeaderLocs; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."HbHttpHeaderLocs" (
```

```
    "Id" integer NOT NULL,
    "Locale" character varying(5) NOT NULL,
    "Description" text NOT NULL,
    "HbHttpHeaderId" integer
);


ALTER TABLE public."HbHttpHeaderLocs" OWNER TO postgres;

--
-- TOC entry 234 (class 1259 OID 16665)
-- Name: HbHttpHeaderLocs_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."HbHttpHeaderLocs" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."HbHttpHeaderLocs_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 231 (class 1259 OID 16654)
-- Name: HbHttpHeaders; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."HbHttpHeaders" (
    "Id" integer NOT NULL,
    "ParentId" integer NOT NULL,
    "Name" character varying(200) NOT NULL
);


ALTER TABLE public."HbHttpHeaders" OWNER TO postgres;

--
-- TOC entry 230 (class 1259 OID 16653)
-- Name: HbHttpHeaders_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."HbHttpHeaders" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT
AS IDENTITY (
    SEQUENCE NAME public."HbHttpHeaders_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 233 (class 1259 OID 16660)
-- Name: HttpHeaders; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."HttpHeaders" (
    "Id" integer NOT NULL,
    "ParentId" integer NOT NULL,
```

```
    "Name" character varying(200) NOT NULL,
    "Value" character varying(200) NOT NULL,
    "State" integer NOT NULL,
    "ScanId" integer
);


ALTER TABLE public."HttpHeaders" OWNER TO postgres;

--
-- TOC entry 232 (class 1259 OID 16659)
-- Name: HttpHeaders_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."HttpHeaders" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT
AS IDENTITY (
    SEQUENCE NAME public."HttpHeaders_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 239 (class 1259 OID 24826)
-- Name: Metadata; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Metadata" (
    "Id" integer NOT NULL,
    "Name" character varying(128) NOT NULL,
    "Value" character varying(500),
    "State" integer NOT NULL,
    "DorkFileId" integer
);


ALTER TABLE public."Metadata" OWNER TO postgres;

--
-- TOC entry 238 (class 1259 OID 24825)
-- Name: Metadata_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."Metadata" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."Metadata_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 223 (class 1259 OID 16608)
-- Name: Scans; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Scans" (
    "Id" integer NOT NULL,
    "TimeStamp" timestamp with time zone NOT NULL,
```

```
    "DomainId" integer
);


ALTER TABLE public."Scans" OWNER TO postgres;

--
-- TOC entry 222 (class 1259 OID 16607)
-- Name: Scans_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."Scans" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."Scans_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 225 (class 1259 OID 16614)
-- Name: Services; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Services" (
    "Id" integer NOT NULL,
    "Port" integer NOT NULL,
    "Name" character varying(500) NOT NULL,
    "Version" character varying(100),
    "State" integer NOT NULL,
    "ScanId" integer
);


ALTER TABLE public."Services" OWNER TO postgres;

--
-- TOC entry 224 (class 1259 OID 16613)
-- Name: Services_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."Services" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."Services_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 229 (class 1259 OID 16642)
-- Name: SslInfos; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."SslInfos" (
    "Id" integer NOT NULL,
    "EffectiveDate" timestamp with time zone NOT NULL,
    "ExpirationDate" timestamp with time zone NOT NULL,
    "SslPolicyError" integer NOT NULL,
```

```
    "ScanId" integer DEFAULT 0 NOT NULL
);


ALTER TABLE public."SslInfos" OWNER TO postgres;

--
-- TOC entry 228 (class 1259 OID 16641)
-- Name: SslInfos_Id_seq; Type: SEQUENCE; Schema: public; Owner: postgres
--

ALTER TABLE public."SslInfos" ALTER COLUMN "Id" ADD GENERATED BY DEFAULT AS
IDENTITY (
    SEQUENCE NAME public."SslInfos_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 227 (class 1259 OID 16627)
-- Name: Vulnerabilities; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."Vulnerabilities" (
    "Id" integer NOT NULL,
    "Code" character varying(50) NOT NULL,
    "Url" character varying(500) NOT NULL,
    "ServiceId" integer
);


ALTER TABLE public."Vulnerabilities" OWNER TO postgres;

--
-- TOC entry 226 (class 1259 OID 16626)
-- Name: Vulnerabilities Id seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."Vulnerabilities" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."Vulnerabilities_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
-- TOC entry 209 (class 1259 OID 16488)
-- Name: __EFMigrationsHistory; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."__EFMigrationsHistory" (
    "MigrationId" character varying(150) NOT NULL,
    "ProductVersion" character varying(32) NOT NULL
);
```

```
ALTER TABLE public."__EFMigrationsHistory" OWNER TO postgres;

--
-- TOC entry 243 (class 1259 OID 24846)
-- Name: hbFileMetadataLocs; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public."hbFileMetadataLocs" (
    "Id" integer NOT NULL,
    "Locale" character varying(5) NOT NULL,
    "Description" text NOT NULL,
    "HbFileMetadataId" integer
);


ALTER TABLE public."hbFileMetadataLocs" OWNER TO postgres;

--
-- TOC entry 242 (class 1259 OID 24845)
-- Name: hbFileMetadataLocs_Id_seq; Type: SEQUENCE; Schema: public; Owner:
postgres
--

ALTER TABLE public."hbFileMetadataLocs" ALTER COLUMN "Id" ADD GENERATED BY
DEFAULT AS IDENTITY (
    SEQUENCE NAME public."hbFileMetadataLocs_Id_seq"
    START WITH 1
    INCREMENT BY 1
    NO MINVALUE
    NO MAXVALUE
    CACHE 1
);


--
--

ALTER TABLE ONLY public."AspNetRoleClaims"
    ADD CONSTRAINT "PK_AspNetRoleClaims" PRIMARY KEY ("Id");


--
-- TOC entry 3261 (class 2606 OID 16499)
-- Name: AspNetRoles PK_AspNetRoles; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."AspNetRoles"
    ADD CONSTRAINT "PK_AspNetRoles" PRIMARY KEY ("Id");


