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
“____” _____ 2026 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix”

Виконав здобувач вищої освіти
IV курсу, групи КН-22
ОПП «Комп’ютерні науки»
спеціальності 122 «Комп’ютерні науки»
_____ Бомко Д.М.
«____» _____ 2026 р.

Керівник проекту
доктор технічних наук, професор
_____ Смірнов О.А.
«____» _____ 2026 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 122 *"Комп'ютерні науки"*

Освітньо-професійна (освітньо-наукова) програма _*"Комп'ютерні науки"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 6 » лютого 2026 року

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

Бомку Данилу Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix*

2. Керівник роботи *Смірнов Олексій Анатолійович. докт. техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 116-02 від 06.02.2026 року

3. Строк подання студентом роботи до захисту *23.05.2026 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix*

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 в промислову експлуатацію.
6. Висновки

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

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

7. Дата видачі завдання « 6 » лютого 2026 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	10.03.2026 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2026 р.	
3.	Розробка моделі компонента	20.03.2026 р.	
4.	Розробка структур даних	25.03.2026 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2026 р.	
6.	Програмування алгоритмів	10.04.2026 р.	
7.	Оформлення ПЗ	17.04.2026 р.	
8.	Попередній захист роботи	23.05.2026 р.	

Дата видачі завдання
« 6 » лютого 2026 р.

Підпис керівника
_____ Смірнов О.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 6 » лютого 2026 р.

Підпис здобувача
_____ Бомко Д.М.
(прізвище та ініціали)

АНОТАЦІЯ

Бомко Д.М. Програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix. 122 Комп'ютерні науки. Центральноукраїнський національний технічний університет. Кропивницький. 2026.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

Метою розробки є програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

Результат роботи – програмна реалізація системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

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 Visual C++.

Ключові слова: комп'ютерні науки, Cloud Controls Matrix

ABSTRACT

Bomko D.M. Software for the intelligent data storage system in the cloud using Cloud Controls Matrix. 122 Computer Science. Central Ukrainian National Technical University. Kropyvnytskyi. 2026.

In this final qualification work for the first (bachelor's) level of higher education, software has been developed, which is intended for the intelligent data storage system in the cloud using Cloud Controls Matrix.

The purpose of the development is the software for the intelligent data storage system in the cloud using Cloud Controls Matrix.

The result of the work is the software implementation of the intelligent data storage system in the cloud using Cloud Controls Matrix.

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 a PC with OS Windows 10/11.

The program was developed in the Visual C++ environment.

Keywords: computer science, Cloud Controls Matrix

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЛОМ	–	локальна обчислювальна мережа
MME	–	міжмережні екрани
ATM	–	асинхронний режим передачі
BSD	–	адаптована для Internet реалізація операційної системи UNIX
ICMP	–	міжмережний протокол управляючих повідомлень
IP	–	Internet Protocol – міжмережний протокол
NFS	–	мережна файлова система
PPP	–	протокол передачі від точки до точки
RFC	–	опис набору протоколів Internet
RPC	–	віддалений виклик процедури
SLIP	–	міжмережний протокол для послідовного каналу
SMTP	–	Simple Mail Transfer Protocol – простий протокол передачі пошти
TCP	–	Transmission Control Protocol – протокол управління передачею
UDP	–	User Datagram Protocol – протокол користувальницьких датаграм
UNIX	–	багатозадачна операційна система
UTP	–	незахищена вита пара
URL	–	уніфікований покажчик інформаційного ресурсу

ВСТУП

Актуальність теми. Некомерційна організація CSA створила це як групу панелей безпеки для управління повноваженнями, управління ризиками та дотримання вимог. За допомогою Cloud Controls Matrix (Матриця хмарних елементів керування, CCM) підприємства та постачальники послуг зв'язку (CSP) можуть гарантувати, що вони вживають необхідних заходів для створення та використання безпечних мережевих умов, застосовуючи відповідні внутрішні засоби контролю, адаптовані для досягнення цілей конфіденційності та управління ризиками. CSA розробила ці засоби захисту, щоб допомогти компаніям закуповувати послуги мережевих обчислень, які відповідають або перевершують стандарти, встановлені галуззю.

CSA – це благодійна організація, яка надає фінансову підтримку дослідженням, що вивчають можливість перенесення знань, отриманих з мережевої безпеки, на інші види фреймворків. CSA спирається на досвід своїх фахівців, асоціацій, урядів та корпоративних членів, щоб надавати аналіз, освіту, дипломи, заходи та продукти, орієнтовані на безпеку. Постачальники послуг зв'язку, кінцеві користувачі, власники бізнесу та законодавці можуть отримати користь від роботи та розуміння асоціації. CSA надає простір для збору численних учасників світу технологій для співпраці над створенням та підтримкою безпечного середовища для своїх послуг. Це додаткова перевага, яку пропонує CSA. Торгова група допомагає CSP вирішувати питання безпеки в моделях доставки програмного забезпечення та навчає компанії на різних етапах впровадження хмарних технологій передовим практикам. Коли справа доходить до покращення конфіденційності мережі, кожен може приєднатися до CSA, якщо він має необхідні навички та знання, щоб запропонувати щось корисне.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

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

– Огляд існуючих систем інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

– Дослідження системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

– Програмна реалізація системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix, 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

Провайдери по-різному підходять до питанню розкриття інформації про застосовувані міри безпеки. «Security by obscurity (закритість інформації як спосіб захисту) – це неправильний підхід до організації захисту. У такому випадку система не буде безпечної тільки тому, що ніхто не знає, що вона собою представляє. Провайдеру немає особливого змісту зберігати таємність, щоб не давати зайвої інформації потенційним зловмисникам: З погляду забезпечення захисту абсолютно не важливо, чи знає атакуючий, який саме міжмережевий екран установлений. Головне, як він налаштований, як часто обновляється, як адмініструється й т.д.

Тим часом багато упереджень можуть бути зняті, якщо провайдер дотримується відкритої політики в питаннях безпеки й надає інформацію про те, які міри фізичного захисту він реалізує, які засоби інформаційної безпеки використовує, як здійснюється резервне копіювання й відновлення, обробляються інциденти й т.д. Вжиті заходи стануть додатковим свідченням надійності провайдеру.

Cloud Controls Matrix

Разом з тим проблема часто полягає в нездатності замовника правильно оцінити ризики. Щоб коректно це зробити, компанії можуть скористатися готовими шаблонами – наприклад Cloud Controls Matrix (CCM) від Cloud Security Alliance (CSA), де приводяться різні контрольні питання відносно:

- кадрової політики;
- контролю доступу;
- фізичного захисту;
- процедур;

– і т.д.

У рамках програми добровільної сертифікації альянс приводить на сайті перелік провайдерів, які заповнили цей опитувальник за власною ініціативою; правда, українських компаній серед них небагато.

Одним з показників захищеності комерційного ЦОДу є наявність у нього сертифіката на відповідність вимогам безпеки, наприклад PCI DSS. Хоча цей стандарт стосується тільки тих, хто займається обробкою даних платіжних карт, у ньому сформульовані загальні вимоги як до інфраструктури, так і до організаційних процедур. Сертифікація ЦОДів – стратегічне питання. Дана міра дозволяє підвищити конкурентну привабливість пропозиції і якість надаваних послуг.

Для одержання сертифіката довелося проробити більшу роботу, у якій брали участь більше 20 чоловік. Для виявлення будь-яких аномалій у всіх пристроях оточення ЦОДу була встановлена система керуваннями подіями й даними інформаційної безпеки IBM Qradar (наявність системи SIEM є обов'язковою вимогою). Реалізація даного проекту виявила 26 помилок, зокрема невідключені пристрої telnet, що дозволило підвищити рівень безпеки інфраструктури.

Без компетентної команди експертів всі розмови про безпеку не мали б змісту – наявність експертів по Linux, UNIX, Microsoft, Oracle, а також SAP і Cisco. Вони дозволяють зробити комплексні рішення для будь-якого запиту. Багаторічний досвід роботи зі створення безпеки власних ЦОДів дозволяє знайти правильні й ефективні рішення для захисту систем замовника.

Одним з показників захищеності комерційного ЦОДу є наявність у нього сертифіката на відповідність вимогам PCI DSS

1.2 Область застосування

Областю застосування є SIEM-системи. Security Information and Event Management є складним і дорогим рішенням по зборі, нормалізації, аналізу й кореляції інформації з лог-файлів всіх IT-систем, однак і результати її роботи при правильній експлуатації є видатними. При розгортанні SIEM-системи виникає ряд питань, на які дамо відповідь у даному розділі.

Як ми буде здійснювати налаштування й використання своєї SIEM?

SIEM повноцінно працює лише в руках професіоналів і, будучи встановленої у великій організації, може зажадати команду з 8 виділених співробітників. Така складна система без обслуговуючого персоналу схожа на ненаселену міцність – на вид неприступна, але не є перешкодою атакуючий. SIEM не заміняє відділ Інформаційної Безпеки, а є інструментом, що вимагає висококваліфікованих фахівців, для досягнення значимих результатів.

Переконаєтеся, що ваша організація здатна використовувати SIEM. Чи є необхідні ресурси й персонал? Чи зможете ви найняти й навчити нових співробітників? Якщо “ні”, те, можливо, кращим рішенням для вас буде розглянути послуги від сторонніх сервісних організацій (інтегратори й/або MSSP).

Що моя організація очікує одержати від впровадження SIEM?

Це здається очевидним, але ви повинні знати вимоги, вибираючи собі SIEM або аналітичну систему безпеки. Розставте пріоритети вимог Бізнесу й Відділу Безпеки перед початком процесу тестування й оцінки систем. Які системи будуть джерелами логів? Потрібно чи збір подій у режимі реального часу? Чи всі логи потрібно збирати, або ж тільки із критичних підсистем? Що потрібно архівувати і як довго зберігати? Як будуть використовуватися зібрані дані – для розслідувань? пошуку уразливостей? аудиту й перевірки на відповідність стандартам?

Потрібно чи нам повноцінне рішення, або досить лише системи обробки логів?

Можливості SIEM рішень вражають, але це недешеве задоволення для Бізнесу й, до того ж, складне в обслуговуванні. Якщо ви заглядаєтеся на саму “круту” систему, але не замислювалися ще про спосіб написання/одержання “крутих” Use Case’ов до неї [скрипти, правила, візуалізація], то вам варто переглянути ваш підхід.

Наприклад, багато вимог відповідності стандартам безпеки можуть бути легко виконані “простий” системою log management (збір, зберігання, аналіз і можливість пошуку). Тому якщо основне ваше завдання саме обробка логів, а не кореляція подій безпеки – не купуйте надлишкове рішення.

Потрібно традиційний SIEM або аналіз безпеки в Big Data?

Системи обробки більших масивів даних і пошуку схованих закономірностей завойовують своє місце під сонцем на ринку SIEM. Це дуже ефективні системи, але ще більш складні. Тому вони “по зубах” лише компаніям з достатнім фінансуванням і зрілим і укомплектованим Відділом ІБ, у таких умовах вони здатні виявити всі свої сильні сторони.

Будьте обережні – якщо ваша компанія має ризик не подужати SIEM, те ще менше шансів, що big data security analytics буде нормально працювати.

Скільки грошей їсти можливість витратити на покупку й налаштування системи?

Серйозна SIEM вимагає серйозних грошей. Це й вартість ліцензії на сам продукт (і, можливо, послуг з інтеграції й налаштування), і витрати на супутню IT-Інфраструктуру (бази даних, системи зберігання й т.д.), і витрати на навчання персоналу. Підсумкова вартість легко може досягати сотень тисяч доларів, залежно від зазначених параметрів.

Як варіант економії можна розглянути урізані версії SIEM в іменитих виробників або повноцінні, але недорогі – у нішевих гравців. Ви не одержите якогось просунутого функціонала, але все-таки зможете вирішувати більшу частину завдань для SIEM, що коштують перед відділом інформаційної безпеки.

Як продукт закриє вимоги відповідності стандартам і аудитам ІБ?

Завдання відповідності вимогам різним ІБ-стандартам є однієї з найчастіших причин придбання SIEM. Тому в більшість рішень уже “з коробки” убудована підтримка аудита й звітності по найбільш популярних стандартах, таким як HIPAAS, PCI DSS і SOX. Компанія значно заощадить у часі й ресурсах, використовуючи таку автоматичну звітність SIEM, але попередньо переконаєтеся, що потрібний вам звіт буде в поставці й підходить вам. Які ще передналаштовані звіти є “з коробки”? Які можливості по їхньому самостійному підстроюванню?

Яка експертиза постачальника по розгортанню й налаштуванню? Навчанню персоналу?

Досить великий ризик невдалого впровадження настільки складної системи, як SIEM. У звіті 2025 року аналітик Gartner Оліфер Рочфорд повідомив, що від 20% до 30% клієнтів незадоволені результатами впровадження. А будучи успішно розгорнутої, система SIEM зажадає кваліфікованих безпечників для щоденної роботи з нею. Запитаєте в постачальника, яку підтримку він здатний зробити при впровадженні рішення й, якщо буде потрібно, при навчанні ваших співробітників.

Чи підтримує дана SIEM роботу із хмарами й Big Data платформами? Тобто чи буде рішення, що здобувається сьогодні, працювати із системами, купленими завтра?

Хмарні (Software, Platform, Infrastructure)-As-A-Service продукти й платформи обробки Big Data уже використовуються у вашій організації або почнуть використовуватися буквально завтра. Якщо ви витрачаєте серйозну суму на покупку SIEM сьогодні, то явно хочете бути впевнені можливості інтеграції з новими системами завтра.

Скільки джерел логів уже підтримує SIEM? Наскільки складно підключити новий маловідомий?

SIEM буде неповноцінною, якщо не зможе приймати логи з важливого джерела подій вашої організації. Переконаєтеся, що більшість ваших систем будуть підключені до SIEM штатними засобами, а складність підключення специфічного встаткування (або самописного застосунку) буде не висока.

Основними джерелами логів будуть системи інформаційної безпеки (такі як файєрвол, IPS/IDS, VPN, поштовий сервер, система антивірусного захисту та ін.), а також клієнтські й серверні операційні системи.

Які можливості системи по аналізі даних?

Крім базових функцій по оповіщенню і звітності, SIEM повинна надавати операторові (аналітик відділу ІБ) інструментарій перегляду й аналізу подій у балках для розслідування інциденту й виробітку реакції на нього. Навіть сама розумна й налаштована SIEM-система гірше самого розумного аналітика. Адже система не спрацює, якщо немає відповідного правила, не зможе запідозрити негарне без контексту що відбуває. Тому ручний аналіз всі так само затребуваний і система повинна надавати аналітикові зручний інструментарій. Розширені можливості пошуку й візуалізації даних однозначно сприяють успішності розслідування інциденту.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix, 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

Хмарна система безпеки – це операційна система для хмарних ризиків: стандарти, власність, контроль, докази та правозастосування, що працюють в AWS, Azure, GCP та локально. Найкращі хмарні системи безпеки не зберігаються у форматі PDF. Вони відображаються в рішеннях IAM, перевірках політик, журналах аудиту, чергах виправлення та оглядах архітектури.

Вибір між хмарними системами безпеки трохи схожий на те, як зайти на огляд безпеки, де сім людей говорять сім версій правди.

– CISO хоче управління.
– Хмарний архітектор хоче реалізовані елементи керування.
– Аудитор хоче доказів.
– Відділ закупівель хоче звіт.
– Інженерія хоче знати, яке правило блокує випуск.

Ніхто не помиляється. Вони просто вирішують різні частини однієї й тієї ж проблеми безпеки хмари.

Ось чому зрілі команди рідко «вибирають одну структуру». Вони будують карту моделі контролю: один рівень управління для ризиків, один технічний базовий рівень для посилення, один рівень гарантування для клієнтів або регуляторів та хмарні перевірки під ним.

Структура кібербезпеки NIST 2.0

NIST CSF 2.0 – це той стандарт, який ви використовуєте, коли рада директорів запитує: «Чи розуміємо ми наші кіберризики?». Він організовує результати безпеки за такими напрямками, як управління, ідентифікація, захист,

виявлення, реагування та відновлення. Великим кроком у версії 2.0 є «управління», що робить його природним якорем для структури управління хмарною безпекою: політика, підзвітність, схильність до ризику, нагляд та права прийняття рішень нарешті знаходяться в моделі, а не зависають над нею.

NIST зазначає, що уряд встановлює та контролює стратегію, очікування та політику управління ризиками кібербезпеки.

Кому це потрібно? Керівникам відділів безпеки, командам управління ризиками та власникам хмарних технологій.

Хмарні засоби керування: інвентаризація активів, пріоритизація ризиків, управління ідентифікацією, ризики третіх сторін, реагування на інциденти, стійкість.

Поширена помилка: використання його як списку прапорців. NIST CSF вказує, які результати важливі. Ваша хмарна платформа все одно має перетворити їх на чинні елементи керування AWS, Azure, GCP, Kubernetes та локальні ресурси.

Критичні засоби контролю безпеки CIS v8 + контрольні показники CIS

Є два елементи. CIS Controls v8 надає пріоритетні дії кіберзахисту. CIS Benchmarks надають рекомендації щодо безпечного налаштування для певних технологій, включаючи хмарні платформи, операційні системи, Kubernetes, бази даних та мережеві пристрої.

CIS також використовує групи впровадження:

1. IG1 для основної кібергігієни.

2. IG2 для більш зрілих команд з вищим ризиком.

3. IG3 для організацій, що стикаються з передовими загрозами або суворішими вимогами.

Хто цим користується? Інженерія платформ, DevSecOps, хмарна безпека, інфраструктура та команди, яким швидко потрібні базові технічні показники.

Хмарні елементи керування: багатофакторна автентифікація (MFA),

ведення журналу, безпечна конфігурація, управління вразливостями, інвентаризація, контроль доступу, безперервний моніторинг.

Остерігайтеся пастки: бенчмарки CIS можуть стати небезпечними, якщо їх застосовувати без контексту. Платежеве навантаження, орієнтоване на публічні ринки, та короткочасна «пісочниця» не заслуговують на однакове операційне ставлення.

ISO/IEC 27017 та ISO/IEC 27018

Стандарти ISO 27017 та 27018 призначені для тих випадків, коли «ми сертифіковані за стандартом ISO 27001» недостатньо.

Стандарт ISO/IEC 27017 поширює наведені вказівки ISO з безпеки на середовища хмарних сервісів. Він допомагає уточнити відповідальність між постачальником та клієнтом, що стосується хмарних технологій.

Стандарт ISO/IEC 27018 зосереджений на захисті персональної інформації в публічних хмарних сервісах, особливо коли постачальник хмарних послуг виступає в ролі обробника персональної інформації.

Кому це має бути цікаво? Постачальникам SaaS, постачальникам хмарних послуг, командам корпоративної безпеки, командам конфіденційності та організаціям, що займаються закупівлями.

Хмарні елементи керування включають розділення орендарів, операції адміністрування хмари, моніторинг, безпечне видалення, повернення даних, обов'язки клієнтів та обробку ідентифікаційної інформації.

Поширена помилка: якщо вважати, що ISO 27001 сам по собі доводить зрілість хмарних технологій, то це підтверджує наявність системи управління. Вам все ще потрібна специфічна для хмари сфера застосування, чіткість розподілу відповідальності та докази, пов'язані з фактичними послугами.

Матриця хмарних елементів керування CSA v4 / v4.1

Якщо NIST – це карта управління, то CSA CCM – це хмарна бібліотека керування, яку ви привносите у військову кімнату.

