ЗАХИСТ'ІНФОРМАЦІЇ'2025

матеріали
науково-практичного симпозіуму

2025


ЗАХІДНОУКРАЇНСЬКИЙНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КІБЕРБЕЗПЕКИ
ГРОМАДСЬКА ОРГАНІАЦІЯ «КІБЕРБЕЗПЕКА І АВТОМАТИЗАЦІЯ»

Матеріали
науково-практичного симпозіуму

"ЗАХИСТ'ІНФОРМАЦІЇ'2025"

28 листопада 2025
Тернопіль

Збірник матеріалів науково-практичного симпозіуму «Захист інформації'2025», Тернопіль, 2025. – 118с.

Адреса редакції:
Громадська організація «Кібербезпека і автоматизація»
м. Тернопіль
Контактний телефон: (066)043-42-10
e-mail: conferencekb@gmail.com

ВСТУП

Даний збірник праць представляє результати науково-практичного симпозіуму "ЗАХИСТ'ІНФОРМАЦІЇ'2025", *що відбувся 28 листопада 2025 року в рамках Міжнародного дня захисту інформації. Захід об'єднує викладачів, дослідників, студентів, представників IT-індустрії та всіх, хто працює над формуванням безпечного цифрового середовища. Проведення заходу саме в цей день підкреслює важливість історичного контексту становлення комп'ютерної безпеки та необхідність підвищення рівня обізнаності щодо сучасних кіберзагроз.*

Мета симпозіуму *- актуалізувати питання інформаційної безпеки, сприяти поширенню сучасних методів протидії кіберзагрозам, формувати культуру відповідального ставлення до обробки та збереження даних у цифровому середовищі. Захід створює платформу для професійного діалогу, обміну досвідом, представлення нових досліджень та прикладних рішень у сфері захисту інформації.*

Залишайся на крок попереду кіберзагроз!

У межах симпозіуму представлено наукові доповіді, практичні напрацювання, огляди сучасних інструментів кіберзахисту та результати досліджень, спрямованих на підвищення стійкості інформаційних систем. Подія покликана не лише інформувати, а й мотивувати до впровадження технологічних та організаційних заходів безпеки, що є критично важливими в умовах зростання кількості атак і складності інформаційних інфраструктур.

Науково-практичний симпозіум "ЗАХИСТ'ІНФОРМАЦІЇ'2025"- *це простір для професійного розвитку та пошуку рішень, необхідних для побудови надійного й безпечного цифрового майбутнього.*

ЗМІСТ

УДК 681.32

Іван АЛБАНСЬКИЙ, Руслан ГАРЛІЦЬКИЙ, Назар КАЧАЛУБА, Валерій ПАВЛІН, Михайло-Сергій ГОРОХІВСЬКИЙ, Володимир КИБА

Західноукраїнський національний університет

ОСОБЛИВОСТІ РОБОТИ АВТОМАТИЗОВАНИХ СИСТЕМ БЕЗПЕКИ НА ПРОМИСЛОВОМУ УСТАТКУВАННІ ТА РОЛЬ КОНТРОЛЕРІВ БЕЗПЕКИ

Вступ. Сучасне промислове обладнання характеризується підвищеною продуктивністю, високими швидкостями обертання та значними енергетичними потоками. Це збільшує вимоги до функціональної безпеки (Functional Safety). Задля запобігання аваріям, травмам персоналу та пошкодженню устаткування застосовуються автоматизовані системи безпеки, побудовані на основі контролерів безпеки (Safety PLC), сертифікованих відповідно до стандартів IEC 61508, IEC 62061, ISO 13849-1/2. Такі системи виконують моніторинг небезпечних зон, здійснюють аварійне вимкнення, контролюють логіку безпечної зупинки та гарантують виконання вимог SIL/PL у складних промислових процесах.

Промислове виробництво залишається одним із секторів з найбільшою кількістю серйозних професійних травм та нещасних випадків у світі. Щорічно від робочих нещасних випадків й професійних хвороб страждають мільйони людей, це підкреслює нагальну потребу у системних заходах з профілактики та контролю ризиків. Автоматизовані системи безпеки (АСБ) на устаткуванні - не тільки засіб виконання нормативних вимог, але й ключовий інструмент зниження ризиків, мінімізації простоїв і збереження репутації підприємства.

Мета роботи дослідження особливостей роботи автоматизованих систем безпеки на промисловому устаткуванні та роль контролерів безпеки.

1. Аналіз складової автоматизованої системи безпеки промислового устаткування

Сучасна АСБ - це інтегрована сукупність сенсорів безпеки (E-STOP, світлові завіси, лазерні сканери, вимикачі блокувань), логіки безпеки (контролери Safety-PLC або спеціалізовані модулі), виконавчих пристроїв (контактори, релейні модулі, приводи з функціями STO/SS) і каналів зв’язку (фізичні лінії, безпечні протоколи PROFIsafe/CIP Safety) [1]. Така архітектура дає змогу автоматично виявляти небезпеку, виконувати керований вихід у безпечний стан і вести діагностику з метою мінімізації помилок оператора.

Проєктування і верифікація систем безпеки у багатьох галузях базуються на міжнародних стандартах (IEC 61508, IEC 62061, ISO 13849), які визначають підходи до розрахунку рівнів функціональної безпеки (SIL, PL), методи оцінки ризику й вимоги до архітектури. Контролери безпеки (Safety-PLC) реалізують логіку, що дозволяє досягнути необхідного Performance Level (PL) або Safety Integrity Level (SIL), виконуючи при цьому самодіагностику, дуплексні (або подвійноканальні) перевірки та контроль «зворотних» контактів виконавчих елементів – усе це критично для підтвердження надійності захисних функцій.

Промислові автоматизовані системи безпеки включають сенсори,

контролери, виконавчі елементи та комунікаційні засоби. Узагальнена структура типової системи наведена на рисунку 1.

Сенсорами безпеки на промисловому устаткуванні є: кінцеві вимикачі позиційні (interlock switches), світлові бар'єри та завіси, лазерні сканери, дворукі пульти активації, кнопки аварійної зупинки (E-STOP Category 0/1), RFID-замки безпеки. Вище згадане обладнання (сенсори) призначене для контролю параметрів безпеки в процесі експлуатації промислових установок та запобігання травмуванню персоналу.

Контролери безпеки (Safety PLC) виконують: обробку сигналів від сенсорів, контроль алгоритмів безпеки, формування сигналів на відключення приводу, діагностику і самоконтроль. На сьогоднішній день на світовому ринку промислового обладнання присутній широкий спектр контролерів безпеки різних світових виробників.

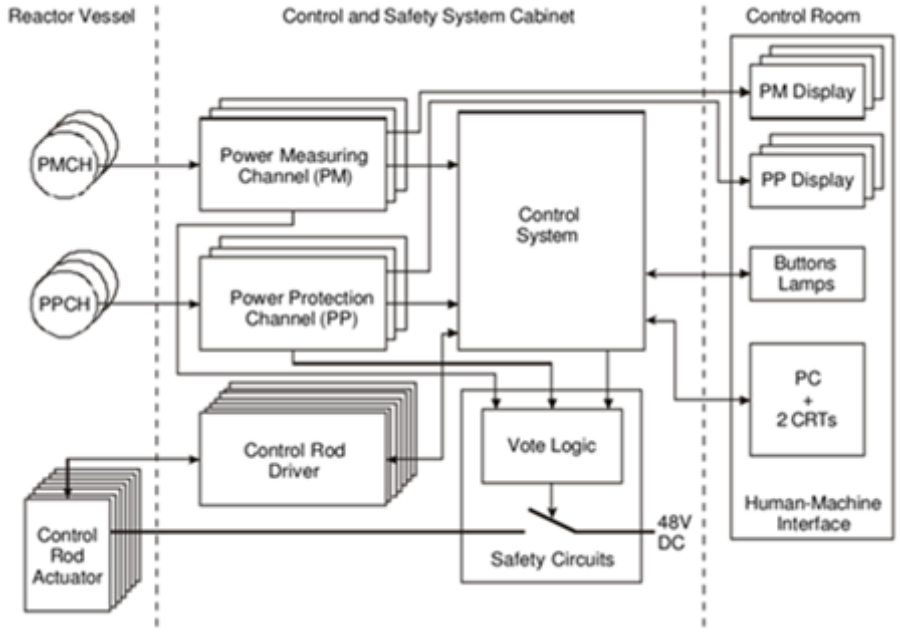


Рисунок 1 – Загальна структура промислової автоматизованої системи безпеки

Виконавчі елементи АСБ це є: контактні групи силового вимикання, муфти вимкнення силового моменту, силові контактори (з контрольними зворотними контактами), приводи з функціями безпечної зупинки (STO, SS1, SS2).

Конкретні переваги впровадження АСБ з контролерами безпеки є умовою виконання міжнародних стандартів безпеки експлуатації промислового устаткування. Зниження кількості травм і смертельних випадків на виробництві основна умова високоефективної діяльності сучасного підприємства. Автоматичне виявлення небезпеки й миттєве приведення обладнання у безпечний стан значно зменшує ймовірність серйозних інцидентів у порівнянні з ручними підходами. Узагальнену статистику глобальних робочих ризиків на промислово-економічних об'єктах веде міжнародна організація International Labour Organization.

Скорочення часу простою та швидке відновлення роботи одна з основних вимог до сучасних АСБ. Сучасні Safety-PLC надають докладну діагностику

причини стопу (який сенсор, яка помилка каналу), що прискорює локалізацію й усунення несправності [2].

Юридична та нормативна захищеність дає можливість поглиблювати адаптацію АСБ у всі аспекти технологічних процесів промислового устаткування. Виконання вимог стандартів та використання сертифікованих контролерів допомагає підприємству пройти інспекції, уникнути штрафів і юридичної відповідальності. Підтримка Industry 4.0 та безпечної цифровізації переводить те чи інше виробниче підприємство на новий етап автоматизаціх технологічних процесів. Інтегровані Safety-PLC все частіше взаємодіють з IIoT-інфраструктурою для централізованого моніторингу, аналізу даних та дистанційного управління процесами. Це дозволяє поєднувати безпеку та аналітику (наприклад, виявлення аномалій у поведінці обладнання) і є частиною концепції «Safety 4.0».

2. Структура та алгоритм роботи контролерів безпеки

Контролери безпеки, принципи та особливості роботи на пряму залежать від рівня складності архітектури та рівня безпекових вимог сфери застосування (рисунок 2). Контролер безпеки - це спеціалізований програмований логічний контролер, призначений для критично важливих функцій. На відміну від звичайних ПЛК, він має подвійну архітектуру (Redundant architecture). Так звана подвійна архітектура включає в себе певний набір функціоналу та компонентів, а саме (рисунок 3) [3]:

– два процесори працюють паралельно;
– кожен цикл виконання програми порівнюється;
– дані маршрутизуються через незалежні канали;
– у разі розбіжності контролер переходить у безпечний стан.

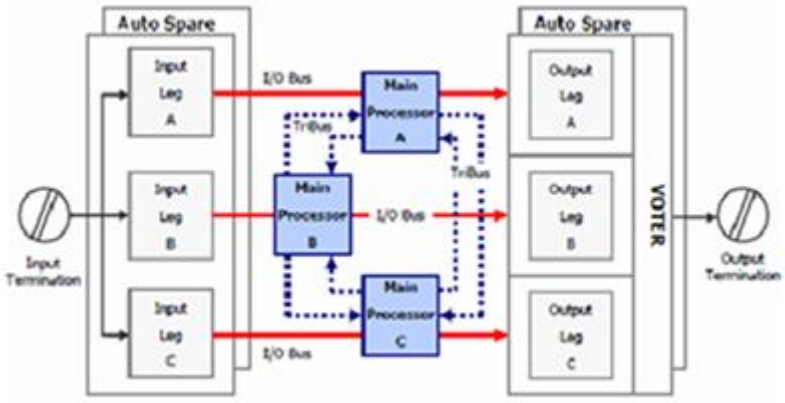


Рисунок 2 – Архітектура контролера безпеки

Технічною особливістю контролерів безпеки (звичайний PLC не годяться) є наявність складної архітектури орієнтованої під виконання вимог техніки безпеки на тому чи іншому промисловому устаткуванні. Особливостями побудови безпекових контролерів є [4]:

– архітектура із самоконтролем - апаратні й програмні механізми визначення розбіжностей між каналами у разі невідповідності – перехід в безпечний стан;
– жорсткі вимоги до часу циклу і детектування помилок, це

гарантовані обмеження затримок, необхідні для виконання критичних функцій;

– сертифіковані бібліотеки функцій безпеки - перевірені програмні блоки (E-STOP, Two-hand control, Muting), що знижують ризик помилки проектування;

– запис подій і трасування - журналування всіх подій безпеки для аудитів і розслідувань.

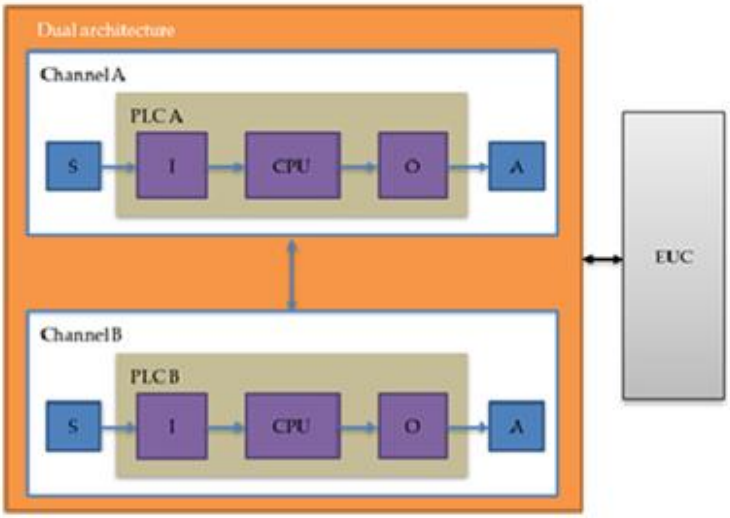


Рисунок 3 – Подвійна архітектура безпекових контролерів

Вище згадані характеристики дають змогу реалізувати PL d/e або SIL2/3 залежно від вимог предметної області. Наявність циклів самодіагностики забезпечує такому контролеру високу ефективність та надійність в експлуатації на відповідальному устаткуванню. Контролер в процесі самодіагностики перевіряє:

– цілісність пам’яті;
– коректність внутрішніх сигналів;
– напругу живлення;
– працездатність входів/виходів;
– цілісність програми безпеки;
– коректність зворотних контактів контакторів.

Наявність окремої пам’ять для програми безпеки підкреслює особливості складної архітектури таких контролерів, що є умовою застосування у складних технологічних процесах виробничої діяльності обладнання. Програма зберігається у незмінюваних сегментах пам’яті, що унеможливлює випадкову модифікацію. Сертифікація під стандарти SIL / PL підкреслює наявність SIL1–SIL4 (IEC 61508) та PL a–e (ISO 13849-1).

Структури підключення засобів безпеки включають застосування органів управління (різнотипних кнопок) та сенсорів. Один із способів підключення контролера безпеки наведений на рисунку 4. Наявний подвійний канал безпеки (Dual-Channel Safety) збільшує функціональні можливості контролера безпеки. Стандартна схема для E-STOP, світлових бар’єрів, замків безпеки показана на рисунку 4. Перевагами аналогічних схем підключень є: виявлення залипання контактів, зниження ймовірності відмови, відповідність PL d/e або SIL2/3.

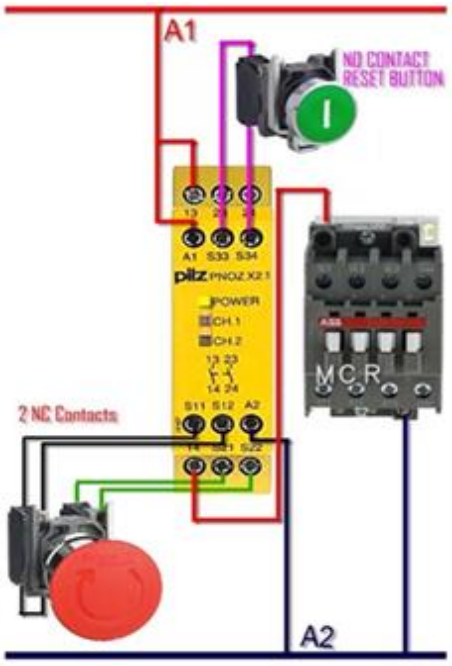


Рисунок 4 – Підключення контролера безпеки

Алгоритми роботи АСБ дають можливість коректно функціонувати складному технологічному обладнанні на безпечному рівні. Алгоритм безпечної зупинки приводу SS1 → STO, а також інших функціональних пристроїв включених в одну схему роботи з контролером безпеки, забезпечує виконання внутрішньої програми функціонування контролера. Алгоритм SS1 використовується, коли потрібно сповільнити механізм, привести його до контрольованої зупинки, лише після цього активувати STO (Safe Torque Off). Алгоритм роботи включає:

– виявлення небезпеки (сенсор зони, E-STOP, сканер);
– передача сигналу до Safety PLC;
– контролер надсилає команду SS1 приводу;
– привід виконує кероване гальмування (зниження частоти, контроль струму);
– після повної зупинки включається STO – повне знеструмлення силових ключів.

Логіка взаємоблокування між різними безпековими зонами дає можливість контролю обладнання що працює в даних зонах перебування персоналу (оператори, наладчики). У складному обладнанні зона 1 та зона 2 можуть впливати одна на одну через Safety PLC. Наглядним прикладом буде вхід персоналу в зону 2 → Зона 1 переводиться у режим Safe Limited Speed (SLS). Потрапляння в спільну небезпечну зону → STO для всіх приводів і робочого обладнання.

3. Сучасні тенденції та ризики впровадження АСБ на малі та середні промислові об’єкти.

Сучасні виклики й тренди, що підсилюють актуальність АСБ у різних сферах промисловості задають нові етапи формування безпеки виробничої

діяльності. Актуальними тенденціями до глибокого впровадження на всіх рівнях експлуатації різнотипного обладнання де задіяний персонал є [5]:

– зростання складності виробничих ліній (роботи швидші, роботизовані клітки зборки, колаборативні роботи) – це підвищує ризики взаємодії людина-машина;

– цифровізація та IoT – більше підключених сенсорів і вузлів, що створює як можливості (аналіз, дистанційна діагностика), так і нові вектори ризику (кібербезпека);

– системи безпеки повинні інтегруватись у мережу без втрати гарантій функціональної безпеки;

– економічний тиск і ринок Safety-PLC - ріст попиту на рішення для функціональної безпеки підтверджується збільшенням ринкових оцінок Safety-PLC, а це означає більшу доступність технологій для малого й середнього бізнесу і посилену конкуренцію постачальників;

– регуляторні очікування та міжнародні стандарти постійно уточнюються, що робить систематичну імплементацію АСБ стратегічною необхідністю для експорту продукції та роботи на міжнародних ринках.

Ризики та обмеження впровадження АСБ на сьогоднішньому етапі функціонування виробничих потужностей малих та середніх виробничих майданчиків дає можливість бачити загальну картину безпеки людинно-машинної взаємодії. Основними аспектами є:

– помилкова впевненість у автоматизованих системах - система безпеки і це не «чорний ящик», вона вимагає регулярних перевірок, тестів і процедур обслуговування;

– кіберризики - підключення Safety-PLC і сенсорів до загальної мережі без належного сегментування може створювати загрози тобто потрібно застосовувати принципи безпечної архітектури (зони, огородження, захищені протоколи);

– економічний бар’єр, хоча ринок росте, первинні інвестиції та навчання персоналу можуть бути відчутними для менших підприємств;

– складність сертифікації й валідації - реалізація системи унеможливлюється без ретельної оцінки ризику та документування.

Переваги використання сучасних контролерів безпеки на виробничих майданчиках усіх рівнів є запорукою безпеки обслуговуючого персоналу та операторів. До таких переваг відносять – гнучкіст програмування, розширена діагностика, масштабованість, мінімізація простоїв. Гнучкість програмування дозволяє застосовувати логіку безпеки у вигляді функціональних блоків: Emergency Stop, Two-Hand Control, Guard Monitoring, Muting / Blanking, STO / SS1 блоки. Розширена діагностика включає: журнал подій, контроль залипання контактів, визначення хибних спрацьовувань, передавання даних через PROFIsafe, CIP Safety, Safety over EtherCAT. Масштабованість - одностанційна машина → велика автоматизована лінія. Мінімізація простоїв це коли оператори бачать: причину аварії, тип сенсора, часову мітку, локалізацію.

Практичними рекомендаціями для розробників проектів і технічного персоналу є ряд пунктів, що висвітлюють передумови інтеграції АСБ у

виробництво, а це:

– розпочинати з оцінки ризику (Risk Assessment) – визначити, які функції потребують PL/SIL і в якому обсязі;

– вибирати сертифіковані контролери та перевірені компоненти – це скорочує час на валідацію;

– проєктувати подвійну архітектуру критичних функцій - розподілити критичні входи/виходи між каналами, контролювати зворотні контакти;

– інтегрувати безпеку й кіберзахист - сегментація мережі, використання безпечних протоколів, оновлення прошивок;

– план сервісного обслуговування - періодичні тести, журнати, навчання персоналу;

– моніторинг KPI безпеки - час простою, кількість інцидентів, частота хибних спрацювань – для економічного обґрунтування інвестицій.

Висновок. Автоматизовані системи безпеки на промисловому обладнанні є необхідним елементом сучасного виробництва. Основу таких систем становлять контролери безпеки**,** що забезпечують надійність функціонування, відповідність міжнародним стандартам, мінімізацію ризиків аварій, ефективну інтеграцію з системами керування.

Завдяки використанню подвійних каналів, самодіагностики, алгоритмів SS1/STO та структурованого підключення сенсорів системи забезпечують високий рівень SIL/PL та підвищують загальну безпеку виробничого процесу.

Актуальність автоматизованих систем безпеки на промисловому устаткуванні сьогодні висока й продовжить зростати під впливом цифровізації, зростання вимог до охорони праці та розвитку складних автоматизованих ліній. Контролери безпеки (Safety-PLC) відіграють центральну роль: вони перетворюють нормативні вимоги і оцінку ризику на реальні, відтворювані функції, здатні гарантувати необхідний рівень захисту. Інвестиції у повну архітектуру безпеки – це одночасно інвестиції у збереження життя працівників, стабільність виробництва й довгострокову економію коштів.

Перелік використаних джерел.

1. Буров Є.В. Автоматизація технологічних процесів та виробництв: навчальний посібник. КПІ ім. Ігоря Сікорського, Київ, 2018. - 272 с.

2. Говорущенко Т.Є. Основи технічної безпеки промислових об’єктів. Національний технічний університет «ХПІ», Харків, 2016. - 198 с.

3. Дробін О.Й., Лисий А.М., Мельник В.Л. Охорона праці та промислова безпека: навчальний посібник. Львівська політехніка, 2019. - 360 с.

4. Кондратюк В. І., Кваша Т. В. Надійність і діагностика автоматизованих систем управління. Кондратюк В. І., Кваша Т. В.; видавництво: НТУУ «КПІ», Київ, 2017. - 184 с.

5. Пшеничний В.В., Кравченко Б.Ф. Функціональна безпека автоматизованих систем: методичні рекомендації. Одеський національний політехнічний університет, 2020. - 96 с.

УДК 004.056.5

Валентин БЕВЗ[1], Степан ІВАСЬЄВ[1], Любов МЕЛЕНЧУК[2]

[1]Західноукраїнський національний університет
[2]Галицький фаховий коледж імені Вячеслава Чорновола

БЕЗПЕКА MICROSOFT OFFICE: ОБ'ЄКТИ, ЩО ВБУДОВУЮТЬСЯ

Вступ. Вбудовані об'єкти в Microsoft Office (Object Linking and Embedding, OLE) дозволяють інтегрувати дані з інших додатків, таких як Excel у Word або PowerPoint у Outlook. Попри зручність, ця функціональність несе серйозні ризики для безпеки, оскільки зловмисники можуть вставляти шкідливі об'єкти, які виконують код при відкритті документа. Часто такі об'єкти використовуються в фішингових атаках або для поширення експлойтів без необхідності взаємодії користувача.

Для захисту від подібних загроз рекомендується вимикати автоматичне виконання активного вмісту, обмежувати використання макросів, а також застосовувати політики груп (GPO) для контролю поведінки OLE–об'єктів.

Мета дослідження є вивчення механізмів вбудовування об'єктів (OLE) у середовищі Microsoft Office, аналіз пов’язаних із ними загроз інформаційній безпеці, а також розробка рекомендацій щодо запобігання експлуатації цих механізмів у зловмисних цілях.

1. Аналіз сучасних загроз офісним застосункам MS Office

Спочатку архітектура Microsoft Office будувалася на основі концепції складових документів, вони ж документи OLE, активно просувається Microsoft на зорі 32–розрядних Windows.

Універсальний спосіб додавання до документів даних (і коду обробки цих даних) став універсальним шляхом появи у продукті вразливостей, який і сьогодні постійно підносить приємні сюрпризи творцям malware та дослідникам безпеки.

Згодом програми пакета отримали досить багатий набір інструментів для додавання в документи зображень, графіків і схем, елементів, що управляють, які створюються і обробляються самим додатком і є його частиною. З точки зору безпеки ці елементи становлять дещо менший інтерес, ніж елементи, про які йтиметься – елементи, які використовують код зовнішніх додатків, які додаються до документів за допомогою OLE.

"Дисковим" поданням складеного документа є CFBF файл. Розглянемо вбудовування об'єктів у документи Microsoft Office (а точніше, лише аспект безпеки) у контексті даних та коду, завантажених під час виконання.

Об'єкти, що формально вбудовуються в документи Microsoft Office, можна розділити на такі групи:

- Керуючі елементи ActiveX (ActiveX Controls).
- Впроваджені елементи даних OLE (OLE Embedded Objects).
- Впроваджувані файли (Packages).
- Вбудовані елементи не–OLE.

2. Керуючі елементи ActiveX

Елементи керування ActiveX можна як елементи вікна програми – скажімо, кнопки, перемикачі, списки, поля введення та інші форми – покликані робити деякі події чи відповідати.

Вбудовані в вебсторінки ActiveX обєкти створювали загрозу в безпеці Internet Explorer, заходи безпеки з часом посилювалися. Браузери інших виробників практично відразу відмовилися від підтримки ActiveX. Новий браузер Microsoft Edge остаточно розлучився із цим пережитком минулого. Проте вбудовування в документи Office все ще можливе.

ActiveX у документах призначені для використання у зв'язку з Visual Basic for Applications. Тим не менш, для їх завантаження та активації VBA не потрібно, а для завантаження елементів з білого списку не потрібен і дозвіл користувача.

Вразливості в останніх особливо небезпечні – налаштування за замовчуванням, що задаються при встановленні програми, не передбачають жодного захисту від завантаження цих елементів, ні попередження користувача. Адміністратору необхідно примусово посилити налаштування, заборонивши завантаження будь–яких елементів ActiveX (зазначимо, що в режимі безпечного перегляду ActiveX не завантажуються).

Однією з найнебезпечніших уразливостей у документах Office у 2012 році була CVE–2012–0158. Код завантаження елемента Microsoft ListView Control 6.0 із бібліотеки MSCOMCTL.OCX містив можливість переповнення буфера, що дозволяло підмінити адресу повернення та виконати довільний код. Оскільки елемент перебував у «білому списку» ActiveX, завантаження починалося відразу при відкритті документа. Наразі вразливість усунена, елемент ListView Control, як і раніше, вважається «безпечним».

Для додавання елемента, що управляє, в документ Microsoft Office (для простоти візьмемо Word) за допомогою інтерфейсу користувача необхідно відкрити вкладку «Розробник» (її видимість налаштовується в меню «Параметри Word») і вибрати Елементи керування –> Інструменти з попередніх версій –> елементи ActiveX (рисунок 1). Меню продемонструє набір значків, які відповідають елементам Microsoft Forms, а також можливість вибрати ActiveX зі списку, складеного з наявних у системі елементів, відібраних за низкою критеріїв.

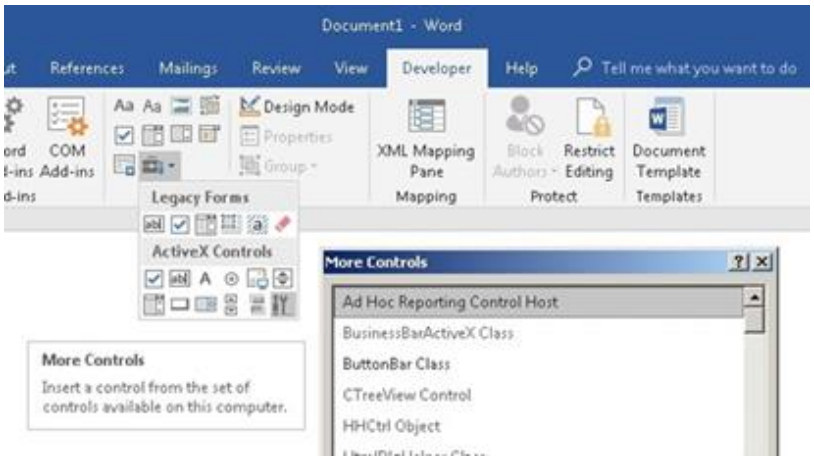


Рисунок 1 – Список ActiveX об’єктів

Список не відповідає набору елементів, які дійсно можуть бути завантажені в документ, тому на нього не можна орієнтуватися при пошуку вразливих елементів. Складна багаторівнева перевірка завантажуваних ActiveX має кілька етапів, відрізняється для версій Office і змінюється від оновлення до оновлення, так що найбільш вірний спосіб перевірити можливість завантаження – "вручну" скомпонувати файл документа з елементом, що цікавить, і спробувати відкрити його в Office. Можливі формати документів наведено нижче.

Кожен елемент ActiveX є об'єктом одного з класів COM, що відповідають певним вимогам. Завантаження елемента відбувається за допомогою підсистеми COM, а код, що виконується, міститься в одному з модулів, як правило, «зовнішніх» по відношенню до додатку–контейнеру. Як і будь–який COM–об'єкт, елемент ActiveX може бути реалізований у вигляді бібліотеки DLL, або у вигляді виконуваного EXE–файлу. У першому випадку бібліотека буде завантажена в адресний простір контейнера, у другому елемент буде оброблятися в окремому процесі, з передачею даних між контейнером і об'єктом за допомогою COM–маршалінгу. Як і будь–який об'єкт COM, ActiveX має інтерфейси, властивості та методи.

Інтерфейси – це насамперед набір стандартних інтерфейсів, які повинен мати клас ActiveX для повноцінного завантаження та взаємодії з контейнером, зокрема, IOleControl та IOleObject. Відсутність якихось необхідних інтерфейсів може скоротити функціональність елемента або перервати його завантаження на якомусь етапі.

Вразливість CVE–2015–2424 була пов'язана з елементом TaskSymbol Class із бібліотеки mmcndmgr.dll. Елемент не був призначений для використання в документах і не експортував інтерфейс IDispatch. У процесі завантаження елемента процедура, що запросила цей інтерфейс, отримувала помилку і руйнувала внутрішню структуру елемента, що призводило до вразливості типу use–after–free. На даний момент елемент заборонено до завантаження (незважаючи на це, його все ще можна виявити у списку для додавання меню «Розробник»). Сама вразливість не усунена.

Крім стандартних, кожен клас ActiveX експортує "основний" інтерфейс, що представляє його власну унікальну функціональність. Наприклад, для класу Forms.CommandButton.1 це ICommandButton. Переглядати інтерфейси ActiveX можна за допомогою інструмента OleView, що входить до Microsoft Visual Studio(рисунок 2).

Інтерфейс елемента визначає його Методи та Властивості. Властивості представляють деякі дані, що визначають вид та роботу елемента. Розробник ActiveX–елемента надає кожній властивості певне ім'я, скажімо BackColor або GridLineWidth, і тип, наприклад, рядок, ціле або речове подвійної точності. Для растрових зображень та значків існує такий тип властивості, як картинка. Клієнтська програма може встановлювати окремі властивості елемента керування, задаючи цілочисленні індекси і значення.

З погляду низькорівневої реалізації розподіл на методи та властивості формальний, оскільки «властивості» представлені набором методів get/set. Однак є і значуща відмінність: Методи елемента (його основного інтерфейсу) можуть

бути викликані тільки програмно, у разі документів Office – тільки з програми VBA, що виконується. З точки зору безпеки це не має великого інтересу, оскільки виконання VBA це вже компрометація операційної системи. Властивості зберігаються в документі і при його відкритті будуть оброблені і завантажені в структури в пам'яті навіть якщо виконання VBA заборонено.

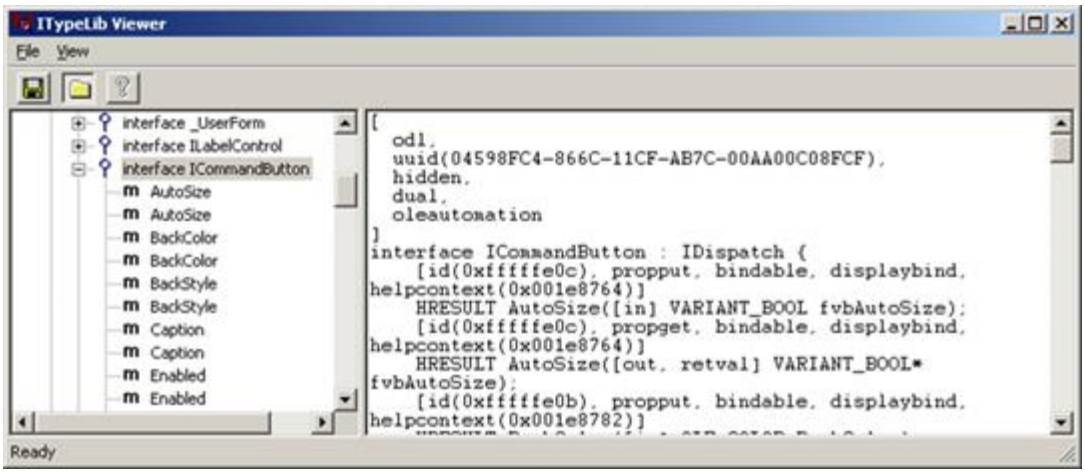


Рисунок 2 – Перегляд інтерфейсів ActiveX

З програмної точки зору, з боку елемента для збереження його властивостей та стану в документі контейнер надає інтерфейси IStream, IStorage та IPropertyBag. Їх реалізація та подання даних у дисковому файлі вже не піклування елемента ActiveX, і повністю залежить від контейнера та формату документа. Слід зазначити, що набір і формат даних, що зберігаються, може відповідати «публічно» експортованому набору властивостей, а може бути й зовсім іншим. Розглянемо приклади реалізації, які стосуються Microsoft Office.