--
-- TOC entry 3275 (class 2606 OID 16533)
-- Name: AspNetUserClaims PK_AspNetUserClaims; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserClaims"
    ADD CONSTRAINT "PK_AspNetUserClaims" PRIMARY KEY ("Id");


--
-- TOC entry 3286 (class 2606 OID 16586)
-- Name: AspNetUserDomains PK_AspNetUserDomains; Type: CONSTRAINT; Schema:
public; Owner: postgres
```

```
--

ALTER TABLE ONLY public."AspNetUserDomains"
    ADD CONSTRAINT "PK_AspNetUserDomains" PRIMARY KEY ("UserId", "DomainId");


--
-- TOC entry 3278 (class 2606 OID 16545)
-- Name: AspNetUserLogins PK_AspNetUserLogins; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserLogins"
    ADD CONSTRAINT "PK_AspNetUserLogins" PRIMARY KEY ("LoginProvider",
"ProviderKey");


--
-- TOC entry 3281 (class 2606 OID 16557)
-- Name: AspNetUserRoles PK_AspNetUserRoles; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserRoles"
    ADD CONSTRAINT "PK_AspNetUserRoles" PRIMARY KEY ("UserId", "RoleId");


--
-- TOC entry 3283 (class 2606 OID 16574)
-- Name: AspNetUserTokens PK_AspNetUserTokens; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserTokens"
    ADD CONSTRAINT "PK_AspNetUserTokens" PRIMARY KEY ("UserId",
"LoginProvider", "Name");


--
-- TOC entry 3265 (class 2606 OID 16506)
-- Name: AspNetUsers PK_AspNetUsers; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."AspNetUsers"
    ADD CONSTRAINT "PK_AspNetUsers" PRIMARY KEY ("Id");


--
-- TOC entry 3269 (class 2606 OID 16512)
-- Name: Domains PK_Domains; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."Domains"
    ADD CONSTRAINT "PK_Domains" PRIMARY KEY ("Id");


--
-- TOC entry 3309 (class 2606 OID 24824)
-- Name: DorkFiles PK_DorkFiles; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."DorkFiles"
```

```
    ADD CONSTRAINT "PK_DorkFiles" PRIMARY KEY ("Id");


--
-- TOC entry 3314 (class 2606 OID 24844)
-- Name: HbFileMetadata PK_HbFileMetadata; Type: CONSTRAINT; Schema: public;
Owner: postgres
--

ALTER TABLE ONLY public."HbFileMetadata"
    ADD CONSTRAINT "PK_HbFileMetadata" PRIMARY KEY ("Id");


--
-- TOC entry 3306 (class 2606 OID 16672)
-- Name: HbHttpHeaderLocs PK_HbHttpHeaderLocs; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."HbHttpHeaderLocs"
    ADD CONSTRAINT "PK_HbHttpHeaderLocs" PRIMARY KEY ("Id");


--
-- TOC entry 3300 (class 2606 OID 16658)
-- Name: HbHttpHeaders PK_HbHttpHeaders; Type: CONSTRAINT; Schema: public;
Owner: postgres
--

ALTER TABLE ONLY public."HbHttpHeaders"
    ADD CONSTRAINT "PK_HbHttpHeaders" PRIMARY KEY ("Id");


--
-- TOC entry 3303 (class 2606 OID 16664)
-- Name: HttpHeaders PK_HttpHeaders; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."HttpHeaders"
    ADD CONSTRAINT "PK_HttpHeaders" PRIMARY KEY ("Id");


--
-- TOC entry 3312 (class 2606 OID 24832)
-- Name: Metadata PK_Metadata; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."Metadata"
    ADD CONSTRAINT "PK_Metadata" PRIMARY KEY ("Id");


--
-- TOC entry 3289 (class 2606 OID 16612)
-- Name: Scans PK_Scans; Type: CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."Scans"
    ADD CONSTRAINT "PK_Scans" PRIMARY KEY ("Id");


--
-- TOC entry 3292 (class 2606 OID 16620)
```

```
-- Name: Services PK_Services; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."Services"
    ADD CONSTRAINT "PK_Services" PRIMARY KEY ("Id");


--
-- TOC entry 3298 (class 2606 OID 16646)
-- Name: SslInfos PK_SslInfos; Type: CONSTRAINT; Schema: public; Owner:
postgres
--

ALTER TABLE ONLY public."SslInfos"
    ADD CONSTRAINT "PK_SslInfos" PRIMARY KEY ("Id");


--
-- TOC entry 3295 (class 2606 OID 16633)
-- Name: Vulnerabilities PK_Vulnerabilities; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."Vulnerabilities"
    ADD CONSTRAINT "PK_Vulnerabilities" PRIMARY KEY ("Id");


--
-- TOC entry 3259 (class 2606 OID 16492)
-- Name: __EFMigrationsHistory PK___EFMigrationsHistory; Type: CONSTRAINT;
Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."__EFMigrationsHistory"
    ADD CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId");


--
-- TOC entry 3317 (class 2606 OID 24852)
-- Name: hbFileMetadataLocs PK_hbFileMetadataLocs; Type: CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."hbFileMetadataLocs"
    ADD CONSTRAINT "PK_hbFileMetadataLocs" PRIMARY KEY ("Id");


--
-- TOC entry 3263 (class 1259 OID 16603)
-- Name: EmailIndex; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "EmailIndex" ON public."AspNetUsers" USING btree
("NormalizedEmail");


--
-- TOC entry 3270 (class 1259 OID 16597)
-- Name: IX_AspNetRoleClaims_RoleId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_AspNetRoleClaims_RoleId" ON public."AspNetRoleClaims" USING
btree ("RoleId");
```

```
--
-- TOC entry 3273 (class 1259 OID 16599)
-- Name: IX_AspNetUserClaims_UserId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_AspNetUserClaims_UserId" ON public."AspNetUserClaims" USING
btree ("UserId");


--
-- TOC entry 3284 (class 1259 OID 16600)
-- Name: IX_AspNetUserDomains_DomainId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_AspNetUserDomains_DomainId" ON public."AspNetUserDomains"
USING btree ("DomainId");


--
-- TOC entry 3276 (class 1259 OID 16601)
-- Name: IX_AspNetUserLogins_UserId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_AspNetUserLogins_UserId" ON public."AspNetUserLogins" USING
btree ("UserId");


