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

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

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за другим (магістерським) рівнем вищої освіти
на тему
“Дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile”

Виконав здобувач вищої освіти
ІІ курсу, групи КІ-24М
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Дроздов М.Р.
«____» _____ 2025 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Буравченко К.О.
«____» _____ 2025 р.
Рецензент _____
_____

м. Кропивницький

АНОТАЦІЯ

Дроздов М.Р. Дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2025.

В даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Метою розробки є дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Об'єктом дослідження є процес захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Предметом дослідження є методи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Методи дослідження базуються на методах захисту інформації у комп'ютерних мережах, методах математичної статистики, методах розробки програмного забезпечення.

Результат роботи – програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

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'ютерна інженерія, захист корпоративних додатків, Agile

ABSTRACT

Drozdov M.R. Research and software implementation of a corporate application protection system developed using the Agile methodology. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2025.

In this final qualification work for the second (master's) level of higher education, software has been developed that is intended for a corporate application protection system developed using the Agile methodology.

The purpose of the development is the research and software implementation of a corporate application protection system developed using the Agile methodology.

The object of the research is the process of protecting corporate applications developed using the Agile methodology.

The subject of the research is methods for protecting corporate applications developed using the Agile methodology.

The research methods are based on methods of information protection in computer networks, methods of mathematical statistics, methods of software development.

The result of the work is a software implementation of a corporate application protection system developed using the Agile methodology.

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 user-friendly user interface has been developed. Instructions for working with the software are provided.

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

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

Keywords: computer engineering, corporate application protection, Agile

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЕН	–	електронний нотаріус
ЕЦП	–	електронний цифровий підпис
СУБД	–	система управління базами даних
УЦ	–	удостовірюючий центр
CRL	–	список відкликаних сертифікатів
DVCS	–	Data Validation and Certification Server Protocols
http	–	HyperText Transfer Protocol – протокол передачі гіпертексту
HTTPS	–	розширення протоколу HTTP, підтримуюче шифрування
IMAP	–	протокол доступу до електроної пошти Інтернету
MAC	–	код автентифікації повідомлення
OCSP	–	Online Certificate Status Protocol
PKI	–	інфраструктура відкритих ключів
POP3	–	Post Office Protocol Version 3 – протокол поштового відділення
SMTP	–	простий протокол передачі пошти
SSL	–	Secure Sockets Layer – рівень захищених сокетів
SQL	–	Structured Query Language – мова структурованих запитів
TCP	–	протокол управління передачею
TLS	–	криптографічний протокол, захищаючий передачу даних
TSP	–	Time-Stamp Protocol
URL	–	єдиний показник ресурсів
XMPP	–	розширяємий протокол обміну повідомленнями й інформацією про присутність

ВСТУП

Актуальність теми. Гнучка розробка програмного забезпечення (Agile) – це тип методології, який зосереджений на основному принципі гнучкості. Методи гнучкої розробки визнають, що для отримання найкращих кінцевих продуктів у найкоротші терміни може знадобитися зміна культури або мислення поряд з іншими змінами в процесі. Незважаючи на широке впровадження з моменту першого впровадження на початку 2000-х років з Agile-маніфест, за останні п'ять років DevOps став стандартною методологією, яка використовується для створення чудових програмних продуктів. Однак сьогодні, DevSecOps – це золотий стандарту розробці безпечних додатків.

Створення гнучкого середовища розробки пропонує кілька ключових переваг. Сучасний бізнес працює швидше, ніж будь-коли раніше, тому команди розробників повинні мати можливість своєчасно реагувати на ринкові сили, водночас надаючи пріоритет ефективній доставці програми. Agile надає платформу, яка дозволяє командам швидко рухатися та створювати високоякісні продукти завдяки таким методам, як постійне тестування протягом усього процесу розробки. Пошук більш гнучких методів розробки дозволяє забезпечити безпечніший життєвий цикл розробки програм від коду до хмари. Успішне впровадження практик безпеки разом зі гнучкою розробкою програмного забезпечення дозволяє швидко розробляти безпечний код, або DevSecOps.

Мета й завдання дослідження. Метою роботи є дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

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

– Огляд існуючих систем захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– Дослідження системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– Програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Об'єктом дослідження є процес захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Предметом дослідження є методи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Методи дослідження базуються на методах захисту інформації у комп'ютерних мережах, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– Розроблено вітчизняний продукт захисту корпоративних додатків, які розробляються за допомогою методології Agile, який має більш широкі можливості, на відміну від існуючих аналогів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Достовірність наукових результатів підтверджена теоретичними викладеннями, даними комп'ютерного моделювання, коректними дослідженнями параметрів на функціонуючій обчислювальній мережі, а також відповідністю отриманих результатів окремим результатам, наведеним у науковій літературі.

Робота апробована на LVII Науково-технічній конференції здобувачів вищої освіти LV науково-технічної конференції «Наука в ЦНТУ: основні досягнення та перспективи розвитку» (2025 р.), основні положення випускної

кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти надруковані у статті збірника праць молодих науковців ЦНТУ, випуск №15.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Бізнесу потрібні новий функціонал, перевірка гіпотез і швидка реакція на зовнішні умови, а вивчення й переналаштування систем захисту займають усе більше часу. Що робити програмістам, поки створені ними нові функції вивчають фахівці з безпеки? Відпочивати? За чий рахунок? Або писати новий код? Тоді навіщо «безпечникам» тестувати те, що застаріє до закінчення тестування? А якщо уразливості будуть знайдені? Прийдеться відкладати розробку нових функцій і виправляти старі? Що на це скажуть представники бізнесу? Як знайти компроміс?

Питань багато. Є кілька методик виходу із цього замкнутого кола й повернення до збіжності процесу змін і процесу захисту систем, у яких ці зміни присутні.

Перший спосіб – введення шару безпеки на всіх рівнях розробки й підтримки системи. Інакше кажучи, фахівці з безпеки повинні залучатися не на етапі тестування закінченого об'єкта, а на етапі проектування чергових змін. Очевидно, швидкість впровадження змін сповільниться, але не настільки, наскільки їх затримує поточний процес.

Другий спосіб – підключення штучного інтелекту й машинного навчання, використання самоналаштувальних систем захисту, які без участі людини в «бойовому» режимі тестують системи на уразливості, видають рекомендації з виправлення й самі ж захищають виявлені уразливі місця за допомогою віртуальних патчів або зміни налаштувань.

Перший спосіб повинен повністю змінити підхід до розробки, а другий – до захисту.

Сьогодні розробка – це «держава в державі», окремий підрозділ, іноді й за штатом основного виробництва. Вона живе за своїми правилами й контактує із внутрішнім або зовнішнім замовником тільки тоді, коли одержує технічне завдання або здає нову версію. По тім же принципі організована й служба безпеки. Зробити процес загальним, «розмазати» розробку й захист додатка по всіх стадіях життєвого циклу – складний виклик, що вимагає повної зміни звичних процесів. Тому такий підхід краще впроваджувати при запуску нового проекту з нуля, надія успішно змінити вже існуючі схеми взаємодії – утопія.

Міняти підхід до безпеки теж непросто. Сьогодні безпека – це величезні центри моніторингу, у яких засобу захисту тільки видають гіпотези про атаки, а рішення на підставі цих даних приймають люди. Автоматичні засоби не активуються з побоювання помилкових спрацьовувань, у результаті яких корпоративні додатки можуть виявитися недоступні легальним користувачам. Передати ухвалення рішення системам зі штучним інтелектом – те ж саме, що сісти в автомобіль із автопілотом. Це, скоріше, не питання якості, а питання довіри.

Але, так чи інакше, вибір прийде робити. Традиційні парадигми побудови й захисту більших систем практично вичерпали свої ресурси. Компанії, бізнес яких невіддільний від додатків: портали, соцмережі, електронні магазини, онлайн-банки й інші, – уже активно пробують, а іноді й застосовують нові методології й технології. Може бути, пора про це задуматися й тим, хто надто прихильний традиціям?

1.2 Область застосування

Областю застосування системи є захист корпоративних додатків. Сьогодні мало хто ставить під сумнів необхідність додаткового захисту сучасних ОС сімейства Windows. Однак виникають питання: які завдання повинне вирішувати додатковий засіб захисту, у якій частині і якому образі варто підсилювати

убудовані механізми ОС? Все різноманіття загроз комп'ютерної безпеки можна звести до двох більших груп: внутрішні й зовнішні IT-загрози.

Поява поняття внутрішніх IT-загроз для ОС Windows пов'язане з початком використання ОС сімейства Windows для обробки конфіденційної інформації в корпоративних додатках. Конфіденційна інформація апріорі є потенційним об'єктом несанкціонованого доступу (НСД), тому що має споживчу вартість для зловмисників, тобто є товаром.

З обліком же того, що в основі архітектури захисту сучасних універсальних ОС сімейства Windows (не будемо забувати, що це універсальні ОС, споконвічно створені як персональні для домашнього використання) лежить принцип повної довіри до користувача, деякі ключові механізми захисту, убудовані в сучасні ОС Windows, із цієї причини не можуть забезпечити ефективної протидії внутрішнім IT-загрозам – загрозам НСД до інформації з боку санкціонованих користувачів (інсайдерів – користувачів, допущених до обробки конфіденційної інформації у рамках виконання своєї службової діяльності), саме санкціоновані користувачі й несуть у собі найбільш імовірну загрозу розкрадання конфіденційних дані підприємства.

Як наслідок, це завдання сьогодні не вирішується убудованими механізмами захисту ОС Windows, тому її рішення повинне бути покладене на додаткові засоби захисту. Поняття зовнішніх IT-загроз для ОС Windows з'явилося набагато раніше, але знову ж повною мірою виявилося з початком використання ОС сімейства Windows для обробки конфіденційної інформації в корпоративних додатках, що знову ж пов'язане з високою споживчою вартістю конфіденційних даних для зловмисників. І тут ключовою причиною, на наш погляд, є історичний підхід до створення універсальних ОС. Незважаючи на те що архітектура захисту при переході від версії до версії перетерплює помітні зміни, вона й донині має принципові архітектурні недоліки (наприклад, помилки в додатках ніяк не повинні позначатися на рівні безпеки інформації, захист якої здійснює ОС).

Інша проблема криється в помилках, що виявляються систематично, програмування. Тут, загалом кажучи, виходить якесь замкнуте коло. Очевидно, необхідно кардинально міняти архітектуру захисту, що вимагає часу й серйозного пророблення рішень, але ринок вимагає зворотного – необхідно максимально швидко створювати й поставляти на ринок рішення з новими споживчими властивостями, а це можливо лише за умови максимального використання існуючого програмного коду. Коли ж мова заходить про споживчі властивості, то в першу чергу розроблювачем розширюються ті властивості продукту, які максимально затребувані (якщо виріб максимально орієнтується на використання приватними особами, те аж ніяк не забезпечуваний рівень інформаційної безпеки стає його основною споживчою властивістю).

Висновок: на сьогоднішній день у корпоративних додатках при захисті конфіденційної інформації рівною мірою актуальні завдання протидії й внутрішнім, і зовнішнім IT-загрозам.

На підставі даного виводу можна укласти, що додатковий засіб захисту конфіденційної інформації, використовуване в корпоративних додатках, повинне бути комплексним – повинне вирішувати завдання захисту інформації в частині протидії як внутрішнім, так і зовнішнім IT-загрозам.

Таким чином, виходячи з вищеперерахованого, дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

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

Недавні дослідження показали, що майже 87% пристроїв на уразливі до атак. Це пов'язане з тим, що не проводяться відновлення системи безпеки. Від інтернету не захиститися, поки пристрій підключений до мережі. Забезпечення безпеки в наш час є однією з основних завдань, тому з'явилося багато сторонніх додатків для її підвищення.

Наступні додатки допоможуть захистити ваші пристрої від загрози безпеки й особистих даних.

Список додатків для захисту пристроїв:

– Avast Mobile Security.
– Sophos Antivirus and security.
– AppLock.
– Signal Private Messenger.
– Secure Call.
– App Ops.
– Lastpass.
– Device Manager.
– NoRoot Firewall.
– Orbot.

1. Avast Mobile Security

Avast – це прекрасний додаток для захисту телефону на базі від вірусів і інших загроз.

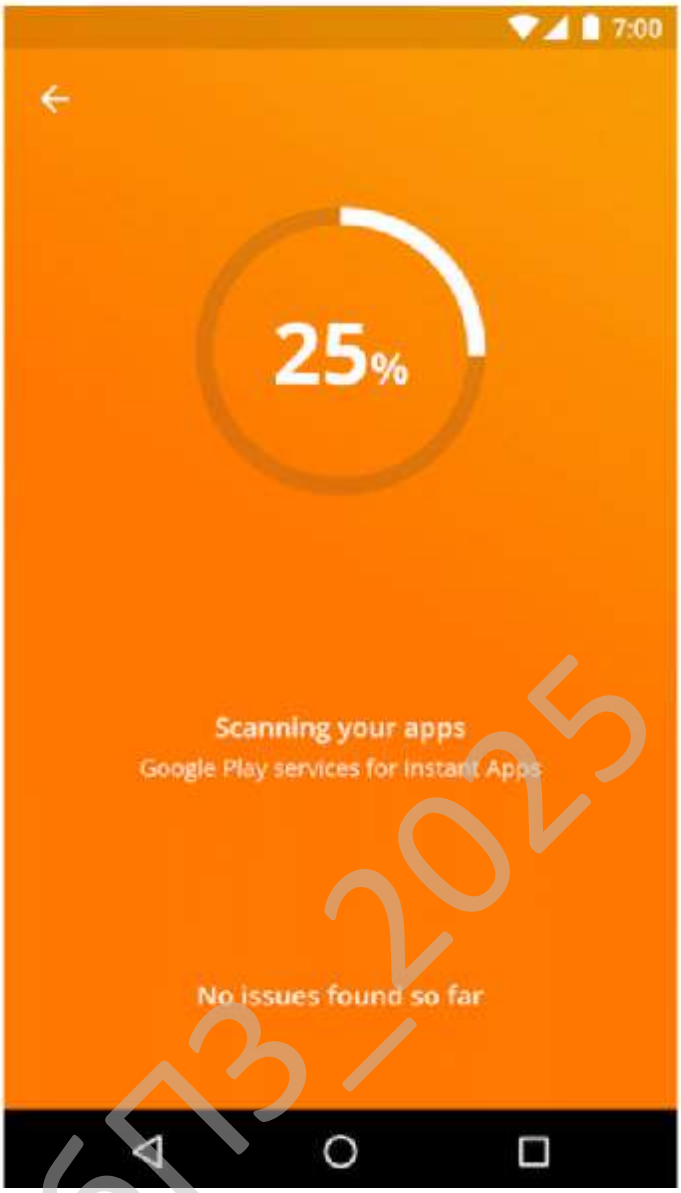

Рисунок 2.1 – Інтерфейс користувача Avast

Avast – це один з найбільш популярних безкоштовних антивірусів. Він повідомляє про установку шпигунських і рекламних програм, які загрожують захищеності ваших особистих даних.

Імовірність виявлення новітніх шкідливих програм приблизно 99,9%, а у випадку шкідливих програм, що з'явилися протягом місяця, імовірність практично 100%.

Висновок: Якщо вам необхідний захист від шкідливих програм і для безпечного перегляду сайтів, то вам цей додаток підійде.

2. Sophos Antivirus and security

Sophos – один із кращих безкоштовних антивірусів.

Рисунок 2.2 – Інтерфейс користувача Sophos

Користувальницький інтерфейс може не викликати замилування. Однак, функціонал дозволить вам перестати турбуватися про безпеку.

Можливості:

– сканування на наявність вірусів установлених і існуючих додатків, а також сховищ даних;

– захист від втрати й злодійства з підтримкою вилученого доступу, що дозволяє форматувати, заблокувати, включити звуковий сигнал на вашім пристрої або встановити його місце розташування;

– фільтрація веб-контенту;

– блокування спаму.

В Sophos найвища ймовірність виявлення новітніх шкідливих програм – 100%. Цим вона разюче відрізняється від інших.

Висновок: Якщо корисні функції для вас важливіше симпатичного дизайну, те краще Sophos ви мало що знайдете.

3. AppLock

Цим додатком досить просто користуватися. AppLock захищає окремі додатки від зломщиків, запитуючи PIN-код або графічний ключ. Таким способом можна захистити SMS, контакти, Gmail, та й взагалі будь-який додаток.

Не переплутайте таке блокування додатків і убудовану в телефон блокування пристрою. Убудоване блокування блокує весь телефон. Немає доступу ні в які додатки. У свою чергу AppLock дозволяє заблокувати вибрані додатки.

Висновок: Якщо вам потрібно запобігти доступу зловмисників до окремих додатків, але ви не хочете захищати паролем пристрій у цілому, то для цього підійде Applock.

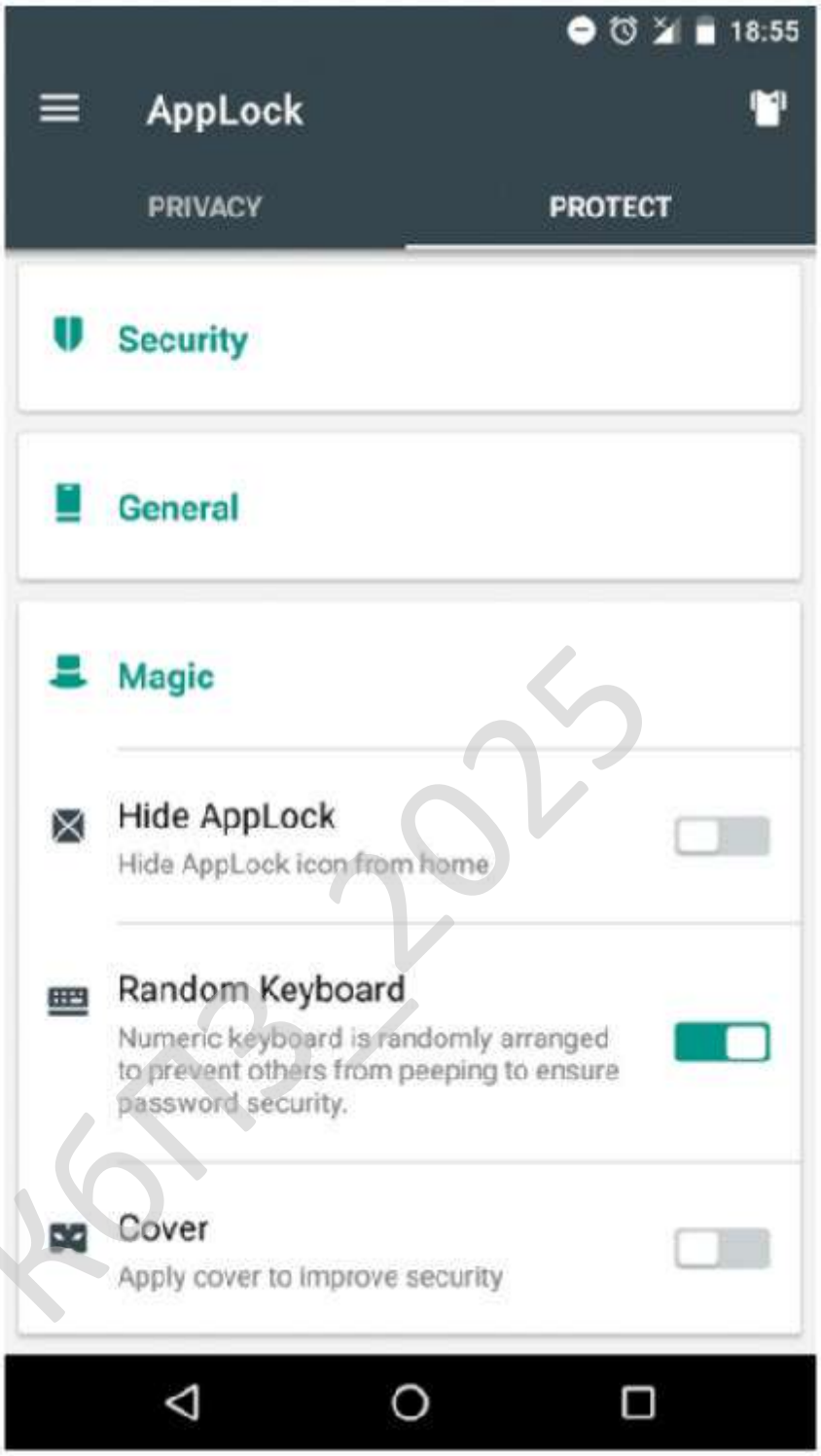

Рисунок 2.3 – Інтерфейс користувача Applock

4. Signal Private Messenger

Існує множин додатків для безпечного обміну повідомленнями. Більшість із них працюють тільки, якщо обоє користувача використовують те саме додаток.