Альянс Cloud Security Alliance описує CCM як систему контролю

кібербезпеки для хмарних обчислень, структуровану за 17 доменами зі 197 цілями контролю. Його новіші матеріали версії 4.1 також поєднують CCM з практичними рекомендаціями щодо моделі спільної відповідальності, що важливо, оскільки збої в хмарі часто починаються з того, що «ми думали, що постачальник цим впорався».

Хто ним користується? Постачальники хмарних послуг, клієнти хмарних послуг, аудитори, команди GRC, архітектори безпеки та організації, що готуються до CSA STAR.

Хмарні засоби керування охоплюють IAM, шифрування, безпеку інфраструктури, безпеку додатків, управління даними, ведення журналу, ланцюг поставок, управління вразливостями та сумісність.

Звичайна помилка: перетворення CCM на електронну таблицю, якою ніхто не керує. Її цінність проявляється, коли кожен елемент керування відповідає активу, власнику сервісу, хмарному обліковому запису, джерелу доказів та шляху виправлення.

Хмарна система безпеки FedRAMP та StateRAMP / GovRAMP

FedRAMP – це шлюз авторизації для хмарних сервісів, що використовуються федеральними агентствами США. GovRAMP, раніше StateRAMP, виконує аналогічну гарантійну роль для державних, місцевих, освітніх та пов’язаних з ними державних покупців.

Обидва побудовані на основі суворого контролю у стилі NIST 800-53, оцінки третьою стороною, статусу авторизації та очікувань постійного моніторингу.

У власних інструкціях FedRAMP щодо постійного моніторингу зазначено, що постачальники хмарних послуг несуть відповідальність за впровадження, підтримку, моніторинг та звітування щодо засобів контролю, задокументованих у їхньому Плані системної безпеки.

Хто повинен ним користуватися? Постачальники хмарних послуг, що продають послуги в охоплених середовищах державного сектору.

Хмарні засоби керування: сканування вразливостей, управління POA&M, реагування на інциденти, шифрування, захист меж, контроль доступу, управління конфігурацією, ведення журналу та безперервне ведення доказів.

Пастка: складання бюджету на документацію, а потім пошук операційного підйому. FedRAMP змінює інженерний ритм, процеси підтримки, реагування на вразливості, збір доказів та управління релізами.

SOC 2 Тип I / Тип II

SOC 2 – це фреймворк, який ваш покупець SaaS запитує одразу після того, як висловлює зацікавленість у демонстрації.

Це не стосується лише хмари, але хмарні компанії працюють у ній, оскільки корпоративні покупці хочуть мати гарантії щодо даних клієнтів. Звіти SOC 2 оцінюють засоби контролю, що стосуються безпеки, доступності, цілісності обробки, конфіденційності та конфіденційності.

– Тип I перевіряє, чи належним чином розроблені засоби контролю в певний момент часу.

– Тип II перевіряє, чи вони дійсно працюють протягом періоду огляду.

Кому це потрібно? SaaS-компаніям, платформам даних, постачальникам API, постачальникам хмарних послуг та будь-якій технологічній компанії, що продає послуги корпоративним закупівлям.

Хмарні засоби контролю часто включають перевірку доступу, управління змінами, шифрування, моніторинг, реагування на інциденти, ризики постачальників, резервне копіювання та доступність.

Поширена помилка: ставлення до SOC 2 як до аудиторського спринту. Тип II винагороджує нудну послідовність: схвалені зміни, перевірений доступ, закриті заявки, оброблені сповіщення, збережені докази.

PCI DSS 4.0 / 4.0.1

Сертифікація PCI DSS надходить до кімнати разом із даними власника картки.

Якщо ваше хмарне середовище зберігає, обробляє або передає дані

платіжних рахунків, сфера застосування PCI стає проблемою проектування, а не лише проблемою відповідності.

Стандарт PCI DSS версії 4.x зберігає основні вимоги до безпеки платежів, але додає більше гнучкості завдяки цілеспрямованому аналізу ризиків та індивідуальним підходам, що звучить непогано, доки ви не зрозумієте, що гнучкість означає сильніше обґрунтування та краще підтвердження.

Хто повинен ним користуватися? Продавці, платіжні системи, SaaS-платформи в процесі платежів, постачальники послуг та будь-яка команда, що має справу з середовищами даних власників карток.

Хмарні засоби керування: сегментація, надійна автентифікація, шифрування, ведення журналу, управління вразливостями, безпечна розробка, обмежений доступ та безперервне тестування.

Пастка полягає в області застосування. Занадто широка, і програма стає болючою. Якщо вона занадто нечітка, ви втрачаєте реальний доступ. У хмарі область застосування PCI знаходиться в мережевих шляхах, межах ідентифікаційних даних, потоках даних, підключених сервісах та доказах.

Хмарна система безпеки для AWS

Хмарна система безпеки AWS поєднує AWS Well-Architected Security Pillar із зовнішнім стандартом, таким як NIST CSF або CIS, для створення перевірених, специфічних для AWS елементів керування. Це важливо, оскільки AWS вже надає вам потужні вбудовані структурні блоки. Найскладніше – перетворити їх на систему керування, яку ваша команда безпеки, інженери платформи, GRC та аудитор можуть інтерпретувати однаково. AWS Well-Architected надає архітектурну призму. NIST CSF забезпечує результати управління. CIS додає посилення дисципліни. Ваші внутрішні політики визначають, що є обов'язковим для виробничих процесів, регульованих даних, робочих навантажень, пов'язаних з Інтернетом, та винятків.

Фраза «хмарна система безпеки AWS» звучить як одне й те саме. На практиці це стек.

AWS Well-Architected Security Pillar: 7 принципів проектування

AWS формує «Пиллар безпеки» навколо захисту даних, систем та активів таким чином, щоб використовувати хмарні можливості для підвищення безпеки. Його сім принципів проектування є корисною відправною точкою, оскільки вони читаються як контрольний список, який кожна хмарна команда хотіла б мати до появи перших 80 облікових записів AWS.

AWS перераховує їх як:

– впровадити міцну основу ідентичності;

– забезпечити відстеження;

– застосовувати безпеку на всіх рівнях;

– автоматизувати найкращі практики безпеки;

– захищати дані під час передачі та в стані спокою;

– тримати людей подалі від даних;

– та підготуватися до заходів безпеки.

Ідентифікація означає IAM, Центр ідентифікації IAM, SCP, межі дозволів та короткочасний доступ. Відстежуваність означає CloudTrail, Config, Security Hub, GuardDuty та зберігання журналів, яке витримує розростання облікових записів. «Тримайте людей подалі від даних» – це доросла версія «припиніть використовувати людей як обробників виробничих даних». Натомість використовуйте автоматизацію, токенізацію, обмежений доступ та контрольовані робочі процеси.

Зіставлення елементів керування NIST CSF та CIS з власними сервісами AWS

AWS Config допомагає оцінювати, перевіряти та аналізувати конфігурації ресурсів, тоді як Resource Explorer допомагає командам шукати та знаходити ресурси в різних облікових записах та регіонах, використовуючи метадані, такі як імена, теги та ідентифікатори.

CloudTrail записує активність користувачів та API, а Security Hub об’єднує сигнали від служб безпеки AWS, щоб надати командам ширше уявлення про

безпеку та відповідність вимогам.

Менеджер інцидентів допомагає координувати плани реагування, ескалації, робочі книги та аналіз після інциденту, коли трапляються критичні збої.

Ваш фреймворк корисний лише тоді, коли він стосується майна. Дізнайтеся, як Cloudaware перетворює хмарні засоби керування на перевірки політик, порушення, докази та виправлення в AWS, Azure, GCP тощо.

2 найкращі практики хмарної безпеки AWS:

1. Розробіть захисні огорожі AWS на організаційному рівні, а потім дозвольте ризикам визначати шаблон облікового запису. Безпека кожного облікового запису виглядає керованою, доки в майні не з’явиться 80 облікових записів, 12 бізнес-підрозділів, три успадковані середовища від придбань та кілька «тимчасових» пісочниць, які тепер стосуються виробничих даних. Почніть з організацій AWS як межі контролю. Використовуйте політики контролю послуг, централізоване ведення журналу, обов'язковий CloudTrail, агрегацію конфігурацій AWS, затверджені регіони, обмеження публічного доступу, налаштування шифрування за замовчуванням та базове тегування, перш ніж окремі команди почнуть створювати. Потім розділіть майно за ризиком. Виробництво не повинно функціонувати під тими ж припущеннями щодо контролю, що й експерименти. Регульовані робочі навантаження повинні мати чіткіше розділення облікових записів, жорсткіші межі IAM, суворіший контроль змін та коротший період винятків. Облікові записи спільних служб потребують спеціального обслуговування, оскільки один слабкий дозвіл може поширитися на всю мережу. Корисний тест простий: чи може новий обліковий запис AWS успадкувати правильні правила ведення журналу, ідентифікації, шифрування, мережі та доказів з першого дня? Якщо ні, то фреймворк залежить від пам'яті. А пам'ять не є контролем.

2. Зробіть так, щоб кожен висновок AWS розповідав повну історію контролю. Невдалої перевірки недостатньо. «Не вдалося знайти центр безпеки»

не допомагає команді платформи визначити пріоритети. «S3 bucket public» – це краще, але все ще неповне рішення. Корисна система хмарної безпеки AWS повинна перетворити цей сигнал на контрольну історію:

– Актив: s3-prod-customer-exports-eu.

– Навколишнє середовище: виробництво.

– Застосування: експорт аналітики клієнтів.

– Власник: команда платформи даних.

– Контроль: публічний доступ до сховища CIS / захист даних NIST Protect.

– Ризик: конфіденційний експорт клієнтів став доступним для громадськості.

– Зміна: політику корзини змінено роллю CI/CD 6 годин тому.

– Дія: заблокувати публічний доступ, переглянути дозвіл на конвеєр, додати докази.

– Доказ: оновлений стан конфігурації, квиток на зміни, сканування після виправлення, запис про затвердження.

Ось у чому різниця.

Сервіси AWS можуть виявляти сигнал. Фреймворк повинен пов'язувати його з власником, серйозністю, бізнес-контекстом, усуненням недоліків та доказами.

Без цього ланцюга команда має сповіщення. З ним у них є робота, яку можна реально перемістити.

Хмарна система безпеки для Azure

Хмарна система безпеки Azure зазвичай базується на методології безпеки Microsoft Cloud Adoption Framework (CAF) та показнику безпеки Azure, що відповідають стандартам NIST CSF або ISO 27001.

Це чиста версія.

– Справжня версія починається, коли компанія має 40 підписок Azure.

– Керівні групи побудували три реорганізації тому.

– Кілька виробничих навантажень, що знаходяться поруч із «тимчасовими» аналітичними проектами.

– А результати дослідження безпеки розподілені між Defender for Cloud, Sentinel, Azure Policy, Entra ID та чиєюсь електронною таблицею.

– Azure надає вам необхідні інгредієнти. Фреймворк підказує команді, як перетворити їх на елементи керування з власниками, захисними поясами, доказами та повторюваними рішеннями.

Структура впровадження хмарних технологій Microsoft: безпечна методологія

Microsoft CAF Secure корисний, оскільки розглядає безпеку як операційну дисципліну, а не як патч аудиту на пізній стадії. Microsoft стверджує, що в проектуванні цільової зони Azure безпеку слід враховувати протягом усього процесу, при цьому область проектування створює основу для середовищ Azure, гібридних та багатохмарних середовищ.

Це також вказує командам на CAF Secure для отримання глибших інструкцій щодо процесу безпеки та інструментів.

Шість дисциплін безпеки надають хмарним лідерам практичну форму:

– Контроль доступу: Microsoft Entra ID, умовний доступ, PIM, RBAC, керовані ідентифікаційні дані, мінімальні привілеї.

– Сучасні операції безпеки: Microsoft Defender for Cloud, Sentinel, маршрутизація сповіщень, реагування на інциденти, інженерія виявлення.

– Безпека інфраструктури: віртуальні мережі, групи підтримки мережі (NSG), приватні кінцеві точки, брандмауер Azure, захист від DDoS-атак, сегментація.

– Безпека розробки: безпечні конвеєри, перевірки IaC, управління секретами, ідентифікація робочого навантаження, контроль релізів.

– Безпека даних: сховище ключів, шифрування, класифікація даних, доступ до сховища, аудит бази даних.

– Захист активів: гігієна підписок, тегування, інвентаризація, контекст

вразливостей, захист від резервного копіювання.

Пастка полягає в тому, щоб розглядати CAF як «найкращі практики Azure». Він корисніший як операційна модель для того, хто чим керує.

Тест безпеки хмарних технологій Microsoft (MCSB)

Ось деталь назви, яку команди пропускають: Microsoft Cloud Security Benchmark є наступником Azure Security Benchmark. Microsoft перейменувала ASB на MCSB у жовтні 2022 року та розширила напрямок до ширшої системи контролю безпеки хмари, узгодженої з такими стандартами, як CIS, NIST та PCI.

Чому це має значення?

Оскільки MCSB допомагає перекласти знайому мову керування на інструкції з впровадження, специфічні для Azure, MCSB версії 2 від Microsoft зіставляє елементи керування з NIST SP 800-53 Rev. 5, PCI DSS версії 4, CIS Controls версії 8.1, NIST CSF версії 2.0, ISO 27001:2022 та SOC 2, водночас попереджаючи, що ці зіставлення показують часткове або повне технічне покриття, а не автоматичну відповідність.

Цей останній фрагмент важливий.

Призначення політики Azure може підтримувати елемент керування PCI або NIST. Це не магічно доводить відповідність усього елемента керування. Вам все одно потрібні область застосування, право власності, перевірка, винятки та докази.

Зіставлення елементів керування фреймворку з нативними службами Azure

Практична система хмарної безпеки для Azure не повинна обмежуватися лише фразою «використовуйте інструменти безпеки Microsoft». Вона повинна відповідати на запитання, що саме має доводити кожен інструмент.

Уявіть собі це як ланцюжок керування. Azure Policy повідомляє, чи відповідає конфігурація правилу. Defender for Cloud додає рекомендації щодо безпеки та контекст ризиків. Entra ID та умовний доступ пояснюють, хто що може робити. Sentinel перетворює телеметрію на розслідування та реагування.

Key Vault доводить контроль над ключами, але лише якщо журнали власності, ротації, доступу та аудиту є частиною історії доказів.

Знахідка без цього ланцюга – це лише сигнал. Корисний, але неповний.

Azure Landing Zone як основа фреймворку

Зони посадки Azure – це точка, в якій фреймворк переходить від теорії до практичного застосування.

Microsoft рекомендує планувати політики підписок перед створенням підписок, використовувати групи керування для масштабного управління, забезпечувати управління за допомогою політики Azure та розділяти виробничі, невиробничі та ізольовані середовища на різні підписки.

Саме про це повинні турбуватися хмарні архітектори.

Автономний продаж підписок пришвидшує роботу команд, не пропонуючи їм порожню хмару. Групи управління створюють успадкування. Політика Azure застосовує захисні бар'єри, перш ніж команди відхиляються. Окремі цільові зони запобігають тому, щоб виробнича, розробницька/тестова та ізольована частини мали однаковий профіль ризику.

Один практичний приклад:

Нова команда розробки застосунку запитує підписку. Команда розробки платформи продає її під відповідною групою керування. Політика Azure застосовує обов'язкове ведення журналу, дозволені регіони, теги, правила приватних кінцевих точок та покриття Defender. Групи Entra та RBAC призначаються у відповідній області. Служба безпеки може бачити робоче навантаження. GRC може зіставляти елементи керування. Інженерія може створювати систему без узгодження кожного захисного бар'єру з нуля.

Хмарна система безпеки для SaaS

Хмарна система безпеки для SaaS має працювати з хмарою, яку ви не повністю контролюєте. Це найнезручніша частина.

В AWS, Azure або GCP ваша команда зазвичай може звести ризик до ресурсу: екземпляра, корзини, підмережі, політики, ключа. SaaS – це інша справа.

Активом може бути клієнт Workday, інтеграція Salesforce, робоча область Slack, грант OAuth Google Workspace, роль адміністратора Rippling або сторінка Notion, синхронізована через сторонній додаток, схвалення якого ніхто не пам’ятає.

Площина керування – це не лише SaaS-додаток.

Це постачальник ідентифікаційних даних. SCIM. OAuth. Ролі адміністратора. Токени API. Налаштування постачальника. Журнали аудиту. Експорт даних. Сеанси браузера. Доступ підрядника. Зовнішній спільний доступ.

Ось чому SaaS потребує власного фреймворку. Чому SaaS потребує власної фреймворкової лінзи. Ризик SaaS рідко виникає з миготливих вогнів. Зазвичай він виникає через зручність.

Команда реєструється для використання інструменту. Хтось підключає SSO. Хтось інший схвалює доступ OAuth, оскільки інтеграція має «читати дані календаря». Налаштування SCIM увімкнено, але деініціалізацію налаштовано лише наполовину. Адміністратори додаються під час налаштування.

Шість місяців по тому, додаток містить експортовані дані клієнтів, записи співробітників, контракти, заявки на підтримку, платіжні дані, фрагменти вихідних кодів та докази відповідності.

У цьому випадку «тіньові IT» – це ще не все. Краще використовувати слово «тіньовий доступ». SSPM, або SaaS Security Posture Management, допомагає, оскільки SaaS має іншу поверхню контролю:

– Гранти OAuth із широкими правами на читання/запис.
– локальні користувачі, обходячи SSO.
– неактивні адміністратори після змін у команді.
– відсутнє деініціалізування SCIM.
– зовнішній обмін файлами, каналами, дошками або записами.
– журнали аудиту, які не експортуються до журналу доказів безпеки.
– Токени API без власника або терміну дії.
– SaaS-додатки, підключені до постачальника ідентифікаційних даних, але не перевірені службою безпеки.

Фреймворки, найбільш релевантні для SaaS

Не кожен стандарт чітко бачить SaaS.

– SOC 2 зазвичай є першочерговим для SaaS-компаній, оскільки корпоративні покупці розуміють його. Вони хочуть знати, чи ваші засоби контролю безпеки, доступності, конфіденційності, цілісності обробки та конфіденційності дійсно працюють з часом.

– ISO 27001 надає рівень системи управління. ISO 27017 додає рекомендації, специфічні для хмарних технологій, що допомагає, коли право власності на контроль розподіляється між постачальником SaaS та клієнтом.

– CSA CCM стає сильнішим, коли нерухомість стає хмарною сучасним способом: SaaS, підключений до IaaS, ідентифікації, API, конвеєрів даних та сторонніх постачальників.

– NIST CSF надає керівництву операційну модель. Управління. Виявлення. Захист. Виявлення. Реагування. Відновлення. Достатньо чиста для керівників, достатньо гнучка для архітекторів безпеки.

Практичні питання контролю SaaS гостріші, ніж «Чи відповідаємо ми вимогам?»

Запитайте це замість цього:

– Які SaaS-додатки схвалюються, толеруються або блокуються?

– Які з них підключаються до Microsoft Entra ID, Okta, Google Workspace або іншого постачальника ідентифікаційних даних?

– Де ввімкнено SCIM, а де депровізіонізація все ще залежить від людини?

– Які гранти OAuth можуть читати файли, експортувати дані, редагувати записи або керувати користувачами?

– Які орендарі містять регульовані, клієнтські, фінансові, вихідні або співробітникові дані?

– Чи можна експортувати та зберігати журнали аудиту?

– Хто володітиме інтеграцією після того, як початковий адміністратор піде?

Хмарна система безпеки HR: захист Workday, BambooHR, Rippling

Хмарна система безпеки HR не є окремим глобальним стандартом. Це призма управління SaaS, що застосовується до HR-платформ, таких як Workday, BambooHR, Rippling, HiBob, Personio, Deel, Greenhouse або Lever.