--
-- TOC entry 3279 (class 1259 OID 16602)
-- Name: IX_AspNetUserRoles_RoleId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_AspNetUserRoles_RoleId" ON public."AspNetUserRoles" USING
btree ("RoleId");


--
-- TOC entry 3267 (class 1259 OID 16606)
-- Name: IX_Domains_Url; Type: INDEX; Schema: public; Owner: postgres
--

CREATE UNIQUE INDEX "IX_Domains_Url" ON public."Domains" USING btree ("Url");


--
-- TOC entry 3307 (class 1259 OID 24859)
-- Name: IX_DorkFiles_ScanId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "IX_DorkFiles_ScanId" ON public."DorkFiles" USING btree
("ScanId");


--
-- TOC entry 3304 (class 1259 OID 16678)
-- Name: IX_HbHttpHeaderLocs_HbHttpHeaderId; Type: INDEX; Schema: public;
Owner: postgres
--
```

```
CREATE INDEX "IX_HbHttpHeaderLocs_HbHttpHeaderId" ON
public."HbHttpHeaderLocs" USING btree ("HbHttpHeaderId");


--
-- TOC entry 3301 (class 1259 OID 24799)
-- Name: IX_HttpHeaders_ScanId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "IX_HttpHeaders_ScanId" ON public."HttpHeaders" USING btree
("ScanId");


--
-- TOC entry 3310 (class 1259 OID 24838)
-- Name: IX_Metadata_DorkFileId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "IX_Metadata_DorkFileId" ON public."Metadata" USING btree
("DorkFileId");


--
-- TOC entry 3287 (class 1259 OID 24798)
-- Name: IX_Scans_DomainId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "IX_Scans_DomainId" ON public."Scans" USING btree ("DomainId");


--
-- TOC entry 3290 (class 1259 OID 16639)
-- Name: IX_Services_ScanId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE INDEX "IX_Services_ScanId" ON public."Services" USING btree
("ScanId");


--
-- TOC entry 3296 (class 1259 OID 24811)
-- Name: IX_SslInfos_ScanId; Type: INDEX; Schema: public; Owner: postgres
--

CREATE UNIQUE INDEX "IX_SslInfos_ScanId" ON public."SslInfos" USING btree
("ScanId");


--
-- TOC entry 3293 (class 1259 OID 16640)
-- Name: IX_Vulnerabilities_ServiceId; Type: INDEX; Schema: public; Owner:
postgres
--

CREATE INDEX "IX_Vulnerabilities_ServiceId" ON public."Vulnerabilities" USING
btree ("ServiceId");


--
-- TOC entry 3315 (class 1259 OID 24858)
-- Name: IX_hbFileMetadataLocs_HbFileMetadataId; Type: INDEX; Schema: public;
Owner: postgres
--
```

```
CREATE INDEX "IX_hbFileMetadataLocs_HbFileMetadataId" ON
public."hbFileMetadataLocs" USING btree ("HbFileMetadataId");


--
-- TOC entry 3262 (class 1259 OID 16598)
-- Name: RoleNameIndex; Type: INDEX; Schema: public; Owner: postgres
--

CREATE UNIQUE INDEX "RoleNameIndex" ON public."AspNetRoles" USING btree
("NormalizedName");


--
-- TOC entry 3266 (class 1259 OID 16604)
-- Name: UserNameIndex; Type: INDEX; Schema: public; Owner: postgres
--

CREATE UNIQUE INDEX "UserNameIndex" ON public."AspNetUsers" USING btree
("NormalizedUserName");


--
-- TOC entry 3318 (class 2606 OID 16521)
-- Name: AspNetRoleClaims FK_AspNetRoleClaims_AspNetRoles_RoleId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetRoleClaims"
    ADD CONSTRAINT "FK_AspNetRoleClaims_AspNetRoles_RoleId" FOREIGN KEY
("RoleId") REFERENCES public."AspNetRoles"("Id") ON DELETE CASCADE;


--
-- TOC entry 3319 (class 2606 OID 16534)
-- Name: AspNetUserClaims FK_AspNetUserClaims_AspNetUsers_UserId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserClaims"
    ADD CONSTRAINT "FK_AspNetUserClaims_AspNetUsers_UserId" FOREIGN KEY
("UserId") REFERENCES public."AspNetUsers"("Id") ON DELETE CASCADE;


--
-- TOC entry 3324 (class 2606 OID 16587)
-- Name: AspNetUserDomains FK_AspNetUserDomains_AspNetUsers_UserId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserDomains"
    ADD CONSTRAINT "FK_AspNetUserDomains_AspNetUsers_UserId" FOREIGN KEY
("UserId") REFERENCES public."AspNetUsers"("Id") ON DELETE CASCADE;


--
-- TOC entry 3325 (class 2606 OID 16592)
-- Name: AspNetUserDomains FK_AspNetUserDomains_Domains_DomainId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserDomains"
    ADD CONSTRAINT "FK_AspNetUserDomains_Domains_DomainId" FOREIGN KEY
("DomainId") REFERENCES public."Domains"("Id") ON DELETE CASCADE;
```

```
--
-- TOC entry 3320 (class 2606 OID 16546)
-- Name: AspNetUserLogins FK_AspNetUserLogins_AspNetUsers_UserId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserLogins"
    ADD CONSTRAINT "FK_AspNetUserLogins_AspNetUsers_UserId" FOREIGN KEY
("UserId") REFERENCES public."AspNetUsers"("Id") ON DELETE CASCADE;


--
-- TOC entry 3321 (class 2606 OID 16558)
-- Name: AspNetUserRoles FK_AspNetUserRoles_AspNetRoles_RoleId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserRoles"
    ADD CONSTRAINT "FK_AspNetUserRoles_AspNetRoles_RoleId" FOREIGN KEY
("RoleId") REFERENCES public."AspNetRoles"("Id") ON DELETE CASCADE;


--
-- TOC entry 3322 (class 2606 OID 16563)
-- Name: AspNetUserRoles FK_AspNetUserRoles_AspNetUsers_UserId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserRoles"
    ADD CONSTRAINT "FK_AspNetUserRoles_AspNetUsers_UserId" FOREIGN KEY
("UserId") REFERENCES public."AspNetUsers"("Id") ON DELETE CASCADE;