Однак, Signal Private Messenger дозволяє додати додатковий рівень захисту до звичайних SMS, навіть якщо один з користувачів не користується Signal Private Messenger. Додаток розроблений Open Whisper System.

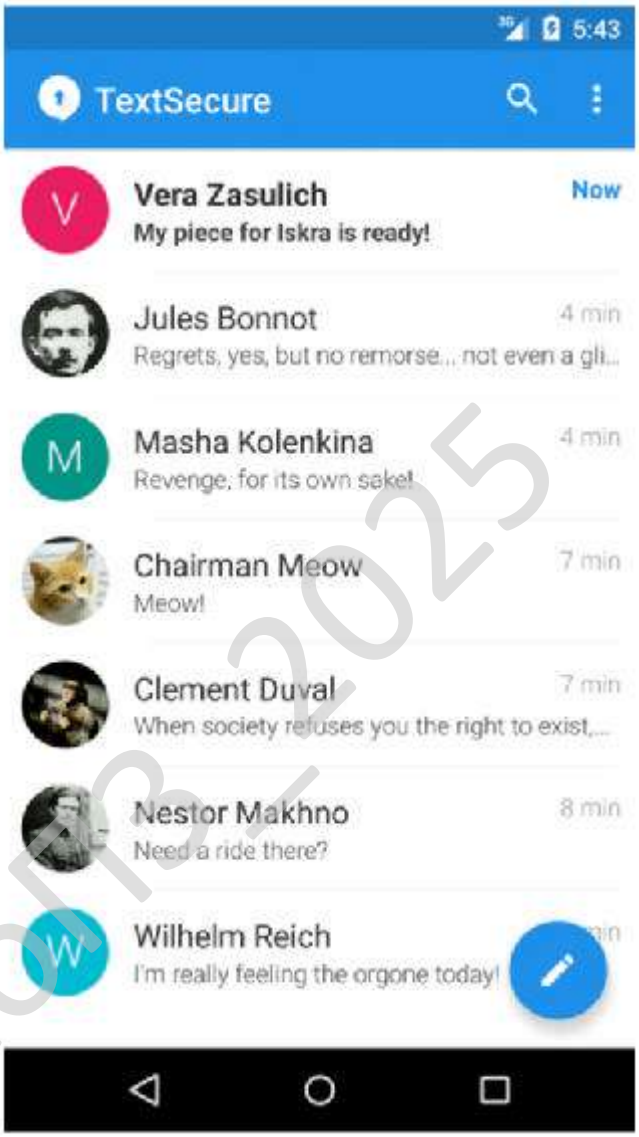

Рисунок 2.4 – Інтерфейс користувача Signal Private Messenger

Додаток має наступні особливості:

– відкритий вихідний код;

– наскрізне шифрування. На сервері додатка не зберігається нічого;

– шифрування можливо навіть якщо в одного з користувачів немає Signal Private Messenger.

Висновок: Для наскрізного шифрування звичайних SMS краще Signal Private Messenger нічого немає.

5. Secure Call

Гарантує, що ніхто не зможе прослуховувати дзвінки. Secure Call забезпечує наскрізне шифрування дзвінків, що запобігає прослуховування сторонніми особами.

Додаток використовується за замовчуванням для вхідних і вихідних дзвінків. Завдяки децентралізованій архітектурі (peer-to-peer) з надійним наскрізним шифруванням ніякі сторонні особи не зможуть прослухати ваші дзвінки, у тому числі самі розроблювачі додатка.

Висновок: якщо вам потрібне наскрізне шифрування дзвінків, користуйтеся Secure Call.

6. App Ops

Основна функція App Ops – анулювати певний набір дозволів в обраних додатків. Багато додатків запитують додаткові дозволи, які жодною мірою не пов'язані з їхніми функціями.

App Ops дозволяє блокувати зайві повноваження. При установці додатка потрібно дозволити доступ до всього, що зажадає додаток.

Якщо ви відхилите який-небудь дозвіл, додаток установлено не буде. App Ops вас виручить, якщо знадобиться установити додаток, при цьому не даючи певних дозволів.

Висновок: App Ops вирішить питання відкликання конкретних непотрібних дозволів.

Рисунок 2.5 – Інтерфейс користувача App Ops

7. Lastpass

В усіх сьогодні по кілька облікових записів і паролів. Пам'ятати їх всі не легко.

LastPass – один із кращих доступних на ринку **менеджерів паролів**. При зберіганні паролів застосовуються додаткові рівні захисту.

Всі ваші конфіденційні дані доступні для вас із будь-якого комп’ютера або мобільного пристрою. Паролі зашифровані один майстер-паролем. Щоб одержати доступ до всіх паролів, потрібно пам'ятати лише пароль LastPass.

Висновок: Просте рішення для зберігання всіх паролів.

8. Device Manager

Device Manager дозволяє включити звуковий сигнал, визначити місце розташування, заблокувати ваш пристрій. Додаток також дозволяє видалити всі

дані із пристрою на випадок, якщо телефон уже остаточно не під вашим контролем.

У багатьох додатках це реалізовано як додаткову функю. Проте, додаток Google установити легше. Також, воно дозволяє зайти у ваш обліковий запис через чужий пристрій і видалити всі дані на ходу.

Висновок: Для вилученого **доступу** Andriod Device Manager ідеальний варіант.

9. NoRoot Firewall

Багато додатків для впусту споживають трафік.

За допомогою NoRoot Firewall ви зможете контролювати доступ додатків без рутинга пристрою. Ви зможете дозволити обраним додаткам доступ у мережу тільки через wifi, тільки через мобільний інтернет або ж повністю заборонити/дозволити.

Висновок: Для тих кому не підходить рутинг, але потрібний файєрвол, підійде NoRoot Firewall.

10. Orbot

Orbot – це додаток для у рамках проекту Tor. Воно дозволяє перенаправляти весь трафік через мережу Tor.

VPN використовує один сервер, а Tor перенаправляє трафік через декілька IP-тунелів, щоб не залишати слідів. Orbot установлює воістину захищене мобільне з'єднання. Дані шифруються повторно.

Шифрування даних проходить багаторазово, доти, поки вони не дійдуть до пункту призначення, де вони дешифруються. Таким чином, відправника не відстежити.

Висновок: Orbot дозволяє вам без праці зашифрувати інтернет-трафік.

Сподіваюся, ці додатки допоможуть зберегти ваш мобільний телефон або пристрій на базі Andriod у цілості й безпеці. Також спробуйте скористатися VPN від SurfEasy для захисту своєї анонімності в мережі.

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 додатків 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 додатків, що мають складний інтерфейс користувача, полегшується підтримка технології OLE і взаємодія з базами даних. Сучасні інтегровані засоби розробки додатків Windows дозволяють автоматизувати процес створення додатка. Для цього використовуються генератори додатків. Програміст відповідає на питання генератора додатків і визначає властивості додатка – чи підтримує воно багатовіконний режим, технологію OLE, тривимірні органи керування, довідкову систему. Генератор додатків, створить додаток, що відповідає вимогам, і надасть вихідні тексти. Користуючись їм як шаблоном, програміст зможе швидко розробляти свої додатки. Подібні засоби автоматизованого створення додатків включені в компілятор Microsoft Visual C++ і називаються MFC AppWizard. Заповнивши кілька діалогових панелей, можна вказати характеристики додатка й одержати його тексти, постачені великими коментарями. MFC AppWizard дозволяє створювати одновіконні й багатовіконні додатки, а також додатки, 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 додатка програмістові прийдеться розробляти самостійно. Вихідний текст додатка, створений MFC AppWizard, стане тільки основою, до якої потрібно підключити інше. Але працюючий шаблон додатка – це вже половина всієї роботи. Вихідні тексти додатків, автоматично отриманих від MFC AppWizard, можуть становити сотні рядків тексту. Набір його вручну був би дуже стомлюючий. Потрібно відзначити, що MFC AppWizard створює тексти додатків 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 додатки 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-додатки 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-додатків, 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 множин 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. Хоча програми, що використовують бібліотеку MFC, звичайно не містять занадто специфічних елементів з арсеналу C++, для їхнього написання проте потрібні солідні знання в даній області.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за другим (магістерським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

В процесі розробки випускної кваліфікаційної роботи за другим (магістерським) 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

Навіть якщо ви бездоганно дотримувалися всіх найкращих практик гнучкої розробки, у гнучкій розробці програмного забезпечення все одно існують ризики. Гнучка розробка програмного забезпечення довела, що вона пропонує більше переваг, ніж недоліків, тому важливо розуміти потенційні ризики безпеки щоб ви могли пом'якшити їх завчасно.

Відсутність документації процесу

Agile надає пріоритет робочим продуктам над документацією. Деякі Agile-команди доводять це до крайності та прагнуть до нульової документації, але більшість типових, добре функціонуючих Agile-команд зосереджуються на мінімізації документації та забезпеченні того, щоб документування додавало цінності.

Однак така практика створює певні ризики в розробці програмного забезпечення. Agile надає пріоритет гнучкості (що добре), але це може призвести до швидкої послідовності швидких змін, що полегшує втрату сліду за документацією процесу.

Коли ви втрачаєте контроль над документацією процесу, ви не можете гарантувати дотримання всіх протоколів безпеки, оскільки довідкової документації може не існувати, або її зберігання в узгодженому вигляді не було пріоритетом. Для забезпечення безпечної розробки додатків ідеально мати певну довідкову документацію, незалежну від продукту. У цьому випадку підвищена гнучкість, пов'язана з меншою кількістю документації, може збільшити вашу вразливість до порушень безпеки.

Щоб зменшити проблеми безпеки, пов'язані з браком документації, зосередьтеся на більш тісному впровадженні безпеки в процес розробки у формі

«чемпіонів безпеки» в команді розробників або шляхом автоматизації деяких форм документації для створення історичного запису, до якого можна буде звертатися пізніше. Однією з головних проблем, з якими сьогодні стикаються керівники інформаційних систем (CISO) та їхні команди безпеки, є розуміння процесів розробки додатків, які їм доручено захистити, оскільки в організації немає документації щодо того, які системи використовуються розробниками. Зараз існують сучасні інструменти безпеки, такі як SaaS-платформа Legit Security, яка може автоматично виявляти та інвентаризувати активи SDLC – автоматично створюючи документацію, до якої команди безпеки можуть отримати доступ за потреби.

Цикл швидкого вивільнення

Гнучка методологія допомагає створити середовище для швидких релізів, але оскільки великий пріоритет надається забезпеченню цінності продукту протягом спринту, безпека часто відходить на другий план порівняно з розробкою основної функціональності продукту на ранніх етапах життєвого циклу розробки програмного забезпечення (SDLC). На жаль, це виявилося антипаттерном. Знижуючи пріоритет безпеки на ранніх етапах розробки програмного забезпечення, ви створюєте борг безпеки, який буде вирішено пізніше, коли для усунення цієї помилки безпеки буде набагато більше ресурсомістких ресурсів. Крім того, бізнес взяв на себе більшу відповідальність за ризик протягом цього часу, якщо помилка безпеки стане використаною вразливістю. Прагніть інтегрувати безпеку в гнучкі методи розробки.

Замість того, щоб просто пріоритезувати випуск коду, зосередьтеся на забезпеченні дотримання протоколів тестування вразливостей безпеки. Це може допомогти впровадити підхід «безпека понад усе» до гнучкої розробки та уникнути потенційних ризиків для безпеки. У дусі «гнучої безпеки» докладайте всіх зусиль для автоматизації перевірок безпеки та програмної оцінки дотримання протоколів безпеки.

Поява нових загроз

Оскільки гнучка розробка зосереджена на постійному вдосконаленні та невеликих обсягах роботи, релізи можуть створювати нові ризики безпеці, яких можна було б уникнути, якби ми продумали це наперед. Крім того, гнучкі методології іноді використовуються як привід для уникнення ретельного планування та підготовки. Команди, які недалекоглядно зосереджені лише на розробці наступного нового релізу, можуть не враховувати, як може змінитися ландшафт загроз з новим релізом (тобто розробники не є експертами з безпеки). В результаті такого сценарію зловмисники можуть легше використовувати будь-які приховані вразливості в гнучкому середовищі розробки.

Одним із поширених прикладів цього в сучасній індустрії програмного забезпечення є використання жорстко закодованих секретів у вихідному коді. Ця практика не викликала великого занепокоєння десять років тому, коли концепція хмарного програмного забезпечення, що надається як послуга, перебувала на ранніх стадіях впровадження на ринку. Однак протягом останнього десятиліття більшість гнучких розробників програмного забезпечення не зробили жодного кроку назад, щоб оцінити вплив SaaS та жорстко закодованих секретів. Це створило новий вектор атаки, де зловмисники прагнуть отримати доступ до систем управління вихідним кодом та вихідного коду, щоб отримати подальший доступ до інших систем (наприклад, серверів збірки), крадучи жорстко закодовані секрети. Короткозоро зосереджуючись на кожному спринті. Програмна галузь в цілому пропустила кричущу проблему безпеки, яка тепер стала головною проблемою.

Коли виникають нові загрози, вирішуйте їх двома способами:

– Аудит вашої існуючої поверхні атаки, щоб оцінити поточний ризик і визначити пріоритети виправлень безпеки.

– Вбудовуйте методи для ітеративного аудиту поверхні атаки, що постійно та автоматично відповідає вашому існуючому процесу розробки, щоб запобігти подальшому ризику.

Щоб впоратися з появою нових загроз за допомогою цього двостороннього підходу, організаціям необхідно впровадити стратегію внутрішніх команд для покращення безпеки та безперервного навчання.

Недостатня обізнаність та навчання з питань безпеки

Загальновідомо, що безпека часто відходить на другий план як з боку внутрішніх команд, так і з боку клієнтів та сторонніх ресурсів на користь більш безпосередньої діяльності, що «генерує дохід». В епоху атаки на ланцюги поставок програмного забезпечення. Однак таке мислення небезпечно застаріло. Простіше кажучи, атака SolarWinds чітко показала, що існує новий тип загрози, яка безпосередньо призведе до втрати клієнтів, оскільки зловмисники тепер прагнуть атакувати ваших клієнтів *через вас.* Ось чому важливо підкреслювати та сприяти підвищенню обізнаності та навчанню з питань безпеки для всіх сторін.

Замість того, щоб зосереджуватися лише на швидкості та термінах виконання, виділіть спеціальний час для навчання з питань безпеки та протоколів безпеки. Прагніть досягти гнучку безпеку. Використовуйте сучасні інструменти безпеки ланцюга постачання програмного забезпечення, які надають агрегований «оцінку безпеки» за лінійкою продуктів та командою розробників. Таким чином, ви можете застосовувати більше навчання з безпеки до команд розробників, які цього найбільше потребують, не обтяжуючи при цьому вже безпечні команди розробників. Завдяки гейміфікації практик безпеки за допомогою оцінок безпеки за лінійкою продуктів, організації можуть перейти до фази посилення гнучкої безпеки.

Необмежений доступ до репозиторіїв вихідного коду та конвеєрів CI/CD

Нерідко трапляється просто надавати загальні дозволи та авторизації великим групам команди розробників, особливо в невеликих організаціях з менш розвиненими практиками безпеки. Більша кількість користувачів з доступом до репозиторіїв коду, систем керування вихідним кодом (SCM), серверів збірки, реєстрів артефактів та конвеєрів CI/CD означає більший ризик для вашого SDLC.

Кожен користувач із надмірними привілеями являє собою ще одне слабке місце у вашому захисті, яке стає мішенню для зловмисників. Крім того, команди розробників програмного забезпечення часто передають певну частину розробки зовнішнім, контрактним командам розробників на проектній основі. Часто ці підрядники зберігають свої дозволи ще довго після завершення проекту. Ризики для гнучкої розробки програмного забезпечення особливо критичні, коли привілеї адміністративного рівня надаються занадто вільно.

Підвищте свою гнучку безпеку, дотримуючись найкращих практик гнучкої розробки. Перегляньте дозволи та протоколи авторизації для вашої команди розробників та систем, які вони використовують для швидкої розробки програмного забезпечення. Використовуйте принцип найменших привілеїв, щоб кожен користувач мав лише ті дозволи, які необхідні для виконання його роботи.

Якщо кількість розробників, які мають доступ до ваших конвеєрів розробки, коливається від 0 до 50 осіб, ручний підхід може бути можливим і бажаним; однак для команд, що налічують понад 50 розробників, найкраще використовувати такий інструмент, як платформа Legit Security, щоб допомогти вам визначити поточні ризики доступу, присутні у ваших конвеєрах розробки, перш ніж зловмисник скомпрометує застарілі облікові записи, які, як ви помилково вважали, більше не мають доступу до ваших конвеєрів.

Невміння керувати конфіденційними даними

Багато команд розробників розуміють основні принципи безпеки, але більшість із них не дуже добре обізнані з тим, як інтегрувати практики безпеки навколо управління даними таким чином, щоб це відповідало найкращим практикам гнучкої розробки.

Повний перелік ризиків безпеки, пов’язаних з управлінням конфіденційними даними, є довгим і виходить за рамки цього блогу, але деякі з найбільших викликів у DataOps – це (1) впровадження проактивного управління даними, (2) маркування конфіденційних даних та (3) навчання команд прийнятним практикам роботи з даними.

Ось кілька порад щодо кращого управління конфіденційними даними:

- Централізувати управління ідентифікацією.
- Визначення прав доступу на основі ролей.
- Маскування конфіденційних даних.
- Відмовтеся від ручного захисту даних та *автоматизуйте його*, коли це можливо.
- Постійно скануйте код на наявність конфіденційних даних.

Оскільки команди безпеки та розробки стають більш узгодженими, вигідно мати «чемпіона безпеки» або когось у команді розробників, хто виступає за покращення загальної ситуації та наполягає на цьому. поза безпеки та управління конфіденційними даними. Врахування безпеки як частини розробки допомагає всім командам долати перешкоди у сфері безпеки.

Погане управління сторонніми розробниками та відкритим кодом

Сторонні таз відкритим кодом. Код може відігравати важливу роль для більшості команд розробників, звільняючи багато ресурсів від необхідності створювати кожен аспект коду власними силами. Однак це не означає, що сторонній код та код з відкритим вихідним кодом не потребують підтримки. Весь цей зовнішній код поєднується з власним кодом, створюючи частину вашого ланцюжка постачання програмного забезпечення, яка є дуже вразливою до атак.

Прості способи управління програмним забезпеченням з відкритим кодом та сторонніми розробниками у вашому ланцюжку поставок включають:

- Встановлення політики щодо використання програмного забезпечення з відкритим вихідним кодом.
- Створення ради управління відкритим кодом.
- Створення взаємопов'язаних відносин з вашими постачальниками, щоб у разі виникнення проблем вони швидко реагували та вживали необхідних заходів для їх усунення..
- Регулярно запускайте сканування за допомогою аналізу складу програмного забезпечення (SCA).

– Інвентаризуйте свій SDLC, щоб зрозуміти, які конвеєри є найважливішими для бізнесу.

– Скануйте свої конвеєри розробки на наявність SCA-сканувань.

Важливо зазначити, що ваш ланцюжок постачання програмного забезпечення – це не лише зовнішній код, який тут використовується. Ваш ланцюжок постачання програмного забезпечення також складається з інструментів розробника та конвеєрів у вашому SDLC, а також будь-якого додаткового коду, бібліотек та ресурсів, що використовуються під час розгортання вашого програмного забезпечення або інструменту, що передаються у ваше робоче середовище.