HR SaaS заслуговує на суворіший підхід, оскільки часто має найщільніший стек PII в компанії.

Офіційні імена. Домашні адреси. Зарплати. Податкові ідентифікаційні номери. Банківські реквізити. Контракти. Довідки про результати роботи. Відпустки. Пільги. Імміграційні документи. Контакти на випадок надзвичайних ситуацій. Іноді інформація, пов'язана зі здоров'ям. Іноді дані перевірки біографічних даних.

Це не «просто ще один бізнес-додаток».

Для HR SaaS базові показники мають бути більш чіткими:

– Застосування єдиного входу (SSO): звичайні користувачі не повинні покладатися на локальні паролі. Для доступу до системи "розбиття скла" потрібен призначений власник, моніторинг та перевірка.

– Доступ на основі ролей: відділи нарахування заробітної плати, адміністратори кадрів, рекрутери, менеджери, фінанси та IT-спеціалісти не повинні успадковувати широкі дозволи для зручності.

– Контроль життєвого циклу SCIM: робочі процеси приєднання, переведення та звільнення повинні працювати чітко. Звільнений працівник не повинен залишатися активним через те, що системи відділу кадрів та IT пропустили синхронізацію.

– Експорт журналу аудиту: події входу, зміни ролей, редагування записів співробітників, експорт даних, дії інтеграції та дії адміністратора повинні залишати портал постачальника та потрапляти до сліду доказів.

– Місцезнаходження та збереження даних: знайте, де зберігаються дані співробітників, де вони обробляються та як довго вони залишаються після звільнення.

– Перевірка субпідрядників: HR-платформи часто покладаються на нарахування заробітної плати, пільги, перевірку біографічних даних, електронний підпис, аналітику та постачальників підтримки. Цей ланцюг має значення.

– Керування OAuth та API: для інтеграцій потрібні обмежені дозволи, іменовані власники, дати перевірки та видалення після завершення бізнес-випадку використання.

Фінальний тест простий. Чи може ваша команда довести, хто мав доступ до даних про компенсацію, яка інтеграція експортувала записи співробітників, чи був користувач активним, яка роль дозволяла доступ і який елемент керування перевіряв цей дозвіл?

Якщо ні, то HR SaaS ще не регулюється. Йому довіряють. Це дуже різні речі.

Стандарти хмарної безпеки постачальників та постачальників послуг

Не кожна корисна система хмарної безпеки походить від NIST, ISO, CIS або CSA. Іноді найпрактичніші шаблони походять від постачальників, яким довелося забезпечити безпеку великих, розподілених, ретельно перевірених середовищ, перш ніж решта з нас мали чисті назви для цього безладу.

Я б не копіював модель постачальника сліпо. Cisco, Google, AWS та Microsoft мають різні архітектури, команди, телеметрію та профілі ризиків. Але їхні опубліковані фреймворки варті вивчення, оскільки вони показують, як поводяться зрілі програми хмарної безпеки, коли теорія зустрічається з виробництвом: елементи керування відображаються, докази використовуються повторно, ідентифікація стає центральною, а безпека наближається до інженерії.

Стандарти безпеки хмарної інфраструктури Cisco

Структура керування хмарними системами Cisco – це рідкісна корпоративна структура, опублікована постачальниками, яка заслуговує на серйозний розгляд поряд з NIST CSF та CSA CCM. Cisco описує CCF як консолідований набір міжнародних та національних вимог до дотримання вимог безпеки та сертифікації, об'єднаних в одну структуру з відображеннями

контролю, інструкціями з впровадження та артефактами аудиту. Це найцікавіша частина – схема роботи «побудувати один раз, відобразити багато разів». Для керівників у сфері хмарної безпеки ця модель корисна, оскільки відповідність вимогам рідко надходить по одній платформі за раз.

SOC 2 вимагає доказів. ISO хоче структуру контролю. NIST хоче мову ризиків. PCI DSS хоче дисципліну щодо обсягу. Вимоги DORA, NIS2 та регіональні вимоги додають більше тиску. Такий контроль, як перевірка привілейованого доступу, не повинен перетворюватися на сім окремих ритуалів. Він має стати одним керованим контролем з одним власником, одним робочим процесом, одним списком доказів та кількома зіставленнями.

Ширша довідкова історія Cisco з хмарної безпеки також торкається двох практичних сфер:

– Безпечне робоче навантаження зосереджене на видимості робочого навантаження та забезпеченні дотримання політик у гібридних та багатохмарних середовищах, включаючи робочі навантаження на основі базового обладнання, віртуалізованих, хмарних та контейнерних ресурсів.

– Secure Cloud Insights, який зараз у матеріалах Cisco називається Attack Surface Management, був спрямований на видимість активів, аналіз стану, картування взаємозв'язків та зниження ризиків відповідності. Cisco також опублікувала повідомлення про закінчення продажу та підтримки, тому у 2026 році безпечніше розглядати його як довідковий шаблон, ніж як рекомендацію щодо нового інструменту.

Урок такий: чи може ваша команда хмарної безпеки зіставити один невдалий елемент керування в AWS, Azure, GCP, SaaS, контейнерах та локальних середовищах, не перебудовуючи історію доказів щоразу?

Інші фреймворки, опубліковані постачальниками, про які варто знати

BeyondProd від Google – це той інструмент, який варто вивчати, коли ваша архітектура виходить за межі комфорту периметра. Google описує BeyondProd як

хмарну архітектуру сервісів та елементів керування, що захищають робочі навантаження, зокрема те, як відбуваються зміни коду та як здійснюється доступ до даних користувачів.

Ідея полягає в ідентифікації сервісів, ізоляції робочого навантаження, контрольованих змінах, межах доступу до даних та перевірках безпеки, ближчих до поведінки у виробництві.

Це важливо для команд, які використовують Kubernetes, мікросервіси, сервісну мережу, неперервну інтеграцію/комп'ютерну підтримку (CI/CD), ідентифікацію робочого навантаження та внутрішні API. BeyondProd надає архітекторам безпеки корисну ментальну модель: довіра до виробництва має бути заслужена на основі ідентифікації сервісу, походження коду, контексту виконання та дотримання політик, а не на основі мережевого розташування.

SLSA належить до шухляди ланцюга поставок, але, чесно кажучи, ця шухляда тепер є частиною хмарної безпеки. SLSA визначає систему стандартів та контролю для запобігання несанкціонованому втручанню, покращення цілісності артефактів програмного забезпечення та захисту пакетів й інфраструктури.

Використовуйте його, коли слабким місцем є не сам хмарний ресурс, а те, що в нього постачається: образи контейнерів, конвеєри збірки, залежності, артефакти, походження та цілісність випуску.

AWS Well-Architected Security Pillar більше орієнтований на проектування робочих навантажень. AWS базує його на принципах проектування, таких як надійні основи ідентифікації, відстеження, безпека на всіх рівнях, автоматизація, захист даних під час передачі та зберігання, зменшення прямого доступу людини до даних та підготовка до подій безпеки.

Чудово підходить для оглядів архітектури AWS. Самого ж цього недостатньо для управління всім підприємством.

Microsoft SDL ближче до інженерної системи. Microsoft описує SDL як свій підхід до інтеграції безпеки в DevOps, з основними фазами, що охоплюють вимоги, проектування, впровадження, верифікацію та випуск.

Це робить його корисним, коли хмарний ризик починається в коді, дизайні, виборі залежностей, моделюванні загроз або затвердженні релізу.

Як насправді використовувати фреймворки постачальників

Ставтеся до систем безпеки хмарних технологій постачальників як до шаблонів впровадження, а не як до замінників відповідності.

– Google допомагає вам подумати про довіру до виробничих послуг.

– SLSA забезпечує структуру цілісності ланцюга постачання програмного забезпечення.

– AWS Well-Architected покращує проектування безпеки робочих навантажень.

– Microsoft SDL забезпечує безпеку доставки.

– Cisco CCF демонструє, як одне підприємство може консолідувати засоби контролю та повторно використовувати докази для багатьох зобов'язань.

Практичний крок полягає в тому, щоб перетягнути корисні частини у власну модель керування.

Одна бібліотека керування.

Перевірки, що відповідають конкретним постачальникам, нижче.

Докази, пов'язані з активами, власниками, змінами та винятками.

Жодного дублювання аудиторських перевірок щоразу, коли покупець, регулятор чи внутрішня команда ставить одне й те саме питання різними мовами.

У цьому і полягає справжня цінність. Фреймворки постачальників – це не кінцевий пункт призначення. Це підказки від компаній, яким уже довелося вирішувати частини проблеми управління хмарою у великих масштабах.

Перетворіть елементи керування на хмарну роботу за допомогою Cloudware

Найскладніша частина хмарних систем безпеки полягає не в пошуку елементів керування.

NIST має контрольні заходи. CIS має контрольні заходи. ISO, CSA CCM, SOC 2, PCI DSS, HIPAA – всі вони мають контрольні заходи.

Найскладніше – узгодити ці елементи керування з неохайною реальністю гібридної інфраструктури: облікові записи AWS, підписки Azure, проекти GCP, робочі навантаження OCI, кластери Kubernetes, SaaS-додатки, VMware, локальні залежності, власники, винятки та заявки, які можуть містити повну історію, а можуть і не містити.

У фреймворку зазначено: захищайте конфіденційні дані.

Хмара запитує у відповідь:

Який актив? Який постачальник? Яке середовище? Яка програма? Який власник? Який виняток? Яким доказам? Якій контрольній групі це відповідає?

Cloudaware використовується командами хмарної безпеки, платформ, DevSecOps, GRC та аудиторськими командами, яким потрібні елементи керування фреймворком для початку роботи. Не «відображено в електронній таблиці». Зіставлено з реальними активами, реальними власниками, активним станом, заявками на виправлення та доказами, готовими до аудиту.

Це корисний рівень: одне місце, де керівники з безпеки бачать управління, інженери бачать контекст виправлень, а команди з дотримання вимог бачать докази.

Від мови фреймворку до хмарної роботи: як Cloudaware робить елементи керування функціональними

Найскладніша частина хмарних систем безпеки полягає в тому, щоб не знайти елементи керування.

NIST має елементи контролю. CIS має елементи контролю. ISO, CSA CCM, SOC 2, PCI DSS, HIPAA та внутрішні бібліотеки політик – усі вони мають елементи контролю.

Справжня проблема починається, коли ці елементи керування стикаються з майном: облікові записи AWS, підписки Azure, проекти GCP, робочі навантаження Oracle Cloud, кластери Kubernetes, VMware, інструменти SaaS, локальні системи, власники, винятки, заявки та сліди доказів, які не завжди узгоджуються між собою.

У фреймворку зазначено: захищайте конфіденційні дані.

Хмара запитує у відповідь:

Який актив? Який постачальник? Яке середовище? Яка програма? Який власник? Який виняток? Які докази? Якій групі елементів керування це відповідає?

Cloudaware допомагає командам з хмарної безпеки, платформи, DevSecOps, GRC та аудиту перетворити елементи керування фреймворком на операційну роботу. Не відображається в електронній таблиці. Зіставляється з реальними активами, власниками, перевірками політик, порушеннями, шляхами виправлення та доказами, готовими до аудиту.

Це корисний рівень: керівники служби безпеки бачать управління, інженери бачать контекст виправлень, а команди з дотримання вимог бачать докази.

Хмарні рішення, що допомагають впроваджувати хмарні системи безпеки:

– Cloudaware CMDB надає командам постійно оновлювану систему обліку в гібридних та багатохмарних середовищах, включаючи AWS, Azure, GCP, Oracle Cloud, Kubernetes, VMware та локальну інфраструктуру. Cloudaware позиціонує CMDB як рівень інвентаризації, який нормалізує хмарні та нехмарні активи в одну операційну модель. Це важливо, оскільки зіставлення фреймворків починається зі знання того, що існує. Контроль NIST, CIS, CSA CCM, SOC 2, PCI DSS або ISO не може бути чітко застосований, якщо команда не може бачити, які облікові записи, робочі навантаження, бази даних, ідентифікаційні дані, ресурси зберігання, теги, власники та середовища входять до області застосування.

– Cloudaware IT Compliance – це основний модуль для перетворення вимог фреймворку на відповідність коду. Він оцінює політики на основі даних CMDB, перетворює невдалі перевірки на відстежувані порушення, підтримує винятки з часовими рамками та пов’язує активи та елементи керування назад до CMDB зі статусом «відповідно/невідповідно», власником, прогалинами, історією виконання та доказами. Невдалий елемент керування не повинен мати лише

значення «невідповідно». Він повинен показувати уражений актив, середовище, власника, результат політики, статус винятку, життєвий цикл виправлення та слід доказів.

– Зіставлення елементів керування між стандартами безпеки та внутрішніми політиками. Документація механізму відповідності Cloudaware описує політики на основі таких стандартів, як PCI, HIPAA, NIST, ISO, GDPR та CIS Benchmarks для AWS, Azure та GCP, а також користувацькі політики, які можуть оцінювати будь-які дані CMDB. Практичною перевагою є повторне використання. Один елемент керування шифруванням може підтримувати кілька фреймворків, коли докази пов'язані з активом, власником, середовищем, історією запуску та результатом керування, замість того, щоб перебудовуватися з нуля для кожного аудиту.

– Заплановані оцінювання, сповіщення про відхилення, відстеження життєвого циклу, сповіщення власників та панелі інструментів, які дозволяють командам переходити від головних ключових показників ефективності до точних результатів перевірки. Саме тут фреймворки хмарної безпеки стають корисними щодня. Хтось редагує політику сегмента. Підписка з'являється поза очікуваною моделлю. Правило ведення журналу перестає застосовуватися після зміни IaC. Суть не в тому, щоб виявити прогалину через три місяці під час підготовки до аудиту, а в тому, щоб помітити її, поки хтось ще може її виправити.

– Робочі процеси виправлення з контекстом заявок. Cloudaware розглядає невдалі перевірки як результати першокласних правил з власником, серйозністю, угодою про рівень обслуговування, доказами та життєвим циклом. Результати можуть відкривати контекстно-багаті заявки в Jira або ServiceNow та маршрутизувати роботу за полями CMDB, такими як програма, команда, обліковий запис та середовище.

– Відстеження винятків та ризиків, що враховує власника. Елементи керування фреймворком ламаються, коли винятки знаходяться поза системою. Cloudaware підтримує обмежені в часі винятки та відстеження життєвого циклу,

тому робоче навантаження, яке порушує політику шифрування, може розглядатися як затверджений виняток, термін дії прийняття ризику минув або нове порушення політики, яке потребує виправлення.

– Панелі інструментів та експорт, готові до аудиту. Cloudaware IT Compliance надає панелі інструментів та експорт із постійною історією виконання, переглядами правил, відмінностями, ключовими показниками ефективності (KPI), елементами керування та доказами, тому аудити стають пошуком, а не реконструкцією.

Контроль стає корисним, коли він може відповісти на операційні питання: що вийшло з ладу, де саме вийшов з ладу, кому належить, наскільки це серйозно, що змінилося, як це виправити та де зберігається доказ.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Для реалізації програми мною була використана мова програмування Visual C++. У зв'язку з тим, що сьогодні рівень складності програмного забезпечення дуже високий, розробка застосунків Windows з використанням тільки якої-небудь мови програмування значно утрудняється. Програміст повинен затратити масу часу на рішення стандартних завдань по створенню багатовіконного інтерфейсу. Реалізація технології зв'язування й вбудовування об'єктів – OLE – зажадає від програміста ще більш складної роботи. Щоб полегшити роботу програміста практично всі сучасні компілятори з мови C++ містять спеціальні бібліотеки класів. Такі бібліотеки містять у собі практично весь програмний інтерфейс Windows і дозволяють користуватися при програмуванні засобами більш високого рівня, чим звичайні виклики функцій. За рахунок цього значно спрощується розробка застосунків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки застосунків Windows

дозволяють автоматизувати процес створення застосунку. Для цього використовуються генератори застосунків. Програміст відповідає на питання генератора застосунків і визначає властивості застосунку – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор застосунків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої застосунки. Подібні засоби автоматизованого створення застосунків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики застосунку й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні застосунки, а також застосунки, що не мають головного вікна, – замість нього використовується діалогова панель. Можна також включити підтримку технології OLE, баз даних, довідкової системи. Звичайно, MFC AppWizard не всесильний. Прикладну частину застосунку програмістові прийдеться розробляти самостійно. Вихідний текст застосунку, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон застосунку – це вже половина всієї роботи. Вихідні тексти застосунків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти застосунків тільки з використанням бібліотеки класів MFC (Microsoft Foundation Class library). Тому тільки вивчивши мову C++ і бібліотеку MFC, можна користуватися засобами автоматизованої розробки й створювати свої застосунки в найкоротший термін. Як уже згадувався, MFC – це базовий набір (бібліотека) класів, написаних мовою C++ і призначених для спрощення й прискорення процесу програмування для Windows. Бібліотека містить багаторівневу ієрархію класів, що нараховує близько 200 членів. Вони дають можливість створювати Windows-застосунки на базі об’єктно-орієнтованого підходу. З погляду програміста, MFC являє собою каркас, на основі якого можна писати програми

для Windows. Бібліотека MFC розроблялася для спрощення завдань, що стоять перед програмістом. Як відомо, традиційний метод програмування під Windows вимагає написання досить довгих і складних програм, що мають ряд специфічних особливостей. Зокрема, для створення тільки каркаса програми таким методом знадобиться близько 75 рядків коду. У міру ж збільшення складності програми її код може досягати воістину неймовірних розмірів. Однак та ж сама програма, написана з використанням MFC, буде приблизно в три рази менше, оскільки більшість приватних деталей приховано від програміста.

Одною з основних переваг роботи з MFC є можливість багаторазового використання того самого коду. В зв'язку з тим, що бібліотека містить багато елементів, загальних для всіх Windows-застосунків, немає необхідності щораз писати їх заново. Замість цього їх можна просто успадковувати (говорячи мовою об'єктно-орієнтованного програмування). Крім того, інтерфейс, забезпечуваний бібліотекою, практично незалежний від конкретних деталей, його що реалізують. Тому програми, написані на основі MFC, можуть бути легко адаптовані до нових версій Windows (на відміну від більшості програм, написаних звичайними методами). Ще однією істотною перевагою MFC є спрощення взаємодії із прикладним програмним інтерфейсом (API) Windows. Будь-який додаток взаємодіє з Windows через API, що містить кілька сотень функцій. Значний розмір API утрудняє спроби зрозуміти й вивчити його цілком. Найчастіше навіть складно простежити, як окремі частини API зв'язані один з одним! Але оскільки бібліотека MFC поєднує (шляхом інкапсуляції) функції API у логічно організовану безліч класів, інтерфейсом стає значно легше управляти.

Оскільки MFC являє собою набір класів, написаних мовою C++, тому програми, написані з використанням MFC, повинна бути в той же час програмами на C++. Для цього необхідно володіти відповідними знаннями. Для початку необхідно вміти створювати власні класи, розуміти принципи спадкування й уміти перевизначати віртуальні функції.

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 інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

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 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 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

Порівнюючи постачальників хмарних послуг (CSP), важливо враховувати не лише функції та ціни їхніх пропозицій. Використовуючи CSA CCM, ваш бізнес може отримати глибше розуміння середовищ інформаційної безпеки, з'ясувати, чи відповідають внутрішні засоби контролю конфіденційності постачальника хмарних послуг вашим цілям управління ризиками та захисту, а також виявити потенційні витоки конфіденційності в хмарі, які можуть поставити під загрозу інформацію та програми, що зберігаються в хмарі.

