Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2025 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності”

Виконав здобувач вищої освіти
IV курсу, групи КБ-22-МБ
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Сучков І.С.
«____» _____ 2025 р.

Керівник проекту
доктор філософії (PhD)
_____ Дрєєва Г.М.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2025 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Сучкову Іллі Сергійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності*

2. Керівник роботи *Дрєєва Ганна Миколаївна, доктор філософії (PhD)*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 51-02 від 17.01.2025 року

3. Строк подання студентом роботи до захисту *23.05.2025 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2025 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2025 р.	
3.	Розробка моделі компонента	20.03.2025 р.	
4.	Розробка структур даних	25.03.2025 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2025 р.	
6.	Програмування алгоритмів	10.04.2025 р.	
7.	Оформлення ПЗ	17.04.2025 р.	
8.	Попередній захист роботи	23.05.2025 р.	

Дата видачі завдання
« 17 » січня 2025 р.

Підпис керівника
_____ Дрєєва Г.М.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2025 р.

Підпис здобувача
_____ Сучков І.С.
(прізвище та ініціали)

АНОТАЦІЯ

Сучков І.С. Програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

Метою розробки є програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

Результат роботи – програмна реалізація системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ з ОС Windows 10/11.

Програму розроблено в середовищі Python.

Ключові слова: кібербезпека, штучний інтелект, оцінка кіберзахищеності

ABSTRACT

Suchkov I.S. Software for the artificial intelligence cybersecurity system for assessing cybersecurity. 125 Cybersecurity. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the artificial intelligence cybersecurity system for assessing cybersecurity.

The purpose of the development is the software for the artificial intelligence cybersecurity system for assessing cybersecurity.

The result of the work is the software implementation of the artificial intelligence cybersecurity system for assessing cybersecurity.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software are provided.

The program can be used on PCs with Windows 10/11.

The program is developed in the Python environment.

Keywords: cybersecurity, artificial intelligence, cybersecurity assessment

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ДВЧ	–	датчик випадкових чисел
ДСТ 28147-89	–	алгоритм шифрування
ЕОМ	–	електронна обчислювальна машина
ІС	–	інформаційна система
ОС	–	обчислювальна система
ПВЧ	–	псевдовипадкові числа
ПЗ	–	програмне забезпечення
РПЗ	–	руйнуючі програмні засоби
СЗІ	–	система захисту інформації
ASCII	–	система кодування
DES	–	алгоритм шифрування
FEAL	–	алгоритм шифрування
IDEA	–	алгоритм шифрування
KOI-8	–	система кодування
RISC	–	архітектура процесора

ВСТУП

Актуальність теми. Стандарт СТБ ІСО/МЕК 27002-2012 «Інформаційні технології. Методи забезпечення безпеки. Кодекс практики для менеджменту інформаційної безпеки», прийнятий в 2012, є основою для розробки політик, правил і процедур з інформаційної безпеки (ІБ) для створення, реалізації, підтримки й керування захистом інформації в організації, і містить докладний опис вимог до засобів керування, залишаючи за фахівцями організацій способи технічної реалізації описаних вимог.

Інститут SANS із залученням фахівців і експертів урядових і неурядових служб, що займаються аналізом, виявленням, захистом і боротьбою з кіберзлочинами запропонував набір з «20 критичних засобів керування безпекою для ефективної кіберзахисту». За основу був узятий документ NIST 800-53 «Рекомендуємі засоби керування безпекою для державних інформаційних систем», що є базисним набором при керуванні безпекою організації в технічній і організаційній областях. У результаті були розроблені засоби керування, які, на думку експертів, призначені для захисту як від відомих на теперішній час атак, так і типів атак, очікуваних до появи в найближчому майбутньому. Документ являє собою покроковий практичний посібник із впровадження технічних засобів керування інформаційною безпекою (ІБ) і дозволяє планомірно підвищувати рівень ІБ організації від базового (захист від найпоширеніших погроз) до зрілого рівня (застосування просунутих, високотехнологічних рішень).

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем штучного інтелекту для оцінки кіберзахищеності.

– Дослідження системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

– Програмна реалізація системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі штучного інтелекту для оцінки кіберзахищеності.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Основні принципи, покладені в основу пропонованих засобів керування безпекою:

– застосування засобів керування ІБ, спрямованих на зниження ризиків, пов'язаних з найпоширенішими атаками;

– повний охват інформаційного середовища організації сумісними один з одним засобами керування ІБ;

– максимально можлива автоматизація засобів безпеки і їхній автоматичний контроль для достовірної, масштабованої й постійної оцінки реалізації засобів керування;

– аналіз інцидентів ІБ і причин порушень режиму ІБ для своєчасного виявлення й запобігання атак;

– використання метрик і ключових показників для оцінки ефективності засобів керування ІБ і для взаємодії технічних фахівців, фахівців з інформаційної безпеки, аудиторів і керівників по безпеці в питаннях оцінки ризиків і стану ІБ в організації.

Документ SANS призначений як для технічних фахівців, так і для керівників з ІБ (CISO) і IT (CIO) організацій, а також аудиторів. Технічним фахівцям із захисту інформації й системному адмініструванню пропонуються процедури, спрямовані на захист локальної мережі організації від існуючих і передбачуваних видів кібератак, метрики із вказівкою конкретних строків і цілей найбільш важливих елементів процедур, і опис, які образом необхідно оцінювати досягнення кожної контрольної метрики.

Менеджери по інформаційній безпеці й інформаційних технологіях можуть використовувати запропоновані показники оцінки для послідовного й об'єктивного контролю ефективності застосування засобів керування ІБ і оцінювати поточний стан і поліпшення безпеки із часом.

Аналіз рекомендацій, розроблених ISO й SANS, показує, що багато хто з вимог стандарту ISO 27002-2012, що ставляться до безпеки інформаційних систем, можуть бути реалізовані шляхом застосування засобів керування SANS.

1.2 Область застосування

Захист від кібератак

Для захисту від кібератак мережі й системи, так само як і обслуговуючий персонал, повинні бути надійно захищені від різних зовнішніх і внутрішніх погроз. Для реагування на успішні атаки засобу захисту повинні **виявляти** й надалі **перешкоджати** атакам на внутрішні локальні мережі. Критичним компонентом таких систем безпеки є постійний **моніторинг** – здатність автоматично тестувати і підтверджувати, що використовувані засоби безпеки працюють і завчасно усувають можливі уразливості.

Основний принцип кіберзахисту – «Порушення служить для зміцнення захисту». Знання про атаки, зроблених на зламані системи, є основою для побудови ефективного захисту надалі.

Для моніторингу, виявлення, захисту, звіту й реагування на відомі уразливості, атаки й спроби проникнення, а також постійного тестування й оцінки функціонування методів і засобів інформаційної безпеки було ухвалене рішення створити основний механізм по забезпеченню інформаційної безпеки, який можна було б постійно автоматично контролювати.

У майбутньому така ініціатива повинна підсилити взаємозв'язок організацій і підсилити захист інформації в державних установах і комерційних організаціях з розподіленою структурою. Тунелювання між мережами, хмарна обробка даних – все це збільшує кількість уразливостей безпеки інформації.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Огляд різних методів захисту інформації

Причини, що впливають на розвиток в області захисту інформації

Зміст проблеми захисту інформації фахівцями інтерпретуються в такий спосіб. У міру розвитку й ускладнення засобів, методів і форм автоматизації процесів обробки інформації підвищується її уразливість. Основними факторами, що сприяють підвищенню цієї уразливості, є:

1. Різке збільшення обсягів інформації, що накопичується, збереженої й оброблюваної за допомогою ЕОМ і інших засобів автоматизації.

2. Зосередження в єдиних базах дані інформації різного призначення й різних приналежностей.

3. Різке розширення кола користувачів, що мають безпосередній доступ до ресурсів обчислювальної системи й, що перебуває в ній даних.

4. Ускладнення режимів функціонування технічних засобів обчислювальних систем: широке впровадження багатопрограмного режиму, а також режимів поділу часу й реального часу.

5. Автоматизація міжмашиного обміну інформацією, у тому числі й на великих відстанях.

У цих умовах виникає уразливість двох видів: з одного боку, можливість знищення або перекручування інформації (тобто порушення її фізичної цілісності), а з іншого боку – можливість несанкціонованого використання інформації (тобто небезпека витоку інформації обмеженого користування). Другий вид уразливості викликає особливу заклопотаність користувачів ЕОМ.

Основними потенційно можливими каналами витоку інформації є:

1. Пряме розкрадання носіїв і документів.

2. Запам'ятовування або копіювання інформації.

3. Несанкціоноване підключення до апаратури й ліній зв'язку або незаконне використання "законної" (тобто зареєстрованої) апаратури системи (найчастіше терміналів користувачів).

4. Несанкціонований доступ до інформації за рахунок спеціального пристосування математичного й програмного забезпечення.

Методи захисту інформації

Можна виділити три напрямки робіт із захисту інформації: теоретичні дослідження, розробка засобів захисту й обґрунтування способів використання засобів захисту в автоматизованих системах.

У теоретичному плані основна увага приділяється дослідженню уразливості інформації в системах електронної обробки інформації, явищу й аналізу каналів витоку інформації, обґрунтуванню принципів захисту інформації в більших автоматизованих системах і розробці методик оцінки надійності захисту.

До теперішнього часу розроблено багато різних засобів, методів, мір і заходів, призначених для захисту інформації, що накопичується, збереженої й оброблюваної в автоматизованих системах. Сюди входять апаратні й програмні засоби, криптографічне закриття інформації, фізичні міри організовані заходи, законодавчі міри.

Іноді всі ці засоби захисту діляться на технічні й нетехнічні, причому, до технічного відносять апаратні й програмні засоби й криптографічне закриття інформації, а до нетехнічних – інші перераховані вище.

а) апаратні методи захисту. До апаратних засобів захисту ставляться різні електронні, електронно-механічні, електронно-оптичні пристрої. До теперішнього часу розроблене значне число апаратних засобів різного призначення, однак найбільше поширення одержують наступні:

– спеціальні регістри для зберігання реквізитів захисту: паролів, що ідентифікують кодів, грифів або рівнів таємності;

– генератори кодів, призначені для автоматичного генерування ідентифікуючого коду пристрою;

– пристрою виміру індивідуальних характеристик людини (голосу, відбитків) з метою його ідентифікації;

– спеціальні біти таємності, значення яких визначає рівень таємності інформації, збереженої в запам’ятовуючих пристроях (ЗП), що належать дані біти;

– схеми переривання передачі інформації в лінії зв'язку з метою періодичної перевірки адреси видачі даних.

Особливу й групу, що одержує найбільше поширення, апаратних засобів захисту становлять пристрої для шифрування інформації (криптографічні методи).

б) програмні методи захисту. До програмних засобів захисту ставляться спеціальні програми, які призначені для виконання функцій захисту й включаються до складу програмного забезпечення систем обробки даних. Програмний захист є найпоширенішим видом захисту, чому сприяють такі позитивні властивості даного засобу, як універсальність, гнучкість, простота реалізації, практично необмежені можливості зміни й розвитку й т.п.

По функціональному призначенню їх можна розділити на наступні групи:

– ідентифікація технічних засобів (терміналів, пристроїв групового керування уведенням-виводом, ЕОМ, носіїв інформації), завдань і користувачів;

– визначення прав технічних засобів (дні й час роботи, дозволені до використання завдання) і користувачів;

– контроль роботи технічних засобів і користувачів;

– реєстрація роботи технічних засобів і користувачів при обробки інформації обмеженого використання;

– знищення інформації в ЗП після використання;

– сигналізації при несанкціонованих діях;

– допоміжні програми різного призначення: контролю роботи механізму захисту, надання грифа таємності на видаваних документах.

в) резервне копіювання. Резервне копіювання інформації полягає в зберіганні копії програм на носії: стримері, гнучких носіях, оптичних дисках, жорстких дисках. На цих носіях копії програм можуть перебуває в нормальному (незжатому) або заархівіруваному виді. Резервне копіювання проводиться для збереження програм від ушкоджень (як навмисних, так і випадкових), і для зберігання рідко використовуваних файлів.

При сучасному розвитку комп'ютерних технологій вимоги до запам'ятовувальних пристроїв у локальній мережі ростуть набагато швидше, ніж можливості. Разом з геометричним ростом ємності дискових підсистем програмам копіювання на магнітну стрічку за час, відпущений на резервування, доводиться читати й записувати все більші масиви даних. Ще більш важливо, що програми резервування повинні навчитися в такий спосіб управляти більшою кількістю файлів, щоб користувачам не було надто складно витягати окремі файли.

Більшість найбільш популярних сучасних програм резервування надають, у тім або іншому виді, базу даних про зарезервовані файли й деяку інформацію про те, на якій стрічці перебувають останні зарезервовані копії. Набагато рідше зустрічається можливість інтеграції (або принаймні співіснування) з технологією структурованого, або ієрархічного зберігання інформації (HSM, Hierarchical Storage Management).

HSM допомагає збільшити ємність доступного простору жорсткого диска на сервері за рахунок переміщення статичних файлів (до яких останнім часом не зверталися) на менш дорогі альтернативні запам'ятовувальні пристрої, такі як оптичні накопичувачі або накопичувачі на магнітній стрічці. HSM залишає на жорсткому диску фіктивний файл нульової довжини, що повідомляє про те, що реальний файл перенесений. У такому випадку, якщо користувачеві буде

потрібно попередня версія файлу, то програмне забезпечення HSM зможе швидко витягти його з магнітної стрічки або з оптичного накопичувача.

г) криптографічне шифрування інформації. Криптографічне закриття (шифрування) інформації полягає в такому перетворенні інформації, яка захищається,, при якому по зовнішньому вигляді не можна визначити зміст закритих даних. Криптографічному захисту фахівці приділяють особливу увагу, уважаючи її найбільш надійної, а для інформації, переданої по лінії зв'язку великої довжини, – єдиним засобом захисту інформації від розкрадань.

Основні напрямки робіт з розглянутого аспекту захисту можна сформулювати в такий спосіб:

– вибір раціональних систем шифрування для надійного закриття інформації;

– обґрунтування шляхів реалізації систем шифрування в автоматизованих системах;

– розробка правил використання криптографічних методів захисту в процесі функціонування автоматизованих систем;

– оцінка ефективності криптографічного захисту.

До шифрів, призначеним для закриття інформації в ЕОМ і автоматизованих системах, пред'являється ряд вимог, у тому числі: достатня стійкість (надійність закриття), простота шифрування й розшифрування від способу внутримашинного подання інформації, нечутливість до невеликих помилок шифрування, можливість внутримашинної обробки зашифрованої інформації, незначна надмірність інформації за рахунок шифрування й ряд інших. У тім або іншому ступені цим вимогам відповідають деякі види шифрів заміни, перестановки, гаммування, а також шифри, засновані на аналітичних перетвореннях шифруємих даних.

Шифрування заміною (іноді вживається термін "підстановка") полягає в тому, що символи шифруємого тексту заміняються символами іншого або того ж алфавіту відповідно до заздалегідь обумовленої схеми заміни.

Шифрування перестановкою полягає в тому, що символи шифруємого тексту переставляються по якомусь правилу в межах якогось блоку цього тексту. При достатній довжині блоку, у межах якого здійснюється перестановка, і складному й неповторюваному порядку перестановці можна досягти достатньої для практичних додатків в автоматизованих системах стійкості шифрування.

Шифрування гаммуванням полягає в тому, що символи шифруємого тексту складаються із символами деякої випадкової послідовності, іменованою гамою. Стійкість шифрування визначається головним чином розміром (довжиною) неповторюваної частини гами. Оскільки за допомогою ЕОМ можна генерувати практично нескінченну гаму, те даний спосіб уважається одним з основних для шифрування інформації в автоматизованих системах. Правда, при цьому виникає ряд організаційно-технічних труднощів, які, однак, не є не переборними.

Шифрування аналітичним перетворенням полягає в тому, що шифруємий текст перетвориться по деякому аналітичному правилу (формулі). Можна, наприклад, використовувати правило множення матриці на вектор, причому множена матриця є ключем шифрування (тому її розмір і зміст повинні збережуться в таємниці), а символи множеного вектора послідовно служать символи шифруємого тексту.

Особливо ефективними є комбіновані шифри, коли текст послідовно шифрується двома або більшим числом систем шифрування (наприклад, заміна й гаммування, перестановка й гаммування). Уважається, що при цьому стійкість шифрування перевищує сумарну стійкість у складених шифрах.

Кожну з розглянутих систем шифрування можна реалізувати в автоматизованій системі або програмним шляхом, або за допомогою спеціальної апаратури. Програмна реалізація в порівнянні з апаратної є більше гнучкою й обходиться дешевше. Однак апаратне шифрування в загальному випадку в кілька разів продуктивніше. Це обставина при більших обсягах інформації, що закривається, має вирішальне значення.

д) фізичні міри захисту. Наступним класом в арсеналі засобів захисту інформації є фізичні міри. Це різні пристрої й спорудження, а також заходу, які утрудняють або унеможливлюють проникнення потенційних порушників у місця, у яких можна мати доступ до інформації, яка захищається. Найчастіше застосовуються такі міри:

– фізична ізоляція споруджень, у яких установлюється апаратура автоматизованої системи, від інших споруджень;

– огородження території обчислювальних центрів заборами на таких відстанях, які достатні для виключення ефективної реєстрації електромагнітних випромінювань, і організації систематичного контролю цих територій;

– організація контрольно-пропускних пунктів у входів у приміщення обчислювальних центрів або обладнаних вхідних дверей спеціальними замками, що дозволяють регулювати доступ у приміщення;

– організація системи охоронної сигналізація.

е) організаційні заходи щодо захисту інформації. Наступним класом мер захисту інформації є організаційні заходи. Це такі нормативно-правові акти, які регламентують процеси функціонування системи обробки даних, використання її пристроїв і ресурсів, а також взаємовідношення користувачів і систем таким чином, що несанкціонований доступ до інформації стає неможливим або істотно утрудняється. Організаційні заходи відіграють більшу роль у створенні надійного механізму захисту інформації. Причини, по яких організаційні заходи відіграють підвищену роль у механізмі захисту, полягає в тому, що можливості несанкціонованого використання інформації значною мірою обумовлюють нетехнічними аспектами: злочинними діями, недбайливістю або недбалістю користувачів або персоналу систем обробки даних. Вплив цих аспектів практично неможливо уникнути або локалізувати за допомогою вище розглянутих апаратних і програмних засобів, криптографічного закриття інформації й фізичних мір захисту. Для цього необхідна сукупність організаційних, організаційно-технічних і організаційно-правових заходів, що виключала б можливість виникнення небезпеки витоку інформації подібним чином.