Невикористання стороннього коду на свою користь

Багато команд надають великого значення розробці внутрішніх рішень безпеки, але розробка власного коду та рішень має кілька недоліків. Через це іноді нове, неперевірене, незахищене внутрішнє рішення використовується *за звичкою,* замість того, щоб просто оцінити сторонні рішення, які вирішують проблему більш нестандартним та менш ресурсоємним способом порівняно з саморобним підходом.

Один гібридний підхід полягає в інвестуванні в low-code та no-code рішення для безпеки та автоматизації, щоб інтегрувати безпеку в гнучкі методи розробки. Такі інструменти допомагають усунути вузькі місця, пов'язані з обмеженнями потужностей команди розробників. Власники гнучких продуктів не завжди по-справжньому розуміють наслідки для безпеки своїх основних функцій, тому важко планувати заздалегідь під час розробки вимог. І не всі команди здатні формулювати вимоги, що стосуються проблем безпеки, не диктуючи потім занадто багато елементів рішення, що може призвести до значного розширення обсягу робіт під час спроби створення внутрішніх рішень.

Коротше кажучи, не бійтеся використовувати надійний та безпечний сторонній код і рішення на свою користь, оскільки ці рішення добре перевірені та значно скорочують час розгортання.

Нездатність визначити пріоритети служби безпеки

Підхід, що ставить безпеку на перше місце, може здаватися суперечливим темпам типової гнучкої розробки для багатьох гнучких команд. Тестування та процедури безпеки часто розглядаються як перешкоди та лежачі поліцейські на шляху до швидшої доставки готового продукту членами гнучкої команди, зосередженими на дотриманні останнього дедлайну спринту.

Один зі способів підвищити безпеку в гнучкій розробці – це спричинити культурні зміни, які зроблять безпечну розробку невід'ємним пріоритетом. Приділіть час переосмисленню процедур безпеки, щоб члени команди розуміли, чому певна практика безпеки є важливою. Інтегруйте команди безпеки та розробки, щоб безпека могла бути інтегрована на ранніх етапах процесу експертом у цій галузі. Обговоріть та встановіть розумну частоту та процедури тестування, щоб допомогти перейти до культури «гнучої безпеки». Існує тісний зв'язок між додаванням занадто великої кількості необґрунтованих перевірок безпеки в процес та часом розробки, який витрачається на винахід способів обійти складні процеси. Наприклад, може бути безпечніше вимагати понад 2 рецензентів коду під час просування коду, але це створює більш ніж удвічі більше труднощів, ніж вимога лише одного рецензента коду. Відділи безпеки повинні враховувати вплив на досвід розробника під час запровадження політики, якої необхідно дотримуватися.

Вибір самостійної безпеки

Нерідко багато команд намагаються вирішувати всі свої потреби в безпеці самостійно. Існує поширена помилкова думка, що повністю внутрішня команда може заощадити вам час і гроші (особливо враховуючи те, що компанії відчайдушно потребують експертів з кібербезпеки.) Часто внутрішнім командам бракує всебічної експертизи, необхідної для забезпечення безпеки всього SDLC – саме тут і знадобляться експерти.

Зовнішня експертиза буває двох форм:

- Постачальники послуг кібербезпеки.

– Постачальники продуктів кібербезпеки.

Використання зовнішньої експертизи з безпеки для управління вашою безпекою дає деякі помітні переваги:

– Звільняє внутрішні ресурси, щоб зосередитися безпосередньо на завданнях, які можуть виконати тільки вони.

– Заповнює прогалини в спеціалізації, для забезпечення яких було б надто важко найняти внутрішній ресурс.

Тепер, коли розробка дедалі більше кодифікується, у індустрії програмного забезпечення спостерігається поява принципу «політика як код». SaaS-рішення, такі як Легітимна платформа безпеки. Зараз існують стандарти, які кодифікували сотні найкращих практик у вузькоспеціалізованих сферах безпеки, таких як безпека ланцюга постачання програмного забезпечення. Замість того, щоб вимагати внутрішнього експерта з предметної області, ви можете використовувати знання, кодифіковані в продукті. Крім того, SaaS краще підходить для контролю того, чи порушують команди розробників необхідні політики, порівняно з внутрішнім ресурсом безпеки. Зрештою, це допомагає досягти більшої безпеки-орієнтованої розробки.

3.2 Розробка структурної схеми

Гнучка розробка допомагає компаніям-розробникам програмного забезпечення впроваджувати легкі, ітеративні цикли розробки. Методологія розробки робить акцент на невеликих, керованих командах з міжфункціональною співпрацею для частих оновлень та випусків.

Хоча Agile залежить від масштабних культурних змін у розробці та тестуванні коду, ця модель також вимагає нового підходу до впровадження безпеки в Agile.

У цьому розділі обговорюються відповідні ризики, що виникають внаслідок неправильного впровадження методів безпеки, а також заглиблюються

в найкращі практики забезпечення циклу Agile-розробки.

Ризики неврахування безпеки у вашій гнучкій розробці

Гнучкі процеси зазвичай відкривають виробниче та розробницьке середовища для розподілених команд, що призводить до витоку важливої інформації через поступове збільшення кількості авторизацій. Хоча гнучкі методології розвивалися, щоб враховувати інновації та терміни виконання, безпека часто відходить на другий план, оскільки оцінки безпеки уповільнюють розробку.

Практики безпеки особливо складно впроваджувати однаково протягом усього життєвого циклу розробки програмного забезпечення. Цикл збірки є безперервним, і тести безпеки необхідно проводити поза робочим процесом розробки. Це створює різні ризики для безпеки організації, зокрема:

Недостатня політика, що регулює використання компонентів з відкритим кодом

Основний принцип Agile полягає в тому, щоб не винаходити велосипед. Крім того, командам рекомендується використовувати рішення з відкритим кодом та комерційні власні рішення для скорочення життєвого циклу розробки. Однак, хоча такі інтеграції з відкритим кодом пропонують різні переваги, вони також створюють значні ризики безпеки, які можуть вплинути на розробку.

Типовий сценарій – це коли розробники не знають про залежності модулів та кількість бібліотек коду, які вони імпортують. Як наслідок, відстеження змін у відкритому коді є складним завданням, що ускладнює забезпечення надійної та безпечної роботи програмного забезпечення.

Залежність від інтеграцій з відкритим кодом також ускладнює отримання інформації про конкретні вразливості та виправлення, виявлені спільнотою учасників. За відсутності чітко визначених політик, що регулюють інтеграції з відкритим кодом, вибір інструментів та управління технологічним стеком, виявляють проблеми для зменшення загроз.

Необмежений доступ до процесів розробки та репозиторіїв вихідного коду

Agile сприяє ефективній співпраці між міжфункціональними, розподіленими командами, які дотримуються ітеративного підходу до розробки. Щоб забезпечити роботу команд з єдиним джерелом достовірної інформації, організації покладаються на публічні та приватні репозиторії вихідного коду для підтримки, обміну та версіонування коду.

Хоча репозиторії забезпечують безперебійний контроль версій, команди часто не можуть інтерпретувати версію вихідного коду, що використовується, що ускладнює виявлення конкретних вразливостей та застосування заходів щодо їх усунення.

Крім того, коли приватне створення коду потрапляє до репозиторіїв вихідного коду, він розкриває вразливі точки входу. Зловмисники націлюються на такі відкриті репозиторії, щоб виявити приховані функції, а потім поєднують їх з розвідувальними тактиками для організації атак.

Небезпечне управління конфіденційними даними

Сучасний процес гнучкої розробки інтегрує численні джерела для аналітики та операцій з даними в масштабах всієї організації. Складна інтеграція вимагає систематичної та точної класифікації даних для контролю рівня доступу до конфіденційних даних організації.

Крім того, фрагменти тестових даних, створені командою тестувальників, часто є значною часткою реальних даних. Хоча такі дані використовуються внутрішньо для цілей тестування, відсутність належної санітарної обробки даних сприяє зловживанню вразливостями з боку векторів атак.

Більшість Agile-організацій не мають політик захисту своїх даних від внутрішніх зловмисників. Через відсутність суворих політик класифікації даних, члени команди зазвичай не знають про комплексні вимоги безпеки, чутливість даних та свою відповідальність за їх захист. Рівень співпраці, необхідний для безперебійної роботи, також ускладнює для команди безпеки впровадження

заходів безпеки.

Методології безпеки в Agile-розробці

Гнучка розробка програмного забезпечення запроваджує культурні зміни в масштабах усієї організації, зосереджуючись на модульних, більш керованих, міжфункціональних командах. Хоча ці зміни забезпечують дуже гнучку розробку, яка реагує на мінливі вимоги, організаціям потрібне впровадження спеціальних методів для захисту коду та середовищ розгортання.

Нижче наведено деякі з найкращих практик для захисту застосунків у гнучкому середовищі:

Захист гнучких конвеєрів доставки

Методологія Agile робить акцент на конвеєрах безперервної інтеграції (CI) та безперервної доставки (CD), щоб забезпечити швидше розгортання. Щоб мінімізувати ризики безпеки, команди Agile повинні розглянути інтеграцію автоматизованих механізмів тестування на кожному етапі SDLC. Крім того, всі коміти, зроблені командою розробників, повинні проходити через експертів з безпеки, щоб гарантувати безпеку розробленого застосунку.

Окрім ручних перевірок коду, також повинні бути статичні аналізатори коду, інтегровані з конвеєром неперервної інтеграції (CI), та автоматизовані модульні тести для запуску запланованих перевірок. Також рекомендується проводити активні аудити безпеки та автоматичні перевірки безпеки під час процесу CI/CD, щоб допомогти виявити вразливості безпеки, перш ніж код перейде у продакшн.

Вартість виявлення вразливості безпеки на ранній стадії значно менша порівняно з пошуком її у виробництві, що може призвести до серйозних наслідків, включаючи крадіжку конфіденційних даних, неякісну продуктивність програм та втрату репутації.

Забезпечити безперервну участь усієї організації

Порівняно з традиційними практиками, Agile-процес представляє собою зсув у практиках розробки, який повинен супроводжуватися змінами в

організаційній культурі в бік практик безпеки. Рекомендується, щоб фахівці з безпеки співпрацювали з розробниками для формування затвердженого керівництва щодо стандартів кодування, шаблонів проектування та рішень щодо проектування.

Організаціям також слід проводити ретельне моделювання загроз, одночасно розробляючи безперервне навчання з безпеки, яке зосереджується на зміні поведінки, а не лише на обізнаності. Фахівці з безпеки також повинні документувати критерії безпечної розробки на спільних платформах, щоб інші організації могли посилатися на них у своїх процесах.

Фірми також можуть впроваджувати гнучкі методи безпеки, впроваджуючи політики безпеки в нещодавно розроблені мікросервіси, додатки, інтеграції та API.

Впроваджуйте пасивні огляди

Методологія гнучкої розробки характеризується коротшими ітераціями, швидкими збірками та частими випусками нових функцій. Тому необхідно переглянути всі зміни в додатку, які потребують ітерацій, та як вони впливають на безпеку системи, перш ніж їх впроваджувати.

Пасивний огляд коду допомагає фахівцям з контролю якості розуміти додатки, не скануючи кожен рядок вихідного коду. Натомість, фахівці з контролю якості можуть переглядати код, а системи огляду коду відстежують їхні переміщення в кодовій базі та пропонують своєчасну, відповідну додаткову інформацію для вихідного коду. Деякі інструменти безпеки пасивного огляду також генерують схеми моделей, які візуалізують логіку, реалізовану кодом, спрощуючи усунення несправностей.

Забезпечення проактивного контролю для команд Agile-розробників

Організація повинна підтримувати розробників у впровадженні заходів контролю, що сприяють безпечній практиці розробки коду. OWASP перераховує різні методи безпеки, які забезпечують побудову всіх рівнів програми, з акцентом на безпеці. Деякі з цих методів включають:

– Визначення вимог безпеки – Гнучкі методи повинні бути побудовані на галузевих стандартах, минулих вразливостях та чинному законодавстві, щоб гарантувати, що продукт відповідає всім властивостям безпеки.

– Використання бібліотек та фреймворків безпеки – ці бібліотеки та фреймворки мають вбудовані заходи безпеки для захисту коду від недоліків проектування та реалізації, які призводять до недоліків безпеки.

– Безпека сховищ даних – команди повинні забезпечити безпеку як NoSQL, так і реляційних баз даних за допомогою безпечних запитів, автентифікації, налаштування та зв'язку.

– Кодування та екранування даних – Кодування та екранування даних використовуються для запобігання вразливостям, пов'язаним з ін'єкціями. Ескейпінг передбачає використання спеціальних символів, щоб уникнути неправильної інтерпретації рядків, тоді як кодування стосується перетворення символів на однакові значення з різними форматами, що робить їх недоступними для зловмисних інтерпретаторів.

Інші проактивні засоби контролю для комплексної Agile-безпеки включають перевірку вхідних даних, керування ідентифікацією та доступом (IAM), обробку помилок та винятків, а також ведення журналу та моніторингу безпеки,

Хоча Agile та традиційна методи безпеки спираються на впровадження подібних політик безпеки, реалізація функцій безпеки значно відрізняється. У Agile-методах швидкі цикли випуску передбачають часті тести безпеки, що означає, що детальний аналіз коду може зрештою уповільнити розробку та реалізацію.

Щоб вирішити цю проблему, індикатори ризиків безпеки слід встановити відповідно до швидкості виконання, що дозволить Agile-командам проводити перевірки безпеки, одночасно справляючись з частими змінами. Agile-практики безпеки також слід класифікувати на ті, що виконуються на початку розробки, та ті, що впроваджуються під час кожного спринту, залежно від того, чи потрібно їх

виконувати одноразово чи постійно.

Найпоширенішою помилкою Agile-команд, які намагаються захистити конфіденційну інформацію, є неправильна класифікація даних. Більшість команд не вказують дані, які потребують спеціального захисту, що ускладнює впровадження ефективної політики класифікації даних.

Крім того, команди розробників програмного забезпечення не можуть оптимально шифрувати конфіденційні дані під час передачі або зберігання, що робить їх легкодоступними для зловмисників. Ще один момент для занепокоєння виникає, коли команди неправильно використовують платформи хмарного зберігання даних, зберігаючи облікові дані та дані конфігурації без правильного тлумачення політик безпеки постачальника. Це ускладнює впровадження комплексної стратегії з точки зору безпеки.

Розроблене програмне забезпечення дозволяє забезпечити захист переданої по мережі інформації, строгу взаємну автентифікацію користувачів і серверів, гнучке розмежування доступу. Для реалізації цих функцій у системі використовуються SSL/TLS протоколи й X.509 цифрові сертифікати, тобто універсальні, що стали стандартом де-факто, механізми, підтримувані практично всіма розповсюдженими Веб-агентами.

За допомогою розробленого програмного забезпечення легко забезпечуються вимоги по інформаційній безпеці, запропоновані різними Інтернет додатками, такими як:

– сервера платіжних систем;

– інтернет-магазини;

– багатопрофільні корпоративні Веб-сервера, що містять інформацію з різним рівнем конфіденційності;

– B2B системи;

– системи захищеного документообігу;

– системи обміну електронною поштою;

– й багато які інші.

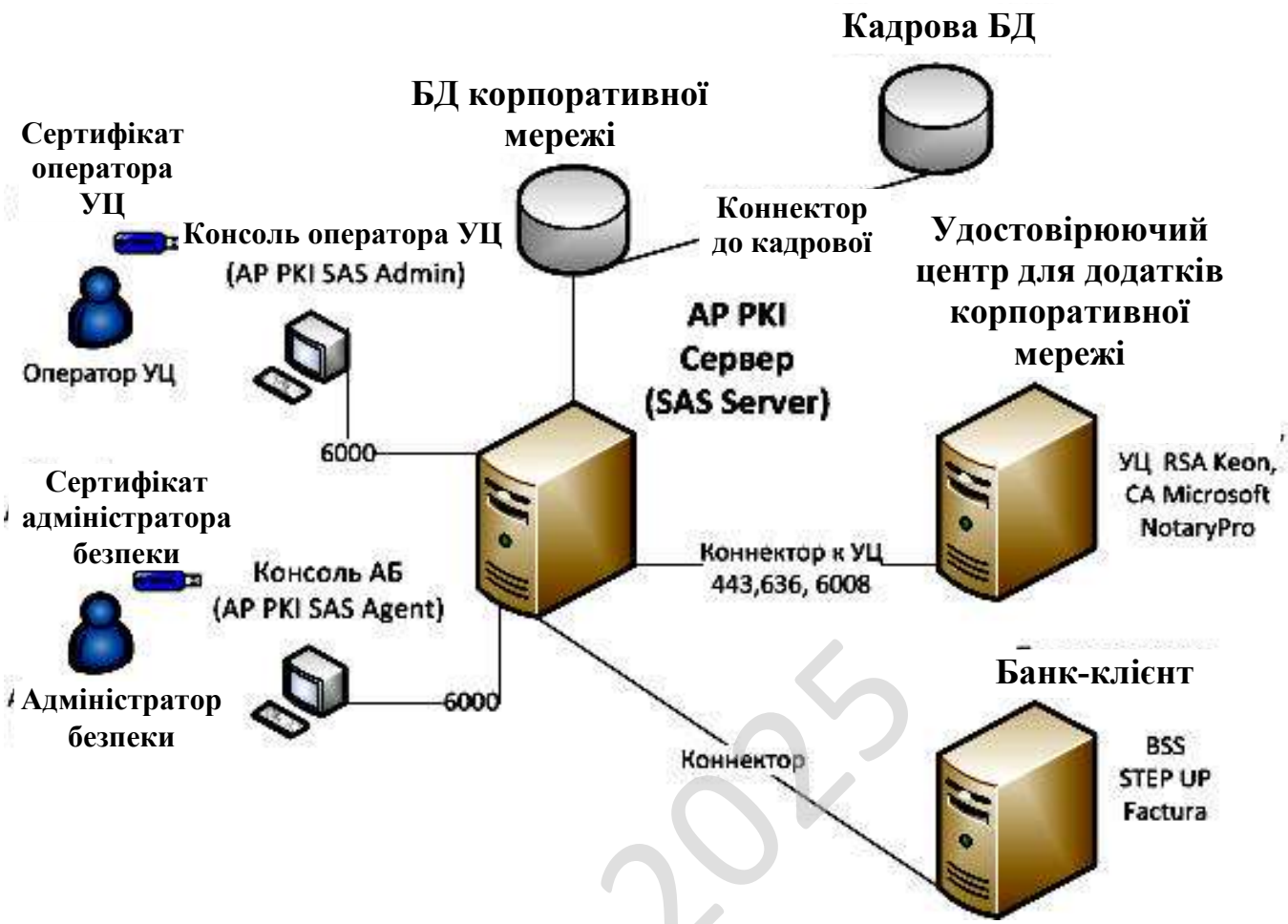

Рисунок 3.1 – Структурна схема системи

На рисунку 3.1 представлена структурна схема розробленої системи. Для поняття роботи системи введені наступні позначення:

– ЕЦП – електроний цифровий підпис;

– УЦ – удостовірюючий центр.

– ЦС – цифровий сертифікат;

– PKI – інфраструктура відкритих ключів;

– DVCS – Data Validation and Certification Server Protocols – протокол підтвердження даних та сертифікації серверу;

– OCSP – Online Certificate Status Protocol – онлайн протокол статусу сертфікату;

– TSP – Time-Stamp Protocol – протокол часових міток;

– TLS – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет;

– RFC – документ, у якому описується той або інший стандарт.

3.3 Розробка функціональної схеми

Agile

Гнучка методологія розробки (англ. Agile software development, agile-методи) – серія підходів до розробки програмного забезпечення, орієнтованих на використання ітеративної розробки, динамічне формування вимог і забезпечення їхньої реалізації в результаті постійної взаємодії усередині робочих груп, що самоорганізуються, що складаються з фахівців різного профілю [1]. Існує кілька методик, що відносяться до класу гнучких методологій розробки, зокрема екстремальне програмування, DSDM, Scrum, FDD.

Застосовується як ефективна практика організації праці невеликих груп (які роблять однорідну творчу роботу) в об'єднанні з керуванням ними комбінованим (ліберальним і демократичним) методом.