Складений файл (compound file, CFBF). Застарілий формат документів Office, де для зберігання даних ActiveX виділялося сховище нижнього рівня ObjectPool та окремі підкаталоги усередині нього. Потік \\001CompObj містить ідентифікатор класу, який в кінцевому підсумку і визначає клас об'єкта, що завантажується. Заміна ідентифікатора безпосередньо в hex призведе до завантаження об'єкта зовсім іншого класу.

Інтерфейс елемента визначає його Методи та Властивості. Властивості представляють деякі дані, що визначають вид та роботу елемента. Розробник ActiveX–елемента надає кожній властивості певне ім'я, скажімо BackColor або GridLineWidth, і тип, наприклад, рядок, ціле або речове подвійної точності. Для растрових зображень та значків існує такий тип властивості, як картинка. Клієнтська програма може встановлювати окремі властивості елемента керування, задаючи цілочисленні індекси і значення.

З погляду низькорівневої реалізації розподіл на методи та властивості формальний, оскільки «властивості» представлені набором методів get/set. Однак є і значуща відмінність: Методи елемента (його основного інтерфейсу) можуть бути викликані тільки програмно, у разі документів Office – тільки з програми VBA, що виконується. З точки зору безпеки це не має великого інтересу, оскільки виконання VBA це вже компрометація операційної системи. Властивості зберігаються в документі і при його відкритті будуть оброблені і завантажені в

структури в пам'яті навіть якщо виконання VBA заборонено.

З програмної точки зору, з боку елемента для збереження його властивостей та стану в документі контейнер надає інтерфейси IStream, IStorage та IPropertyBag. Їх реалізація та подання даних у дисковому файлі вже не піклування елемента ActiveX, і повністю залежить від контейнера та формату документа. Слід зазначити, що набір і формат даних, що зберігаються, може відповідати «публічно» експортованому набору властивостей, а може бути й зовсім іншим. Розглянемо приклади реалізації, які стосуються Microsoft Office.

Застарілий формат документів Office, де для зберігання даних ActiveX виділялося сховище нижнього рівня ObjectPool та окремі підкаталоги усередині нього. Потік \\001CompObj містить ідентифікатор класу, який в кінцевому підсумку і визначає клас об'єкта, що завантажується. Заміна ідентифікатора безпосередньо в hex(рисунок 3) призведе до завантаження об'єкта зовсім іншого класу.

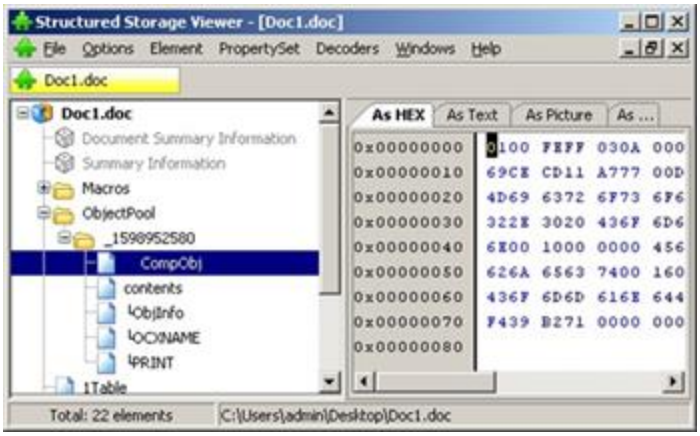


Рисунок 3 – Перегляд Structured Storage Viewer

Сучасний формат XML документів. Файл є zip–архів(рисунок 4). Дані елементів ActiveX зберігаються в підкаталозі ActiveX у файлах з хитромудрими назвами типу activeX1.xml.

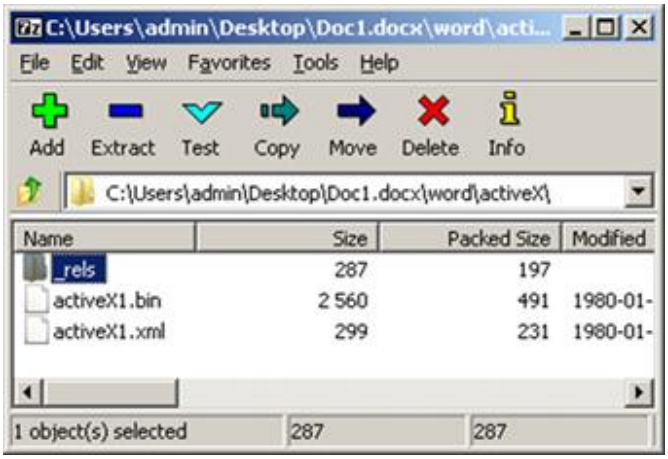


Рисунок 4 – Перегляд структури файлу за допомогою 7zip

Приклад файлу:

<?xml version="1.0" encoding="UTF–8" standalone="no"?>

<ax:ocx ax:classid="{D7053240–CE69–11CD–A777–00DD01143C57}"

ax:persistence="persistPropertyBag"
xmlns:ax="http://schemas.microsoft.com/office/2006/activeX"
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/relationships">
<ax:ocxPr ax:name="Caption" ax:value="CommandButton1"/>
<ax:ocxPr ax:name="Size" ax:value="2540;847"/>
<ax:ocxPr ax:name="FontName" ax:value="Calibri"/>
<ax:ocxPr ax:name="FontHeight" ax:value="225"/>
<ax:ocxPr ax:name="FontCharSet" ax:value="204"/>
<ax:ocxPr ax:name="FontPitchAndFamily" ax:value="2"/>
<ax:ocxPr ax:name="ParagraphAlign" ax:value="3"/>
</ax:ocx>

У цих файлах у текстовій формі вказані ідентифікатори класів. Заміна ідентифікатора також спробує завантажити елемент іншого класу.

Далі файл містить вказівку на тип зберігання даних об'єкта: persistPropertyBag, persistStorage або persistStream. Якщо елемент підтримує збереження властивостей persistPropertyBag, його дані можуть бути збережені в тому ж текстовому файлі, див. Якщо ж йому потрібне сховище або двійковий потік, дані будуть збережені у файлі з ім'ям типу activeX1.bin, що є файлом CFBF.

У документі rtf елемент ActiveX визначається тэгами \object\objocx. Тег \objdata містить сховище властивостей елемента у вигляді hex–подання файлу CFBF.

{\object\objocx\f37\objsetsize\objw1440\objh480{*\objclass
Forms.CommandButton.1}
{*\objdata 01050000020000001600000000
466f726d732e436f6d6d616e64427574746f6e2e3100000000000000000000000e0000
d0cf11e0a1b11ae1000000000000000000000000000000000000003e000300feff090006

Процес завантаження ActiveX із документа загалом досить простий. Додаток–контейнер створює чистий об'єкт зазначеного класу, запитує в нього вказаний інтерфейс зберігання, і надає покажчик на сховище, потік або пакет властивостей.

Фільтрація елементів, які можуть бути завантажені, має багато етапів. Насамперед відсіваються класи, зазначені в чорному списку, відомому як Office COM Kill Bit (підвіска реєстру *OFFICE_KEY*\Common\COM Compatibility). Наприклад, прапори, що запобігають завантаженню, мають такі класи як Microsoft Scriptlet Component та Microsoft Web Browser.

Інші класи пройдуть початкове завантаження. Це означає, що бібліотека DLL буде завантажена в програму–контейнер, або буде запущено процес COM–сервера, реалізованого в EXE–файлі. Тільки після цього будуть виконуватися інші перевірки, включаючи елементарну – а чи є об'єкт, власне представником ActiveX.

У 2015 році дослідник виявив, що попереднє завантаження модулів COM можна використовувати для обходу ASLR та виконання довільного коду за рахунок завантаження підроблених динамічних бібліотек. В описі CVE–2015–6128, що вийшов згодом, немає жодного слова про Microsoft Office.

Якщо ідентифікатор дійсно визначає ActiveX, він пройде кілька перевірок у

кількох чорно–білих списках. Список ActiveX, що завантажуються з .docx на чистих Windows 7 і Office 2016 при налаштуваннях за промовчанням.

Можна побачити, що значне місце у списку займають компоненти групи Microsoft Forms. Це набір керуючих елементів, що постачаються з Office, ви можете бачити їх на панелі «елементи ActiveX». Спочатку всі вони реєструвалися як "безпечні", але згодом з'ясувалося, що для окремих елементів це не так. Наприклад, елемент Frame завантажує будь–які інші ActiveX, не перевіряючи жодних списків (в останніх версіях це «виправлено», але власний блеклист Frame відрізняється від загального Office). Тому частина елементів Microsoft Forms може бути завантажена в документ тільки з дозволу користувача. Microsoft Forms Frame також вимагає згоди користувача (за замовчуванням), зате дозволяє завантажити деякі елементи з Kill Bit списку, які не могли б бути завантажені за інших умов.

Отже, якщо атакуючому вдається переконати користувача дозволити завантаження ActiveX, Frame допоможе йому суттєво розширити «арсенал» за рахунок таких елементів як Web Browser.

Формат зберігання властивостей Microsoft Forms частково документований специфікацією [MS–OFORMS].

У процесі сканування ActiveX з'ясувалося, що набір класів для doc, docx і rtf різний, а також різні списки доступних ActiveX для програми, запущеного звичайним чином і запущеного в режимі автоматизації.

Багато популярних програм доповнюють ці списки власними ActiveX. У разі виявлення вразливості вона буде відображена в бюлетені як така, що має відношення до додатку до складу якого входить. При цьому єдиним шляхом експлуатації вразливості можуть бути документи Office.

Приклад: Flash ActiveX особливо полюбився зловмисникам за вразливості, що стабільно виявляються, і постійне місце в «білих списках» IE і Office. Перші відомі вразливості у цьому компоненті з'явилися ще у 2008 році, одна з останніх CVE–2018–4878 закрита. Зі згасанням популярності IE документи Office стали основним шляхом поширення експлойтів для Flash.

3. Впроваджувані елементи даних OLE

Впроваджувані елементи OLE покликані реалізувати концепцію «документа в документі» з можливістю редагування «на місці» даних різних форматів, що обробляються іншими програмами. Подібно до ActiveX, OLE–документи реалізовані на основі COM.

Додати OLE–елемент до документа Word можна наступним чином: відкрити вкладку «Вставка» і вибрати Текст –> Об'єкт(рисунок 5).

Програма виведе список типів документів, для яких зареєстровані OLE–обробники. Як і у випадку з ActiveX, цей список мало відповідає набору класів, які дійсно можуть бути завантажені як документи OLE.

Як і ActiveX реалізація будь–якого OLE–документа представлена відповідним класом COM, виконаним як DLL чи EXE. Компонент експортує необхідні службові інтерфейси, а збереження стану у документі–контейнері виконується за допомогою інтерфейсів IPersist*.

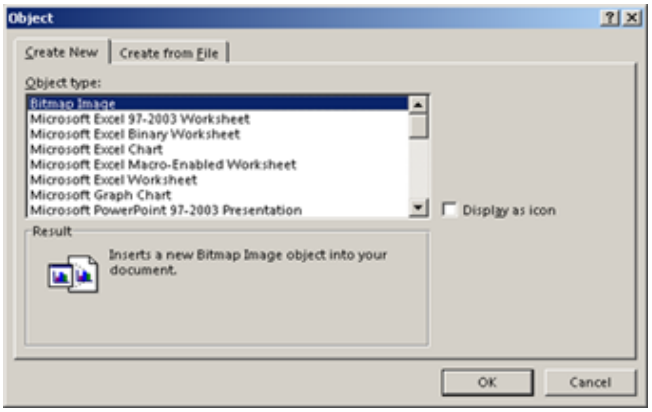


Рисунок 5 – Додавання OLE–елемента

У документі формату CFBF дані об'єктів OLE зберігаються у сховищі другого рівня ObjectPool. Набір потоків схожий на відповідний елементам ActiveX (рисунок 6).

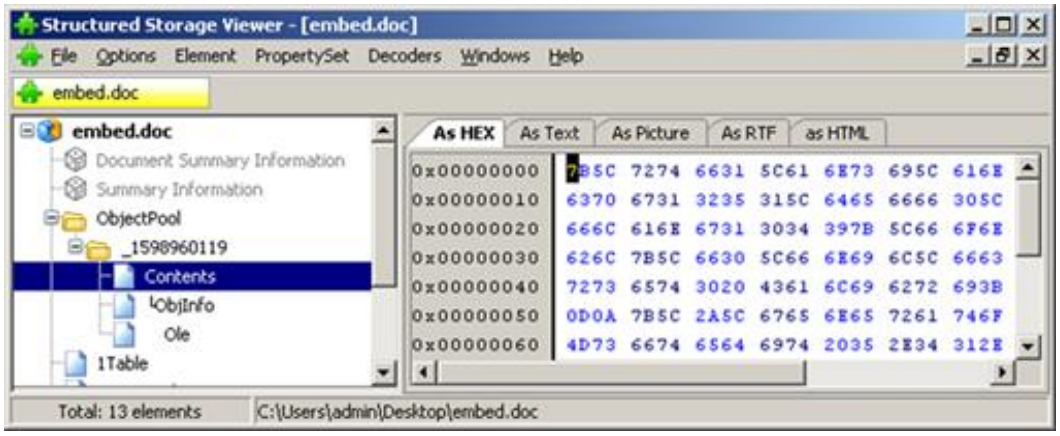


Рисунок 6 – Перегляд документу формату CFBF

У документах Open Office XML дані об'єкта OLE зберігаються у підкаталозі embeddings, у файлі–сховищі CFBF з іменем типу oleObject1.bin (рисунок 7).

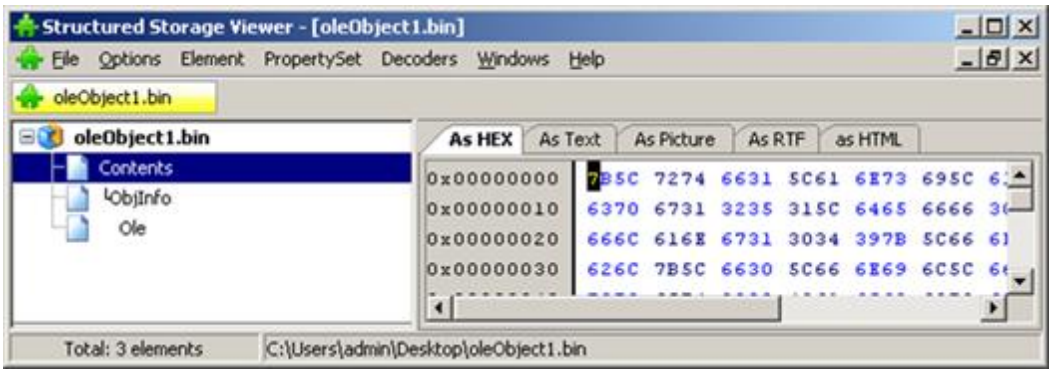


Рисунок 7 – Документ Open Office XML

У документах RTF інформація про об'єкт зберігається під тегом \object\objemb\. Розділ містить також сховище, закодоване як hex–подання файлу CFBF.

{\object\objemb\objw8307\objh553{*\objclass WordPad.Document.1}
{*\objdata
0105000002000000130000000576f72645061642e446f63756d656e742e3100000000000

000000000a0000
d0cf11e0a1b11ae100000000

Формат RTF виділяється тим, що підтримує тег \objupdate, що викликає автоматичну активацію елемента, в той час як за замовчуванням OLE–елементи неактивні при завантаженні.

Приклад: Вразливість CVE–2017–11882 OLE компонента Equation Editor завдяки обробці об'єкта в окремому процесі давала можливість стабільної та універсальної експлуатації. Тег \objupdate змушував Word завантажувати вразливий компонент відразу після відкриття документа.

Приклад: вбудовані елементи Excel з макровірусом. Дослідниками виявлено шкідливі rtf–документи, які не використовують жодних нових вразливостей. Документи містять як вбудовані об'єкти кілька документів Excel з макросом. Розрахунок зроблено те що, що користувач, змушений після відкриття документа кілька разів поспіль відмовлятися від виконання макросу, у результаті «здасться» і дозволить виконання(рисунок 8). На даний момент техніка все ще працює.

Рисунок 8 – Попередження про наявність макросів

Значна відмінність від ActiveX у випадку впроваджуваних елементів OLE полягає в тому, що ідентифікатор класу записується безпосередньо в файл сховища функцією WriteClassStg. впроваджуваного елемента, але завантажений буде об'єкт саме того класу, який вказаний у сховищі.

Можливо відредагувати дані елемента, що у певних випадках призводить до виявлення вразливостей.

Об'єкти OLE також проходять численні перевірки на можливість завантаження, що ускладнює отримання повного списку потенційно завантажуваних елементів. Набір елементів, які можуть бути завантажені як об'єкти OLE, відрізняється від списку ActiveX, що завантажуються ActiveX. Explorer\ActiveX Compatibility).

OLE розрізняє два механізми вбудовування вмісту в документ – безпосередньо вбудовування OLE–документа і створення посилання всередині основного документа на інший документ. бути зареєстрований операційній системі.

Приклад: CVE–2017–0199. Вразливість CVE–2017–0199 полягала в

можливості додавання в документ «об'єкта за посиланням» формату hta. Перш ніж оновити вбудований об'єкт, програма запитує дозвіл користувача.

4. Впроваджувані файли (Packages)

Документи Office підтримують можливість додавання будь–якого файлу (Об'єкт –> Створення з файлу або просто перетягнути іконку файлу в поле редагування). файли «за посиланням», коли відкриття файлу відбувається з власного сховища, а зазначеним шляхом, зокрема і мережному.

Останнім часом функціональність Object Packager була значно підрізана, а спочатку елемент міг зберігати будь–які файли, в тому числі посилання, і навіть командний рядок.

Приклад: Повідомлення Outlook, які також є складовими документами, дозволяють додавати елементи Object Packager у тіло листа (рисунок 9). Для користувача елемент виглядає як довільно обраний зловмисником.

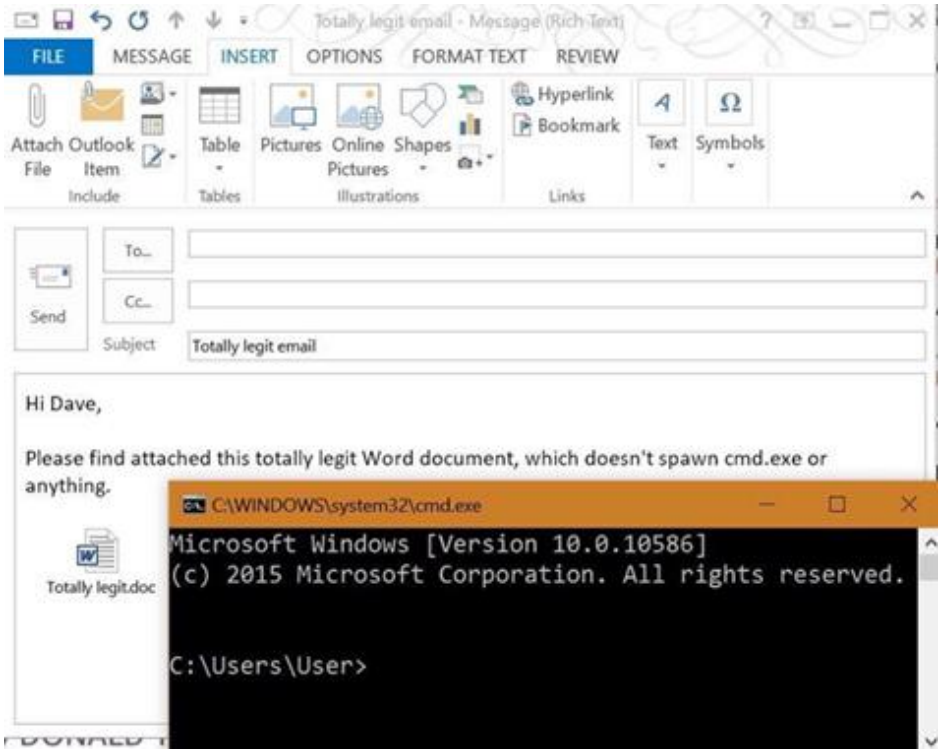


Рисунок 9 – Повідомлення Outlook з елементом Object Packager

5. Вбудовані елементи, реалізовані не за допомогою OLE

На даний момент найбільшу загрозу/інтерес із не–OLE елементів можуть представляти зображення, що додаються до документа за посиланням. При відкритті документа не в захищеному режимі зображення завантажуються автоматично, що може призводити до розкриття розташування та особистості користувача, який завантажив документ через анонімні проксі або отримав конфіденційний документ з третіх рук. Ця методика, зокрема, була реалізована в інструменті Scribbles, що знаходиться на озброєнні спецслужб США.

У локальній мережі Windows автоматичне завантаження зображень за посиланням уможливлює експлуатацію вразливості NTLMRelay. Механізм посилань на картинки не сумісний з вимогами безпеки мереж ActiveDirectory,

оскільки адміністратор, який отримує подібний документ, по суті виконує код зловмисника з повними адміністративними привілеями.

6. Методи захисту

Найдієвіший на сьогоднішній момент метод захисту від уразливостей у вбудованих у документи Office об'єктах – режим захищеного перегляду (рисунок 10).

Рисунок 10 – Повідомлення про потенційні загрози

У цьому режимі виключається як завантаження об'єктів, і завантаження даних із зовнішніх джерел. На жаль, для переходу в повнофункціональний режим потрібні елементарні дії користувача, які з легкістю провокуються методами соціальної інженерії.

Керуючі елементи ActiveX можна вимкнути у налаштуваннях Trust Center (рисунок 11).

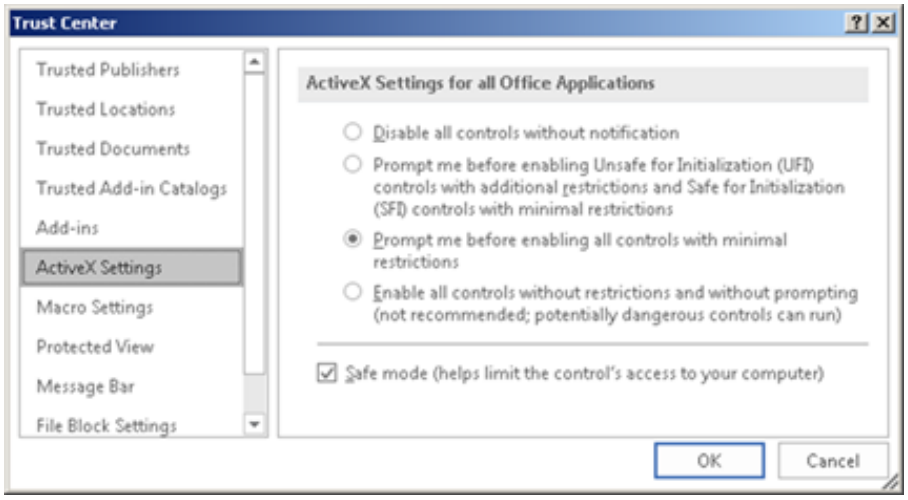


Рисунок 11 – Налаштування елементів ActiveX

Варто звернути увагу, що це не працює для вбудованих елементів OLE. Щоб вимкнути або обмежити використання вбудованих об'єктів OLE у Microsoft

Office, можна скористатися кількома методами, зокрема через налаштування Центру безпеки, редактор групових політик або реєстр Windows. Кожен із цих способів дозволяє посилити захист офісного середовища від потенційно небезпечного вмісту, який може бути вбудований у документи.У середовищі Microsoft Word, Excel або PowerPoint користувач може перейти до розділу «Файл», далі вибрати «Параметри», після чого відкрити «Центр безпеки» та «Параметри Центру безпеки». У розділі, що відповідає за активний вміст, необхідно налаштувати параметри елементів ActiveX, обравши варіанти, які блокують автоматичне виконання вбудованих елементів або повністю їх забороняють. Також варто звернути увагу на налаштування макросів, адже вбудовані OLE–об’єкти нерідко використовуються у зв’язці з ними.

Для адміністраторів корпоративних систем доцільним буде використання редактора групових політик. У цьому випадку потрібно запустити консоль групових політик, перейти до гілки конфігурації користувача, обрати відповідну версію Microsoft Office, знайти параметри безпеки та активувати політики, які забороняють вставлення або оновлення зв’язаних OLE–об’єктів. Це дозволить централізовано обмежити використання небезпечних елементів у межах організації.

Ще одним способом вимкнення OLE–об’єктів є редагування системного реєстру Windows. Для цього слід відкрити редактор реєстру, знайти відповідний розділ, що відповідає за параметри безпеки Microsoft Office, і створити новий параметр типу DWORD із назвою DisableOLEObjects, встановивши його значення рівним одиниці. Такий підхід вимагає обережності й попереднього резервного копіювання, однак він дозволяє гнучко налаштувати роботу з OLE навіть у відсутність доступу до групових політик.

Загалом, вимкнення або обмеження використання вбудованих об’єктів у Microsoft Office є ефективним кроком для зменшення ризиків, пов’язаних із виконанням шкідливого коду зсередини документів. У поєднанні з іншими заходами, такими як захищений режим перегляду чи заборона макросів, це створює надійний бар'єр проти розповсюджених типів атак.

Висновок. Вбудовані об'єкти (OLE) у Microsoft Office, попри свою функціональність, становлять суттєву загрозу для інформаційної безпеки, оскільки можуть використовуватись для прихованого запуску шкідливого коду. Аналіз показав, що найбільш поширеними сценаріями зловживання є фішингові атаки та використання документів як носіїв експлойтів. Ефективними засобами захисту є обмеження виконання активного вмісту, налаштування політик безпеки на рівні організації та регулярне оновлення програмного забезпечення. Усвідомлення цих ризиків і впровадження відповідних заходів дозволяє значно знизити ймовірність успішної атаки через OLE–об'єкти.

Перелік використаних джерел.

1. Security Update Guide. [Електронний ресурс]. – Режим доступу: https://msrc.microsoft.com/update–guide/vulnerability

УДК 004.056.5

Надія ГАВРИШКІВ[1], Владислав БАГМЕТ[2]

[1]Галицький фаховий коледж імені Вячеслава Чорновола
[2]Західноукраїнський національний університет

GAME VULNERABILITIES ЯК ЗАГРОЗА КІБЕРБЕЗПЕКИ

Вступ. Баги та вразливості у комп'ютерних іграх є поширеним явищем, особливо в продуктах, що були випущені значний час тому. Така тенденція пояснюється тим, що розробники зосереджують основні ресурси на створенні нових проєктів, тоді як підтримка старих версій поступово скорочується. Унаслідок цього популярні ігри з часом перетворюються на зручне середовище для дослідження та експлуатації вразливостей. До кола потенційних цілей кіберзловмисників належать як розробники, так і користувачі ігор, а в окремих випадках навіть організації, у межах яких ці користувачі працюють.

Мета: дослідити поширені вразливості у популярних продуктах гейміндустрії.

1. Загрози використання службових ПК з ігровим ПЗ

Комп'ютерна гра є загалом програмним продуктом, що розробляється людьми й тому може містити помилки в реалізації чи пропуски під час тестування. Наявність таких дефектів підтверджується публічними реєстрами уразливостей (CVE), які фіксують відомі вразливості та їхню класифікацію за рівнем загрози. З метою ілюстрації цієї проблеми було проведено аналіз даних агрегатора вразливостей і перевірено інформацію щодо окремих проєктів на платформі Steam; для прикладу було відібрано п'ять CVE, пов'язаних із клієнтом гри Dota 2, середній рейтинг тяжкості яких за шкалою CVSS становив 7,8 із 10.[1].

Серед виявлених інцидентів як найсерйозніша була зафіксована у 2023 році внаслідок дослідження, проведеного фахівцями Avast: клієнт Dota 2 використовував застарілу версію рушія JavaScript (V8), яка містила вразливість, що дозволяла виконувати небажаний JavaScript–код на машині користувача. Наслідком таких дефектів може бути несанкціоноване виконання коду на комп'ютері жертви та повна компрометація її середовища.

Подібні ситуації відзначалися й у інших популярних проєктах. Так, для серії Counter–Strike виявлено кілька CVE із середнім значенням CVSS близько 7,76, а у грудні 2023 року було продемонстровано реалізацію XSS–вектору в контексті нової функції додавання зображень у чаті, що створювало можливості цілеспрямованих атак на користувачів. У випадку GTA Online дефект, позначений як CVE–2023–24059, було виявлено та виправлено на початку 2023 року; цей збій дозволяв не лише витягувати дані облікових записів, а й розміщувати шкідливе програмне забезпечення на пристроях жертв.

Слід підкреслити, що запис у реєстрі CVE відображає лише вразливості, які вже стали загальновідомими й, як правило, були усунені та опубліковані. Отже, загальна кількість дефектів у продуктах ігрової індустрії, ймовірно, значно перевищує кількість задокументованих CVE. Додатково практикою є те, що постачальники програмного забезпечення іноді мають відомості про певні

вразливості, але затримують їхнє усунення через пріоритети розробки або інші операційні причини. Ілюстративним прикладом є ситуація з проєктом Call of Duty: Black Ops III (реліз 2015 року), де було задокументовано повідомлення про RCE–уразливості, які залишалися непофіксеними тривалий час; унаслідок відсутності офіційних виправлень частина спільноти розробників–ентузіастів змушена була створювати власні модифікації та виправлення, доступні у відкритих репозиторіях.

Отже, із встановленим ігровим програмним забезпеченням з'являється низка ризиків: від локальних компрометацій окремих робочих станцій до потенційного розповсюдження загроз у внутрішній корпоративній мережі. Це обґрунтовує необхідність оцінки ризиків використання ігрового ПЗ на службі та запровадження політик контролю встановлення й оновлення програмного забезпечення, а також засобів виявлення й реагування на інциденти кібербезпеки.

2. Основні загрози ігровим акантам

Переважна більшість інцидентів із компрометацією облікових записів геймерів мають соціо–інженерний характер, причому найпоширенішим інструментом виступає фішинг. Атаки такого типу організовуються через чати, тематичні форуми та інші майданчики спільнот; їхня популярність пояснюється простотою реалізації та низькими витратами для зловмисника. Типовим прикладом є шахрайські схеми «я продав тобі, але ти не заплатив», які спрямовані на отримання облікових даних або доступу до платіжних інструментів жертви.

Технічно складніші вектори, що використовують програмні вразливості ігрових клієнтів або плагінів, зустрічаються рідше, оскільки вимагають відповідних навичок і часу на розробку експлойтів. Водночас такі атакувальні сценарії не є поодинокими: використання вразливостей може призводити до віддаленого виконання коду, підміни сесійних токенів або похищення облікових даних без прямого залучення користувача. Наслідком експлуатації вразливостей найчастіше стає крадіжка акаунтів, що, з огляду на розвиток внутрішньоігрових ринків (наприклад, торгівлю косметичними предметами), може мати значну матеріальну шкоду. Ілюстративним випадком є інцидент 2022 року з відомим колекціонером предметів у грі серії Counter–Strike, в результаті якого було втрачене право розпоряджатися скінами загальною вартістю порядку мільйонів доларів. Значну загрозу також становить використання модифікованих або нелегально отриманих інсталяційних образів. Піратський софт, поширюваний через торренти й подібні ресурси, часто постачається разом із бекдорами та іншими типами шкідливого програмного забезпечення, що робить його джерелом компрометації кінцевих пристроїв. У багатьох випадках саме завантаження та запуск модифікованих інсталяторів стають початковою точкою проникнення.

Окрему категорію ризиків формують користувацькі практики, які навмисно або мимоволі знижують рівень захисту кінцевої системи. Частина гравців вимикає антивірусні рішення або брандмауери, посилаючись на їхній вплив на продуктивність або сумісність із грою; інші користувачі взагалі відмовляються від активних засобів захисту. Ефективність таких налаштувань у сенсі підвищення продуктивності є предметом дискусій, натомість їхній внесок у підвищення

вразливості системи виявлена й документована: зниження рівня захисту істотно збільшує ймовірність успішної компрометації облікових записів і пристроїв.

Загрози ігровим акаунтам мають багатовимірний характер: від простих соціально–інженерних схем до технічно складних експлуатацій вразливостей і постачання шкідливого ПЗ разом із піратським контентом; ускладнює ситуацію також поведінка самих користувачів, що іноді свідомо знижує заходи захисту. Це підкреслює необхідність комплексного підходу – поєднання технічних засобів, освітніх ініціатив для спільнот і проактивного моніторингу інцидентів.

3. Загрози додаткового ПЗ

Багато загроз також несуть у собі програми, які геймери активно використовують крім ігор. Вони теж бувають критичні дефекти безпеки, як це показано в таблиці 1.

Таблиця 1 – Вразливості ігрових майданчиків та спеціального ПЗ

Steam та інші майданчики для розміщення ігор	Вразливість у Steam була виявлена у 2020 році, використовуючи яку, зловмисник міг захопити сотні тисяч комп'ютерів, не вимагаючи від геймерів натискати на шкідливий лист або посилання. На відміну від інших уразливостей, жертви несвідомо попадали під вплив хакера. Для цього їм потрібно було просто увійти до гри. У 2023 р. зловмисники зламали облікові записи сотні розробників на платформі Steam і додали до їхніх ігор шкідливе ПЗ. Але вендор швидко виявив проблему і повідомив про це користувачам.
Discord та утиліти для спілкування з командою	Повідомлень про проблеми у продукті чимало. Наприклад, розробник визнав витік даних 760 тис. користувачів, яка сталася з вини співробітника.
GeForce Experience, OBS Studio та інші програми для запису відео, оцінки FPS тощо	У 2020 році розробник GeForce Experience залатав відразу дві серйозні дірки. Одна з уразливостей (CVE–2020–5977) отримала CVSS 8,2 і могла призвести до шкідливих атак, включаючи виконання коду, відмову в обслуговуванні, підвищення привілеїв та розкриття інформації.
AutoHotKey та аналоги для налаштування кнопок клавіатури та миші	З його допомогою злочинці поширювали трояни для віддаленого доступу до пристроїв жертв, у тому числі Revenge RAT, LimeRAT, AsyncRAT, Houdini та Vjw0rm.
Spotify та інші сервіси для прослуховування музики під час гри	У 2020 році через виток даних Spotify скинув 350 тис. паролів користувачів. Хоча в офіційній заяві власник продукту повідомив, що проблема торкнулася лише невеликої частини акаунтів

CVE–2020–5977 – уразливість типу «небезпечний/неконтрольований пошук шляху завантаження модулів» (untrusted search path) в компоненті, що використовує середовище виконання Node.js. Через цю слабину процес, який завантажує модулі Node.js (через require() / import), може підхопити шкідливий

модуль із непередбачуваного або керованого зловмисником каталогу. Унаслідок цього можливе виконання довільного коду в контексті вразливого процесу, що може призвести до локальної компрометації системи. Схема вразливості CVE–2020–5977 приведена на рисунку 1.