Основними заходами в такій сукупності є наступні:

– заходи, здійснювані при проектуванні, будівництві й устаткуванні обчислювальних центрів (ОЦ);

– заходи, здійснювані при підборі й підготовки персоналу ОЦ (перевірка прийнятих на роботу, створення умов при яких персонал не хотів би втратитися роботи, ознайомлення з мірами відповідальності за порушення правил захисту);

– організація надійного пропускного режиму;

– організація зберігання й використання документів і носіїв: визначення правил видачі, ведення журналів видачі й використання;

– контроль внесення змін у математичне й програмне забезпечення;

– організація підготовки й контролю роботи користувачів;

Один з найважливіших організаційних заходів – зміст в ОЦ спеціальної штатної служби захисту інформації, чисельність і состав якої забезпечували б створення надійної системи захисту й регулярне її функціонування.

Висновок

Основні виводи про способи використання розглянутих вище засобів, методів і заходів захисту, зводиться до наступного:

1. Найбільший ефект досягається тоді, коли всі використовувані засоби, методи й заходи поєднуються в єдиний, цілісний механізм захисту інформації.

2. Механізм захисту повинен проектуватися паралельно зі створенням систем обробки даних, починаючи з моменту виробітку загального задуму побудови системи.

3. Функціонування механізму захисту повинне плануватися й забезпечуватися поряд із плануванням і забезпеченням основних процесів автоматизованої обробки інформації.

4. Необхідно здійснювати постійний контроль функціонування механізму захисту.

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Python – динамічна інтерпретована об'єктно-орієнтована скриптова мова програмування із строгою динамічною типізацією. Офіційний сайт мови програмування Python https://www.python.org/. Python – багатоцільова мова програмування, яка дозволяє писати код, що добре читається. Відносний лаконізм мови Python дозволяє створити програму, яка буде набагато коротше свого аналога, написаного на іншій мові. Python – багатоплатформова мова програмування. Це означає, що програми на Python можна запускати в різних операційних системах без будь-яких змін.

Ще однією перевагою Python є його стандартна бібліотека, яка встановлюється разом з Python і містить готові інструменти для роботи з операційною системою, веб-сторінками, базами даних, різними форматами даних, для побудови графічного інтерфейсу програм тощо. Програми, написані на мові програмування Python, можуть бути як невеликими скриптами, так і складними системами. Python абсолютно безкоштовний.

Швидкість виконання коду Python

Один з можливих недоліків Python – швидкість виконання коду. Python не є компільованою мовою. Код на Python спочатку компілюється у внутрішній байт-код, який потім виконується інтерпретатором Python. У більшості випадків при використанні Python виходять програми повільніші в порівнянні з такими мовами, як C.

Втім, сучасні комп'ютери мають таку обчислювальну потужність, що для більшості застосунків швидкість розробки важливіша швидкості виконання, а програми на Python зазвичай пишуться набагато швидше.

Окрім того, Python легко розширюється модулями, написаними на C або C++. Такі модулі можуть використовуватися для виконання частин програми, що створюють інтенсивне навантаження на процесор.

Використання Python

Python використовується для різних цілей: для створення ігор і веб-застосунків, розробки внутрішніх інструментів для різноманітних проектів. Мова також широко застосовується в науковій області для досліджень і розв'язування прикладних завдань.

Застосування мови програмування Python:

1. BitTorrent – протокол для обміну даними.

2. Ubuntu Software Center – вільне програмне забезпечення для пошуку, установки і видалення пакунків в системі Ubuntu Linux.

3. Blender – програма для створення тривимірної комп'ютерної графіки, що включає засоби моделювання, анімації, вимальовування, пост-обробки відео, а також створення відеоігор.

4. GIMP – растровий графічний редактор, із підтримкою векторної графіки.

5. World of Tanks.

6. Вільна енциклопедія Вікіпедія.

7. Пошукова система Google.

8. DropBox – файловий хостинг, що включає персональне хмарне сховище, синхронізацію файлів і програму-клієнт.

9. YouTube – популярне відеосховище.

Версії Python

Мови програмування з часом змінюються – розробники додають в них нові можливості, а також виправляють помилки. Так з'являються різні версії мови. Наприклад, код написаний на Python 2 у більшості випадків не буде працювати у версії Python 3 без внесення додаткових змін.

Процесор є найважливішим компонентом в комп'ютері. Одна з основних функцій процесора – це обробка даних згідно комп'ютерної програми, яка є списком інструкцій, шляхом виконання арифметичних і логічних операцій над фрагментами даних.

Кожна інструкція в програмі – це команда, яка «повідомляє» процесору, яку операцію він повинен виконати. Процесор комп'ютера може розуміти лише ті інструкції, які написані на машинній мові. Машинна мова – це штучна мова, створена для передачі команд комп'ютеру. За допомогою машинної мови створюються ефективні програми, оскільки розробник отримує доступ до всіх можливостей процесора. Машинна мова – мова низького рівня.

Інструкція машинної мови існує для кожної операції, яку процесор здатний виконати – є інструкція для додавання чисел, є інструкція для віднімання чисел і т.д. Увесь набір інструкцій, який центральний процесор може виконати, відомий як набір інструкцій процесора.

Наприклад, у вас є певна програма, яка зберігається на диску вашого комп'ютера. Для виконання програми, ви здійснюєте подвійний клік на значку програми. Це змушує програму копіюватися з диска в оперативну пам'ять, після чого процесор комп'ютера виконує копію програми, яка знаходиться в оперативній пам'яті.

Коли процесор виконує інструкції програми, він бере участь у процесі, який є відомим як цикл fetch – decode – execute (отримати – декодувати – виконати). Цей цикл виконується для кожної інструкції у програмі і складається з трьох кроків:

Отримати

Програма – це послідовність інструкцій на машинній мові. Першим кроком циклу є завантаження (отримання) наступної інструкції з пам'яті в процесор.

Декодувати

Інструкція машинної мови – це двійкове число, яке представляє команду, що повідомляє процесору виконати певну операцію. На цьому кроці процесор декодує інструкцію, яку було «витягнуто» з пам'яті, для визначення того, яка операція повинна виконуватись.

Виконати

Останній крок циклу – виконати операцію.

Хоча процесор комп'ютера розуміє тільки машинну мову, людині непрактично писати програми на машинній мові. Така програма може мати тисячі або навіть мільйони бінарних інструкцій, і написання такої програми буде дуже обтяжливим процесом.

З цієї причини була створена мова асемблера як альтернатива машинній мові. Замість використання двійкових чисел для написання інструкцій, мова асемблера використовує короткі слова, відомі як мнемокоди.

Незважаючи на те, що мова асемблера не вимагає двійкових інструкцій, як у випадку машинної мови, проте вона вимагає високих знань про процесор. Використовуючи мову асемблера, навіть для найпростішої програми, необхідно написати велику кількість інструкцій.

Мова програмування високого рівня дозволяє створювати складні програми, не знаючи, як працює процесор, і не записуючи великої кількості інструкцій низького рівня. Крім того, більшість мов програмування високого рівня використовують слова, які легко зрозуміти.

Python – одна із популярних сучасних мов програмування високого рівня. Python – інтерпретована мова програмування. Python – це високорівнева інтерпретована мова програмування, на відміну від C++, яка є прикладом компільованої мови програмування. Назва Python відноситься як до мови програмування, так і до інтерпретатора – комп'ютерної програми, яка зчитує початковий код (написаний на Python) і виконує інструкції (команди).

Для перекладу мови високого рівня на машинну мову доступні два типи програм:

1. Компілятор.

2. Інтерпретатор.

Завантаження Python

Версії інтерпретатора Python для різних операційних систем доступні для безкоштовного завантаження за адресою https://www.python.org/downloads.

Середовище програмування для Python

Для написання програм використовують текстові редактори або інтегровані середовища розробки, які включають в себе різні інструменти для роботи з кодом: засіб для написання коду (текстовий редактор), інтерактивний інтерпретатор, відлагоджувач тощо. Текстові редактори та інтегровані середовища програмування для Python:

– IDLE – стандартний редактор Python. Встановлюється разом з Python для користувачів Windows, окремим пакунком для користувачів Linux.

– Notepad++ – безкоштовний текстовий редактор початкового коду, який підтримує велику кількість мов, в тому числі і Python. Лише для користувачів Windows.

– Visual Studio Code – це легкий, але потужний редактор початкового коду, який розповсюджується безкоштовно і доступний у версіях для платформ Linux, Windows і macOS.

– PyScripter – інтегроване середовище розробки для мови програмування Python. Для користувачів Windows. Поширюється безкоштовно.

– Wing IDE 101 – вільне інтегроване середовище для Python, розроблене для навчання програмістів-початківців. Для користувачів Linux, Windows і macOS. Поширюється безкоштовно.

– Geany – вільний текстовий редактор з базовими елементами інтегрованого середовища розробки, доступний для операційних систем Linux, Windows і macOS.

– PyCharm – інтегроване середовище розробки для мови програмування Python. PyCharm є власницьким програмним забезпеченням. Наявна безкоштовна версія Community з усіченим набором можливостей. Для користувачів Linux, Windows і macOS.

– Thonny – IDE для вивчення програмування мовою Python. Для користувачів Linux, Windows і macOS.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

У Посібнику з погодженого аудита виділено **20 критичних засобів керування безпекою (**для стислості далі будемо називати їхніми **Контролями)**, які ефективні для блокування як відомих на теперішній час атак, так і типів атак, очікуваних до появи в найближчому майбутньому. Опису кожного з 20 Контролів включають окремі елементи, що описують дії, які організація може почати для посилення захищеності.

Області застосування Контролів і їхній окремий елементи фокусуються на різних технічних аспектах інформаційної безпеки з головною метою – організувати захист ресурсів організації від найпоширеніших і руйнівних комп'ютерних і мережних атак.

Однак варто пам'ятати, що крім технічного захисту інформації, **комплексна** програма безпеки також повинна враховувати й інші сфери – включаючи загальну політику безпеки, організаційну структуру, питання персоналу й фізичну безпеку – які не ввійшли в область даного документа.

Основні принципи, покладені в основу пропонованих засобів керування безпекою**:**

– міри безпеки повинні бути спрямовані на протидію найпоширенішим і руйнівним видам атак, відомих у теперішньому часі й очікуваних у найближчому майбутньому;

– сумісні один з одним засобу керування ІБ повинні охоплювати все інформаційне середовище організації (особливо, при розподіленій структурі) для ефективної протидії атакам;

– засоби безпеки повинні бути по можливості автоматизовані й періодично або постійно автоматично контролюватися;

– для захисту від найпоширеніших атак, спрямованих на велику кількість організацій, повинні вживати відповідні технічні міри;

– необхідно встановлювати основні причини виникаючих проблем для своєчасного виявлення й запобігання атак;

– вимір ефективності засобів керування безпекою для спрощення й однаковості повинне вироблятися по **метриках**, що приводиться в описі кожного Контролю, що дозволить забезпечити взаєморозуміння виконавців, фахівців з інформаційної безпеки, аудиторів і керівників по безпеці в питаннях оцінки ризиків в організації.

Розглянуті засоби керування розроблені для організацій з різними потребами в рівні захисту інформації.

20 Контролів містять елементи як для того, щоб допомогти організаціям розробити надійну основу безпеки, так і вдосконалити її надалі.

Ці елементи розбиті на категорії залежно від досягається за рахунок їхнього застосування рівня організації інформаційної безпеки:

– **Швидка перемога**. Ці основні аспекти інформаційної безпеки можуть допомогти організації за короткий час поліпшити позиції безпеки звичайно без великих процедурних, архітектурних або технічних змін у середовищі. Однак дані засоби не створюють комплексного захисту від найбільш критичних атак. «Швидкі перемоги» – це області, у яких безпека може бути швидко поліпшена.

– **Прозорість/атрибуція**. Ці елементи сфальцьовані на вдосконаленні виробничого процесу, архітектури й технічних можливостей організацій, для полегшення моніторингу їхніх мереж і комп'ютерних систем і кращої візуалізації застосовуваних ІТ. Атрибуція означає визначення, які комп'ютерні системи й, по можливості, які користувачі генерують окремі види подій. Така поліпшена прозорість і атрибуція допомагає організаціям визначати спроби атак, локалізувати крапки доступу успішних атак, ідентифікувати заражені машини, переривати діяльність порушників, що проникнули в мережу, і одержувати інформацію про джерела атаки.

– **Конфігурація/Гігієна**. Ці елементи розроблені для зміцнення інформаційної безпеки організація шляхом зниження кількості й розміру потенційних уразливостей безпеки й удосконалення функціонування мережних комп'ютерних систем. Вони спрямовані на захист від низької підготовки в області інформаційної безпеки системних адміністраторів і кінцевих користувачів, що може дати порушникові перевага при проведенні атаки. Посібника з керування в цій категорії формулюються з розумінням того, що мережа з ефективним керуванням звичайно являє собою більше важку мету.

– **Розширення**. Ці елементи розроблені для подальшого поліпшення безпеки організації в додавання до перерахованим вище категорій. До впровадження цієї категорії організації приступають після реалізації інших елементів.

Використання Керівництва припускає, що організації порівнюють опис 20 областей керування зі своїми ресурсами й розробляють конкретний план для реалізації засобів керування як критичних компонентів загальної програми безпеки. Зрештою, організації повинні прагнути привести у відповідність Керівництву кожну контрольну область, застосовуючи всі елементи в кожному Контролі, випливаючи від Швидких перемог, Прозорості-атрибуції й Конфігурації/Гігієни до Розширень.

Багато які з описаних засобів керування ІБ можуть бути реалізовані й оцінені існуючими інструментами (ПЗ), які присутні в більшості державних організацій. Інші засоби керування можуть реалізовуватися комерційним або навіть у деяких випадках безкоштовним відкритим ПЗ. А деякі засоби керування можуть зажадати вкладень у їхню розробку й проведення підготовки персоналу.

Опис Контролю включає частина **Метрики**, що містить докладну інформацію про конкретні строки й цілях найбільш важливих елементів даного Контролю.

Крім того, у Контролях також утримується частина **Тестування**, що описує, як організації можуть оцінити досягнення кожної контрольної метрики.

Ці тести призначені для автоматизації процесів, щоб організації могли забезпечити достовірну, масштабовану й постійну оцінку реалізації Контролів і відповідності метрикам.

Організація й контроль ефективності захисту інформації, оцінка безпеки. Погодженість діяльності по захисту інформації на всіх рівнях підприємства й державних організацій, що регулюють питання інформаційної безпеки

Керівники підприємств зацікавлені в поліпшенні стану інформаційної безпеки в державних і комерційних організаціях, на що витрачається багато грошей. Захист таких систем дуже складна, комплексна, тому важливо знати, на які критичні області ризику варто звернути особливу увагу (і направити більші гроші). Заступники директора по інформаційній безпеці (CISO) і інформаційним технологіям (CIO) також хотіли б мати спеціальне Керівництво, яке можна було б застосовувати на всіх рівнях підприємства і яке б також дозволяло послідовно й об'єктивно **контролювати ефективність застосування Керівництва** й поліпшення безпеки. У той же час державні органи, що здійснюють регулювання діяльності по забезпеченню захисту інформації, юристи, органи виконавчої влади й аудитори (CISA) також мають потребу в окремому Посібнику з **оцінки безпеки**. І нарешті, технічний персонал, зайнятий діяльністю по захисту інформації й системному адмініструванню також має потребу в окремому переліку технічних мір для того, щоб **організувати захист** від існуючих і очікуваних видів атак.

У документі «20 критичних Контролів» зроблений перший крок до розробки окремих Керівництв, які CISOs, CIOs, CISAs і різні групи по реагуванню на інциденти – техвідділи інформаційної безпеки можуть надати технічним системним адміністраторам і персоналу відділів безпеки для створення основи механізму керування безпекою на місцях. Пропоновані засоби керування поєднують досвід по аналізі успішних атак, проведених проти державних систем, промислових і комерційних мереж.

У використаному підході пропонуються стандартизовані концепції ідентифікації, зв'язки й документування даних/характеристик, що ставляться до безпеки. Ці стандарти включають загальну ідентифікацію уразливостей і їхнього розміру (серйозності), визначення безпечних конфігурацій, інвентаризацію систем і платформ і ідентифікацію недоліків додатків.

Як приклад такого підходу можна відзначити Програму Автоматизації Контенту Безпеки (Security Content Automation Program (SCAP)), розроблену за замовленням NIST, що стандартизує загальновідому термінологію безпеки й критерії оцінки для оцінки уразливостей, патчів і конфігурації й призначена для використання в автоматизованих інструментах, у тому числі й описуваних у цьому документі.

Відповідність 20 Критичних Контролів керівництвам NIST (National Institute of Information Security and Technology США)

«20 критичних Контролів» описують сукупність дій по керуванню безпекою, на яких керівникам підприємства варто зосередити зусилля для захисту від існуючих видів кіберзлочинів. Контролі включають тільки технічні аспекти, які становлять біля однієї третьої з 145 засобів керування ІБ, певних в 3їй редакції документа NIST 800-53 «Рекомендуються средства, що, керування безпекою для державних інформаційних систем», що володіють найвищою пріоритетністю й критичністю і які є базисним набором при керуванні безпекою організації, як мінімум у технічній і організаційній областях. Після використання 20 критичних Контролів рекомендується також використовувати 800-53 для вибору найбільш ефективних засобів керування й посилення безпеки в певних областях ризику.

Розроблювачі документа

Фахівці Міністерства оборони й урядових служб, що займаються виявленням вторгнень у мережі й захистом від них на початковому етапі; група швидкого реагування на комп'ютерні інциденти США й інші невоєнні групи й консультанти; військові фахівці, ФБР і інші урядові організації, що борються з

кіберзлочиністю; експерти по аналітиці кіберзлочинів державних і неурядових організацій, дослідницьких центрів; тестувальники державних і часток систем на можливість проникнення; керівники, що мають досвід у сфері боротьби з кіберзлочиністю.

У перелік входять засобу керування ІБ, які можна періодично контролювати й оцінювати ефективність, принаймні, частково автоматично й ті, які можуть контролюватися вручну.

Вибір категорій 20 Контролів обґрунтований рейтингом нейтралізації наслідків атак, сфальцьований на створенні найбільш важливих захисних засобів і зустрічається в більшості керівництв і документів по аудиті. Кожна з 20 контрольних областей важлива й містить найбільш пріоритетні методики для запобігань реальних видів атак:

1. Інвентаризація авторизованого й неавторизованого встаткування.