--
-- TOC entry 3323 (class 2606 OID 16575)
-- Name: AspNetUserTokens FK_AspNetUserTokens_AspNetUsers_UserId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."AspNetUserTokens"
    ADD CONSTRAINT "FK_AspNetUserTokens_AspNetUsers_UserId" FOREIGN KEY
("UserId") REFERENCES public."AspNetUsers"("Id") ON DELETE CASCADE;


--
-- TOC entry 3332 (class 2606 OID 24860)
-- Name: DorkFiles FK_DorkFiles_Scans_ScanId; Type: FK CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."DorkFiles"
    ADD CONSTRAINT "FK_DorkFiles_Scans_ScanId" FOREIGN KEY ("ScanId")
REFERENCES public."Scans"("Id");


--
-- TOC entry 3331 (class 2606 OID 16673)
-- Name: HbHttpHeaderLocs FK_HbHttpHeaderLocs_HbHttpHeaders_HbHttpHeaderId;
Type: FK CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."HbHttpHeaderLocs"
    ADD CONSTRAINT "FK_HbHttpHeaderLocs_HbHttpHeaders_HbHttpHeaderId" FOREIGN
KEY ("HbHttpHeaderId") REFERENCES public."HbHttpHeaders"("Id");
```

```
--
-- TOC entry 3330 (class 2606 OID 24800)
-- Name: HttpHeaders FK_HttpHeaders_Scans_ScanId; Type: FK CONSTRAINT;
Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."HttpHeaders"
    ADD CONSTRAINT "FK_HttpHeaders_Scans_ScanId" FOREIGN KEY ("ScanId")
REFERENCES public."Scans"("Id");


--
-- TOC entry 3333 (class 2606 OID 24833)
-- Name: Metadata FK_Metadata_DorkFiles_DorkFileId; Type: FK CONSTRAINT;
Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."Metadata"
    ADD CONSTRAINT "FK_Metadata_DorkFiles_DorkFileId" FOREIGN KEY
("DorkFileId") REFERENCES public."DorkFiles"("Id");


--
-- TOC entry 3326 (class 2606 OID 24805)
-- Name: Scans FK_Scans_Domains_DomainId; Type: FK CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."Scans"
    ADD CONSTRAINT "FK_Scans_Domains_DomainId" FOREIGN KEY ("DomainId")
REFERENCES public."Domains"("Id");


--
-- TOC entry 3327 (class 2606 OID 16621)
-- Name: Services FK_Services_Scans_ScanId; Type: FK CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."Services"
    ADD CONSTRAINT "FK_Services_Scans_ScanId" FOREIGN KEY ("ScanId")
REFERENCES public."Scans"("Id");


--
-- TOC entry 3329 (class 2606 OID 24812)
-- Name: SslInfos FK_SslInfos_Scans_ScanId; Type: FK CONSTRAINT; Schema:
public; Owner: postgres
--

ALTER TABLE ONLY public."SslInfos"
    ADD CONSTRAINT "FK_SslInfos_Scans_ScanId" FOREIGN KEY ("ScanId")
REFERENCES public."Scans"("Id") ON DELETE CASCADE;


--
-- TOC entry 3328 (class 2606 OID 16634)
-- Name: Vulnerabilities FK_Vulnerabilities_Services_ServiceId; Type: FK
CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."Vulnerabilities"
```

```
    ADD CONSTRAINT "FK_Vulnerabilities_Services_ServiceId" FOREIGN KEY
("ServiceId") REFERENCES public."Services"("Id");


--
-- TOC entry 3334 (class 2606 OID 24853)
-- Name: hbFileMetadataLocs
FK_hbFileMetadataLocs_HbFileMetadata_HbFileMetadataId; Type: FK CONSTRAINT;
Schema: public; Owner: postgres
--

ALTER TABLE ONLY public."hbFileMetadataLocs"
    ADD CONSTRAINT "FK_hbFileMetadataLocs_HbFileMetadata_HbFileMetadataId"
FOREIGN KEY ("HbFileMetadataId") REFERENCES public."HbFileMetadata"("Id");


-- Completed on 2022-05-03 13:26:06

--
```

Лістинг програмного коду розроблюваної інформаційної системи

```
using System;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Localization;
using MonitorServer.Models;
using MonitorServer.Models.HostDomain;
using MonitorServer.DAL.Models.HostDomain;
using HtmlAgilityPack;
using Whois;

namespace MonitorServer.Services.HostDomain
{
   public class DomainVerification: IDomainVerification
   {
      private readonly IHttpClientFactory _httpClientFactory;
      private readonly ILogger<DomainVerification> _logger;
      private readonly IStringLocalizer<MessageResources> _messages;

      public DomainVerification(IHttpClientFactory httpClientFactory,
                        ILogger<DomainVerification> logger,
                        IStringLocalizer<MessageResources> messages)
      {
         _httpClientFactory = httpClientFactory;
         _logger = logger;
         _messages = messages;
      }

      public async Task<DomainVerificationResult> Verify(string url, string userEmail, DomainVerificationType
verificationType)
      {
         DomainVerificationResult result = null;

         switch (verificationType)
         {
            case DomainVerificationType.metaTag:
               result = await CheckMetaTag(url, userEmail);
               break;
            case DomainVerificationType.domainInfo:
               result = await CheckRegistrationInfo(url, userEmail);
               break;
         }

         return result;
      }

      /// <summary>
      /// Check
      /// </summary>
      /// <param name="url"></param>
      /// <param name="userEmail"></param>
      /// <returns></returns>
      private async Task<DomainVerificationResult> CheckMetaTag(string url, string userEmail)
      {
         DomainVerificationResult result = new DomainVerificationResult();

         HttpClient client = _httpClientFactory.CreateClient();
         HttpResponseMessage response = await client.GetAsync(url);

         if (response.IsSuccessStatusCode)
         {
            string html = await response.Content.ReadAsStringAsync();
            if (!string.IsNullOrEmpty(html))
```

```
            {
                var doc = new HtmlDocument();
                doc.LoadHtml(html);