Більшість гнучких методологій націлені на мінімізацію ризиків шляхом відомості розробки до серії коротких циклів, називаних ітераціями, які звичайно тривають дві-три тижні. Кожна ітерація сама по собі виглядає як програмний проект у мініатюрі й включає всі завдання, необхідні для видачі міні-приросту по функціональності: планування, аналіз вимог, проектування, програмування, тестування й документування. Хоча окрема ітерація, як правило, недостатня для випуску нової версії продукту, мається на увазі, що гнучкий програмний проект готовий до випуску наприкінці кожної ітерації. По закінченні кожної ітерації команда виконує переоцінку пріоритетів розробки.

Agile-методи наголошують на безпосереднє спілкування віч-на-віч. Більшість agile-команд розташовані в одному офісі, іноді називаному англ. bullpen. Як мінімум, вона включає й «замовників» (англ. product owner –

замовник або його повноважний представник, що визначає вимоги до продукту; цю роль може виконувати менеджер проекту, бізнес-аналітик або клієнт). Офіс може також включати тестувальників, дизайнерів інтерфейсу, технічних письменників і менеджерів.

Основною метрикою agile-методів є робочий продукт. Віддаючи перевагу безпосередньому спілкуванню, agile-методи зменшують обсяг письмової документації в порівнянні з іншими методами. Це привело до критики цих методів як недисциплінованих.

У лютому 2001 у штаті Юта США був випущений «Маніфест гнучкої методології розробки програмного забезпечення» (англ. Agile Manifesto). Він був альтернативою керованим документацією «великоваговим» практикам розробки програмного забезпечення, таким як «метод водоспаду», що був золотим стандартом розробки в той час. Даний маніфест був схвалений і підписаний представниками методологій: екстремального програмування, Crystal Clear [en], DSDM, Feature driven development, Scrum, Adaptive software development [en], Pragmatic Programming. Гнучка методологія розробки використовувалася багатьма компаніями й до прийняття маніфесту, однак входження Agile-Розробки в маси відбулося саме після цієї події.

Agile – сімейство процесів розробки, а не єдиний підхід у розробці програмного забезпечення, і визначається Agile Manifesto [2]. Agile не включає практик, а визначає цінності й принципи, якими керуються команди.

Agile Manifesto розроблений і прийнятий 11-13 лютого 2001 року на лижному курорті The Lodge at Snowbird у горах Юти. Agile Manifesto містить 4 основні ідеї й 12 принципів. Примітно, що Agile Manifesto не містить практичних рад.

Основні ідеї:

– люди й взаємодія важливіше процесів і інструментів;
– працюючий продукт важливіше вичерпної документації;
– співробітництво із замовником важливіше узгодження умов контракту;

– готовність до змін важливіше проходження первісному плану.

Принципи, які роз'ясняє Agile Manifesto [3]:

– задоволення клієнта за рахунок ранньої й безперебійної поставки коштовного програмного забезпечення;

– вітання змін вимог навіть наприкінці розробки (це може підвищити конкурентоспроможність отриманого продукту);

– часта поставка робочого програмного забезпечення (щомісяця або тиждень або ще частіше);

– тісне, щоденне спілкування замовника з розроблювачами протягом усього проекту;

– проектом займаються мотивовані особистості, які забезпечені потрібними умовами роботи, підтримкою й довірою;

– метод передачі, що рекомендується, інформації – особиста розмова (віч-на-віч);

– працююче програмне забезпечення – кращий вимірник прогресу;

– спонсори, розроблювачі й користувачі повинні мати можливість підтримувати постійний темп на невизначений строк;

– постійна увага поліпшенню технічної майстерності й зручному дизайну;

– простота – мистецтво не робити зайвої роботи;

– кращі технічні вимоги, дизайн і архітектура виходять у самоорганізованої команди;

– постійна адаптація до обставин, що змінюються. Команда повинна систематично аналізувати можливі способи поліпшення ефективності й відповідно коректувати стиль своєї роботи. [4]

Один з повторюваних пунктів критики: при agile-підході часто зневажають створенням плану («дорожньої карти») розвитку продукту, так само як і керуванням вимогами, у процесі якого й формується така «карта». Гнучкий підхід до керування вимогами не має на увазі далеко, що йдуть планів (по суті, керування вимогами просто не існує в даній методології), а має на увазі

можливість замовника раптом і зненацька наприкінці кожної ітерації виставляти нові вимоги, що часто суперечать архітектурі вже створеного й поставляти^ся продукта, що. Таке іноді приводить до катастрофічного «авралам» з масовим рефакторингом і переробками практично на кожній черговій ітерації.

Крім того, уважається, що робота в agile мотивує розроблювачів вирішувати всі завдання, що надійшли, найпростішим і найшвидшим можливим способом, при цьому найчастіше не обертаючи уваги на правильність коду з погляду вимог нижчележачої платформи (підхід – «працює, і добре», при цьому не враховується, що може перестати працювати при найменшій зміні або ж дати важкі до відтворення дефекти після реального впровадження в клієнта). Це приводить до зниження якості продукту й нагромадженню дефектів.

Існують методології, які дотримуються цінностей і принципів заявлених в Agile Manifesto, деякі з них:

– Agile Modeling (англ.) – набір понять, принципів і прийомів (практик), що дозволяють швидко й просто виконувати моделювання й документування в проектах розробки програмного забезпечення. Не містить у собі детальну інструкцію із проектування, не містить описів, як будувати діаграми на UML. Основна мета: ефективне моделювання й документування; але не охоплює програмування й тестування, не включає питання керування проектом, розгортання й супроводи системи. Однак містить у собі перевірку моделі кодом [5].

– Agile Unified Process (англ.) (AUP) спрощена версія IBM Rational Unified Process (RUP), розроблена Скоттом Амблером, що описує просте й зрозуміле наближення (модель) для створення програмного забезпечення для бізнес-додатків.

– Agile Data Method (англ.) – група ітеративних методів розробки програмного забезпечення, у яких вимоги й рішення досягаються в рамках співробітництва різних крос-функціональних команд.

– DSDM заснований на концепції швидкої розробки додатків (Rapid Application Development, RAD). Являє собою ітеративний і інкрементний підхід, що надає особливого значення тривалій участі в процесі користувача/споживача.

– Essential Unified Process (англ.) (EssUP).

– Екстремальне програмування (англ. Extreme programming, XP).

– Feature driven development (FDD) – функціонально-орієнтирована розробка. Використовуване в FDD поняття функції або властивості (англ. feature) системи досить близько до поняття прецеденту використання, використовуваній в RUP, істотна відмінність – це додаткове обмеження: «кожна функція повинна допускати реалізацію не більш, ніж за два тижні». Тобто якщо сценарій використання досить малий, його можна вважати функцією. Якщо ж великий, то його треба розбити на трохи щодо незалежних функцій.

– Getting Real – ітеративний підхід без функціональних специфікацій, що використовується для веб-додатків. У даному методі спершу розробляється інтерфейс програми, а потім її функціональна частина.

– OpenUP – це ітеративно-інкрементальний метод розробки програмного забезпечення. Позиціонується як легкий і гнучкий варіант RUP. OpenUP ділить життєвий цикл проекту на чотири фази: початкова фаза, фази уточнення, конструювання й передачі. Життєвий цикл проекту забезпечує надання зацікавленим особам і членам колективу крапок ознайомлення й прийняття рішень протягом усього проекту. Це дозволяє ефективно контролювати ситуацію й вчасно ухвалювати рішення щодо прийнятності результатів. План проекту визначає життєвий цикл, а кінцевим результатом є остаточний додаток.

– Scrum установлює правила керування процесом розробки й дозволяє використовувати вже існуючі практики кодування, коректуючи вимоги або вносячи тактичні зміни. Використання цієї методології дає можливість виявляти й усувати відхилення від бажаного результату на більше ранніх етапах розробки програмного продукту.

– Ощадлива розробка програмного забезпечення (англ. lean software development) використовує підходи з концепції ощадливого виробництва.

Розроблене програмне забезпечення представляє із себе набір компонентів призначених для забезпечення політики безпеки як у вже існуючих, так і в створюваних мережних корпоративних додатках, які розробляються за допомогою методології Agile.

Сертифікат X.509 v3

Сертифікат X.509 v3 визначається в такий спосіб. Для обчислення підпису дані, які повинні бути підписані, представляються з використанням ASN.1 однозначних правил подання (DER).

```
Certificate  ::=  SEQUENCE  {
    tbsCertificate     TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue     BIT STRING
}
TBSCertificate  ::= SEQUENCE  {
    version [0] EXPLICIT Version DEFAULT v1,
    serialNumber CertificateSerialNumber,
    signature AlgorithmIdentifier,
    issuer Name,
    validity Validity,
    subject Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID [1] IMPLICIT
        UniqueIdentifier OPTIONAL,
    ---і якщо є присутнім, версія повинна
    ---і бути v2 або v3
    subjectUniqueID [2] IMPLICIT
        UniqueIdentifier OPTIONAL,
    ---і якщо є присутнім, версія повинна бути
    ---і v2 або v3
    extensions [3] EXPLICIT Extensions OPTIONAL
    ---і якщо є присутнім, версія повинна бути v3
}
Version ::= INTEGER { v1(0), v2(1), v3(2) }
CertificateSerialNumber ::= INTEGER
Validity ::= SEQUENCE {
    notBefore Time,
```

```
    notAfter  Time
}
Time ::= CHOICE {
    utcTime     UTCTime,
    generalTime GeneralizedTime
}
UniqueIdentifier ::= BIT STRING
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm        AlgorithmIdentifier,
    subjectPublicKey BIT STRING
}
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension
Extension ::= SEQUENCE  {
      extnID    OBJECT IDENTIFIER,
      critical  BOOLEAN DEFAULT FALSE,
      extnValue OCTET STRING
}
```

Поля сертифіката. Сертифікат є послідовність трьох обов'язкових полів: tbsCertificate, signatureAlgorithm і signatureValue.

– tbsCertificate. Поле містить імена суб'єкта й випускаючого, відкритий ключ, пов'язаний із суб'єктом, період дійсності й іншу пов'язану із цим сертифікатом інформацію. Поля докладно описані далі; tbsCertificate звичайно включають розширення, які теж будуть описані нижче.

– signatureAlgorithm. Поле signatureAlgorithm містить ідентифікатор криптографічного алгоритму, використовуваного УЦ для підписування даного сертифіката. Існують стандартні алгоритми, які повинні підтримуватися всіма реалізаціями, але конкретна реалізація може підтримувати й інші алгоритми.

Ідентифікатор алгоритму визначається наступної ASN.1-структурою:

```
AlgorithmIdentifier ::= SEQUENCE  {
  algorithm  OBJECT IDENTIFIER,
  parameters ANY DEFINED BY algorithm OPTIONAL
}
```

Ідентифікатор алгоритму використовується для визначення криптографічного алгоритму. Компонент OBJECT IDENTIFIER ідентифікує алгоритм (такий як DSA з SHA-1). Компоненти поля параметрів змінюються

відповідно до зазначеного алгоритму. Поле повинне містити той же самий ідентифікатор алгоритму, що й поле підпису в tbsCertificate.

– signatureValue. Поле signatureValue містить цифровий підпис, обчислений для поля tbsCertificate, записаному в DER-поданні ASN.1. Це означає, що поле tbsCertificate, представлене як ASN.1 DER, використовується як вхід у функцію підпису. Отримане значення підпису представлене як BIT STRING і включено в поле підпису. Деталі даного процесу можуть відрізнятися для кожного конкретного алгоритму підпису. Створенням даного підпису УЦ підтверджує дійсність інформації в поле tbsCertificate. Зокрема, УЦ підтверджує зв'язок між матеріалом відкритого ключа й суб'єктом сертифіката.

– TBSCertificate. Послідовність TBSCertificate містить інформацію, пов'язану із суб'єктом сертифіката й УЦ, що випустив сертифікат. Кожний TBSCertificate містить імена суб'єкта й випускаючого, відкритий ключ, пов'язаний із суб'єктом, період дійсності, номер версії й серійний номер сертифіката; деякі поля можуть (але це не обов'язково) містити унікальний ідентифікатор. Розглянемо синтаксис і семантику таких полів. TBSCertificate звичайно включає розширення. Розглянемо також найбільше часто використовувані в Internet розширення.

– Version. Дане поле описує версію подання сертифіката. Якщо використовуються розширення, то версія повинна бути 3 (значення – 2). Якщо розширення не зазначені, але UniqueIdentifier представлений, версія може бути 2 (значення – 1); але версія може бути й 3. Якщо представлені тільки базові поля, версія може бути 1 (значення в сертифікаті опущене як значення за замовчуванням); але версія може бути 2 або 3. Реалізації повинні бути готові приймати будь-яку версію сертифіката. Як мінімум конформні реалізації повинні розпізнавати версію 3 сертифікатів.

– Serial number. Серійний номер повинен бути позитивним цілим, призначуваним УЦ для кожного сертифіката. Він повинен бути унікальним для кожного сертифіката, випущеного даним УЦ. Таким чином, ім'я що выпустили й

серійний номер однозначно визначають сертифікат. Cas повинні забезпечувати, щоб серійні номери були ненегативними цілими. Уважається, що серійні номери можуть мати довжину до 20 октетів.

– Signature. Дане поле містить ідентифікатор алгоритму, використовуваного УЦ для підписування сертифіката.

– Дане поле повинне містити той же самий ідентифікатор алгоритму, що й поле signatureAlgorithm в Certificate. Зміст необов'язкового поля параметрів залежить від конкретного алгоритму.

– Issuer. Поле issuer ідентифікує того, хто підписав і випустив сертифікат. Поле issuer повинне містити непусте унікальне ім'я (DN). Ім'я визначається у відповідності з наступної ASN.1 структурою:

```
Name ::= CHOICE { RDNSequence }
RDNSequence ::= SEQUENCE OF
    RelativeDistinguishedName
RelativeDistinguishedName ::=
    SET OF AttributeTypeAndValue
AttributeTypeAndValue ::= SEQUENCE {
    type   AttributeType,
    value  AttributeValue
}
AttributeType ::= OBJECT IDENTIFIER
AttributeValue ::= ANY DEFINED BY
    AttributeType
```

Ім'я описує ієрархічне ім'я, що складається з атрибутів, таких, наприклад, як назва країни, і відповідних значень, таких як RU. Тип компонента AttributeValue визначається значенням AttributeType. Стандарт X.509 не обмежує набір типів атрибутів, які можуть з'явитися в ім'ї. Проте, стандартом рекомендується підтримувати наступні типи атрибутів в іменах випускаючі й суб'єкта:

– Країна.

– Організація.

– Організаційна одиниця.

– Позначення унікального імені.

– Назва штату або регіону.

– Загальноприйняте ім'я (наприклад, Іванов Іван).

– Серійний номер.

Додатково можуть бути присутнім деякі інші типи атрибутів в іменах випускаючі й суб'єкта, наприклад:

– Локалізація.

– Заголовок.

– По батькові.

– Призначене ім'я.

– Ініціали.

– Псевдонім.

– Спеціальна назва (наприклад, "Jr.", " 3-ій" або "IV").

Також може бути присутнім атрибут domainComponent. DNS надає собою ієрархічну систему позначення ресурсів. Даний атрибут надає зручний механізм для організацій, які хочуть використовувати унікальні DN імена паралельно зі своїми DNS-Іменами. Це не заміняє dNSName компонент альтернативного поля ім'я. Стандарт не вимагає конвертувати такі імена в DNS-Імена.

Сторона, що перевіряє, повинна обробляти поля унікального ім'я випускаючого й унікального ім'я суб'єкта для одержання ланцюжка імен при перевірці *дійсності сертифікаційного* шляху. Ланцюжок імен виходить у випадку відповідності унікального ім'я випускаючого в першому сертифікаті ім'я суб'єкта в сертифікаті УЦ.

У якості функціональної схеми, так як вона є складовою частиною структурної схеми наведемо функціональну схему роботи протоколу SSL/TLS. Цей протокол є складовою системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

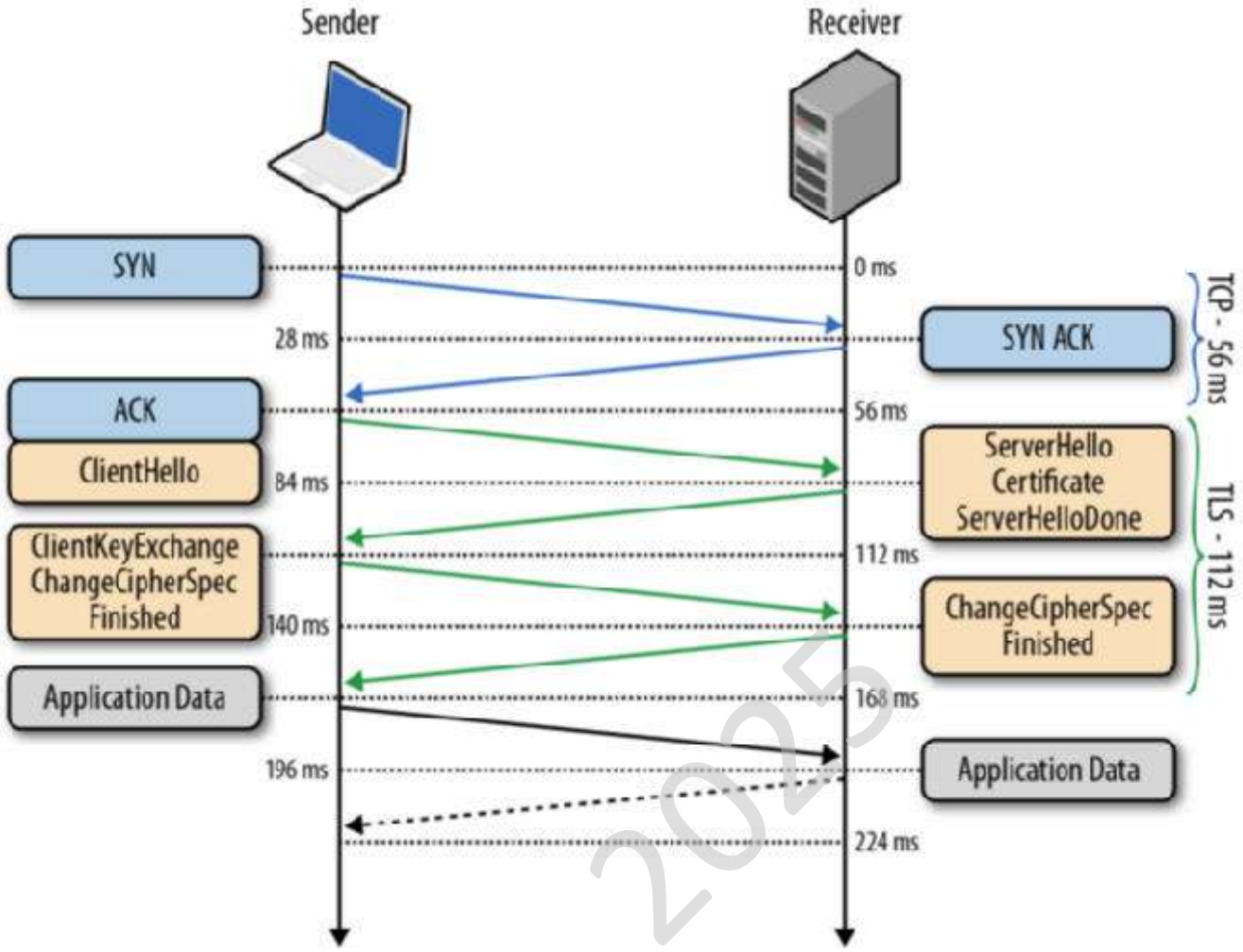

Рисунок 3.2 – Функціональна схема системи

Опис SSL/ TLS

SSL – криптографічний протокол, що забезпечує безпечну передачу даних по мережі Інтернет. При його використанні створюється захищене з'єднання між клієнтом і сервером. SSL споконвічно розроблений компанією Netscape Communications. Згодом на підставі протоколу SSL 3.0 був розроблений і прийнятий стандарт RFC, що одержав ім'я TLS.