2. Інвентаризація авторизованого й неавторизованого програмного забезпечення.

3. Безпечна конфігурація програмного й апаратного забезпечення ноутбуків, робочих станцій і серверів.

4. Безперервний аналіз і усунення уразливостей.

5. Захист від шкідливого коду.

6. Безпека прикладного ПЗ.

7. Контроль і захист бездротових пристроїв.

8. Можливість відновлення даних (перевіряється вручну).

9. Оцінка навичок по безпеці й проведення тренінгів для підтвердження ефективності (перевіряється вручну).

10. Безпечна конфігурація мережного встаткування (міжмережні екрани, маршрутизатори, комутатори).

11. Обмеження й контроль мережних протоколів, портів і служб.

12. Контроль використання адміністративних привілеїв.

13. Захист периметра.

14.Ведення, моніторинг і аналіз журналів реєстрації подій безпеки.

15. Контроль доступу на основі мінімально необхідних прав (принцип «необхідно знати»).

16. Моніторинг і контроль облікових записів.

17. Запобігання витоку даних.

18. Можливість реагування на інциденти інформаційної безпеки (перевіряється вручну).

19. Архітектура безпеки мережі (перевіряється вручну).

20. Тестування на проникнення, вправи й навчання (перевіряється вручну).

Опис Контролів містить опис уразливості системи при його відсутності, список елементів, що деталізують, що описують, що організації варто зробити в цій області, і вимоги по оцінці цієї діяльності, а також пропозиції по застосуванню стандартизованих оцінок. Після застосувань в організації засобів керування й автоматизації, документ може бути використаний як посібник з аудита, що керівники можуть використовувати для підтвердження ефективності дій кіберзахисту й CISAs – для перевірки результатів тестування.

Зовнішні й внутрішні погрози

Внутрішні погрози враховуються в розглянутих засобах керування двома способами. По-перше, ведення журналів реєстрації подій безпеки, контроль використань адміністративних привілеїв, керування доступом на основі мінімально необхідних прав, запобігання витоку даних і ефективне реагування на інциденти безпеки прямо ставляться до запобігання внутрішніх погроз. По-друге, зовнішні погрози іноді можуть перетворюватися у внутрішні, коли порушник проникає через периметр безпеки. Всі засоби керувань, що обмежують несанкціонований доступ в організації, спрямовані на запобігання як внутрішніх так і зовнішніх погроз і розраховані на різні типи комп'ютерних порушників, включаючи несумлінних штатних співробітників і підрядників, незалежних одиночних зовнішніх порушників, організовані кримінальні групи, терористів, державні органи й т.д., а також комбінації різних типів погроз. Однак порушники

з великим фінансуванням і підготовкою, можуть застосовувати методи атаки, які вимагають спеціальних, набагато більших захисних засобів, не описаних у даному документі.

Засобу керування служать для блокування початкового вторгнення в системи, а також для виявлення вже заражених машин і запобігання й блокування дій порушників.

Захисні засоби, описувані в Контролях, спрямовані на зменшення початкової площі атаки шляхом посилення безпеки, ідентифікацію вже заражених машин для визначення довгострокових погроз усередині мережі організації, контроль привілеїв суперкористувача в системах і блокування захвата керування убудованим шкідливим ПЗ.

Ці дії включають первинне зараження машини для підготовки бази для використання однієї або декількох уразливостей. Порушники можуть потім установлювати довгостроковий доступ до системи, найчастіше шляхом створення облікових записів, руйнування наявних облікових записів або зміни встановленого ПЗ для створення «чорних ходів» і руткітів. Порушники, що одержали доступ до машин можуть також завдавати шкоди інформації шляхом її розкрадання, зміни або знищення, послабляючи функціональні можливості й ефективність системи; або використовуючи заражену машину як відправну крапку для зараження інших систем у середовищі.

При перетинанні кіл у порушників з'являється більше можливостей для одержання доступу до конфіденційної інформації або нанесення іншого збитку.

Різні стратегії захисту, розташовані за кожним набором кіл, описуються Контролями в цьому документі. Захисні засоби у всіх кільцях допомагають обмежувати можливості порушників, однак для всіх трьох кіл і їхніх перетинань потрібні більше зроблені засоби захисту. Важливо відзначити, що 20 Критичних засобів керування розроблені для посилення захисту в кожному з кіл, а не на запобігання окремих атак, тобто це спроба створення повної системи комп'ютерної безпеки.

Схеми атак постійно змінюються й важливо, щоб організації були повністю інформовані про стан свого середовища й забезпечували її регулярний моніторинг.

Однієї зі сфер посиленої уваги для порушників є зараження критичної інфраструктури (наприклад, системи доменних імен і протоколів шлюзів периметра). Із все більшим використанням Інтернет і протоколу IPv6 ці проблеми зростають.

Збільшується кількість атак «нульового дня» (zero day). Оскільки важко прямо захиститися від атак «нульового дня», у багатьох випадках запускається непотрібне зовнішнє ПЗ. В атаці Aurora для захвата клієнтської сторони використовувалася уразливість старої версії ПЗ. Зловмисники використовували цей механізм для зараження Google і багатьох інших великих організацій у спробі доступу до конфіденційної інформації організації у внутрішніх мережах. Атака Aurora здійснювалася за допомогою експлуатації уразливості «нульового дня», для її активізації всього лише було необхідно використання Internet Explorer 6.

Звичайно організації зосереджують заходу щодо безпеки на системах, видимих з Інтернету або інших недовірених мереж. Однак найчастіше ці системи з'єднані із внутрішніми системами, які можуть містити конфіденційну інформацію. Порушники часто створюють, а потім використовують опорні крапки, переміщаючись від системи до системи, поки не досягнуть своєї мети, будь це розкрадання інформації або втручання в роботу.

Періодичне й безперервне тестування засобів керування

Кожний Контроль, включений у даний документ, описує послідовність тестів, які організація може проводити періодично або в деяких випадках безупинно для підтвердження функціонування й ефективності захисних засобів.

Метою тестування є максимальна автоматизація процесу. Стандартизуючи й використовуючи репозитарії – бази контенту, такі як SCAP, – ці автоматичні тестувальні набори й сценарії можуть бути просто й узгоджено застосовані в різних організаціях і використовуватися аудиторами для підтвердження

ефективності Контролів. Ключовий елемент для підтримки автоматичної оцінки – інфраструктура керування мережі організації. Контрольовані мережі звичайно мають спеціальні мілкомодульні засоби для віддаленого збору даних, аналізу й відновлення конфігурації робочих станцій, серверів і мережного встаткування

Однак автоматизація тестування все-таки вимагає участі тестувальників-людей для запуску тестів або оцінки результатів у випадках, які не можуть бути автоматизовані. Тестувальники, відповідальні за оцінку таких засобів (що підтверджують вручну їхня результативність і ефективність), повинні бути довіреними особами, оскільки тестування може зажадати доступ до конфіденційних систем або інформації.

Без відповідної авторизації, перевірки минулого й, можливо, оформлення допуску, такі тести можуть бути нереалізовані. Такі тести також повинні контролюватися або перевірятися відповідними посадовими особами організації, що мають досвід у моніторингу й аналізі систем IT, а також знайомих з вимогами інструкцій із захисту конфіденційної інформації.

Майбутнє 20 Контролів

Зміна технологій і схем атак робить необхідним і майбутньої зміни в списку Контролів, що визначає постійний розвиток документа. Контролі, описані в цій версії – надійний старт для того, щоб зробити основний захист комп'ютера добре зрозумілим, відтвореним, нарощуваним і надійним процесом для всіх державних і комерційних організацій.

Як порушники використовують відсутність цього засобу керування?

Багато кримінальних груп, та й державні організації, безупинно сканують адресні простори їхніх організацій, що цікавлять, для виявлення нових незахищених систем, підключених до мережі. Порушників також цікавлять старі моделі ноутбуків із застарілими патчами, які нечасто підключаються до мережі. Одна з розповсюджених атак експлуатує нове обладнання, тільки встановлене в мережі (наприклад, увечері) і ще не зконфигуроване й незахищене відповідними оновленими версіями захисного ПЗ до початку наступного робочого дня.

Порушники з будь-якої крапки земної кулі можуть швидко знайти й використовувати такі системи, доступні через Інтернет. Порушники, що вже мають внутрішній доступ до мережі, можуть шукати й заражати неправильно захищені внутрішні комп'ютерні ресурси. Деякі порушники можуть використовувати місцевий неробочий час (ніч), щоб установити «чорні ходи» у системах перед тим, як вони будуть захищені.

Крім того, порушники часто шукають експериментальні або тестові системи, які звичайно прямо включаються в мережу, але не входять у стандартний перелік ресурсів організації. Такі експериментальні системи звичайно не мають повного, ґрунтовного захисту, як інші системи в мережі. Хоча такі тестові системи звичайно не містять конфіденційної інформації, вони надають порушникові шлях в організацію й відправну крапку для більше глибокого вторгнення.

З розвитком нових технологій багато працівників приносять персональні пристрої на роботу й підключають їх до мережі. Ці пристрої можуть уже бути заражені й заражати внутрішні ресурси. Порушники також частіше використовують спосіб захвата однієї системи й використання її, як відправної крапки для злому інших систем, які можуть не бути прямо видимі для порушника.

3.2 Розробка структурної схеми

Реалізація, автоматизація й оцінка ефективності Контролю 1

Ведення точного й актуального реєстру – інвентаризація – контрольованого активним моніторингом і керуванням конфігурацією, може зменшити ймовірність виявлення порушником неавторизованих і незахищених систем, якими він міг би скористатися.

1. Швидкі перемоги. Необхідно забезпечити розгортання автоматизованого інструмента по виявленню ресурсів і використання його для

побудови попереднього переліку ресурсів систем, підключених до мережі організації. Для цього повинні використовуватися як активні інструменти, скануючі області мережних адрес, так і пасивні інструменти, які ідентифікують активне встаткування на основі аналізу їх трафіку.

2. Прозорість/атрибуція. Необхідно забезпечити ведення реєстру ресурсів всіх систем, підключених до мережі, і самих мережних пристроїв, реєструючи, принаймні, мережні адреси, машинні імена, призначення кожної системи, власників ресурсів, відповідальних за кожний пристрій, і відділ, до якого ставиться кожний пристрій. Реєстр повинен включати всі системи, що використовують IP-адреси, включаючи робітники станції, ноутбуки, сервери, мережні встаткування (маршрутизатори, концентратори, міжмережні екрани й т.д.), принтери, системи зберігання даних, VoIP-телефони й інші можливі види пристроїв.

3. Прозорість/атрибуція. Створений реєстр ресурсів повинен містити дані, чи є пристрій портативним. Такі пристрої, як мобільні телефони, планшети, ноутбуки й інші портативні електронні пристрої, що зберігають або обробляють інформацію, повинні бути ідентифіковані незалежно від того, чи з'єднані вони з мережею організації.

4. Прозорість/атрибуція. Необхідно гарантувати, що інструменти моніторингу мережного реєстру працездатні й безупинно функціонують, підтримуючи реєстр ресурсів актуальним у реальному режимі часу, у пошуках відхилень від передбачуваного реєстру ресурсів мережі й попереджаючи співробітників служби (відділу/керування) інформаційної безпеки й/або IT персонал при виявленні відхилень.

5. Конфігурація/гігієна. Необхідно забезпечити захист бази даних реєстру ресурсів, забезпечивши їхнє включення в періодичні перевірки на уразливість і шифрування інформації ресурсів. Потрібно здійснити обмеження доступу до цих систем тільки авторизованим персоналом і ведення докладного журналу одержання доступу. Для підвищення безпеки захищена копія реєстру

ресурсів може зберігатися в автономній системі, рознесеної в просторі з виробничою мережею.

6. Конфігурація/гігієна. На додаток до реєстру встаткування організаціям варто розробити реєстр інформаційних ресурсів, що описує критичну інформацію й прив'язує її до апаратних ресурсів (включаючи сервери, робочі станції й ноутбуки), на яких вона розташована. Варто встановити, зареєструвати й контролювати відділ і персонально відповідального за кожний інформаційний ресурс.

7. Конфігурація/гігієна. Необхідно встановити автентифікацію на мережному рівні за протоколомі 802.1x для обмеження й контролю пристроїв, які можуть підключатися до мережі. Протокол 802.1x повинен бути прив'язаний до даних реєстру для визначення авторизованих і неавторизованих систем.

8. Розширення. Керування мережним доступом може використовуватися для моніторингу авторизованих систем так, що при виникненні атаки її вплив може бути зменшене шляхом переміщення недовіреної системи у віртуальну локальну мережу з мінімальними можливостями доступу.

Процедури й інструменти для реалізації й автоматизації цього засобу керування

Для початку організації повинні визначити власників інформації й інформаційних ресурсів, документуючи рішення, які організації й люди відповідають за кожний компонент інформації й пристрій. Деякі організації ведуть реєстри ресурсів, використовуючи спеціальні, розроблені для них, продукти, або використовують безкоштовні або комерційні рішення й переглядають періодично мережа на наявність нових ресурсів, підключених до неї. Зокрема, коли організації здобувають нові системи, вони реєструють власника й властивості кожного нового ресурсу, включаючи його адресу мережного інтерфейсу керування доступом до середовища (MAC) – унікальний ідентифікатор, жорстко запрограмований у більшість карт і пристроїв мережного інтерфейсу. Ці дані можуть зберігатися в безкоштовних або комерційних системах керування базами даних.

Потім, після складання реєстру ресурсів, багато організацій використовують інструменти для одержання інформації з мережних ресурсів, таких як концентратори, комутатори й маршрутизатори, про активне встаткування, підключеному до мережі. Використовуючи безпечно засвідчені й зашифровані протоколи керування мережею, інструменти, можуть одержувати MAC-адреси й іншу інформацію з мережних пристроїв, що звіряється з реєстром ресурсів організації – серверів, робочих станцій, ноутбуків і інших пристроїв. При підтвердженні MAC-адреси комутатори повинні використовувати 802.1x тільки для дозволу підключення до мережі авторизованих систем.

Організації набудовують безкоштовні або комерційні інструменти сканування мережі для регулярних перевірок мережі, наприклад, кожних 12 годин, передаючи різні типи пакетів для ідентифікації підключених пристроїв. Перед скануванням необхідно переконатися, що смуга пропущення достатня для проведення періодичного сканування, аналізуючи історію навантаження мережі і її можливостей.

При скануванні реєстру скануючі інструменти можуть передавати звичайні ping-пакети (Packet Internet Groper) – використовувані для перевірки доступності адресата шляхом передачі йому спеціального сигналу (ICMP echo request – запит відгуку ICMP) і очікування відповіді, очікуючи відповіді для ідентифікації системи по даному IP-адресі. Оскільки деякі системи блокують вхідні ping-пакети, крім того до них сканери можуть ідентифікувати пристрою в мережі, використовуючи пакети синхронізації (SYN) або квітирування (підтвердження прийому) (ACK) протоколів керування передачею (TCP). Крім ідентифікації IP-адреси пристрою в мережі деякі сканери визначають особливі характерні ознаки для визначення типу операційної системи на виявленій машині.

На додаток до активних інструментів, які сканують мережа, інші інструменти ідентифікації ресурсів пасивно аналізують мережні інтерфейси в пошуках пристроїв шляхом виявлення їх трафіку. Такі пасивні засоби можуть

підключатися до span портів у критичних ділянках мережі для перегляду всіх даних, переданих через такі комутатори, збільшуючи ймовірність ідентифікації обміну між системами через дане встаткування.

Бездротові пристрої (і ноутбуки, що харчуються від мережі) можуть періодично підключатися до мережі й потім пропадати, істотно перемішуючи реєстр доступних у цей час систем. Точно так само важко відслідковувати в реєстрі ресурсів віртуальні машини, коли вони виключені або зупинені, оскільки тоді вони являють собою просто файли в якійсь системі файлів на хості. Крім того, віддалені машини, одержуючи доступ до мережі через технологію віртуальної приватної мережі, можуть з'являтися в мережі на якийсь час і потім від'єднуватися. Фізично або віртуально, кожна машина, прямо підключена до мережі або через VPN, що функціонує або виключена, повинна бути врахована в реєстрі ресурсів організації.

Метрика Контролю 1

Система повинна ідентифікувати будь-які нові неавторизовані пристрої, підключені до мережі, за 24 години, і видавати сигналізацію або посилати електронні повідомлення заданому списку адміністративного персоналу організації. Система повинна автоматично ізолювати неавторизовану систему від мережі протягом 1 години з первинного виявлення й генерувати додаткові сигнали або електронні повідомлення при досягненні ізоляції. Кожні 24 години після цього, система повинна генерувати сигнали або посилати електронні повідомлення про статус заблокованої системи, поки вона не буде віддалена з мережі. База даних реєстру ресурсів і система сигналізації повинні ідентифікувати місце розташування, відділ і інші деталі місця підключення авторизованих і неавторизованих пристроїв до мережі.

У той час як 24-вартовий і вартовий тимчасові інтервали становлять дану метрику, щоб дозволити організації поліпшити стан системи безпеки в майбутньому організації повинні прагнути до більше швидкої генерації сигналів і ізоляції виявленої системи – бажано, щоб повідомлення про підключення до

мережі неавторизованого ресурсу висилалося через 2 хвилини, а ізоляція досягалася за 5 хвилин.

Тестування Контролю 1

Для організації періодичної оцінки реалізації Контролю 1 група оцінки повинна приєднати тестові системи до щонайменше 10 крапок у мережі, включаючи фрагменти мережі, що ставляться до демілітаризованих зон, робочим станціям і серверам.

Дві з тестових систем повинні входити в базу даних реєстру ресурсів, а інші – немає. Група оцінки повинна потім переконатися, що система генерує сигнал і електронне повідомлення про заново підключені системи протягом 24 годин з моменту підключення тестових машин до мережі.

Група оцінки повинна проконтролювати, що система надає дані про розташування всіх тестових машин, підключених до мережі. Для тестових машин, включених до реєстру ресурсів, група також повинна проконтролювати, що система надає інформацію про власника ресурсу.

Група оцінки повинна потім перевірити, що всі тестові системи автоматично ізолюються від виробничої мережі протягом однієї години з первинного повідомлення й що генеруються сигналізація або електронне повідомлення про досягнення ізоляції.

Потім виробляється перевірка ізольованості тестових систем, намагаючись передачею ping-пакетів і інших протоколів з них одержати доступ до систем у виробничій мережі й засвідчуючи, що з'єднання відсутнє.