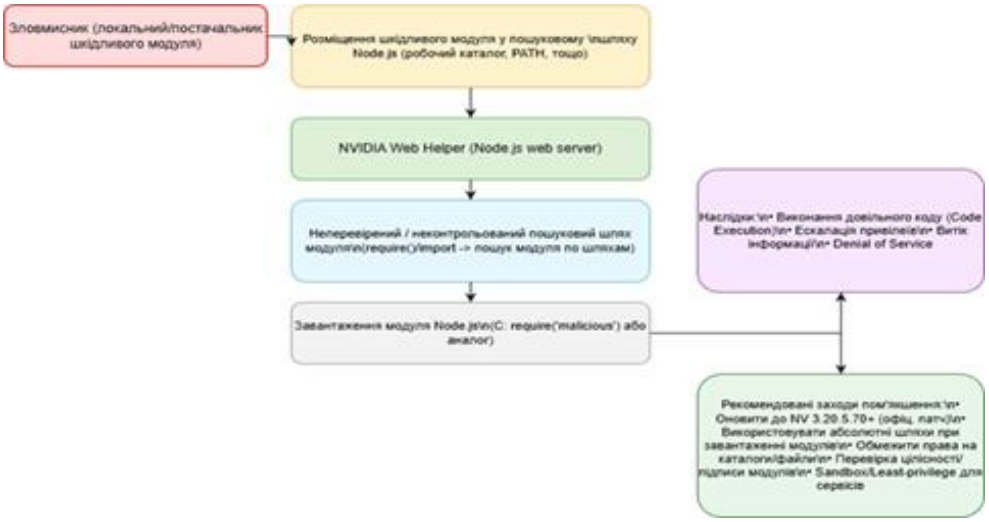


Рисунок 1 – Схема вразливості CVE–2020–5977

CVE–2020–5977 належить до класу вразливостей, які використовують недоліки в управлінні шляхами завантаження модулів у середовищах виконання, таких як Node.js. Хоча сам механізм експлуатації є концептуально простим, практичні наслідки можуть бути критичними – від локального виконання довільного коду до повної компрометації системи, особливо якщо вразливий процес має підвищені права. Найефективнішими заходами захисту є своєчасне оновлення постачальницького ПЗ, суворе управління правами доступу, використання абсолютних шляхів і перевірок цілісності модулів, а також впровадження контролю завантажених компонентів у рантаймі.

Висновок. Безпека відеоігор є багатовимірною проблемою, що поєднує технічні вразливості програмного коду, ризики соціальної інженерії та операційні загрози, пов’язані з практиками розповсюдження та використання ПЗ. Наслідки експлуатації вразливостей охоплюють як індивідуальні втрати користувачів (компрометація акаунтів, витік майна), так і бізнес–ризики для розробників та їхньої інфраструктури. Ефективна протидія потребує комбінованого підходу, що включає безпечну розробку (secure–by–design), регулярне патчування, контроль цілісності бінарників і жорстке управління правами доступу. Додатково необхідні правові механізми та інструменти моніторингу (EDR, HIDS, SIEM), а також просвітницькі заходи для користувацької спільноти щодо фішингу та безпечних практик. Лише системне поєднання технічних, організаційних і правових заходів здатне істотно знизити експлуатаційний ризик і підтримати довгострокову стійкість ігрової екосистеми.

Перелік використаних джерел.

1. CVE–2015–7985. [Електронний ресурс]. – Режим доступу: https://nvd.nist.gov/vuln/detail/CVE–2015–7985/

2. CVE–2020–5977. [Електронний ресурс]. – Режим доступу: https://nvd.nist.gov/vuln/detail/CVE–2020–5977.

УДК 004.65

Аліна ДАВЛЕТОВА

Західноукраїнський національний університет

ПРОЄКТУВАННЯ ТА ЗАХИСТ БАЗ ДАНИХ В УМОВАХ СУЧАСНИХ КІБЕРЗАГРОЗ

Вступ. Бази даних (БД) є невід'ємним компонентом інформаційних систем, що забезпечують безперервне функціонування бізнес–процесів, аналітики, управління корпоративними ресурсами та критично важливих технологічних операцій. Зі зростанням обсягів даних і кількості цифрових сервісів збільшується спектр загроз, спрямованих на порушення конфіденційності, цілісності та доступності інформації. Актуальною задачею є формування сучасних підходів до проєктування систем зберігання даних, які поєднують архітектурну стійкість, криптографічний захист, контроль доступу та інструменти моніторингу інцидентів.

Метою є аналіз принципів проєктування та механізмів захисту БД, визначення рішень, що забезпечують їх стійкість до сучасних кібератак.

1. Загрози та вразливості баз даних

Проєктування БД охоплює вибір архітектури, логічного та фізичного моделювання, визначення вимог до продуктивності, масштабованості та безпеки [1]. На етапі логічного моделювання процес нормалізації структури даних дозволяє усунути надмірність, уникнути аномалій оновлення та забезпечити узгодженість даних. Застосування нормальних форм (1НФ–3НФ, BCNF) сприяє структурованій організації таблиць і формуванню чітких зв'язків між сутностями.

Фізична модель визначає розподіл даних між сегментами чи вузлами, використання індексації та типів даних, а також організацію зберігання файлів БД і резервних копій, що визначає продуктивність системи, забезпечує ізоляцію критичних даних, підвищення відмовостійкості та мінімізацію ризиків несанкціонованого доступу.

Поєднання продуктивності, надійності та безпеки забезпечує архітектура БД. Клієнт–серверна та централізована архітектури забезпечують централізований контроль транзакцій і синхронізацію даних. Розподілена або хмарна архітектура дозволяє забезпечити відмовостійкість і безперервний доступ до даних навіть при збоях окремих вузлів. Архітектура визначає точки застосування політик безпеки: автентифікацію, авторизацію, контроль доступу, сегментацію мережі та ізоляцію середовища виконання. Залежно від вимог безпеки застосовуються концепції Zero Trust, що забезпечує постійну перевірку прав користувачів і пристроїв на кожному рівні доступу.

Попри ретельне проєктування, БД залишаються одним із ключових об'єктів атак. Складність систем, велика кількість користувачів, розподіленість даних, інтеграція з різними сервісами та використання застарілих конфігурацій створюють значну поверхню атаки. На практиці основні загрози виникають як через технічні вразливості, так і через помилки адміністрування, слабку політику доступу та недостатній моніторинг.

Статистичні звіти провідних аналітичних центрів демонструють стійку тенденцію до зростання інцидентів, пов’язаних із несанкціонованим доступом, витоками даних, компрометацією облікових записів, ін’єкційними атаками та модифікацією інформації [2–4]. На рисунку 1 наведено класифікацію основних загроз БД.

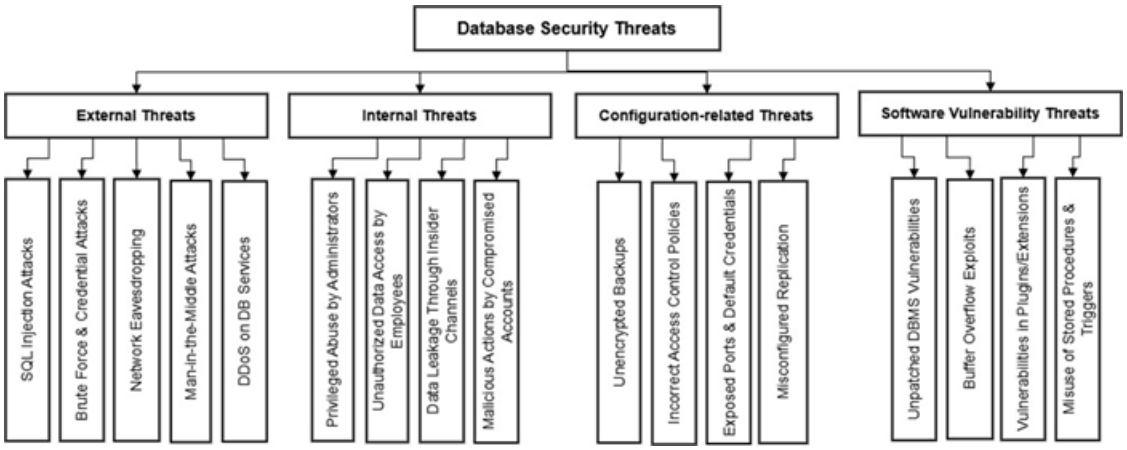


Рисунок 1 – Класифікація загроз БД

Сучасні системи керування базами даних (СКБД) функціонують в умовах постійно зростаючого спектра кіберзагроз, що ускладнює їх захист і вимагає комплексного аналізу механіки атак. Представлена схема на рисунку 2 відображає типовий ланцюг компрометації СКБД, який охоплює всі ключові етапи від первинної розвідки до ексфільтрації та довготривалого закріплення в системі.

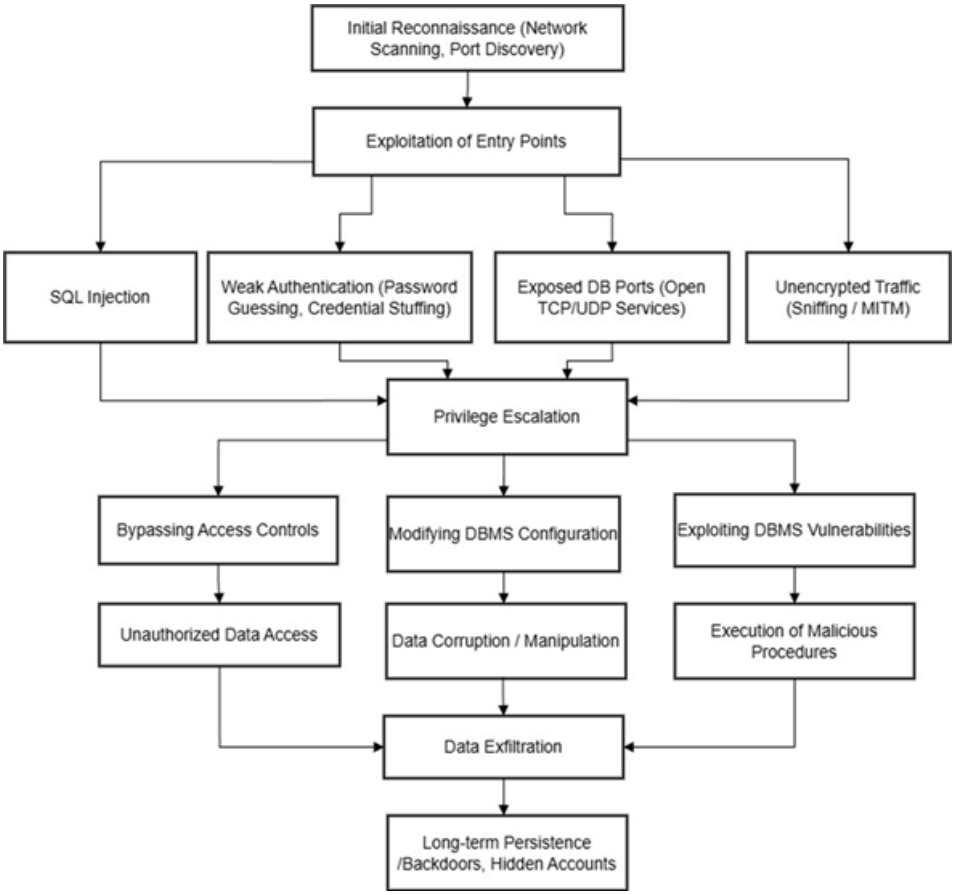


Рисунок 2 – Вектори атак СКБД

Такий підхід дозволяє цілісно зрозуміти логіку дій зловмисника та визначити критичні точки контролю безпеки.

Атака починається з етапу розвідки, під час якого здійснюється збір технічної інформації про мережеву інфраструктуру, відкриті порти та сервіси СКБД. На основі виявлених слабких місць зловмисник переходить до експлуатації точок входу, використовуючи SQL–ін'єкції, слабку автентифікацію, відкриті порти або нешифрований трафік для отримання первинного доступу. Після проникнення реалізується підвищення привілеїв, що дозволяє розширити контроль над СКБД та обійти механізми доступу. Це створює умови для пост–експлуатації: зміни конфігурації, використання вразливостей СКБД, виконання шкідливих процедур та отримання несанкціонованого доступу до даних. Подальші дії зловмисника спрямовані на маніпулювання, пошкодження або копіювання даних, що безпосередньо порушує цілісність та конфіденційність інформації. Завершальним етапом є ексфільтрація даних та встановлення довготривалого прихованого доступу через бекдори або створені облікові записи, що забезпечує можливість повторної компрометації системи.

2. Забезпечення цілісності та відмовостійкості баз даних

У контексті сучасних кіберзагроз забезпечення цілісності та відмовостійкості баз даних неможливе без застосування комплексних криптографічних механізмів, що охоплюють як клієнтський рівень, так і серверну інфраструктуру [5, 6]. Як показано на рисунку 3, криптографічний захист формується багаторівнево, від створення та передачі даних до їх зберігання й контролю доступу. Криптографія виступає фундаментальним інструментом підтримання конфіденційності, автентичності та цілісності інформації в БД.

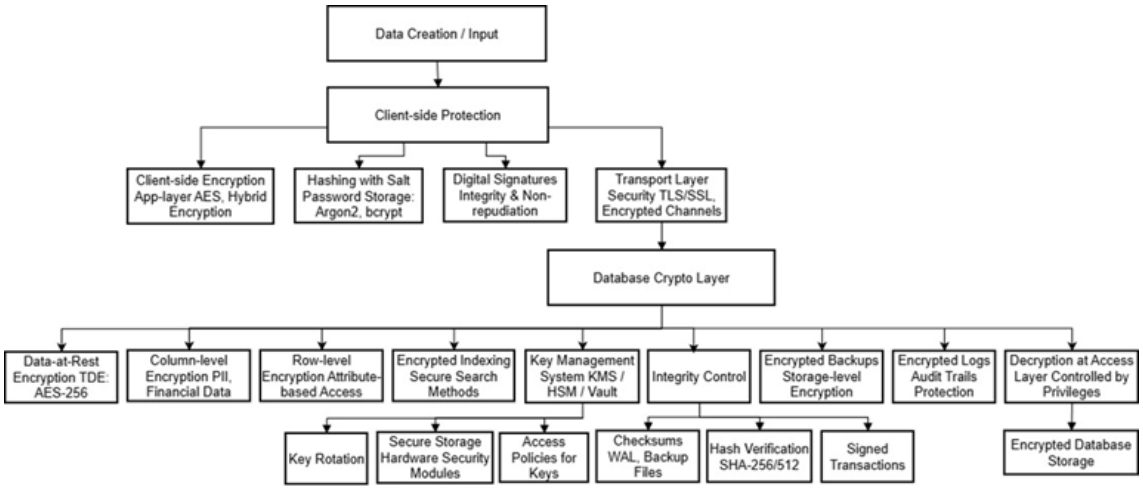


Рисунок 3 – Криптографічний захист в СУБД

Шифрування даних у стані спокою (Data–at–Rest Encryption) застосовується для захисту файлів бази даних, окремих таблиць або колонок, а також сторінок пам'яті. Найпоширенішими технологіями є Transparent Data Encryption (TDE), Always Encrypted та pgcrypto, які забезпечують мінімізацію ризику несанкціонованого доступу у разі компрометації серверів або резервних копій. Для захисту даних у русі використовується TLS/SSL, що гарантує криптографічну стійкість каналів зв'язку між клієнтом і сервером. Критично важливим є

застосування сучасних шифрів та вимкнення застарілих версій протоколів.

Хеш–функції, такі як SHA–256/512, відіграють ключову роль у перевірці цілісності даних, формуванні цифрових підписів транзакцій та створенні стійких до відновлення паролів, що зберігаються у вигляді сольових хешів (bcrypt, Argon2). Використання таких механізмів унеможливлює непомітну модифікацію даних і підвищує надійність контролю достовірності операцій.

Для систем підвищеної критичності застосовуються розподілені та апаратно прискорені криптографічні рішення, включно з апаратними модулями HSM для управління ключами, атрибутивними схемами шифрування (ABE) та постквантовими алгоритмами (CRYSTALS–Kyber, Dilithium). Такі технології формують додатковий рівень захисту, що підвищує стійкість БД до складних атак, забезпечує надійне відновлення після інцидентів та гарантує довготривалу криптографічну безпеку.

Відмовостійкість БД ґрунтується на сукупності механізмів, спрямованих на збереження доступності та безперервності роботи СУБД навіть у разі збоїв, відмов апаратного забезпечення або зовнішніх інцидентів. Як показано на рисунку 4, ключовим елементом є використання основного вузла (Primary Database) разом із кластером реплік, що забезпечують синхронну або асинхронну реплікацію даних у режимі реального часу.

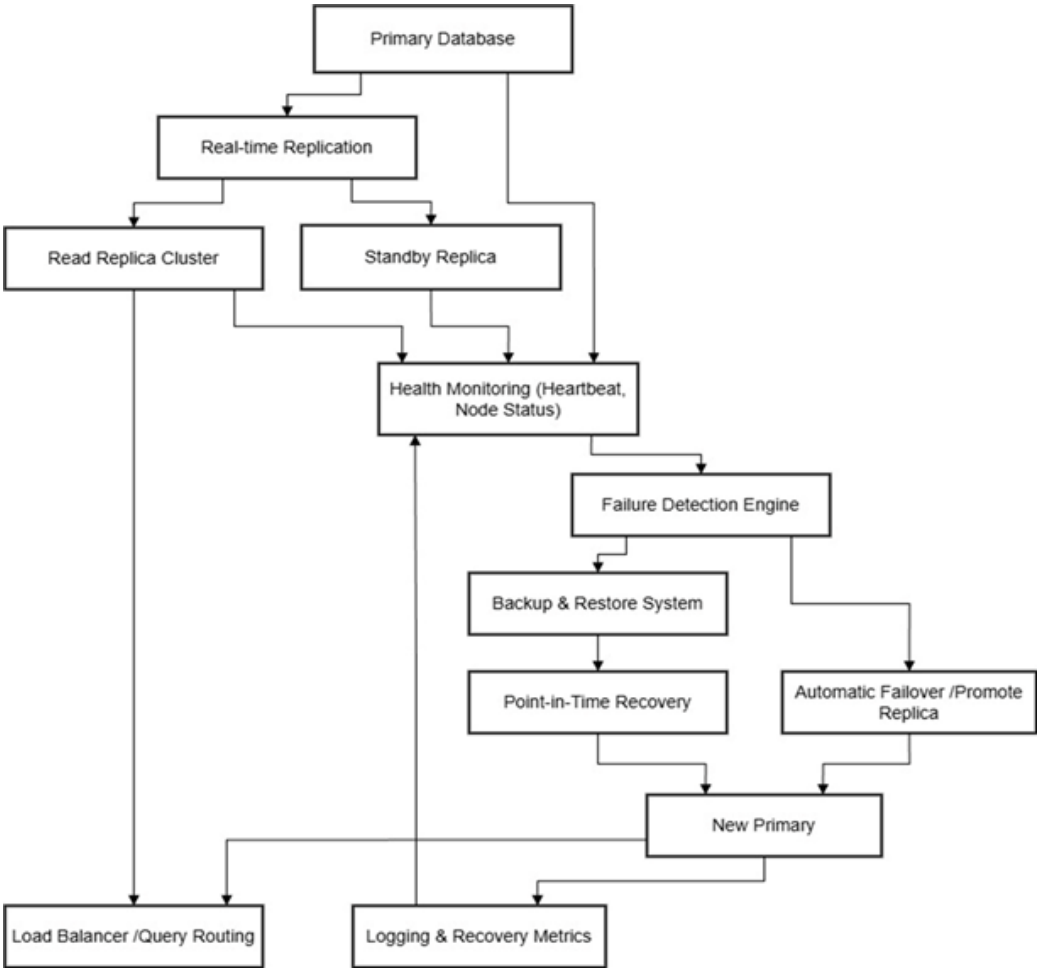


Рисунок 4 – Модель відмовостійкості та доступності БД

Така архітектура дозволяє оперативно перемикатися на резервний вузол у випадку збою, що мінімізує час простою системи.

Транзакційна модель ACID є фундаментом для забезпечення цілісності в умовах відмовостійких конфігурацій, оскільки її властивості атомарності, узгодженості, ізоляції та довговічності гарантують коректність виконання операцій під час роботи з репліками та у процесі відновлення. Система резервного копіювання включає журнальне копіювання (WAL), технології Point–in–Time Recovery та автоматизовані процедури відтворення даних, що дає змогу повернути базу даних до попереднього стану після помилок, компрометації або збою основного вузла.

Центральне місце відводиться механізмам моніторингу працездатності, які використовують heartbeat–сигнали та контроль стану вузлів для раннього виявлення відмов. Після фіксації збою система автоматизованого перемикання виконує підвищення ролі репліки до нового основного вузла, підтримуючи безперервність сервісу. Додатковим рівнем захисту від зовнішніх інцидентів є застосування DDoS–захисту, сегментації мережі, мережевих екранів та проксі–сервісів, які зменшують ризик порушення доступності СКБД.

Висновок. Сучасні кіберзагрози вимагають комплексного підходу до проєктування та захисту баз даних, який поєднує безпечну архітектуру, криптографічні механізми та ефективний контроль доступу. Аналіз векторів атак демонструє необхідність багаторівневого захисту, що охоплює як попередження компрометації, так і виявлення пост–експлуатаційної активності. Забезпечення цілісності та відмовостійкості баз даних досягається завдяки шифруванню, транзакційній моделі ACID, реплікації, резервному копіюванню та автоматизованим механізмам відновлення. Сукупність цих рішень формує стійку інфраструктуру, здатну протистояти атакам і гарантувати безперервність роботи інформаційних систем.

Перелік використаних джерел.

1. Jain A. Architecting Scalable and Distributed Cloud Database Systems. International Journal of Technology, Management and Humanities. 11. 2025. 10.21590/ijtmh.11.02.04.

2. ITRC 2023 Data Breach Report. [Електронний ресурс].– Режим доступу: https://www.idtheftcenter.org/wp–content/uploads/2024/01/ITRC_2023–Annual–Data–Breach–Report.pdf

3. Global Data Breaches and Cyber Attacks in March 2024. [Електронний ресурс].– Режим доступу: https://www.itgovernance.co.uk/blog/global–data–breaches–and–cyber–attacks–in–march–2024–299368075–records–breached

4. Global number of data breaches with confirmed data loss from November 2023 to October 2024. [Електронний ресурс].– Режим доступу: https://www.statista.com/statistics/329608/security–incidents–confirmed–data–loss–industry–size

5. Sun B., Zhao S., Tian G. SQL queries over encrypted databases: a survey. Connection Science, 2024, 36(1). https://doi.org/10.1080/09540091.2024.2323059

6. Malhotra S., Singh W. An efficacy analysis of data encryption architecture for cloud platform. *Procedia Computer Science*, 218, 2023, pp. 989–1002. ISSN 1877–0509. https://doi.org/10.1016/j.procs.2023.01.079

УДК 004.056.5

Віктор ДЗЯДИК[1], Степан ІВАСЬЄВ[12]

[1]Галицький фаховий коледж імені В'ячеслава Чорновола
[2]Західноукраїнський національний університет

АУДИТ ЦИФРОВИХ ПІДПИСІВ ВСТАНОВЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Перевірка цифрових підписів програмного забезпечення є одним із ключових механізмів забезпечення цілісності, автентичності та довіри до ПЗ у сучасних інформаційних системах.

У умовах постійного зростання складності атак, націлених на ланцюги постачання, використання шкідливих бібліотек, інжекцію коду та модифікацію виконуваних файлів, цифровий підпис виступає критично важливим інструментом, який підтверджує, що файл був створений та підписаний надійним виробником і не був змінений третіми сторонами.

Метою дослідження є розроблення та обґрунтування методів і засобів автоматизованої оцінки автентичності та цілісності програмного забезпечення шляхом аналізу цифрових підписів, ланцюгів сертифікації та часових міток, а також виявлення потенційно небезпечних або недовірених сертифікатів у системному сховищі для підвищення рівня кіберзахисту.

1. Забезпечення цілісності програмного забезпечення

Цифровий підпис гарантує, що після підписання виконавчий файл, бібліотека або драйвер не були модифіковані. Будь-яка зміна навіть одного байта призводить до недійсності підпису. У кібербезпеці це дозволяє: виявляти компрометацію програм перед імпортуванням у систему, запобігати атакам шляхом підміни файлів, контролювати наявність сторонніх або модифікованих компонентів.

У середовищах, де проводиться автоматизація розгортання або оновлення систем, перевірка підписів є обов'язковою частиною безпечного циклу CI/CD.

2. Перевірка автентичності автора та ланцюга довіри

Цифровий підпис прив'язує програму до конкретного виробника та сертифікаційного центру (CA), який підтвердив його особу. Це забезпечує: можливість ідентифікувати, ким саме створено файл, контроль над тим, які постачальники є довіреними, виявлення файлів, підписаних фальшивими або відкликаними сертифікатами (рисунок 1).

Дерево довірених сертифікатів має декілька етапів перевірки, що включають перевірку кожного елемента в звопротньому напрямку. Дерево довіри (Certificate Chain / Certification Path)**,** починається від кореневого сертифіката (Root CA), проходить через проміжні (Intermediate CA) і завершується підписаним сертифікатом (Leaf Certificate). Для створення сертифікатів можуть бути застосовані наступні криптовлгоритми: RSA, ECC, ECDSA, EdDSA, алгоритми національних стандартів.

Перевірка ланцюга сертифікації та наявності сертифікату у списку відкликаних (CRL, OCSP) допомагає виявити спроби використання викрадених ключів, що є поширеним методом атак.

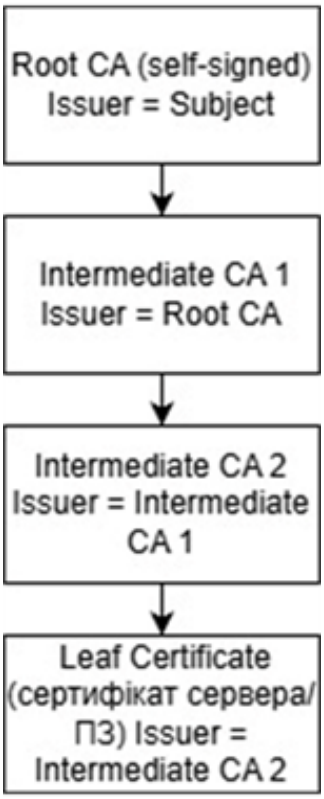


Рисунок 1 - Дерево довірених сертифікатів

3. Протидія атакам на ланцюг постачання (Supply Chain Attacks)

Останні роки показали, що однією з найнебезпечніших загроз є компрометація програм на етапі розробки або оновлення. Відомі випадки:

– SolarWinds (2020) - у легальний продукт був інтегрований бекдор [1].

– Ін’єкція шкідливих модулів у NPM/PyPI[2].

– CCleaner – підписаний, але заражений легітимним виробником файл [3].

Перевірка підписів дозволяє: переконатися, що отримане ПЗ офіційне, а не модифіковане, виявляти підроблені або підозрілі сертифікати, автоматизувати механізми контролю при розгортанні ПЗ.

4. Відповідність корпоративним політикам безпеки та нормативним вимогам

Багато організацій впроваджують політику: «Запускати лише програмне забезпечення, підписане довіреним видавцем».

Це є вимогою: фінансових установ, державних структур, сертифікованих SOC, критичної інфраструктури, підприємств, що використовують стандарти ISO/IEC 27001, NIST, CIS. Автоматизована перевірка підписів допомагає дотримуватися політик та аудитних вимог.

5. Виявлення сторонніх або потенційно небажаних компонентів

Перевірка підписів дозволяє знайти файли: без підпису, підписані невідомими видавцями, підписані простроченими або відкликаними сертифікатами.

Такі файли часто вказують на: встановлене неофіційне ПЗ, “portable” інструменти неясного походження, malware, який імітує легітимний софт, підміну бібліотек (DLL hijacking), встановлення сторонніх драйверів rootkit-рівня. Це критично в рамках кібергігієни підприємства та роботи SOC.

6. Аналіз кореневих сертифікатів у системі як додатковий захист

Перевірка ланцюга сертифікатів передбачає довіру до системного сховища кореневих ЦСК. Атаки часто проводять так:

1. У систему встановлюється фальшивий кореневий сертифікат.
2. Зловмисник підписує malware приватним ключем.
3. Windows вважає цей підпис валідним, бо ЦСК є "довіреним".

Тому необхідно перевіряти: самопідписані сертифікати, ЦСК, видані незрозумілими організаціями, сертифікати з нетипово коротким строком дії, сертифікати, встановлені вручну поза системним оновленням. Це одна з головних причин, чому твій інструмент - важлива частина безпеки.

7. Підвищення рівня контролю та зменшення ризику інфікування

Регулярна перевірка цифрових підписів: унеможливлює випадковий запуск неперевірених файлів; зменшує вектор атак через підміну виконуваних файлів, дозволяє автоматично блокувати ризикове ПЗ, підвищує загальну стійкість системи до компрометації.

Висновок. Перевірка цифрових підписів ПЗ є одним з найефективніших і найнадійніших методів контролю цілісності, автентичності та безпечного походження програмних компонентів. Це обов’язкова частина сучасного підходу до кіберзахисту, дозволяє запобігати атакам на ланцюг постачання, виявляє підміну ПЗ і захищає від встановлення шкідливих або неперевірених компонентів у системі.

Перелік використаних джерел.

1. Analyzing Solorigate: the compromised DLL file that started a sophisticated cyberattack and how Microsoft Defender helps protect. 18 грудня 2020.Режим доступу: https://www.microsoft.com/en-us/security/blog/2020/12/18/analyzing-solorigate-the-compromised-dll-file-that-started-a-sophisticated-cyberattack-and-how-microsoft-defender-helps-protect/

2. SolarWinds Worldwide, LLC. An Investigative Update of the Cyberattack.2020.Режим доступу:https://www.solarwinds.com/blog/an-investigative-update-of-the-cyberattack

3. Greenberg A. Inside the Unnerving Supply-Chain Attack That Corrupted CCleaner. WIRED Magazine, 2018.Режим доступу: https://www.wired.com/story/inside-the-unnerving-supply-chain-attack-that-corrupted-ccleaner/

УДК 004.056.5

Олександр ДРОЖАК

Західноукраїнський національний університет

ПОЛІНОМІАЛЬНИЙ АЛГОРИТМ ПЕРЕВІРКИ ЧИСЕЛ НА ПРОСТОТУ: ТЕСТ АГРАВАЛА–КАЯЛА–САКСЕНИ

Вступ. Тест простоти AKS є першим детермінованим поліноміальним алгоритмом, який доводить простоту натуральних чисел без використання додаткових гіпотез або випадковості. Його поява стала фундаментальним проривом у теорії чисел та алгоритмічній криптографії, оскільки забезпечила строгий математичний підхід до перевірки простоти. Алгоритм базується на властивостях многочленів у кільці $Z_n[x]$ та дозволяє гарантовано класифікувати число як просте чи складене. Попри значну теоретичну цінність, практичне застосування AKS обмежується його високою обчислювальною складністю порівняно з імовірнісними тестами. Дослідження алгоритму є важливим для глибшого розуміння меж детермінованих методів у криптографічних системах та сучасних обчислювальних моделях.

Метою є дослідження ефективності поліноміального алгоритму перевірки чисел на простоту, зокрема тесту Агралила–Каяла–Саксени. У роботі розглядаються теоретичні основи алгоритму, його математична обґрунтованість та порівняння з іншими методами перевірки простоти чисел, що дозволяє оцінити швидкість та точність його виконання на великих числах. Також вивчаються практичні аспекти застосування цього алгоритму для криптографічних задач та обчислювальних обмежень.

1. Всластивості AKS тесту простоти

Індійськими математиками Агравалом, Кайялом та Саксеною у роботі «PRIMES is in P» було вперше запропоновано алгоритм перевірки простоти чисел, який одночасно є поліноміальним, універсальним, детермінованим та безумовним. До цього були відомі алгоритми, які мали максимум три із чотирьох властивостей.

Поліноміальність означає, що складність алгоритму обмежена поліномом від кількості біт у числі. Прикладом тесту, який не задовольняє властивості поліноміальності, є тест Адлемана–Померанца–Румелі.

Універсальність має на увазі, що алгоритм застосовний до будь–яких чисел, а не лише до чисел спеціального вигляду. Прикладом неуніверсального алгоритму є тест Люка–Лемера, який можна застосовувати лише для чисел Мерсенна.

Детермінованість означає, що алгоритм завжди видає ту саму відповідь на одних і тих самих вхідних даних. Прикладом недетермінованого (імовірнісного) тесту може бути тест Міллера–Рабіна.

Безумовність – це незалежність від недоведених гіпотез. Чи не задовольняє властивості безумовності, наприклад, тест Міллера, який спирається на узагальнену гіпотезу Рімана.

Введемо позначення:

Zn – кільце відрахувань по модулю *n*.

Запис $(x) = h(x)(mod\ n)$ означає, що многочлени, коефіцієнти яких сприймаються як відрахування з Z_n , рівні.

Запис $g(x)=h(x)$ (mod x^r-1,n) означає, що багаточлени з коефіцієнтами Z_n рівні по модулю многочлена x^r-1.

$ord_r(n)$ – порядок числа n за модулем r. Мінімальна кількість k, таке що $n^k=1$ (mod r).

$\varphi(n)$ – функція Ейлера. Кількість натуральних чисел, менших n і взаємно простих із нею.

$\mathrm{Log}(n)$ – логарифм за основою два числа n.

2. Принцип роботи алгоритму

Основна ідея алгоритму спирається наступну теорему: якщо числа a і n взаємно прості, то тотожність $(x+a)^n=x^n+a$ (mod n) виконується тоді й лише тоді, коли n – просте.

Цю тотожність вже можна використовувати для перевірки простоти. Але алгоритм не буде поліноміальним, тому що потрібно зробити перевірок – за кількістю коефіцієнтів у поліномах. Ідея полягає в тому, щоб розглядати рівність поліномів за модулем іншого полінома, ступенем меншим, ніж n. З теореми випливає слідство:

Як наслідок: якщо n просте, то для будь–якого натурального r та будь–якого натурального $0 < a < n$ виконується $(x+a)^n=x^n+a$ (mod x^r-1,n).

Зворотне твердження у випадку неправильно. У статті індійських математиків доведено, що для деякої r і деякої множини значень a зворотне твердження буде вірним:

Якщо існує r, таке що $ord_r(n) > \log^2 n$, при якому тотожність

$$(x+a)^n=x^n+a \text{ (mod } x^r-1, n).$$

Виконано для всіх $1 \le a \le \lfloor \sqrt{\varphi(n)} \log n \rfloor$, то n або просто, або ступінь простого.

Це дозволяє побудувати поліноміальний алгоритм перевірки чисел на простоту, що є основним результатом роботи.

3. Алгоритм роботи тесту простоти AKS

На вхід алгоритму подається натуральне число $n > 1$.