Використовує шифрування з відкритим ключем для підтвердження дійсності передавача й одержувача. Підтримує надійність передачі даних за рахунок використання коригувальних кодів і безпечних хеш-функцій.

SSL складається із двох рівнів. На нижньому рівні багаторівневого транспортного протоколу (наприклад, TCP) він є протоколом запису й використовується для інкапсуляції (тобто формування пакета) різних протоколів (SSL працює разом з таким протоколами як POP3, IMAP, XMPP, SMTP і HTTP). Для кожного інкапсульованого протоколу він забезпечує умови, при яких сервер і клієнт можуть підтверджувати один одному свою дійсність, виконувати алгоритми шифрування й робити обмін криптографічними ключами, перш ніж протокол прикладної програми почне передавати й одержувати дані.

Для доступу до веб-сторінок, захищеним протоколом SSL, в URL замість звичайного префікса http, як правило, застосовується префікс https, що вказує на те, що буде використовуватися SSL-з'єднання. Стандартний TCP-порт для з'єднання по протоколу https – 443.

Для роботи SSL потрібно, щоб на сервері був SSL-сертифікат.

TLS – криптографічний протокол, що забезпечує захищену передачу даних між вузлами в мережі Інтернет. TLS-протокол заснований на Netscape SSL-протоколі версії 3.0 і складається із двох частин – TLS Record Protocol і TLS Handshake Protocol. Розходження між SSL 3.0 і TLS 1.0 незначні, тому далі в тексті термін «SSL» буде відноситися до них обох. TLS Working Group, заснована в 1996 році, продовжує працювати над протоколом.

Опис

TLS надає можливості автентифікації й безпечної передачі даних через Інтернет з використанням криптографічних засобів. Часто відбувається лише автентифікація сервера, у той час як клієнт залишається неавтентифікованим. Для взаємної автентифікації кожна зі сторін повинна підтримувати інфраструктуру відкритого ключа (PKI), що дозволяє захистити клієнт-серверні додатки від перехоплення повідомлень, редагування існуючих повідомлень і створення підроблених.

SSL містить у собі три основних фази:

– Діалог між сторонами, метою якого є вибір алгоритму шифрування.

– Обмін ключами на основі криптосистем з відкритим ключем або автентифікація на основі сертифікатів.

– Передача даних, шифруємих за допомогою симетричних алгоритмів шифрування.

Алгоритм процедури встановлення з'єднання по протоколу TLS handshake

Клієнт і сервер, що працюють по TLS, установлюють з'єднання, використовуючи процедуру handshake ("рукостискання"). Протягом цього handshake, клієнт і сервер приймають угоду щодо параметрів, використовуваних для встановлення захищеного з'єднання.

Послідовність дій при встановленні TLS з'єднання:

– клієнт підключається до TLS – підтримуваного сервера й запитує захищене з'єднання;

– клієнт надає список підтримуваних алгоритмів шифрування й хеш-функцій;

– сервер вибирає зі списку, наданого клієнтом, найбільш стійкі алгоритми, які також підтримуються сервером, і повідомляє про свій вибір клієнтові;

– сервер відправляє клієнтові цифровий сертифікат для власної ідентифікації. Звичайно цифровий сертифікат містить ім'я сервера, ім'я довіреного центра сертифікації й відкритий ключ сервера;

– клієнт може зв'язатися із сервером довіреного центра сертифікації й підтвердити автентичність переданого сертифіката до початку передачі даних;

– для того щоб згенерувати ключ сесії для захищеного з'єднання, клієнт шифрує випадково згенерировану цифрову послідовність відкритим ключем сервера й посилає результат на сервер. З огляду на специфіку алгоритму асиметричного шифрування, використовуваного для встановлення з'єднання, тільки сервер може розшифрувати отриману послідовність, використовуючи свій закритий ключ;

Handshake у деталях

Відповідно до протоколу TLS додатки обмінюються записами, інкапсулюючими (що зберігають усередині себе) інформацію, яка повинна бути передана. Кожен із записів може бути стисла, доповнена, зашифрована або ідентифікована MAC залежно від поточного стану з'єднання (стану протоколу). Кожний запис в TLS містить наступні поля: content type (визначає тип умісту запису), поле, що вказує довжину пакета, і поле, що вказує версію протоколу TLS.

Коли з'єднання тільки встановлюється, взаємодія йде по протоколу TLS handshake, content type якого 22.

Нижче описаний простий приклад установлення з'єднання:

1. Клієнт посилає повідомлення **ClientHello**, указуючи найбільш останню версію підтримуваного TLS протоколу, випадкове число й список підтримуваних методів шифрування й стиски, що підходять для роботи з TLS.

2. Сервер відповідає повідомленням ServerHello, що містить: обрану сервером версію протоколу, випадкове число, послане клієнтом, що підходить алгоритм шифрування й стиски зі списку наданого клієнтом.

3. Сервер посилає повідомлення **Certificate**, що містить цифровий сертифікат сервера (залежно від алгоритму шифрування цей етап може бути пропущений)

4. Сервер може запросити сертифікат у клієнта, у такому випадку з'єднання буде взаємно автентифіковано.

5. Сервер відсилає повідомлення **ServerHelloDone**, що ідентифікує закінчення handshake.

6. Клієнт відповідає повідомленням **ClientKeyExchange**, що містить PreMasterSecret відкритий ключ, або нічого (знову ж залежить від алгоритму шифрування).

7. Клієнт і сервер, використовуючи PreMasterSecret ключ і випадково згенеровані числа, обчислюють загальний секретний ключ. Вся інша інформація

про ключ буде отримана із загального секретного ключа (і згенерованих клієнтом і сервером випадкових значень).

8. Клієнт посилає **ChangeCipherSpec** повідомлення, що вказує на те, що вся наступна інформація буде зашифрована встановленим у процесі handshake алгоритмом, використовуючи загальний секретний ключ. Це повідомлення рівня записів і тому має тип 20, а не 22.

9. Клієнт посилає повідомлення **Finished**, що містить хеш і MAC, згенеровані на основі попередніх повідомлень handshake.

10. Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити хеш і MAC. Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване.

11. Сервер посилає **ChangeCipherSpec** і зашифроване **Finished** повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

Із цього моменту handshake уважається завершеним, протокол установленим. Весь наступний уміст пакетів іде з типом 23, а всі дані будуть зашифровані.

Алгоритми, що використовуються в TLS

У даній поточній версії протоколу доступні наступні алгоритми:

– Для обміну ключами й перевірки їхньої дійсності застосовуються комбінації алгоритмів: RSA (асиметричний шифр), Diffie-Hellman (DH) (безпечний обмін ключами), DSA (алгоритм цифрового підпису) і алгоритми технології Fortezza.

– Для симетричного шифрування: RC2, RC4, IDEA, DES, Triple DES або AES;

– Для хеш-функцій: MD5 або SHA.

Алгоритми можуть доповняться залежно від версії протоколу.

У архітектурі SSL/TLS задіяні наступні протоколи:

– HTTP – протокол прикладного рівня передачі даних у першу чергу у вигляді текстових повідомлень. Основою HTTP є технологія « клієнт-сервер»,

тобто передбачається існування споживачів (клієнтів), які ініціюють з'єднання й надсилають запит, і постачальників (серверів), які очікують з'єднання для одержання запиту, роблять необхідні дії й повертають обернено повідомлення з результатом. HTTP використовується також у якості «транспорту» для інших протоколів прикладного рівня, таких як SOAP. Основним об'єктом маніпуляції в HTTP є ресурс, на який вказує URI (Uniform Resource Identifier) у запиті клієнта. Звичайно такими ресурсами є файли, що зберігаються на сервері, але ними можуть бути логічні об'єкти або щось абстрактне. Особливістю протоколу HTTP є можливість указати в запиті й відповіді спосіб подання того самого ресурсу по різних параметрах: формату, кодуванню, мові й т.д. Саме завдяки можливості вказівки способу кодування повідомлення клієнт і сервер можуть обмінюватися двійковими даними, хоча даний протокол є текстовим. HTTP – протокол прикладного рівня, аналогічними йому є FTP і SMTP. Обмін повідомленнями йде за звичайною схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує глобальні URI. На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаної з останніми запитами й відповідями. Браузер, що посилає запити, може відслідковувати затримки відповідей. Сервер може зберігати IP-адреси й заголовки запитів останніх клієнтів. Однак сам протокол не обізнаний про попередні запити й відповіді, у ньому не передбачена внутрішня підтримка стану, до нього не пред'являються такі вимоги.

– FTP – протокол, призначений для передачі файлів у комп'ютерних мережах. FTP дозволяє підключатися до серверів FTP, переглядати вміст каталогів і завантажувати файли із сервера або на сервер; крім того, можливий режим передачі файлів між серверами. Протокол FTP відноситься до протоколів прикладного рівня й для передачі даних використовує транспортний протокол TCP. Команди й дані, на відміну від більшості інших протоколів передаються по

різних портах. Порт 20 використовується для передачі даних, порт 21 для передачі команд. Протокол не шифрується, при автентифікації передає логін і пароль відкритим текстом. Якщо зловмисник перебуває в одному сегменті мережі з користувачем FTP, те, використовуючи сніффер, він може перехопити логин і пароль користувача, або, при наявності спеціального ПЗ, одержувати передані по FTP файли без авторизації. Щоб запобігти перехопленню трафіку, необхідно використовувати протокол шифрування даних SSL, що підтримується багатьма сучасними FTP-серверами й деякими FTP-Клієнтами.

– LDAP – це мережний протокол для доступу до служби каталогів X.500, розроблений IETF як полегшений варіант розробленого ITU-T протоколу DAP. LDAP – відносно простий протокол, що використовує TCP/IP і дозволяє робити операції автентифікації (bind), пошуку (search) і порівняння (compare), а також операції додавання, зміни або видалення записів. Звичайно LDAP-сервер приймає вхідні з'єднання на порт 389 по протоколах TCP або UDP. Для LDAP-сеансів, інкапсульованих в SSL, звичайно використовується порт 636. Усякий запис у каталозі LDAP складається з одного або декількох атрибутів і має унікальне ім'я (DN). Унікальне ім'я складається з одного або декількох відносних унікальних імен (RDN), розділених комою. На одному рівні каталогу не може існувати двох записів з однаковими відносними унікальними іменами. У силу такої структури унікального ім'я запису в каталозі LDAP можна легко представити у вигляді дерева. Запис може складатися тільки з тих атрибутів, які визначені в описі класу запису (object class), які, у свою чергу, об'єднані в схеми (schema). У схемі визначено, які атрибути є для даного класу обов'язковими, а які – необов'язковими. Також схема визначає тип і правила порівняння атрибутів. Кожний атрибут запису може зберігати кілька значень.

– TCP – один з основних мережних протоколів Internet, призначений для керування передачею даних у мережах і під мережах TCP/IP. Виконує функції протоколу транспортного рівня спрощеної моделі OSI. IP-ідентифікатор – 6. TCP – це транспортний механізм, що надає потік даних, з попередньою

установкою з'єднання, за рахунок цього даючий впевненість у вірогідності одержуваних даних, здійснює повторний запит даних у випадку втрати даних і усуває дублювання при одержанні двох копій одного пакета. На відміну від UDP, гарантує, що додаток одержить дані точно в такій же послідовності, у якій вони були відправлені, і без втрат. Реалізація TCP як правило, убудована в ядро системи, хоча є й реалізації TCP у контексті додатка. TCP часто позначають «TCP/IP». Коли здійснюється передача від комп'ютера до комп'ютера через Internet, TCP працює на верхньому рівні між двома кінцевими системами, наприклад, інтернет-браузер і інтернет-сервер. Також TCP здійснює надійну передачу потоку байт від однієї програми на деякому комп'ютері в іншу програму на іншому комп'ютері. Програми для електронної пошти й обміну файлами використовують TCP. TCP контролює довжину повідомлення, швидкість обміну повідомленням, мережний трафік.

– IP – маршрутизуємий мережний протокол, основа стека протоколів TCP/IP. Протокол IP використовується для негарантованої доставки даних (поділюваних на так звані пакети) від одного вузла мережі до іншого. Це означає, що на рівні цього протоколу (третій рівень мережної моделі OSI) не дається гарантій надійної доставки пакета до адресата. Зокрема, пакети можуть прийти не в тому порядку, у якому були відправлені, продублюватися (коли приходять дві копії одного пакета; у реальності це буває вкрай рідко), виявитися ушкодженими (звичайно ушкоджені пакети знищуються) або не прибути зовсім. Гарантії безпомилкової доставки пакетів дають протоколи більш високого (транспортного) рівня мережної моделі OSI – наприклад, TCP – які IP використовують як транспорт. У сучасній мережі Інтернет використовується IP четвертої версії, також відомий як IPv4. У протоколі IP цієї версії кожному вузлу мережі відноситься у відповідність IP-адреса довжиною 4 октети (іноді говорять «байта», маючи на увазі розповсюджений восьмибітовий мінімальний адресуємий фрагмент пам'яті ЕОМ). При цьому комп'ютери в підмережах поєднуються загальними початковими бітами адреси. Кількість цих біт, загальна

для даної підмережі, називається маскою підмережі (раніше використовувалося ділення простору адрес по класах – A, B, C; клас мережі визначався діапазоном значень старшого октету й визначав число адресуємих вузлів у даній мережі, зараз використовується безкласова адресація). У поточний час вводиться в експлуатацію шоста версія протоколу – IPv6, що дозволяє адресувати значно більшу кількість вузлів, чим IPv4. Ця версія відрізняється підвищеною розрядністю адреси, убудованою можливістю шифрування й деяких інших особливостей. Перехід з IPv4 на IPv6 пов'язаний із трудомісткою роботою операторів зв'язку й виробників програмного забезпечення й не може бути виконаний одномоментно.

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

3.4 Розробка діаграми процесів

Діаграма процесів розробленої системи зображена на рисунку 3.3. При детальному її розгляді можна побачити як саме проходе взаємодія у розробленій системі. Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі.

Діаграма взаємодії процесів використовується для візуалізації процесів обробки даних (структурне проектування). Для розробника вважається звичним спочатку креслити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи. Ця діаграма в подальшому підлягає уточненню шляхом деталізації процесів та потоків даних з метою показати систему що розробляється. Діаграми потоків даних містять чотири типи елементів: Процеси які являють собою трансформацію даних в рамках описуваної системи; Сховища даних (репозиторії); Зовнішні по відношенню до системи сутності; Потоки даних між елементами трьох попередніх типів.

Рисунок 3.3 – Діаграма взаємодії процесів

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

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

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

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

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

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

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

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

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

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

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

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

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

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

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

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає оптимізації програми високого рівня, також те, що при розробці програми слід надати особливу увагу модулю захисту корпоративних додатків, які розробляються за допомогою методології Agile.

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

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

Було використано підходи з використанням UML, це уніфікована мова моделювання, використовується у парадигмі об'єктно-орієнтованого програмування. Є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML є мовою широкого профілю, це відкритий стандарт, що використовує графічні позначення для створення абстрактної моделі системи, називаної UML-моделлю.

UML був створений для визначення, візуалізації, проектування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

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

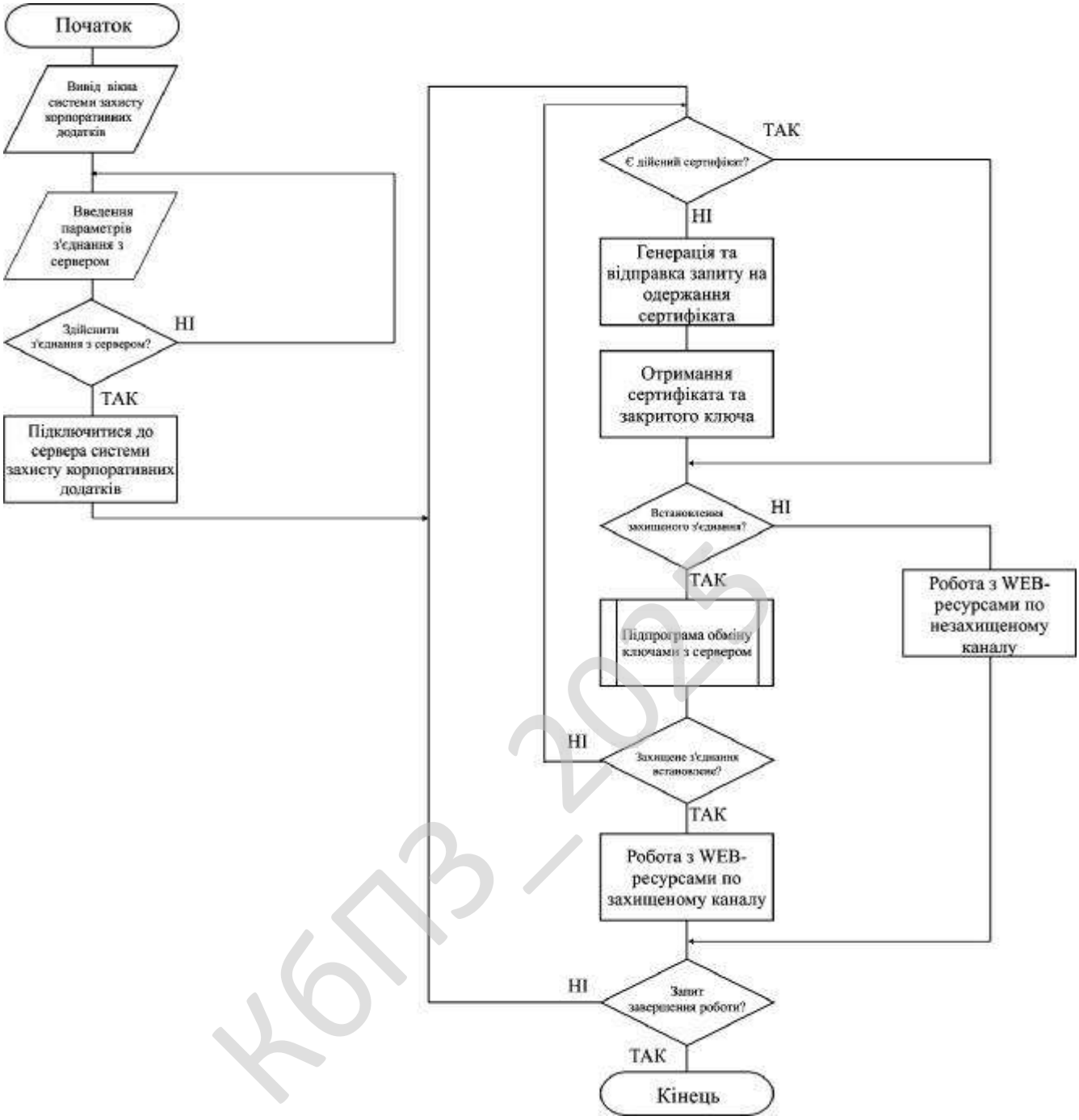

Рисунок 4.1 – Блок-схема основної програми

Діаграми дають можливість представити систему (як ділову, так і програмну) у такому вигляді, щоб її можна було легко перевести в програмний код.

Основною причиною використання мови UML є спілкування розробників між собою.