Сенсори, вимір і оцінка в Контролі 1

Сенсор: автоматична система ведення реєстру ресурсів.

Вимір: пошук інструментів, наприклад, Sourcefire Network RNA, GFI Network Inventory Management Tool для установки й функціонування.

Оцінка: оцінка ґрунтується на тім, як часто й давно проводиться сканування.

Сенсор: автентифікація на мережному рівні.

Вимір: перевірка, що для керування підключенням ресурсів використовується 802.1x або подібне патентоване рішення. Наприклад, Cisco Identity Based Networking.

Оцінка: оцінка являє собою відсоток контрольованих портів організації.

Контроль 1: Діаграма системи міжоб'єктних відносин (System Entity RelationshipDiagram):

По діаграмі міжоб'єктних відносин, певних у даному контролі, буде легше визначити, як його реалізувати, протестувати керування, і визначити, де можуть відбутися потенційні збої в системі.

Рисунок 3.1 – Структурна схема системи

Система керування являє собою пристрій або набір пристроїв, використовуваних для керування іншими пристроями або системами. У даному

прикладі ми розглядаємо встаткування в мережі організації. Ці системи повинні могти визначити випадки підключення до мережі організації неавторизованого встаткування. Нижче наведений перелік кроків представлених у діаграмі, що показують як об'єкти працюють разом для досягнення бізнес-цілей, певних у даному елементі керування (контролі). Список також допомагає ідентифікувати кожний з етапів процесу для того, щоб допомогти виявити потенційні точки збою.

Крок 1: Активний сканер сканує мережні системи й ресурси.

Крок 2: Пасивний сканер здійснює захват трафіку між системами й ресурсами.

Крок 3: Активний сканер передає звіти в базу дані інвентаризації.

Крок 4: Пасивний сканер передає звіти в базу дані інвентаризації.

Крок 5: База дані інвентаризації зберігається офлайн.

Крок 6: База дані інвентаризації ініціює спрацьовування системи оповіщення через систему SIEM.

Крок 7: Система оповіщення повідомляє фахівців служби безпеки.

Крок 8: Фахівці служби безпеки здійснюють моніторинг захищеної бази дані інвентаризації.

Крок 9: Фахівці служби безпеки при необхідності обновляють дані в захищеній базі дані інвентаризації.

Крок 10: Система керування доступом безупинно контролює мережа.

Крок 11: Система керування доступом здійснює перевірку відновлень у базі дані інвентаризації активів.

Комп'ютерні порушники використовують системи, що безупинно сканують адресні простори цільових організацій у пошуках уразливих версій ПЗ. Деякі порушники поширюють документи, медіафайли й інший контент через свої веб-сторінки або чужі, довірені сайти. Нічого не жертва, що підозрює, одержує доступ до такому контенту, використовуючи браузер або іншу клієнтську програму, що містять уразливості, і порушники заражають їхньої машини,

найчастіше встановлюючи програми з мережними закладками й боти, які надають порушникові довгостроковий контроль над системою.

Більше досвідчені хакери можуть використовувати уразливість нульової дати, для якої виробником ще не випущена версія ПЗ з патчем. Без відповідного знання або контролю ПЗ, використовуваного в організації, фахівці з безпеки не зможуть відповідно захистити свої ресурси.

Без можливості скласти реєстр і контролювати, які програми інстальовані й дозволені до виконання на машинах, інформаційна система підприємства уразлива. Такі неконтрольовані машини можуть функціонувати з ПЗ, не використовуваним у виробничому процесі, вносячи потенційні вади в безпеку, або зі шкідливим ПЗ (вірусами), внесеними хакером при зараження системи.

При вдалому зараженні однієї машини хакери звичайно використовують її як пункт для збору конфіденційної інформації із зараженої системи й підключених до неї систем. Крім того заражені машини використовуються як відправна крапка для просування по даній мережі й партнерських мережах. При цьому хакери можуть перетворити одну заражену машину в безліч. Організації, що не мають повного реєстру ПЗ, нездатні виявити уразливі працюючі системи або шкідливе ПЗ для рішення описаних проблем або рятування від хакерів.

Реалізація, автоматизація й оцінка ефективності Контролю 2

1. Швидкі перемоги. Визначите список авторизованого ПЗ, що вимагається підприємству для кожного типу систем, включаючи сервери, робочі станції й ноутбуки різних видів і призначень.

2. Прозорість/Атрибуція. Розгорніть інструменти ведення реєстру ПЗ по всій організації, включаючи всі типи використовуваних операційних систем на серверах, робочих станціях і ноутбуках. Система ведення реєстру ПЗ повинна відслідковувати версії основних операційних систем, а також інстальованих додатків. Інструмент повинен реєструвати не тільки тип ПЗ, інстальованого в кожній системі, але й номер версії й рівень патчів.

3. Прозорість/Атрибуція. Система ведення реєстру ПЗ повинен також переглядати мережа на наявність неавторизованого ПЗ, установленого на кожній машині. Цим неавторизованим ПЗ може бути й легальне ПЗ системного адміністрування, установлене в невідповідній системі, де для нього немає завдань.

4. Конфігурація/Гігієна. Активуйте технологію білого списку додатків, що дозволяє системам запускати тільки дозволене ПЗ й запобігає виконання інших програм у системі, ґрунтуючись на автоматично генеруємому списку дійсного ПЗ з еталонної машини-зразка. Такі інструменти ведення білих списків повинні ґрунтуватися на прийнятних хешуючих алгоритмах для визначення авторизованих послідовностей для виконання системою.

5. Розширення. Для ізоляції й запуску додатків, виконання яких необхідно, але пов'язане з високим ризиком, і які не повинні бути встановлені в мережному середовищі, повинні використовуватися віртуальні машини й/або виділені системи.

6. Розширення. Конфігуруйте клієнтські робочі станції у вигляді непостійної віртуалізованого операційного середовища, що періодично можна легко й швидко відновлювати з копії довіреної системи.

Процедури й інструменти для реалізації й автоматизації цього засобу керування

Комерційне ПЗ й системи ведення реєстру ресурсів широко доступні й використовуються на багатьох підприємствах. Кращі із цих інструментів забезпечують перевірку реєстру сотень загальних додатків, використовуваних на підприємствах, збираючи інформацію про рівень патчів у кожній інстальованій програмі, щоб упевнитися, що це остання версія, і транслюють стандартизовані імена додатків так, як вони приводяться в специфікації загального переліку платформ.

Функції, що реалізують білі й чорні списки програм, дозволені або заборонені до виконання, включені в безліч сучасних комплексів безпеки терміналів. Крім того, комерційні рішення всі частіше включають у комплект

антивірусні й антишпионажні и персональний міжмережний екран (ММЕ) і системи виявлення й запобігання вторгнень, розташовувані на хостах, разом з додатками ведення білих і чорних списків.

Зокрема, більшість рішень безпеки терміналів може аналізувати ім'я, розташування файлової системи й/або криптографічний хеш даного файлу, що виконується, для визначення, чи варто дозволити виконання додатка що захищається. Найбільш ефективні із цих інструментів пропонують користувальницькі білий і чорний списки, засновані на хеші шляху, що виконується, або відповідності регулярних виражень. Деякі мають функцію сірого списку, що дозволяє адміністраторам визначати правила для виконання окремих програм тільки конкретними користувачами й у конкретний час дня, і чорні списки, засновані на специфічних сигнатурах.

Метрика Контролю 2

Система повинна бути здатна ідентифікувати неавторизоване ПЗ, виявляючи як спроби інсталювати, так і виконати його, сповіщаючи адміністративний персонал підприємства протягом 24 годин через сигналізацію або електронне повідомлення e-mail. Системи повинні блокувати інсталяцію, запобігати виконанню або поміщати неавторизоване ПЗ в карантин протягом наступної години, генеруючи оповіщення, коли це відбудеться.

Кожні 24 години після цього система повинна сповіщати персонал про статус системи, поки та не буде віддалена з мережі. 24 вартовий і 1-вартовий інтервали часу становлять поточну метрику, однак для поліпшення стану системи безпеки, у майбутньому необхідно прагнути до більше швидкої генерації сигналів оповіщення, бажано, щоб повідомлення про неавторизований ПЗ висилалося через 2 хвилини, а ізоляція досягалася за 5 хвилин.

Тестування Контролю 2

Для регулярної оцінки виконання Контролю 2 група оцінки повинна помістити доброякісну програму тестування, не включену в список авторизованого ПЗ, в 10 систем мережі.

Дві із систем повинні бути включені в базу даних реєстру ресурсів, інші – немає. Група оцінки потім повинна перевірити, що системи генерують сигналізацію або e-mail повідомлення про новий ПЗ протягом 24 годин. Група повинна також перевірити генерацію сигналів оповіщення протягом наступної години, що вказують, що ПЗ блоковане або поміщено в карантин. Група оцінки повинна переконатися, що система надає деталі розташування кожної машини з новим тестовим ПЗ, включаючи інформацію про власника ресурсу.

Група оцінки потім повинна перевірити, що ПЗ блоковано, спробувавши його запустити, і переконатися, що програма не виконується.

Об'єкти оцінки, Оцінка й Результати в Контролі 2

Об'єкт оцінки: Система ведення реєстру ПЗ.

Оцінка: Щомісяця системи скануються й визначається кількість інстальованих неавторизованих компонентів ПЗ. Перевірка, того, що якщо в одному місяці виявлений неавторизований компонент ПЗ, у наступному місяці він віддаляється із системи.

Результат: Привласнюється 100 %, якщо неавторизованого ПЗ не виявлено. Віднімається 1 % за кожний виявлений неавторизований компонент ПЗ. Якщо неавторизоване ПЗ не віддаляється, віднімається 2 % кожний наступний місяць.

Об'єкт оцінки: ПЗ ведення білого списку додатків.

Оцінка: запустите білий список додатків на всіх ключових серверах і переглядайте журнали щомісяця. Визначите кількість відхилень від списку або кількість серверів, на яких ПЗ заблоковано.

Результат: пройдений, якщо за місяць виникло менш 25 відхилень і в менш 15 системах було відключено ПЗ. В іншому випадку, оцінка не пройдена.

По діаграмі міжоб’єктних відносин, певних у даному контролі, буде легше визначити, як його реалізувати, протестувати керування, і визначити, де можуть відбутися потенційні збої в системі.

Система керування являє собою пристрій або набір пристроїв, використовуваних для керування іншими пристроями або системами. У даному прикладі ми розглядаємо програмне забезпечення (ПЗ), установлене в мережі організації. Ці системи повинні могти визначити, якщо нове програмне забезпечення вводиться в середовище без авторизації вповноваженим персоналом підприємства. Нижче наведений перелік кроків, представлених у діаграмі, що показують, як об'єкти працюють разом для досягнення бізнес-цілей, певних у даному елементі керування (контролі). Список також допомагає ідентифікувати кожний з етапів процесу для того, щоб допомогти виявити потенційні точки збою.

Крок 1: Активний сканер сканує системи й ресурси.

Крок 2: Активний сканер передає звіти в базу дані інвентаризації.

Крок 3: База дані інвентаризації рівняється з переліком авторизованого ПЗ.

Крок 4: База дані інвентаризації ініціює спрацьовування системи оповіщення через систему SIEM.

Крок 5: Система оповіщення повідомляє фахівців служби безпеки

Крок 6: Фахівці служби безпеки здійснюють моніторинг захищеної бази дані інвентаризації.

Крок 7: Фахівці служби безпеки при необхідності обновляють дані в базі дані інвентаризації ПЗ.

Крок 8: Інструмент, що використовує технологію білого списку, безупинно контролює всі системи в мережі.

Крок 9: Інструмент, що використовує технологію білого списку, здійснює перевірку й вносить зміни в базу дані інвентаризації ПЗ.

3.3 Розробка функціональної схеми

Розробка нейрокомп'ютерной мережі в застосуванні до кіберзахищеності

При рішенні задачі кіберзахищеності ми вибрали варіант у якому на етапі навчання в нас є наступні вхідні дані: відкритий текст, ключ, алгоритм шифрування, закритий текст.

На етапі кіберзахищеності в нас є тільки закритий текст.

На етапі розпізнавання відбувається виділення із криптотекста знайомих системі зразків і подання їхнім одним нейроном або нейронним ансамблем на наступних рівнях. Як при навчанні, так і при розпізнаванні вхідні вектора є нечіткими, тобто є невеликий розкид векторів, що належать до одного класу. У зв'язку із цим нейромережа, що здійснює цю операцію, повинна мати певну здатність до статистичного усереднення. Навпроти, може виявитися, що група векторів перебуває в безпосередній близькості друг до друга, але всі вони представляють різні класи. Тоді нейромережа повинна визначати тонкі розходження між векторами. Ще одна вимога до нейромережі низького рівня обробки сигналу – навчання без учителя, тобто здатність самостійно розділяти вхідні сигнали на класи.

Велика кількість нейромережних алгоритмів виконують функцію поділу вхідного криптотекста на класи.

Відомі 3 математичні моделі цього поділу:

1. Поділ вхідних даних гіперплощинами (простий персептрон).

Застосування цього алгоритму виправдано тільки для задач, що володіють високою лінійністю. Наприклад, можна побудувати нейромережу, що розбиває крапки (0,0) і (1,1) на два класи для двовимірного сигналу, але неможливо вирішити задачу по розбивці крапок (0,0), (1,1) – перший клас, і (0,1), (1,0) –

другий. Це широко відомий приклад нездатності простого персептрона вирішити задачу «або що виключає»

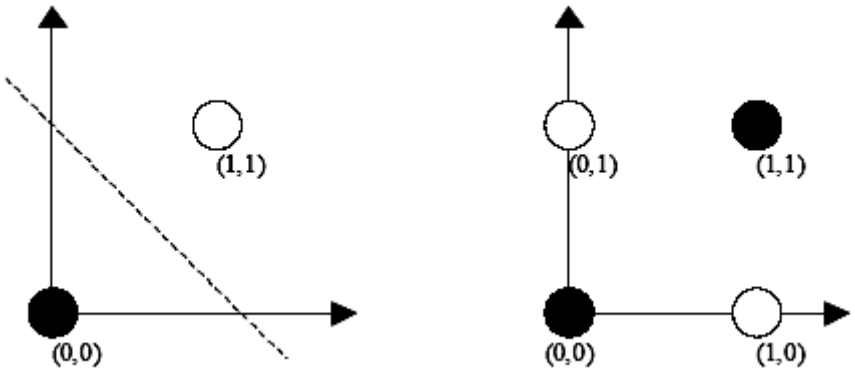

Рисунок 3.2 – Теорема Мінського

2. Поділ вхідних даних гіперповерхнями (багатошарові персептрони).

При послідовному з'єднанні шарів, подібних простому персептрону, з'являється можливість комбінувати гіперплощини й одержувати гіперповерхні досить складної форми, у тому числі й замкнуті. Така нейромережа у принципі при достатнім числі нейронів здатна розділяти сигнали на класи практично будь-якої складності. Але застосування таких нейромереж обмежене складністю їхнього навчання. Був розроблений потужний алгоритм, називаний «алгоритмом зворотного поширення помилки», але й він вимагає значного часу навчання й не гарантує мінімального значення помилки (небезпека влучення в локальні мінімуми).

3. Пошук найбільшої відповідності (найменшого кутового або лінійного стану). При нормалізованих векторах входу, усі вектора розташовуються на поверхні гіперсфери.

Існує модель нейромережі, що відповідає цим вимогам – це мережа зустрічного поширення. В оригіналі вона являє собою об'єднання двох гарно відомих алгоритмів: карти Кохонена, що самоорганізується, й шару Гроссберга. У процесі навчання вхідні вектори асоціюються з відповідними вихідними векторами. Коли мережа навчена, додаток вхідного вектора приводить до

необхідного вихідного вектора. Узагальнююча здатність мережі дозволяє одержувати правильний вихід навіть при додатку вхідного вектора, що є неповним або злегка невірним.

Схематично мережа зустрічного напрямку зображена на рисунку 3.14.

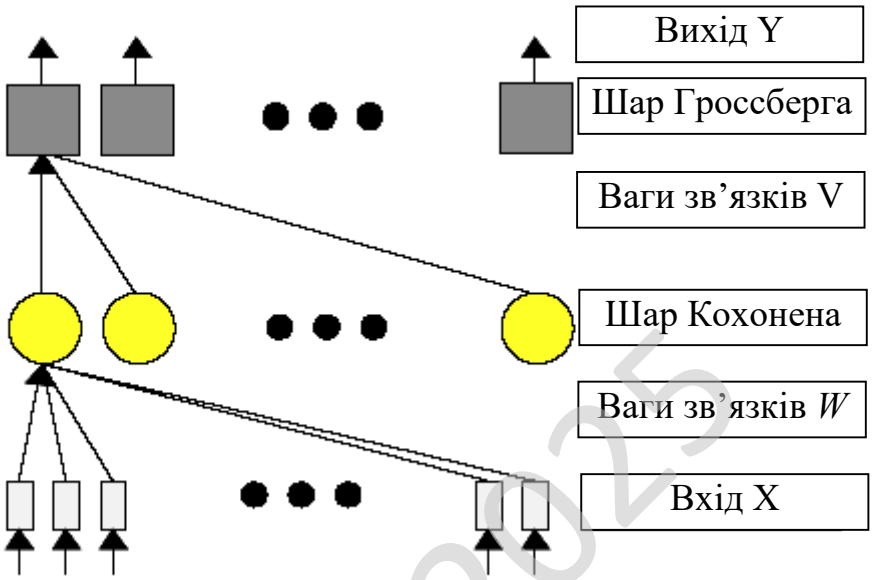

Рисунок 3.3 – Мережа зустрічного поширення

Поширення даних у такій мережі відбувається в такий спосіб: вхідний вектор нормується на 1.0 і подається на вхід, що розподіляє його далі через матрицю ваг *W*. Кожний нейрон у шарі Кохонена обчислює суму на своєму вході й залежно від стану навколишніх нейронів цього шару стають активні або неактивним (рівні 1.0 і 0.0). Нейрони цього шару функціонують за принципом конкуренції, тобто в результаті певної кількості ітерацій активним залишається один нейрон або невелика група, «пухирець активності». Цей механізм називається латеральним гальмуванням і докладно розглянутий у багатьох джерелах. Так як відпрацьовування цього механізму вимагає значних обчислювальних ресурсів, у даній моделі він замінений знаходженням нейрона з максимальною активністю й присвоєнням йому активності 1.0, а всім іншим

нейронам 0.0. Таким чином, спрацьовує нейрон, для якого вектор входу ближче всього до вектора ваг зв'язків.