Вихід: ПРОСТЕ число, якщо n – просте число. СКЛАДНЕ в іншому випадку. Якщо n – точний степінь деякого числа $(n = a^b, a, b \in N, b > 1)$, повернути СКЛАДНЕ.

Знайти найменше r, що $ord_r(n) > \log^2 n$. Якщо $1 <$ НСД$(a, n) < n$ для деякого $a \le r$, повернути СКЛАДНЕ. Якщо $n \le r$ повернути СКЛАДНЕ.

Для кожного $1 \le a \le \lfloor \sqrt{\varphi(n)} \log n \rfloor$: Якщо $(x+a)^n \ne x^n+a$ (mod x^r-1, n) повернути СКЛАДНЕ.

Повернути ПРОСТЕ.

Розглянемо докладно реалізацію кожного кроку алгоритму та розберемо виконання з прикладу числа 47.

На першому кроці слід визначити, чи є число n точним ступенем іншого числа, тобто $n=a^b$. Мабуть, найпростіший спосіб зробити це – перевірити, що

$\lfloor n^{1/b} \rfloor^b \neq n$. При цьому достатньо перевіряти лише значення $2 \leq b \leq log\ n$ (оскільки якщо , то b>log n).

Функція для виконання перевірки буде мати наступний вигляд:

```
function isPerfectPower(n)
{
   for (b = 2; b < ceil(log(n)); b++) {
      if (floor(n ** (1.0 / b)) ** b) == n:
         return true;
   }
   return false;
}
```

Розглянемо приклад :

$$\lfloor log\ 47 \rfloor = 5.$$

Перевіряємо від 2 до 4:

$$\lfloor 47^{1/2} \rfloor^2 = 36;$$
$$\lfloor 47^{1/3} \rfloor^3 = 27;$$
$$\lfloor 47^{1/4} \rfloor^4 = 16.$$

Отже, 47 не є точним ступенем іншого числа.

Відповідне *r* знаходиться перебором. Для кожного *r* потрібно:

Перевірити, що $n^k \neq 1 \pmod r$ для всіх $k \leq \lfloor log^2\ n \rfloor$.

Якщо одне із значень дорівнює одиниці – спробувати наступне *r*.

Якщо всі рівні одиниці – *r* знайдено.

Відомо, що шукане *r* має порядок не більше $O(\log^5 n)$.

Функція знаходження *r* буде складатись з циклу, котрий зупиниться при виконанні умови пошуку.

```
function findR(n) {
   r = 2;
   while (true) {
      find = true;
      for (k = 1; k <= ceil(log(n) ** 2); k++) {
         if (n ** k % r == 1) {
            find = false;
            break
         }
      }

      if (find) {
         return r;
      } else {
         r++;
      }
   }
}
```

Приклад:

$$\lfloor log\ ^2 47 \rfloor = 30.$$

Шукане r = 41:

$47^1 (\text{mod } 41)=6$

$47^2 (\text{mod } 41)=36$

…

$47^{30} (\text{mod } 41)=9$

Наступні кроки елементарні, потрібно лише реалізувати обчислення НСД двох чисел.

Досягнення теоретичної складності досить використовувати швидкі алгоритми для арифметичних операцій із многочленами. Тут розглянемо одну ефективну практично оптимізацію: перебування залишку від поділу без прямого поділу многочленів, використовуючи лише додавання.

Якщо многочлен ступеня r–1 зводиться у квадрат, то результаті виходить многочлен ступеня 2r–2:

$$f(x) = c_{2r-2}x^{2r-2}+\ldots+c_r x^r + c_{r-1}x^{r-1} + c_{r-1}x^{r-1}+\ldots+c_1 x + c_0.$$

Розглянемо поділ на x^r–1 $x^r - 1$:

$f(x) = a(x)(x^r - 1) + b(x)$, де

$a(x) = a_{r-2}x^{r-2}+\ldots+a_0$

$b(x) = b_{r-2}x^{r-2}+\ldots+b_0$

Розкриваючи дужки, знаходимо коефіцієнти b:

$b_0 = c_0 + a_0 = c_0 + c_r$

$b_1 = c_1 + a_1 = c_1 + c_{1+r}$

$b_{r-2} = c_{r-2} + a_{r-2} = c_{r-2} + c_{2r-2}$

```
function mod (p) {
   for (i = p.degree(); i >= r; i—){
      p[i – r] = p[i – r] + p[i];
      p[i] = 0;
   }
}
```

Приклад: $\left\lfloor \sqrt{\varphi(41) log\ 41} \right\rfloor = 36$

$$a = 1: (x+1)^{47} = x^{47} + 1\ (mod\ x^{41} - 1,47) = x^6 + 1$$

$$a = 2: (x+2)^{47} = x^{47} + 2\ (mod\ x^{41} - 1,47) = x^6 + 2$$

…

$$a = 36: (x+36)^{47} = x^{47} + 36\ (mod\ x^{41} - 1,47) = x^6 + 36$$

4. Оцінка складності

Введемо позначення: $\tilde{O}(t(n)) = O(t(n)) + poly(log(t(n)))$, де t(n) – Деяка функція від n.

При оцінках спиратимемося те що, що множення і розподіл m–бітних чисел мають складність $\tilde{O}(m)$. Відповідно, операції над поліномами ступеня d мають складність $\tilde{O}(dm)$. НСД двох m–бітних чисел можна знайти за $O(log\ n)$.

На першому етапі визначення, чи є число точним ступенем, має складність $\tilde{O}(log^3 n)$.

На другому кроці слід знайти r. Це робиться перебором можливих значень r

та перевіркою для всіх. Для кожного r це вимагає множення по модулю r, отже для кожного r складність $\check{O}(log^2 n\, log\, r)$. Враховуючи, що r має порядок $O(log^5 n)$, складність всього кроку дорівнює $\check{O}(log^7 n)$.

Третій крок вимагає обчислення НСД для r чисел. Загальна складність кроку $O(r\ log\ \ n) = O(\log^6 n)$.

На п'ятому кроці слід перевірити рівності $\lfloor\sqrt{\varphi(n)}\log n\rfloor$. Кожна рівність включає $O(log\ \ n)$ множення поліномів ступеня r, які мають коефіцієнти порядку $O(log\ \ n)$. Кожна рівність може бути перевірена за $\check{O}(log^2 n)$. Разом складність кроку $\check{O}(r\sqrt{\varphi(r)}\, log^3\, n = \check{O}(r^{3/2}\, log^3 n) = \check{O}(log^{21/2} n)$. Цю оцінку можна покращити. Найкраща оцінка на даний момент $\check{O}(log^6 n)$. Оцінка складностей тестів простоти приведена в таблиці 1.

Таблиця 1 – Обчислювальні складності відомих тестів простоти

Тест	Тип	Ймовірність помилки	Орієнтовна складність	Придатність
Ферма	Ймовірнісний	до 1/2	O((log n)^3)	Теоретичний, попередній відсів
Соловей–Штрассен	Ймовірнісний	≤ 1/2	O((log n)^3)	Рідко використовується
Міллер–Рабін	Ймовірнісний	≤ (1/4)^k	O(k(log n)^3)	Основний у криптографії
Люка	Детермінований / ймовірнісний	0 / низька	O((log n)^3)	Додаткова перевірка
Люка–Лемера	Детермінований (для Мерсена)	0	O((log n)^2)	Спеціалізований, ефективний
AKS	Детермінований	0	O((log n)^6)	Теоретичний, не практичний

Порівняння обчислювальних складностей на логарифмічній шкалі можна зобразити на графіку як показано на рисунку 1.

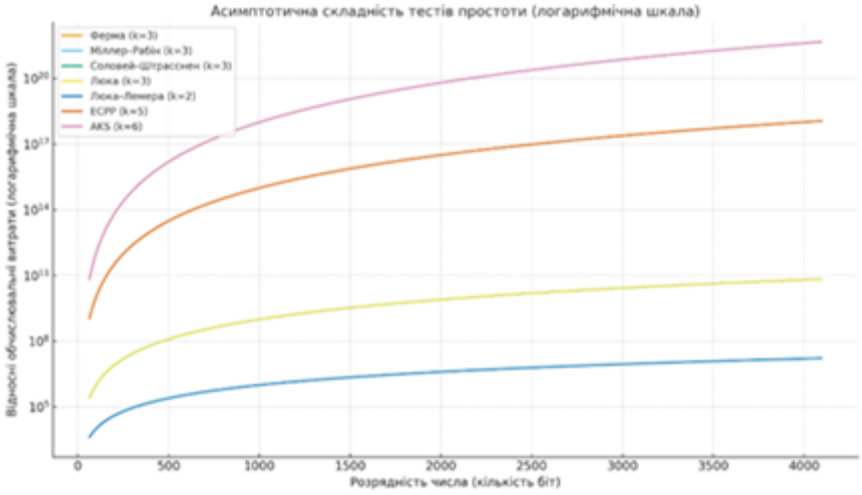


Рисунок 1 – Порівняння обчислювальних складностей тестів простоти на логарифмічній шкалі

Перспективи вдосконалення тесту простоти AKS доцільно розглядати у двох взаємопов'язаних площинах: теоретичній, що стосується асимптотичної складності алгоритму, та прикладній, пов'язаній із його практичною придатністю у криптографічних системах. Попри те, що AKS став фундаментальним результатом, який уперше довів можливість детермінованої перевірки простоти в поліноміальному часі без додаткових гіпотез, його базова форма залишається істотно повільнішою за ймовірнісні тести та спеціалізовані детерміновані методи для чисел певних класів. Саме тому подальші дослідження зосереджуються не стільки на зміні структури алгоритму, скільки на оптимізації його ключових компонентів, зокрема вибору параметра r, організації обчислень у кільці многочленів та скороченні кількості перевірок конгруенцій.

Висновок. Проведене дослідження дозволило встановити, що тест простоти AKS є ключовим теоретичним результатом у сучасній алгоритмічній теорії чисел, оскільки вперше продемонстрував можливість детермінованої поліноміальної перевірки простоти. Алгоритм AKS має важливе теоретичне значення, проте не застосовується практично. По–перше, оцінка складності містить поліном високого ступеня. По–друге, алгоритм вимогливий з пам'яті. Навіть при оцінці параметра для перевірки числа розміром 1024 біт знадобиться близько 1 гігабайта пам'яті. Його математична строгість і відсутність залежності від імовірнісних припущень роблять алгоритм фундаментальним елементом у вивченні меж обчислюваності та побудові надійних криптографічних методів. Разом із тим, експериментальна оцінка показує, що через високу практичну складність AKS поступається сучасним стохастичним тестам за швидкодією та ефективністю, що обмежує його застосування в реальних системах. Незважаючи на це, він залишається важливим еталоном для розробки нових детермінованих методів і поглибленого аналізу алгоритмів перевірки простоти. Отримані результати підтверджують значення AKS як теоретичної основи для подальших досліджень у сфері криптографії та обчислювальної теорії чисел.

Перелік використаних джерел.

1. J.P. Buhler Algorithmic Number Theory: Proc. ANTS–III – Portland, OR, v.1423, Lect.Not.Comp.Sci. Springer–Verlag, 1998, 640 p.

2. D. Venturi Lecture Notes on Algorithmic Number Theory. – Springer–Verlag, New–York, Berlin, 2009, 217 p.

3. Sh.T. Ishmukhametov Methods of factorization of natural numbers: a tutorial.– Kazan, Kazan University, 2011, 190 p.

4. Ya.M.Nikolaichuk, Kasianchuk M.M., Yakymenko I.Z., Ivasiev S.V Vector and modular method of multiplication of multidigit numbers in RademacherKrestenson basis. Herald of the National University "Lviv Polytechnic" "Computer systems and networks", no. 694, 2014, pp. 118–125.

5. M.Kasyanchuk, I. Yakymenko, Y.Nykolajchuk. Matrix Algorithm of Processing of the Information Flow in Computer Systems Based on Theoretical and Numerical Krestenson's Based. Proceedings of the Integrational Conference TCSET'2010, February 23–27, 2010, p. – C: 241

УДК 004.056

Віталій КЛІМ, Тарас ЦАВОЛИК

Західноукраїнський національний університет

АРХІТЕКТУРА СИСТЕМИ БЕЗПЕКИ KUBERNETES

Вступ. Сучасні організації дедалі частіше впроваджують контейнеризовані середовища для забезпечення масштабованості, ефективності та автоматизації розгортання додатків.

Однак разом із перевагами контейнеризації з'являються нові вектори атак, які потребують розроблення спеціалізованих рішень безпеки. Kubernetes, як домінуюча система оркестрації контейнерів, вимагає чітко спроєктованої архітектури безпеки, що забезпечує конфіденційність, цілісність і доступність сервісів у динамічному середовищі.

Мета. Метою дослідження є аналіз архітектури системи безпеки Kubernetes, визначення ключових компонентів і механізмів захисту, а також розроблення концептуальної моделі інтегрованої безпеки з урахуванням сучасних викликів кіберзагроз.

1. Компоненти архітектури безпеки Kubernetes

Система безпеки Kubernetes ґрунтується на багаторівневій моделі, що охоплює безпеку кластера, вузлів і контейнерів [1].

Основні її складові:

1. Authentication та Authorization – механізми перевірки автентичності користувачів і сервісних облікових записів із подальшим контролем доступу через Role-Based Access Control (RBAC).

2. Admission Controllers – політики, які перехоплюють запити до API Server та застосовують правила безпеки (наприклад, заборона привілейованих контейнерів, перевірка labels, namespace-ізоляція).

3. Network Policies – контроль взаємодії між Pods через Cilium або Calico згідно з Zero Trust-підходом.

4. Secrets Management – безпечне зберігання та передача конфіденційних даних за допомогою Kubernetes Secrets і інтеграції з зовнішніми KMS (наприклад, HashiCorp Vault).

5. Audit Logging – фіксація усіх подій у кластері для подальшого аналізу та виявлення аномалій.

Важливою умовою ефективного функціонування є інтеграція всіх цих компонентів через єдиний центр моніторингу (SIEM або Elastic Stack), що дозволяє здійснювати кореляцію подій у реальному часі.

2. Інтеграція динамічних політик та автоматизація захисту

У сучасних DevOps-процесах статичні налаштування безпеки є недостатніми, оскільки середовище Kubernetes змінюється динамічно. Тому використовуються інструменти Kyverno та OPA (Gatekeeper), які забезпечують політикоорієнтований контроль і автоматичне оновлення правил відповідно до контексту системи [2].

Наприклад, якщо контейнер намагається запуститися з надмірними привілеями, система може автоматично заблокувати цей процес. Додатково інструменти Falco та Tracee, що використовують eBPF, дозволяють відстежувати поведінку контейнерів на рівні ядра, виявляючи спроби несанкціонованого доступу або ескалації прав.

Інтеграція таких рішень у CI/CD-конвеєри реалізує концепцію “security as code”, де безпека є невід’ємною частиною життєвого циклу розробки програмного забезпечення. Це забезпечує постійний контроль і мінімізацію ризиків уразливостей на всіх етапах розгортання.

Таблиця 1 – Основні компоненти системи безпеки Kubernetes

Компонент	Призначення	Приклад інструменту
RBAC	Контроль доступу до ресурсів	Kubernetes API Server
Network Policy	Обмеження трафіку між Pods	Calico, Cilium
Admission Controller	Динамічне застосування правил	Kyverno, OPA Gatekeeper
Runtime Security	Моніторинг аномалій	Falco, Tracee

Висновок. Архітектура системи безпеки Kubernetes повинна розглядатися як єдина багаторівнева структура із динамічним застосуванням політик і автоматизованим виявленням аномалій. Поєднання RBAC, Network Policies, Admission Controllers та інструментів на основі eBPF забезпечує комплексний захист кластерів і створює передумови для побудови систем самоадаптивної кібербезпеки в контейнеризованих середовищах.

Подальший розвиток архітектури безпеки передбачає впровадження механізмів машинного навчання для прогнозування потенційних атак і адаптацію політик безпеки до змін середовища в режимі реального часу.

Перелік використаних джерел:

1. Kubernetes Documentation – Security Overview. [Електронний ресурс]. Режим доступу: https://kubernetes.io/docs/concepts/security/

2. Kyverno Policy Engine for Kubernetes. [Електронний ресурс]. Режим доступу: https://kyverno.io/

3. Falco Runtime Security Project. [Електронний ресурс]. Режим доступу: https://falco.org/

4. OPA Gatekeeper – Policy Controller for Kubernetes. [Електронний ресурс]. Режим доступу: https://openpolicyagent.org/

5. HashiCorp Vault – Secrets Management. [Електронний ресурс]. Режим доступу: https://www.vaultproject.io/

УДК 004.62

Сергій КУЛИНА

Західноукраїнський національний університет

АНАЛІЗ ЕФЕКТИВНОСТІ ГОМОМОРФНОГО ШИФРУВАННЯ ДЛЯ ЗАХИЩЕНИХ ХМАРНИХ ОБЧИСЛЕНЬ

Вступ. Швидке зростання обсягів даних та необхідність їх обробки призводить до широкого застосування хмарних технологій, зокрема хмарних обчислень. Проте, передача конфіденційної інформації не довіреній стороні, якою виступають постачальники хмарних послуг створює значні ризики конфіденційності даних. Традиційні методи шифрування захищають дані лише під час передачі та зберігання, вимагаючи розшифрування для обчислень, що є слабким місцем захисту даних. Для подолання цієї фундаментальної проблеми застосовується концепція гомоморфного шифрування (HE). Цей криптографічний алгоритм дозволяє виконувати математичні операції безпосередньо над зашифрованими даними, при цьому результат обчислення залишається зашифрованим і відповідає результату, який був би отриманий при обчисленні над відкритими даними. Це забезпечує повну конфіденційність даних навіть під час їх обробки в небезпечному середовищі, наприклад, у публічній хмарі.

Метою дослідження є аналіз існуючих проблем хмарних обчислень та можливості їх вирішення шляхом застосування гомоморфного шифрування.

1. Парадигма захищених хмарних обчислень

Сучасна екосистема інформаційних технологій переживає фундаментальну трансформацію, зумовлену масовою міграцією обчислювальних потужностей та сховищ даних у хмарні середовища. Хмарні обчислення забезпечують безпрецедентну масштабованість, гнучкість та економічну ефективність, дозволяючи організаціям зосередитися на бізнес–логіці, делегуючи інфраструктурні завдання спеціалізованим провайдерам (рис. 1).

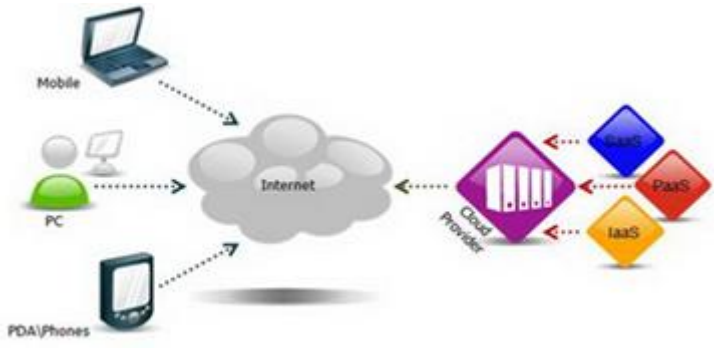


Рисунок 1 – Схема підключення користувачів до хмарних сервісів

Проте ця парадигма породжує критичну дилему довіри та безпеки, оскільки для виконання будь–яких операцій над даними хмарний провайдер повинен мати до них доступ у відкритому вигляді. Це створює можливість для атак, а конфіденційна інформація стає вразливою до внутрішніх загроз, компрометації серверів або несанкціонованого доступу з боку державних структур [1].

Традиційні методи криптографічного захисту ефективно вирішують проблему захисту даних у стані спокою та під час передачі, однак вони виявляються безсилими, коли мова йде про захист даних під час операцій. Як тільки дані завантажуються для обробки або передаються на віддалений ресурс для обчислень, вони повинні бути розшифровані, що порушує конфіденційність.

2. Методи застосування гомоморфного шифрування

У контексті захисту конфіденційних даних при передачі на обчислення повністю гомоморфне шифрування (FHE) виступає як технологія, що здатна докорінно змінити ситуацію. FHE дозволяє виконувати довільні обчислювальні операції над зашифрованими даними без необхідності доступу до секретного ключа розшифрування. Результат таких обчислень залишається у зашифрованому вигляді, і лише власник ключа може перетворити його на зрозумілий результат, який математично ідентичний результату, отриманому при обробці відкритих даних [4]. Це відкриває шлях до реалізації концепції "конфіденційних хмарних обчислень", де хмарний сервер виконує роль "сліпого" обчислювача, не отримуючи жодної інформації про зміст даних, які він обробляє.

Серед широкого спектру запропонованих криптосистем FHE, є дві схеми, що базуються на проблемі навчання з помилками в кільцях (RLWE). Вони стали де–факто стандартами для різних класів обчислювальних задач.

Схема Brakerski/Fan–Vercauteren (BFV) орієнтована на точну модульну арифметику над цілими числами, що робить її ідеальною для криптографічних протоколів та фінансових транзакцій [6].

Натомість схема Cheon–Kim–Kim–Song (CKKS) пропонує революційний підхід до наближених обчислень над дійсними та комплексними числами, емулюючи поведінку арифметики з плаваючою комою, що є критично важливим для сучасних алгоритмів машинного навчання та статистичного аналізу [4].

Висновок. Проведене дослідження показало, що розвиток хмарних обчислень залежить від реалізації захисту даних при передачі сторонньому користувачу. В такій ситуації гомоморфне шифрування є оптимальним, оскільки закритий ключ для розшифрування має у власності тільки законний користувач. Результати роботи з зашифрованими даними в хмарній інфраструктурі повністю відповідають діям проведеним над незашифрованими даними на пристрої користувача, проте завдяки більшій потужності обчислюються значно швидше.

Перелік використаних джерел.

1. Gao C. X., Wang Y., Huang L. Privacy–preserving railway data sharing: A comparative study of homomorphic encryption schemes // Proceedings of The International Conference on Electronic Business: Zhuhai, China, 2024 P. 329–337.

2. Wiryen, Y. B., Vigny, N. W., Joseph, M. N., & Aimé, F. L. (2024). A Comparative Study of BFV and CKKs Schemes to Secure IoT Data Using TenSeal and Pyfhel Homomorphic Encryption Libraries. International Journal of Smart Security Technologies (IJSST), 10(1), 1–17. https://doi.org/10.4018/IJSST.333852

3. Balch, T., Diamond, B. E., & Polychroniadou, A. (2020). SecretMatch: Inventory matching from fully homomorphic encryption. In Proceedings of the First ACM International Conference on AI in Finance (pp. 1–7).

УДК 004.5

Олександр КУХАРУК

Західноукраїнський національний економічний університет

РИЗИКИ ТА ВРАЗЛИВОСТІ У СМАРТ–КОНТРАКТАХ

Вступ. Смарт–контракти стали одним з найперспективніших інноваційних рішень в епоху цифрової трансформації. Вони є самовиконуваними угодами з умовами, закодованими безпосередньо в програмному коді, що працює на блокчейн–платформах, зокрема Ethereum. Однак зростання їх популярності супроводжується збільшенням кількості безпекових інцидентів. За даними ImmuneFi, у 2023 році через вразливості в смарт–контрактах було втрачено понад 1,8 мільярда доларів США. Це свідчить про нагальну потребу в системному дослідженні ризиків та розробці ефективних механізмів захисту.

Незважаючи на технологічні переваги блокчейну – децентралізацію, прозорість та незмінність – ці ж характеристики ускладнюють виправлення помилок у смарт–контрактах після їх розгортання. Існує суттєвий розрив між швидкістю впровадження технології та зрілістю механізмів її безпеки.

Мета. Систематизувати основні типи ризиків та вразливостей у смарт–контрактах та запропонувати комплексні підходи до їх мінімізації.

1. Аналіз предметної області та сучасних підходів до забезпечення безпеки смарт–контрактів

Смарт–контракти є ключовим елементом екосистеми блокчейн–технологій, забезпечуючи автоматизацію та децентралізоване виконання угод між сторонами без необхідності довіреної третьої особи. Незважаючи на високий потенціал та широке застосування, смарт–контракти мають низку притаманних ризиків і вразливостей, які обумовлені як особливостями програмування, так і специфікою децентралізованих систем. Аналіз предметної області дає змогу визначити, які загрози є найпоширенішими та які методи використовуються для їхнього усунення. Смарт–контракти характеризуються незмінністю після публікації в блокчейн–мережі. Це означає, що помилки коду практично неможливо виправити без масштабних оновлень або форків. Саме цей фактор робить вразливості в коді критичними. Серед основних проблем варто виділити:

- недостатня кваліфікація розробників у сфері безпеки;
- складність аудиту та тестування децентралізованих систем;
- відсутність уніфікованих стандартів безпеки;
- високий рівень автоматизації процесів, що підвищує наслідки помилок.

Типові помилки у смарт–контрактах часто пов'язані з неправильним використанням логіки доступу, некоректним обробленням зовнішніх викликів або помилками у математиці (переповнення, неправильно пораховані значення).

Сьогодні існує низка технологій і методів, що застосовуються для забезпечення безпеки смарт–контрактів:

1. Статичний аналіз коду – використовується для автоматизованого пошуку поширених помилок без виконання програми. Приклади інструментів: Slither,

Mythril, Oyente.

2. Формальна верифікація – метод доказу коректності програми відносно математичної моделі. Хоча процес складний, він дозволяє гарантувати логічну правильність критично важливих функцій.

3. Тестування та fuzzing – fuzz–тестування досліджує роботу контракту під дією непередбачуваних вхідних даних. Це дозволяє виявити приховані помилки.

4. Аудит безпеки – професійні команди аудиторів проводять глибокий аналіз коду, логіки та взаємодій смарт–контрактів. Це найбільш надійний спосіб виявити помилки, але і найдорожчий.

Таблиця 1 – Основні типи вразливостей у смарт–контрактах

Тип вразливості	Опис	Приклад наслідків
Re–entrancy	Повторний виклик функції до завершення попереднього	Крадіжка коштів (DAO Hack)
Integer Overflow /Underflow	Неправильні обчислення через переповнення змінних	Помилкові баланси і операції
Неправильні права доступу	Некоректне використання ролей або модифікаторів	Неавторизований доступ до функцій
Timestamp dependence	Використання часу як джерела випадковості	Маніпуляції майнерами
External call issues	Виклик сторонніх контрактів без перевірки	Зміна стану системи ззовні

2. Розробка моделі підвищення безпеки смарт–контрактів

У цьому розділі пропонується концепція системного підходу до забезпечення безпеки смарт–контрактів, яка ґрунтується на багаторівневій моделі аналізу, тестування та верифікації. Запропонована модель допомагає зменшити ризики експлуатаційних помилок та підвищити надійність програмних компонентів. Концептуальна модель оцінювання ризиків складається з чотирьох рівнів (рисунок 1):

Рисунок 1 – Узагальнена схема моделі безпеки смарт–контракту

1. Аналіз архітектури та логіки контракту – включає перевірку бізнес–логіки, визначення критичних функцій, ролей та взаємодій з іншими контрактами.

2. Статичний аналіз та формальна верифікація – застосування інструментів автоматизованої перевірки вразливостей.

3. Модульне тестування та fuzzing – дослідження поведінки контракту під різними сценаріями.

4. Аудит та симуляція експлуатаційних сценаріїв – перевірка контракту сторонніми експертами та моделювання можливих атак.Алгоритм оцінювання безпеки смарт–контракту:

– Аналіз вимог та опис функціоналу.

– Визначення критичних точок та потенційних векторів атаки.

– Запуск статичного аналізу та усунення знайдених помилок.
– Застосування fuzz–тестування та генерації випадкових даних.
– Проведення модульних тестів та тестів інтеграції.
– Формальна верифікація ключових функцій.
– Проведення аудиту безпеки.
– Публікація контракту у тестову мережу.
– Моніторинг поведінки контракту після розгортання.

Таблиця 2 – Порівняння ефективності методів забезпечення безпеки

Метод	Ефективність	Вартість	Застосування
Формальна верифікація	Висока	Висока	Критичні контракти
Аудит безпеки	Дуже висока	Середня/висока	Масові проєкти
Статичний аналіз	Середня	Низька	Початкова перевірка
Fuzzing	Висока	Середня	Виявлення прихованих помилок

Висновок. У ході дослідження проведено аналіз предметної області та визначено ключові проблеми, пов'язані з ризиками та вразливостями у смарт–контрактах. Встановлено, що найбільш поширеними є логічні помилки, зовнішні виклики та неправильні права доступу. Розглянуто сучасні інструменти забезпечення безпеки та підходи, які застосовуються у сфері блокчейн–розробки.

У другому розділі запропоновано модель комплексної оцінки безпеки смарт–контрактів, що включає аналіз архітектури, статичний аналіз, тестування та аудит. Запропоновані методи дозволяють значно підвищити рівень надійності та мінімізувати ризики експлуатаційних атак, що є важливим для розвитку безпечних децентралізованих рішень у майбутньому.

Перелік використаних джерел:

1. Buterin V. A Next–Generation Smart Contract and Decentralized Application Platform: Ethereum White Paper. 2014.

2. Wood G. Ethereum: A Secure Decentralised Generalised Transaction Ledger – Yellow Paper. 2023.

3. Atzei N., Bartoletti M., Cimoli T. A Survey of Attacks on Ethereum Smart Contracts (SoK). In: Proceedings of the 6th International Conference on Principles of Security and Trust (POST). 2017.

4. Luu L., Chu D.–H., Olickel H., Saxena P., Hobor A. Making Smart Contracts Smarter. ACM Conference on Computer and Communications Security (CCS). 2016.

5. Delmolino K., Arnett M., Kosba A., Miller A., Shi E. Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab. 2016.

7. Torres C. F., Steichen M. The Art of The Scam: Demystifying Honeypots in Ethereum Smart Contracts. 2019.

8. Solidity Documentation. Solidity Programming Language – Version 0.8+. [Електронний ресурс].– Режим доступу: https://docs.soliditylang.org

9. ConsenSys Diligence. Smart Contract Best Practices. [Електронний ресурс].– Режим доступу: https://consensys.github.io/smart–contract–best–practices/

10. Bhargavan K., Delignat–Lavaud A., Fournet C. та ін. Formal Verification of Smart Contracts: Short Paper. ACM PLAS. 2016.

УДК 004.056.5:004.75:004.738.5

Іванна МЕЛЬКО, Ігор ІГНАТЄВ

Західноукраїнський національний університе

РОЗРОБКА ПРОТОТИПУ СИСТЕМИ КЕРУВАННЯ ДОСТУПОМ У БАЗІ ДАНИХ ІЗ ФУНКЦІОНАЛЬНИМ ШИФРУВАННЯМ

Вступ. У роботі розглянуто підхід до побудови системи керування доступом до бази даних із використанням функціонального шифрування (Functional Encryption, FE). Такий метод дозволяє забезпечити селективний доступ до зашифрованих даних без необхідності їх повного розшифрування. Це підвищує рівень безпеки при роботі з конфіденційною інформацією в корпоративних та хмарних середовищах.

Традиційні системи шифрування не дозволяють виконувати запити над зашифрованими даними без розкриття всього вмісту. Це створює ризики несанкціонованого доступу. Натомість функціональне шифрування дає змогу користувачу отримати лише обчислене значення певної функції над зашифрованими даними, не розкриваючи їх повністю.

Мета дослідження розробити прототип системи керування доступом у базі даних із застосуванням функціонального шифрування, який забезпечує виконання запитів над зашифрованими даними відповідно до прав користувача.

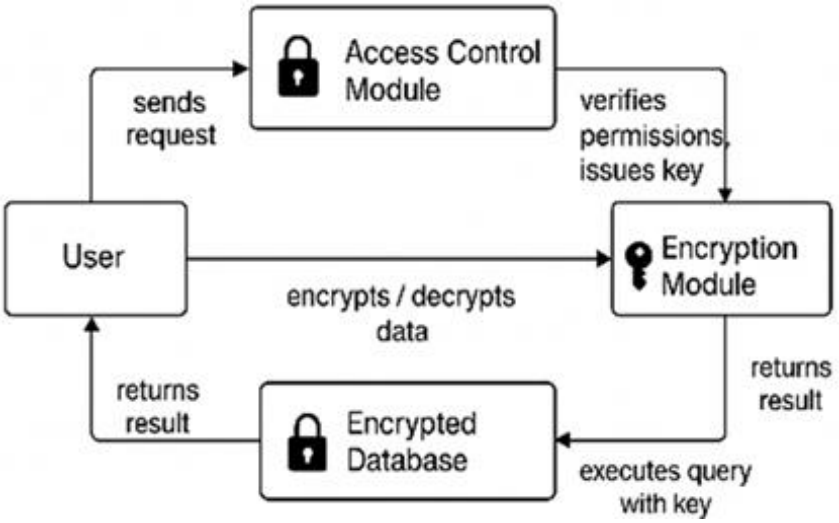


Рисунок 1 – Схема обробки HTTP–запиту з ML–детектором SQL–ін’єкцій

Для реалізації прототипу розроблено архітектуру, що включає такі компоненти:

– Модуль аутентифікації та контролю доступу – перевіряє права користувача та видає функціональний ключ.

– Модуль шифрування – здійснює шифрування та розшифрування даних із використанням бібліотеки Charm–Crypto.

– Зашифрована база даних – зберігає дані у вигляді зашифрованих записів (PostgreSQL).

– API–рівень обробки запитів – приймає запити користувачів і передає їх на виконання з відповідним ключем доступу.

Для формалізації процесу доступу використано наступне позначення:

$$f(x) = Decskf\big(Encpk(x)\big),$$

де $Encpk(x)$ – зашифровані дані,

$Decskf$ – часткове розшифрування за функціональним ключем

$f(x)$ – результат дозволеної функції над даними.

В таблиці 1 наведено показники ефективності виконання запитів у прототипі системи

Таблиця 1 – Ефективність виконання запитів у прототипі системи

Тип запиту	Середній час виконання	Рівень безпеки
Простий SELECT	2.3 мс	Високий
Агрегований SUM	3.5 мс	Високий
Обмежений доступ (FE)	4.1 мс	Дуже високий

Розроблений прототип показав, що застосування функціонального шифрування забезпечує високу безпеку доступу при незначному впливі на продуктивність системи. Під час тестування спостерігалося лише незначне збільшення часу обробки запитів у порівнянні з традиційними методами симетричного шифрування, що підтверджує можливість його використання у реальних корпоративних і хмарних середовищах.

Розроблена система підтримує гранульовану модель доступу, яка дозволяє визначати права користувача не лише на рівні таблиць чи атрибутів, а й на рівні функцій, що можуть бути виконані над зашифрованими даними. Це значно підвищує гнучкість та безпечність управління інформацією.

Висновок.

Запропонована система продемонструвала практичну реалізованість підходу функціонального шифрування для баз даних і довела, що обробка зашифрованих запитів можлива без повного розкриття інформації. Прототип має модульну архітектуру, що спрощує інтеграцію в існуючі системи керування базами даних.