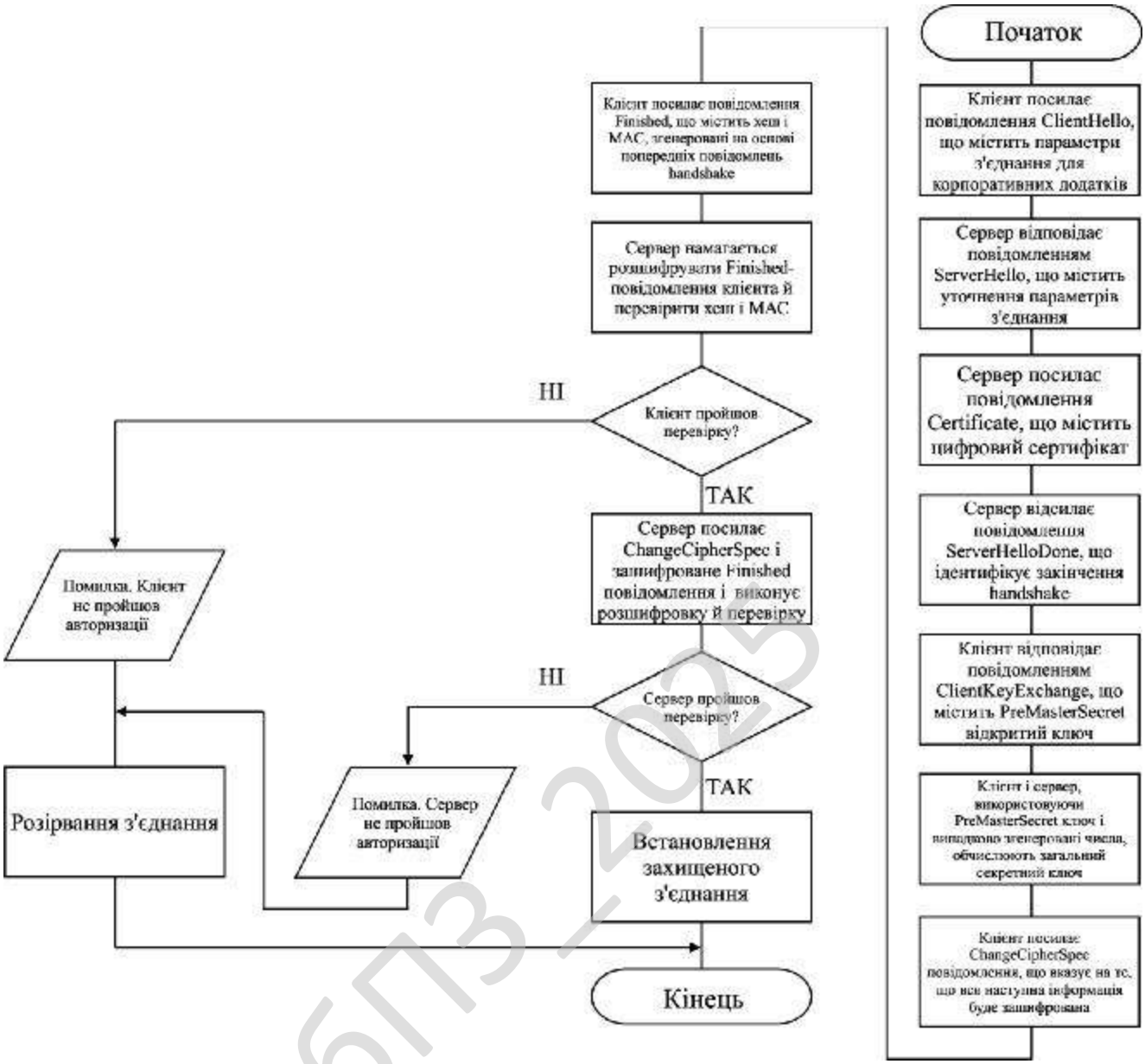

Рисунок 4.2 – Блок-схема роботи підпрограми

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

UML прекрасно зарекомендувала себе в багатьох успішних програмних проектах. Засоби автоматичної генерації кодів дозволяють перетворювати моделі мовою UML у вихідний код об'єктно-орієнтованих мов програмування, що ще більш прискорює процес розробки. Практично усі CASE-засоби (програми

автоматизації процесу аналізу і проектування) мають підтримку UML. Моделі розроблені в UML, дозволяють значно спростити процес кодування і направити зусилля програмістів безпосередньо на реалізацію системи.

Діаграми підвищують супроводжуваність проекту і полегшують розробку документації.

UML необхідний:

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

– Проектувальникам інформаційних систем які розробляють технічні завдання для програмістів.

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

– Програмістам які реалізовують модулі інформаційної системи.

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

Процедура ініціалізації рукопотискання (handshake) у розробленій у результаті виконання роботи програмі виглядає наступним чином:

```
void SSLConnection::InitiateHandShake(String* ipAddres, Byte thumbPrint[],
Common::Misc::SecurityProviderProtocol prot, Object* state)
    {
        if(m_ServerIP != NULL)// Йдемо по списку серверів
        {
            Dispose();
            Init(); //ініціалізуємо
        }
        m_ServerIP = ipAddres; // визначаємо IP адресу сервера
        try
        {
            SetupCredentials(thumbPrint, prot);//Встановлюємо автентифікатор
            PerformHandShake(state);
```

```
        }
        catch(Common::Exceptions::SSLException*)
        {
            Dispose();
            throw;
        }
    }
```

Процедура посилання клієнтом повідомлення ClientHello та його обробки виглядає наступним чином:

```
    void SSLConnection::PerformHandShake(Object* state)
    {
        DWORD           dwSSPIFlags;
        DWORD           dwSSPIOutFlags;
        TimeStamp       tsExpiry;
        SECURITY_STATUS scRet = S_OK;
        dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT   |
                      ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR  |
                      ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;
        //
        //  Ініціалізуємо повідомлення ClientHello та генеруємо токен.
        //
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);
        OutBuffer.SetSecurityBufferToken(0, NULL, 0);

        IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemAnsi(m_ServerIP);
        scRet = m_pSecurityFunc->InitializeSecurityContextA( m_phClientCreds,
NULL,
                        static_cast<SEC_CHAR*>(ptrServerName.ToPointer()),
                        dwSSPIFlags,0,
                        SECURITY_NATIVE_DREP,
                        NULL, 0, m_phContext, &OutBuffer,
                        &dwSSPIOutFlags, &tsExpiry);
        System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);
        if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
        {
            DoRenegotiate(this);
        }
        else if(scRet != SEC_I_CONTINUE_NEEDED)
        {
          throw new Common::Exceptions::
```

```
            SSLException(S"InitializeSecurityContext Помилкове. Помилка: ", scRet);
        }
        m_bInHandShake = true;
        // Відправлення відповіді на сервер, якщо він готовий.
        if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
        {
            bool bSent = DispatchSend(static_cast<char*>(OutBuffer[0].pvBuffer),
OutBuffer[0].cbBuffer, state);
            if(!bSent)
            {
                throw new Common::Exceptions::SSLSendException(S"Відправлення
помилки Сервера.");
            }
        }
    }
```

Далі сервер посилає повідомлення Certificate, що містить цифровий сертифікат сервера та повідомлення ServerHelloDone, що ідентифікує закінчення handshake.

Клієнт відповідає повідомленням ClientKeyExchange, що містить PreMasterSecret відкритий ключ.

Після цього клієнт і сервер, використовуючи PreMasterSecret ключ і випадково згенеровані числа, обчислюють спільний секретний ключ. Вся інша інформація про ключ буде отримана із спільно секретного ключа (і згенерованих клієнтом і сервером випадкових значень).

Клієнт посилає ChangeCipherSpec повідомлення, що вказує на те, що вся наступна інформація буде зашифрована встановленим у процесі handshake алгоритмом, використовуючи спільний секретний ключ.

Клієнт посилає повідомлення Finished, що містить хеш і MAC, згенеровані на основі попередніх повідомлень handshake.

Сервер намагається розшифрувати Finished-повідомлення клієнта й перевірити хеш і MAC. Якщо процес розшифровки або перевірки не вдається, handshake вважається невдалим і з'єднання повинне бути обірване. У разі вдалої перевірки клієнта, сервер посилає ChangeCipherSpec і зашифроване Finished повідомлення й у свою чергу клієнт теж виконує розшифровку й перевірку.

Якщо і клієнт і сервер пройшли перевірку, то встановлюється захищене з'єднання. Процедура перегляду руко потискань клієнтом описується наступним чином:

```
    bool SSLConnection::ClientHandshakeLoop(void* IoBuffer, int& ActualLen,
SecBuffer *pExtraData, Object* state)
    {
        CAutoSecBuffer<2> InBuffer(m_pSecurityFunc, false);
        CAutoSecBuffer<1> OutBuffer(m_pSecurityFunc, true);

        TimeStamp tsExpiry;
        DWORD dwSSPIOutFlags;
        DWORD dwSSPIFlags = ISC_REQ_SEQUENCE_DETECT | ISC_REQ_REPLAY_DETECT |
                            ISC_REQ_CONFIDENTIALITY | ISC_RET_EXTENDED_ERROR |
                            ISC_REQ_ALLOCATE_MEMORY | ISC_REQ_STREAM;

        SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
        while(scRet == SEC_I_CONTINUE_NEEDED || scRet ==
SEC_I_INCOMPLETE_CREDENTIALS)
         {
            //
// Встановлення вхідних буферів. Буфер 0 використовує шифрування в даних отриманих
// з серверу. Schannel читає усі дані або тільки признаки. Відложені дані будуть
// накопичуватися у буфері  1 та надавати буферу тип SECBUFFER_EXTRA.
            //

            InBuffer.SetSecurityBufferToken(0, IoBuffer, ActualLen);
            InBuffer.SetSecurityBufferEmpty(1);
            OutBuffer.SetSecurityBufferToken(0, NULL, 0);
            scRet = m_pSecurityFunc->InitializeSecurityContextA(m_phClientCreds,
                                        m_phContext,
                                        NULL,
                                        dwSSPIFlags,
                                        0,
                                        SECURITY_NATIVE_DREP,
                                    &InBuffer,
                                    0,
                                    NULL,
                                    &OutBuffer,
                                    &dwSSPIOutFlags,
                                    &tsExpiry);
```

```
// Якщо InitializeSecurityContext працює правильно (або якщо помилка припустима),
// відправляємо зміст вихідного буфера на сервер.
            if(scRet == SEC_E_OK || scRet == SEC_I_CONTINUE_NEEDED ||
(FAILED(scRet)
                && (dwSSPIOutFlags & ISC_RET_EXTENDED_ERROR)))
            {
                if(OutBuffer[0].cbBuffer != 0 && OutBuffer[0].pvBuffer != NULL)
                {
         bool bSent = DispatchSend(static_cast<char*>(OutBuffer[0].pvBuffer),
                      OutBuffer[0].cbBuffer, state);
                    if(!bSent)
                    {
throw new Common::Exceptions::
SSLSendException(S"Відправлення на сервер не відбулося.");
                    }
                }
                OutBuffer.FreeBuffer(0);
            }
            //
// Якщо InitializeSecurityContext повертає SEC_E_INCOMPLETE_MESSAGE,
// тоді ми  читаємо наступні данні з сервера.
            //
            if(scRet == SEC_E_INCOMPLETE_MESSAGE)
            {
// Викликаємо повідомлення для збереження змісту буферу у разі
// помилки при подальшому вводі;
            }
//
// Якщо InitializeSecurityContext повертає SEC_E_OK, тоді
// рукопотискання завершено успішно.
            //
            if(scRet == SEC_E_OK)
            {
                //
// Якщо "додатковий" буфер містить дані - то це закодована інформація для
// прикладного протоколу OSI. Це повинно бути збережено. Данні для прикладного
// рівня будуть декодовані з  DecryptMessage.
                //
                if(InBuffer[1].BufferType == SECBUFFER_EXTRA)
                {
                    pExtraData->pvBuffer = malloc(InBuffer[1].cbBuffer);
                    if(pExtraData->pvBuffer == NULL)
```

```
                    {
                        throw new OutOfMemoryException();
                    }
MoveMemory(pExtraData->pvBuffer,(BYTE*)IoBuffer +
(ActualLen - InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
                    pExtraData->cbBuffer   = InBuffer[1].cbBuffer;
                    pExtraData->BufferType = SECBUFFER_TOKEN;
                }
                else
                {
                    pExtraData->pvBuffer   = NULL;
                    pExtraData->cbBuffer   = 0;
                    pExtraData->BufferType = SECBUFFER_EMPTY;
                }
                //
                // Для виходу зберігається у БД
                //
                m_bInHandShake = false;
                if(DoServerCertVerify != NULL)
                {
IntPtr ptrServerName =
System::Runtime::InteropServices::Marshal::StringToCoTaskMemUni(m_ServerIP);
                 Common::Misc::CeriticateInfo ServCertInfo;
                 VerifyCertificate(true, m_pSecurityFunc, m_phContext,
static_cast<wchar_t*>(ptrServerName.ToPointer()), 0, &ServCertInfo);
                        DoServerCertVerify(ServCertInfo);
System::Runtime::InteropServices::Marshal::FreeCoTaskMem(ptrServerName);
                    DoServerCertVerify = NULL;
//заборона інших викликів під час виконання процедури renegotiation.
                }
                if(DoHandShakeSuccess != NULL)
                {
                    DoHandShakeSuccess();
                }
                break;
            }
//
// Крапка невиправної помилки .
//
            if(FAILED(scRet))
            {
```

```
throw new Common::Exceptions::
SSLException(S"Руко потискання з сервером помилкове. Помилка: ", scRet);
            }
            //
// Якщо InitializeSecurityContext повертає SEC_I_INCOMPLETE_CREDENTIALS,
// тоді сервер автентифікує клієнта.
            //
            if(scRet == SEC_I_INCOMPLETE_CREDENTIALS)
            {
                if(DoRenegotiate != NULL)
                    DoRenegotiate(this);
                SECURITY_STATUS scRet = SEC_I_CONTINUE_NEEDED;
                continue;
            }
//
// Копіюємо любі відложені дані з «додаткового» буфера й виконуємо
// наступний раунд.
//
            if ( InBuffer[1].BufferType == SECBUFFER_EXTRA )
{ int temp = ActualLen;
MoveMemory(IoBuffer,(BYTE*)IoBuffer + (ActualLen -
 InBuffer[1].cbBuffer),InBuffer[1].cbBuffer);
ActualLen = InBuffer[1].cbBuffer; }
            else
            {
ActualLen = 0;
break;
            }
        }
return true;
}
```

Redmine – вільне серверне ПЗ для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на мові Ruby і є ПЗ розробленим з використанням відомого веб-фреймворку Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

Функціональні можливості:

– Ведення декількох проектів.

– Гнучка система доступу з використанням ролей.

– Система відстеження помилок.

– Діаграми Ганта та календар.

– Ведення новин проекту, документів та управління файлами.

– Сповіщення про зміни за допомогою RSS-потоків та електронної пошти.

– Власна Wiki для кожного проекту.

– Форуми для кожного проекту.

– Облік часових витрат.

– Налаштування власних (custom) полів для задач, затрат часу, проектів та користувачів.

– Легка інтеграція із системами керування версіями (SVN, CVS, Git, Mercurial, Bazaar и Darcs).

– Створення записів про помилки на основі отриманих листів

– Підтримка LDAP автентифікації.

– Можливість самореєстрації нових користувачів.

– Багатомовний інтерфейс (у тому числі українська мова).

– Підтримка СКБД: MySQL, PostgreSQL, SQLite.

Діаграма Ганта (*Gantt chart*, також стрічкова діаграма, графік Ганта) – це популярний тип діаграм, який використовується для ілюстрації плану, графіка робіт за будь-яким проектом. Є одним з методів планування та управління проектами.

Діаграма Ганта являє собою відрізки (графічні плашки), розміщені на горизонтальній шкалі часу. Кожен відрізок відповідає окремому завданню або підзадачі. Завдання і підзадачі, складові плану, розміщуються по вертикалі. Початок, кінець і довжина відрізка на шкалі часу відповідають початку, кінцю і тривалості завдання. На деяких діаграмах Ганта також показується залежність між завданнями.

Діаграма може використовуватися для представлення поточного стану виконання робіт: частина прямокутника, що відповідає завданню, заштриховується, відзначаючи відсоток виконання завдання; показується вертикальна лінія, що відповідає моменту «сьогодні».

Часто діаграма Ганта використовується спільно з таблицею зі списком робіт, рядки якої відповідають окремо взятій задачі, зображеній на діаграмі, а стовпці містять додаткову інформацію про задачу.

Система відстеження помилок Багтрекер – прикладна програма для допомоги розробникам програмного забезпечення (програмістам, тестувальникам тощо) враховувати і контролювати помилки, знайдені у програмах, питання щодо функціональності, рішення та оновлення, побажання користувачів, а також стежити за процесом їх виконання.

Кожному, хто розробляв програмні продукти, добре знайоме співвідношення «20/80» – останні 20 % роботи тривають 80 % часу.

Як це не парадоксально, але нічого дивного в цій пропорції немає, адже саме на завершальній стадії починається тестування проекту, коли виявляються помилки, і що більший проект, то більше буде знайдено помилок.

Водночас досить часто виявляється, що більшість цих помилок були відомі та могли бути виправлені з меншими витратами на попередніх стадіях роботи, але не були вчасно описані, а потім загубилися серед інших важливих завдань.

Отже, система відстеження помилок у найпростішому варіанті – це процес, що включає в себе виявлення помилки, її опис, виправлення і перевірку цього виправлення, тобто процес «стеження» за багом протягом всього як його життєвого циклу, так і життєвого циклу розробки в цілому.

Сукупність інформації про дефект. Головний компонент такої системи – база даних, що містить відомості про виявлені дефекти. Ці відомості можуть включати в себе:

– номер (ідентифікатор) дефекту;

– хто повідомив про дефект;

– дата і час виявлення дефекту;

– версія продукту, в якій виявлено дефект;

– серйозність (критичність) дефекту та пріоритет рішення;

– опис кроків для відтворення дефекту (неправильної поведінки програми);

– відповідальний за усунення дефекту;

– обговорення можливих рішень та їх наслідків;

– поточний стан виправлення дефекту;

– версії продукту, в якій дефект виправлений.

Крім того, розвинені системи надають можливість прикріплювати файли, які допомагають описати проблему, наприклад, дамп пам'яті або скріншот.

Використання. Основна перевага систем відстеження помилок полягає в забезпеченні чітких централізованих оглядів, запитів на розробку (включаючи помилки і виправлення) та їх стан. У корпоративному середовищі, системи відстеження помилок можуть бути використані для генерації звітів по продуктивності програмістів виправлення помилок. Однак, це може іноді приводити до неточних результатів, тому що різні помилки можуть мати різні ступені пріоритету та серйозності, що пов'язано з складністю їх фіксації.

Життєвий цикл дефекту. Як правило, система відстеження помилок використовує той чи інший варіант «життєвого циклу» помилки, стадія якого визначається поточним станом помилки.

Типовий життєвий цикл дефекту:

1. Новий – дефект зареєстрований тестувальником.

2. Призначений – призначений відповідальний за виправлення дефекту.

3. Дозволений – дефект переходить назад у сферу відповідальності тестувальника. Як правило, супроводжується резолюцією, наприклад:

– Виправлено (виправлення включені у версію таку-то).

– Дубль (повторює дефект, що вже знаходиться в роботі).

– Не виправлено (працює відповідно до специфікації, має занадто низький пріоритет, виправлення відкладено до наступної версії тощо).

– «В мене все працює» (запит додаткової інформації про умови, в яких дефект проявляється).

4. Далі тестувальник проводить перевірку виправлення, залежно від чого дефект або знову переходить у стан «Призначений» (якщо він описаний як виправлений, але не виправлений), або у стан «Закрито».

5. Відкрито повторно – дефект знайдено знову в іншій версії.

Система може надавати адміністраторові можливість налаштування користувачі, які можуть переглядати і редагувати помилки залежно від їх стану, переводити їх в інший стан або видаляти.

У корпоративному середовищі, система відстеження помилок може використовуватися для отримання звітів, що показують продуктивність програмістів при виправленні помилок. Однак, часто такий підхід не дає достатньо точних результатів через те, що різні помилки мають різну ступінь серйозності та складності. При цьому серйозність проблеми прямо не стосується складності її усунення.

Також при розробці магістерської дипломної роботи було використано наступні підходи UML: діаграма діяльності (діаграми поведінки типу.

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

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

дозволяти виконання декількох (логічних) потоків, та існування механізмів синхронізації для гарантування виконання дій у правильному порядку.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Sinople – симетричний блоковий криптоалгоритм, побудований на основі незбалансованої «мережі Фейстеля». Алгоритм розроблено у 2003 році.

Основні вимоги до алгоритму при його розробці:

– Можливість програмної і апаратної реалізації.

– Висока швидкість.

– Простота.

– Низькі вимоги до пам'яті.

– Високий рівень безпеки.