Якщо мережа перебуває в режимі навчання, то для нейрона, що виграв, відбувається корекція ваг матриці зв'язку за формулою

$$w_н = w_c + \alpha(x - w_c), \quad (3.1)$$

де $w_н$ – нове значення ваги,

w_c – старе значення,

α – швидкість навчання,

x – величина входу.

Геометрично це правило ілюструє наступний рисунок.

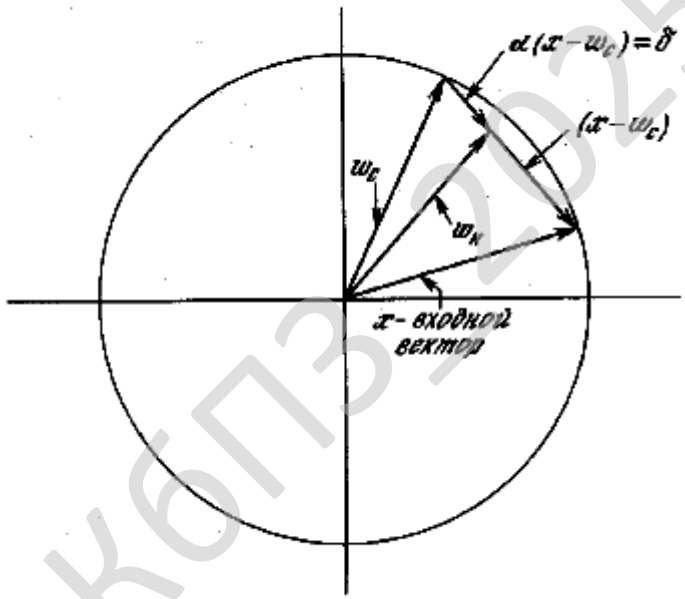

Рисунок 3.4 – Корекція ваг нейрона Кохонена

Так як вхідний вектор x нормований, тобто розташований на гіперсфері одиничного радіуса в просторі ваг, то при корекції ваг за цим правилом відбувається поворот вектора ваг убік закритого тексту. Поступове зменшення швидкості повороту α дозволяє зробити статистичне усереднення вхідних векторів, на які реагує даний нейрон. Проблема: вибір початкових значень ваг. Так як наприкінці навчання вектора ваг будуть розташовуватися на одиничній

окружності, то на початку їх також бажано віднормувати на 1.0. У розглянутій нами моделі вектора ваг вибираються випадковим образом на окружності одиничного радіуса.

Проблема: якщо ваговий вектор виявиться далеко від області входу, він ніколи не дасть найкращої відповідності, завжди буде мати нульовий вихід, отже, не буде коректуватися й виявиться марним. Нейронів, що залишилися, може не вистачити для поділу вхідного простору на класи. Для рішення цієї проблеми пропонується багато алгоритмів, тут же застосовується правило «бажання працювати»: якщо який або нейрон довго не перебуває в активному стані, він підвищує ваги зв'язків доти, поки не стане активним і не почне піддаватися навчанню. Цей метод дозволяє також вирішити проблему тонкої класифікації: якщо утвориться група вхідних даних, розташованих близько друг до друга, із цією групою асоціюється й велика кількість нейронів Кохонена, які розбивають її на класи. Правило «бажання працювати» записується в наступній формі:

$$w_н = w_c + w_c\beta\,(1 - a), \qquad (3.2)$$

де $w_н$ – нове значення ваги,

w_c – старе значення,

β – швидкість модифікації,

a – активність нейрона. Чим менше активність нейрона, тим більше збільшуються ваги зв'язків.

Далі сигнал через матрицю ваг V надходить на шар Гроссберга тут шар спрацьовує по старому методу.

Алгоритм навчання

Вхідні дані: навчальна вибірка (набір вхідних векторів).

Вихідні дані: скоректовані зв'язки.

1. Пред'явити мережі вхідний вектор.

2. Виконувати ітерації до встановлення стабільного стану.

3. Для всіх вузлів мережі виконати корекцію зв'язків згідно (2) або (3).

4. Повторювати [1-3] для кожного вхідного вектора.

Розроблена система кіберзахищеності є досить універсальною, не вимоглива до пам'яті й показала себе ефективніше, ніж класичні методи кіберзахищеності. Достоїнствами даної системи є: швидкість аналізу, легкість адаптації, універсальність (існуючі алгоритми не захищені від такого виду аналізу).

Розглянемо функціональну схему розробленої системи. Вона зображена на рисунку 3.5.

Як видно з рисунку система складається з наступних елементів:

– Зашифрованого тексту.
– Нейронної мережі зустрічного розподілу.
– Блоку порівняння вхідних даних з образами відомими системі.
– Блоку навчаючої вибірки.
– Відкритого тексту.

Спочатку на вхід системи подається навчаюча вибірка, за допомогою якої відбувається навчання нейронної мережі. Потім в програмі відкривається зашифрований текст. Нейронна мережа намагається розпізнати алгоритм шифрування та розшифрувати його без ключа, ще відбувається за допомогою порівняння вхідних даних з образами відомими системі після навчання.

Після розшифрування тексту відбувається додавання нової інформації до бази знань. З кожним наступним використанням програма має все більше інформації для дешифрування.

У якості системи кіберзахищеності використана нейронна мережа зустрічного розподілу, що має три шари:

1. Вхідний шар, X.
2. Шар Кохонена, K.
3. Шар Гроссберга, V.

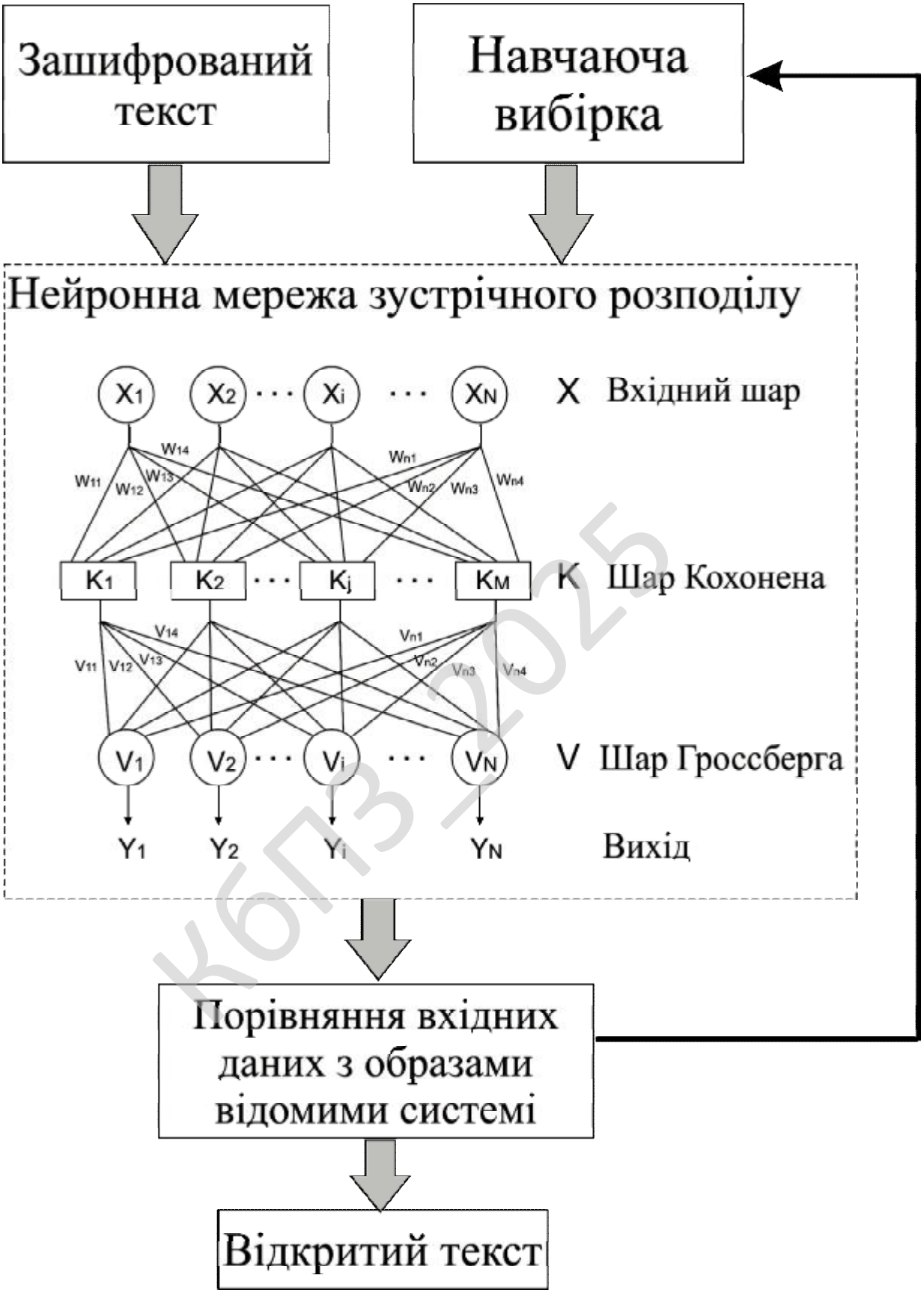

Рисунок 3.5 – Функціональна схема системи кіберзахищеності

Архітектура зустрічного розподілу вдало поєднує у собі переваги можливості узагальнення інформації мережі Кохонена й простоту навчання вихідної зірки Гроссберга. Ця архітектура ідеально підходить для швидкого моделювання систем на початкових етапах досліджень із подальшим переходом, якщо це буде потрібно, на значно більш дорожчий, але більш точний метод навчання зі зворотним поширенням помилок.

Нейронна мережа зустрічного розподілу навчається на вибірці пар векторів (X,Y) задачі подання відображення XY. Чудовою особливістю цієї мережі є здатність навчанню також і відображенню сукупності XY у себе. При цьому, завдяки узагальненню, з'являється можливість відновлення пари (XY) по одному відомому компоненту (X або Y). При пред'явлені на етапі розпізнавання тільки вектора X (з нульовим початковим Y) виконується пряме відображення – відновлюється Y, і навпаки, при відомому Y може бути відновлений відповідний йому X. Можливість рішення як прямої, так і зворотної задачі, а також гібридної задачі по відновленню окремих відсутніх компонентів робить дану нейромережну архітектуру унікальним інструментом.

Мережа зустрічного розподілу складається із двох шарів нейронів: шару Кохонена та шару Гроссберга. У режимі функціонування (розпізнавання) нейрони шару Кохонена працюють за принципом "переможець-забирає-все", визначаючи кластер, до якого належить вхідний образ. Потім вихідна зірка шару Гроссберга по сигналу нейрона-переможця в шарі Кохонена відтворює на виходах мережі відповідний образ.

Навчання ваг шару Кохонена виконується без учителя на основі самоорганізації. Вхідний вектор спочатку нормується, зберігаючи напрямок. Після виконання однієї ітерації навчання визначається нейрон переможець, стан його порушення встановлюється рівним одиниці, і тепер можуть бути модифіковані ваги відповідної йому зірки Гроссберга. Темпи навчання нейронів Кохонена й Гроссберга повинні бути погоджені . У шарі Кохонена навчаються ваги всіх нейронів в околиці переможця, що поступово звужується до одного

нейрона.

Навчена нейронна мережа зустрічного розподілу може функціонувати й у режимі інтерполяції, коли в шарі Кохонена залишається не один, а декілька переможців. Тоді рівні їхньої активності пропорційно нормуються, щоб у сумі становити одиницю, а вихідний вектор визначається по сумі вихідних векторів кожної з активних зірок Гроссберга. У такий спосіб нейронна мережа робить лінійну інтерполяцію між значеннями вихідних векторів, що відповідають декільком кластерам.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.6. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Діаграми потоків даних містять чотири типи елементів:

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Сховища даних (репозиторії).

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

Рисунок 3.6 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ ПРОЕКТУ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ПРАВИЛЬНІСТЬ ПРОЕКТНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Під час роботи над бакалаврською дипломною роботою було створено блок-схеми. Перед їх розглядом необхідно провести роз’яснення який саме тип блок-схем використовується.

Блок-схема це представлення задачі для її аналізу або розв'язування за допомогою спеціальних символів (геометричних образів), які позначають такі елементи, як операції, потік, дані тощо. Блок вхідних та вихідних даних прийнято позначати паралелограмом, блок обчислень (обробки) даних – прямокутником, блок прийняття рішень – ромбом, еліпсом – початок та кінець алгоритму.

У інформаційних технологіях функціональна схема складається з функціональних блоків, які являють собою конструктивно відособлені частини (елементи або пристрої) автоматичних систем, які виконують певні функції. Функціональні блоки на схемі позначають прямокутниками, всередині яких надписують їх найменування відповідно до функцій, що виконуються. Зв'язки між функціональними блоками (внутрішні впливи) позначаються лініями зі стрілками, які вказують напрям впливів.

Функціональні схеми можуть виконуватися в укрупненому і розгорненому вигляді. У першому випадку на схемі зображають найважливіші блоки системи і зв'язки між ними.

У другому варіанті схема відображається більш детально, що полегшує її читання та ілюструє принцип роботи.

Основні елементи схем алгоритму це термінатор, процес, рішення, зумовлений процес (підпрограма), дані та з'єднувач.

Термінатор це елемент відображає вхід із зовнішнього середовища або вихід з неї (найчастіше застосування – початок і кінець програми). Всередині фігури записується відповідна дія.

Процес це виконання однієї або кількох операцій, обробка даних будь-якого виду (зміна значення даних, форми подання, розташування). Всередині фігури записують безпосередньо самі операції.

Рішення це показує рішення або функцію перемикального типу з одним входом і двома або більше альтернативними виходами, з яких тільки один може бути обраний після обчислення умов, визначених всередині цього елементу. Вхід в елемент позначається лінією, що входить зазвичай у верхню вершину елементу. Якщо виходів два чи три то зазвичай кожен вихід позначається лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх слід показувати однією лінією, що виходить з вершини (частіше нижній) елемента, яка потім розгалужується. Відповідні результати обчислень можуть записуватися поруч з лініями, що відображають ці шляхи.

Зумовлений процес (підпрограма) це символ відображає виконання процесу, що складається з однієї або кількох операцій, що визначені в іншому місці програми (у підпрограмі, модулі). Всередині символу записується назва процесу і передані в нього дані.

Дані це перетворення у форму, придатну для обробки (введення) або відображення результатів обробки (виведення). Цей символ не визначає носія даних (для вказівки типу носія даних використовуються специфічні символи).

З'єднувач це символ відображає вихід в частину схеми і вхід з іншої частини цієї схеми. Використовується для обриву лінії та продовження її в іншому місці (приклад: поділ блок-схеми, що не поміщається на листі). Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення.

Блок-схеми є першоджерелами стратегії розвитку ПЗ. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю штучного інтелекту для оцінки кіберзахищеності.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підпрограми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ. При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю. UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

UML може бути застосовано на всіх етапах життєвого циклу аналізу бізнес-систем і розробки прикладних програм. Різні види діаграм які підтримуються UML, і найбагатший набір можливостей представлення певних аспектів системи робить UML універсальним засобом опису як програмних, так і ділових систем.

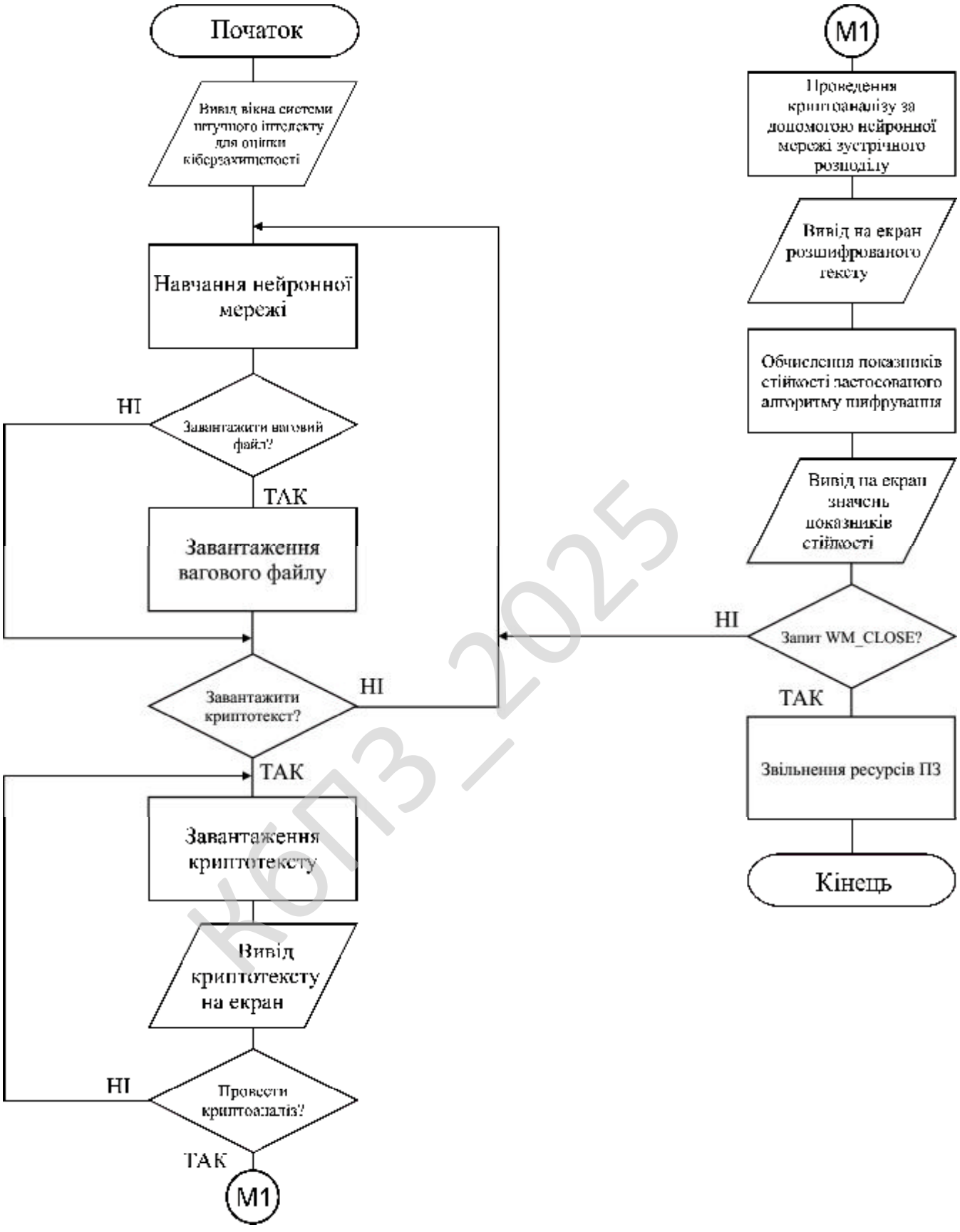

Рисунок 4.1 – Блок-схема основної програми

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код. Основною причиною використання мови UML є спілкування розробників між собою.