Надалі планується розширення функціональності для підтримки багатокористувацького доступу, реалізація механізму ротації ключів і оптимізація криптографічних операцій для підвищення ефективності роботи системи.

Перелік використаних джерел.

1. Boneh D., Sahai A., Waters B. Functional Encryption: Definitions and Challenges. Theory of Cryptography Conference (TCC 2011).– Springer, Berlin, Heidelberg, 2011. – P. 253–273.

2. Abdalla M., Bourse F., De Caro A., Pointcheval D. Simple Functional Encryption Schemes for Inner Products. IACR Cryptology ePrint Archive, 2015:017.

3. Charm–Crypto Library. [Електронний ресурс].– Режим доступу: https://github.com/JHUISI/charm

4. PostgreSQL Global Development Group. PostgreSQL Documentation. [Електронний ресурс].– Режим доступу: https://www.postgresql.org/docs/

УДК 004.056.53

Іван МУДРИЙ, Людмила БАБАЛА

Західноукраїнський національний університет

ПОРІВНЯЛЬНИЙ АНАЛІЗ МЕТОДІВ БІОМЕТРИЧНОЇ АВТЕНТИФІКАЦІЇ НА ОСНОВІ КРИТЕРІЮ ВІДНОСНОЇ ЕНТРОПІЇ

Вступ. Біометричні системи ідентифікації особистості на основі фізіологічних або поведінкових характеристик набули широкого розповсюдження в різних областях забезпечення безпеки – від контролю доступу до правоохоронних об'єктів.

Біометричні характеристики є унікальними для кожної людини і не можуть бути вкрадені, підроблені або забуті на відміну від паролів чи PIN–кодів. Актуальною залишається задача вибору оптимального методу біометричної автентифікації на основі об'єктивних критеріїв оцінювання.

Мета дослідження: розробити методику порівняльного аналізу біометричних систем на основі критерію відносної ентропії та визначити найбільш інформативні джерела біометричних даних для надійної автентифікації особистості.

1. Алгоритм обчислення біометричної інформації

Біометрична інформація визначається як зменшення невизначеності ідентичності людини за рахунок вимірювання набору біометричних характеристик.

Найкращою мірою для характеристики біометричної інформації є відносна ентропія $D(p||q)$, або відстань Кульбака–Лейблера, яка визначається як «додаткові біти» інформації, необхідні для подання розподілу характеристик людини p(x) відносно розподілу характеристик населення $q(x)$ [1]:

$$D(p||q) = \int p(x) log_2(p(x)/q(x))dx \quad (1)$$

Розроблений алгоритм включає п'ять основних етапів: висування вимог до біометричної інформації, обчислення відносної ентропії біометричних характеристик, застосування Гаусової моделі для обчислення характеристик і відносної ентропії, методи регуляризації для вироджених характеристик, методи регуляризації для неповних даних [2].

Основні вимоги до особливостей біометричної інформації визначають, що:

– якщо розподіл характеристик людини дорівнює розподілу характеристик між людьми, біометрична інформація дорівнює нулю;

– при збільшенні точності вимірювань біометрична інформація зростає;

– незвичайні значення характеристик збільшують біометричну інформацію;

– інформація про некорельовані характеристики є адитивною.

Для біометричної системи з S характеристиками створюється вектор біометричних характеристик $x(S \times 1)$ для кожної людини.

Обчислюються середні значення μ та матриці коваріації Σ для людини (p) і населення (q).

Для подолання проблеми вироджених характеристик застосовується метод головних компонент (PCA) з декомпозицією єдиного значення (SVD):

$$US_qU^T = svd(cov(X)) = svd(\Sigma_q) \quad (2)$$

де U – ортогональна матриця власних векторів,

S_q – діагональна матриця власних значень [3].

Використовуючи базу даних Aberdeen (18 зображень кожної з 16 осіб, розмір 150×200 пікселів), обчислено PCA компоненти та лінійні дискримінанти Фішера (FLD). Для 100 найбільш вагомих характеристик розраховано відносну ентропію.

Для PCA компонент середня біометрична інформація

$$D(p||q) = 45 \text{ біт},$$

для FLD дискримінант

$$D(p||q) = 37 \text{ біт},$$

для ICA компонент

$$D(p||q) = 39 \text{ біт}.$$

Сумарна середня біометрична інформація для PCA та FLD компонент становить

$$D(p||q) = 55{,}6 \text{ біт [4]}.$$

Характерною особливістю є поступове зменшення біометричної інформації після другої головної компоненти для PCA, що пояснюється природою декомпозиції: вищі номери характеристик відповідають більш високим частотам деталей і містять більше шуму.

Для дослідження використовувалась база даних CASIA (689 зображень райдужних оболонок від 109 осіб, 6–7 зображень кожної людини). Обчислено 327 PCA та ICA компонент.

Середня біометрична інформація райдужної оболонки ока для PCA компонент складає 278 біт, для ICA – 288 біт [5].

Кількість біометричної інформації для ICA і PCA компонент є дуже близькою, причому ICA компоненти містять дещо більше інформації через кращу відповідність моделі характеристичних даних райдужки.

Результати порівняння методів біометричної автентифікації показують суттєву перевагу розпізнавання за райдужною оболонкою ока над розпізнаванням за обличчям (рисунок 1).

Райдужна оболонка містить у 5–6 разів більше біометричної інформації (278–288 біт проти 45–55 біт для обличчя), що забезпечує значно вищу надійність автентифікації.

Отримані результати узгоджуються з попередніми дослідженнями: Daugman заявляв про комбінаторну складність фазової інформації райдужної оболонки приблизно 2^{49} ступенів свободи, Cover та Thomas розрахували біометричну інформацію 241 біт для райдужної оболонки діаметром 11 мм [5].

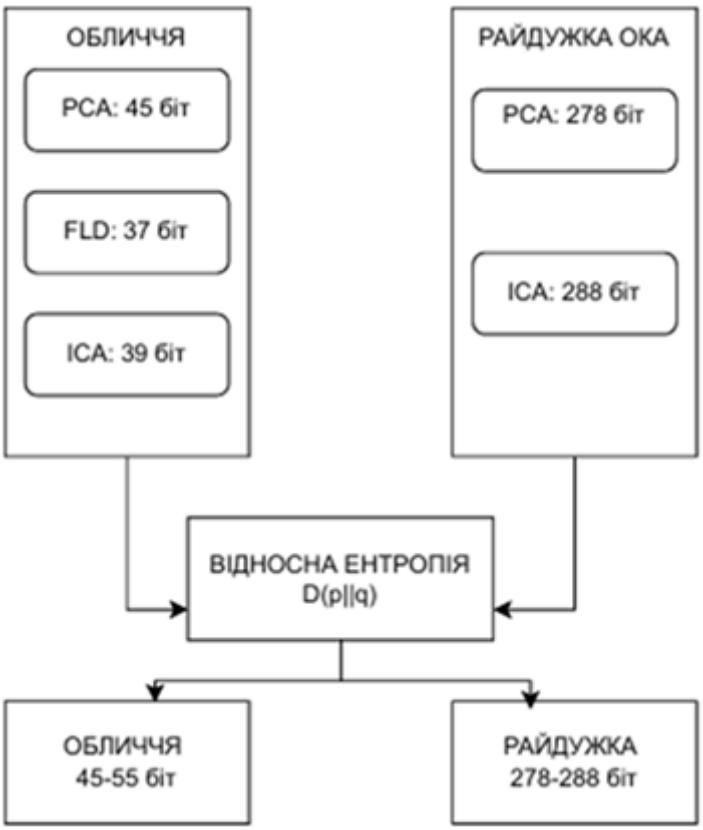


Рисунок 1 – Порівняльний аналіз біометричних методів автентифікації

Висновок. Розроблена методика оцінювання біометричної інформації на основі критерію відносної ентропії дозволяє об'єктивно порівнювати різні методи біометричної автентифікації. Встановлено, що метод розпізнавання на основі райдужної оболонки ока має найбільшу ентропію і дозволяє найбільш надійно виконувати автентифікацію особи. Використання відносної ентропії як критерію ефективності робить можливим порівняння не тільки біометричних ознак між собою, але й з PIN–кодами, паролями та іншими методами автентифікації, що створює єдину метрику для оцінювання безпеки систем ідентифікації.

Перелік використаних джерел

1. Adler A., Youmaran R., Loyka S. Towards a measure of biometric feature information. Pattern Anal. Appl. 2009. № 12(3). P. 261–270.

2. BS ISO/IEC 19794–6:2011 Information technology. Biometric data interchange formats. Iris image data. BSI, 2011. 30 p.

3. Alter O., Brown O., Botstein D. Singular value decomposition for genome–wide expression data processing and modeling. Proc Natl. Acad. Sci. 2000. Vol. 97. P. 10101–10106.

4. Draper B., Baek K., Bartlett M., Beveridge J. Recognizing faces with PCA and ICA. Computer Vision and Image Understanding. 2003. Vol. 91. P. 115–137.

5. Xiang C., Fan X.A., Lee T.H. Face recognition using recursive Fisher linear discriminant. Communications, Circuits and Systems. 2004. Vol. 2. P. 27–29.

УДК 004.056.5

Владислав ОСІДАК, Степан ІВАСЬЄВ

Західноукраїнський національний університет

ОНЛАЙН ЗАСОБИ ДИНАМІЧНОГО АНАЛІЗУ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Вступ. Актуальність теми "Онлайн засоби динамічного аналізу шкідливого програмного забезпечення" зумовлена стрімким зростанням кількості складних і добре замаскованих кіберзагроз. Сучасні шкідливі програми часто використовують техніки ухилення від виявлення, що робить традиційні методи недостатньо ефективними. Онлайн–платформи динамічного аналізу дозволяють швидко й ефективно досліджувати поведінку підозрілих файлів у контрольованому середовищі. Це робить їх важливим інструментом для оперативного реагування на загрози та покращення кіберзахисту.

Метою дослідження є вивчення можливостей та ефективності онлайн засобів динамічного аналізу для виявлення та дослідження шкідливого програмного забезпечення. Дослідження передбачає аналіз функціональності, переваг і недоліків популярних онлайн–платформ, а також оцінку їх практичного застосування в умовах реальних кіберзагроз. Особлива увага приділяється ролі цих засобів у сучасних системах кіберзахисту та швидкому реагуванні на інциденти.

1. Онлайн–служби для аналізу шкідливого програмного забезпечення.

VirusTotal – безкоштовна служба для аналізу підозрілих файлів та посилань. Подання не потребує.

Intezer – детектор зловредів, що підтримує динамічний та статичний аналіз.

Triage – онлайн–сервіс для аналізу великих обсягів шкідливого програмного забезпечення з функцією статичного аналізу зразків.

FileScan.IO – служба аналізу шкідливих програм, з динамічним аналізом та функцією вилучення індикаторів компрометації (IOC).

Sandbox.pikker – онлайн–версія відомої системи аналізу шкідливих програм Cuckoo Sandbox. Надає докладний звіт з описом поведінки файлу під час виконання у реалістичному, але ізольованому середовищі далеко у хмарі.

Manalyzer – безкоштовний сервіс для статичного аналізу PE–файлів та виявлення маркерів небажаної поведінки. Має офлайн–версію.

Opswat – сканує файли, домени, IP–адреси та хеші за допомогою технології Content Disarm & Reconstruction.

InQuest Labs – сервіс для сканування текстових документів файлів Microsoft та Open Office, електронних таблиць та презентацій. Працює з урахуванням механізмів Deep File Inspection (DFI).

Any Run – ще одна онлайн–пісочниця з гарним інтерфейсом та додатковими опціями.

Yoroi – італійська служба аналізу підозрілих файлів на базі пісочниці. Перетравлює PE (наприклад, .exe–файли), документи (doc і PDF), файли сценаріїв (типу wscript, Visual Basic) та APK, але болісно повільно готує звіти.

Unpacme – онлайн–сервіс для автоматичного розпакування шкідливих програм та вилучення артефактів.

Malwareconfig – веб-додаток для вилучення, декодування та відображення параметрів конфігурації найпоширеніших шкідливих програм.

Malsub – фреймворк Python RESTful для роботи з API онлайн–сервісів для аналізу шкідливого ПЗ.

2. Аналіз можливостей онлайн засобів виявлення шкідливого програмного забезпечення.

Для тестування засобів динамічного аналізу ШПЗ створимо невеликий скрипт мовою python, котрий копіює себе в тимчасову папку та додає в автозавантаження. Скрипт наведений на рисунку 1.

```python
import os
import shutil
import sys
import winreg

def add_to_startup(filepath=None, name="MyPythonApp"):
    if filepath is None:
        filepath = sys.executable  # Поточний виконуваний файл (у .exe або .py)

    key = winreg.HKEY_CURRENT_USER
    reg_path = r"Software\Microsoft\Windows\CurrentVersion\Run"

    try:
        reg_key = winreg.OpenKey(key, reg_path, 0, winreg.KEY_SET_VALUE)
    except FileNotFoundError:
        reg_key = winreg.CreateKey(key, reg_path)

    winreg.SetValueEx(reg_key, name, 0, winreg.REG_SZ, filepath)
    winreg.CloseKey(reg_key)
```

```python
def copy_to_temp():
    temp_dir = os.environ.get("TEMP")
    target_path = os.path.join(temp_dir, "MyPythonApp.exe")

    if not os.path.exists(target_path):
        shutil.copy(sys.argv[0], target_path)
    return target_path

if __name__ == "__main__":
    copied_path = copy_to_temp()
    add_to_startup(filepath=f'python "{copied_path}"')
```

Рисунок 1 – Тестовий скрипт

Спочатку проведемо дослідження за допомогою сервісу https://tria.ge/. Triage – це потужне рішення для автоматизованого аналізу шкідливих програм, розроблене компанією Hatching (ізазначено, що команда створювала Cuckoo Sandbox). Платформа підтримує масштаб до 500 000 аналізів на добу, що робить її оптимальним інструментом для великих організацій і SOC/CERT команд.

На рисунку 2 приведено процес завантаження файлу та вибір віртуальної машини.

Triage дозволяє динамічно досліджувати зразки на таких операційних системах: Windows 7 та 10, Linux, macOS, Android .

Додатково є можливість інтерактивної взаємодії з віртуальною машиною у реальному часі, включаючи ручне введення дій або огляд виконання зразка “вживу” .

Triage автоматично проводить статичний аналіз файлу, видобуває конфігурації шкідливого ПЗ, та генерує динамічні behavioral–оцінки (signature-based, мережеві контакти, доменні репутації), які об’єднуються в загальну оцінку загрози.

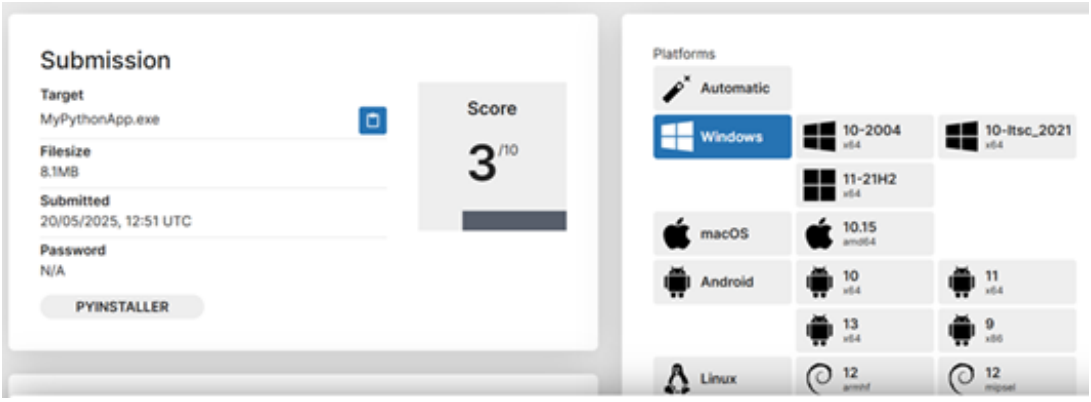


Рисунок 2 – Процес завантаження файлу та вибір віртуальної машини

Інтерфейс з фокусом на найважливішу інформацію: труєтвінг файла, конфігурації, потенційна поведінка, натомість для автоматизації доступна REST API та профілі збереження налаштувань (timeout, інтернет, VM тип) .Результати тестування та роботи програмного засобу можна побачити на рисунку 3. Якщо були якісь повідомлення та виводи на екран, система надасть скрін з екрану в звіт.

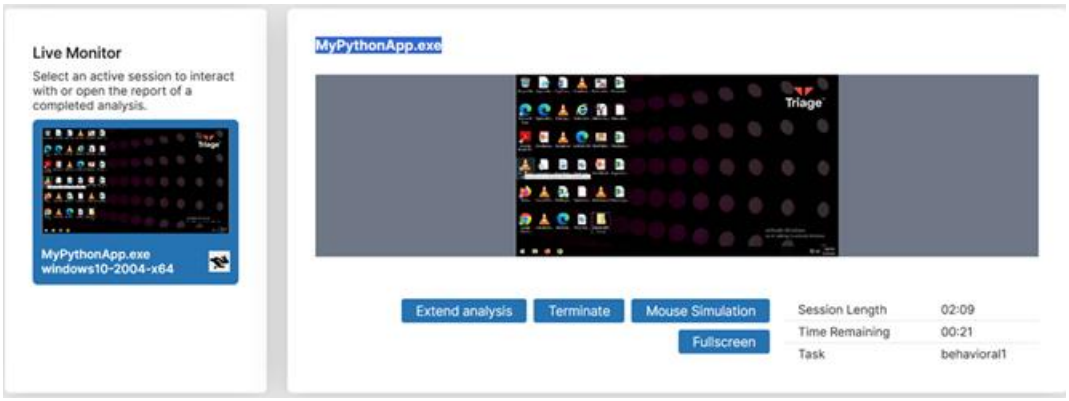


Рисунок 3 – Зображення отримані в результаті виконання програмного засобу

Також автоматизована пісочниця надає звіт про системні дії програмного засобу: зміни в реєстрі, зміни в файловій системі, системні виклики, як це показано на рисунку 4.

Плюси Triage: висока масштабованість, підтримка багатьох ОС, інтерактивний live-режим, зручна UI та гнучке API для інтеграції з SOC, MSSP або IR-системами. Обмеження:немає підтримки Windows 11, усі результати аналізів у публічній версії є доступні спільноті, якщо не використовується корпоративний контракт.

Ще одним доступним та зручним засобом є https://www.hybrid–analysis.com/. Hybrid Analysis (що працює на базі CrowdStrike Falcon Sandbox) – це хмарна платформа для статично–динамічного аналізу шкідливого ПЗ, що дозволяє безкоштовно надсилати файли чи URL для дослідження. Основні можливості:

Комбінований аналіз: поєднує статичну перевірку (розбір бінарних структур), динамічне виконання у sandbox–оточенні з моніторингом змін пам'яті, реєстру та мережевих подій.

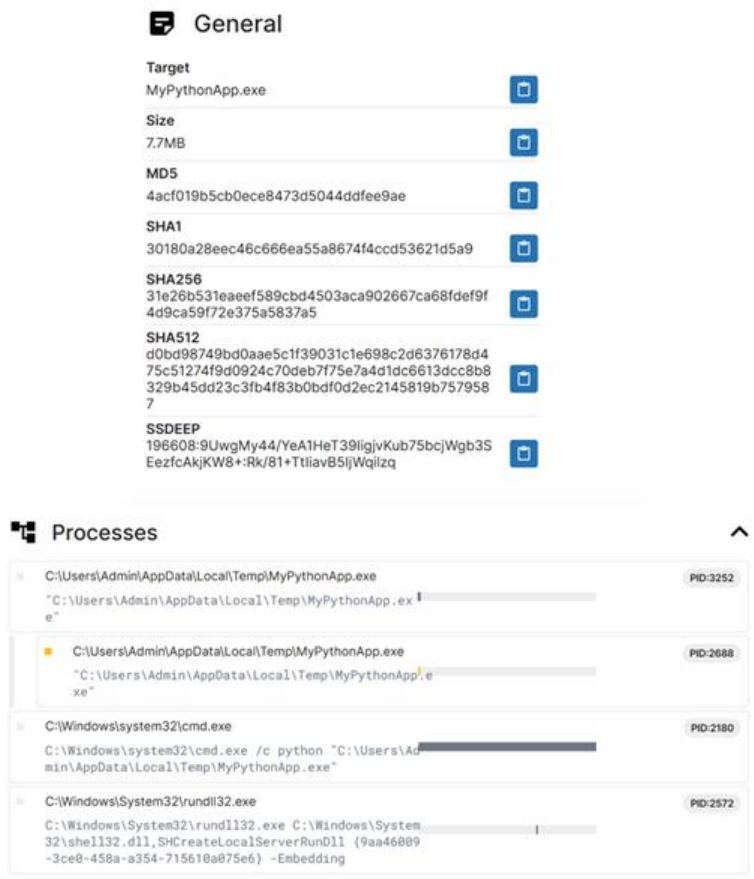


Рисунок 4 – Звіт про дії програмного засобу

Автоматично згенеровані IOCs: платформа збирає індикатори і включає їх до детальних звітів, доступних через UI чи API Hybrid Analysis.

Інтеграція Threat Intelligence: включає такі сервіси як Criminal IP, Bfore.Ai та CrowdStrike AI-модуль, що покращують аналіз доменів, URL і злочинних IP–адрес. Процес завантаження програмного засобу приведено на рисунку 5.

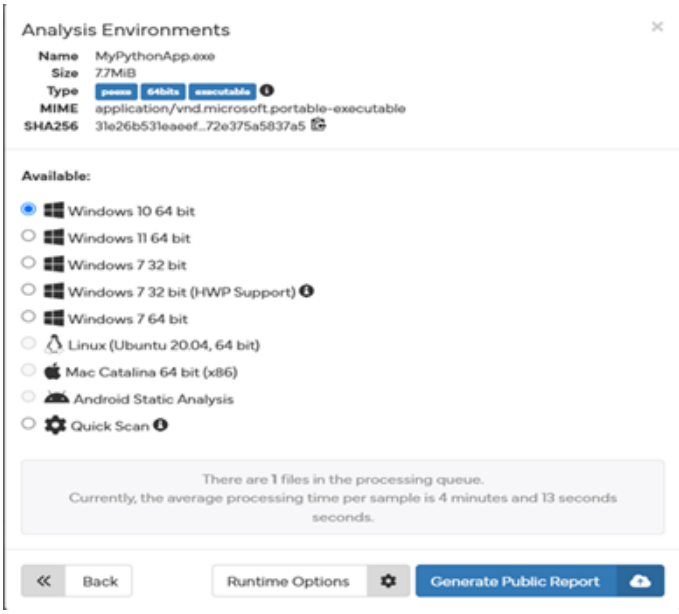


Рисунок 5 – Завантаження зразка для тесту в Hybrid Analysis

В результаті система сформує звіт по діях зразка та надасть оцінку ризику поведінковим аналізом, як це показано на рисунку 6.

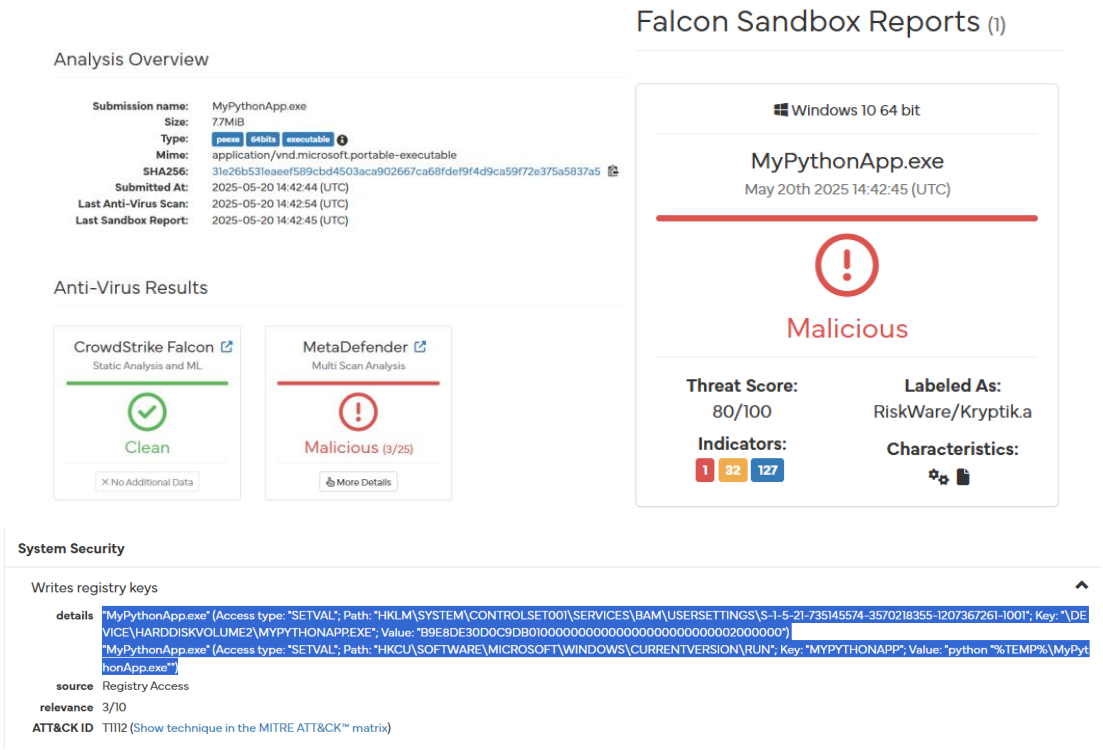


Рисунок 6 – Результат аналізу зразка Hybrid Analysis

Також доступний сервіс для динамічного аналізу ШПЗ https://www.virustotal.com/. VirusTotal – це потужна онлайн–платформа, яка поєднує як статичний, так і динамічний аналіз файлів та URL–адрес з метою виявлення шкідливого програмного забезпечення. Динамічний аналіз у VirusTotal виконується за допомогою внутрішніх sandbox–середовищ, таких як Jujubox для Windows, Droidy для Android та спеціалізованих рішень для macOS. У цих середовищах підозрілі файли запускаються у віртуальних машинах, де система фіксує їхню поведінку: створення нових процесів, зміну реєстру, доступ до мережі, спроби завантаження додаткових компонентів тощо.

Результати динамічного аналізу доступні у вигляді детального звіту, що містить індикатори компрометації (IOC), пов'язані домени та IP–адреси, а також мітки відповідності до тактик MITRE ATT&CK. Такі звіти дозволяють аналітикам швидко оцінити потенційну загрозу та встановити зв'язки з відомими зразками шкідливого ПЗ. VirusTotal також інтегрує зовнішню аналітику – наприклад, від Mandiant або CrowdStrike – що підсилює глибину поведінкової оцінки.

Безкоштовна версія сервісу забезпечує базовий доступ до результатів динамічного аналізу, але всі завантажені зразки стають публічними. У платній версії доступні приватні sandbox–аналізи, API–інтеграція, гнучке управління середовищем виконання та доступ до додаткових аналітичних функцій. Загалом, VirusTotal є універсальним інструментом для початкового та поглибленого аналізу загроз, який широко використовується як фахівцями з кібербезпеки, так і дослідниками з усього світу.

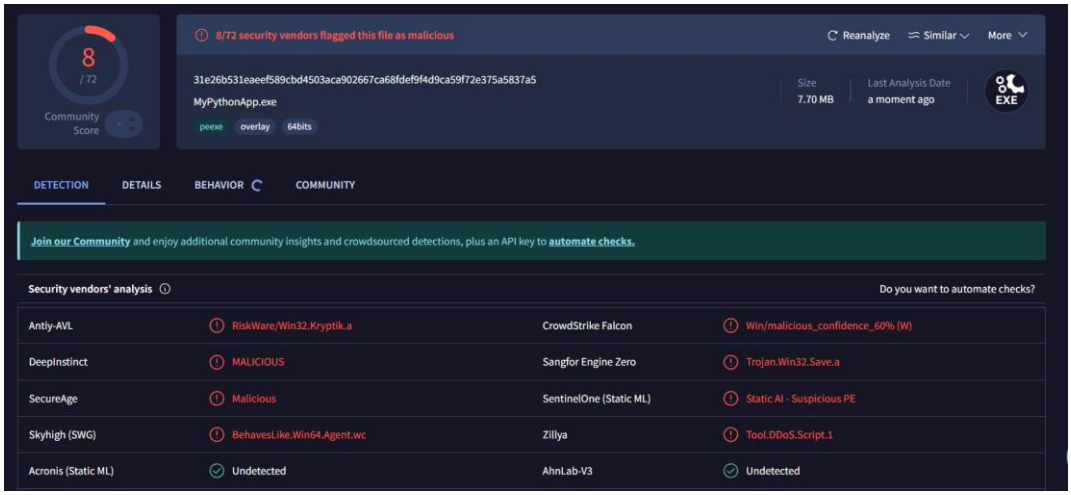


Рисунок 7 – Результати аналізу зразка за допомогою VirusTotal

Висновок. Онлайн засоби динамічного аналізу шкідливого програмного забезпечення є важливим компонентом сучасної системи кіберзахисту. Вони дозволяють швидко й безпечно досліджувати поведінку підозрілих програм у ізольованому середовищі, що значно підвищує ефективність виявлення нових та складних загроз.

Попри деякі обмеження, такі як залежність від інтернет–з'єднання чи обмежена функціональність безкоштовних сервісів, ці інструменти демонструють високу практичну цінність. У майбутньому їх розвиток і інтеграція з іншими аналітичними системами сприятимуть ще більш ефективному реагуванню на кіберінциденти.

Перелік використаних джерел.

1. Aslan Ö., Samet R., "Investigation of Possibilities to Detect Malware Using Existing Tools," 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), Hammamet, Tunisia, 2017, pp. 1277–1284, doi: 10.1109/AICCSA.2017.24.

2. Saurabh, "Advance Malware Analysis Using Static and Dynamic Methodology," 2018 International Conference on Advanced Computation and Telecommunication (ICACAT), Bhopal, India, 2018, pp. 1–5, doi: 10.1109/ICACAT.2018.8933769.

3. D. Kim, D. Mirsky, A. Majlesi–Kupaei and R. Barua, "A Hybrid Static Tool to Increase the Usability and Scalability of Dynamic Detection of Malware," 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), Nantucket, MA, USA, 2018, pp. 115–123, doi: 10.1109/MALWARE.2018.8659373.

4. Sinha A. K., Sai S., "Integrated Malware Analysis Sandbox for Static and Dynamic Analysis," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1–5, doi: 10.1109/ICCCNT56998.2023.10306805.

УДК 004.056

Дмитро ПЕРЕРВА

Західноукраїнський національний університет

УДОСКОНАЛЕНІ ПІДХОДИ ДО ЗМЕНШЕННЯ ВИТОКУ МЕТАДАНИХ У СИСТЕМАХ БЕЗПЕЧНОГО ОБМІНУ ПОВІДОМЛЕННЯМИ

Вступ. Питання захисту метаданих стало одним із центральних аспектів сучасних криптографічних досліджень. Навіть за умов використання надійних схем шифрування вмісту повідомлень, значний обсяг чутливої інформації може бути отриманий шляхом аналізу структурних характеристик трафіку: часу надсилання, обсягу переданих пакетів, частоти комунікації між певними користувачами або характеру маршрутизації. У багатьох випадках ці метадані дозволяють побудувати точні профілі користувачів, визначити соціальні зв’язки, ритм активності та інші особливості поведінки.

Тому дослідження моделей обміну повідомленнями, що забезпечують не лише конфіденційність вмісту, а й мінімальне розкриття службових ознак, стає важливою складовою підвищення цифрової безпеки. Особливе місце в таких системах посідають месенджери нового покоління, які застосовують розподілені мережеві архітектури, анонімні маршрутизатори, механізми затримок та рандомізації трафіку.

Мета. Аналіз сучасних підходів до зменшення витоку метаданих при передачі зашифрованих повідомлень та визначення технологій, які дозволяють ефективно маскувати структурні характеристики трафіку у децентралізованих засобах комунікації.

1. Метадані в захищених комунікаційних системах та їх загрози

До метаданих зазвичай відносять інформацію про факт надсилання повідомлення, тривалість взаємодії між користувачами, технічні характеристики пакетів, ідентифікатори маршрутів та інші службові елементи. Навіть за умов відсутності доступу до змісту, такі дані можуть бути використані для визначення структури соціальних графів або відстеження переміщень користувача. У традиційних схемах обміну даними основні ризики пов’язані з тим, що маршрутизація ідентифікаторів, часові позначки, а інколи і задіяні мережеві вузли піддаються спостереженню на стороні провайдера або зловмисника. Це створює передумови для атак на приватність, включно з кореляційним аналізом та пасивним моніторингом[1].

У низці сучасних месенджерів реалізовані складні підходи до приховування службових характеристик трафіку. Одним із найпоширеніших є маршрутизація через проміжні вузли, що застосовується в децентралізованих мережах. Принцип полягає у тому, що повідомлення передається не напряму, а через кілька незалежних серверів, що маскує справжнє джерело. Другим підходом є використання однакових за обсягом пакетів, де весь трафік штучно вирівнюється. Така схема не дозволяє встановити, чи передає користувач текст, файл або просто сигнальні дані.

На рисунку 1 наведено узагальнену схему формування трафіку з мінімізацією метаданих.

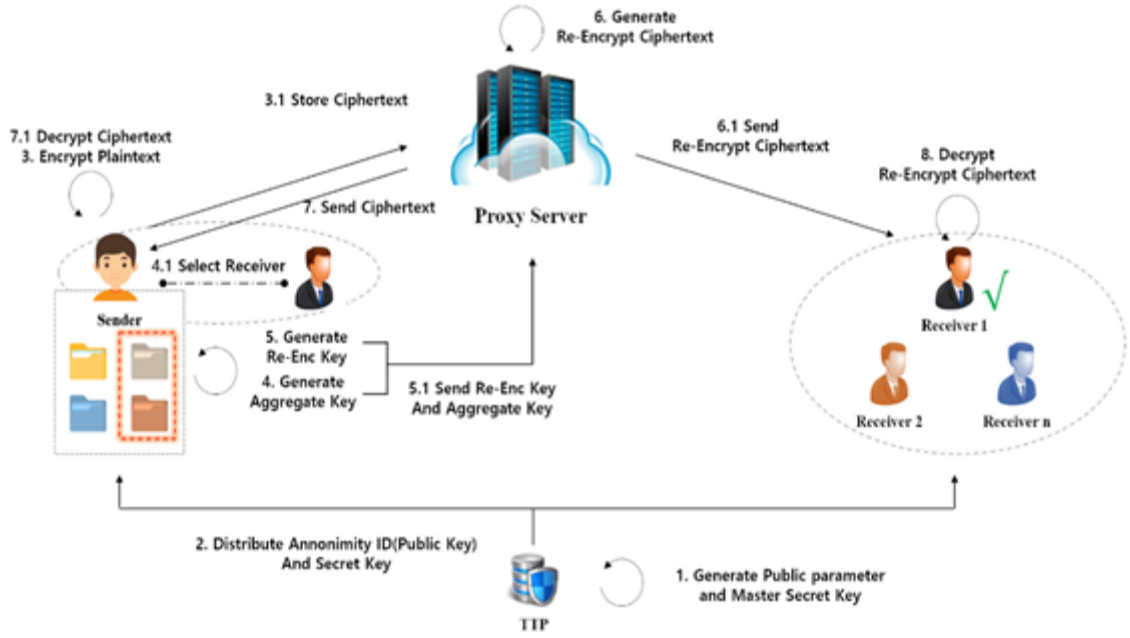


Рисунок 1 – Схема формування трафіку з механізмами маскування метаданих

У ряді протоколів застосовується так звана обов'язкова генерація фіктивного трафіку, коли навіть за відсутності активності надсилаються порожні зашифровані об'єкти. Це значно ускладнює визначення реальних подій. У деяких реалізаціях застосовуються затримки доставки, метою яких є руйнування часових кореляцій. Один із ефективних варіантів – введення випадкового діапазону затримки для кожного пакету. Це дозволяє уникати однотипних шаблонів у передаванні повідомлень [2].