Використовуючи CSA CCM, компанія може визначити функції та налаштування, які вона очікує від хмарного центру обробки даних свого постачальника хмарних послуг. Впроваджуючи ці заходи, ви можете розробити систему управління операційною безпекою, яка є одночасно адаптивною та проактивною, що ідеально підходить до унікальної бізнес-структури вашої компанії. Ви можете оцінити функції та послуги, що пропонуються постачальниками хмарних послуг, і прийняти обґрунтоване рішення про те, який з них найкраще відповідатиме вашим потребам з точки зору безпеки даних, використовуючи розроблені вами внутрішні команди.

Найновіша версія CSA CCM, версія 4.0, наразі перебуває в розробці. Завдяки реорганізації структури з метою включення нового домену для журналювання та моніторингу (LOG) та внесенню коректив до існуючих доменів, CCM версії 4 являє собою значне покращення порівняно з попередницею, версією 3.0.1. (GRC, A&A, UEM, CEK). З впровадженням нових та вдосконаленням старих заходів контролю, ця редакція також призведе до суттєвого збільшення обов'язкових заходів.

Оновлення CCM версії 4 містить додаткові функції, такі як:

– Потреби, що виникають внаслідок передових технічних інновацій, будуть повністю задоволені.

– Оновлена матриця управління конфіденційністю та відповідальності

– Більша сумісність з бенчмарками, краща чутність та легша сумісність – все це переваги.

CCM версії 4 охоплює такі сфери:

– Аудит та гарантія захисту додатків/інтерфейсів.

– Управління змінами та стійкість BCM M&O.

– Забезпечення конфіденційності інформації.

– Безпека центру обробки даних.

– Шифрування та криптографія.

– Ризик, дотримання вимог та управління.

– Безпека кадрів.

– Ідентифікація та адміністрування.

– Кібербезпека та віртуалізація.

– Портативність/Інтероперабельність.

– UEMT.

– SIM, електронне розкриття інформації та хмарна криміналістика.

– Прозорість, підзвітність та управління ринковими цінностями (SCM).

– Управління телебаченням та відео.

– Відстеження.

Це база даних типової архітектури та законів, яких повинні дотримуватися підприємства. Виконуючи команди CCM, ви також задовольняєте вимоги багатьох інших контрольних показників безпеки, політик та вайрфреймів, на які вони відповідають. Зібрання найпопулярніших стандартів хмарної безпеки в одному місці усуває необхідність використання різних вайрфреймів. Користувач може бачити для кожної команди всі різні критерії, яким вона відповідає. Наприклад, виконання передумов трьох різних структур і правил може бути

досягнуто шляхом демонстрації згоди за допомогою однієї перевірки.

Кожен елемент керування CCM описує відповідальну сторону (CSP) та конкретну хмарну модель (IaaS, PaaS, SaaS) або середовище (публічне, гібридне, приватне), до якого застосовується управління. Окреслюючи, які розділи керівного документа застосовні до CSP, CCM допомагає визначити відповідні обов'язки та відповідальність кожного з них.

Цей план призначений для постачальників послуг зв'язку (CSP) для оцінки та звітування про свої засоби контролю конфіденційності. Він також використовується компаніями, які впроваджують хмарні рішення, для оцінки безпеки свого mesh-середовища. Таким чином, як CSP, так і підприємствам, що використовують mesh-сервіси, може знадобитися використовувати CSA CCM. Він надає комплексний набір засобів контролю безпеки та відповідає їх всесвітньо визнаним стандартам конфіденційності, таким як ISO 27001 та NIST SP 800-53, допомагаючи підприємствам гарантувати, що їхня хмарна конфіденційність відповідає найкращим галузевим практикам.

CCM для хмарних постачальників

Архів безпеки, довіри, гарантії та ризиків (STAR) оцінює корпоративну конфіденційність за допомогою CCM. Пакет STAR заохочує гнучкі, прогресивні, різноманітні акредитації, які взаємодіють із поширеними зовнішніми дослідженнями для зменшення зусиль та витрат. Щоб продемонструвати згоду з галузевими цінностями, нормами та законодавством, постачальники послуг конфіденційності можуть опублікувати тривалий набір питань CCM та надіслати його потенційним та поточним клієнтам. Постачальники повинні завантажити свої CAIQ до архіву STAR, щоб клієнти мали до них доступ.

CCM для хмарних клієнтів або бізнес-організацій

CAIQ – це когортний модуль CCM, який дозволяє споживачам або аудиторам хмарних послуг ставити постачальникам хмарних послуг запити «ствердно чи ні». Ці запити можуть містити дані про засоби контролю безпеки IaaS, PaaS та SaaS постачальника на основі CCM. Компанії використовують дані

CAIQ для створення RFP (запитів пропозицій) для подальшого посилення. Співбесіди щодо RFP дозволяють підприємствам перевіряти відповіді продавців. Понад 500 компаній проводять самооцінку реєстру STAR за допомогою CAIQ.

3.2 Розробка структурної схеми

З усього наведеного вище, можливо зробити наступні ключові висновки

– Структура хмарної безпеки – це не PDF-файл, який ваша команда стирає з пилку перед аудитом. Це операційна модель, яка пов'язує елементи керування з реальними активами, власниками, середовищами, винятками, шляхами виправлення та доказами в AWS, Azure, GCP, SaaS, Kubernetes та локальних середовищах.

– Найкраща структура залежить від критичної ситуації. NIST CSF 2.0 надає вам основу для управління. CSA CCM є найсильнішим для хмарного картування контролю. CIS Controls and Benchmarks допомагають командам платформи посилити реальні конфігурації. SOC 2, ISO 27017, PCI DSS, HIPAA та FedRAMP мають значення, коли клієнти або регуляторні органи можуть блокувати бізнес.

– Структура управління хмарною безпекою відповідає на питання, на яке зазвичай не можуть відповісти інструменти: хто вирішує, хто виправляє, хто приймає ризик і хто доводить, що контроль спрацював? Без цього рівня власності навіть хороші висновки перетворюються на перенаправлені заявки та повільне виправлення.

– Мультихмарність порушує відкладене зіставлення елементів керування. «Шифрування конфіденційних даних» стає AWS KMS, Azure Key Vault, Google Cloud KMS, політиками зберігання, налаштуваннями бази даних, правилами резервного копіювання, межами IAM та аудиторськими доказами. Мета керування залишається незмінною. Реалізація – ні.

– SaaS потребує власного фреймворку, оскільки ризики криються в

грантах OAuth, прогалинах SCIM, неактивних адміністраторах, обході SSO локальними користувачами, зовнішньому обміні та журналах аудиту на стороні постачальника. HR SaaS заслуговує на особливу увагу, оскільки Workday, BambooHR, Rippling та подібні інструменти часто мають найщільніший стек PII в компанії.

– Важливі показники – це не марнотратство. Відстежуйте охоплення захисним щитом, MTTR для критичних неправильних конфігурацій, відсоток контролю з автоматизованими доказами та прогалини в охопленні структури. Ці чотири числа показують, чи справді управління переміщує ризики, чи просто повідомляє про них.

– Фреймворки стають корисними лише тоді, коли вони стосуються хмарної інфраструктури. Невдалий контроль повинен показувати актив, постачальника, власника, середовище, зіставлений стандарт, пов'язані зміни, статус винятку, запит на виправлення та підтвердження. У цьому полягає різниця між «ми дотримуємося NIST» та «ми можемо довести, що це робоче навантаження на виробництві регулюється».

Структура хмарної безпеки – це операційна модель, яка повідомляє вашим хмарним командам, як безпека має працювати в реальних середовищах: що потрібно захищати, які засоби контролю застосовуються, хто відповідає за кожен ризик, як виявляються порушення та які докази підтверджують, що проблему було вирішено.

Ось поширений випадок: в облікових записах AWS є 400 забутих груп безпеки. У підписках Azure одна команда володіє додатком, а інша – мережею. У GCP проекти розгортаються для аналітики, а потім непомітно перетворюються на критично важливі для бізнесу робочі процеси.

Саме ця структура запобігає тому, щоб усе це перетворилося на «ми вважаємо, що хтось це перевірив».

Структура управління хмарною безпекою йде на ще глибший рівень. Вона визначає правила відповідальності:

– Хто схвалює винятки.

– Хто підписує порушення політики.

– Як швидко необхідно усунути виробничий ризик.

– Де зберігаються аудиторські докази, коли відповідність вимогам вимагає підтвердження через шість місяців.

Контроль не є реальним, доки ви не зможете простежити його до активу, власника, зміни та доказу. В іншому випадку це просто гарне речення в документі політики.

Правило на кшталт «шифрування виробничих даних» звучить просто, поки не торкнеться AWS RDS, Azure SQL, хмарного сховища GCP, резервних томів, знімків, ключів, винятків та схвалення змін. Фреймворк має відповідати на операційні питання.

Саме це трасування перетворює управління на щось, що команди безпеки, платформи та відповідності можуть реально використовувати. У цьому вся суть. Фреймворк перетворює хмарну безпеку з розрізнених добрих намірів на підзвітну систему.

Що включає система хмарної безпеки

Ось чиста ментальна модель:

– Каркас – це система.

– Стандарт є еталоном.

– Політика – це правило, якого насправді має дотримуватися ваша хмара.

В умовах єдиної хмари команди іноді можуть розмити ці межі та вижити. Багатохмарна система менш поблажлива. Одна вимога «шифрування даних у стані спокою» перетворюється на налаштування корзини S3, конфігурації сховища Azure, елементи керування хмарним сховищем GCP, стани шифрування бази даних, правила керування ключами, перевірки Terraform, схвалення винятків та збір доказів.

Ось чому системи хмарної безпеки важливі у 2026 році. Не тому, що командам потрібні чіткіші формулювання щодо управління. Їм потрібен спосіб

забезпечити чітку передачу рішень щодо безпеки від рівня ради директорів до рівня контролю за дотриманням вимог конкретних постачальників, не втрачаючи при цьому відповідальності на середині процесу.

Хмарна безпека у 2026 році не розвалюється через брак контролю у командах. Руйнування відбувається пізніше, коли елементу керування потрібно пережити AWS, Azure, GCP, Kubernetes, SaaS, CI/CD, постачальників ідентифікаційних даних, локальні залежності, експерименти зі штучним інтелектом, бізнес-винятки та найнебезпечнішу фразу в хмарних операціях: «Я думав, що це належить іншій команді».

Ось чому зараз важлива структура хмарної безпеки. Вона дає лідерам у сфері безпеки спосіб перетворити стандарти на відповідальність, відповідальність на дії, а дії на докази.

1. Гібридна хмара зробила принцип «один елемент керування, одна реалізація» застарілим 73% організацій використовували гібридні хмарні ресурси у 2026 році. (Flexera). Стара модель була акуратною. Одна платформа. Одна площина керування. Одна команда, яка знала, де що знаходиться.

Тепер єдине правило безпеки сховища має застосовуватися до публічного доступу до блоків S3, доступу до публічної мережі Azure Storage, IAM-бакета GCP, приватних кінцевих точок, ключів KMS, журналів аудиту, елементів керування службами VPC та будь-якого спільного файлового ресурсу, який досі знаходиться локально, оскільки команда міграції вичерпала бюджет.

Ось аргумент: гібридна хмара не порушує безпеку, тому що вона має більше платформ. Вона порушує безпеку, тому що кожна платформа виражає той самий контроль по-різному.

Фреймворк забезпечує стабільність наміру. Реалізація може змінюватися залежно від постачальника, але питання безпеки залишається чистим:

– Чи є цей актив відкритим?
– Це зашифровано?
– Кому це належить?

– Виняток було схвалено?

– Чи можемо ми довести поточний стан?

Без цього рівня команди збирають результати. Вони не керують ризиками.

2. Складність стала проблемою безпеки ще до того, як ця проблема взагалі з'явилася. 55% організацій кажуть, що забезпечення безпеки хмарних середовищ є складнішим, ніж забезпечення безпеки локальних середовищ. (Thales Group). Архітектор безпеки відкриває панель інструментів і бачить 1800 неправильних конфігурацій. Не чудово. Але справжня проблема не в кількості. Справжня проблема полягає в тому, що 600 з них не мають чіткого власника програми, 200 прив'язані до «тимчасових» середовищ, 90 знаходяться в підписках, суміжних з робочим середовищем, і ніхто не може пояснити, чи цей застарілий VPN-маршрут все ще потрібен.

Це і є складність хмарних технологій у її найдорожчій формі.

Це не завжди оголошує себе порушенням. Іноді це проявляється як:

– застарілий обліковий запис сервісу, який ніхто не хоче видаляти;

– простір імен Kubernetes, що належить команді, яка двічі реорганізувалася;

– публічна кінцева точка, схвалена для тестування та більше ніколи не перевірялася;

– SaaS-конектор із широкими дозволами, оскільки інтеграція «знадобилася йому один раз»;

– прогалина в веденні журналу, прихована за трьома різними панелями інструментів.

Структура управління хмарною безпекою важлива, оскільки вона надає складності системі маршрутизації. Область застосування. Контроль. Власник. Угода про рівень обслуговування. Виняток. Докази. Ніякої драми. Тільки відповідальність.

3. Конфіденційні дані переміщуються в місця, які служби безпеки не завжди перевіряють спочатку. 54% хмарних даних були класифіковані як

конфіденційні у 2025 році, порівняно з 47% у 2024 році. Лише 8% організацій зашифрували 80% або більше своїх хмарних даних. Це факт, підтверджений дослідженням Thales Group. Ось тут розмова стає незручною. Конфіденційні дані більше не зберігаються лише в основній виробничій базі даних. Вони містяться в журналах, озерах даних, знімках, резервних копіях, підказках моделей, експортованих звітах, тимчасових виходах конвеєрів, реплікованих кошиках та робочих просторах аналітики, створених командами, які «просто щось тестували».

Тож питання не може залишатися на рівні політики. «Чи захищаємо ми конфіденційні дані?» – це занадто м’яке формулювання.

Кращі питання звучать як запитання аудитора з ліхтариком:

– Який хмарний ресурс зберігає дані?
– Яку класифікацію він має?
– Чи ввімкнено шифрування?
– Кому належить ключ?
– Які ідентичності можуть це прочитати?
– Резервні копії також захищені?
– Коли востаннє перевіряли доступ?
– Де докази?

Аргумент простий. Захист даних не спрацьовує, коли класифікація, доступ, шифрування, зберігання та докази знаходяться в різних діалогах. Фреймворк об'єднує ці потоки в одну операційну модель.

4. Ризик ідентичності більше не є побічним контролем. Це площина контролю. Серед організацій, де спостерігалися порушення, пов’язані з хмарними ресурсами, надмірні дозволи були причиною у 31% випадків, непослідовний контроль доступу – у 27%, а слабка гігієна ідентифікації – у 27%. (Cloud Security Alliance). Ризик для ідентифікації стає небезпечним, коли ніхто не може пояснити, чому існує дозвіл. Не лише хто має доступ, але й чому цей доступ все ще дійсний, яке робоче навантаження від нього залежить і хто відповідатиме за

ризик, якщо він залишиться відкритим. Саме цю частину більшість програм ідентифікації недооцінюють.

Ідентифікація в хмарі – це не просто вхід співробітників у консоль. Це облікові записи служб, ідентифікації робочих навантажень, федеративні ролі, токени CI/CD, ключі API, гранти OAuth, користувачі типу «break-glass», підрядники, інтеграції SaaS та стара автоматизація, яка все ще має доступ на запис, оскільки ніхто не хоче порушувати розгортання.

Зріла структура не обмежується «щоквартальним переглядом доступу».

Він визначає, що означає доступ з високим рівнем ризику в ролях адміністратора AWS IAM, Azure Entra ID, GCP IAM, Kubernetes RBAC та SaaS. Він примусово надає право власності особам, які не є людьми. Він припиняє термін дії винятків. Він відстежує застарілі ключі. Він записує, чому існує привілейований доступ і хто його схвалив.

Аргумент: принцип найменших привілеїв не є принципом, доки ваша команда не зможе застосовувати його як для людей, так і для машин.

5. Штучний інтелект перетворив «тіньові IT» на «тіньовий рух даних». Дослідження IBM «Вартість витоку даних» за 2025 рік показало, що 63% досліджених організацій, у яких зазнали витоків, не мали політик управління штучним інтелектом. Штучний інтелект рідко входить до компанії з позначкою «ризик нового підприємства». Це прибуває як скорочений шлях. Підтримка підключає чат-бота до історії заявок. Інженерний відділ дозволяє помічнику читати нотатки про розгортання. Фінансовий відділ завантажує експортовані дані про рахунки в інструмент прогнозування. Аналітика продукту перевіряє дані клієнтів на відповідність моделі. Спеціаліст із відповідності використовує штучний інтелект для узагальнення пакетів доказів.

Корисно? Так.

Керовані? Часто ні.

IBM також повідомила, що в кожній п’ятій дослідженій організації спостерігалися порушення, пов’язані з тіньовим штучним інтелектом, причому ці

інциденти додавали до середніх витрат на витік даних до 670 тисяч доларів США. URL-адреса джерела: Аналітика витоків даних IBM.

Ось чому командам хмарної безпеки потрібно ставитися до робочих процесів штучного інтелекту як до хмарних ресурсів. Не вібрації. Не паніка. Логіка активів:

– До чого може отримати доступ інструмент штучного інтелекту?
– Яких хмарних даних це стосується?
– Куди йде вихід?
– Чи визначено утримання?
– Чи можуть запити містити регульовані дані?
– Хто схвалив інтеграцію?
– Що відбувається, коли інструмент змінюється?

Аргумент: управління ШІ не може бути поза управлінням хмарою, оскільки ШІ тепер споживає хмарні дані, хмарні журнали, хмарні ідентифікаційні дані та дані про витрати на хмару.

6. Хмарні відходи часто є першою ознакою порушення відповідальності. Очікувані витрати на хмарні технології зросли до 29% у 2026 році, змінивши п'ятирічну тенденцію до зниження. Це схоже на FinOps. Це не лише FinOps.

Неактивна віртуальна машина може бути виправлена. Забута база даних може все ще містити регульовані дані. Старий знімок може бути незашифрованим. Тестове середовище може бути публічно доступним, оскільки всі припускали, що його видалять після спринту. Ресурс без тегів також може не мати власника, моніторингу, життєвого циклу та шляху виправлення.

Марнотратство хмарних ресурсів не є автоматично проблемою безпеки. Але некеровані витрати часто стоять поруч із некерованим ризиком.

Фреймворк допомагає командам розділити важливі категорії:

– дорогий, схвалений та критично важливий для бізнесу;
– бездіяльний, але навмисно утримуваний;
– забутий, викритий;

– невідповідний, але охоплений затвердженим винятком;

– ризиковані, пов'язані з виробництвом та прострочені для відновлення.

Аргумент: коли дані про інвентаризацію активів, витрати, власність та статус полісів не пов'язані між собою, як FinOps, так і безпека працюють на основі часткової істини. Саме так виживають марнотратство. Так виживає і ризик.

7. Зловмисники перенесли терміни. Щоквартальне управління не отримало службової записки. Вхід на основі стороннього програмного забезпечення становив 44,5% початкових векторів доступу у другій половині 2025 року, порівняно з 2,9% у першій половині 2025 року. Проміжок часу між розкриттям та масовою експлуатацією скоротився з тижнів до днів. Такі дані зі звіту Google Cloud Threat Horizons Report H1 2026. Тепер помістіть це у справжню хмару. Понеділок: падіння критичного CVE. Вівторок: спроби експлойту починають вражати робочі навантаження, пов'язані з Інтернетом. Середа: хтось виявляє, що один із уражених сервісів пов'язаний з виробничим API. Четвер: власник програми каже, що патч потребує тестування. П'ятниця: служба безпеки запитує, чи вже встановлено правило WAF, ізоляцію або компенсаційний контроль.