                HtmlNode metaNode = doc.DocumentNode.SelectSingleNode($"//meta[@name='altsec_{userEmail}']");
                if(metaNode != null)
                {
                    result.Verified = true;
                    return result;
                }
            }

            result.VerificationState = _messages["AbsentMetaTag"];
        }
        else
        {
            _logger.LogInformation("Http request error for the domain {Url}. Error code: {Code}", url,
response.StatusCode);
            result.VerificationState = String.Format(_messages["ServiceIsNotAvailable"], url);
        }

        return result;
    }


    private async Task<DomainVerificationResult> CheckRegistrationInfo(string url, string userEmail)
    {
        DomainVerificationResult result = new DomainVerificationResult();

        Uri hostUri = new Uri(url);
        WhoisLookup whoisClient = new WhoisLookup();

        try
        {
            var response = await whoisClient.LookupAsync(hostUri.Host);
            if(response.Registrant != null && !string.IsNullOrEmpty(response.Registrant.Email) &&
response.Registrant.Email == userEmail)
            {
                result.Verified = true;
                return result;
            }

            result.VerificationState = _messages["AbsentDomainInfo"];
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Whois lookup error. Domain {Domain};", hostUri.Host);
            result.VerificationState = _messages["AuthVerificationFail"];
        }

        return result;
    }
  }
}
using System;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Options;
using MonitorServer.Models;
using MonitorServer.Models.Cve;
using MonitorServer.DAL.Models.HostService;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace MonitorServer.Services.Cve
{
```

```
    public class CveApi : ICveApi
    {
        private readonly Config _config;
        private readonly IHttpClientFactory _httpClientFactory;
        private readonly ILogger<CveApi> _logger;

        public CveApi(IOptions<Config> config,
                      IHttpClientFactory httpClientFactory,
                      ILogger<CveApi> logger)
        {
            _config = config.Value;
            _httpClientFactory = httpClientFactory;
            _logger = logger;
        }

        public async Task GetVulnData(Service service)
        {
            List<Vulnerability> vulnerabilities = new List<Vulnerability>();
            HttpClient cveClient = _httpClientFactory.CreateClient();

            string query = $"{_config.CveApiUrl}?keyword={service.Name + (string.IsNullOrEmpty(service.Version) ?
string.Empty : " " + service.Version)}";
            HttpResponseMessage response = await cveClient.GetAsync(query);

            if (response.IsSuccessStatusCode)
            {
                string respString = await response.Content.ReadAsStringAsync();
                if (respString.Contains("CVE_Items"))
                {
                    dynamic jsonData = JObject.Parse(respString)["result"]["CVE_Items"];
                    List<CveItem> cveList = JsonConvert.DeserializeObject<List<CveItem>>(Convert.ToString(jsonData));

                    if(cveList.Count > 0)
                    {
                        foreach(CveItem data in cveList)
                        {
                            vulnerabilities.Add(new Vulnerability
                            {
                                Code = data.cve.metadata.id,
                                Url = data.cve.references.reference_data[0].url
                            });
                        }
                    }
                }

                _logger.LogInformation("CVE empty response. Product:{0}; Version:{1}", service.Name, service.Version);
            }
            else
            {
                _logger.LogInformation("CVE api is unavailable. Error code: {Code}", response.StatusCode);
            }

            service.Vulnerabilities = vulnerabilities;
        }
    }
}
```

```
using System;
using System.Linq;
using System.Net.Http;
using System.Threading.Tasks;
using System.Collections.Generic;
using MonitorServer.DAL.Models.HostHttpHeaders;
using MonitorServer.DAL.Models.Hb;
using MonitorServer.DAL.Stores.HostHttpHeaders;
```

```
namespace MonitorServer.Services.HostHttpHeaders
{
   public class HttpHeadersValidation: IHttpHeadersValidation
   {
      private readonly IHttpClientFactory _httpClientFactory;
      private readonly IHttpHeadersStore _headersStore;

      private readonly Dictionary<string, Func<IEnumerable<string>, bool>> _validators = new Dictionary<string,
Func<IEnumerable<string>, bool>>()
      {
         { "X-Frame-Options", new Func<IEnumerable<string>, bool>(CheckXFrameOptions) },
         { "X-Content-Type-Options", new Func<IEnumerable<string>, bool>(CheckXContentTypeOptions) }
      };

      public HttpHeadersValidation(IHttpClientFactory httpClientFactory,
                                   IHttpHeadersStore headersStore)
      {
         _httpClientFactory = httpClientFactory;
         _headersStore = headersStore;
      }

      public async Task<List<HttpHeader>> ValidateHeaders(string url)
      {
         List<HttpHeader> result = new List<HttpHeader>();
         List<HbHttpHeader> headersToCheck = await _headersStore.GetHb();

         HttpClient client = _httpClientFactory.CreateClient();
         HttpResponseMessage response = await client.GetAsync(url);

         foreach(var item in headersToCheck)
         {
            HttpHeader header = new HttpHeader
            {
               Name = item.Name
            };

            response.Headers.TryGetValues(item.Name, out IEnumerable<string> values);
            if(values != null && values.Any())
            {
               header.Value = string.Join(";", values);
               bool valid = _validators[item.Name](values);

               header.State = valid ? HttpHeaderState.Ok : HttpHeaderState.ValidationFailed;
            }
            else
            {
               header.State = HttpHeaderState.Missed;
            }

            result.Add(header);
         }

         return result;
      }

      private static bool CheckXFrameOptions(IEnumerable<string> values)
      {
         bool valid = false;
         if(values.Count() > 0)
         {
            foreach(string value in values)
            {
               if(value == "DENY" || value == "SAMEORIGIN" || value.Contains("ALLOW-FROM"))
               {
                  valid = true;
                  break;
               }
```

```
            }
        }

        return valid;