Алгоритм заснований на 32-розрядних операціях і має 64 раунду, серед яких два типи – C і D. D раунди спроектовані для досягнення максимальної дифузії, C раунди – для досягнення перемішування. F-функція D раунду використовує один з елементів блоку даних (D[3]) та поточного з'єднання (K[r]) для трансформації 3-х елементів блоку даних. F-функція C раунду, навпаки, використовує перші три елемента блоку даних і поточний з'єднання (K[r]) для трансформації останнього елемента блоку даних (D[3]). Раунди D-типу виконуються до раундів C-типу. Додавання ключів з даними проводиться тільки через таблиці замін. Операції XOR (додавання за модулем 2) обов'язково поєднуються з операціями ADD (додавання за модулем 2^{32}).

Таблиці замін спочатку запозичені з алгоритму MARS і містять 512 32-розрядних елементів, проте були жорстко проаналізовано на предмет посилення.

Ключове розклад було спроектовано з урахуванням вимог:

– Простота

– Використовується та ж процедура, що і при шифруванні та розшифрування

– Установка ключа займає менше часу, ніж зашифрування

– Виключення еквівалентних ключів

– Виключення слабких ключів

Алгоритм, згідно із заявою авторів, стійкий до лінійного і диференціального аналізу.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Розглянемо розроблене ПЗ системи захисту корпоративних додатків, які розробляються за допомогою методології Agile яке зображено на рисунку 5.1.

З рисунку можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи: Навігаційне меню (Файл, Налаштування, Параметри захисту, Шаблони, Довідка); Розділу виведення результату роботи системи – стан роботи системи; Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші; Функціональних кнопок ПЗ.

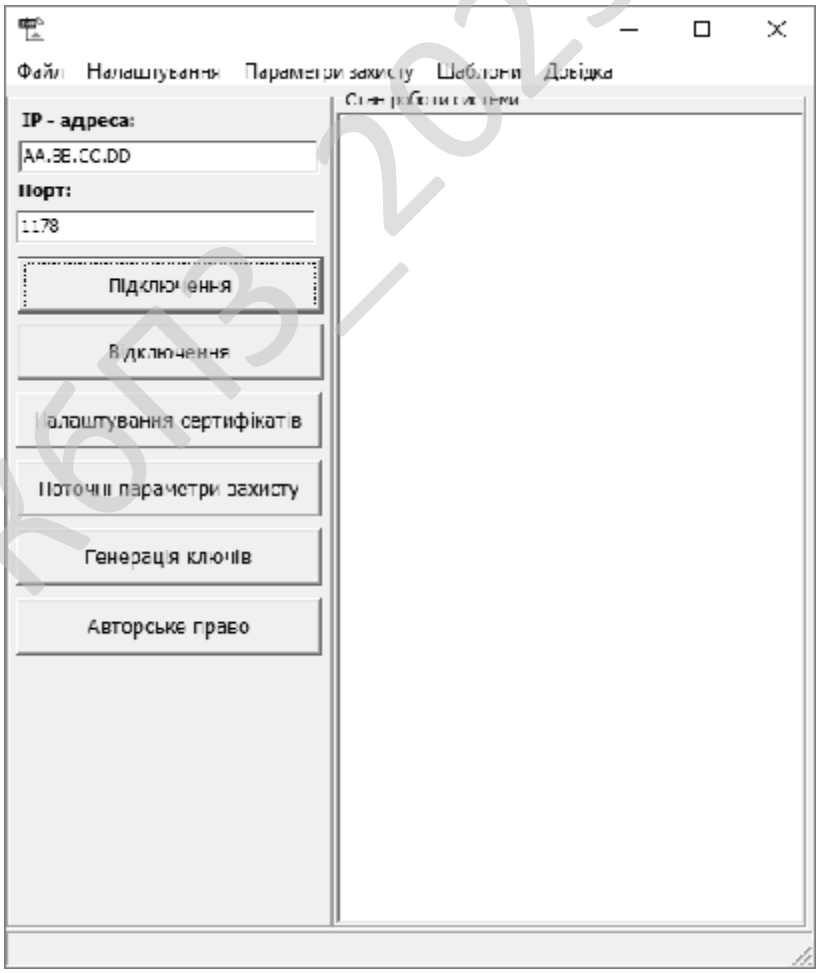

Рисунок 5.1 – Головне вікно ПЗ

Розроблена програма має дуже простий і інтуїтивно 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.

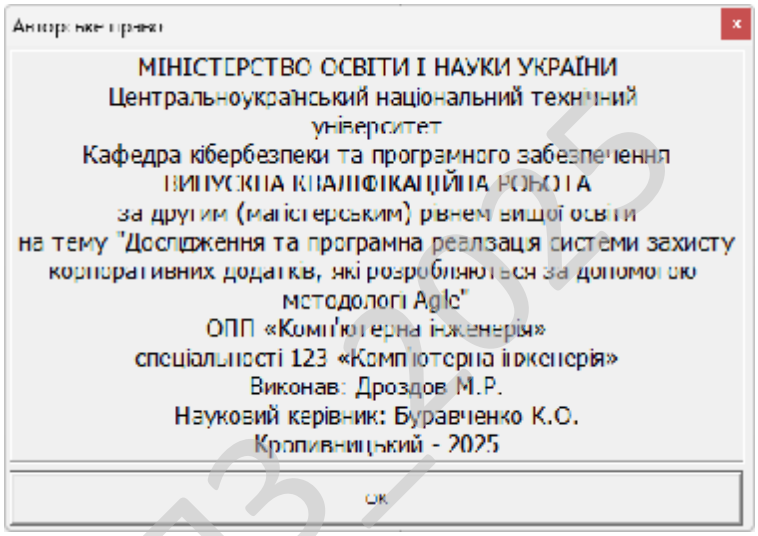

Рисунок 5.2 – Авторське право

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 функцій;

– 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 НАУКОВА НОВИЗНА

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Метою розробки є дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Об'єктом дослідження є процес захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Предметом дослідження є методи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Методи дослідження базуються на методах захисту інформації у комп'ютерних мережах, методах математичної статистики, методах розробки програмного забезпечення.

Наукова новизна отриманих результатів. У процесі рішення завдань, обумовлених цілями дослідження, отримані наступні результати:

– Удосконалено метод захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– Розроблено вітчизняний продукт захисту корпоративних додатків, які розробляються за допомогою методології Agile, який має більш широкі можливості, на відміну від існуючих аналогів.

7 МАРКЕТИНГОВЕ ТА ЕКОНОМІЧНЕ ОБГРУНТУВАННЯ ІТ-ПРОЄКТУ

7.1 Визначення цільової аудиторії кінцевого готового продукту

Результати дослідження та програмної реалізації захисту корпоративних додатків на основі методології Agile можуть бути цікаві багатьом категоріям фахівців і організацій. Зокрема, керівникам ІТ-відділів в компаніях, які активно займаються розробкою програмних рішень, оскільки вони зацікавлені в забезпеченні надійного захисту даних і ефективності процесів. ІТ-фахівці та системні адміністратори, які займаються впровадженням і підтримкою захисту корпоративних систем, також отримають корисну інформацію, адже сучасні методи захисту та інтеграції з Agile дозволяють досягти високої гнучкості та швидкості в роботі з безпекою.

Крім того, менеджери проектів, які керують розробкою корпоративних додатків, будуть зацікавлені у використанні таких рішень для покращення контролю над безпекою без шкоди для швидкості розробки. Фінансові директори та управлінці на вищому рівні також повинні бути зацікавлені в результатах цього дослідження, оскільки ефективне впровадження системи захисту дозволяє знизити витрати на обслуговування і мінімізувати ризики фінансових втрат через кібератаки.

7.2 Оцінка привабливості шляхом застосування методів експертних оцінок

Оцінка привабливості програмної реалізації системи захисту корпоративних додатків за допомогою методів експертних оцінок може бути проведена через застосування методу парних порівнянь. Наприклад, група

експертів може оцінити різні аспекти реалізації таких систем, як ефективність захисту, зручність інтеграції з існуючими рішеннями, масштабованість рішення та вартість впровадження.

Кожен експерт порівнює два аспекти і дає оцінку, який з них є важливішим для конкретної компанії або проекту. Потім всі оцінки зводяться в загальний бал, що дозволяє визначити, яке рішення є найбільш привабливим. Наприклад, якщо експерти з високим балом оцінюють інтеграцію системи з Agile-процесами і її здатність швидко адаптуватися до змін, то це стане вирішальним фактором у виборі рішення для компанії, що прагне підтримувати швидкий темп розробки при високому рівні безпеки.

7.3 Вибір методу оцінки вартості ПЗ

Для оцінки вартості програмної реалізації системи захисту корпоративних додатків, побудованої за методологією Agile, оптимальним методом є метод вартості життєвого циклу (LCC – Life Cycle Costing). Цей метод дозволяє оцінити загальні витрати на проект від етапу планування до виведення з експлуатації.

Оцінка вартості життєвого циклу включає первісні витрати на розробку та впровадження системи, а також поточні витрати на технічне обслуговування, оновлення програмного забезпечення та оновлення безпекових протоколів протягом усього періоду використання.

Окрім того, цей метод дозволяє виявити потенційні витрати на виправлення помилок та обслуговування в майбутньому, що може бути критично важливим при роботі з безпекою, адже витрати на виправлення проблем безпеки можуть значно перевищувати початкові витрати на розробку.

7.4 Розрахунок економічної ефективності від впровадження реалізованого ПЗ як фактору його привабливості

Підприємство активно розробляє корпоративні додатки за методологією Agile, проте процес захисту даних і доступів здійснюється фрагментарно, із залученням сторонніх сервісів і ручних налаштувань без єдиної централізованої системи контролю. Це призводить до дублювання витрат на ліцензії різних систем безпеки; зростання кількості інцидентів (несанкціонований доступ, витік даних); сповільнення розробки через ручне управління сертифікатами, токенами, правами доступу; підвищене навантаження на команду DevOps.

Впровадження єдиної системи захисту корпоративних додатків (SecureAgile Suite) забезпечує: централізоване управління правами доступу та журналами активності; автоматичну інтеграцію з CI/CD-процесами; шифрування даних і зберігання секретів у безпечному сховищі; швидше реагування на інциденти безпеки; зменшення витрат на підтримку та аудит. Вхідні дані зафіксовано в таблиці 7.1.

Розрахунок економічного ефекту демонструє наступне: економія на усуненні інцидентів безпеки – 1 000 000 грн, економія на зовнішніх сервісах захисту – 470 000 грн, економія робочого часу DevOps – 160 000 грн, економія на навчанні персоналу – 100 000 грн, сумарна річна економія – 1 730 000 грн/рік, чистий економічний ефект (1-й рік) – 530 000 грн, термін окупності (Payback Period) ≈ 0,7 року (8–9 місяців), рентабельність інвестицій (ROI) ≈ 144 %.

Додаткові нефінансові переваги: підвищення рівня безпеки корпоративних систем – завдяки централізованому контролю доступу та автоматичному виявленню загроз, скорочення часу виходу продуктів на ринок (Time-to-Market) – завдяки інтеграції механізмів безпеки безпосередньо в Agile-спринти, зменшення людського фактору – автоматизоване управління секретами та токенами, покращення репутації компанії – зниження ризику витоків даних підвищує довіру клієнтів і партнерів, відповідність стандартам ISO/IEC 27001 та GDPR – що спрощує проходження аудитів і сертифікацій, масштабованість – система легко адаптується до нових проєктів без зростання витрат на безпеку.

Таблиця 7.1 – Вихідні дані для розрахунку

Показник	**До впровадження**	**Після впровадження**	**Економічний ефект**
Кількість проєктів, що розробляються одночасно	12	12	—
Середня кількість інцидентів безпеки на рік	18	4	–78%
Середня вартість усунення одного інциденту (аудит, відновлення, простій)	60 000 грн	20 000 грн	–40 000 грн/інцидент
Щорічні витрати на зовнішні сервіси захисту (сертифікати, VPN, моніторинг)	720 000 грн	250 000 грн	–470 000 грн
Витрати часу DevOps на управління безпекою (людино-годин/рік)	800	400	–50%
Вартість 1 людино-години DevOps	400 грн	400 грн	—
Щорічні витрати на навчання персоналу з безпеки	200 000 грн	100 000 грн	–100 000 грн
Початкові інвестиції у впровадження SecureAgile Suite (розробка, ліцензії, інтеграція)	—	—	1 200 000 грн

Все це робить проєкт економічно обґрунтованим і стратегічно вигідним для подальшого розвитку ІТ-інфраструктури підприємства.

7.5 Пропозиція алгоритму просування проєкту розробки ПЗ

Алгоритм просування проєкту програмної реалізації системи захисту корпоративних додатків може включати кілька ключових етапів. Першим етапом є планування та визначення цільової аудиторії, зокрема визначення, хто є кінцевим користувачем цієї системи та які його основні вимоги до безпеки.

Наступним кроком є аналіз конкурентів і ринку: потрібно зібрати інформацію про аналогічні рішення на ринку та визначити конкурентні переваги. Потім слід розробити та презентувати прототипи системи, щоб продемонструвати її основні функції та можливості.

Після цього важливо провести пілотне впровадження системи на обмеженій кількості користувачів або в одному підрозділі компанії. На основі отриманого зворотного зв’язку можна провести оптимізацію продукту та адаптувати його під реальні умови використання. В кінці потрібно запустити масштабування і маркетинг, щоб зробити систему доступною для більш широкої аудиторії і забезпечити її інтеграцію в повний робочий процес компанії.

7.6 Оптимізація каналів збуту та шляхів реалізації ПЗ

Для оптимізації каналів збуту та шляхів реалізації проєкту можна запропонувати кілька стратегій. По-перше, варто звернути увагу на створення партнерських альянсів з компаніями, що займаються впровадженням і підтримкою програмних рішень у сфері безпеки, а також з постачальниками хмарних технологій. Це дозволить розширити канали збуту і залучити нових клієнтів.

По-друге, важливо створити онлайн-платформу для демонстрації можливостей системи: відеоогляди, кейси застосування, вебінари. По-третє, можна реалізувати стратегію "product-led growth", що передбачає надання безкоштовної пробної версії системи для потенційних клієнтів, що дозволить їм самостійно переконатися в ефективності рішення до прийняття рішення про покупку.

7.7 Визначення ключових факторів успіху конкретного проєкту

Ключовими факторами успіху проєкту програмної реалізації захисту корпоративних додатків є інтеграція безпеки на всіх етапах розробки (від планування до тестування), що дозволяє запобігти виникненню вразливостей на пізніх етапах. Важливим фактором є гнучкість і адаптивність системи до змін, адже методологія Agile передбачає постійні зміни вимог та умов розробки. Швидке впровадження та підтримка є також важливими аспектами, оскільки це дозволяє забезпечити неперервність процесів підприємства. Нарешті, зворотний зв'язок від користувачів та безперервне вдосконалення продукту на основі цього зворотного зв’язку є ключовими факторами для підтримки високої якості безпеки в додатках на довгострокову перспективу.

8 ЗАХОДИ З ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

8.1 Вступ

Згідно закону України “Про охорону праці” [3] кожна компанія впроваджує заходи з охорони праці. Реалізується трудові відносини з вживанням необхідних засобів з охорони праці та розробки відповідних документів:

– Інструкцій з охорони праці по кожній професії і загальні.

– Положення про охорону праці.

– Накази з охорони праці.

– Журнали реєстрації та інструктажу.

Роботодавець створює відділ який працює відповідно до типового положення, яку затверджується центральним органом виконавчої влади і забезпечує виконання вимог державної політики у сфері охорони праці.

За недотриманням вимог, керівники ІТ компаній можуть бути притягнуті до відповідальності, яка виглядає у виді накладання штрафу. Якщо в результаті порушення умов охорони праці є постраждалі працівники то керівні особи ІТ компаній притягуються до кримінальної відповідальності.

Законом України “Про охорону праці” [3] регламентуються загальні положення державної політики в галузі охорони праці, а конкретизуються ці положення нормативно-правовими актами про охорону праці, зокрема Наказом Міністерства соціальної політики України 14.02.2018 № 207, який зареєстровано в Міністерстві юстиції України 25 квітня 2018 р. за №508/31960 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» [5], яким затверджено нормативно-правовий акт з охорони праці НПАОП 0.00-7.15-18, «Правила охорони праці під час експлуатації електронно-обчислювальних машин», та «Державні санітарні правила і норми

роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98.

Програмісти у процесі роботи мають негативний вплив на органи зору, а також мають значну розумову напругою і нервово-емоційне навантаження. Руки (суглоби пальців та м'язи рук) при роботі з клавіатурою мають теж істотне навантаженням. До шкідливих факторів, які впливають на робітників галузі інформаційних технологій (ІТ), фахівці відносять високочастотні електромагнітні коливання (випромінювання) роботи апаратної частини ЕОМ та виділення шкідливих газів.

Ці шкідливі фактори можуть привести до професійних захворювань.

Розглянемо шкідливі чинники роботи програмістів керуючись наступними нормативно-правовими актами: «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин» ДСанПіН 3.3.2-007-98 [5], та «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями» НПАОП 0.00-7.15-18.

Умови праці програміста включають наступні фактори:

– параметри повітряного середовища в приміщенні;

– вентиляція приміщення;

– освітлення приміщення;

– параметри повітряного середовища в приміщенні, тощо.

Щоб запропонувати заходи щодо зменшення негативного впливу комп’ютера на організм людини визначимо фактори, які можуть викликати професійне захворювання і впливають на працездатність програміста.

8.2 Аналіз умов праці на робочому місці ІТ-фахівця

На робочому місці ІТ-фахівця (або програміста) виникають небезпечні та шкідливі для безпечної життєдіяльності фактори:

– підвищений рівень шуму;

– несприятливі мікрокліматичні умови;

– недостатній рівень освітленості;

– шкідливі речовини;

– підвищений рівень електромагнітних випромінювань радіочастот;

– висока напруга електричної мережі;

– статична електрика та інші.

Робота програміста супроводжується також підвищеним ступенем напруженості трудового процесу. При систематичному впливі виробничих факторів, які не відповідають нормативним показникам, зростає рівень професійно зумовленої захворюваності працюючих та можуть виникнути професійні захворювання органів зору, руху, нервової системи. Таким чином, вивчення умов праці на робочому місці програміста є необхідною умовою запобігання негативних наслідків впливу небезпечних та шкідливих факторів. Робоче місце, добре пристосоване до трудової діяльності інженера, правильно і доцільно організоване, щодо простору, форми, розміру забезпечує йому зручне положення при роботі і високу продуктивність праці при найменшому фізичному і психічному напруженні.

Нормування параметрів проводиться в залежності від періоду року та категорії важкості виконуваних робіт. Для постійних робочих місць, якими є робочі місця ІТ-фахівців, встановлені оптимальні параметри мікроклімату, а за неможливості їх дотримання використовують допустимі параметри. Робота ІТ-фахівця за важкістю відноситься до Іа (роботи, що виконуються сидячи і не потребують фізичного напруження) та Іб (роботи, що виконуються сидячи, стоячи або пов'язані з ходінням та супроводжуються деяким фізичним напруженням) категорій. В таблиці 8.1. наведені оптимальні параметри мікроклімату в приміщеннях.

Виміряні за допомогою приладів температура та вологість у приміщеннях праці ІТ-фахівців повинні відповідати зазначеним у таблиці для теплого періоду року.

Слід зазначити, що для нормалізації параметрів мікроклімату слід використовувати у приміщеннях кондиціювання повітря, або забезпечити подачу свіжого повітря системами вентиляції. Норми подачі свіжого повітря наведені у таблиці 8.2.

Таблиця 8.1 – Параметри мікроклімату для приміщень з ПК

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	22...24°С; 40... 60%; до 0,1 м/с
Теплий	Температура повітря в приміщенні; відносна вологість; швидкість руху повітря	23...25 °С; 40...60%; 0,1...0,2 м/с

Таблиця 8.2 – Норми подачі свіжого повітря в приміщення

Характеристика приміщення	Об'ємна витрата свіжого повітря, що подається в приміщення, $м^3$ на одну людину в годину
Об'єм до 20 $м^3$ на людину	Не менше 30
20... 40 $м^3$ на людину	Не менше 20
Більше 40 $м^3$на людину	Може бути використана природна вентиляція

Створення сприятливих умов праці і правильне естетичне оформлення робочих місць на виробництві має велике значення як для полегшення праці, так і для підвищення його привабливості, позитивно впливає на продуктивність праці. Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. У службових приміщеннях, у яких виконується одноманітна розумова робота, що вимагає значної нервової напруги і великого зосередження, забарвлення повинно бути спокійних тонів –

малонасичені відтінки холодного зеленого або блакитного кольорів.