Google спостерігав, як майнери XMRig були розгорнуті протягом приблизно 48 годин після публічного розкриття CVE-2025-55182 у кількох інцидентах. У тому ж звіті рекомендується перейти до автоматизованого захисту, включаючи віртуальні цілі встановлення патчів протягом 24 годин та повне виправлення протягом 72 годин.

Ось у чому суть аргументу. Перевірка контролю, яка проводиться раз на квартал, не може регулювати вікно загрози, що вимірюється днями. Структура, готова до 2026 року, потребує постійного виявлення активів, контексту впливу, визначення пріоритетів вразливостей, маршрутизації в екстрених випадках, компенсаційних заходів контролю, затвердження винятків, перевірки виправлень та збору доказів. Інакше хмарна безпека перетворюється на гонку між зловмисником і тим, хто першим це помітить. А це не стратегія.

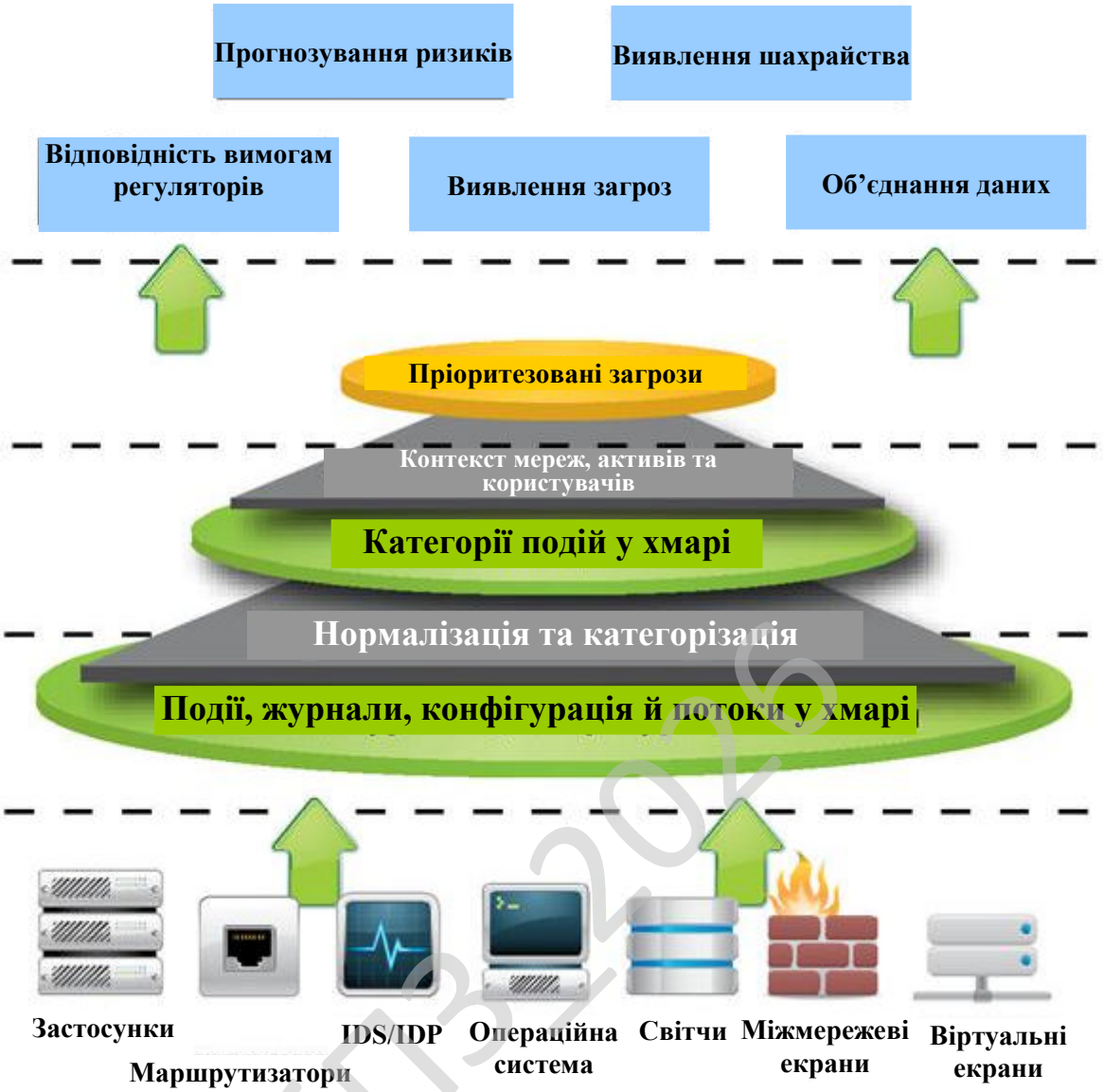

Рисунок 3.1 – Структурна схема системи

Програмне забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі допомагає службам безпеки:

– виявляти погрози;
– відповідати нормативним вимогам;
– прогнозувати ризики;
– виявляти інсайдерів;
– поєднувати розрізнені дані.

Як вибрати систему хмарної безпеки

Неправильний спосіб вибору системи хмарної безпеки – це почати з самої системи. Звучить навпаки, я знаю. Але хмарні команди роблять це постійно. Хтось каже: «Нам слід узгодити це з NIST». Інша людина каже: «Покупці постійно просять SOC 2». Відділ відповідності хоче ISO. Платіжна команда раптом згадує про PCI DSS. Потім хмарний архітектор, який тихо спостерігав, як AWS, Azure, GCP, Kubernetes та локальні інфраструктури рухаються в п'яти напрямках, ставить краще питання:

Якщо регулятор може зупинити запуск, розпочніть роботу з нормативно-правовою базою. Якщо корпоративні покупці можуть відкладати угоди, надайте пріоритет системі гарантування якості. Якщо справжньою проблемою є розростання хмарних технологій, дрейф ідентифікаційних даних та невідповідність контролю між постачальниками, спочатку використовуйте систему управління, а потім налаштуйте схему технічних контролів під нею.

Найкращий фреймворк – це не той, що має найгарнішу бібліотеку елементів керування. Це той, який ваші команди можуть використовувати, коли реальний актив не спрацьовує в реальному елементі керування у виробництві. Якщо ніхто не може зіставити несправне правило з власником, середовищем, шляхом виправлення та доказами, фреймворк – це просто паперова робота з кращим форматуванням.

Це і є фільтр. Не «Який стандарт звучить найбільш зріло?» Швидше типу: який з них допомагає нам прийняти рішення в понеділок вранці, коли результат оприлюднено, аудитор запитує, а власник застосунку хоче доказів, що це справді його проблема?

1. Знайдіть джерело тиску. У кожній компанії є такий. Іноді їх кілька. SaaS-компанія, яка продає корпоративним обліковим записам, зазвичай першою відчуває тиск покупців. SOC 2 Type II стає засобом довіри. ISO 27001 додає довіри. ISO 27017 допомагає, коли покупці ставлять питання, що стосуються

хмари, щодо спільної відповідальності, адміністративних операцій, розділення орендарів та контролю постачальників.

Платіжна платформа має менше свободи. Якщо дані власників карток зберігаються, обробляються або передаються, PCI DSS є центральним фактором. Розмова про дизайн стає гострішою: сегментація, шифрування, ведення журналу, обмеження доступу, управління вразливостями, безпечна розробка та докази щодо середовища даних власників карток.

Постачальник хмарних послуг, який продає послуги федеральним агентствам США, знову має іншу реальність. FedRAMP – це не допоміжне обладнання. Він змінює операційну модель: засоби контролю NIST 800-53, безперервний моніторинг, сертифікати відповідності та управління (POA&M), сканування вразливостей, реагування на інциденти, контроль змін та постійний збір доказів.

Ще немає жорсткого тиску на відповідність вимогам? Вам пощастило. Використовуйте цей час добре. NIST CSF 2.0 надає вам основу для управління, особливо з функцією Govern. CIS Controls IG1 надає команді розумну технічну відправну точку, перш ніж хмарна інфраструктура розшириться.

2. Потім перевірте, чи фреймворк зможе витримати вашу архітектуру. Саме тут багато програм і стають видимими. Фреймворк може виглядати чистим на слайді та розвалитися всередині. Запитайте:

– Чи може він обробляти облікові записи AWS, підписки Azure, проекти GCP, кластери Kubernetes, SaaS-додатки та локальні системи?

– Чи відокремлює це намір керування від реалізації, специфічної для постачальника?

– Чи може одна вимога шифрування бути пов'язана з S3, сховищем Azure, хмарним сховищем, базами даних, знімками та резервними копіями?

– Чи визначає це, кому належать винятки?

– Чи може воно надати докази, не перетворюючи дотримання вимог на полювання за скарбами?

– Чи зрозуміють інженери, що потрібно виправити?

Якщо відповідь негативна, у вас все ще може бути корисний стандарт. У вас просто ще немає операційної моделі.

Ось чому багатохмарним командам часто потрібен CSA CCM як посередник. Він забезпечує хмарну структуру контролю, особливо щодо спільної відповідальності. Не тому, що він замінює NIST, CIS, SOC 2, ISO або PCI. Він допомагає перемикатися між ними, коли одне робоче навантаження стосується трьох постачальників та двох моделей підзвітності.

Більшості хмарних команд не варто вибирати один фреймворк і вважати його завершеним. Більш надійна модель виглядає так:

1. Використовуйте NIST CSF 2.0 для організації управління та відповідальності за ризики.

2. Додайте елементи керування CIS та контрольні показники CIS, щоб команди платформи та DevSecOps мали конкретні рекомендації щодо зміцнення.

3. Застосовуйте CSA CCM, коли багатохмарне картографування стає складним.

4. Контрольні засоби Layer SOC 2, ISO 27017, PCI DSS, FedRAMP або HIPAA, коли покупці, регуляторні органи або ринки вимагають доказів.

Саме так стають корисними структури хмарної безпеки. Не як конкуруючі документи. Як одна система контролю, яка розповідає ту саму історію: від ризиків для керівництва до конфігурації хмари та аудиторських доказів.

Структура управління хмарною безпекою

Структура управління хмарною безпекою визначає, хто вирішує, хто виконує та хто перевіряє засоби контролю безпеки у вашій хмарній інфраструктурі. Це речення звучить спокійно. Реальна хмарна інфраструктура – ні. Одна команда володіє обліковим записом AWS. Інша – підпискою Azure. GCP було відкрито для аналітики та якимось чином стало суміжним з виробничою платформою. Kubernetes рухається швидше, ніж рада затвердження.

Локальна ідентифікація все ще забезпечує доступ. Відповідність вимагає доказів шестимісячної давності.

Управління – це та частина, яка не дає всім сказати: «Я думав, що цим займається безпека».

Невдалий контроль легко знайти. Найскладніше – довести, хто ним володіє, чи був прийнятий ризик, яка зміна його створила та які докази свідчать про поточний стан. Саме тут управління стає оперативним, а не політичним.

Основи управління хмарною безпекою: SPRC:

1. Використовуйте модель SPRC: Стратегія, Політика, Ризик, Відповідність. Досить проста для запам'ятовування. Досить серйозна для запуску гібридної хмарної програми.

2. Стратегія встановлює межу ризику. Саме тут керівництво вирішує, що найважливіше. Персональна інформація клієнта у виробництві? Нульова толерантність до публічного розкриття. Відсутні теги в "пісочниці" розробки? Виправте це, але не будіть CISO. Стратегія визначає затверджених постачальників, критичні послуги, схильність до ризику, шляхи ескалації та очікування щодо виправлення. Без неї кожне сповіщення бореться за один і той самий кисень.

3. Політика перетворює наміри на правила, яких можуть дотримуватися інженери. Корисна політика не говорить про «безпечний доступ». Вона говорить, що привілейовані хмарні ролі потребують багатофакторної автентифікації (MFA), іменованого власника, щоквартального перегляду та закінчення терміну дії винятків. Публічне сховище заблоковано за замовчуванням. Шифрування виробничого процесу використовує затверджене керування ключами. Робочі навантаження, що підключаються до Інтернету, потребують реєстрації. RACI має бути тут, оскільки хмарним командам потрібні чіткі межі між безпекою, платформою, власниками програм, відповідністю вимогам та власником контролю.

4. Ризик підтримує зв'язок реєстру з реальністю. Реєстр ризиків не повинен бути цвинтарем розпливчастих заяв. У хмарі він повинен бути пов'язаний з активами, обліковими записами, підписками, проектами, середовищами, вразливостями, неправильними конфігураціями, власниками та бізнес-сервісами. Широка роль IAM у робочому навантаженні виробничих платежів не є тим самим ризиком, що слабкий тег на тестовій віртуальній машині.

5. Відповідність доводить, що контроль справді працював. Аудиторам не потрібна поезія. Їм потрібні часові позначки, історія затверджень, записи змін, посилання на заявки, статус винятків, результати сканування та поточні дані про конфігурацію.

Ролі та RACI для управління хмарною безпекою

Таблиця RACI виглядає нудною, доки у четвер о 16:40 не виникне збій у системі управління виробництвом. Тоді стає дуже цікаво. Роль AWS IAM має дозволи, подібні до прав адміністратора. Обліковий запис сховища Azure дозволяє доступ до загальнодоступної мережі. У проекті GCP вимкнено ведення журналу. Проблема має високий рівень серйозності. Контроль відповідає SOC 2, ISO 27017 і, можливо, PCI DSS, якщо дані про оплату є поблизу.

Тепер постає справжнє питання. Хто вирішує, чи є це прийнятним ризиком? Хто змінює конфігурацію? Кому належить захисне огородження, щоб цього не повторилося? Хто зберігає докази? Хто перевіряє, чи процес дійсно працював?

Ось чому RACI важливий у хмарних системах безпеки. Не як корпоративний артефакт. Як рівень маршрутизації між ризиками, інженерією, відповідністю та аудитом.

Швидке нагадування:

– Відповідальний: виконує роботу.

– Відповідальний: несе відповідальність за результат.

– Консультований: надає інформацію перед початком дій.

– Поінформований: потребує видимості, але не керує виконанням завдання.

Скажімо, виробниче сховище відкрите. Хмарний інженер змінює налаштування. Керівник хмарної платформи оновлює модуль guardrail або Terraform. GRC підтверджує, який елемент керування вийшов з ладу, і зберігає докази: знаходить позначку часу, уражений актив, власника, квиток змін, журнал затвердження та сканування після виправлення. CISO вирішує, чи є прийнятним будь-який виняток. Внутрішній аудит пізніше перевіряє шаблон, а не лише одне виправлення. Елемент керування без іменованого власника стає проблемою для пошуку когось, хто його пересилає. Елемент керування з RACI перетворюється на роботу: хто його виправляє, хто приймає ризик, хто доводить результат і хто перевіряє процес пізніше. Ось і вся гра. Для управління хмарними ресурсами не потрібно більше людей на нарадах. Йому потрібно менше рішень, що не враховуються. Хороший RACI робить кожен невдалий крок контролю корисним: для власника, для виправлення, для винятку або для доказу.

Показники управління, які дійсно мають значення

Таблиця RACI надала кожному елементу керування маршрут. Тепер управлінню потрібен спідометр. Бо фраза «підзвітний CISO, відповідальний інженер, консультації з GRC» звучить нормально, доки критичне рішення щодо публічного сховища не залишиться відкритим протягом 19 днів, оскільки ніхто не знає, чи є додаток робочим, чи регулюються дані, чи все ще застосовується виняток з минулого кварталу. Саме цей розрив мають виявити показники.

Структура хмарної безпеки корисна лише тоді, коли вона показує, куди рухається ризик, де він зупиняється та де бракує доказів. Підрахунок результатів – це найлегша частина. Вимірювання стану управління складніше.

Почніть з покриття захисного огородження

Перше число, яке я хотів би бачити на інформаційній панелі, просте: який відсоток хмарних облікових записів насправді перебуває під базовими обмеженнями? Не «відомо». Не «перевірено». Під захисними огорожами.

Скажімо, у системі є 240 облікових записів AWS, підписок Azure та проектів GCP. Якщо 198 з них застосовують базові елементи керування, охоплення становитиме 82,5%. З цих відсутніх 17,5% починається історія. Це облікові записи з придбання? Нові «пісочниці» розробки? Аналітичні проекти? Тіньові середовища? Навантаження, суміжні з виробничим середовищем, зі слабкими тегами? Відстежуйте охоплення захисним бар'єром за постачальником, середовищем, бізнес-підрозділом та критичністю. Виробництво має бути близьким до 100%. Все, що пов'язано з регульованими даними, слід розглядати як провід під напругою.

Потім слідкуйте за MTTR на наявність критичних помилок у конфігурації.

Середній час до виправлення показує, чи призводить до дій щодо відповідальності. Критична помилка в конфігурації не закривається, коли Jira отримує запит. Вона закривається, коли виправлення застосовується, перевіряється та додається до доказів.

Гарні приклади для відстеження:

– публічне сховище, що торкається конфіденційних даних;
– IAM на рівні адміністратора без схвалення;
– відкритий SSH або RDP;
– вимкнено ведення журналу виробництва;
– незашифроване регульоване сховище даних;
– навантаження, пов'язане з Інтернетом, з критичною вразливістю.

Якщо MTTR залишається високим, можливо, вузьким місцем не є інженер. Причиною затримки може бути відсутність права власності на актив, відсутність затвердженого шаблону виправлення, нечітка серйозність затримки або вимога GRC надати докази після того, як робота вже виконана.

Автоматизація доказів – це стрес-тест аудиту

Ручний збір доказів виглядає керованим, доки SOC 2, ISO 27017, PCI DSS, внутрішній аудит та огляд безпеки клієнтів не з'являться в одному кварталі. Потім команда починає археологію. Скріншоти. Посилання на квитки.

Повідомлення у Slack. Старі експорти сканів. Хтось запитує: «Шифрування було ввімкнено до чи після вікна аудиту?». Вимірюйте відсоток контролю за допомогою автоматизованих доказів: стан конфігурації, запис активу, призначення власника, історія змін, журнал затверджень, заявка на виправлення, статус винятку та позначка часу перевірки після виправлення.

Почніть з IAM, шифрування, ведення журналу, публічного розкриття інформації, усунення вразливостей та затвердження змін у виробничому середовищі. Ці засоби контролю створюють найбільше проблем, коли докази виконуються вручну.

Виміряйте розрив у покритті структури

Це метрика «чи обманюємо ми себе?». Візьмемо фреймворки, яких ви нібито дотримуєтеся: NIST CSF 2.0, CIS Benchmarks, CSA CCM, SOC 2, ISO 27017, PCI DSS, FedRAMP. Тепер перевірте, чи кожен елемент керування відповідає реальним активам, власникам, середовищам та доказам. Розрив покриття виникає, коли контроль існує в структурі, але не в операціях.

Приклад: CSA CCM очікує чіткого визначення спільної відповідальності. Добре. Чи може ваша команда довести, хто володіє ключами шифрування в AWS KMS, Azure Key Vault та Google Cloud KMS? Чи можуть вони показати право власності на резервні копії, збереження журналів, статус перевірки доступу та схвалення винятків для кожного робочого навантаження? Якщо ні, то розрив не є теоретичним. Воно стоїть у системі. Метрики належного управління ускладнюють приховування ризиків. Вони показують незахищений обліковий запис, повільне виправлення, слід ручних доказів та контроль, який так і не перейшов з мови фреймворку в хмарну реальність.