Крім того, UML спеціально створювалася для оптимізації процесу розробки програмних систем, що дозволяє збільшити ефективність їх реалізації у кілька разів і помітно поліпшити якість кінцевого продукту.

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об’єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

– Керівникам проектів, які керують розподілом завдань і контролем за проектом.

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

– Бізнес-аналітикам, які досліджують реальну систему і здійснюють інжиніринг і реінжиніринг бізнесу компанії.

– Програмістам які реалізовують модулі інформаційної системи.

При модифікації системи об'єктний підхід дозволяє легко включати в систему нові об'єкти і виключати застарілі без істотної зміни її життєздатності. Використання побудованої моделі при модифікаціях системи дає можливість усунути небажані наслідки змін, оскільки вони не ламають структури системи, а тільки змінюють поведінку об'єктів.

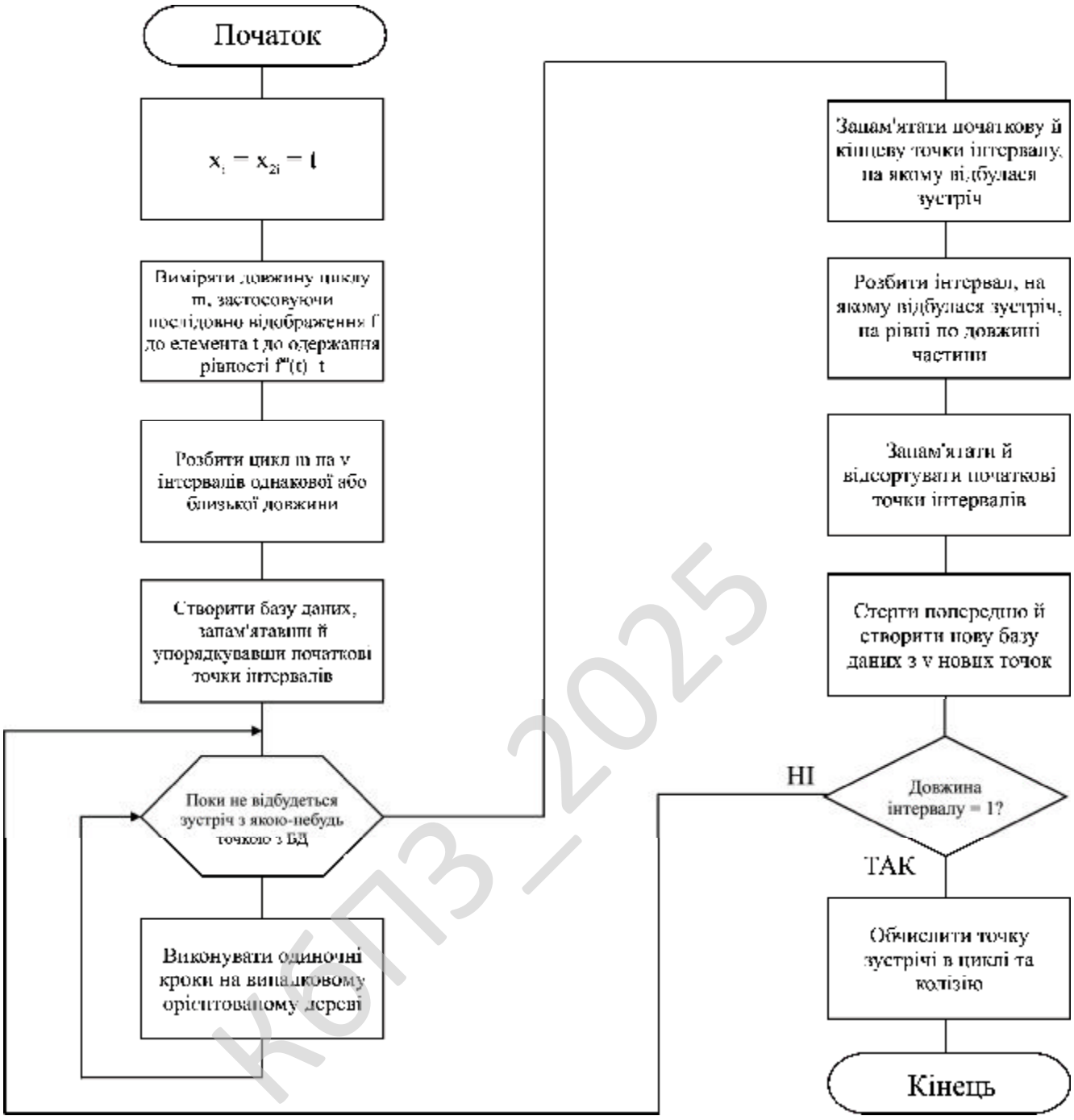

Рисунок 4.2 – Блок-схема роботи підпрограми

Також при розробці бакалаврської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу); діаграма прецедентів (діаграми поведінки типу); Діаграма класів; Діаграма компонент; Діаграма об'єктів; Діаграма розгортання.

Діаграма діяльності. Це візуальне представлення графу діяльностей. Граф діяльностей є різновидом графу станів скінченного автомату, вершинами якого є певні дії, а переходи відбуваються по завершенню дій. Дія є фундаментальною одиницею визначення поведінки в специфікації. Дія отримує множину вхідних сигналів, та перетворює їх на множину вихідних сигналів.

Одна із цих множин, або обидві водночас, можуть бути порожніми. Виконання дії відповідає виконанню окремої дії. Подібно до цього, виконання діяльності є виконанням окремої діяльності, буквально, включно із виконанням тих дій, що містяться в діяльності. Кожна дія в діяльності може виконуватись один, два, або більше разів під час одного виконання діяльності. Щонайменше, дії мають отримувати дані, перетворювати їх та тестувати, деякі дії можуть вимагати певної послідовності.

Специфікація діяльності (на вищих рівнях сумісності) може дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

Діаграма прецедентів це діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених границею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

Суть даної діаграми полягає в наступному: проектована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання (use case) використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система при діалозі з актором.

При цьому нічого не говориться про те, яким чином буде реалізована взаємодія акторів із системою.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

– асоціації (association relationship);

– включення (include relationship);

– розширення (extend relationship);

– узагальнення (generalization relationship).

При цьому загальні властивості варіантів використання можуть бути представлені трьома різними способами, а саме – за допомогою відношень включення, розширення і узагальнення.

Відношення асоціації – одне з фундаментальних понять у мові UML і в тій чи іншій мірі використовується при побудові всіх графічних моделей систем у формі канонічних діаграм.

Включення (include) у мові UML – це різновид відношення залежності між базовим варіантом використання і його спеціальним випадком. При цьому відношенням залежності (dependency) є таке відношення між двома елементами моделі, при якому зміна одного елемента (незалежного) приводить до зміни іншого елемента (залежного).

Відношення розширення (extend) визначає взаємозв'язок базового варіанта використання з іншим варіантом використання, функціональна поведінка якого задіюється базовим не завжди, а тільки при виконанні додаткових умов.

Діаграма класів це статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення.

Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини.

В UML існують наступні типи зв'язків які використовуються у діаграмі класів: Асоціації; Агрегація; Композиція.

Асоціації це якщо між двома класами визначена асоціація, то можна переміщатися від об'єктів одного класу до об'єктів іншого. Цілком припустимі випадки, коли обидва кінці асоціації відносяться до одного і того ж класу. Це означає, що з об'єктом деякого класу дозволено зв'язати інші об'єкти з того ж класу. Асоціація, що зв'язує два класи, називається бінарної. Можна, хоча це рідко буває необхідним, створювати асоціації, що зв'язують відразу кілька класів. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Асоціації може бути присвоєно ім'я, яке описує природу відносини. Зазвичай ім'я асоціації не вказується, якщо тільки ви не хочете явно задати для неї рольові імена або у вашій моделі настільки багато асоціацій, що виникає необхідність посилатися на них і відрізняти один від одного. Ім'я буде особливо корисним, якщо між одними і тими ж класами існує кілька різних асоціацій.

Клас, що бере участь в асоціації, грає в ній деяку роль. По суті, це "обличчя", яким клас, що знаходиться на одній стороні асоціації, звернений до класу з іншого її боку. Можна явно позначити роль, яку клас грає в асоціації.

Часто при моделюванні буває важливо вказати, скільки об'єктів може бути пов'язано допомогою одного примірника асоціації. Це число називається кратністю (Multiplicity) ролі асоціації та записується або як вираз, значенням якого є діапазон значень, або в явному вигляді.

Вказуючи кратність на одному кінці асоціації, ви тим самим говорите, що на цьому кінці саме стільки об'єктів повинно відповідати кожному об'єкту на протилежному кінці. Кратність можна задати рівною одиниці (1), можна вказати

діапазон: "нуль або одиниця" (0..1), "багато" (0 .. *), "одиниця або більше" (1 .. *). Дозволяється також вказувати певне число (наприклад, 3). За допомогою списку можна задати і більш складні кратності, наприклад 0. . 1, 3..4, 6 .. *, що означає "будь-яке число об'єктів, крім 2 і 5".

Агрегація це проста асоціація між двома класами відображає структурний відношення між рівноправними сутностями, коли обидва класу знаходяться на одному концептуальному рівні і ні один не є більш важливим, ніж інший. Але іноді доводиться моделювати відношення типу «частина/ціле», в якому один з класів має більш високий ранг (ціле) і складається з декількох менших за рангом (частин).

Ставлення такого типу називають агрегацією; воно зараховане до відносин типу «має» (з урахуванням того, що об'єкт-ціле має кілька об'єктів-частин). Агрегація є окремим випадком асоціації і зображується у вигляді простої асоціації з незафарбованим ромбом з боку «цілого». Графічно агрегація представляється порожнім ромбом на блоці класу, і лінією, яка від цього ромба до міститься класу.

Композиція це більш суворий варіант агрегації. Відома також як агрегація за значенням.

Композиція має жорстку залежність часу існування екземплярів класу контейнера та примірників містяться класів.

Якщо контейнер буде знищений, то весь його вміст буде також знищено. Графічно представляється як і агрегація, але з зафарбовані ромбиком.

Діаграма компонент в UML це діаграма, на якій відображаються компоненти, залежності та зв'язки між ними.

Діаграма компонент відображає залежності між компонентами програмного забезпечення, включаючи компоненти вихідних кодів, бінарні компоненти, та компоненти, що можуть виконуватись.

Модуль програмного забезпечення може бути представлено в якості компоненти.

Деякі компоненти існують під час компіляції, деякі – під час компонування, а деякі під час роботи програми.

Діаграма компонент відображає лише структурні характеристики, для відображення окремих екземплярів компонент слід використовувати діаграму розгортування.

Компоненти об'єднуються разом використовуючи структурні зв'язки (assembly connector) щоб об'єднати інтерфейси двох компонент. Це ілюструє зв'язок типу «клієнт-сервер».

Структурна взаємодія – «зв'язок двох компонент, який передбачає, що один з них надає послуги, потрібні іншому компоненту».

При використанні діаграми компонент щоб показати внутрішню структуру компонента, клієнтські та серверні інтерфейси можуть утворювати пряме з'єднання з внутрішніми. Таке з'єднання називається з'єднанням делегації.

Діаграма об'єктів в UML це діаграма, що відображає об'єкти та їх зв'язки в певний момент часу. Діаграма об'єктів може розглядатись як окремий випадок діаграми класів, на якій можуть бути представлені як класи, так і екземпляри (об'єкти) класів. Схожою за змістом є діаграма взаємодії (collaboration diagram).

Діаграми об'єктів не мають власної нотації. Оскільки діаграми класів можуть відображати об'єкти, то діаграма класів, на якій відображено лише об'єкти, та не відображено класи, може вважатись діаграмою об'єктів.

Діаграма об'єктів відображає об'єкти та зв'язки в певний момент роботи програми. Об'єкти можуть містити інформацію про власні значення а не про описання. Для відображення загальних шаблонів об'єктів та зв'язків, що можуть багаторазово створюватись під час роботи програми, слід використовувати діаграму взаємодії, яка може відображати характеристики об'єктів та зв'язків. Екземпляр діаграми взаємодії створює діаграму об'єктів.

Діаграма об'єктів не відображає еволюцію системи під час роботи. Натомість, слід використовувати діаграми взаємодії з повідомленнями, або діаграми послідовності.

Діаграма розгортання (deployment diagram) це діаграма в UML, на якій відображаються обчислювальні вузли під час роботи програми, компоненти, та об'єкти, що виконуються на цих вузлах. Компоненти відповідають представленню робочих екземплярів одиниць коду. Компоненти, що не мають представлення під час роботи програми на таких діаграмах не відображаються; натомість, їх можна відобразити на діаграмах компонент. Діаграма розгортання відображає робочі екземпляри компонент, а діаграма компонент, натомість, відображає зв'язки між типами компонент.

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення захистимо за допомогою наступного алгоритму захисту інформації DSA.

Відправник і одержувач електронного документа використовують при обчисленні великі цілі числа: G і P – прості числа, L біт кожне ($512 < L < 1024$); q – простої число довжиною 160 біт (дільник числа (P-1). Числа G, P, q є відкритими й можуть бути загальними для всіх користувачів мережі.

Відправник вибирає випадкове ціле число X, $1 < X < q$. Число X є секретним ключем відправника для формування електронного цифрового підпису. Потім відправник обчислює значення $Y = G^X \bmod P$. Число Y є відкритим ключем для перевірки підпису відправника. Число Y передається всім одержувачам документів.

Цей алгоритм також передбачає використання однобічної функції гешування h(-). У стандарті DSS визначений алгоритм безпечного гешування SHA. Для того щоб підписати документ M, відправник гешує його в ціле геш-значення m: $m = h(M)$, $1 < m < q$, потім генерує випадкове ціле число K, $1 < K < q$, і обчислює число r: $r = (G^K \bmod P) \bmod q$. Потім відправник обчислює за допомогою секретного ключа X ціле число s:

$$s = \frac{m + r * X}{K} \bmod q \ .$$

Пара чисел r і s утворить цифровий підпис $S = (r, s)$ під документом M. Таким чином, підписане повідомлення являє собою трійку чисел $[M,r,s]$. Одержувач підписаного повідомлення $[M, r, s]$ перевіряє виконання умов $0 < r < q$, $0 < s < q$ і відкидає підпис, якщо хоча б одна із цих умов не виконана. Потім одержувач обчислює значення $w=1/s \bmod q$, геш-значення $m = h(M)$ і числа $u_1 = (m*w) \bmod q$, $u_2 = (r*w) \bmod q$. Далі одержувач за допомогою відкритого ключа Y обчислює значення $v = ((G^{U1} * Y^{U2}) \bmod P) \bmod q$ і перевіряє виконання умови $v = r$. Якщо умова $v = r$ виконується, тоді підпис $S = (r,s)$ під документом M визнається одержувачем справжнім.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс програмного забезпечення, розробленого у результаті виконання бакалаврської дипломної роботи. Розроблене програмне забезпечення штучного інтелекту для оцінки кіберзахищеності складається з наступних функціональних блоків:

– Навігаційне меню: Файл; Нейрона мережа; Оцінки; Налаштування; Довідка.

– Блоку функціональних кнопок ПЗ: Завантаження; Навчання; Криптоаналіз; Перегляд вагових файлів; Налаштування ПЗ; Налаштування показників стійкості.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

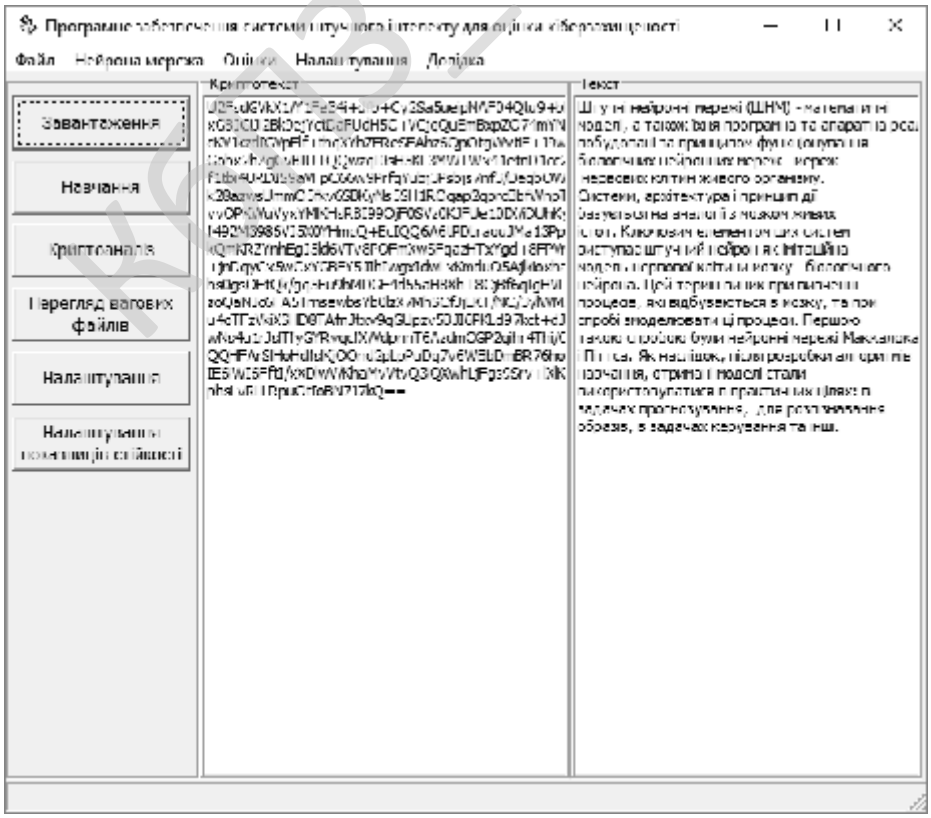

Рисунок 5.1 – Головне вікно розробленого ПЗ

Для перегляду короткої довідки про програму слід натиснути на основному вікні кнопку авторського права, після чого на екрані з’явиться вікно показане на рисунку 5.2.