При розробці оптимальних умов праці програміста необхідно враховувати освітленість. Раціональне освітлення робочого місця є одним з найважливіших факторів, що впливають на ефективність трудової діяльності людини, що попереджають травматизм і професійні захворювання. Правильно організоване освітлення створює сприятливі умови праці, підвищує працездатність і продуктивність праці. Освітлення на робочому місці програміста повинно бути таким, щоб працівник міг без напруги зору виконувати свою роботу. Стомлюваність органів зору залежить від ряду причин: недостатність освітленості; надмірна освітленість; неправильний напрям світла. Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань. [4]

8.3 Пропозиції щодо підвищення працездатності ІТ-фахівців

Поява та впровадження нових інформаційно-комунікаційних технологій зумовлює необхідність подальшого вдосконалення охорони праці фахівців ІТ-індустрії. Все це потребує розробки нових нормативно-правових актів з регламентації праці та відпочинку фахівців ІТ-індустрії і стандартів підприємств, центрів комп'ютерної техніки, центрів інформаційних технологій, сучасних комп'ютерних класів.

Для підвищення розумової працездатності то зорової роботи повинна здійснюватися ергономічна оптимізація в рамках системи «оператор-термінал», яка сприятиме результативній фізичній та інтелектуальний працездатності і відновленню психосоматичного здоров'я фахівців ІТ-індустрії. Всі наведені заходи щодо вдосконалення охорони праці фахівців ІТ-індустрії повинні

контролюватися службою охорони праці та комісією з охорони праці підприємства.

Особливе значення у соціальному захисті цієї категорії працівників належить прийняття комплексного договору, який може забезпечити фахівців додатковими пільгами та компенсаціями.

Пропозиції щодо підвищення працездатності ІТ-фахівців, розіб'ємо на декілька категорій:

1. Середовище і розпорядок праці.

Для мінімізації негативних ефектів, що пов'язані з перевтомленням ІТ-фахівців, потрібно чітко прописати і реалізувати графік періодів праці-відпочинку, щоб фахівець міг можливість переключити увагу, дати можливість відпочити очам, мозку, елементарно, встати розім'яти ноги.

Також потрібно зробити максимально комфортними умови мікроклімату у офісному приміщенні, де працюють ІТ-фахівці. Мається на увазі встановлення і експлуатація, коли виникає необхідність, кондиціонерів, опалення, та системи вентиляції, задля попередження перегрівання, переохолодження ІТ-фахівців, і подальшої неможливості ними виконувати свої функції.

Також, за можливості, нами пропонується введення практики віддаленої праці ІТ-фахівцями, якщо роботодавець не може забезпечити оптимальні і безпечні умови в офісному приміщенні, або якщо фахівця вони не влаштовують із певних причин.

2. Фізичні і психоемоційні чинники.

Першим і найважливішим чинником, що впливає на працездатність ІТ-фахівців є робоче місце, і саме тому, роботодавець має забезпечити максимальний його комфорт і безпеку.

Гарантією цих факторів може слугувати сертифікація меблів, що використовуються на підприємстві ІТ-галузі. Тому нами пропонується закупівля тільки меблів, які пройшли сертифікацію на відповідність. Під психоемоційними чинниками ми розуміємо гарне самопочуття фахівців, позитивний настій, гарний

психологічний клімат у колективі, тощо. Задля того, щоб психоемоційні чинники мали максимально позитивний ефект, керівництву слід поводити заходи, які сприятимуть укріпленню і покращенню міжособистісних стосунків у колективі, таких як психологічні тренінги, таймбілдінг, спортивні змагання і естафети.

Також, сюди можна віднести розробку і впровадження системи мотивації працівників, як фінансової, так моральної і адміністративної.

8.4 Розрахункова частина

Для захисного штучного заземлення застосовуються вертикальні електроди: металевий куток 50·50·5 мм., довжиною L=3 м., та горизонтальний електрод – металева полоса з перетином 50·5 мм. Напруга – 220/380 В. Розрахункова схема розташування заземлюючих електродів – по контуру (прямокутником).

Розрахунок проведемо за допустимим опором розтіканню струму заземлювача.

Початкові дані для розрахунку захисного заземлення: тип верхнього шару ґрунту – чорнозем, нижнього шару ґрунту – глина (питомий опір ρ_2 = 40 Ом·м). Умовна товщина верхнього шару ґрунту: H=0,6 м. Відстань між вертикальними заземлювачами (електродами) A=3 м. Глибина закладення горизонтального контура заземлення t=0,75 м. Опір заземлювача, який нормується: $R_{ЗН}$ = 4 Ом. Необхідно визначити необхідну кількість вертикальних заземлювачів та довжину полоси (горизонтального заземлювача) (рис. 8.1).

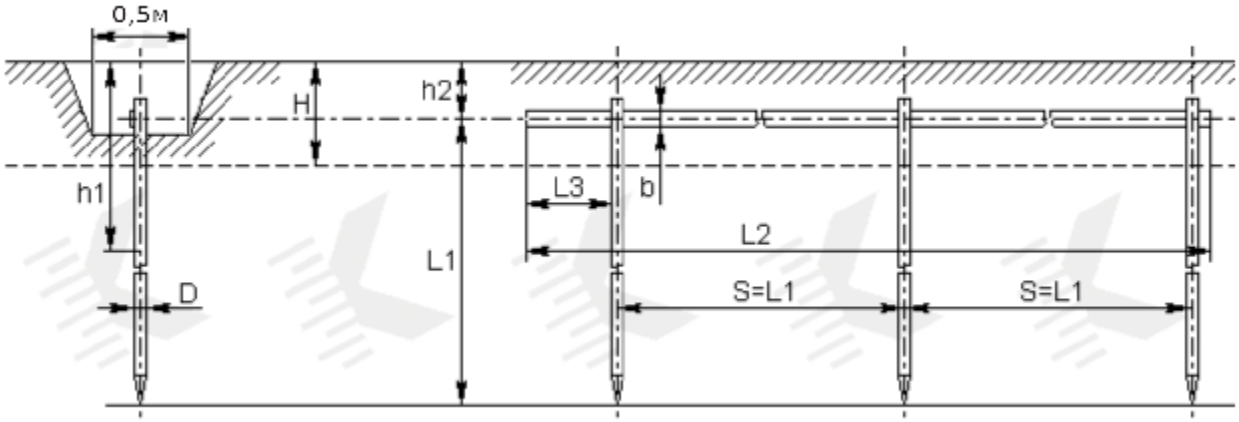

Рисунок 8.1 – Схема штучного заземлення

Виконуємо розрахунок.

Відстань від центра вертикального заземлювача до поверхні землі:

$$T = t + L/2 = 0,75 + 3/2 = 2,25 \text{ м.}$$

Розрахунковий питомий опір ґрунту (з врахуванням того, що фактично вся конструкція заземлювача розташовується у нижньому шарі ґрунту):

$$\rho = \psi \rho = 1,36 \cdot 40 = 54,5 \text{ Ом·м.}$$

де $\psi = 1,36$ – табличне значення коефіцієнта сезонності для відповідної кліматичної зони у багатошаровому ґрунті [11];

$\rho_2 = 40$ Ом·м – табличне значення питомого опору нижнього шару ґрунту (глина) [11].

Еквівалентний діаметр вертикального електрода (кутка) [11]:

$$D_в = 0,95 \cdot K = 0,95 \cdot 50 = 47,5 \text{ мм.} = 0,0475 \text{ м.}$$

де К= 50 мм – розмір металевого кутка (задано).

Відношення A/L=3/3=1.

Опір розтіканню електричного струму одного електрода вертикального заземлювача з урахуванням заглиблення заземлювача [11]:

$$R_O = 0,366 \cdot (\rho/L) \cdot [\lg(2L/D_в) + (1/2) \cdot \lg((4T+L)/(4T-L))] =$$

$$= 0,366 \cdot (54,5/3) \cdot [(\lg(2 \cdot 3/0,0475) + (1/2) \cdot \lg((4 \cdot 2,25+3)/(4 \cdot 2,25-3))] = 14,9 \text{ Ом.}$$

Визначаємо коефіцієнт екранування вертикальних електродів Кев=0,53 при орієнтовній кількості вертикальних електродів, яке дорівнює 5 [11].

Визначаємо необхідну кількість вертикальних електродів заземлювача (без врахування горизонтального заземлювача), при $R_{ЗН}$ = 4 Ом:

$$N=R_О /(Кев\ R_{ЗН})= 14,9 / (0,53\cdot4)= 7,05 \approx 7 \text{ шт.}$$

Визначаємо довжину з'єднуючої полоси:

$$L_П =1,05\cdot A\cdot N= 1,05\cdot3\cdot7=18,5\approx 18 \text{ м.}$$

Опір розтіканню електричного струму з'єднуючої полоси з урахуванням кліматичного коефіцієнта питомого опору ґрунту $K_П$ [11]:

$$R_П =0,366(\rho\cdot K_П/L_П)\lg(2\cdot L_П^{\,2}/(B\cdot t))=$$

$$=0,366(40\cdot5/18)\cdot\lg((2\cdot18^{\,2})/(0,05\cdot0,75))=20,1 \text{ Ом.}$$

де $K_П$=5 – табличне значення кліматичного коефіцієнта питомого опору ґрунту для відповідної кліматичної зони для з'єднуючої полоси [11]:

B = 50 мм = 0,05 м.- ширина з'єднуючої полоси (задана).

Загальний опір розтіканню електричного струму заземлювача [11]:

$$R =(R_О\cdot R_П)/(R_О\cdot\eta_П+ N\cdot R_П\cdot Кев)=$$

$$=(14,9\cdot 20,1)/(14,9\cdot 0,55+ 7\cdot 20,1\cdot 0,53)=3,56 \text{ Ом.}$$

де $\eta_П$ = 0,55 – табличне значення коефіцієнта екранування з'єднуючої полоси [11].

Умова $R \le R_{ЗН}$ виконується ($3,56 \le 4$).

Остаточно кількість вертикальних електродів дорівнює 7.

За потреби можна зменшити кількість електродів заземлювача, зменшивши загальний опір розтіканню електричного струму заземлювача методом зменшення питомого опору ґрунту, домішуючи у ґрунт безпосередньо навколо електродів заземлювача розчини солей NaCl, CaCl, сажу, соду, шлак або спеціальні суміші.

8.5 Висновки до розділу

Дотримання всіх необхідних умов праці не лише сприяє збереженню здоров’я працівників, а також підвищує ефективність виробництва в цілому.

З цих міркувань було здійснено аналіз приміщення, призначеного для праці програмістів, проведено розгляд небезпечних та шкідливих факторів, що негативно впливають на програмістів під час роботи.

Тільки повна усвідомленість працівника про можливі небезпеки, що можуть підстерігати його на робочому місці та дотримання вимог нормативних актів з питань охорони праці та відповідних рекомендацій фахівців, дозволять значною мірою знизити негативний вплив шкідливих та небезпечних факторів при роботі з комп’ютером на організм людини.

Виконано розрахунок захисного штучного заземлення, як одного з ключових факторів безпеки програміста.

9 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти, призначено для системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

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

У випускній кваліфікаційній роботі за другим (магістерським) рівнем вищої освіти наведені теоретичне узагальнення й рішення наукового завдання дослідження методів захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Рішення даного завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– Досліджена система захисту корпоративних додатків, які розробляються за допомогою методології Agile.

– На основі отриманих результатів досліджень створена програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Розроблені під час виконання випускної кваліфікаційної роботи за другим (магістерським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання захисту корпоративних додатків, які розробляються за допомогою методології Agile.

Проведено аналіз предметної галузі в ході якого були виявлені об'єкти, взаємодія яких носить істотний характер для функціональної діяльності предметної галузі, і їхні основні характеристики; побудована алгоритм і вибраний середовище розробки.

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. 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 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм Sinople.

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.

Проведено маркетингове та економічне обґрунтування ІТ-проєкту, що дозволило визначити ключові фактори успіху даного проекту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дроздов М.Р. Дослідження та програмна реалізація системи захисту корпоративних додатків, які розробляються за допомогою методології Agile // Збірник праць молодих науковців ЦНТУ. – Вип. 15. – Кропивницький: ЦНТУ, 2025.

2. Mark S. Merkow. Practical Security for Agile and DevOps. CRC Press. 2022. 236 p

3. Derek Fisher. Application Security Program Handbook. Manning Publications. 2021. 155 p.

4. Cameron Wyatt PH.D. Kali Linux Tutorial. Independently published. 2021. 60 p.

5. Alex Matrosov, Eugene Rodionov, Sergey Bratus. Rootkits and Bootkits. No Starch Press. 2019. 450 p.

6. Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед та ін. «Системи інформаційної зброї та технології інформаційної війни»: підручник / Петрик В.М., Присяжнюк М.М., Аль-Файюмі Халед, Жарков Я.М., Смірнов О.А, Буравченко К.О., Давидюк А.В., Кононович В.Г., Корчинский В.В., Кудирко В.М., Фесенко А.О.; за заг. ред. В.М. Петрика, М.М. Присяжнюка.– К.: Видавничий центр “Кафедра”, 2025.– 320 с.

7. Усік, П.С., Смірнова, Т.В., Буравченко, К.О., Смірнов, О.А., Улічев, О.С., Смірнов, С.А. «Дослідження технологій забезпечення кібербезпеки банківських систем з використанням штучного інтелекту». *Кібербезпека: освіта, наука, техніка.* 2025. Том 1 № 29. С.704–716, 2025

8. Kuznetsov, O., Frontoni, E., Kuznetsova, K., Arnesano, M., Smirnov, O. «A secure biometric authentication architecture for blockchain-driven cyber-physical systems». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 193–224.

9. Kuznetsov, O., Atzeni, G., Arnesano, M., Randieri, C., Smirnov, O. «Secure IoT-based smart wheelchair system: From implementation to security enhancement strategy». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 225–257.

10. Kuznetsov, O., Smirnov, O., Kuznetsova, T., Shaikhanova, A., Svatowsky, I. «Privacy-utility trade-offs in IoT networks: A comparative analysis of differential privacy mechanisms for sensor data aggregation». *Security and Privacy of Cyber Physical Systems Emerging Trends Technologies and Applications*, 2025, pp. 589–622.

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

12. Kuznetsov O., Frontoni E., Kuznetsova Y., Smirnov O., Moskovchenko I. «Trust-Based Security Architecture for Edge Computing: A Simulation Study of Dynamic Trust Evolution and Attack Detection». *CEUR Workshop Proceedings,* 2024, 3909, pp. 227–241.

13. Kuznetsov, O., Frontoni, E., Kryvinska, N., Chevardin, V., Smirnov, O. «Wireless Network Encryption Stream Ciphers, Computational Modeling, and Security Analysis». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 379–402.

14. Kuznetsov, O., Frontoni, E., Kryvinska, N., Smirnov, O., Imoize, G.L. «Computational Modeling of Enhanced Spread Spectrum Codes for Asynchronous Wireless Communication». *Computational Modeling and Simulation of Advanced Wireless Communication Systems*, 2024, pp. 403–447.

15. Ткаченко, О., Ільєнко, А., Улічев, О., Мелешко, Є., Смірнов, О. «Правові засади поширення інформаційних впливів в соціальних мережах».

Електронне фахове наукове видання «Кібербезпека: освіта, наука, техніка», 2024. № 2(26), С. 170–188.

16. 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.

17. Вінтенко, Б., Миронець, І., Смірнов, О., Кравчук, О., Козірова, Н., Савеленко, Г., Коваленко, А. «Дослідження вимог та аналіз кібербезпеки програмного забезпечення інформаційно-керуючих систем АЕС, важливих для безпеки». *Кібербезпека: освіта, наука, техніка.* 2024. №3(23), С. 111-131.

18. 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.

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

20. Akhalaia, G., Iavich, M., Iashvili, G., Prysiazhnyy, D., Smirnova, T. «Secure Encrypted Connection on Georgian Website». *CEUR Workshop Proceedings,* 2023, 3550, pp. 313-320.

21. 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

22. 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.

23. Kuznetsov, O., Kandiy, S., Frontoni, E., Smirnov, O. «Trade-offs in Post-Quantum Cryptography: A Comparative Assessment of BIKE, HQC, and Classic McEliece». *CEUR Workshop Proceedings,* Volume 3504, 2023, pp. 1-11.

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

25. Смірнова Т.В., Гнатюк С.О., Бердибаєв Р.Ш., Сидоренко В.М., Жигаревич О.К., «Система корелювання подій та управління інцидентами кібербезпеки на об'єктах критичної інфраструктури». *Кібербезпека: освіта, наука, техніка*, №*3*(19), 2023, С. 176-196.

26. Смірнов О.А. Козлов Я.О., Смірнова Т.В. «Дослідження застосування SIEM-систем для забезпечення кібербезпеки та захисту інформації». *II Міжнародна науково-практична Інтернет-конференція «Інновації та перспективні шляхи hозвитку інформаційних технологій (ІПШРІТ-2023)»* м.Черкаси 6 грудня 2023 року – Черкаси: ЧДТУ.– 2023. – С.251-252.

27. Козлов Я.О., Смірнова Т.В., Смірнов О.А. «Дослідження SIEM-систем для забезпечення кібербезпеки». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 26.

28. Козлов Я.О., Козірова Н.Л., Смірнов О.А. «Дослідження структури та принципу роботи SIEM-системи». *VII міжнародна науково-практична конференція "Інформаційна безпека та комп'ютерні технології" до 30-ти річчя кафедри кібербезпеки та програмного забезпечення,* м. Кропивницький. 1 листопада 2023 р. – Кропивницький: ЦНТУ. – 2023. – С. 59.

29. Вінтенко Б.Ю., Смірнов О.А., Коваленко О.В., Смірнов С.А., Коваленко А.С. «Дослідження нормативних документів та галузевих стандартів розробки програмного забезпечення комп'ютерних систем управління АЕС,

важливих для безпеки». *Системи управління, навігації та зв'язку*, 2023, вип. 2(72), С. 170-178.

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

31. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. Springer, Cham. 2022, pp. 2463-2477.

32. 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.

33. 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.

34. 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.

35. 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.

36. Smirnov O., Kuznetsov A., Zhora V., Onikiychuk A., Pieshkova O. «Hiding Messages in Audio Files Using Direct Spread Spectrum». *11th IEEE*

International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021, Cracow, Poland, 22-25 September 2021. P. 414-418

37. Smirnov O., Kuznetsov A., Lokotkova I., Kuznetsova T., Florov S., Lebid O. «Using Orthogonal Signals to Hide Information in Images». *4 IEEE International Conference on Advanced Information and Communication Technologies (AICT) – 2021*, Lviv, Ukraine, September 21-25, 2021. P. 255-260.

38. Smirnov O., Kuznetsov A., Girzheva O., Kiian A., Nakisko O., Kuznetsova T. «Advanced Code-Based Electronic Digital Signature Scheme». *2020 IEEE International Conference on Problems of Infocommunications Science and Technology, PIC S and T 2020,* Kharkiv, 6 October 2020-9 October 2020, P. 358-362.

39. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.*

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

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

42. 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.

43. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131.

44. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14.

45. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. Springer, Cham. 2021, pp 66-84.

46. Smirnov O., Kuznetsov A., Onikiychuk A., Makushenko T., Anisimova O., Arischenko A. «Adaptive pseudo-random sequence generation for spread spectrum image steganography». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT)*, Ukraine, Kyiv, May 14-18. 2020. P. 161-165.

47. Smirnov O., Kuznetsov A., Kiian A., Babenko V., Perevozova I.,Chepurko I. «New Approach to the Implementation of Post-Quantum Digital Signature Scheme». *2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT),* Ukraine, Kyiv, May 14-18. 2020. P. 166-171.

48. 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.

49. 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.

50. 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.