3.3 Розробка функціональної схеми

Завдання програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблене в даній роботі –

аналізувати інформацію, що надходить від різних систем, таких як антивіруси, DLP, IDS, маршрутизатори, міжмережеві екрани, операційні системи серверів і користувальницьких ПК, і при цьому детектувати відхилення від норм за якимись критеріями. Якщо таке відхилення виявлене – система генерує інцидент. Тобто, серед безлічі записів у системних журналах програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі-рішення виявляє сліди якихось підозрілих дій. Немаловажно, що з її допомогою можна виявити дії, які зовні виглядають цілком необразливими, але в сукупності являють загрозу.

Основні функції програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі:

– Збір і об'єднання даних.

– Централізоване зберігання журналів.

– Інтелектуальний аналіз даних (Кореляція подій).

– Оповіщення про інциденти.

– Оцінка відповідності хмари вимогам стандартів і регламентів.

Компоненти програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі

– Сховища журналів.

– Сенсори.

– Центр керування програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі.

Від кого програмне забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі одержує інформацію?

Access Control, Authentication. Застосовуються для моніторингу контролю доступу до інформаційних систем і використання привілеїв.

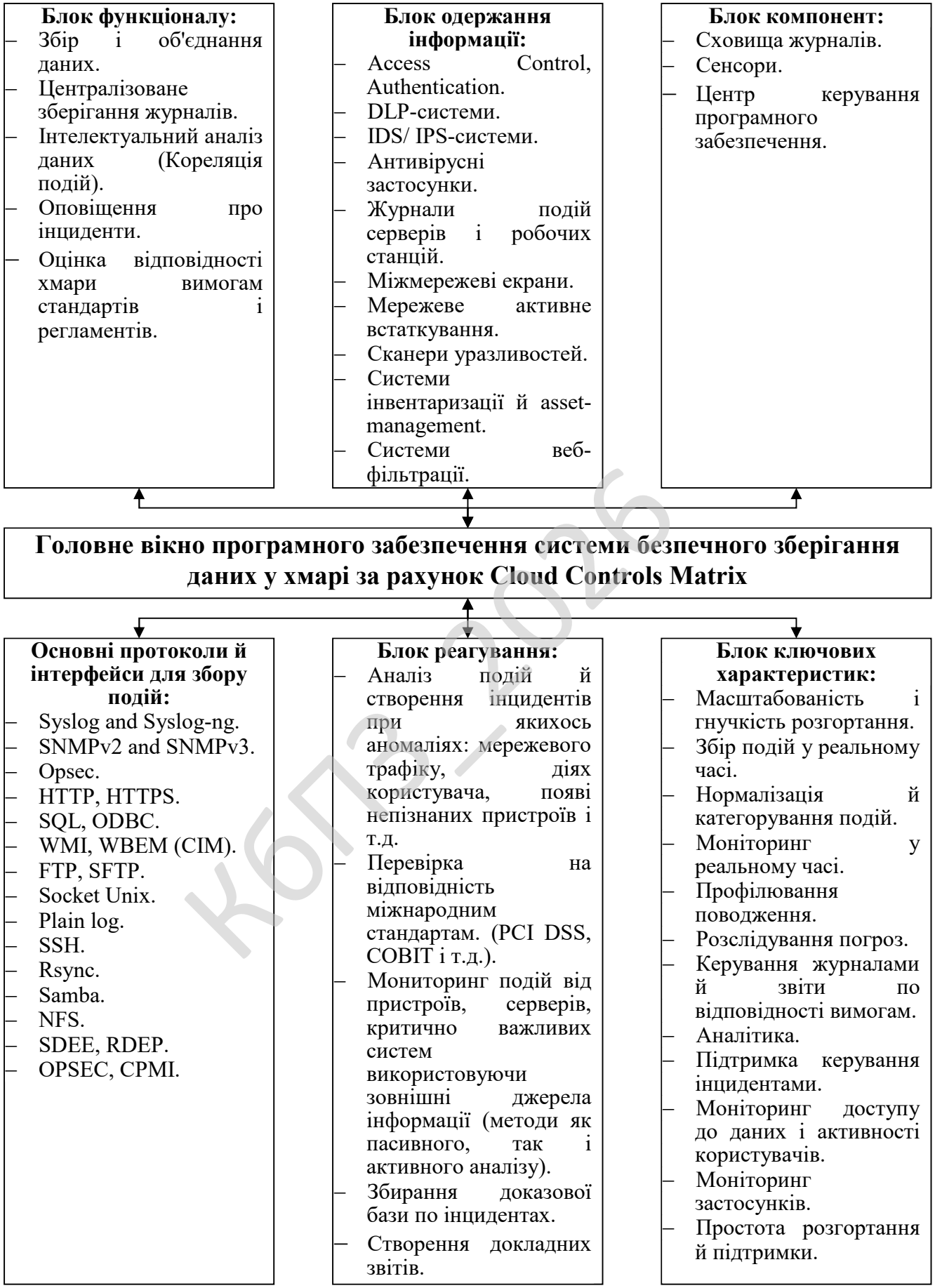

Рисунок 3.2 – Функціональна схема системи

DLP-системи. Відомості про спроби інсайдерських витоків, порушенні прав доступу.

IDS/ IPS-системи. Несуть дані про мережеві атаки, зміни конфігурації й доступу до пристроїв.

Антивірусні застосунки. Генерують події про працездатність ПЗ, базах даних, зміні конфігурацій і політик, шкідливому коді.

Журнали подій серверів і робочих станцій. Застосовуються для контролю доступу, забезпечення безперервності, дотримання політик інформаційної безпеки.

Міжмережеві екрани. Відомості про атаки, шкідливому ПЗ й іншому.

Мережеве активне встаткування. Використовується для контролю доступу, обліку мережевого трафіку.

Сканери уразливостей. Дані про інвентаризацію активів, сервісів, ПЗ, уразливостей, поставка інвентаризаційних даних і топологічної структури.

Системи інвентаризації й asset-management. Поставляють дані для контролю активів в інфраструктурі й виявлення нових.

Системи веб-фільтрації. Надають дані про відвідування співробітниками підозрілих або заборонених веб-сайтів.

Основні протоколи й інтерфейси для збору подій:

- Syslog and Syslog-ng.
- SNMPv2 and SNMPv3.
- Opsec.
- HTTP, HTTPS.
- SQL, ODBC.
- WMI, WBEM (CIM).
- FTP, SFTP.
- Socket Unix.
- Plain log.
- SSH.

– Rsync.

– Samba.

– NFS.

– SDEE, RDEP.

– OPSEC, CPMI.

Що може робити програмне забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблене в даній роботі:

– Аналізувати події й створювати інциденти при якихось аномаліях: мережевого трафіку, діях користувача, появі непізнаних пристроїв і т.д.

– Перевірка на відповідність міжнародним стандартам. (PCI DSS, COBIT і т.д.).

– Моніторити події від пристроїв, серверів, критично важливих систем використовуючи зовнішні джерела інформації (методи як пасивного, так і активного аналізу).

– Збирати доказову базу по інцидентах.

– Створення докладних звітів.

Ключові характеристики програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі:

– Масштабованість і гнучкість розгортання.

– Збір подій у реальному часі.

– Нормалізація й категорування подій.

– Моніторинг у реальному часі.

– Профілювання поводження.

– Розслідування погроз.

– Керування журналами й звіти по відповідності вимогам.

– Аналітика.

– Підтримка керування інцидентами.

– Моніторинг доступу до даних і активності користувачів.

– Моніторинг застосунків.

– Простота розгортання й підтримки.

Основні етапи впровадження програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі:

– Обстежити інфраструктуру й вибрати спосіб впровадження (всі події обробляються в одному місці, або буде розпаралелювання?).

– Формуємо й затверджуємо ТЗ.

– Розробляємо керівництва адміністраторів і посібник користувача.

– Установлюємо й підготовляємо сервер програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі (інтеграція залозки, її прописка в мережі).

– Налаштування джерел подій. (налаштовуємо джерела на відправлення подій програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі).

– Пишемо правила реагування на події. (на даному етапі себе покажуть погано налаштовані джерела).

– Тестова експлуатація й нагромадження статистики. (Один з найважливіших етапів).

– Коректування й написання додаткових правил. (Навчання програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі).

– Завершення тестування.

– Переклад програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі в повну бойову готовність.

Потрібно розуміти, що впровадження програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що

розроблене в даній роботі – це не справа 1-й тижня. У середньому, правильне впровадження програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі займає від 6 місяців.

Переваги від програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі відчутні через 4-6 місяців його роботи в бойовому режимі.

Основні питання, на які ви повинні дати собі відповідь перед впровадженням:

– Які повсякденні завдання плануєте вирішувати за допомогою програмного забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі?

– Яке у вас загальне число користувачів і мережевих елементів?

– Є чи у вас філії, віддалені офіси, події яких потрібно враховувати?

– Які системи інформаційної безпеки у вас є?

– Чи плануєте ви аналізувати мережеві потоки? (sFlow, netFlow).

– Є чи у вашій інфраструктурі спец. пристрою, події яких потрібно враховувати?

– Як часто відбуваються інциденти, які вам необхідно розслідувати?

– Чи необхідно ви відповідати яким-небудь міжнародним стандартам?

Кому потрібний програмне забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix, що розроблено в даній роботі в першу чергу?

– Банківська сфера.

– Великі підприємства.

– Географічно розподілені підприємства.

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

3.4 Розробка діаграми процесів

Розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3. Основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей. Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Рисунок 3.3 – Діаграма взаємодії процесів

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.

При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграми потоків даних містять чотири типи елементів:

– Зовнішні по відношенню до системи сутності.

– Потоки даних між елементами трьох попередніх типів.

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

– Сховища даних (репозиторії).

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

Розглянемо реалізацію бакалаврської дипломної роботи. Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень.

Було створено блок-схеми роботи системи. 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.

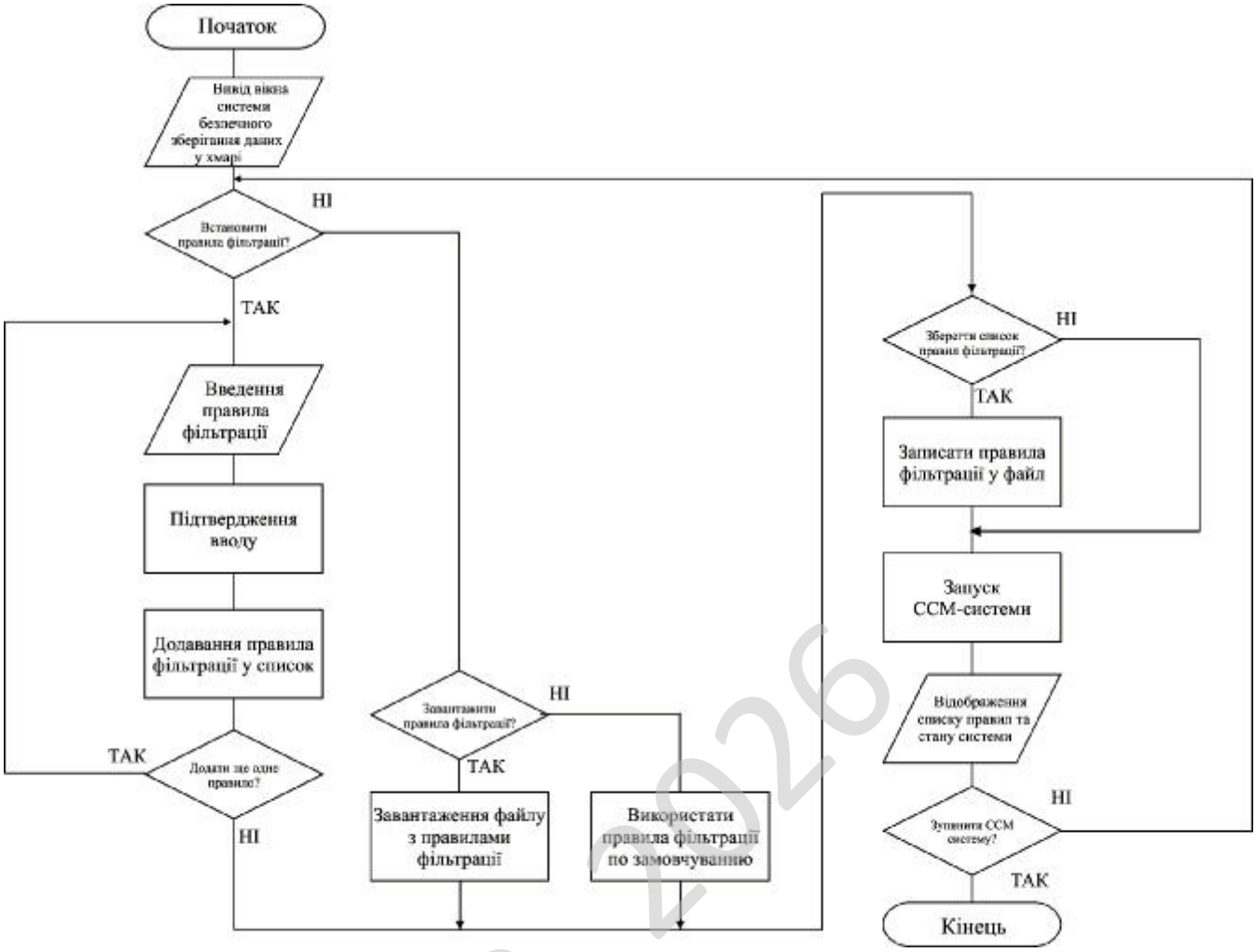

Рисунок 4.1 – Блок-схема основної програми

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.

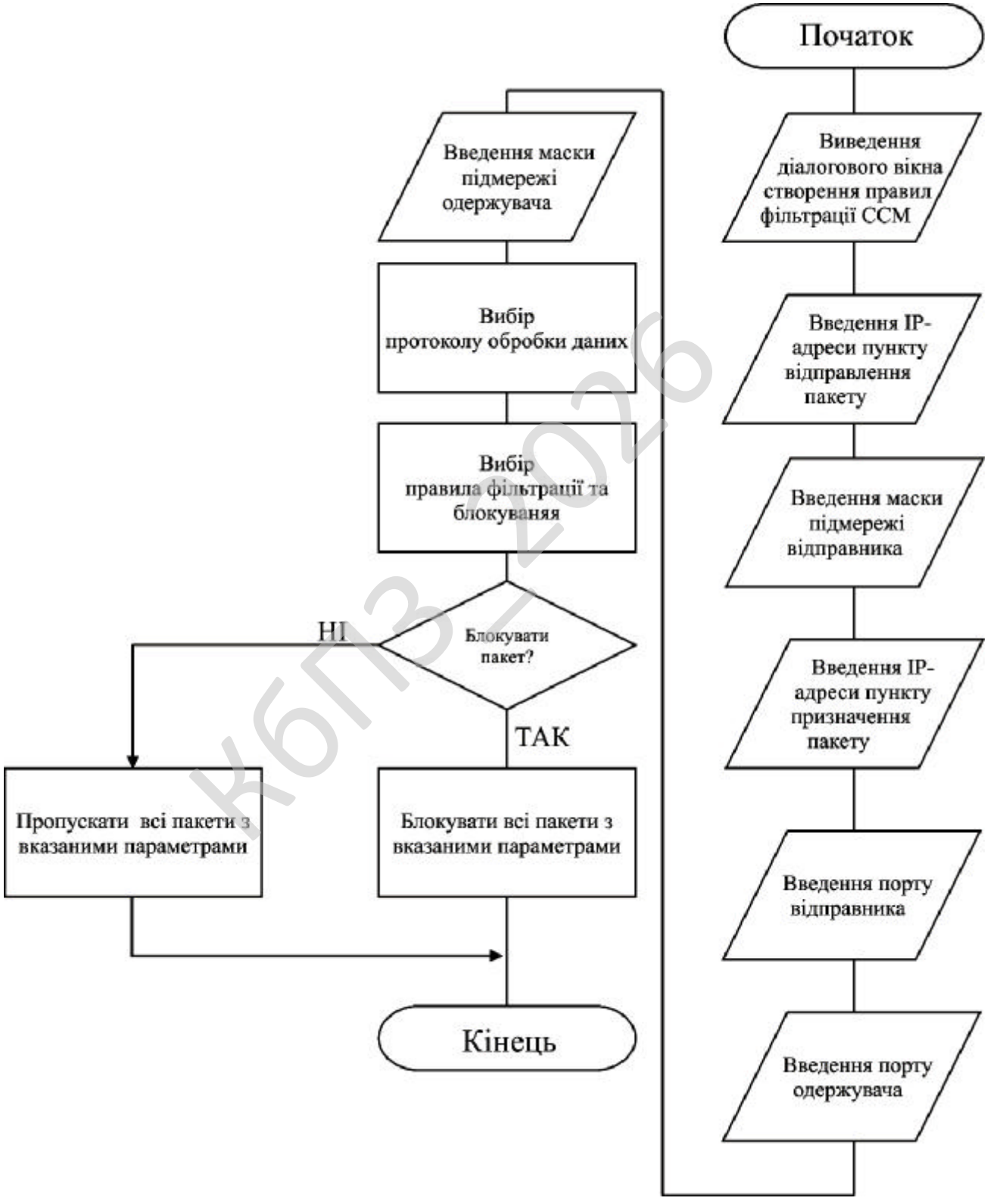

Рисунок 4.2 – Блок-схема роботи підпрограми

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 показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. 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 високого рівня декомпозиції задач на класи.

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

підсистеми 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.

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних програм і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

– набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;

– набір клієнтів, які використовують сервіси, що надаються серверами;

– мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Дуже важливо ясно уявляти, хто або що розглядається як «клієнт». Можна говорити про клієнтський комп'ютер, з якого відбувається звернення до інших комп'ютерів. Можна говорити про клієнтське та серверне програмне забезпечення. Нарешті, можна говорити про людей, які бажають за допомогою відповідного програмного та апаратного забезпечення отримати доступ до тієї чи іншої інформації.

Загальноприйнятим є положення, що клієнти та сервери – це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми – і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов'язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

– рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і введення від нього керуючих команд;

– прикладний рівень, який реалізує основну логіку ПЗ і на якому здійснюється необхідна обробка інформації;

– рівень управління даними, який забезпечує зберігання даних та доступ до них.

Дворівнева клієнт-серверна архітектура передбачає взаємодію двох програмних модулів – клієнтського та серверного. В залежності від того, як між ними розподіляються наведені вище функції, розрізняють:

– модель тонкого клієнта, в рамках якої вся логіка ПЗ та управління даними зосереджена на сервері. Клієнтська програма забезпечує тільки функції рівня представлення;

– модель товстого клієнта, в якій сервер тільки керує даними, а обробка інформації та інтерфейс користувача зосереджені на стороні клієнта. Товстими клієнтами часто також називають пристрої з обмеженою потужністю: кишенькові комп'ютери, мобільні телефони та ін.

Типовим прикладом клієнт-серверної взаємодії є WWW. Існує величезна кількість веб-серверів, на яких розміщується та чи інша інформація. У найпростішому випадку ця інформація являє собою набір веб-сторінок, які можуть зберігатися на сервері у вигляді файлів, розмічених за допомогою мови розмітки HTML. Але ситуація, як правило, є складнішою; значна частина веб-ресурсів на сучасному етапі є динамічними, тобто вони не існують в заздалегідь підготовленому вигляді, а створюються безпосередньо в процесі обробки запиту від користувача.

Для того, щоб людина, яка працює в Інтернеті, могла переглянути ту чи іншу сторінку, на її комп'ютері повинно бути встановлено відповідне програмне забезпечення. Програми для перегляду веб-сторінок називаються браузерами.