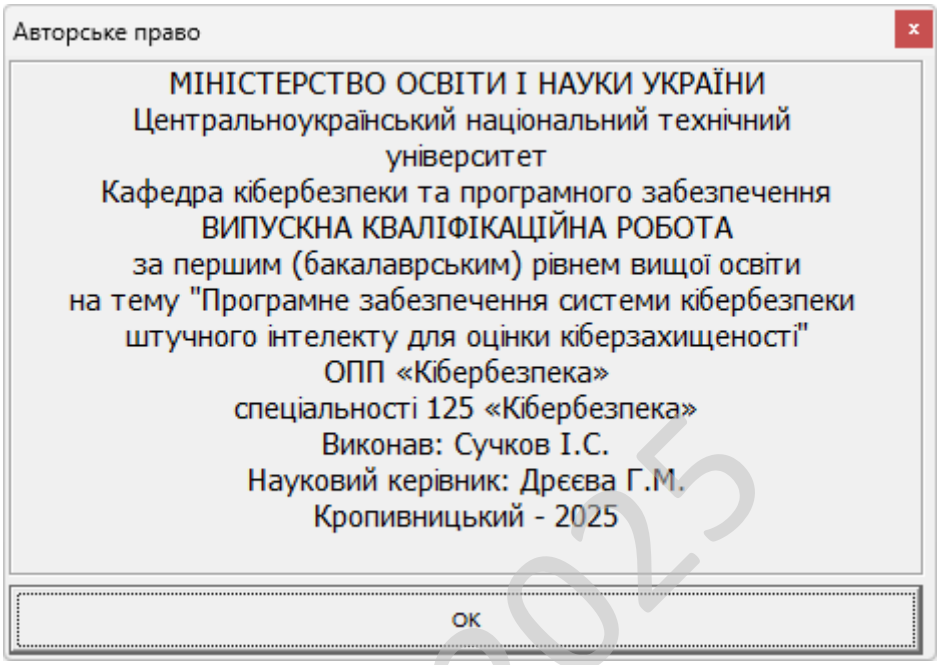

Рисунок 5.2 – Вікно розробника ПЗ

Під час роботи над програмою було проведено тестування програмного забезпечення, тобто технічне дослідження, призначене для виявлення інформації про якість продукту відносно контексту, в якому воно має використовуватись.

Тестування включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою їх оцінки.

Проводилась оцінка:

– відповідності поставленим вимогам;

– правильна відповідь для усіх можливих вхідних даних;

– виконання функцій за прийнятний час;

– практичність;

– сумісність з ОС та стороннім ПЗ.

Оскільки число можливих тестів для програмних компонент практично нескінченне, тому стратегія тестування полягала в тому, щоб провести всі можливі тести з урахуванням наявного часу та ресурсів.

Як результат ПЗ тестувалось стандартним виконанням програми з метою виявлення помилок або інших дефектів.

Проводилось тестування форматом білої засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графа управління.

У цьому випадку формуються тестові варіанти, в яких:

– Гарантується перевірка всіх незалежних маршрутів програми.

– Знаходяться гілки True, False для всіх логічних рішень.

– Виконуються всі цикли (у межах їхніх кордонів та діапазонів).

– Аналізується правильність внутрішніх структур даних.

Недоліки тестування "білої скриньки":

– Кількість незалежних маршрутів може бути дуже велика.

– Повне тестування маршрутів не гарантує відповідності програми вихідним вимогам до неї.

– У програмі можуть бути пропущені деякі маршрути.

– Не можна виявити помилки, поява яких залежить від даних.

Переваги тестування "білої скриньки" пов'язані з тим, що принцип «білої скриньки» дозволяє врахувати особливості програмних помилок:

– Кількість помилок мінімально в «центрі» і максимально на «периферії» програми.

– Попередні припущення про ймовірність потоку керування або даних у програмі часто бувають некоректними. У результаті типовим може стати маршрут, модель обчислень за яким опрацьована слабо.

– При записі алгоритму програмного забезпечення у вигляді тексту на мові програмування можливе внесення типових помилок трансляції (синтаксичних та семантичних).

– Деякі результати в програмі залежать не від вихідних даних, а від внутрішніх станів програми.

Обрано умови розповсюдження – commercial software.

Програмне забезпечення, створене комерційною організацією з метою отримання прибутку від його використання іншими, наприклад, шляхом продажу копій.

Найважливішою особливістю комерційний програмних продуктів є підтримка великих компаній, прямо зацікавлених у поширенні програм. Багато організацій надають виключно платну підтримку своїх продуктів, такий підхід, як правило, використовують організації надають відкриті вихідні коди. Для продуктів, що розповсюджуються на комерційній основі діють зазвичай безкоштовні служби підтримки, покликані збільшити рівень довіри у клієнтів і потенційних покупців.

Далеко не завжди, але як правило терміни критично важливих змін в комерційних продуктах значно менше, ніж у некомерційних проектів. Це пов'язано з тим, що над комерційним продуктом працюють цілі групи розробників і ця робота є їх основним заняттям. Розробникам-початківцям як правило доводиться шукати додаткові способи заробітку, і це збільшує час, що витрачається на доповнення і зміни програм. Так як основним рушійним фактором створення комерційного ПЗ є одержання прибутку, то комерційні програмні продукти першими заповнюють вільні ніші та пропонують варіанти вирішення завдань відразу по мірі виявлення вакууму в будь-якому секторі ринку.

Окремий вид комерційних програм, коли їх розробка оплачується безпосередньо замовником. Такі програми найчастіше позбавлені всіх переваг комерційних продуктів, оскільки мають обмежений бюджет, але більш адаптовані до вимог замовника, ніж аналоги.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем штучного інтелекту для оцінки кіберзахищеності.

– Досліджена система штучного інтелекту для оцінки кіберзахищеності.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання штучного інтелекту для оцінки кіберзахищеності.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Python. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід,

зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм DSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aggarwal C.C. Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition. – Springer, 2018. – 520 p

2. Aho A.V., Hopcroft J.E., Ullman J.D. Data Structures and Algorithms. – Pearson, 2001. – 620 c.

3. Brink H., Richards J., Fetherolf M. Real-World Machine Learning. – Manning, 2016. – 474 p.

4. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. Introduction to Algorithms, 3rd Edition (The MIT Press) 3rd Edition – The MIT Press, 2019. – 1292 p.

5. Fenner M. Machine Learning with Python for Everyone (Addison-Wesley Data & Analytics Series) 1st Edition, Kindle Edition. - Addison-Wesley Professional, 2019. – 586 p.

6. Foreman J.W. Data Smart: Using Data Science to Transform Information into Insight 1st Edition. – Wiley, 2013. – 432 p.

7. Hurbans R. Grokking Artificial Intelligence Algorithms. – Manning, 2020. – 631 p.

8. Gusfield D. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology 1st Edition. – Cambridge University Press, 2008. – 556 p.

9. Kotu V., Deshpande B. Data Science: Concepts and Practice. – Elsevier Science, 2018. – 953 p.

10. Knowledge Base A Complete Guide - 2021 Edition // The Art of Service - Knowledge Base Publishing, 2020. – 306 p.

11. Knuth D. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd Edition 3rd Edition. – Addison-Wesley Professional, 2019. – 672 p.

12. Mattmann C. Machine Learning with TensorFlow, Second Edition. – Manning, 2020. – 1124 p.

13. Mueller J.P., Massaron L. Machine Learning For Dummies. – Wiley, 2016. – 714 p.

14. Teofili T. Deep Learning for Search. – Manning, 2019. – 695 p.

15. Rungta K. TensorFlow in 1 Day: Make your own Neural Network. – Publishdrive, 2019. – 587 p.

16. Weidman S. Deep Learning from Scratch: Building with Python from First Principles. – O'Reilly. 2019. – 252 p.

17. Rajasekaran S., Vijayalakshmi Pai G.A. Neural networks, fuzzy logic, and genetic algorithms: synthesis and applications (with cd-rom) Kindle Edition. – PHI, 2013. – 628 p.

18. Lakhno, V., Malyukov, V., Smirnov, O., Bebeshko, B., Chubaievskiy, V., Zhumadilova, M., Malyukova, I., Smirnov, S. «Multifactorial Model for Targeted Attacks Counteracting Within the Framework of a Multi-Step Quality Game with Fuzzy Information». *8th International Symposium on Intelligent Informatics, ISI 2023*, 2025. vol 389. pp 377-389. Springer, Singapore.

19. Kuznetsov, O., Kryvinska, N., Ilchenko, O., Smirnova, T., Ulianovska, Y. «Comparative Analysis of Cryptocurrency Trading Platforms Using the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, 2023, 3628, pp. 106-115.

20. Smirnov O., Fedorov E., Neskorodieva A., Neskorodieva T. «Intellectual Classification method of Gymnastic Elements Based on Combinations of Descriptive and Generative Approache». *CEUR Workshop Proceedings* Volume 3664, 2024, Pages 11-23.

21. Malyukov V., Bebeshko B., Lakhno V., Smirnov O., Malyukova I., Mohylnyi H. «Managing the Purchase-Sale Process of Digital Currencies Under Fuzzy Conditions». *Lecture Notes in Networks and Systems,* 2023, 729 LNNS, pp. 104–112.

22. Al-Mudhafar Aqeel, A.M., Smirnova, T., Buravchenko, K., Smirnov, O. «The method of assessing and improving the user experience of subscribers in software-configured networks based on the use of machine learning». *Advanced Information Systems*, 2023, 7(2), pp. 49-56.

23. Smirnov, O., Sydorenko, V., Aleksander, M., Zhyharevych, O., Yenchev, S. «Simulation of the cloud IoT-based monitoring system for critical infrastructures». *CEUR Workshop Proceedings,* Volume 3530, 2023, pp. 256-265.

24. Smirnov, O., Karapetyan, A., Fedorov, E., «Creating Neural Network and Single Solution Human-Based Metaheuristic Methods of Solving the Traveling Salesman Problem». CEUR Workshop Proceedings, Volume 3312, 2022, pp. 47-58.

25. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» CEUR Workshop Proceedings, Volume 3187, 2022, pp. 1-12.

26. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». Sensors (Basel, Switzerland) Volume 22, Issue 16, 6223, 2022.

27. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies, vol 131. 2023. Springer, Singapore. pp. 21-34.

28. Kuznetsov, A., Oleshko, I., Chernov, K., Bagmut, M., Smirnova, T. «Biometric authentication using convolutional neural networks». Lecture Notes in Networks and Systems. Volume 152, 2021, Pages 85-98.

29. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». SN Computer Science, Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w.

30. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». CEUR Workshop Proceedings Volume 3101, 2021, Pages 192-207.

31. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». International Journal of Computing; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256.

32. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». CEUR Workshop Proceedings. Volume 2740, 2020, Pages 102-114.

33. Smirnov O., Kuznetsov A., Kiian A., Cherep A., Kanabekova M., Chepurko I. «Testing of code-based pseudorandom number generators for post-quantum application». 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), Ukraine, Kyiv, May 14-18. 2020. P. 172-177.

34. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. Springer, Cham. 2021. pp 557-587.

35. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». CEUR Workshop Proceedings Volume 2616, 2020, Pages 366-379.

36. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», CEUR Workshop Proceedings Volume 2608, 2020, Pages 633-645.

37. Smirnov O. Kuznetsov A., Zaicenko Yu., Pastukhov M., Oleshko O., Kuznetsova K., «Formation of Discrete Signals with Special Correlation Properties». International Conference on Information and Telecommunication Technologies and Radio Electronics, UkrMiCo 2019; Odessa; Ukraine; 9-13 September 2019. P.22-28.

38. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407.

39. Smirnov, O., Kuznetsov, A., Reshetniak, O., Ivko, N., Katkova, T., Kuznetsova, T., «Generators of Pseudorandom Sequence with Multilevel Function of Correlation». 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), Kyiv, Ukraine, 8 – 11 October 2019 . P.517-522.

40. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». CEUR Workshop Proceedings, Vol 2588, P. 215-227, 2019.

41. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». CEUR Workshop Proceedings, Vol 2588, P. 90-106, 2019.

42. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Averchev, A., Pastukhov, M., Kuznetsova, K., «Formation of Pseudorandom Sequences with Special Correlation Properties», 2019 3rd International Conference on Advanced Information and Communications Technologies, AICT -2019/ Lviv, Ukraine, 2-6 July, 2019, P. 395-399.

43. Smirnov, O., Kuznetsov, A., Kiian, A., Zamula, A., Rudenko, S., Hryhorenko, V., «Variance Analysis of Networks Traffic for Intrusion Detection in Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 353-358.

44. Smirnov, O., Kuznetsov, A., Kavun, S., Babenko, B., Nakisko, O., Kuznetsova, K., «Malware Correlation Monitoring in Computer Networks of Promising Smart Grids», 2019 IEEE 6th International Conference On Energy Smart Systems (2019 IEEE ESS), Kyiv, Ukraine April 17-19, 2019 P. 347-352.

45. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», CEUR Workshop Proceedings Volume 2353, CEUR Workshop Proceedings 2019, Pages 618-629.

46. Smirnov, S., Bulekbaeva, G., Kikvidze, O.G., Lakhno, V., Brzhanov, R., Tabylov, A. «Computer simulation in the MathCAD package of plastic deformation of the deposited layer on the flat surface of the part». Journal of Theoretical and Applied Information Technology Volume 97, Issue 20, 2019, Pages 2467-2484. (Scopus).

47. Smirnov A.A., Kuzne tsov A.A., Danilenko D.A., Berezovsky A., «The statistical analysis of a network traffic for the intrusion detection and prevention systems», Telecommunications and Radio Engineering. – Volume 74, Issue 1. – Begel House Inc. – 2015. – P. 61-78.

48. Аль-Мудхафар Акіл Абдулхуссейн М., Смірнова Т.В., Буравченко К.О., Смірнов О.А. «Метод оцінки та підвищення користувальницького досвіду абонентів в програмно-конфігурованих мережах на основі використання машинного навчання». Сучасні інформаційні системи, 2023, том 7, № 2, С. 49-56.

49. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного процесу відновлення поверхонь конструкцій і деталей машин». Сучасні інформаційні системи. 2021. Т. 5, № 4. С. 79-95

50. Смірнов, О.А., Усік П.С., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю. «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». Проблеми телекомунікацій. № 1(26). С. 83-96. 2020.

51. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» Вісник Черкаського державного технологічного університету. Технічні науки. №4. С. 103-110. 2020.

Додаток А
(обов’язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 51-02 від 17.01.2025 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Python.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 129 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2025 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 5.06.2025 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

_____ Дрєєва Г.М.

Програмне забезпечення системи кібербезпеки штучного інтелекту для оцінки кіберзахищеності

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 20

Літера: РП

Кропивницький – 2025 року

Основна програма

```
import os
import sys
import time
import json
import random
import hashlib
import logging
import threading
import requests
import datetime
import socket
import subprocess
import pandas as pd
import numpy as np
import re
import psutil
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Логування подій системи
logging.basicConfig(filename="security_log.txt", level=logging.INFO,
format="%(asctime)s - %(levelname)s - %(message)s")

# Функція для збору інформації про систему
def get_system_info():
    info = {}
    info["os"] = os.name
    info["platform"] = sys.platform
    info["architecture"] = os.uname()
    info["cpu_count"] = psutil.cpu_count()
    info["memory"] = psutil.virtual_memory().total
    info["disk"] = psutil.disk_usage("/").total
    logging.info("Зібрана інформація про систему")
    return info

# Функція для отримання поточних підключень
def get_network_connections():
    connections = psutil.net_connections()
    network_data = []
    for conn in connections:
        conn_info = {
            "fd": conn.fd,
            "family": str(conn.family),
            "type": str(conn.type),
            "local_address": conn.laddr,
            "remote_address": conn.raddr,
            "status": conn.status
        }
        network_data.append(conn_info)
    logging.info("Отримані активні підключення")
    return network_data

# Функція перевірки відкритих портів
def check_open_ports():
    open_ports = []
    for port in range(20, 1025):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(0.1)
        if sock.connect_ex(("127.0.0.1", port)) == 0:
            open_ports.append(port)
        sock.close()
    logging.info(f"Виявлені відкриті порти: {open_ports}")
    return open_ports

# Функція перевірки безпеки паролів
```

```
def check_password_security(passwords):
    common_passwords = {"123456", "password", "admin", "qwerty", "letmein"}
    weak_passwords = []
    for password in passwords:
        if password in common_passwords or len(password) < 8:
            weak_passwords.append(password)
            logging.warning(f"Небезпечний пароль виявлено: {password}")
    return weak_passwords

# Функція моніторингу підозрілих процесів
def monitor_suspicious_processes():
    suspicious_processes = ["malware_service", "unwanted_process", "keylogger",
"ransomware"]
    detected_processes = []
    for proc in psutil.process_iter(attrs=["pid", "name", "cpu_percent",
"memory_percent"]):
        if proc.info["name"].lower() in suspicious_processes:
            detected_processes.append(proc.info)
            logging.warning(f"Виявлено підозрілий процес: {proc.info}")
    return detected_processes

# Функція аналізу мережевого трафіку
def analyze_network_traffic():
    prev_sent, prev_recv = psutil.net_io_counters().bytes_sent,
psutil.net_io_counters().bytes_recv
    time.sleep(5)
    current_sent, current_recv = psutil.net_io_counters().bytes_sent,
psutil.net_io_counters().bytes_recv
    sent_diff = current_sent - prev_sent
    recv_diff = current_recv - prev_recv
    logging.info(f"Передано: {sent_diff} байтів, Отримано: {recv_diff} байтів")
    return sent_diff, recv_diff

# Функція аналізу атак DDoS
def detect_ddos():
    connections = len(psutil.net_connections(kind='inet'))
    if connections > 100:
        logging.warning(f"Можливий DDoS-атак! Кількість підключень:
{connections}")

# Функція перевірки оновлень системи
def check_system_updates():
    try:
        result = subprocess.run(["apt", "list", "--upgradable"],
capture_output=True, text=True)
        updates = result.stdout.strip()
        if updates:
            logging.info("Є доступні оновлення:\n" + updates)
        else:
            logging.info("Система оновлена")
    except Exception as e:
        logging.error(f"Помилка перевірки оновлень: {str(e)}")

# Функція візуалізації даних загроз
def visualize_threats():
    threats = ["Malware", "DDoS", "Phishing", "Ransomware"]
    counts = [random.randint(10, 100) for _ in threats]
    plt.bar(threats, counts, color=["red", "blue", "green", "purple"])
    plt.xlabel("Тип загрози")
    plt.ylabel("Кількість атак")
    plt.title("Статистика кіберзагроз")
    plt.show()

# Функція створення звіту
def generate_security_report():
    report = {
        "system_info": get_system_info(),
        "network_connections": get_network_connections(),
        "open_ports": check_open_ports(),
```

```
        "password_security": check_password_security(["123456",
"SuperSecurePass", "admin"]),
        "suspicious_processes": monitor_suspicious_processes(),
        "network_traffic": analyze_network_traffic(),
        "ddos_detection": detect_ddos(),
        "system_updates": check_system_updates()
    }
    with open("security_report.json", "w") as file:
        json.dump(report, file, indent=4)
    logging.info("Звіт про безпеку створено")
    return report

# Основна функція програми
def main():
    logging.info("Запуск системи кібербезпеки")
    generate_security_report()
    visualize_threats()
    logging.info("Програма завершила роботу")

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

Файл network_traffic_analysis.py.py