    }

    private static bool CheckXContentTypeOptions(IEnumerable<string> values)
    {
        bool valid = false;
        if (values.Count() > 0 && values.Contains("nosniff"))
        {
            valid = true;
        }

        return valid;
    }
  }
}

using System;
using System.Collections.Generic;
using Microsoft.Extensions.Options;
using MonitorServer.Models;
using MonitorServer.DAL.Models.HostService;
using MonitorServer.NMap.Schema;
using MonitorServer.NMap.Schema.Models;
using MonitorServer.NMap.Scanner;
using Service = MonitorServer.DAL.Models.HostService.Service;

namespace MonitorServer.Services.Nmap
{
  public class NmapScanning: INmapScanning
  {
    private readonly List<int> Ports = new List<int> { 21, 22, 25, 53, 465, 80, 443 };

    private readonly List<KeyValuePair<NmapFlag, string>> Options = new List<KeyValuePair<NmapFlag, string>>
    {
        new KeyValuePair<NmapFlag, string>(NmapFlag.Privileged, string.Empty)
    };

    private readonly Dictionary<string, ServiceState> ServiceStateMap = new Dictionary<string, ServiceState>
    {
        { "open", ServiceState.Open },
        { "closed", ServiceState.Closed },
        { "filtered", ServiceState.Filtered },
        { "unfiltered", ServiceState.Unfiltered },
        { "open|filtered", ServiceState.OpenFiltered },
        { "closed|filtered", ServiceState.ClosedFiltered },
    };

    private readonly Config _config;

    public NmapScanning(IOptions<Config> config)
    {
        _config = config.Value;
    }

    public List<Service> ScanSingleHost(string url)
    {
        List<Service> services = new List<Service>();

        Uri uri = new Uri(url);
        Target target = new Target(uri.Host);

        Scanner scanner = new Scanner(_config.NmapPath, target);
        scanner.Options.AddAll(Options);
```

```
            NMapResult result = scanner.PortScan(ScanType.Default, Ports);
            List<Port> ports = result.Hosts[0].Ports[0].Port;

            foreach(Port port in ports)
            {
                string portState = port.State?.StateProperty;
                services.Add(new Service
                {
                    Port = int.Parse(port.PortID),
                    Name = port.Service?.Product?? "",
                    Version = port.Service?.Version?? "",
                    State = ServiceStateMap.ContainsKey(portState) ? ServiceStateMap[portState] : ServiceState.Undefined
                });
            }

            return services;
        }
    }
}
using System;
using System.Net.Http;
using System.Net.Security;
using System.Threading.Tasks;
using MonitorServer.Models.Ssl;
using MonitorServer.DAL.Models.SslState;

namespace MonitorServer.Services.SslState
{
    public class SslValidation: ISslValidation
    {
        private readonly IHttpClientFactory _httpClientFactory;

        private readonly HttpClientHandler _sslValidationHandler = new HttpClientHandler
        {
            UseDefaultCredentials = true,
            ServerCertificateCustomValidationCallback = (sender, cert, chain, error) =>
            {
                if (cert != null)
                {
                    sender.Options.Set(new HttpRequestOptionsKey<string>(SslInfoKeys.EffectiveDate),
cert.GetEffectiveDateString());
                    sender.Options.Set(new HttpRequestOptionsKey<string>(SslInfoKeys.ExpirationDate),
cert.GetExpirationDateString());
                    sender.Options.Set(new HttpRequestOptionsKey<string>(SslInfoKeys.PolicyErrors), error.ToString());
                }

                return true;
            }
        };

        public SslValidation(IHttpClientFactory httpClientFactory)
        {
            _httpClientFactory = httpClientFactory;
        }

        public HttpClientHandler GetValidationHandler()
        {
            return _sslValidationHandler;
        }

        public async Task<SslInfo> Validate(string url)
        {
            HttpClient httpClient = _httpClientFactory.CreateClient("SslValidation");

            HttpResponseMessage resp = await httpClient.GetAsync(url);
            SslInfo sslInfo = GetSslInfo(resp.RequestMessage);

            return sslInfo;
```

```
        }

        private SslInfo GetSslInfo(HttpRequestMessage message)
        {
            message.Options.TryGetValue(new HttpRequestOptionsKey<string>(SslInfoKeys.EffectiveDate), out string
effectiveDateString);
            message.Options.TryGetValue(new HttpRequestOptionsKey<string>(SslInfoKeys.ExpirationDate), out string
expirationDateString);
            message.Options.TryGetValue(new HttpRequestOptionsKey<string>(SslInfoKeys.PolicyErrors), out string
policyErrorsString);

            return new SslInfo
            {
                EffectiveDate = !string.IsNullOrEmpty(effectiveDateString)?
DateTime.Parse(effectiveDateString).ToUniversalTime(): DateTime.MinValue,
                ExpirationDate = !string.IsNullOrEmpty(expirationDateString)?
DateTime.Parse(expirationDateString).ToUniversalTime(): DateTime.MinValue,
                SslPolicyError = !string.IsNullOrEmpty(policyErrorsString)?
(SslPolicyErrors)Enum.Parse(typeof(SslPolicyErrors), policyErrorsString):
SslPolicyErrors.RemoteCertificateNotAvailable
            };
        }
    }
}
using System;
using System.Linq;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using MonitorServer.Services.Nmap;
using MonitorServer.Services.Cve;
using MonitorServer.Services.SslState;
using MonitorServer.Services.HostHttpHeaders;
using MonitorServer.DAL.Models.HostScan;
using MonitorServer.DAL.Models.HostDomain;
using MonitorServer.DAL.Models.HostService;
using MonitorServer.DAL.Models.Identity;

namespace MonitorServer.Services.Monitoring
{
    public class MonitoringPipeline: IMonitoringPipeline
    {
        private readonly INmapScanning _nmapScanning;
        private readonly ICveApi _cveApi;
        private readonly ISslValidation _sslValidation;
        private readonly IHttpHeadersValidation _httpHeadersValidation;
        private readonly ILogger<MonitoringPipeline> _logger;
        private readonly ApplicationContext _db;

        public MonitoringPipeline(INmapScanning nmapScanning,
                                  ICveApi cveApi,
                                  ISslValidation sslValidation,
                                  IHttpHeadersValidation httpHeadersValidation,
                                  ILogger<MonitoringPipeline> logger,
                                  ApplicationContext db)
        {
            _nmapScanning = nmapScanning;
            _cveApi = cveApi;
            _sslValidation = sslValidation;
            _httpHeadersValidation = httpHeadersValidation;
            _logger = logger;
            _db = db;
        }

        public async Task<Scan> PerformMonitoring(Domain domain)
        {
            Scan result = new Scan();
            result.TimeStamp = DateTime.UtcNow;
```