Але, крім браузерів, до серверів можуть звертатися і інші клієнти, а саме – автономні програми. Вони можуть передбачати взаємодію з людиною, а можуть працювати в цілком автоматичному режимі. Типовим класом таких програм є роботи, призначені для автоматичного перегляду веб-ресурсів. Зокрема, роботи є важливим елементом пошукових систем і використовуються ними для перегляду сторінок і збору інформації про них.

Для запиту до веб-сервера клієнтська програма повинна задати місцезнаходження комп'ютера, на якому розміщується серверна програма, назву потрібного документа і, можливо, інші дані, які специфікують запит. Мережа забезпечує знаходження сервера і передачу йому клієнтського запиту. Серверні програми обробляють цей запит, відповідь пересилається по мережі клієнтові.

Трирівнева клієнт-серверна архітектура, яка почала розвиватися з середини 90-х років, передбачає відділення прикладного рівня від управління даними. Відокремлюється окремий програмний рівень, на якому зосереджується прикладна логіка ПЗ. Програми проміжного рівня можуть функціонувати під управлінням спеціальних серверів ПЗ, але запуск таких програм може здійснюватися і під управлінням звичайного веб-сервера. Нарешті, управління даними здійснюється сервером даних.

Для роботи з системою користувач використовує стандартне програмне забезпечення – звичайний браузер. Це позбавляє його необхідності завантажувати та інсталювати спеціальні програми (хоча інколи така необхідність все-таки виникає).

Але користувачеві слід надати в розпорядженні інтерфейс, який дозволяв би йому взаємодіяти з системою і формувати запити до неї. Форми, що визначають цей інтерфейс, розміщуються на веб-сторінках та завантажуються разом з ними.

Веб-оглядач формує запит та пересилає його до сервера, який здійснює обробку. При необхідності сервер викликає серверні програмні модулі, які забезпечують обробку запиту і в разі потреби звертаються до сервера даних. Сервер даних здійснює операції з даними, що зберігаються в системі та складають її інформаційну основу. Зокрема, він може здійснити вибірку з інформаційної бази відповідно до запиту та передати її модулю проміжного рівня для подальшої обробки. Дані, з якими працює сервер даних, найчастіше організовані як реляційна база даних.

Найчастіше веб-сервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоч і являють собою окремі і логічно незалежні програмні модулі.

На сучасному етапі для програмування модулів проміжного рівня використовується мова серверних сценаріїв PHP, а для управління даними – СУБД MySQL. Таким чином, зв'язку PHP-MySQL слід розглядати як стандартний інструмент для створення порівняно простих інтерактивних веб-сайтів та систем електронної комерції; близько 90% комерційних систем сьогодні створюється саме на цій основі. Водночас як засоби управління даними, так і middleware-засоби можуть бути найрізноманітнішими. Так, для створення серверних програм, крім PHP, широко застосовуються Java, Perl, Python, Delphi.

Взагалі, технології створення розподілених, зокрема веб-програм, стрімко розвиваються. Слід згадати про технології EJB (Enterprise Java Beans), CORBA, а також про .NET – порівняно нову ініціативу компанії Microsoft. Для зберігання даних та їх передачі часто використовується так звана розширювана мова розмітки XML (Extensible Markup Language).

Розши́рювана мо́ва розмі́тки (Extensible Markup Language, скорочено XML) – запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними застосунками, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML.

XML-документ складається із текстових знаків, і придатний до читання людиною. Стандарт XML (Recommendation, перше видання від 10 лютого 1998, останнє, четверте видання 29 вересня 2006) визначає набір базових лексичних та синтаксичних правил для побудови мови описання інформації шляхом застосування простих тегів.

Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях. Іншими словами, запропонований стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та зміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем (Schema), такими як XML Schema (XSD), DTD або RELAX NG. Прикладами мов, заснованих на XML, є: XSLT, XAML, XUL, RSS, MathML, GraphML, XHTML, SVG, а також XML Schema.

Основні поняття

Коректність

Коректний документ (well-formed document) відповідає всім синтаксичним правилам XML. Документ, що не є коректним, не може називатись XML-документом. Сумісний синтаксичний аналізатор (Conforming parser) не повинен обробляти такі документи. Зокрема, коректний XML-документ має:

1. Лише один елемент у корені.

2. Непорожні елементи розмічено початковим та кінцевим тегами (наприклад, <пункт> Пункт 1</пункт>). Порожні елементи можуть позначатися «закритим» тегом, наприклад <IAmEmpty />. Така пара еквівалентна <IAmEmpty></IAmEmpty>.

3. Один елемент не може мати декілька атрибутів з однаковою назвою. Значення атрибутів перебувають або в одинарних ('), або у подвійних (") лапках.

4. Теги можуть бути вкладені, але не можуть перекриватись. Кожен некореневий елемент мусить повністю перебувати в іншому елементі.

5. Документ має складатися тільки з правильно закодованих дозволених символів Юнікоду. Єдиними кодуваннями, які обов'язково має розуміти XML-процесор, є UTF-16 та UTF-8. Фактичне та задекларованe кодування (character encoding) документа мають збігатись. Кодування може бути задекларовано ззовні, як у заголовку «Content-Type» при передачі по протоколу HTTP, або в самому документі використанням явної розмітки на самому початку документа. У разі відсутності інформації про кодування, документ має бути в кодуванні UTF-8 (або його підмножині ASCII).

Валідність

Документ називається валідним (valid), якщо він є коректним, містить посилання на граматичні правила та повністю відповідає обмеженням, вказаним у цих правилах (DTD або XML Schema або іншому подібному документі).

Синтаксичний аналізатор

Синтаксичним аналізатором (часто парсер, від parser) називається програма або компонент, що читає XML-документ, проводить синтаксичний аналіз та відтворює його структуру. Якщо синтаксичний аналізатор перевіряє документ на валідність, то такий аналізатор називають валідатором (validating).

Назви елементів чутливі до регістру літер. Наприклад, наступна пара елементів правильна: <Step> … </Step> у той час як ця – ні: <Step> … </step>.

Правильний вибір назв для XML-елементів підкреслюватиме значення даних у створеній мові розмітки. Це сприятиме полегшенню роботи людей з такими документами, зберігаючи можливості для комп'ютерної обробки даних.

Вибір змістовних назв передає семантику елементів та атрибутів для людини, без посилання на зовнішню документацію. Однак це може призвести до надмірності розмітки, що ускладнює редагування й збільшує розмір файлів.

Правильний вибір назв для XML-елементів підкреслюватиме значення даних у створеній мові розмітки. Це сприятиме полегшенню роботи людей з такими документами, зберігаючи можливості для комп'ютерної обробки даних. Вибір змістовних назв передає семантику елементів та атрибутів для людини, без

посилання на зовнішню документацію. Однак це може призвести до надмірності розмітки, що ускладнює редагування й збільшує розмір файлів. XML-документи мають як фізичну, так і логічну структуру.

Фізична структура

Сутності (Entity). Головною сутністю є зміст документа. Інші можливі сутності вказуються за допомогою. Посилання на сутності (&назва; в самому документі, та, наприклад %назва; у визначені його типу) можуть слугувати в ролі позначень спеціальних символів, посилань на спеціальні символи або окремих документів чи фрагментів тексту.

XML-декларація, в ній вказується версія XML, кодування та інша допоміжна інформація.

Декларація типу документа може застосовуватись для того, аби додавати нові типи сутностей та визначати логічну структуру документа.

Логічна структура

XML-документ має ієрархічну логічну структуру, і може представлятись у вигляді дерева. Вузлами цього дерева можуть бути: елементи, фізична структура яких складається із коректної пари відкриваючого та закриваючого тегів (<Назва-тега>) та (</Назва-тега>), або тега порожнього елемента (<Назва-тега />).

Атрибути, що мають вигляд пар ключ-значення (назва атрибута="значення атрибута") і знаходяться або у відкриваючому, або у порожньому тезі (подібно до метаданих).

Вказівки щодо обробки документа (Processing Instruction) (<?Обробник параметр ?>).

Коментарі (<!-- Текст коментаря -->).

Текст, або у вигляді простого тексту, або фрагментів CDATA (<![CDATA[довільний текст]]>).

XML-документ повинен мати лише один кореневий елемент. Решта елементів є піделементами цього кореневого елемента. Деякі веб-браузери здатні безпосередньо відображати XML-документи. Це може досягатись шляхом

застосування таблиці стилів (Stylesheet). Вказані у таблиці стилів операції можуть призводити до перетворення XML-документа в інший, відмінний від XML формат.

Коректність XML-документів

Залишивши назви, дозволену ієрархію, та значення елементів і атрибутів відкритою та можливою бути визначеною в спеціалізованих схемах або визначеннях типу документа (DTD).

XML утворює синтаксичну основу для створення спеціалізованих, заснованих на XML мовах розмітки даних.

Загальний синтаксис таких документів стабільний і наперед визначений – документи мають відповідати базовим вимогам XML, гарантуючи те, що довільне програмне забезпечення з підтримкою XML буде здатне щонайменше зчитати і відтворити відносну структуру інформації, що міститься в них.

Схема лише доповнює синтаксичні правила множиною обмежень. Зазвичай схеми обмежують назви елементів та атрибутів, дозволені типи значень і допустиму ієрархію елементів, наприклад, дозволяючи лише елементу з назвою «народження» містити під-елемент з назвою «місяць» та з назвою «день», і кожен із них мусить містити лише літери. Обмеження, вказані в схемі, можуть також включати присвоєння певних типів даних для впливу на те, як обробляється інформація; наприклад, дані елемента «місяць» можна визначити як такі, що містять лише місяць, як це визначено відповідно до використаної мови схем.

Коректний XML-документ, що відповідає обмеженням схеми або DTD, називається валідним.

DTD (Document Type Definition). Найдавнішим форматом схем для XML є успадкований від SGML формат визначення типу документа (Document Type Definition, DTD). У той час, як через включення до стандарту XML 1.0 DTD став поширеним форматом схем, він має такі обмеження:

1. Відсутність нових можливостей XML, із найважливішою з посеред них простори назв.

2. Брак виразності. Деякі формальні аспекти XML-документів неможливо відобразити в DTD.

3. Використовується спеціалізований, заснований не на XML синтаксис для опису схем.

DTD все ще використовується в багатьох програмах, оскільки він вважається найпростішим форматом для аналізу та збереження.

XML Schema. На заміну DTD було розроблено нову мову схем – XML Schema (буквально – XML-схема), скорочено позначається як XSD (від XML Schema Definition). XSD набагато потужніші за DTD в описанні заснованих на XML мов. Вони використовують багатий набір типів даних, підтримують детальніші обмеження на структуру документів, та повинні оброблятись складнішими системами. XSD побудовано на основі XML, що робить можливим використання звичайних інструментів XML для їхньої обробки, хоча реалізації XSD вимагають набагато більше аніж просто можливість читати XML.

При розробці ПЗ було використано підходи ри́зик-ме́неджменту – це система управління ризиками, яка включає в себе стратегію та тактику управління, направлені на досягнення основних цілей. Ефективний ризик-менеджмент включає:

– систему управління;

– систему ідентифікації і вимірювання;

– систему супроводження (моніторингу та контролю).

Сучасна наука представляє ризик як вірогідну подію, в результаті настання якої можуть відбутися позитивні, нейтральні або негативні наслідки. Якщо ризик припускає наявність як позитивних, так і негативних результатів, він відноситься до спекулятивних ризиків. Якщо ж наслідки негативні, або відсутні взагалі, такий ризик іменується чистим.

Мета ризик-менеджменту – підвищення конкурентоспроможності господарюючих суб'єктів за допомогою захисту від реалізації чистих ризиків. Теорія ризик-менеджменту ґрунтується на трьох базових поняттях: корисності,

регресії і диверсифікації. У 1738 швейцарський математик Даніель Бернуллі доповнив теорію вірогідності методом корисності або привабливості того або іншого результату подій. Ідея Бернуллі полягала в тому, що в процесі ухвалення рішення люди приділяють більше уваги розміру наслідків різних результатів, ніж їх вірогідність.

В кінці XIX століття англійський дослідник Ф. Гальтон запропонував вважати регресію або повернення до середнього значення універсальною статистичною закономірністю. Суть регресії трактувалася ним як повернення явищ до норми з часом. Згодом було доведено, що правило регресії діє в найрізноманітніших ситуаціях, починаючи з азартних ігор та розрахунку вірогідності виникнення нещасних випадків, і закінчуючи прогнозуванням коливань економічних циклів.

У 1952 аспірант Університету Чикаго Гарі Марковіц в статті «Диверсифікація вкладень» («Portfolio Selection») математично обґрунтував стратегію диверсифікації інвестиційного портфеля, зокрема, він показав, як шляхом продуманого розподілу вкладень мінімізувати відхилення прибутковості від очікуваного показника. У 1990 Г. Марковіцу присуджена Нобелівська премія за розробку теорії і практики оптимізації портфеля фондових активів.

Етапи ризик-менеджменту

У ризик-менеджменті прийнято виділяти декілька ключових етапів:

– на першому етапі відбувається виявлення ризику з супутньою оцінкою вірогідності його реалізації і масштабу наслідків;

– на другому етапі здійснюється розробка ризик-стратегії з метою зниження вірогідності реалізації ризику і мінімізації можливих негативних наслідків;

– на третьому етапі вибираються методи і інструменти управління виявленим ризиком;

– на четвертому етапі проводиться безпосереднє управління ризиком;

– на завершальному етапі оцінюються досягнуті результати і коректується ризик-стратегія.

За ключовий етап ризик-менеджменту вважається етап вибору методів і інструментів управління ризиком.

Методи і інструментарій ризик-менеджменту

Базовими методами ризик-менеджменту є відмова від ризиків, зниження, передача і ухвалення.

Ризик-інструментарій значно ширший. Він включає політичні, організаційні, правові, економічні, соціальні інструменти, причому ризик-менеджмент як система допускає можливість одночасного застосування декількох методів і інструментів ризик-управління.

Найбільш часто вживаним інструментом ризик-менеджменту є страхування. Страхування припускає передачу відповідальності за відшкодування передбачуваного збитку сторонній організації (страхової компанії).

Прикладами інших інструментів можуть бути відмова від надмірно ризикової діяльності (метод відмови), профілактика або диверсифікація (метод зниження), аутсорсинг витратних ризикових функцій (метод передачі), формування резервів або запасів (метод ухвалення).

Розглянемо розробку проекта в контрольованому середовищі (PRojects IN Controlled Environments – PRINCE) – це метод управління проектами. Метод включає в себе управління, контроль і організацію проекту. «PRINCE2» – це другий реліз зазначеного методу, що є зареєстрованим торговим знаком, державної торгово–промислової палати (Office of Government Commerce – OGC), незалежного підрозділу державного казначейства Сполученого Королівства.

PRINCE2 походить від більш раннього методу PROMPT та методу проектного управління PRINCE, який був розроблений у 1989 р. Центральним телекомунікаційним та комп'ютерним агентством (Central Computer and Telecommunications Agency – CCTA), як державний стандарт Великобританії з управління проектами у сфері інформаційних технологій (IT). Незабаром

зазначений стандарт почали використовувати за межами IT сфери. У 1996 р. PRINCE2 був визнаний універсальним методом управління проектами. PRINCE2 став надзвичайно популярним і зараз де–факто є стандартом проектного управління в Великобританії.

Остання версія була випущена у 2009 р., як результат проекту оновлення Prince2:2009 державної торгово–промислової палати Великобританії.

PRINCE2:2009 Оновлення: З 2006 р. метод переглядався, а 16 Червня, 2009 р. був опублікований під назвою «Оновлення PRINCE2:2009». Назва «PRINCE2» (замість «PRINCE3» чи схожої) використано задля демонстрації збереження головних принципів методу.

Все ж таки, це найбільш суттєвий перегляд методу з 1996 р. з метою його адаптації до мінливого бізнес середовища, спрощення і «полегшення», а також кращої інтеграції з іншими методами державної торгово–промислової палати Великобританії (ITIL, P3O, P3M3, MSP, etc.). Головна різниця між релізом 2009 р. та більш ранніми релізами полягає в появі двох довідників замість одного. «Управління проектами, використовуючи PRINCE2» (Managing Projects Using PRINCE2) – оновлена методика управління, що призначена для менеджерів проектів, для яких управління проектами є щоденною діяльністю. «Стратегічне управління проектами, використовуючи PRINCE2» (Directing Projects Using PRINCE2) – новий документ для Керівника проектів і Ради управління проектам, тобто спонсорів/замовників проектів. Екзамен як з Теоретичної так і з Практичної частини буде базуватися на новому довіднику «Управління проектами» та не буде включати матеріали з довідника «Стратегічне управління проектами». Прохідний бал для Теоретичного екзамену залишиться не змінним. Прохідний бал для Практичного екзамену збільшиться з 50 % до 55 %. Тривалість екзамену з Практичної частини буде зменшено з 3 годин до 2,5 годин.

PRINCE2 – це структурований підхід до управління проектами, який спрямований на управління проектами в рамках чітко визначеної організаційної структури. PRINCE2 описує процедури координації людей та завдань в проекті, я

к створювати/планувати проект та що робити, якщо проект потребує внесення змін через невідповідність фактичного стану виконання проекту плану його виконання. Кожний процес зазначено з ключовими вхідними та вихідними даними, а також з специфічними цілями та завданнями, які необхідно виконати, що загалом дає можливість контролю відхилень від плану.

Поділ методу на частини, що підлягають управлінню, забезпечує ефективний контроль ресурсів, завдяки чому можливо здійснювати контрольований та організований моніторинг проекту. PRINCE2 забезпечує єдину термінологію для всіх учасників проекту. Різноманітні ролі управлінців та зони відповідальності повністю описані та можуть бути адаптовані відповідно до складності проекту та компетенцій організації.

Помилки використання

Іноді PRINCE2 вважають не доцільним для дуже маленьких проектів через обсяг роботи, необхідної для створення та оновлення документів, протоколів та реєстрів. Досить часто це трапляється через нерозуміння, які частини PRINCE2 використовувати, адже PRINCE2 можливо повністю масштабувати.

Звичайно все це чудово, але сприйняття PRINCE2 погіршене тим, що для кожного ефективного проекту, який використовує методологію PRINCE2 існує певна кількість не ефективних проектів, які використовують, так звану, методологію PINO (PRINCE In Name Only – дослівно, PRINCE лише в назві).

Загальними проблемами є:

– Стадія початку проекту опускається/проштовхується і організація переходить відразу до створення Документу ініціювання проекту (Project initiation document – PID).

– Рада проекту не є ефективною – ескалації уходять в «чорну діру» тощо.

– Відсутній Опис продукту (Product description – PD), без якого управління якістю в PRINCE2 не працює.

– Припустимі відхилення не встановлюються, або встановлюються лише для часу виконання та вартості – чітко не визначено, в якій мірі повноваження

передані виконавцям або коли необхідно використовувати ескалацію (піднімати рівень прийняття рішень).

– Проект фактично складається з однієї стадії (або «фази» використовуються, щоб уникнути Огляду виконання стадії завершення (End Stage Review)).

– Документи ініціювання проекту фактично копі–паст (copy–paste – копіювати – вставити) Документу ініціювання проекту з останнього проекту – їм не слідують і вони створюються тому що «потрібно».

– Низький рівень залучення «бізнесу», адже PRINCE2 вважається «технічним» методом – виходи (продукти) проекту можуть не призвести до бажаних результатів.

– Успіхи та невдачі використання PRINCE2 були однією з причин створення Моделі зрілості PRINCE2 (PRINCE2 Maturity Model – P2MM).