```
import psutil
import time
import logging

# Логування подій аналізу мережевого трафіку
logging.basicConfig(filename="network_traffic_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання статистики мережі
def get_network_stats():
    stats = psutil.net_io_counters()
    return stats.bytes_sent, stats.bytes_recv, stats.packets_sent,
stats.packets_recv

# Функція перевірки змін у трафіку
def analyze_network_traffic():
    previous_sent, previous_recv, previous_packets_sent, previous_packets_recv =
get_network_stats()

    while True:
        time.sleep(5)
        current_sent, current_recv, current_packets_sent, current_packets_recv =
get_network_stats()

        delta_sent = current_sent - previous_sent
        delta_recv = current_recv - previous_recv
        delta_packets_sent = current_packets_sent - previous_packets_sent
        delta_packets_recv = current_packets_recv - previous_packets_recv

        logging.info(f"Передано: {delta_sent} байтів, Отримано: {delta_recv}
байтів")
        logging.info(f"Передано пакетів: {delta_packets_sent}, Отримано пакетів:
{delta_packets_recv}")

        if delta_sent > 100000 or delta_recv > 100000:
            logging.warning("Аномально висока активність мережі!")

        previous_sent, previous_recv, previous_packets_sent,
previous_packets_recv = current_sent, current_recv, current_packets_sent,
current_packets_recv

if __name__ == "__main__":
    logging.info("Запуск моніторингу мережевого трафіку")
    analyze_network_traffic()
```

Файл siem_integration.py

```
import requests
import logging
import time

# Логування подій SIEM
logging.basicConfig(filename="siem_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# URL для надсилання даних у SIEM
SIEM_URL = "http://example.com/siem_api"

# Функція створення тестового журналу
def generate_test_logs():
    logs = []
    for i in range(5):
        log_entry = {
            "event_id": i,
            "event_type": "security_alert",
            "message": f"Тестова подія {i}",
            "timestamp": time.strftime("%Y-%m-%d %H:%M:%S")
        }
        logs.append(log_entry)
    return logs

# Функція інтеграції з SIEM
def send_logs_to_siem(log_data):
    for log in log_data:
        try:
            response = requests.post(SIEM_URL, json=log)
            if response.status_code == 200:
                logging.info(f"Дані успішно надіслані до SIEM: {log}")
            else:
                logging.warning(f"Помилка відправки даних:
{response.status_code}")
        except Exception as e:
            logging.error(f"Виникла помилка підключення до SIEM: {str(e)}")

if __name__ == "__main__":
    logging.info("Початок інтеграції з SIEM")
    test_logs = generate_test_logs()
    send_logs_to_siem(test_logs)
    logging.info("Завершення роботи модуля SIEM")
```

Файл behavioral_analysis.py

```
import psutil
import logging
import time

# Логування аномалій
logging.basicConfig(filename="behavioral_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання списку процесів
def get_processes():
    process_list = []
    for proc in psutil.process_iter(attrs=["pid", "name", "cpu_percent",
"memory_percent"]):
        process_list.append(proc.info)
    return process_list

# Функція для аналізу поведінки процесів
def detect_behavioral_anomalies():
    while True:
        time.sleep(10)
        processes = get_processes()
        for proc in processes:
            if proc["cpu_percent"] > 50:
                logging.warning(f"Аномальний процес (високе навантаження CPU):
{proc}")
            if proc["memory_percent"] > 30:
                logging.warning(f"Аномальний процес (високе використання RAM):
{proc}")

if __name__ == "__main__":
    logging.info("Запуск поведінкового аналізу")
    detect_behavioral_anomalies()
```

Файл active_sessions_monitor.py

```
# Моніторинг активних сеансів підключень
import psutil
import logging
import time

# Логування активних сеансів
logging.basicConfig(filename="active_sessions_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання активних користувачів
def get_active_sessions():
    users = psutil.users()
    session_data = []
    for user in users:
        session_info = {
            "username": user.name,
            "terminal": user.terminal,
            "host": user.host,
            "started": time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime(user.started))
        }
        session_data.append(session_info)
        logging.info(f"Виявлено активний сеанс: {session_info}")

    if not session_data:
        logging.warning("Немає активних сеансів!")
    return session_data

# Функція моніторингу активних сеансів
def monitor_sessions():
    previous_sessions = set()
    while True:
        time.sleep(10)
        active_sessions = get_active_sessions()
        current_sessions = {session["username"] for session in active_sessions}

        new_sessions = current_sessions - previous_sessions
        closed_sessions = previous_sessions - current_sessions

        for session in new_sessions:
            logging.info(f"Новий сеанс підключення: {session}")
        for session in closed_sessions:
            logging.warning(f"Сеанс закрито: {session}")

        previous_sessions = current_sessions

if __name__ == "__main__":
    logging.info("Запуск моніторингу активних сеансів")
    monitor_sessions()
```

Файл suspicious_services.py

```
# Виявлення підозрілих запущених сервісів

import psutil
import logging
import time

# Логування підозрілих сервісів
logging.basicConfig(filename="suspicious_services_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Перелік потенційно небезпечних сервісів
SUSPICIOUS_SERVICES = ["malware_service", "unwanted_process", "hacker_tool",
"ransomware", "keylogger"]

# Функція перевірки запущених сервісів
def check_services():
    suspicious = []
    for proc in psutil.process_iter(attrs=["pid", "name", "cpu_percent",
"memory_percent"]):
        process_name = proc.info["name"].lower()
        if process_name in SUSPICIOUS_SERVICES:
            suspicious.append(proc.info)
            logging.warning(f"Виявлено підозрілий сервіс: {proc.info}")
            if proc.info["cpu_percent"] > 50 or proc.info["memory_percent"] >
30:
                logging.error(f"Сервіс {process_name} використовує занадто
багато ресурсів!")

    if not suspicious:
        logging.info("Підозрілих сервісів не знайдено")

# Функція циклічного моніторингу
def monitor_services():
    while True:
        time.sleep(10)
        check_services()

if __name__ == "__main__":
    logging.info("Запуск моніторингу сервісів")
    monitor_services()
```

Файл software_update_checker.py

```
# Перевірка оновлень програмного забезпечення

import subprocess
import logging
import time

# Логування перевірки оновлень
logging.basicConfig(filename="update_checker_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція перевірки оновлень
def check_updates():
    try:
        result = subprocess.run(["apt", "list", "--upgradable"],
capture_output=True, text=True)
        updates = result.stdout.strip()
        if updates:
            logging.info("Є доступні оновлення: \n" + updates)
        else:
            logging.info("Система оновлена")
    except Exception as e:
        logging.error(f"Помилка перевірки оновлень: {str(e)}")

# Функція періодичної перевірки оновлень
def monitor_updates():
    while True:
        time.sleep(3600)
        check_updates()

if __name__ == "__main__":
    logging.info("Запуск перевірки оновлень")
    monitor_updates()
```

Файл real_time_threat_analysis.py

```
# Аналіз кіберзагроз у реальному часі

import psutil
import logging
import time

# Логування загроз у реальному часі
logging.basicConfig(filename="threat_analysis_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція перевірки підозрілих запитів
def analyze_real_time_threats():
    previous_connections = 0
    while True:
        time.sleep(5)
        connections = psutil.net_connections(kind='inet')
        connection_count = len(connections)

        if connection_count > 100:
            logging.warning(f"Висока кількість активних підключень:
{connection_count}, можливий DDoS!")
            if connection_count > previous_connections * 1.5:
                logging.error("Різке зростання підключень! Висока ймовірність
DDoS-атаки!")

        else:
            logging.info(f"Кількість активних підключень: {connection_count}")

        previous_connections = connection_count

if __name__ == "__main__":
    logging.info("Запуск аналізу кіберзагроз у реальному часі")
    analyze_real_time_threats()
```

Файл botnet_detection.py

```
# Виявлення бот-мереж у мережі

import psutil
import logging
import time
import socket

# Логування виявлення бот-мереж
logging.basicConfig(filename="botnet_detection_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання активних підключень
def get_network_connections():
    connections = psutil.net_connections(kind="inet")
    conn_list = []
    for conn in connections:
        if conn.status == "ESTABLISHED":
            conn_info = {
                "local_address": conn.laddr,
                "remote_address": conn.raddr,
                "status": conn.status
            }
            conn_list.append(conn_info)
            logging.info(f"Виявлено підключення: {conn_info}")

    if not conn_list:
        logging.info("Немає активних підключень.")
    return conn_list

# Функція виявлення підозрілих IP
def detect_botnet():
    suspicious_ips = ["192.168.1.100", "203.0.113.50", "198.51.100.75",
"185.220.101.1"]

    while True:
        time.sleep(5)
        active_connections = get_network_connections()

        for conn in active_connections:
            remote_ip = conn["remote_address"][0] if conn["remote_address"] else
"N/A"
            if remote_ip in suspicious_ips:
                logging.warning(f"Виявлено підозріле підключення: {conn}")
                logging.warning(f"IP-адреса {remote_ip} знаходиться у списку
загроз!")

        logging.info("Перевірка на бот-мережі завершена.")

if __name__ == "__main__":
    logging.info("Запуск виявлення бот-мереж")
    detect_botnet()
```

Файл sql_xss_detection.py

```
# Аналіз SQL-ін’єкцій та XSS-атак

import re
import logging

# Логування виявлення атак
logging.basicConfig(filename="sql_xss_detection_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Шаблони для SQL-ін’єкцій та XSS
SQL_PATTERNS = [r"SELECT .* FROM", r"UNION SELECT", r"INSERT INTO", r"DROP
TABLE", r"OR 1=1"]
XSS_PATTERNS = [r"<script>", r"onerror=", r"alert\(", r"document.cookie",
r"<iframe>"]

# Функція перевірки вхідних даних
def check_for_sql_xss(input_string):
    is_sql = any(re.search(pattern, input_string, re.IGNORECASE) for pattern in
SQL_PATTERNS)
    is_xss = any(re.search(pattern, input_string, re.IGNORECASE) for pattern in
XSS_PATTERNS)

    if is_sql:
        logging.warning(f"Виявлена SQL-ін’єкція: {input_string}")
        return "SQL Injection Detected!"
    if is_xss:
        logging.warning(f"Виявлено XSS-атаку: {input_string}")
        return "XSS Attack Detected!"

    logging.info(f"Безпечний ввід: {input_string}")
    return "Safe Input"

if __name__ == "__main__":
    logging.info("Запуск аналізу SQL-ін’єкцій та XSS-атак")
    test_inputs = ["SELECT * FROM users", "<script>alert('XSS!')</script>",
"DROP TABLE accounts", "admin' OR '1'='1"]

    for test in test_inputs:
        check_for_sql_xss(test)
```

Файл X dashboard_visualization.py

```
# Створення дашбордів для візуалізації загроз

import matplotlib.pyplot as plt
import logging
import random

# Логування створення дашборду
logging.basicConfig(filename="dashboard_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція генерації випадкових даних загроз
def generate_threat_data():
    threat_types = ["Malware", "DDoS", "Phishing", "Ransomware", "Spyware"]
    threat_counts = [random.randint(10, 100) for _ in threat_types]

    logging.info(f"Згенеровані дані загроз: {dict(zip(threat_types,
threat_counts))}")
    return threat_types, threat_counts

# Функція побудови графіку
def create_dashboard():
    labels, values = generate_threat_data()
    plt.bar(labels, values, color=["red", "blue", "green", "purple", "orange"])
    plt.xlabel("Тип загрози")
    plt.ylabel("Кількість виявлених атак")
    plt.title("Аналіз кіберзагроз")
    plt.grid(axis="y", linestyle="--")
    plt.show()

if __name__ == "__main__":
    logging.info("Запуск візуалізації загроз")
    create_dashboard()
```

Файл config_security_checker.py

```
# Виявлення слабких місць у конфігураціях системи

import os
import logging

# Логування перевірки конфігурації
logging.basicConfig(filename="config_security_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція перевірки конфігураційних файлів
def check_config_files():
    insecure_permissions = []

    config_paths = ["/etc/passwd", "/etc/shadow", "/etc/ssh/sshd_config",
"/etc/nginx/nginx.conf"]

    for path in config_paths:
        if os.path.exists(path):
            permissions = oct(os.stat(path).st_mode)[-3:]
            if permissions != "600":
                insecure_permissions.append((path, permissions))
                logging.warning(f"Небезпечні права доступу: {path}
({permissions})")

    if not insecure_permissions:
        logging.info("Конфігураційні файли безпечні")
    else:
        logging.warning("Знайдено файли з небезпечними правами доступу!")

if __name__ == "__main__":
    logging.info("Запуск перевірки конфігураційних файлів")
    check_config_files()
```

Файл X.py

```
# Аналіз ризиків сторонніх підключень

import psutil
import logging
import time
import socket

# Логування зовнішніх підключень
logging.basicConfig(filename="external_connections_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання активних підключень
def get_external_connections():
    connections = psutil.net_connections(kind="inet")
    conn_list = []
    for conn in connections:
        if conn.status == "ESTABLISHED":
            remote_ip = conn.raddr[0] if conn.raddr else "N/A"
            conn_info = {
                "local_address": conn.laddr,
                "remote_address": remote_ip,
                "status": conn.status
            }
            conn_list.append(conn_info)
            logging.info(f"Зовнішнє підключення: {conn_info}")

    return conn_list

# Функція виявлення підозрілих зовнішніх IP
def detect_suspicious_connections():
    blacklist = ["203.0.113.50", "198.51.100.75", "185.220.101.1"]

    while True:
        time.sleep(5)
        active_connections = get_external_connections()

        for conn in active_connections:
            if conn["remote_address"] in blacklist:
                logging.warning(f"Підозріле підключення:
{conn['remote_address']} в чорному списку!")

        logging.info("Аналіз сторонніх підключень завершено.")

if __name__ == "__main__":
    logging.info("Запуск моніторингу сторонніх підключень")
    detect_suspicious_connections()
```

Файл rdp_ssh_monitor.py

```
# Виявлення віддалених підключень RDP/SSH

import psutil
import logging
import time

# Логування віддалених підключень
logging.basicConfig(filename="rdp_ssh_monitor_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція перевірки активних RDP/SSH сесій
def check_rdp_ssh():
    suspicious_ports = [22, 3389]
    active_connections = psutil.net_connections(kind='inet')

    for conn in active_connections:
        if conn.laddr.port in suspicious_ports and conn.status == "ESTABLISHED":
            logging.warning(f"Активне віддалене підключення: {conn.laddr} ->
{conn.raddr}")

if __name__ == "__main__":
    logging.info("Запуск моніторингу RDP/SSH")
    while True:
        time.sleep(10)
        check_rdp_ssh()
```

Файл critical_files_monitor.py

```
# Контроль змін у критичних файлах системи

import os
import hashlib
import logging
import time

# Логування змін у критичних файлах
logging.basicConfig(filename="critical_files_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Критичні файли для моніторингу
CRITICAL_FILES = ["/etc/passwd", "/etc/shadow", "/etc/ssh/sshd_config"]

# Функція обчислення хешу файлу
def get_file_hash(file_path):
    try:
        with open(file_path, "rb") as f:
            return hashlib.sha256(f.read()).hexdigest()
    except Exception as e:
        logging.error(f"Помилка читання файлу {file_path}: {str(e)}")
        return None

# Функція моніторингу змін
def monitor_critical_files():
    file_hashes = {file: get_file_hash(file) for file in CRITICAL_FILES}

    while True:
        time.sleep(10)
        for file in CRITICAL_FILES:
            new_hash = get_file_hash(file)
            if new_hash and new_hash != file_hashes[file]:
                logging.warning(f"Виявлено зміну у файлі {file}!")
                file_hashes[file] = new_hash

if __name__ == "__main__":
    logging.info("Запуск моніторингу критичних файлів")
    monitor_critical_files()
```

Файл usb_device_monitor.py

```
# Виявлення сторонніх USB-пристроїв

import subprocess
import logging
import time

# Логування USB-пристроїв
logging.basicConfig(filename="usb_device_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція отримання списку USB-пристроїв
def list_usb_devices():
    try:
        result = subprocess.run(["lsusb"], capture_output=True, text=True)
        devices = result.stdout.strip().split("\n")
        logging.info(f"Підключені USB-пристрої:\n" + "\n".join(devices))
        return devices
    except Exception as e:
        logging.error(f"Помилка отримання USB пристроїв: {str(e)}")
        return []

# Функція моніторингу USB
def monitor_usb():
    known_devices = list_usb_devices()

    while True:
        time.sleep(10)
        current_devices = list_usb_devices()
        new_devices = set(current_devices) - set(known_devices)

        if new_devices:
            for dev in new_devices:
                logging.warning(f"Новий USB-пристрій виявлено: {dev}")

if __name__ == "__main__":
    logging.info("Запуск моніторингу USB-пристроїв")
    monitor_usb()
```

Файл penetration_test.py

```
# Автоматичне тестування на проникнення

import subprocess
import logging

# Логування пентесту
logging.basicConfig(filename="penetration_test_log.txt", level=logging.INFO,
format="%(asctime)s - %(message)s")

# Функція виконання сканування портів
def run_port_scan(target):
    try:
        result = subprocess.run(["nmap", "-p", "1-1024", target],
capture_output=True, text=True)
        logging.info(f"Результати сканування:\n{result.stdout}")
    except Exception as e:
        logging.error(f"Помилка виконання пентесту: {str(e)}")

if __name__ == "__main__":
    logging.info("Запуск тестування на проникнення")
    run_port_scan("127.0.0.1")
```