```
            try
            {
                result.Services = _nmapScanning.ScanSingleHost(domain.Url);
                await CheckVulnerabilities(result.Services.Where(x=> !string.IsNullOrEmpty(x.Name) &&
!string.IsNullOrEmpty(x.Version)));

                result.SslInfo = await _sslValidation.Validate(domain.Url);
                result.Headers = await _httpHeadersValidation.ValidateHeaders(domain.Url);
            }
            catch(Exception ex)
            {
                _logger.LogError(ex, "Monitoring error; Url: {Url};", domain.Url);
            }

            Domain domainDb = _db.Domains.FirstOrDefault();
            domainDb.Scans.Add(result);
            _db.Domains.Update(domainDb);
            _db.SaveChanges();

            return result;
        }

        private async Task CheckVulnerabilities(IEnumerable<Service> services)
        {
            int batchSize = 2;
            int numberOfBatches = (int)Math.Ceiling((double)services.Count() / batchSize);

            for (int i = 0; i < numberOfBatches; i++)
            {
                var currentServices = services.Skip(i * batchSize).Take(batchSize);
                var tasks = services.Select(s => _cveApi.GetVulnData(s));

                await Task.WhenAll(tasks);
            }
        }
    }
}
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Authorization;
using MonitorServer.Models;
using MonitorServer.Models.HostDomain;
using MonitorServer.Services.HostDomain;
using MonitorServer.Utility;
using MonitorServer.DAL.Models.HostDomain;
using MonitorServer.DAL.Models.Identity;
using MonitorServer.DAL.Stores.HostDomain;
using AutoMapper;

namespace MonitorServer.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    [Authorize]
    public class DomainController : ControllerBase
    {
        private readonly UserManager<ApplicationUser> _userManager;
        private readonly ResourcesWrapper _resources;
        private readonly ILogger<DomainController> _logger;
        private readonly IMapper _mapper;
        private readonly IDomainVerification _domainVerification;
        private readonly IHostDomainStore _domainStore;
```

```
        public DomainController(UserManager<ApplicationUser> userManager,
                                ResourcesWrapper resources,
                                ILogger<DomainController> logger,
                                IMapper mapper,
                                IDomainVerification domainVerification,
                                IHostDomainStore domainStore)
        {
            _userManager = userManager;
            _resources = resources;
            _logger = logger;
            _mapper = mapper;
            _domainVerification = domainVerification;
            _domainStore = domainStore;
        }

        [HttpGet]
        public IActionResult GetAll()
        {
            return Ok(new List<DomainDto>());
        }

        [HttpPost]
        public async Task<IActionResult> CreateDomain(DomainDto domainDto)
        {
            if (!ModelState.IsValid)
            {
                return this.InvalidParameters(_resources.Messages["IncorrectInputParameters"], ModelState.Errors());
            }

            try
            {
                Domain domain = _mapper.Map<Domain>(domainDto);
                UserContext ctx = this.CurrentUser();

                var result = await _domainVerification.Verify(domain.Url, ctx.Email, domain.VerificationType);
                if (!result.Verified)
                {
                    return this.ProcessingError(result.VerificationState);
                }

                ApplicationUser user = await _userManager.FindByIdAsync(ctx.Id);

                domain.LastVerification = DateTime.UtcNow;

                domain.Users.Add(user);

                bool success = await _domainStore.CreateAsync(domain);
                if (success)
                {
                    return this.Success();
                }
            }
            catch(Exception ex)
            {
                _logger.LogError("{Method} error: {Message}", nameof(CreateDomain), ex.InnerException);
            }

            return this.ProcessingError(_resources.Messages["ErrorWhileProcessing"]);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.AspNetCore.Mvc;
```

```
using MonitorServer.Models;
using MonitorServer.Models.Account;
using MonitorServer.Models.Response;
using MonitorServer.Services.Jwt;
using MonitorServer.Utility;
using MonitorServer.DAL.Models.Identity;

namespace MonitorServer.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class AccountController : ControllerBase
   {
      private readonly UserManager<ApplicationUser> _userManager;
      private readonly ResourcesWrapper _resources;
      private readonly IJwtService _jwtService;

      public AccountController(UserManager<ApplicationUser> userManager,
                               ResourcesWrapper resources,
                               IJwtService jwtService)
      {
         _userManager = userManager;
         _jwtService = jwtService;
         _resources = resources;
      }

      [HttpPost]
      [Route("login")]
      public async Task<IActionResult> Login([FromBody] LoginDto model)
      {
         if (!ModelState.IsValid)
         {
            return this.InvalidParameters(_resources.Messages["IncorrectInputParameters"], ModelState.Errors());
         }

         var user = await _userManager.FindByNameAsync(model.Email);
         if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
         {
            var authClaims = new List<Claim>
            {
               new Claim(ClaimTypes.NameIdentifier, user.Id),
               new Claim(ClaimTypes.Name, user.UserName),
               new Claim(ClaimTypes.Email, user.Email),
               new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
            };

            var userRoles = await _userManager.GetRolesAsync(user);

            foreach (var userRole in userRoles)
            {
               authClaims.Add(new Claim(ClaimTypes.Role, userRole));
            }

            string token = _jwtService.GenerateToken(authClaims, out DateTime expires);
            if (!string.IsNullOrEmpty(token))
            {
               var response = new LoginResponse(ApiResponseCode.OK, _resources.Messages["SuccessfullOperation"])
               {
                  Token = token,
                  ExpiredAt = expires
               };

               return Ok(response);
            }
         }

         return Unauthorized();
      }
```

```
        [HttpPost]
        [Route("register")]
        public async Task<IActionResult> Register([FromBody] RegisterDto model)
        {
            if (!ModelState.IsValid)
            {
                return this.InvalidParameters(_resources.Messages["IncorrectInputParameters"], ModelState.Errors());
            }

            var userExists = await _userManager.FindByNameAsync(model.Email);
            if (userExists != null)
            {
                return this.InvalidParameters(_resources.Messages["UserAlreadyExists"]);
            }