Модель зрілості PRINCE2. Модель зрілості PRINCE2 описує набір Сфер Ключових Процесів (Key Process Areas – KPAs), які необхідні для ефективного впровадження та використання PRINCE2 в організації. Довідник PRINCE2 описує управління проектом і не включає опис процесів впровадження PRINCE2 в організації, в той час як P2MM описує саме процеси впровадження PRINCE2.

P2MM описує ключові практики, які додаються до процесів та складових PRINCE2 з метою забезпечення повторюваного використання методу (Рівень 2 KPAs), а також ключові практики необхідні для впровадження методу (Рівень 3 KPAs), як стандартного бізнес–процесу управління проектами, що включає: призначення власника (3.1), адаптацію методу (3.2), навчання (3.3), інтеграцію з іншими системами управління (3.4), механізм забезпечення якості (3.5).

Огляд методу

Блок–схема процесів PRINCE2. Стрілки означають потоки інформації. PRINCE2 – це процесуальний метод управління проектами на відміну від адаптивних методів, таких як Scrum. PRINCE2 2009 визначає 40 окремих активностей (завдань) і об'єднує їх у сім процесів.

Початок проекту.

На цьому етапі визначаються учасники проектної команди та готується короткий опис проекту (опис цілей проекту та комерційне обґрунтування проекту). Відповідно до загальних засад методу на цьому ж етапі планується наступний етап проекту. Після завершення зазначених робіт Рада проекту має погодити наступний етап проекту – ініціювання проекту.

Головні завдання включають: призначення керівника проекту та менеджера проекту, визначення та призначення команди управління проектом, створення короткого опису проекту, визначення методу управління проектом, планування наступного етапу проекту (ініціювання).

Ініціювання проекту

Цей етап базується на результатах виконання завдань етапу початку проекту. Короткий опис проекту розширюється до бізнес плану (Business case). Точки контролю якості проекту узгоджуються з точками контролю стану виконання проекту. Створюється план виконання наступного етапу проекту. Результати етапу виносяться на погодження Ради проекту з метою погодження продовження роботи над проектом.

Головні завдання включають: планування показників якості, створення плану проекту, деталізація бізнес плану та ризиків, визначення точок контролю проекту, створення Документу ініціювання проекту (Project Initiation Document).

Стратегічне управління проектом

Цей процес визначає яким чином Рада проекту (що уособлює роль спонсору проекту) буде загалом контролювати проект. Рада проекту дозволяє/санкціонує етап ініціювання проекту, а також сам проект. Стратегічне управління також визначає яким чином Рада проекту має погоджувати план етапів, включаючи погодження будь–якого плану етапу, що може змінити існуючий план проекту у зв'язку з будь–якими змінами строків або ключових показників проекту. Також визначається, яким чином Рада проекту може змінити план проекту, способи закриття проекту. Головні завдання включають: засоби

погодження ініціювання проекту, погодження проекту, погодження етапу або вилучення етапу проекту, підтвердження завершення проекту.

Контроль

PRINCE2 передбачає поділ проекту на етапи, які підлягають контролю. Загалом визначено яким чином пакети завдань призначаються та погоджуються. Також визначається яким чином відслідковується статус виконання завдань і яким чином такий статус доповідається Раді проекту. Засоби для відслідковування та оцінювання виконання проекту пропонуються разом з способами застосування коригуючих дій. Також закріплюються методи ескалації визначеного кола проблем Раді проекту.

Головні завдання включають: призначення та погодження пакетів завдань, оцінювання статусу виконання завдань, звітування статусу виконання завдань, застосування коригуючи дій, ескалація проблем проекту, підтвердження завершення пакету завдань.

Управління межами етапів

Контроль виконання етапів проекту – це визначення переліку завдань, які мають бути виконані в межах етапу. Управління межами етапу визначає, що має бути виконано при завершені етапу. При завершені етапу необхідно створити план виконання наступного етапу. Загальний план проекту, перелік ризиків проекту та бізнес план за необхідності переглядається та оновлюється. Також визначається перелік дій на випадок, якщо етап не відповідає показникам виконання. Загалом визначається, яким чином звітується про завершення етапу.

Головні завдання включають: планування етапу, оновлення плану проекту, оновлення бізнес плану проекту, оновлення ризиків проекту, звітування про завершення етапу, створення плану дій на випадок не виконання певних завдань проекту.

Завершення проекту

Визначаються завдання, які повинні бути виконані при завершені проекту. Проект має бути формально завершений (ресурси мають бути вивільнені для

виконання інших завдань), результат виконання завдань має бути визначений, а сам проект має отримати формальну оцінку.

Головні завдання включають: завершення проекту, визначення результату виконання завдань, оцінку виконання проекту.

4.2 Захист розробленого програмного забезпечення

Для захисту розробленого програмного забезпечення запропоновано використовувати алгоритм MMB, в основі якого лежить змішування операцій різних алгебраїчних груп. MMB – ітеративний алгоритм, що складається з лінійних дій (XOR і використання ключа) і паралельного застосування чотирьох великих оборотних нелінійних підстановок. Ці підстановки визначаються за допомогою множення по модулю 2^{32}-1 з постійними множниками. У підсумку з'являється алгоритм, що використовує 128-бітовий ключ і 128-бітовий блок.

Алгоритм MMB оперує 32-бітовими підблоками тексту (x_0, x_1, x_2, x_3) і 32-бітовими підблоками ключу (k_0, k_1, k_2, k_3). Це спрощує реалізацію алгоритму на сучасних 64-бітових процесорах. Чергуючись із операцією XOR, шість разів використовується нелінійна функція *f*. Запишемо операції алгоритму (всі операції з індексами виконуються по модулю 4):

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_i \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+1} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

$$x_i = x_i \oplus k_{i+2} \text{ для } i = 0..3$$

$$f(x_0, x_1, x_2, x_3)$$

Функція *f* виконується в три кроки:

1. $x_i = c_i * x_i$ для $i = 0..3$ (Якщо на вході множення одні одиниці, то на виході – теж одні одиниці).

2. Якщо молодший значущий біт $x_0 = 1$, то $x_0 = x_0 \oplus C$. Якщо молодший значущий байт $x_3 = 0$, то $x_3 = x_3 \oplus C$.

3. $x_i = x_{i-1} \oplus x_i \oplus x_{i+1}$ для $i = 0..3$.

Всі операції з індексами виконуються по модулю 4. Операція множення на кроці 1 виконується по модулі 2^{32}-1. Спеціальний випадок для даного алгоритму: якщо другий операнд дорівнює 2^{32}-1, результат теж дорівнює 2^{32}-1. В алгоритмі використовуються наступні константи:

$$C = 2\text{aaaaaaa}, c_0 = 025\text{f1cdb}, c_1 = 2*c_0, c_2 = 2^3 * c_0, c_3 = 2^7 * c_0.$$

Константа C – «найпростіша» константа без кругової симетрії, високою трійковою вагою й нульовим молодшим значущим бітом. У константи c_0 є інші особливі характеристики. Константи c_1, c_2 і c_3 – зрушені версії c_0, і служать для запобігання атак, заснованих на симетрії.

Розшифрування виконується у зворотному порядку, Етапи 2 і 3 інверсні їм самим. На етапі 1 замість c_i використовується c_i^{-1}. Значення c_0^{-1} = 0dad4694.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено розроблене у бакалаврської дипломної роботі програмне забезпечення системи безпечного зберігання даних у хмарі за рахунок Cloud Controls Matrix. З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші.

– Верхнього меню: Файл; Фільтри; Налаштування; ССМ дані; Довідка.

– Розділу обрання групи.

– Розділу виведення результату роботи системи.

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення.

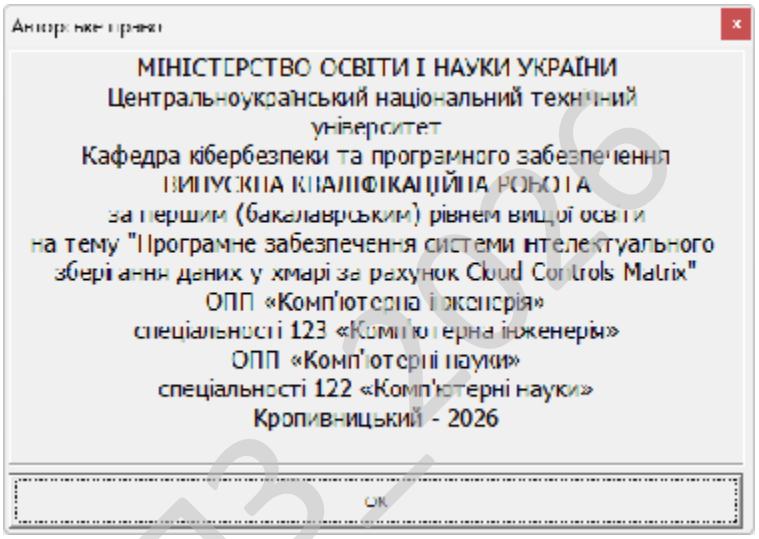

Рисунок 5.2 – Авторське право

Розглянемо процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення.

Загалом процес розгортання складається з кількох взаємопов'язаних дій із можливими переходами між ними. Ця активність може відбуватися як з боку виробника так і з боку споживача. Оскільки кожна програмна система є

унікальною, то усі процеси та процедури під час розгортання важко передбачити. Тому, "розгортання" можна трактувати як загальний процес відповідно до певних вимог та характеристик. Розгортання може здійснюватись програмістом і в процесі розробки програмного забезпечення.

До діяльностей пов'язаних із розгортанням програмного забезпечення відносять:

– Випуск.

– Встановлення та активація.

– Деактивація.

– Адаптація.

– Обновлення.

– Вмонтування.

– Відстежування версій.

– Видалення.

– Вилучення з обігу.

При впровадженні програмного забезпечення потрібно урахувати наступні дії:

– Виділення критичних, з точки зору загального результату, процедур в діяльності організації. Коли набір таких процедур визначений, необхідно в першу чергу використовувати IT рішення для автоматизації операцій усередині саме цих процедур. Таким чином, розроблене IT рішення автоматично стає життєво важливим і затребуваним для організації, а також буде забезпечена публічність процесу впровадження;

– Розширення нормативної бази організації шляхом включення до неї регламентів, що описують порядок виконання процедур автоматизованих процесів. В іншому випадку є небезпека виникнення неузгодженості між автоматизованими процедурами та іншими процесами організації.

– Виконання робіт з загальної стандартизації існуючої діяльності організації, коли виділяються кращі практики виконання процедур і включаються

в ІТ рішення за принципом найбільшої корисності для більшості учасників. Відсоток таких процедур щодо загального обсягу автоматизації може бути невеликий, але це надає процесу побудови рішення вагу в організації за рахунок збільшення його необхідності.

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 скриньки 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.

Проводилось тестування чорної скриньки.

Основне місце програми тестів «чорної скриньки» — інтерфейс ПЗ. Відомі: функції програми. Досліджується: робота кожної функції на всій області визначення.

Ці тести демонструють:

– Як виконуються функції програми.

– Як приймаються вихідні дані.

– Як виробляються результати.

– Як зберігається цілісність зовнішньої інформації.

При тестуванні «чорної скриньки» розглядаються системні характеристики програм, ігнорується їхня внутрішня логічна структура. Вичерпне тестування, як правило, неможливе.

Наприклад, якщо в програмі 10 вхідних величин і кожна приймає по 10 значень, то кількість тестових варіантів становитиме 10^10. Тестування «чорної скриньки» не реагує на багато особливостей програмних помилок.

Тестування «чорної скриньки» (функціональне тестування) дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Програмний виріб тут розглядається як «чорна скринька», чию поведінку можна визначити тільки дослідженням його входів та відповідних виходів. При такому підході бажано мати:

– Набір, утворений такими вхідними даними, які призводять до аномалій у поведінці програми (назвемо його ITc).

– Набір, утворений такими вхідними даними, які демонструють дефекти програми (назвемо його OT).

Будь-який спосіб тестування «чорної скриньки» повинен:

– Виявити такі вхідні дані, які з високою ймовірністю належать набору ITc;

– Сформулювати такі очікувані результати, які з високою імовірністю є елементами набору OT.

Принцип «чорної скриньки» не альтернативний принципу «білої скриньки». Скоріше це доповнює підхід, який виявляє інший клас помилок.

Тестування «чорної скриньки» забезпечує пошук наступних категорій помилок:

– Некоректних чи відсутніх функці.

– Помилок інтерфейсу.

– Помилок у зовнішніх структурах даних або в доступі до зовнішньої бази даних.

– Помилок характеристик (необхідна ємність пам'яті і т.д..

– Помилок ініціалізації та завершення.

Обрано умови розповсюдження – Shareware.

Під умовно-безплатним програмним забезпеченням можна розуміти спосіб або метод розповсюдження комерційного ПЗ на ринку (тобто на шляху до кінцевого користувача), при якому випробувачеві пропонується обмежена за можливостями (не повнофункціональна або демонстраційна версія), терміном дії (тріал версія) або версія з вбудованим набридливим нагадуванням про необхідність оплати використання програми.

В угоді про використання (ліцензії для кінцевого користувача, EULA) також може бути обумовлена заборона на комерційне або професійне (не тестове) її використання.

Основний принцип умовно-безплатного ПЗ – «спробуй, перш ніж купити» (try before you buy). ПЗ що поширюється як умовно-безплатний, надається користувачам безоплатно. Звичайно користувач платить тільки за час завантаження файлів через Інтернет або за носій (CD диск, флешку, ключ). Протягом певного терміну, що становить зазвичай тридцять днів, він може користуватися програмою, тестувати її, освоювати її можливості.

Якщо після закінчення цього терміну користувач вирішить продовжити використання ПЗ, він зобов'язаний купити його (зареєструватися), заплативши авторові певну суму.

В іншому випадку користувач повинен припинити використання ПЗ та видалити його зі свого комп'ютера.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

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

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

– Досліджена система інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

– На основі отриманих результатів досліджень створена програмна реалізація системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix.

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 Visual C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи інтелектуального зберігання даних у хмарі за рахунок Cloud Controls Matrix. 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 й включає програми, що реалізують її функції.

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

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм ММВ.

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. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 1». Cisco Press. 2020. – 848 p.

2. Wendell Odom. «CCNA 200-301 Official Cert Guide, Volume 2 Premium Edition eBook and Practice Test». Cisco Press. 2020. – 624 p.

3. Scott Jernigan «CompTIA Network+ Certification All-in-One Exam Guide, Eighth Edition». 2022. – 976 p.

4. Doug Lowe «Networking For Dummies 12th Edition». 2020. – 480 p.

5. Ramon Nastase «Computer Networking: The Beginner's guide for Mastering Computer Networking, the Internet and the OSI Model». 2018. – 186 p.

6. Russ White & Ethan Banks «Computer Networking Problems and Solutions: An Innovative Approach to Building Resilient, Modern Networks». 2017. – 832 p.

7. Вінтенко Б., Смірнов О., Миронець І., Смірнова Т., Смірнов С. «Імітаційна модель шляхів вхідних даних комп'ютерної інтелектуальної системи підтримки оператора енергоблоку АЕС». *Комбінаторні конфігурації та їхні застосування: Матеріали XXVII Міжнародного науково-практичного семінару, присвяченого 125-річчю Національного університету «Запорізька політехніка»* (Запоріжжя-Кропивницький-Київ, 4-6 червня 2025 р.). Запоріжжя: НУ «Запорізька політехніка», 2025. С.82-91.

8. Al-Azzeh, J., Ayyoub, B., Mesleh, A., Smirnova, T., Gnatyuk, S., Drieiev, O., Smirnov, O., Dorenskyi, O. «Cloud-Based Information System for Evaluating Caverns in the Process of Blasting Metal Surfaces of Details». *International Review on Modelling and Simulations* 18 (1), 2025. pp. 32-42.

9. Смірнова Т.В., Конопліцька-Слободенюк О.К., Буравченко К.О., Смірнов С.А., Кравчук О.В., Козірова Н.Л., Смірнов О.А. «Дослідження технологій забезпечення кібербезпеки хмарних сервісів IaaS, PaaS та SaaS». *Кібербезпека: освіта, наука, техніка.* 2024. №4(24), С. 6-27.

10. Батрак О., Смірнова Т., Гнатюк В., Одарченко Р., Смірнов О. «Дослідження показників ефективності функціонування та перспектив розвитку систем IP-телефонії». *Підводні технології,* 2024, № 13, с. 28-35.

11. 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.

12. 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.

13. 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.

14. Smirnov, O., Odarchenko, R., Smirnova, T., Bondar, S., Volosheniuk, D. «Optimal Structure Construction of Private 5G Network for the Needs of Enterprises». *Lecture Notes on Data Engineering and Communications Technologies*, 2023, 178, pp. 208–223.

15. 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.

16. Smirnova, T., Gnatyuk, S., Yudin, O., Sydorenko, V., Polozhentsev, A., «The Model for Calculating the Quantitative Criteria for Assessing the Security Level of Information and Telecommunication Systems». CEUR Workshop Proceedings Volume 3156, 2022, Pages 390-399.

17. Смірнова Т.В., Гнатюк С.О., Сидоренко В.М., Юдін О.Ю., Сидоренко С.Ю., «Модель визначення критичності галузевих інформаційно-телекомунікаційних систем». *Проблеми інформатизації та управління,* № 2(70). 2022. С. 28-37.

18. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98.

19. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

20. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

21. Smirnova T., Gnatyuk S., Berdibayev R., Avkurova Zh., Iavich M. «Cloud-Based Cyber Incidents Response System and Software Tools». *Communications in Computer and Information Science*, 2021, vol 1486. Springer, Cham. pp 169-184.

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

23. Smirnov O., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346.

24. 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.

25. 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.

26. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136.

27. 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.

28. 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.

29. 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.

30. 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.

31. 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.

32. Smirnov, O., Odarchenko, R., Abakumova, A., Usik, P., Kundyz, M., «QoE optimization technique for media delivery in 5G networks». *2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T),* Kyiv, Ukraine, 8 – 11 October 2019. P.597-601.

33. 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.

34. 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.

35. 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.

36. 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.

37. 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.

38. Smirnov A.A., Kuznetsov 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.

39. 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

40. 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.

41. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

42. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В., Поліщук Л.І. Інформаційна безпека в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2020. – 294 с.

43. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

44. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

45. Смірнов О.А., Конопліцька-Слободенюк О.К., Смірнов С.А., Буравченко К.О., Смірнова Т.В. Поліщук Л.І. Проектування комп'ютерних систем та мереж. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2019. – 264 с.

46. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. Collective monograph. Edited by Ivan

D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

47. Смірнов О.А., Дрєєва Г.М. Метод генерування фрактального трафіку за допомогою моделі генератора на графі. Монографія: Інформаційна безпека та інформаційні технології : монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139

48. Дрєєва Г.М., Смірнов О.А., Дрєєв О.М. Метод генерування фрактальноподібної числової послідовності на основі скінченного автомату для моделювання трафіку у мережі. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 173-183, 2019.

49. Смірнова Т.В., Солових Є.К., Смірнов О.А., Дрєєв О.М. Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей. Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

50. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

51. Смірнов О.А., Гнатюк С.О., Кавун С.В., Терейковський І.А., Жмурко Т.О., Смірнов С.А., Коваленко А.С. Основи безпеки в комп'ютерних мережах. Навчальний посібник – Кропивницький: вид. Лисенко В.Ф. 2018. – 177 с.

52. Смірнов О.А., Котелянець В.В. Стійкі до колізій стохастичні моделі функціонування безпроводових сенсорних мереж. Вісник інженерної академії України, №3, с. 145-152, 2018