2. Удосконалені стратегії зменшення витоку

Однією з найбільш прогресивних моделей, які орієнтовані на приватність, є архітектури, що не використовують класичних серверів зберігання даних. Замість цього вони базуються на розподілених мережах, де кожне повідомлення передається анонімним маршрутом без збереження інформації про взаємодію. Особливістю таких систем є відсутність номерів телефонів, централізованих профілів або інших ідентифікаторів, які могли б бути використані для аналізу активності. Центральна роль у таких мережах відводиться вузлам, що передають зашифрований трафік. Зовні ці пакети не містять жодних ідентифікаторів користувача. Сам трафік виглядає уніфіковано, незалежно від змісту чи типу інформації.

На рисунку 2 подано узагальнену архітектуру такого підходу.

Окрему увагу в сучасних дослідженнях приділяють підходам, пов'язаним з криптографічним приховуванням службових полів. До них належать:

– шифрування заголовків з використанням симетричних і асиметричних механізмів;

– упакування декількох повідомлень в один контейнер задля

маскування реальної частоти передачі;

– адаптивні протоколи, що коригують обсяг трафіку залежно від статистики середовища;

– – використання анонімних транспортних рівнів – наприклад, onion–routing або mixnet–підходів.

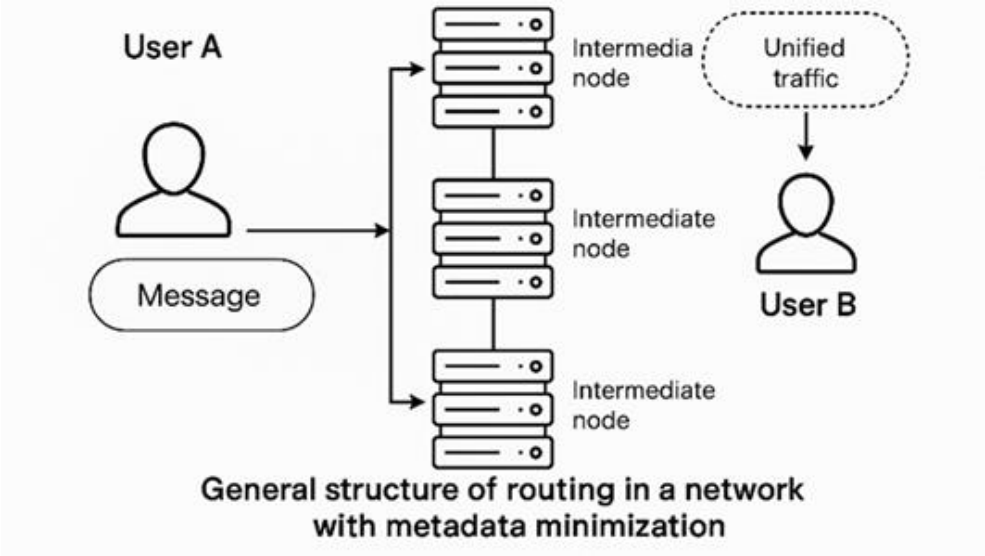


Рисунок 2 – Загальна структура маршрутизації в мережі з мінімізацією метаданих

Поєднання цих методів дозволяє суттєво підвищити приватність користувачів і формує єдину модель захищеного обміну, де навіть пасивне прослуховування не дає можливості простежити взаємозв'язки між учасниками [3].

Висновок. На основі розглянутих моделей можна стверджувати, що сучасні системи обміну повідомленнями дедалі більше орієнтуються на захист не лише змісту, але й структурних характеристик трафіку. Використання багаторівневих методів маскування метаданих, уніфікації пакетів, прихованої маршрутизації та затримок дозволяє суттєво знизити ризики аналізу взаємодій між користувачами. Подальші дослідження можуть бути спрямовані на оптимізацію продуктивності таких систем без зниження рівня приватності, а також на розширення моделей оцінювання ризиків у різних типах мережевого середовища.

Перелік використаних джерел.

1. Kwon A., Liu S., et al. Loopix: Practical Anonymous Messaging with Strong Metadata Protection. [Електронний ресурс]. – Режим доступу: https://www.usenix.org/conference/usenixsecurity20

2. Signal Protocol Documentation. [Електронний ресурс]. – Режим доступу: https://signal.org/docs/

3. Session Foundation. Session: Private MessagingProtocol. [Електронний ресурс]. – Режим доступу: https://getsession.org/whitepaper

УДК 004

Максим ПЕЧЕНЮК, Тарас ЦАВОЛИК

Західноукраїнський національний університет

БАГАТОРІВНЕВІ АРХІТЕКТУРИ БЕЗПЕКИ IOT: ПОРІВНЯЛЬНИЙ АНАЛІЗ ФРЕЙМВОРКІВ NIST, ISO/IEC 27400 ТА OWASP

Вступ. Стрімке зростання екосистеми Інтернету речей супроводжується критичною необхідністю стандартизації підходів до забезпечення інформаційної безпеки. За прогнозами аналітиків Cisco, кількість підключених IoT–пристроїв досягне 50 мільярдів одиниць, створюючи потенційний ринок обсягом понад 14 трильйонів доларів. Однак це зростання супроводжується критичним збільшенням кіберзагроз – за даними 2025 року, IoT–інфраструктура зазнає в середньому 820 тисяч спроб зламу щоденно, що становить зростання на 46% порівняно з попереднім роком. Відсутність уніфікованих стандартів безпеки та різноманітність підходів різних організацій до стандартизації створюють складнощі для виробників IoT–обладнання, розробників платформ та організацій, що експлуатують великомасштабні IoT–системи. Провідні міжнародні організації – Національний інститут стандартів та технологій США (NIST), Міжнародна організація зі стандартизації (ISO/IEC) та проєкт Open Web Application Security Project (OWASP) – розробили комплексні фреймворки безпеки IoT, кожен з яких має свої особливості, переваги та сфери застосування.

Мета: Провести систематичний порівняльний аналіз трьох провідних фреймворків безпеки IoT – NIST Cybersecurity for IoT Program, ISO/IEC 27400:2022 та OWASP IoT Top 10, визначити їх структурні особливості, рівень покриття загроз безпеки, застосовність для різних класів IoT–систем та можливості інтеграції для створення комплексного підходу до захисту IoT–інфраструктури.

1. Структурний аналіз фреймворків безпеки IoT

Національний інститут стандартів та технологій США (NIST) розробив комплексну програму кібербезпеки для IoT, яка підтримує створення стандартів, настанов та інструментів для покращення захисту IoT–систем. Програма NIST Cybersecurity for IoT Program була заснована наприкінці 2016 року з метою розвитку та застосування стандартів для IoT–систем та середовищ їх розгортання [1]. Ключовими публікаціями програми є NIST IR 8259A "IoT Device Cybersecurity Capability Core Baseline", що визначає базові можливості кібербезпеки для IoT–пристроїв, та NIST SP 800–213 "IoT Device Cybersecurity Guidance for the Federal Government", що надає керівництво федеральним агенціям щодо встановлення вимог до кібербезпеки пристроїв [2, 3].

Фреймворк NIST базується на п'яти основних принципах, що забезпечують системний підхід до безпеки IoT. Перший принцип – розуміння ризиків – передбачає застосування Risk Management Framework до IoT–специфічних ризиків з урахуванням обмежених ресурсів пристроїв та специфіки їх розгортання. Системний підхід, як другий принцип, вимагає розгляду IoT–пристрою в контексті всієї екосистеми, включаючи взаємодію з іншими пристроями,

мережевою інфраструктурою та хмарними сервісами. Адаптивність як третій принцип дозволяє налаштовувати базові вимоги під конкретні застосування, враховуючи різноманітність IoT–пристроїв від простих сенсорів до складних промислових контролерів [4].

NIST IR 8259A визначає шість категорій базових можливостей кібербезпеки для IoT–пристроїв. Ідентифікація пристрою включає унікальну ідентифікацію пристрою та його конфігурації для забезпечення відслідковуваності та управління активами. Захист пристрою охоплює механізми захисту від несанкціонованого доступу, включаючи логічний та фізичний захист, а також захист даних у стані спокою та при передачі. Виявлення подій безпеки передбачає можливості моніторингу та логування подій для виявлення аномалій та потенційних загроз. Реагування на події безпеки включає механізми автоматичного та ручного реагування на виявлені інциденти. Оновлення пристрою забезпечує можливості безпечного оновлення firmware та програмного забезпечення. Захист даних охоплює криптографічні механізми для забезпечення конфіденційності, цілісності та автентичності даних [5].

Міжнародний стандарт ISO/IEC 27400:2022 "Cybersecurity – IoT security and privacy – Guidelines" надає комплексні настанови щодо ризиків, принципів та контролів безпеки і конфіденційності для IoT–рішень [6]. Стандарт визначає три ключові ролі зацікавлених сторін: розробник IoT–сервісів (IoT Service Developer), постачальник IoT–сервісів (IoT Service Provider) та користувач IoT (IoT User). ISO/IEC 27400 містить 45 контролів безпеки та конфіденційності, структурованих у дві основні категорії: 28 контролів для забезпечення безпеки та 17 контролів для захисту конфіденційності [7, 8].

Контролі безпеки ISO/IEC 27400 організовані за життєвим циклом IoT–рішення. На етапі проєктування та розробки застосовуються контролі SecDev–1 до SecDev–7, що включають безпечне проєктування архітектури, управління ризиками, захист ланцюга постачання, безпечне кодування та тестування безпеки. Етап впровадження та конфігурації охоплює контролі SecDep–1 до SecDep–5, включаючи безпечне встановлення, конфігурацію, управління ідентифікацією та доступом. Операційна фаза вимагає застосування контролів SecOps–1 до SecOps–10, що забезпечують моніторинг безпеки, управління вразливостями, реагування на інциденти [9].

На рисунку 1 представлено порівняльну структуру трьох фреймворків безпеки IoT.

Проєкт OWASP розробив список "OWASP IoT Top 10 2018", що визначає десять найкритичніших вразливостей IoT–пристроїв. На відміну від NIST та ISO/IEC, які надають комплексні фреймворки безпеки, OWASP зосереджується на конкретних вразливостях та практичних рекомендаціях щодо їх усунення [10]. Десять категорій включають: I1 – слабкі паролі, I2 – незахищені мережеві сервіси, I3 – незахищені інтерфейси, I4 – відсутність безпечного оновлення, I5 – застарілі компоненти, I6 – недостатній захист конфіденційності, I7 – незахищене зберігання даних, I8 – відсутність управління пристроями, I9 – небезпечні налаштування за замовчуванням, I10 – відсутність фізичного захисту [11, 12].

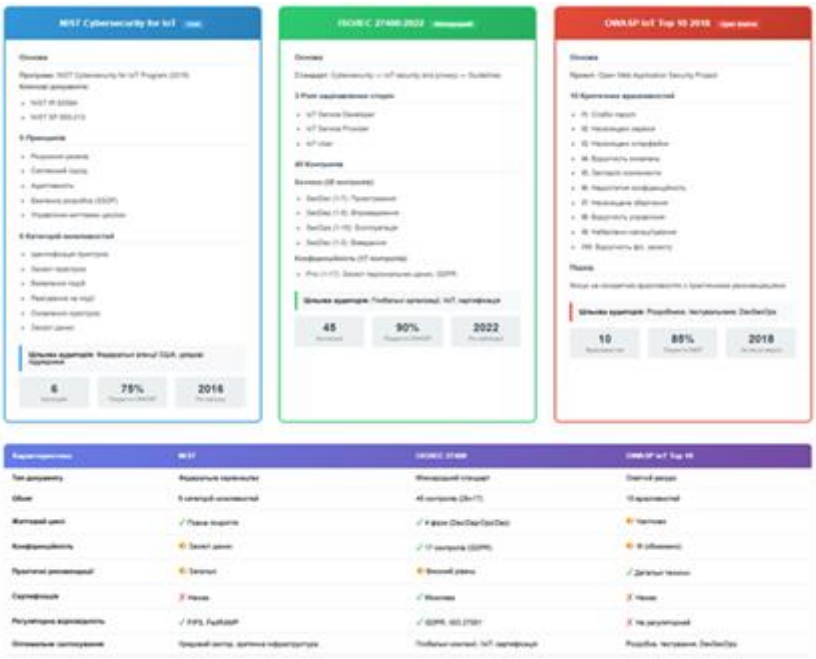


Рисунок 1 – Порівняльна структура фреймворків NIST, ISO/IEC 27400 та OWASP IoT Top 10

Дослідження показують, що OWASP IoT Top 10 залишається актуальним для класифікації реальних вразливостей. Аналіз атак 2017–2018 років виявив, що вразливості типу I1 (слабкі паролі) та I10 (недостатній фізичний захист) були найпоширенішими векторами компрометації IoT–пристроїв у розумних будинках та промислових середовищах. У 2021 році зафіксовано понад 1,5 трильйона атак на IoT–пристрої, що на 100% більше, ніж у 2020 році, підтверджуючи зростаючу актуальність проблеми [13].

2. Порівняльний аналіз покриття загроз та рекомендації щодо застосування

Порівняльний аналіз трьох фреймворків виявляє як спільні елементи, так і унікальні особливості кожного підходу. NIST надає найбільш деталізоване керівництво для федеральних агенцій США з чіткими вимогами до можливостей пристроїв та процесом верифікації відповідності. Фреймворк NIST особливо ефективний для організацій, що працюють з урядовими контрактами або повинні відповідати федеральним вимогам безпеки. Аналіз показує, що NIST забезпечує 75% покриття вразливостей OWASP Top 10, зосереджуючись на базових можливостях кібербезпеки пристроїв [14].

ISO/IEC 27400 пропонує найбільш всеосяжний набір контролів (45 контролів проти 6 категорій у NIST), що охоплюють весь життєвий цикл IoT–рішень від проєктування до утилізації. Унікальною особливістю ISO/IEC 27400 є інтеграція контролів конфіденційності (17 контролів) на рівні з контролями безпеки, що відповідає вимогам GDPR та інших регуляторних актів про захист персональних даних. Стандарт забезпечує 90% покриття вразливостей OWASP Top 10 та особливо підходить для організацій, що прагнуть отримати міжнародну сертифікацію або працюють на глобальних ринках [15].

OWASP IoT Top 10 надає найбільш практичний та доступний підхід,

зосереджуючись на десяти найкритичніших вразливостях з конкретними рекомендаціями щодо їх усунення. На відміну від NIST та ISO/IEC, які є нормативними документами, OWASP є освітнім ресурсом, орієнтованим на розробників, тестувальників безпеки та технічних спеціалістів. OWASP покриває 85% вимог NIST та 80% контролів ISO/IEC, що вказує на його комплементарний характер [16].

Аналіз застосовності фреймворків для різних класів IoT–систем показує диференційовану ефективність. Для споживчих IoT–пристроїв (розумний дім, носимі пристрої) найбільш практичним є підхід OWASP IoT Top 10 завдяки його фокусу на критичних вразливостях та простоті імплементації. Для промислового IoT (IIoT) рекомендується застосування ISO/IEC 27400 через всеосяжність контролів та інтеграцію з системами управління інформаційною безпекою. Для критичної інфраструктури та урядових систем оптимальним є фреймворк NIST через строгі вимоги верифікації та відповідності федеральним стандартам.

Gap–аналіз виявив, що жоден з фреймворків не забезпечує повного покриття всіх аспектів безпеки IoT. NIST має обмежене покриття питань конфіденційності (лише в контексті захисту даних) та не надає детальних рекомендацій щодо безпеки ланцюга постачання. ISO/IEC 27400, незважаючи на всеосяжність, має недостатньо конкретних технічних рекомендацій для розробників та тестувальників. OWASP IoT Top 10, будучи орієнтованим на вразливості, не покриває організаційні та процесні аспекти безпеки.

На рисунку 2 представлено матрицю покриття загроз різними фреймворками.

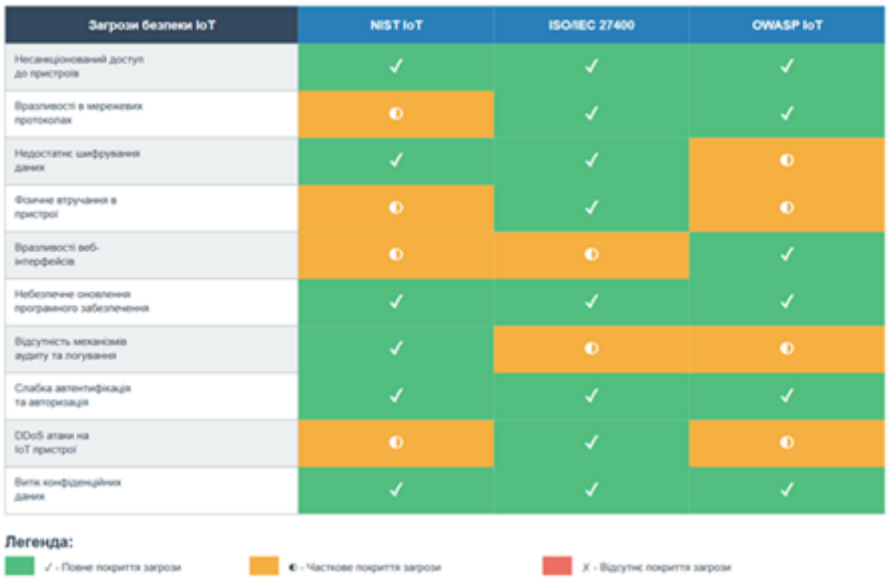


Рисунок 2 – Матриця покриття загроз безпеки IoT фреймворками NIST, ISO/IEC 27400 та OWASP

Для створення комплексної системи безпеки IoT рекомендується використовувати гібридний підхід, що поєднує сильні сторони кожного фреймворку. На етапі проєктування слід застосовувати контролі ISO/IEC 27400 (SecDev–1 до SecDev–7) для забезпечення вбудованої безпеки (security by design) та дотримання регуляторних вимог конфіденційності. Етап розробки та

тестування має керуватися OWASP IoT Top 10 для виявлення та усунення критичних вразливостей через penetration testing та security code review. Розгортання та експлуатація повинні відповідати вимогам NIST SP 800–213 для забезпечення базових можливостей кібербезпеки пристроїв та процесу безперервного моніторингу [17].

Інтеграційна модель передбачає картування контролів різних фреймворків на рівні IoT–архітектури. На рівні пристрою (Device Layer) критичними є вимоги NIST щодо ідентифікації та захисту пристрою, контролі OWASP I1, I4, I10 та контролі ISO/IEC SecDep–2, SecDep–3. На мережевому рівні (Network Layer) застосовуються вимоги NIST щодо захисту даних, контролі OWASP I2, I7 та контролі ISO/IEC SecOps–1, SecOps–3. На прикладному рівні (Application Layer) інтегруються вимоги до захисту інтерфейсів (OWASP I3), управління доступом (NIST, ISO/IEC SecOps–5) та конфіденційності (ISO/IEC Priv–1 до Priv–12).

Висновок. Порівняльний аналіз трьох провідних фреймворків безпеки IoT демонструє їх комплементарний характер та необхідність інтегрованого підходу до захисту IoT–інфраструктури. NIST забезпечує структурований підхід з чіткими вимогами до базових можливостей кібербезпеки (75% покриття OWASP Top 10), оптимальний для федерального сектору та урядових підрядників. ISO/IEC 27400 пропонує найбільш всеосяжний набір з 45 контролів (90% покриття OWASP Top 10), інтегруючи вимоги безпеки та конфіденційності для глобальних ринків. OWASP IoT Top 10 надає практичний фокус на критичні вразливості (85% покриття NIST, 80% ISO/IEC), ефективний для розробників та тестувальників.

Gap–аналіз виявив, що жоден фреймворк не забезпечує повного покриття всіх аспектів безпеки IoT. Рекомендований гібридний підхід інтегрує ISO/IEC 27400 на етапі проєктування для security by design, OWASP IoT Top 10 на етапі розробки/тестування для усунення критичних вразливостей, та NIST SP 800–213 для розгортання/експлуатації з безперервним моніторингом.

Диференційована застосовність фреймворків визначається класом IoT–системи: OWASP для споживчих пристроїв, ISO/IEC 27400 для IIoT, NIST для критичної інфраструктури. Картування контролів на рівні IoT–архітектури дозволяє створити ешелоновану систему захисту від пристрою до хмари.

Перспективи подальших досліджень включають розробку автоматизованих інструментів для верифікації відповідності множині фреймворків, створення галузево–специфічних профілів безпеки та інтеграцію з новітніми технологіями Zero Trust Architecture та квантово–стійкої криптографії.

Перелік використаних джерел.

1. Barrett M., Marron J., Pillitteri V. Y., Boyens J., Quinn S., Witte G., Feldman L. NIST Cybersecurity for IoT Program. National Institute of Standards and Technology. 2020. [Електронний ресурс]. – Режим доступу: https://www.nist.gov/itl/applied–cybersecurity/nist–cybersecurity–iot–program

2. Fagan M., Megas K. N., Scarfone K., Smith M. IoT Device Cybersecurity Capability Core Baseline. NIST Interagency Report 8259A. 2020. DOI: 10.6028/NIST.IR.8259A

3. Fagan M., Megas K.N., Scarfone K., Smith M. IoT Device Cybersecurity

Guidance for the Federal Government: Establishing IoT Device Cybersecurity Requirements. NIST–SP 800–213. 2021. DOI:10.6028/NIST.SP.800–213

4. Thompson D., Ellis R. Implementing NIST IoT Guidelines For Modern Network Security. IS Partners LLC. 2024. [Електронний ресурс]. – Режим доступу: https://www.ispartnersllc.com/blog/nist–iot–guidelines/

5. Souppaya M., Scarfone K. SSDF and IoT Cybersecurity Guidance: Building Blocks for IoT Product Security. NIST Cybersecurity Insights. 2023. [Електронний ресурс]. – Режим доступу: https://www.nist.gov/blogs/cybersecurity–insights/ssdf–and–iot–cybersecurity–guidance

6. ISO/IEC 27400:2022. Cybersecurity – IoT security and privacy – Guidelines. International Organization for Standardization. 2022. [Електронний ресурс]. – Режим доступу: https://www.iso.org/standard/44373.html

7. Rahman A., Singh K. ISO/IEC 27400:2022 – Cybersecurity: IoT Security and Privacy Guidelines. Pacific Certifications. 2023. [Електронний ресурс]. – Режим доступу: https://pacificcert.com/iso–iec–27400–certification/

8. Kumar M. ISO/IEC 27400 IoT Security and Privacy: A Comprehensive Overview. All About Testing. 2023. [Електронний ресурс]. – Режим доступу: https://allabouttesting.org/iso–iec–27400–iot–security–privacy/

9. Weber J., Martinez L. ISO/IEC 27400 IoT Security and Privacy Training Course. PECB. 2024. [Електронний ресурс]. – Режим доступу: https://pecb.com/en/education–and–certification–for–individuals/iso–iec–27400

10. OWASP Internet of Things Project. Open Web Application Security Project. 2024. [Електронний ресурс]. – Режим доступу: https://owasp.org/www–project–internet–of–things/

11. Davis R., Thompson S. The OWASP IoT Top 10 List of Vulnerabilities. InfoSec Insights. 2020. [Електронний ресурс]. – Режим доступу: https://sectigostore.com/blog/owasp–iot–top–10–iot–vulnerabilities/

12. Anderson P., Miller K. OWASP IoT Top 10 2018: Security Guidelines. Whitehats Security. 2023. [Електронний ресурс]. – Режим доступу: https://www.whitehats.nl/en/resources/owasp–iot–top–10

13. Chen Y., Wang L., Zhang H. Real World Implications of OWASP IoT Top 10 2018: An Empirical Study. HackMD Research Platform. 2022. [Електронний ресурс]. – Режим доступу: https://hackmd.io/@oDfzlUPiRg2DrSP35fcd3A/r14HAnJqE

14. Rodriguez M., Garcia A. Comparative Analysis of IoT Security Frameworks: NIST vs ISO/IEC 27400. IEEE Internet of Things Journal. 2024. Vol. 11, No. 3. P. 4521–4538.

15. Singh P., Kumar R., Sharma V. Integration of International Standards for IoT Security: A Systematic Review. Computers & Security. 2024. Vol. 138. Article 103674.

16. Williams J., Brown T., Johnson M. Gap Analysis of IoT Security Frameworks: Identifying Coverage Overlaps and Deficiencies. Journal of Cybersecurity. 2023. Vol. 9, No. 2. Article tyad018.

17. Martinez L., Anderson K., Lee S. A Hybrid Approach to IoT Security: Integrating NIST, ISO/IEC, and OWASP Frameworks. International Journal of Information Security. 2024. Vol. 23, No. 4. P. 2847–2869.

УДК 004

Роман ПИТЕЛЬ, Євген СЕГЕДА

Західноукраїнський національний університет

АЛГОРИТМ ВИЯВЛЕННЯ ШКІДЛИВОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА КІНЦЕВИХ ВУЗЛАХ МЕРЕЖІ

Вступ. Підвищення складності кіберзагроз створює нові виклики для систем виявлення та захисту інформаційних систем. Адаптивні постійні загрози та поліморфні сімейства шкідливого програмного забезпечення (ШПЗ) здатні обходити традиційні методи захисту. Для їх ефективного виявлення необхідно застосовувати комбіновані підходи, що поєднують сигнатурний, поведінковий аналіз та зовнішню репутаційну інформацію. Одним із ефективних рішень є інтеграція системи моніторингу безпеки кінцевих вузлів Wazuh із зовнішнім аналітичним сервісом VirusTotal [1]. Такий підхід дозволяє здійснювати багаторівневу перевірку та швидко реагувати на інциденти безпеки.

Мета дослідження полягає у розробці ефективного алгоритму виявлення шкідливого програмного забезпечення, що інтегрує локальні можливості Wazuh з віддаленим аналітичним сервісом VirusTotal.

1. Аналіз інтеграції інструментів виявлення ШПЗ

Запропонований підхід базується на інтеграції системи моніторингу безпеки Wazuh із зовнішнім аналітичним сервісом VirusTotal для виявлення, валідації та класифікації підозрілих файлів або процесів, що з’являються на кінцевих вузлах мережі [2, 3]. В таблиці 1 наведено основні характеристики, можливості та переваги обох рішень.

Таблиця 1 – Порівняння Wazuh та VirusTotal

Параметр	Wazuh	VirusTotal
Основні функції	Контроль цілісності файлів (FIM), поведінковий аналіз процесів, локальні правила (YARA), генерація алертів, Active Response	Репутаційний аналіз файлів і URL, антивірусні сканування, агреговані дані від сотень AV, додаткові метадані
Рівень дії	Локальний	Глобальний
Тип даних для аналізу	Логи, хеші файлів, поведінкові ознаки процесів	Хеші файлів, URL, IOC (індикатори компрометації)
Виявлення	Використовує локальні сигнатури, поведінкові правила, FIM	Використовує агреговані AV–дані та репутаційні бази
Переваги	Швидкий аналіз, автоматизоване реагування, інтеграція з SIEM, Active Response	Широке охоплення загроз, верифікація підозрілих об’єктів, доповнення локальних індикаторів
Недоліки	Обмежене знання глобальних загроз без інтеграції зовнішніх сервісів	Обмеження публічного API, необхідність передачі хешів для приватності

Порівняльний аналіз показує, що дані інструменти функціонально доповнюють один одного у межах реалізації алгоритмів виявлення загроз. Wazuh

забезпечує локальний рівень захисту, який включає контроль цілісності файлів, поведінковий аналіз процесів і механізми активного реагування на інциденти. VirusTotal виконує роль зовнішнього аналітичного компонента, що надає глобальні бази сигнатур і репутаційні дані для верифікації та класифікації підозрілих об'єктів.

Інтеграція цих двох компонентів дозволяє реалізувати багаторівневий підхід до виявлення ШПЗ. Локальні механізми Wazuh забезпечують безперервний контроль стану системи та поведінки процесів, тоді як VirusTotal надає зовнішню репутаційну інформацію для валідації підозрілих об'єктів.

Взаємодія компонентів (рисунок 1) може бути реалізована за наступною схемою: Кінцевий вузол → Wazuh Agent → Wazuh Manager → VirusTotal API → Wazuh Alerts → Active Response.

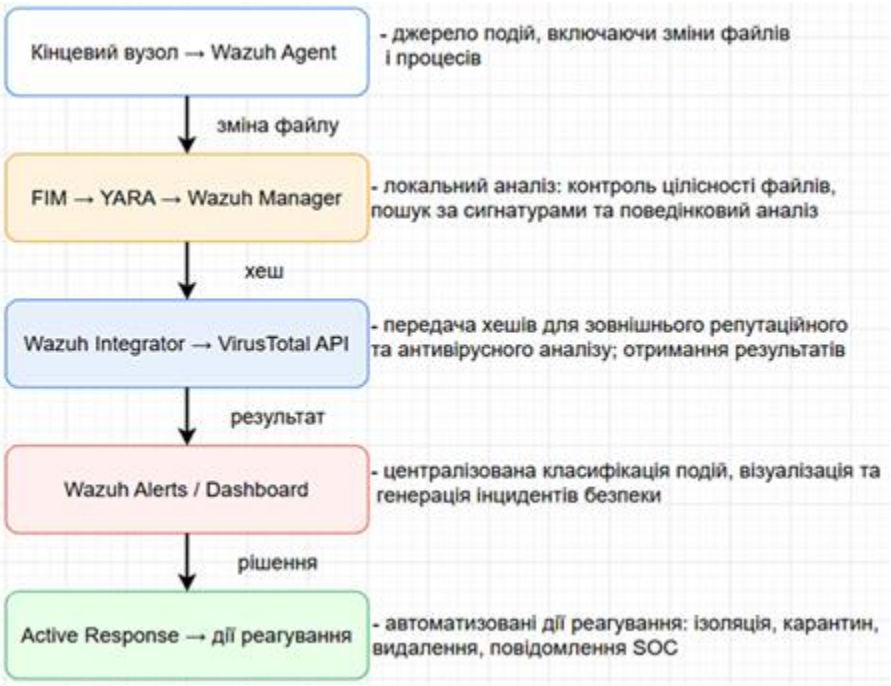


Рисунок 1 – Схема взаємодії компонентів

2. Комбінований алгоритм виявлення ШПЗ

Запропонований алгоритм виявлення ШПЗ включає наступні етапи:

Етап 1. Моніторинг цілісності файлів (File Integrity Monitoring, FIM).

Модуль FIM у Wazuh постійно відстежує створення, зміну, видалення або зміну атрибутів файлів у критичних каталогах системи (наприклад, /usr/bin, C:\Windows\System32, каталоги користувачів). При виявленні змін генерується подія, яка фіксується у журналі Wazuh. До події додаються метадані: шлях, користувач, контрольна сума (SHA256/MD5), час зміни, процес–ініціатор.

Мета даного етапу полягає у виявленні нових або змінених файлів, що можуть бути потенційними зразками ШПЗ.

Етап 2. Локальний аналіз і попередня фільтрація.

Для зменшення кількості помилкових спрацьовувань застосовуються локальні механізми, зокрема:

– YARA–сканування – перевірка файлів за набором правил, що описують сигнатури або поведінкові шаблони шкідливого ПЗ.

Правила Wazuh (local_rules.xml) – поведінковий аналіз подій, наприклад, запуск скриптів PowerShell із мережевими запитами, аномальні процеси з підвищеними привілеями тощо.

Якщо результат аналізу вказує на підозрілу активність, формується подія з рівнем небезпеки (severity level 7–12).

Етап 3. Передача хешів до VirusTotal через Wazuh Integrator.

Після виявлення підозрілого файлу Wazuh може автоматично надіслати його хеш (MD5, SHA1, SHA256) до VirusTotal API через компонент Wazuh Integrator. Конфігурація інтегратора (ossec.conf → <integration name="virustotal">) дозволяє:

– використати API–ключ VirusTotal.
– визначити тип подій, за якими виконується запит.
– автоматично отримувати відповідь про репутацію файлу.

В результаті VirusTotal повертає JSON–файл, що містить:

– кількість антивірусних рушіїв, які класифікували файл як шкідливий;
– назву виявленого сімейства (наприклад, Trojan.Win32.Generic);
– дату останнього аналізу;
– унікальний ідентифікатор зразка.

Етап 4. Валідація та класифікація артефакту.

На основі отриманого результату Wazuh формує подію типу "VirusTotal alert" із зазначенням рівня довіри (наприклад, якщо ≥10 рушіїв підтвердили шкідливість – подія класифікується як інцидент безпеки). Результати валідації додаються у журнал подій Wazuh і можуть бути передані до SIEM або SOC–системи (наприклад, Elastic Stack).

Етап 5. Автоматичне реагування (Active Response).

При виявленні підтвердженого шкідливого файлу активується Active Response, який виконує одну або кілька дій:

– ізоляція хоста (відключення інтерфейсу або блокування IP);
– видалення або карантин файлу;
– завершення процесу, що створив файл;
– сповіщення адміністратора через електронну пошту або SIEM.

Ці дії реалізуються через скрипти (active–response/bin/) і викликаються на основі правил із високим рівнем пріоритету. На рисунку 2 показано потік подій на кінцевому вузлі.

Критерії оцінки ефективності запропонованого алгоритму включають наступні показники:

– середній час виявлення (Time To Detect, TTD) – інтервал між появою файлу та реєстрацією події в системі Wazuh;
– середній час реагування (Time To Respond, TTR) – час від моменту підтвердження загрози до виконання дії Active Response;
– кількість помилкових спрацьовувань (False Positives Rate);
– рівень кореляції локальних і зовнішніх результатів (співвідношення підтверджених випадків VirusTotal до загальної кількості виявлень).