            ApplicationUser user = new ApplicationUser()
            {
                Email = model.Email,
                SecurityStamp = Guid.NewGuid().ToString(),
                UserName = model.Email
            };

            var result = await _userManager.CreateAsync(user, model.Password);
            if (!result.Succeeded)
            {
                var errors = new List<KeyValuePair<string, string[]>>();

                foreach (var error in result.Errors)
                {
                    errors.Add(new KeyValuePair<string, string[]>(error.Code, new string[] { error.Description }));
                }

                return this.ProcessingError(_resources.Messages["ErrorWhileProcessing"], errors);
            }

            return this.Success();
        }
    }
}
using System.Collections.Generic;
using System.Security.Claims;
using Microsoft.AspNetCore.Mvc;
using MonitorServer.Models;
using MonitorServer.Models.Response;

namespace MonitorServer.Utility
{
    public static class ControllerExtensions
    {
        public static UserContext CurrentUser(this ControllerBase controller)
        {
            var user = controller.HttpContext.User;

            UserContext userContext = new UserContext
            {
                Id = user.FindFirstValue(ClaimTypes.NameIdentifier),
                UserName = user.FindFirstValue(ClaimTypes.Name),
                Email = user.FindFirstValue(ClaimTypes.Email)
            };

            return userContext;
        }

        public static BadRequestObjectResult InvalidParameters(this ControllerBase controller, string message)
        {
            ApiResponseBase response = new ApiResponseBase(ApiResponseCode.InvalidInputParameters, message);
```

```
            return controller.BadRequest(response);
        }

        public static BadRequestObjectResult InvalidParameters(this ControllerBase controller, string message,
IEnumerable<KeyValuePair<string, string[]>> errors)
        {
            ApiResponseBase response = new ApiResponseBase(ApiResponseCode.InvalidInputParameters, message, errors);
            return controller.BadRequest(response);
        }

        public static BadRequestObjectResult ProcessingError(this ControllerBase controller, string message)
        {
            ApiResponseBase response = new ApiResponseBase(ApiResponseCode.ProcessingError, message);
            return controller.BadRequest(response);
        }

        public static BadRequestObjectResult ProcessingError(this ControllerBase controller, object data)
        {
            ApiDataResponse response = new ApiDataResponse(ApiResponseCode.ProcessingError, data);
            return controller.BadRequest(response);
        }

        public static BadRequestObjectResult ProcessingError(this ControllerBase controller, string message,
IEnumerable<KeyValuePair<string, string[]>> errors)
        {
            ApiResponseBase response = new ApiResponseBase(ApiResponseCode.ProcessingError, message, errors);
            return controller.BadRequest(response);
        }

        public static OkObjectResult Success(this ControllerBase controller)
        {
            ApiResponseBase response = new ApiResponseBase(ApiResponseCode.OK);
            return controller.Ok(response);
        }

        public static OkObjectResult Success(this ControllerBase controller, object data)
        {
            ApiDataResponse response = new ApiDataResponse(ApiResponseCode.OK, data);
            return controller.Ok(response);
        }
    }
}
using System.Text;
using System.Globalization;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Localization;
using Microsoft.IdentityModel.Tokens;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.EntityFrameworkCore;
using MonitorServer.Models;
using MonitorServer.Services.Extensions;
using MonitorServer.DAL.Models.Identity;
using Newtonsoft.Json;

namespace MonitorServer
{
    public class Startup
    {
        private readonly IConfiguration _config;

        public Startup(IConfiguration config)
```

```
{
    _config = config;
}

public void ConfigureServices(IServiceCollection services)
{
    services.Configure<CookiePolicyOptions>(options =>
    {
        options.MinimumSameSitePolicy = SameSiteMode.Lax;
    });

    services.AddLocalization(options => options.ResourcesPath = "Resources");

    services.AddControllers()
            .ConfigureApiBehaviorOptions(options => options.SuppressModelStateInvalidFilter = true)
            .AddDataAnnotationsLocalization(options =>
            {
                options.DataAnnotationLocalizerProvider = (type, factory) => factory.Create(typeof(GeneralResources));
            });

    services.AddHttp();

    services.Configure<RequestLocalizationOptions>(options =>
    {
        var supportedCultures = new[]
        {
            new CultureInfo("uk-UA")
        };

        options.DefaultRequestCulture = new RequestCulture("uk-UA");
        options.SupportedCultures = supportedCultures;
        options.SupportedUICultures = supportedCultures;
    });


    services.AddDbContext<ApplicationContext>(options =>
    {
        options.UseNpgsql(_config.GetConnectionString("MonitorServer"));
    });

    services.AddIdentity<ApplicationUser, IdentityRole>()
            .AddEntityFrameworkStores<ApplicationContext>()
            .AddDefaultTokenProviders();

    services.AddAuthentication(options =>
    {
        options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
        options.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
    })
    .AddJwtBearer(options =>
    {
        options.SaveToken = true;
        options.RequireHttpsMetadata = true;
        options.TokenValidationParameters = new TokenValidationParameters()
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            RequireExpirationTime = true,
            ValidAudience = _config["AuthToken:Audience"],
            ValidIssuer = _config["AuthToken:Issuer"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["AuthToken:Secret"]))
        };
    });

    services.AddApp(_config);
```

```
            services.AddAutoMapper(typeof(Startup));
        }

        public void Configure(IApplicationBuilder app,
                              IWebHostEnvironment env,
                              ILoggerFactory log)
        {
            log.AddFile("logs/app-{Date}.txt");

            var supportedCultures = new[]
            {
                new CultureInfo("uk-UA")
            };

            app.UseRequestLocalization(new RequestLocalizationOptions
            {
                DefaultRequestCulture = new RequestCulture("uk-UA"),
                SupportedCultures = supportedCultures,
                SupportedUICultures = supportedCultures,
            });

            //app.UseStatusCodePagesWithReExecute("/error/{0}");

            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseStaticFiles();
            }
            else
            {
                app.UseHsts();
            }

            app.UseCookiePolicy();
            app.UseXContentTypeOptions();
            app.UseReferrerPolicy(options => options.NoReferrer());

            app.UseRouting();

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```