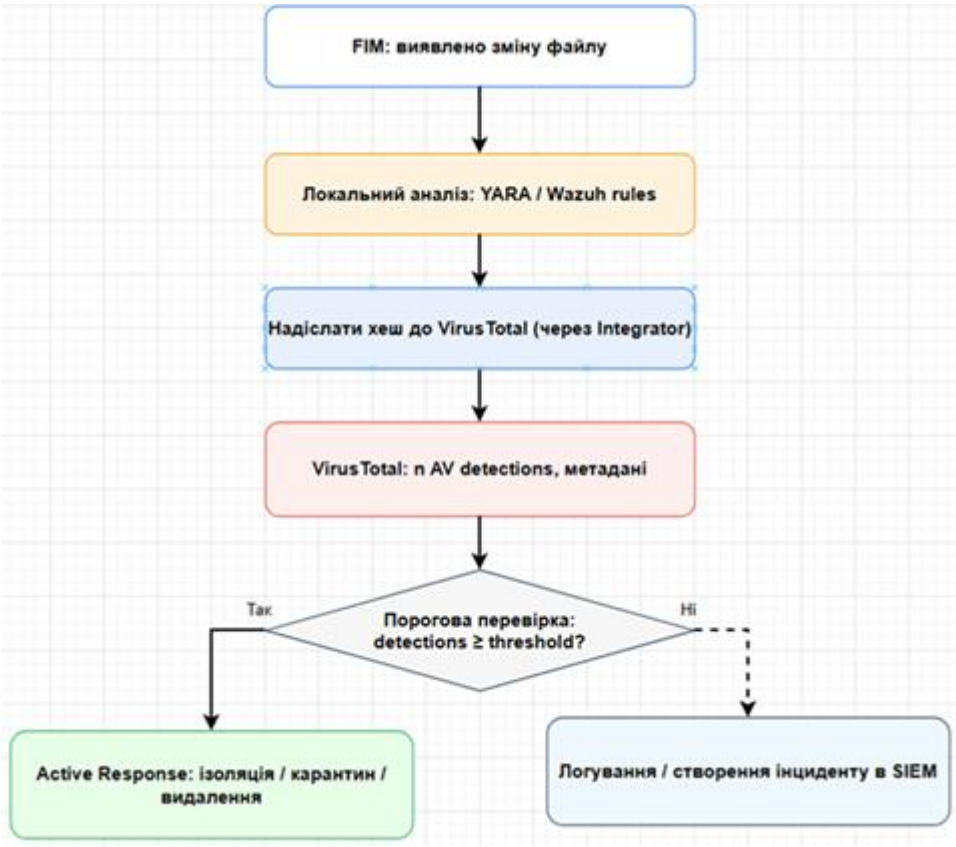


Рисунок 2 – Потік подій на кінцевому вузлі

Висновок. Інтеграція платформи Wazuh із сервісом VirusTotal забезпечує комплексний підхід до виявлення ШПЗ шляхом поєднання сигнатурного, поведінкового та репутаційного аналізів. Такий підхід дозволяє підвищити точність і швидкість детекцій без потреби у великих локальних базах сигнатур, а також реалізувати автоматизоване реагування на загрози та інтеграцію з системами SIEM/SOC для централізованого моніторингу інцидентів.

Запропонований алгоритм дозволяє забезпечити зниження навантаження на традиційні антивірусні рішення завдяки попередній фільтрації та аналітичній кореляції результатів з репутаційними джерелами. Проте, практична реалізація вимагає дотримання політики конфіденційності при передачі хешів або зразків до зовнішніх сервісів, врахування лімітів API–ключів, а також належного налаштування порогових значень кількості спрацьовувань антивірусних рушіїв, а також регулярного оновлення YARA–правил та політик FIM для підтримання актуальності та ефективності системи.

Перелік використаних джерел.

1. Wazuh. External API integration. [Електронний ресурс].– Режим доступу: https://documentation.wazuh.com/current/user–manual/manager/integration–with–external–apis.html

2. Wazuh. File integrity monitoring and YARA. [Електронний ресурс].– Режим доступу: https://documentation.wazuh.com/current/user–manual/capabilities/malware–detection/fim–yara.html

3. YARA in a nutshell. [Електронний ресурс].– Режим доступу: https://virustotal.github.io/yara

УДК 004.9

Підгурський Д.В.

Західноукраїнський національний університет

ІНТЕЛЕКТУАЛЬНІ МЕТОДИ КЛАСИФІКАЦІЇ ДЕФЕКТІВ ВІТРОВИХ ТУРБІН ТА ЗАХИСТУ КАНАЛІВ ПЕРЕДАЧІ ДІАГНОСТИЧНИХ ДАНИХ

Вступ. Сучасні вітрові турбіни є складними кіберфізичними системами, у яких механічні, електричні та цифрові компоненти працюють у тісній взаємодії. Зростання потужності та технічної складності турбін супроводжується збільшенням обсягів діагностичних даних, що надходять від вібраційних, температурних та електричних сенсорів. У таких умовах ефективне виявлення дефектів основних вузлів є критично важливим для забезпечення надійності та економічної ефективності роботи вітроустановок.

Традиційні методи технічної діагностики часто виявляються недостатньо ефективними в умовах мінливих навантажень та високої варіативності сигналів. Тому все більшого значення набувають інтелектуальні методи аналізу даних, які дозволяють автоматично виділяти інформативні ознаки, виявляти приховані закономірності та точно класифікувати технічні дефекти. Алгоритм Ванга–Менделя, як один із найбільш інтерпретованих нечітких підходів, забезпечує побудову прозорих моделей, здатних працювати з неповними або зашумленими даними.

Разом із тим, точність інтелектуальних моделей безпосередньо залежить від достовірності діагностичних даних, що циркулюють у мережах керування та SCADA–системах. Перехоплення, модифікація або підміна цих даних може призвести до помилкової класифікації стану турбіни та порушення роботи алгоритмів моніторингу. Тому захист каналів передавання інформації є необхідною умовою забезпечення надійності кіберфізичної системи вітроенергетичної установки.

Мета: розробити підхід до класифікації дефектів вітрових турбін на основі інтелектуальних методів та визначити ключові вимоги до кіберзахисту каналів передавання діагностичної інформації.

1. Інтелектуальна класифікація дефектів вітрових турбін

Інтелектуальна класифікація дефектів вітрових турбін є ключовим напрямом сучасної діагностики стану обладнання. Вона ґрунтується на аналізі великих обсягів вібраційних, температурних, електричних та SCADA–даних, які характеризують роботу основних вузлів турбіни. Як підкреслюється у роботі [1], ефективна діагностика неможлива без глибокого розуміння природи дефектів таких компонентів, як підшипники, редуктор, генератор та лопаті, оскільки саме ці вузли найчастіше стають причиною аварій та зупинок.

У роботах [2] та [3] детально описано конструкції вітрових установок, що дозволяє сформувати повну картину джерел діагностичної інформації. Розуміння конструкції важливе для інтелектуальних методів тому, що кожен вузол володіє унікальними вібраційними та електричними «сигнатурами», за якими система навчання може розпізнавати тип дефекту.

Для створення інтелектуальних моделей класифікації використовують статистичні, спектральні та часові ознаки, які виділяються з діагностичних сигналів. Це узгоджується з підходами, сформованими у праці [4], де підкреслено високу ефективність нечітких та нейромережевих систем у задачах діагностики лопатей та акустичних аномалій. Нечіткі моделі особливо цінні у випадках, коли дані містять шум, не повністю вимірювані або перебувають у проміжному стані між «справністю» і «дефектом».

Алгоритм Ванга–Менделя займає особливе місце у цій групі методів, оскільки дозволяє автоматично сформувати інтерпретовану базу правил із реальних діагностичних даних. Це дає змогу поєднувати точність машинного навчання з прозорістю експертних систем. Такий підхід добре інтегрується в структури SCADA–моніторингу, що підтверджується результатами, наведеними у роботі [5], де для аналізу SCADA–даних використовуються методи глибинного навчання та автокодувальники.

Таким чином, описані вище принципи конструювання вітрових турбін, характер їхніх несправностей та методи обробки технічних даних узгоджуються з можливостями сучасних інтелектуальних алгоритмів. Перевагами інтелектуальних методів класифікації (табл. 1) є здатність працювати з великими масивами даних, враховувати нелінійні залежності, забезпечувати раннє виявлення дефектів і прогнозувати їхній розвиток. На відміну від традиційних порогових підходів, моделі штучного інтелекту здатні адаптуватися до змін у режимах роботи турбіни, враховувати взаємозв’язки між різними параметрами, а також працювати з сигналами низької якості. Застосування таких методів дозволяє значно зменшити ймовірність аварійних ситуацій, оптимізувати технічне обслуговування та знизити експлуатаційні витрати.

Таблиця 1 – Переваги інтелектуальних методів у порівнянні з традиційними

Традиційні методи	Інтелектуальні методи
Ґрунтуються на жорстких порогах	Навчаються за даними
Погано працюють із шумовими сигналами	Ефективні для зашумлених/неповних даних
Виявляють лише виражені дефекти	Здатні виявляти ранні стадії
Не адаптуються	Адаптивні, самонавчальні
Не описують складні нелінійності	Враховують взаємодію параметрів

Отже, узагальнення огляду літературних джерел та аналіз отриманих даних свідчать, що інтелектуальні методи класифікації є невід’ємною складовою сучасних систем моніторингу стану вітрових турбін. Водночас ефективність таких систем значною мірою залежить від достовірності та цілісності діагностичних даних, що передаються каналами зв’язку в межах SCADA–інфраструктури. Навіть мінімальне спотворення сигналів може призвести до хибної класифікації стану, помилкових рішень або передчасного виведення турбіни з роботи. Тому подальший розгляд питань кіберзахисту каналів передавання діагностичних даних є необхідним кроком для забезпечення комплексної надійності та безпеки вітроенергетичних систем.

2. Захист каналів передавання діагностичних даних.

Захист каналів передавання діагностичних даних є критично важливим компонентом сучасної системи технічного моніторингу вітрових турбін, оскільки процес класифікації дефектів повністю залежить від достовірності вимірювальної інформації. Як зазначається у роботах [2], [3], вітрові електроустановки мають складну архітектуру керування, що включає локальні контролери, промислові мережі, SCADA–системи та комунікаційні шлюзи, якими передаються як оперативні, так і діагностичні дані.

У роботі [5] розглядається структура даних системи SCADA вітрових турбін, а також підкреслюється важливість обробки великих масивів телеметрії у режимі реального часу. Оскільки SCADA–системи історично були розроблені без урахування сучасних кіберзагроз, вони є чутливими до атак типу Man–in–the–Middle, перехоплення даних, ін'єкції хибних сигналів та підміни команд керування. Спотворення даних навіть на рівні однієї температурної або вібраційної ознаки може призвести до невірної класифікації стану вузлів, що, у свою чергу, може спричинити або хибні дії оператора, або передчасне зупинення турбіни. Не менш важливим є те, що, згідно з аналізом дефектів у роботі [4], моделі машинного навчання та нечіткі системи є особливо чутливими до якості вхідних сигналів, оскільки працюють з нелінійними залежностями та ваговими коефіцієнтами, які можуть бути спотворені атакою на канал зв'язку. Це робить системи діагностики подвійно вразливими: як у фізичному, так і у кіберінформаційному вимірі.

З метою захисту каналів передавання інформації застосовуються криптографічні методи, технології контролю доступу та сегментації мереж. Використання протоколів TLS, IPSec, захищених варіантів OPC UA, а також VPN–тунелювання дозволяє забезпечити конфіденційність переданих даних. У роботах, що стосуються моделювання вітрових електроустановок [1, 2], підкреслюється важливість захищеної взаємодії між блоками керування pitch/yaw та системою генератора, оскільки саме ці ланки містять найбільш критичні параметри для діагностики стану турбіни. Отже, інтеграція засобів кіберзахисту з інтелектуальною класифікацією дефектів створює єдиний комплексний підхід, де технічний аналіз доповнюється гарантуванням достовірності даних, а ефективність діагностики напряму залежить від захищеності комунікаційної інфраструктури вітрової електростанції.

Висновок. Таким чином, робимо висновок, що сучасні вітрові турбіни функціонують як складні кіберфізичні системи, у яких технічний стан обладнання та надійність роботи безпосередньо залежать як від точності методів діагностики, так і від безпеки цифрової інфраструктури. Аналіз літературних джерел засвідчує, що інтелектуальні методи обробки даних, зокрема нечіткі моделі та алгоритм Ванга–Менделя, є ефективними засобами для класифікації дефектів основних вузлів вітрових турбін. Вони дозволяють враховувати нелінійні закономірності, працювати з неповними або зашумленими сигналами та виявляти початкові стадії розвитку несправностей, що значно підвищує рівень технічної безпеки та оптимізує процеси обслуговування.

Водночас встановлено, що інтелектуальні класифікаційні системи можуть забезпечити очікуваний рівень точності лише за умови повної достовірності вхідних діагностичних даних. Уразливість каналів передавання інформації, особливо в межах SCADA–систем, створює ризики спотворення сигналів, атак з підміною даних, несанкціонованого доступу та перехоплення телеметрії. Навіть мінімальні зміни у вібраційному або температурному потоці можуть призвести до хибної діагностики, некоректного прогнозу розвитку дефекту або до помилкових керуючих рішень. Отже, побудова ефективної системи моніторингу стану вітрових турбін вимагає комплексного підходу, який поєднує інтелектуальну класифікацію з надійними засобами кіберзахисту. Використання криптографічних протоколів, захищених інтерфейсів обміну даними, сегментації мереж та механізмів автентифікації забезпечує цілісність і конфіденційність інформації, на основі якої працюють моделі штучного інтелекту. Таким чином, інтеграція методів машинного навчання з сучасними принципами кібербезпеки формує основу для створення стійких, адаптивних і високоточних систем діагностики технічного стану вітроенергетичних установок.

Перелік використаних джерел.

1. Горлачук М. А. Розробка рекомендацій для попередження пошкодження вітрових турбін : робота на здобуття кваліфікаційного ступеня магістра : спец. 141 – електроенергетика, електротехніка та електромеханіка / наук. кер. В. П. Коваль. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024. 72 с. [Електронний ресурс]. – Режим доступу: http://elartu.tntu.edu.ua/handle/lib/46863.

2. Салашний В.М. Проектування вітрогенераторної установки для Національного університету «Полтавська політехніка імені Юрія Кондратюка» : робота на здобуття кваліфікаційного ступеня магістра : спец. 144 – теплоенергетика / наук. кер. Б. А. Кутний. Полтава : Національний університет «Полтавська політехніка імені Юрія Кондратюка», 2025. 78 с. [Електронний ресурс]. – Режим доступу: https://reposit.nupp.edu.ua/bitstream/PoltNTU/18598/1/%D0%A1%D0%B0%D0%BB%D0%B0%D1%88%D0%BD%D0%B8%D0%B9%20%D0%92.%D0%9C.pdf.

3. Хоміцький О. І. Розробка системи керування синхронним генератором в автономному режимі для стабілізації напруги і частоти : робота на здобуття кваліфікаційного ступеня магістра : спец. 141 – електроенергетика, електротехніка та електромеханіка / наук. кер. Б. Я. Оробчук. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2024. 66 с. [Електронний ресурс]. – Режим доступу: http://elartu.tntu.edu.ua/handle/lib/46889.

4. Дубчак, Л. О. Аналіз дефектів лопатей вітрових турбін засобами нейро–нечіткої системи. Scientific Bulletin of UNFU, 2025. 35(3), С. 108–113. https://doi.org/10.36930/40350311.

4. Мішенін Д. О. Гібридна модель автокодувальника LSTM для аналізу даних системи SCADA вітрових турбін : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 123 Комп'ютерна інженерія / Д. О. Мішенін ; М–во освіти і науки України, Харків. нац. ун–т радіоелектроніки. – Харків, 2025. – 55 с.

УДК 004.056.5

Дмитро ПІДЛИСЬКИЙ, Аліна ДАВЛЕТОВА

Західноукраїнський національний університет

ПЛАТФОРМА МОНІТОРИНГУ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ НА БАЗІ KIBANA

Вступ. Сучасні інформаційні системи перебувають під постійним впливом кіберзагроз, що вимагає застосування ефективних засобів виявлення, аналізу та прогнозування атак. Одним із ключових напрямів забезпечення кіберзахисту є розвідка загроз (Threat Intelligence, TI) – процес збирання, оброблення та аналітичної інтерпретації індикаторів компрометації (IoC), тактик, технік і процедур зловмисників.

Зважаючи на потребу в автоматизації цих процесів, особливого значення набувають програмні платформи, здатні інтегрувати механізми збору, індексації та візуалізації даних. Одним із найбільш надійних та гнучких рішень є Elastic Stack [1], у якому Kibana відіграє роль аналітичної системи та інструменту для побудови панелей моніторингу кіберзагроз [2].

Метою є розробка та тестування інтегрованої платформи розвідки загроз із використанням Kibana, яка забезпечує автоматизований збір індикаторів, їх структурування та інтерактивну аналітику.

1. Аналіз можливостей Kibana у системах розвідки загроз

Kibana є аналітичною платформою візуалізації, побудованою як складова Elastic Stack. Її архітектура включає ядро та набір модулів (рисунок 1) розширення функціональності, що дозволяють обробляти великі обсяги даних, будувати інтерактивні звіти та реалізовувати моніторинг у режимі реального часу.

Основні функції Kibana, що відповідають для TI–платформ [3–5]:

- аналіз структурованих і неструктурованих даних;
- побудова Dashboards для оцінки динаміки загроз;
- модуль Discover для первинного аналізу індикаторів;
- пошукові запити KQL/ES|QL;
- підтримка інтеграцій зі сторонніми джерелами TI.

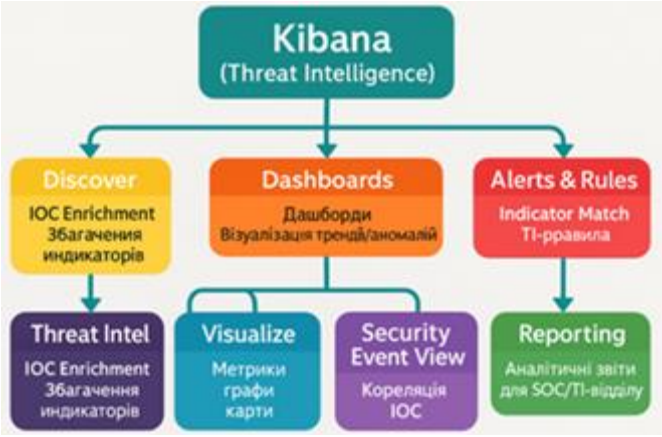


Рисунок 1 – Базові модулі Kibana

Можливості Kibana дозволяють ефективно підтримувати процеси розвідки загроз, включаючи агрегацію IoC, пошук аномалій та формування зрозумілих аналітичних звітів.

2. Архітектура інтегрованої TI–платформи

На основі Elastic Stack розроблено архітектуру (рисунок 2) інтегрованої платформи розвідки загроз (ПРЗ), що включає такі компоненти:

- інфраструктуру для індексації даних;
- сховище індикаторів у вигляді data stream;
- ingest–процеси для попередньої обробки інформації;
- модуль завантаження IoC;
- аналітичний інтерфейс у Kibana.

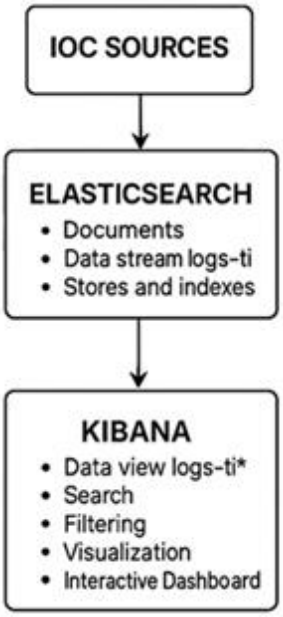


Рисунок 2 – Архітектура інтегрованої платформи розвідки загроз

Запропонована структура забезпечує модульність, масштабованість та гнучкість платформи, необхідну для сучасних TI–систем.

Інтегрована платформа розвідки загроз функціонує за принципом наскрізної обробки даних, що охоплює отримання індикаторів компрометації (IoC), їх індексацію, кореляцію з подіями безпеки та формування аналітичних результатів для подальшого реагування. Загальна логіка роботи системи відображена на рисунку 3.

Потоки індикаторів надходять із відкритих і приватних TI–фідів, що містять різні категорії IoC (IP–адреси, домени, URL, хеші файлів, ознаки кампаній), які використовуються для виявлення шкідливої діяльності. Для завантаження IoC використовується модуль Threat Intel, що входить до складу Filebeat або Elastic Agent. Його функції полягають у періодичному отримання індикаторів з TI–фідів, збагаченні та нормалізації даних, формуванню документів у форматі ECS (Elastic Common Schema) та передавання IoC для індексації у Elasticsearch. Агент виступає проміжною ланкою між зовнішніми джерелами та системою аналітики.

Elasticsearch виконує роль основного компонента зберігання, індексації та пошуку як IoC, так і подій безпеки. Індикатори зберігаються в індексах типу logs–ti* згідно з визначеним шаблоном. Події безпеки (DNS–логи, Netflow, Endpoint–події) зберігаються в індексах типу logs–*. Сховище підтримує швидкий пошук, агрегації та можливість масштабування. Elasticsearch об'єднує технічні індикатори та операційні події в єдиному аналітичному просторі.

Рисунок 3 – Потік даних в інтегрованій ПРЗ

Kibana забезпечує доступ до всіх зібраних даних за допомогою модулів Security / Threat Intelligence Utilities, візуалізацій Lens, інтерактивних Dashboard та засобів пошуку в Discover.

3. Моделювання та тестування роботи платформи

Для реалізації ПРЗ було розгорнуто Elastic Stack у середовищі Docker, активовано модулі безпеки та створено необхідні шаблони даних. Для індикаторів загроз створено спеціальний data stream logs–ti, який автоматично генерує часові сегменти. Створено Python–модуль для імпорту індикаторів компрометації із CSV–файлів. Його завдання полягає у зчитуванні IoC, трансформації у JSON–структуру, надсилання у data stream тареєстрацію результатів індексації. Модуль може бути використаний і у реальних умовах – як компонент автоматизованої системи збору TI–даних.

За допомогою модуля імпорту було завантажено тестовий набір IoC. Elasticsearch автоматично сформував новий сегмент data stream, а Kibana відобразила дані у модулі Discover (рисунок 4). Дані успішно індексувалися разом із часовою міткою, типом загрози, джерелом та описом.

У Kibana сформовано представлення даних TI Indicators, що дозволяє використовувати індикатори у Dashboards, Lens, таблицях та діаграмах. Для демонстрації роботи платформи створено аналітичний Dashboard Threat Intelligence Overview, який включає:

- таблицю IoC;
- розподіл загроз за джерелами;
- динаміку появи індикаторів;
- топ–значення за IP та threat_type.

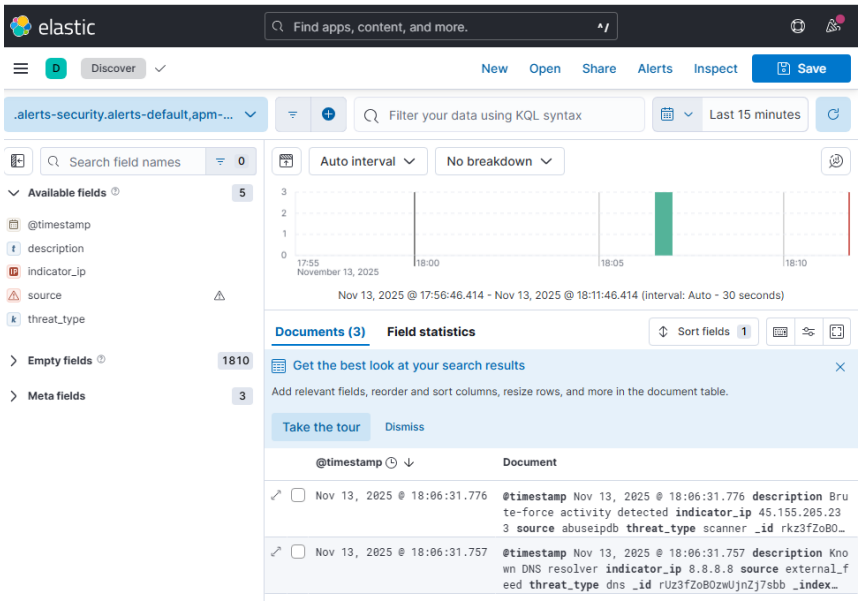


Рисунок 4 – Відображення IoC у модулі Discover

Створені візуалізації (рисунок 5) забезпечують виявлення домінуючих типів загроз, наприклад botnet, scanner, аналіз джерел даних, manual, external feed, abuseipdb, оцінку динаміки появи IoC, визначення найбільш частих IP–адрес, комплексну оцінку загроз на інтегрованій панелі тощо.

Рисунок 5 – Приклад створених графіків у Dashboard

Візуальне представлення у Dashboard дозволяє оцінити активність загроз, джерела компрометації та часові тренди.

Висновок. Створена система підтвердила працездатність та здатність підтримувати ключові процеси розвідки загроз, а також може бути розширена для реальних сценаріїв кіберзахисту.

Перелік використаних джерел.

1. The Elastic Stack. – Офіційна документація. [Ектронний ресурс].– Режим доступу: https://www.elastic.co/guide/en/kibana/current/index.html

2. Kibana plugins. [Ектронний ресурс].– Режим доступу: https://www.elastic.co/docs/reference/kibana/kibana–plugins

3. What is Kibana? [Ектронний ресурс].– Режим доступу: https://logit.io/blog/post/what–is–kibana/

4. Kibana dashboards. [Ектронний ресурс].– Режим доступу: https://www.elastic.co/guide/en/kibana/current/lens.htmlhttps://www.elastic.co/docs/explore–analyze/dashboards

5. Discover. [Ектронний ресурс].– Режим доступу: https://www.elastic.co/docs/explore–analyze/discover

УДК 004.62

Василь ПОМАЗИБІДА, Микола НЕТРЕБЯК

Західноукраїнський національний університет

АНАЛІЗ РОЗВИТКУ ХМАРНИХ ОБЧИСЛЕНЬ ТА ПРОБЛЕМИ ЇХ БЕЗПЕКИ

Вступ. Перехід до хмарних технологій є наступним етапом розвитку проведення обчислень. Хмарні обчислення пропонують низку переваг своїм клієнтам. Оплачуючи використання апаратних та програмних ресурсів (серверів, розміщених у дата–центрах, додатків, програмного забезпечення тощо) користувачі можуть отримати через Інтернет доступ до потужних обчислювальних систем або систем зберігання даних великої ємності і без необхідності їх придбання та сплати додаткових витрат на обслуговування обладнання застосовувати їх у своїй діяльності. Спільне використання таких центрів декількома клієнтами за допомогою концепції віртуалізації не повинно загрожувати витоком даних, тому задля забезпечення захисту інформації застосовуються алгоритми шифрування, зокрема гомоморфне шифрування.

Метою дослідження є дослідження сфери застосування гомоморфного шифрування, зокрема хмарних обчислень та аналіз проблем, що виникають при їх впровадженні.

1. Основні криптографічні алгоритми

Останнім часом криптографія перетворилася на поле битви деяких з найкращих математиків і комп'ютерних вчених світу. Здатність безпечно зберігати та передавати конфіденційну інформацію виявилася критичним фактором успіху у війні та бізнесі. Оскільки уряди не хочуть, щоб певні організації в їхніх країнах та за їх межами мали доступ до засобів отримання та надсилання прихованої інформації, яка може становити загрозу національним інтересам, криптографія піддається різним обмеженням у багатьох країнах, починаючи від обмежень використання та експорту програмного забезпечення до публічного поширення математичних концепцій, які можуть бути використані для розробки криптографічних алгоритмів.

Криптографічний алгоритм – це набір кроків, які можна використовувати для перетворення відкритого тексту в зашифрований текст. Криптографічний алгоритм також відомий як алгоритм шифрування [1].

Криптографічний алгоритм використовує ключ шифрування для приховування інформації та перетворення її в шифротекст, а ключ дешифрування можна використовувати для перетворення її назад у відкритий текст.

Існують різні типи криптографічних алгоритмів, але можна узагальнити 4 основні типи криптографічних алгоритмів:

- стандарт шифрування даних (DES);
- розширений стандарт шифрування (AES);
- алгоритм RSA (алгоритм Рівеста, Шаміра, Адлемана);
- алгоритм безпечного хешування (SHA).

Кожен з яких має свої переваги при реалізації, а гомоморфне шифрування дає змогу поєднати декілька алгоритмів у одну систему, що дозволяє підсилити переваги та нейтралізувати недоліки.

2. Розвиток хмарних обчислень та проблеми безпеки

Хмарні обчислення за останні роки значно зросли в популярності та застосуванні. Переваги хмарних обчислень численні, серед них економія коштів, підвищена масштабованість, покращена доступність та спрощене управління IT. Компанії будь–якого розміру переходять на хмарні технології завдяки їхній здатності надавати обчислювальні ресурси, сховища та програмні послуги на вимогу без необхідності розбудови великої внутрішньої інфраструктури.

Хмарні обчислення вже існували під різними назвами, такими як «аутсорсинг» та «хостинг серверів» [2]. Але низька продуктивність використовуваних процесорів, повільне підключення до Інтернету та надмірна вартість використовуваних матеріалів не дозволяють використовувати послуги та місця для зберігання даних. Однак останні досягнення в сучасних технологіях (завдяки віртуалізації) проклали шлях для цих операцій із більш швидкою обробкою. Проблеми безпеки хмарних обчислень також є актуальними для багатьох дослідників; першочерговим завданням було зосередитися на безпеці, яка є найбільшою проблемою для організацій, що розглядають можливість переходу до хмарних технологій. Використання хмарних обчислень дає багато переваг, включаючи зниження витрат, легке обслуговування та повторне надання ресурсів.

Однак зростаюча залежність від хмарних обчислень також викликала значні проблеми з безпекою. Коли організації передають свої дані та обчислення хмарним постачальникам послуг, вони відмовляються від прямого контролю над своєю конфіденційною інформацією. Це створює новий набір ризиків для безпеки, оскільки дані тепер зберігаються та обробляються на інфраструктурі, яка не належить організації та не контролюється нею повністю [3].

Система хмарних обчислень поділяється на дві частини: фронтенд і бекенд. Фронтенд – це частина, через яку користувач може взаємодіяти з сервером, а бекенд – це сервер, який надає дані клієнту. Між сервером і клієнтом мережа працює як проміжне програмне забезпечення (рис. 1).

Хмарні обчислення є одним з найвизначніших досягнень у світі інформаційних технологій за останнє десятиліття. IT–компанії надають гнучкі хмарні послуги та інфраструктуру практично для будь–яких цілей, що дозволяє клієнтам швидко масштабувати свої послуги за допомогою моделей надання послуг з оплатою за фактичне використання. Питання про потенційні ризики, пов'язані з втратою конфіденційності, цілісності та доступності даних користувачів у хмарних мережах через технічні збої або навіть цілеспрямовані атаки як ззовні, так і зсередини мережі хмарного провайдера [4].

Хмарні провайдери зазвичай використовують загальні моделі захисту безпеки, які вимагають довіри до хмарного провайдера, та класичні прозорі для користувача заходи безпеки для захисту своїх мереж та задоволення цілей безпеки своїх клієнтів.

Ці заходи все ще перебувають на початковій стадії та схильні до загальних загроз безпеки, таких як примус хмарних провайдерів під тиском місцевої (політичної) влади розкривати дані користувачів хмари [6].

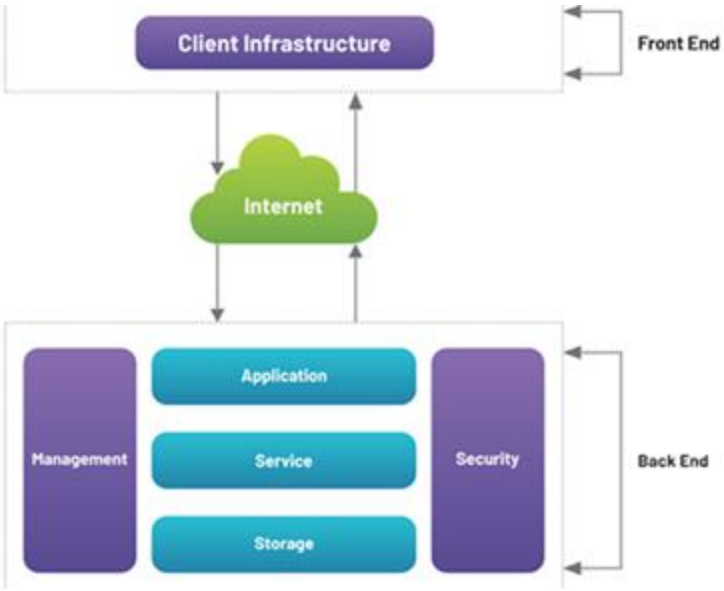


Рисунок 1 – Система хмарної архітектури

Обчислення зашифрованих даних базуються на двох незалежних алгоритмах шифрування, що забезпечують гомоморфність. Модель складається з агента, який виступає делегатором, та принаймні одного працівника в хмарі, який виконує обчислення, однак необхідність повторного шифрування даних для переходу між операціями додавання та множення обмежувала застосовність.

Висновок. Проведене дослідження показало, що визначення хмарних обчислень, про які згадувалося раніше, не згадують жодних понять безпеки даних, що зберігаються в хмарних обчисленнях, навіть незважаючи на те, що це нещодавнє визначення. Тому ми розуміємо, що хмарним обчисленням бракує безпеки, конфіденційності та прозорості. Надання інфраструктури, платформи або програмного забезпечення як послуги є недостатнім, якщо хмарний провайдер не гарантує кращої безпеки та конфіденційності даних клієнтів. В такій ситуації застосування гомомрфного шифрування на боці користувача дасть можливість безпечної обробки даних та дозволить забезпечити їх конфіденційність.

Перелік використаних джерел.

1. Gaborit, Philippe, Olivier Ruatta, and Julien Schrek. "On the complexity of the rank syndrome decoding problem." IEEE Transactions on Information Theory 62.2 (2020): 1006–1019.

2. Sarker, Iqbal H., et al. "Cybersecurity data science: an overview from machinelearning perspective." Journal of Big data 7 (2020): 1–29.

3. Chillotti I., Gama N., Georgieva M., Izabachène M.: TFHE: fast fully homomorphic encryption over the torus. J. Cryptol. 33(1), 34–91 (2021)

4. Marcolla, Chiara, et al. "Survey on fully homomorphic encryption, theory, and applications." Proceedings of the IEEE 110.10 (2022): 1572–1609.

5. Shaukat, Kamran, et al. “Performance Comparison and Current Challenges of Using Machine Learning Techniques in Cybersecurity.” Energies, vol. 13, no. 10, May 2020, p. 2509. https://doi.org/10.3390/en13102509.

УДК 004.056.5

Владислав РУЩАК

Західноукраїнський національний університет

ПОРІВНЯННЯ FLOW ТА TYPESCRIPT В JAVASCRIPT

Вступ. Екосистема JavaScript за останні роки перетворилася з простої мови для браузерів на повноцінне середовище для розробки складних веб– і серверних застосунків. Одним із ключових викликів при цьому стала відсутність статичної типізації – механізму, що дозволяє виявляти помилки ще до запуску коду. Відповіддю на цю потребу стали два підходи до типізації JavaScript: Flow, запропонований Facebook (Meta), і TypeScript від Microsoft.

Обидва інструменти прагнули розв'язати одну й ту саму проблему – зробити JavaScript безпечнішим та передбачуванішим, зберігаючи його гнучкість. Обидва реалізують статичний аналіз типів, інтегруються в процес розробки та транспілюють код у стандартний JavaScript, сумісний із будь–яким браузером, навіть такими застарілими, як Internet Explorer 8. Попри схожість у підходах, ці системи розвивалися у різних напрямах і сьогодні займають нерівні позиції на ринку. У цьому контексті порівняння Flow та TypeScript не лише виявляє технічні відмінності, але й дозволяє зрозуміти, чому одна з технологій набула широкої підтримки, а інша поступово втратила актуальність.

Мета: Проаналізувати та порівняти можливості застосування Typescript та flow.

1. Безпека типів в JavaScript

Як і в більшості скриптових (інтерпретованих) мов програмування, у JavaScript можна написати код, що не працює. Якщо виконання цього коду не ініціюється браузером, помилка не виникає – відсутні будь–які повідомлення або попередження. З одного боку, це може бути перевагою: навіть якщо на великому вебсайті в обробнику натискання кнопки присутня синтаксична помилка, це не завадить повному завантаженню сторінки для користувача.

Водночас така поведінка є потенційно небезпечною. Наявність коду, що не працює, негативно впливає на якість проєкту. Щоб уникнути таких ситуацій, ще до розгортання сайту бажано перевірити всі скрипти на відсутність помилок – принаймні синтаксичних. У ідеалі слід також впевнитися, що код працює коректно. Для цього використовуються різноманітні інструменти – зокрема, такі як npm, webpack, babel або tsc, а також засоби для тестування (karma, jsdom, mocha, chai тощо).

У теоретично ідеальному середовищі всі скрипти, включно з однорядковими, покриті автоматизованими тестами. У реальності це рідкість, і значна частина коду залишається без тестового покриття. Для такої частини необхідно застосовувати автоматизовані засоби перевірки, які дозволяють:

Перевірити коректність синтаксису JavaScript. Інструменти аналізу переконуються, що текст програми може бути розпізнаний інтерпретатором: що всі дужки закриті, рядки – правильно обмежені лапками тощо. Таку перевірку зазвичай виконують збирачі, транспайлери, мініфікатори та обфускатори.

Перевірити правильність використання семантики мови. Наприклад, у коді:

```
var x = null;
x.foo();
```

синтаксис формально правильний, але семантично код некоректний – спроба викликати метод у null призведе до помилки під час виконання.

Окрім семантичних, існують і логічні помилки – найнебезпечніший тип, адже програма виконується без збоїв, але результат не відповідає очікуванням. Класичний приклад:

```
console.log(input.value);    // "1"
console.log(input.value + 1); // "11"
```

Інструменти статичного аналізу коду, як–от ESLint, здатні виявити значну кількість потенційних помилок, наприклад:

– некоректні умови завершення циклу for;

– заборона використання async–функцій як аргументів конструктора Promise;

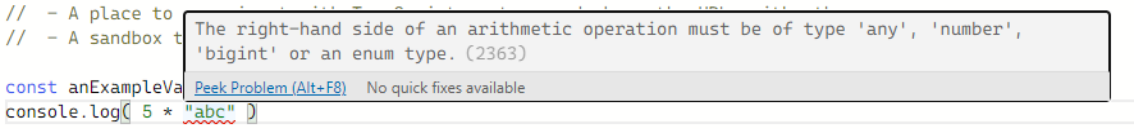
– присвоєння в умовах;

– дублікати ключів у літералах об’єктів;

– та інші подібні ситуації.

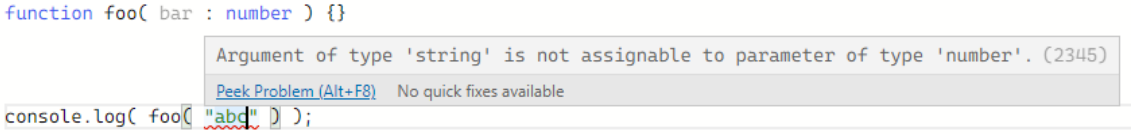
Загалом, усі подібні правила є свого роду обмеженнями, які лінтер накладає на розробника. Вони зменшують гнучкість JavaScript, аби знизити ризик помилок. Наприклад, за активованих правил не допускається присвоєння в умовах, хоча стандарт мови це дозволяє. У деяких конфігураціях забороняється навіть використання console.log().

Додавання типів змінних і перевірка викликів з урахуванням типів – це ще один рівень обмежень, покликаний зменшити кількість можливих помилок у коді JavaScript.



Спроба помножити число на рядок

Спроба звернутися до неіснуючої (неописаної у типі) властивості об'єкта



Спроба викликати функцію з незбігаючим типом аргументу. Якщо ми напишемо цей код без засобів перевірки типів, код успішно траспилюється. Ніякі засоби статичного аналізу коду, якщо вони не використовують (явно чи неявно) інформацію про типи об'єктів, не зможуть знайти ці помилки.

Тобто додавання типізації JavaScript додає додаткові обмеження на код, який пише програміст, зате дозволяє знайти такі помилки, які в іншому випадку

трапилися б під час виконання скрипта (тобто швидше за все у користувача в браузері). Обидва популярні рушії типізації для JavaScript – TypeScript (Microsoft) і Flow (Meta / Facebook) – надають приблизно однакові можливості. Водночас існує важлива концептуальна відмінність від мов із суворою класичною типізацією, таких як Java чи C++. У JavaScript всі типи фактично є інтерфейсами – тобто описами набору властивостей (разом із їх типами й аргументами функцій). Якщо два інтерфейси сумісні за складом властивостей, вони вважаються взаємозамінними. В таблиці 2 приведено можливості типізації JavaScript. Наприклад, наведений нижче код є абсолютно коректним у TypeScript, хоча був би некоректним у Java чи C++:

```
type MyTypeA = { foo: string; bar: number; };
type MyTypeB = { foo: string; };
function myFunction(arg: MyTypeB): string {
  return `Hello, ${arg.foo}!`;}
const myVar: MyTypeA = { foo: "World", bar: 42 };
console.log(myFunction(myVar)); // "Hello, World!"
```

Цей приклад демонструє структурну типізацію: достатньо, щоб тип аргументу myVar мав усі обов'язкові поля типу MyTypeB – наявність додаткових полів не є проблемою. Код також можна скоротити, використавши літеральний об’єкт без явного зазначення типу:

```
type MyTypeB = { foo: string };
function myFunction(arg: MyTypeB): string {
  return `Hello, ${arg.foo}!`;}
const myVar = { foo: "World", bar: 42 };
console.log(myFunction(myVar)); // "Hello, World!"
```

Таблиця 1 – Можливості типізації JavaScript

	Flow	TypeScript

Можливість задати тип змінної, аргументу або тип значення функції, що повертається	`a : number = 5;`
	`function foo( bar : string) : void {`
	`  /*...*/`
	`}`	

Можливість описати свій тип об'єкта (інтерфейс)	`type MyType {`
	`  foo: string,`
	`  bar: number`
	`}`	

Обмеження допустимих значень для типу	`type Suit = "Diamonds" | "Clubs" | "Hearts" | "Spades";`	

Окремий type–level extension для перерахувань		`enum Direction { Up, Down, Left, Right }`
«Складання» типів	`type MyType = TypeA & TypeB;`	
Додаткові типи для складних випадків	`$Keys<T>, $Values<T>, $ReadOnly<T>, $Exact<T>, $Diff<A, B>, $Rest<A, B>, $PropertyType<T, k>, $ElementType<T, K>, $NonMaybeType<T>, $ObjMap<T, F>, $ObjMapi<T, F>, $TupleMap<T, F>, $Call<F, T...>, Class<T>, $Shape<T>, $Exports<T>, $Supertype<T>, $Subtype<T>, Existential Type (*)`	`Partial<T>, Required<T>, Readonly<T>, Record<K,T>, Pick<T, K>, Omit<T, K>, Exclude<T, U>, Extract<T, U>, NonNullable<T>, Parameters<T>, ConstructorParameters<T>, ReturnType<T>, InstanceType<T>, ThisParameterType<T>, OmitThisParameter<T>, ThisType<T>`

У цьому випадку інтерфейс об’єкта myVar – { foo: string, bar: number } – є надмножиною MyTypeB, тому сумісність зберігається. Такий підхід спрощує взаємодію з бібліотеками, зовнішнім кодом та різними API. Часто це дає змогу передавати об’єкти з потрібною структурою без необхідності явно імпортувати відповідні типи:

```
// У бібліотеці interface OptionsType {
        optionA?: string;
        optionB?: number;
}export function libFunction(arg: number, options = {} as OptionsType) {
         /* ... */}// У коді користувача
        import { libFunction } from "lib";
        libFunction(42, { optionA: "someValue" });
```

У цьому прикладі інтерфейс OptionsType не імпортовано в код користувача, але система типізації автоматично перевіряє сумісність об’єкта з очікуваною структурою. У класичних мовах програмування, таких як Java, така поведінка призводить до помилок компіляції. TypeScript та Flow не підтримуються браузерами напряму. Як і багато нових можливостей JavaScript, типи потребують попереднього перетворення – транспіляції. Це процес, під час якого код з нестандартними можливостями перетворюється на звичайний JavaScript, який розуміють браузери. Що стосується типів – усі описи інтерфейсів, анотації типів та інші елементи типізації просто видаляються.

Наприклад, наведений TypeScript–код:

```
type MyTypeA = { foo: string; bar: number };
type MyTypeB = { foo: string };
function myFunction(arg: MyTypeB): string {
  return `Hello, ${arg.foo}!`;}
const myVar: MyTypeA = { foo: "World", bar: 42 };
console.log(myFunction(myVar));
```

після транспіляції перетворюється на:

```
function myFunction(arg) {
  return `Hello, ${arg.foo}!`;} const myVar = { foo: "World", bar: 42 };
console.log(myFunction(myVar)); // "Hello, World!"
```

Тобто вся інформація про типи видаляється ще до виконання коду браузером. Саме ця особливість дозволяє використовувати нові синтаксичні можливості мови без втрати сумісності. Для перетворення коду, що містить типи, використовуються різні інструменти:

Flow – з Babel використовується плагін @babel/plugin–transform–flow–strip–types, який видаляє типи Flow. TypeScript: можна використовувати Babel з плагіном @babel/plugin–transform–typescript, який також видаляє анотації типів; або застосовувати офіційний транспілятор tsc від Microsoft – окрему утиліту, яка може виконувати як видалення типів, так і компіляцію проекту. Таким чином, використання типів у JavaScript (через TypeScript або Flow) – це не про виконання, а про розробку: вони дозволяють виявляти помилки ще до запуску коду, значно підвищуючи якість програмного забезпечення без шкоди для сумісності з браузерами.

Висновок. TypeScript і Flow мають схожі можливості для додавання статичної типізації в JavaScript, проте TypeScript отримав значно ширшу підтримку спільноти та індустрії. Його екосистема активно розвивається, інтегрується з популярними редакторами коду та підтримується більшістю сучасних фреймворків. Flow, хоча й був технічно потужним, поступово втратив актуальність через меншу гнучкість у конфігурації та слабшу підтримку сторонніх бібліотек. У реальних проектах TypeScript демонструє кращу сумісність і простоту використання, що робить його переважним вибором для більшості команд.

Перелік використаних джерел.

1. S. Kushwah, C. Bansal, S. Rathore, V. Kate, D. Bargal and D. Vishwakarma, "TypeScript: An Open–Source Programming Language with Options for Robust Development and Large–Scale Applications," 2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET), Indore, India, 2024, pp. 1–5, doi: 10.1109/ACROSET62108.2024.10743426.

2. Carl Rippon, ASP.NET Core 5 and React: Full–stack web development using .NET 5, React 17, and TypeScript 4 , Packt Publishing, 2021.

УДК 004.056.53

Сарапук О.І.[1], Черняк В.А.[2]

[1]Західноукраїнський національний університет

[2] Відокремлений структурний підрозділ «Рівненський фаховий коледж Національного університету біоресурсів і природокористування України»

СТРУКТУРА МЕРЕЖІ КВАНТОВОГО РОЗПОДІЛУ КЛЮЧІВ ЗА ВЕРСІЄЮ ETSI

Вступ. Розробка уніфікованої структури мережі квантового розподілу ключів (КРК) за специфікаціями ETSI [1] є надзвичайно актуальною через швидке наближення практичної загрози з боку квантових комп'ютерів — вже зараз зростає ризик «збирання сьогоднішніх зашифрованих даних для розшифровки пізніше». Стандартизована архітектура дозволяє трансформувати поодинокі пілотні лінії досліджень у масштабовані, сумісні між виробниками мережі та додатками рішення.

ETSI як галузева ініціатива вже оприлюднила набір документів, що визначають ролі вузлів, інтерфейси, протоколи доставки ключів та захисні профілі (наприклад, GS QKD 015 – архітектура, GS QKD 014 – REST-API доставки ключів, GS QKD 016 – захисний профіль). Використання цих специфікацій забезпечує інтероперабельність обладнання від різних постачальників, стандартизовані інтерфейси для оркестрації та управління ключовими ресурсами, а також підґрунтя для сертифікації та оцінки безпеки.

Мета: проаналізувати структуру мережі КРК за версією ETSI.

1. Структура мережі квантового розподілу ключів

Група стандартизації ETSI у сфері квантового розподілу ключів охоплює різні сфери застосування КРК: безпеку протоколів і реалізацій КРК, інтерфейси взаємодії, методи вимірювань тощо. Перший варіант мережі КРК був представлений у описі інтерфейсу взаємодії квантової апаратури та СЗІ.

Мережа КРК побудована з довірених вузлів (рисунок 1).

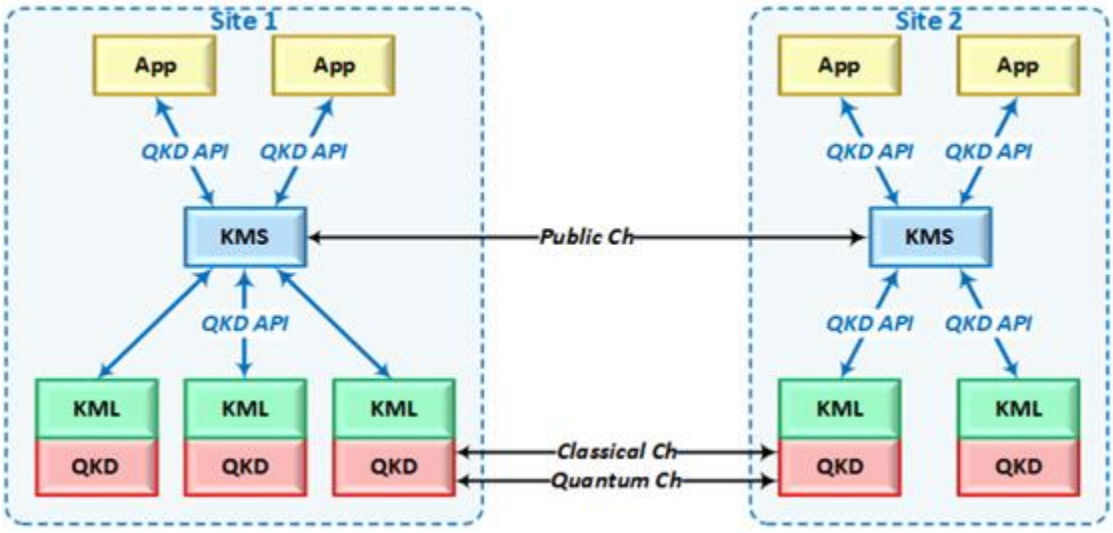


Рисунок 1 – Структура мережі КРК за документом ETSI GS QKD 004

У сценарії взаємодії відповідно до ETSI GS QKD 004 беруть участь; апаратне забезпечення КРК – QKD; сервер управління квантовими ключами – KML; сервер управління ключами – KMS; клієнти мережі, користувацькі застосунки – App.

Система побудована з незалежних частин. Довірений вузол містить кілька екземплярів квантової апаратури, сервери управління квантовими ключами, сервер управління ключами. При цьому до довіреного вузла включається також користувацький застосунок, для якого генеруються секретні ключі. Далі проводиться аналіз кожного з об'єктів довіреного вузла та його функцій.

Базовим елементом мережі КРК є квантова апаратура. При цьому сполучені пари комплектів квантової апаратури, з'єднані як квантовим, так і класичним автентифікованим каналом, є самодостатнім елементом. Генерація квантових ключів відбувається незалежно від інших процесів у мережі КРК.

Квантова апаратура генерує не ключ, а випадкову послідовність нефіксованої довжини. Для формування квантових ключів із сукупності випадкових послідовностей застосовуються сервери управління квантовими ключами (KML). На кожен блок квантової апаратури в кожному вузлі комутації мережі (ВКМ) припадає по одному серверу управління квантовими ключами. Ця частина ВКМ формує квантові ключі з послідовностей, отриманих від квантової апаратури, зокрема присвоюючи ідентифікатори квантових ключів та іншу метаінформацію. Сервер управління містить сховище квантових ключів, у яке поміщаються сформовані квантові ключі з усією необхідною метаінформацією. При цьому їх ідентичність ґрунтується лише на довірі до протоколу КРК, що теоретично гарантує ідентичність сформованих послідовностей.

Для передавання спільних секретів у СЗІ-споживачі, закріплені за ВКМ, не з'єднаними безпосередньо квантовим каналом, використовується сервер управління ключами (KMS). Цей елемент ВКМ у ранніх документах ETSI описаний поверхнево, без зазначення способу розподілу спільного секрету між несусідніми вузлами. Як захист передавання квантового ключа використовується кодування одноразовим блокнотом.

Таким чином, перший варіант мережі КРК за версією ETSI являє собою чотирирівневу структуру. Два рівні мережі оперують квантовими ключами, третій рівень призначений для передавання квантових ключів, згенерованих на одному сегменті мережі КРК, на інші сегменти. Четвертий рівень мережі – рівень користувацьких застосунків. Другий рівень мережі досить чисельний за кількістю елементів, кожен з яких виконує лише невелику частину функцій, реалізованих у ВКМ, і може бути оптимізований. Третій рівень, навпаки, є найбільш навантаженим елементом ВКМ.

Пізніше був випущений стандарт ETSI GS QKD 014 V1.1.1, у якому внесено суттєві зміни до опису структури мережі КРК, схематичне зображення якої наведено на рисунку 2. У сценарії роботи беруть участь: апаратне забезпечення КРК – QKDE; сутність управління ключами – KME; клієнти мережі – SAE. Видно, що відбулося суттєве спрощення структури мережі шляхом об'єднання сервера управління квантовими ключами та сервера управління ключами в єдиний об'єкт.

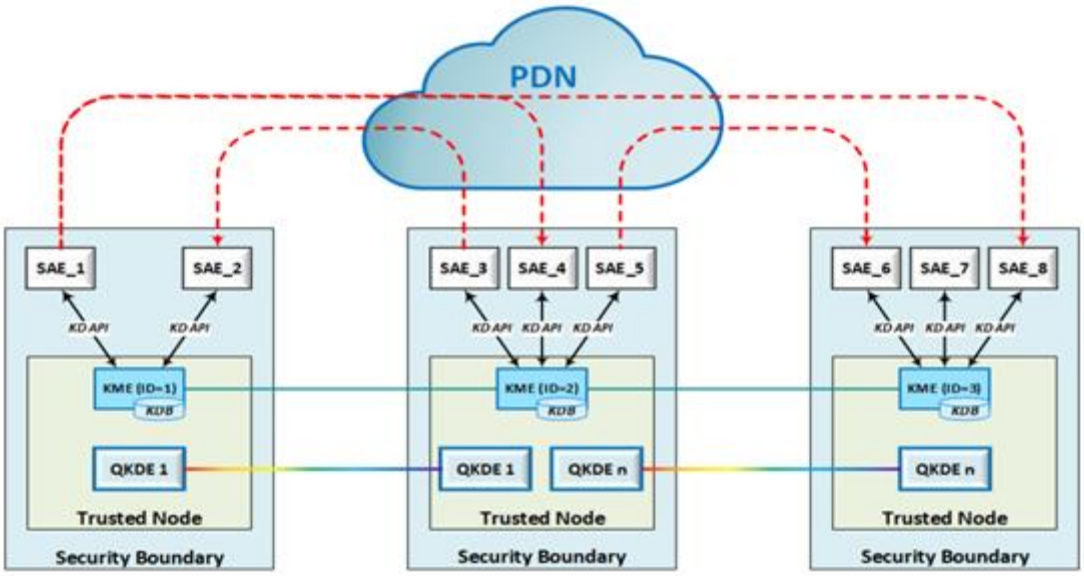


Рисунок 2 – Структура мережі КРК згідно з ETSI GS QKD 014

При цьому інтерфейс взаємодії описується з таких припущень: вузол працює та управляється безпечно; KME і SAE перебувають у межах одного вузла; інтерфейс взаємодії не виходить за межі контрольованої зони довіреного вузла; KME є безпечним; SAE є безпечним; KME має мати унікальний ідентифікатор у межах мережі КРК; SAE має мати унікальний ідентифікатор у межах мережі КРК.

Таким чином, початкова структура мережі містила чотири рівні, розділяючи управління квантовими ключами та формування спільних секретів згідно з запитами користувачів застосунків. При цьому користувацькі застосунки включалися до складу довірених вузлів мережі, що не зовсім коректно. У наступній версії структури мережі цей недолік був виправлений: користувацькі застосунки були винесені за межі мережі,. Водночас спрощення структури вузла шляхом об'єднання блоків управління квантовими ключами та взаємодії з користувацькими застосунками ускладнює вузол мережі, оскільки майже всі функції вузла концентруються в єдиному блоці.

В обох версіях мережі КРК не розглядаються способи розподілу спільного секрету для користувацьких застосунків, питання захисту каналів взаємодії вузлів та управління вузлами мережі. Для другої версії мережі КРК додані абстрактні вимоги щодо безпеки вузлів мережі, які не сприяють спрощенню побудови мережі КРК загалом і довірених вузлів зокрема.

Висновок. Детально проаналізовано структуру мережі КРК за версією ETSI, що дало можливість обгрунтувати вибір структури згідно конкретної задачі.

Перелік використаних джерел.

1. ETSI GS QKD 004 v.2.1.1 Quantum Key Distribution (QKD); Application Interface. Режим доступу: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/004/02.01.01_60/gs_QKD004v020101p.pdf.

2. ETSI GS QKD 014 V1.1.1 Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API.2019. Режим доступ: https://www.etsi.org/deliver/etsi_gs/QKD/001_099/014/01.01.01_60/gs_QKD014v010101p.pdf.

УДК 004.93

Максим СОКОЛІК, Сергій КУЛИНА

Західноукраїнський національний університет

АНАЛІЗ СУЧАСНИХ АЛГОРИТМІВ ВИДІЛЕННЯ ОЗНАК В БІОМЕТРІЇ

Вступ. У сучасному інформаційному суспільстві обсяги та важливість обробки даних стрімко зростають. Організації та окремі учасники оперують різноманітними даними, що потребують захисту – персональними даними, бізнес–інформацією, державними або науковими таємницями. У той же час зростає кількість кіберзагроз – фішинг, злом облікових записів, атаки «людина посередині», злочини із використанням витоків паролів, соціальна інженерія та інші [1]. Одним із методів забезпечення необхідного рівня доступу є застосування багатофакторної автентифікації (MFA). Її основою є те, що користувач має надати два або більше незалежних факторів підтвердження особи перед тим, як отримати доступ до ресурсу або системи [2]. Одним із таких факторів підтвердження є біометрична ідентифікація, яка застосовує риси унікальні для кожного користувача.

Метою дослідження є підвищення рівня захищеності систем доступу до даних та швидкодії їх роботи шляхом аналізу та покращення існуючих алгоритмів виділення ознак в біометриці.

1. Дослідження існуючих біометричних модальностей

Біометричні ідентифікатори, або як їх ще називають модальності, класифікуються на основі характеристик, які вимірюються. Вони поділяються на дві основні категорії [3]:

1. Фізіологічні модальності, категорія охоплює статичні, вроджені риси, пов'язані з формою та структурою тіла людини.

2. Фізіологічні риси, які демонструють високі показники унікальності та постійності протягом життя, так звані поведінкові модальності. Вони можуть бути менш постійними, оскільки на них можуть впливати інші фактори.

У таблиці 1 представлено аналіз біометричних модальностей, ключовим для яких є висока універсальність кожного із методів.

Таблиця 1 - Порівняльний аналіз біометричних модальностей

Модальність	Унікальність	Постійність	Збираність	Прийнятність	Захист
Відбиток пальця	Висока	Висока	Висока	Середня	Середня
Обличчя	Середня	Середня	Висока	Висока	Низька
Райдужна оболонка	Дуже висока	Дуже висока	Середня	Низька	Висока
Голос	Середня	Середня	Висока	Висока	Середня
Геометрія долоні	Середня	Висока	Висока	Висока	Середня
Хода	Низька	Середня	Висока	Висока	Середня
ДНК	Дуже висока	Дуже висока	Низька	Низька	Дуже висока

Отже, як бачимо з таблиці 1, кожну із модальностей можна характеризувати по одному із параметрів:

– унікальність – це оцінка того, наскільки кожна із розглянутих модальностей унікальна для користувача;

– постійність – це оцінка того, наскільки модальність користувача змінюється протягом життя;

– збираність – це оцінка того, наскільки складно зібрати зразок модальності;

– прийнятність – це оцінка того, наскільки система є задовільною, доречною та бажаною для використання у конкретному контексті;

– захист – це оцінка того, наскільки складно підробити саму модальність, а отже яким чином можливо її підробити.

Аналіз таких критеріїв виявляє що не існує "ідеальної" модальності, яка б мала найвищі бали за всіма сімома показниками. Така відсутність єдиного, простого, миттєвого чи чудодійного рішення для складної проблеми є основною рушійною силою для двох основних напрямків досліджень в біометрії, а саме:

1) розробки мультимодальних систем, які комбінують сильні сторони різних модальностей (наприклад, обличчя та голос);

2) постійного вдосконалення алгоритмів виділення ознак та зіставлення з метою максимізації ефективності та унікальності легкодоступних модальностей.

Окрім цього будь–яка біометрична система, незалежно від модальності, функціонує в одному з двох режимів: верифікація або ідентифікація. Його вибір визначає основне питання, на яке відповідає система, та має кардинальні наслідки для архітектури, обчислювальної складності та вимог до точності.

2. Аналіз сучасних алгоритмів виділення ознак

Процес біометричної ідентифікації складається з двох основних етапів: виділення ознак та зіставлення ознак. Виділення ознак – це процес перетворення вхідних необроблених даних (наприклад, зображення обличчя або відбитка пальця) у компактний числовий вектор (відомий як вектор ознак або шаблон). Цей вектор має бути достатньо інформативним, щоб бути унікальним для кожної особи, але водночас достатньо стабільним, щоб залишатися незмінним при незначних варіаціях (наприклад, різне освітлення або кут огляду). Історично, цей процес еволюціонував від "рукотворних" алгоритмів до підходів, що базуються на глибокому навчанні. Традиційні алгоритми виділення ознак покладаються на знання експертів у предметній галузі для розробки алгоритмів, які виділятимуть специфічні, заздалегідь визначені патерни. До таких алгоритмів відносять: виділення Мінуцій з відбитків, локальні бінарні шаблони та гістограми орієнтованих градієнтів. Традиційні методи (HOG, LBP, мінуції) базуються на інженерії ознак – процесі, де експерт вручну розробляє алгоритм, базуючись на припущеннях про те, які ознаки є важливими (наприклад, "градієнти важливі" або "текстура важлива"). На відміну від них, згорткові нейронні мережі (CNN) представляють фундаментальний зсув до навчання ознакам. Замість того, щоб програмувати екстрактор ознак вручну, CNN автоматично вивчає оптимальну ієрархію ознак безпосередньо з необроблених пікселів під час процесу навчання.

В таблиці 2 представлено порівняння ключових характеристик для традиційних метрик та згорткових нейронних мереж.

Таблиця 2 - Порівняння парадигм виділення ознак

Метрика	Традиційні методи	CNN
Принцип реалізації	Інженерія ознак	Навчання ознакам
Необхідність знань	Висока	Низька
Потреба в даних	Низька / Середня	Дуже висока
Обчислювальна складність	Низька	Висока
Енергоефективність	Висока	Низька
Потенційна точність	Середня / Висока	Дуже висока
Стійкість до перенавчання	Висока	Низька

Вибір між традиційними методами та CNN не є вибором між старою технологією та новою, це більше компроміс, що базується на трьох факторах.

1. Точність CNN полягає в тому, що зазвичай цей метод забезпечує вищу точність при складних завданнях.

2. Обчислювальна вартість. Традиційні методи, такі як HOG, є *значно* ефективнішими з точки зору енергоспоживання. Дослідження [4] вказує на "значну різницю у споживанні енергії" між CNN та HOG. Енергоспоживання та енергозаощадження є критично важливим для вбудованих та мобільних пристроїв, де час роботи від батареї є пріоритетом.

3. CNN потребують *великих* наборів даних для ефективного навчання. CNN "вигідні переважно для великих баз даних", оскільки на *малих* наборах даних добре налаштований алгоритм LBP або HOG може показати кращі результати через значно менший ризик перенавчання.

Висновок. Проведене дослідження показало, що методи біометричної аутентифікації є необхідним та доступним шдяхом покращення захищеності систем та даних, а для покращення їх швидкодії необхідно оптимізовувати методи виділення ознак. При практичній реалізації перед розробниками постає багато додаткових проблем, що є неявними при теоретичних дослідженнях. Відповідно, потужні CNN, є кращим вибором для *серверних* 1:N систем з великими об'ємами даних, де точність системи є пріоритетом. Натомість традиційні методи та спеціалізовані CNN є необхідними для *клієнтських* 1:1 систем з обмеженими ресурсами.

Перелік використаних джерел.

1. Яіцький А.О.(2022). Ефективні методи та засоби захисту фішингових атак.Problems of science and practice, tasks and ways to solve them, 417–419.

2. Suleski T, Ahmed M, Yang W, Wang E. A review of multi–factor authentication in the Internet of Healthcare Things. Digital Health. 2023; 9. doi:10.1177/20552076231177144.

3. Єжова, Є. (2025). Методи Захисту Від Атак На Біометричні Системи Аутентифікації. Міжнародна науково–практична конференція Інформаційні технології та комп'ютерне моделювання, 161–163.

4. Klingler, N. (2024). MobileNet – Efficient deep learning for mobile vision. [Ектронний ресурс].– Режим доступу: https://viso.ai/deep–learning/mobilenet–efficient–deep–learning–for–mobile–vision/

УДК 004

Остап ЛУКАШ

Західноукраїнський національний університет

ЗАСТОСУВАННЯ ШТУЧНОГО ІНТЕЛЕКТУ ТА МАШИННОГО НАВЧАННЯ ДЛЯ АУДИТУ БЕЗПЕКИ БЛОКЧЕЙН-СИСТЕМ

Вступ. Проблема безпеки смарт-контрактів залишається критичною, оскільки традиційні методи (статичний аналіз, фаззінг) мають суттєві обмеження, такі як високий рівень хибно-позитивних результатів та нездатність виявляти нові типи атак. Це зумовлює необхідність впровадження методів штучного інтелекту (AI) для автоматизації аудиту коду [1] зокрема:

- машинного (ML)
- глибокого навчання (DL).

Мета. Метою статті є дослідження можливостей застосування алгоритмів ML/DL у кібербезпеці блокчейну, а також порівняльний аналіз підходів на основі обробки природної мови (NLP) та графів потоку керування (CFG).

1. Штучний інтелект у задачах аналізу коду.

ML та DL трансформували підходи до безпеки, дозволяючи системам знаходити закономірності у даних без прямого втручання людини.

Machine Learning (ML) використовує алгоритми класифікації та регресії. Вдосконалення моделей значною мірою залежить від якості та кількості даних.

Deep Learning (DL) базується на штучних нейронних мережах і здатне обробляти великі масиви даних, проте часто характеризується як «чорна скринька».

2. Порівняння NLP та CFG підходів.

При аналізі коду смарт-контрактів виділяють два основні напрямки представлення даних.

- Обробка природної мови (NLP).
- Графи потоку керування (CFG).

NLP підхід розглядає код як послідовність символів або токенів (Text-based). Використовуються ембеддінги (наприклад, CodeBERT [2]) та n-грами. До переваг є низька/середня обчислювальна складність, ефективність для великих обсягів коду, стосовно недоліків - низька стійкість до обфускації (зміна імен змінних сильно впливає на результат), втрата інформації про потік даних.

Підхід CFG розглядає код як структуру, де вузли - це базові блоки, а ребра - переходи керування (Graph-based). Ключовою перевагою є висока стійкість до обфускації, збереження логічної структури програми, а до недоліків - висока обчислювальна складність, оскільки порівняння графів є NP-складною задачею.

Сучасні дослідження пропонують об'єднувати ці методи [3]. В таблиці 1 наведено порівняння розглянутих підходів.

Таблиця 1 – Порівняння NLP та CFG підходів

Критерій порівняння	Підхід NLP	Підхід CFG
Представлення	Послідовність токенів (CodeBERT).	Графова структура (вузли та ребра).
Стійкість до обфускації	Низька. Чутливість до перейменування.	Висока. Структура залишається незмінною.
Складність	Лінійна, швидка обробка.	Висока (NP-складна задача).
Втрата інформації	Втрачається потік даних між функціями.	Втрачається контекст (назви, коментарі).

Графові нейронні мережі (GNN) використовують структуру з CFG, але кожному вузлу присвоюють векторні ознаки, отримані через NLP. Схема з довгостроковою короткочасною пам'яттю (LSTM) для вбудовування базових блоків та індуктивним навчанням на основі GraphSAGE дозволяє отримати як точність графів, так і семантичне розуміння коду. Це вирішує проблему накладних витрат, характерну для чистих CFG методів [4].

Висновок. Аналіз смарт-контрактів вимагає гібридного погляду. Код слід розглядати не просто як текст (NLP) і не тільки як математичну структуру (CFG), а як послідовність інструкцій із чіткими логічними зв'язками. Використання GNN у поєднанні з методами глибокого навчання дозволяє створити системи самоадаптивної кібербезпеки, здатні прогнозувати потенційні атаки ефективніше за традиційні інструменти.

Перелік використаних джерел:

1. Smart Contract Vulnerability Detection using Graph Neural Network. [Електронний ресурс]. Режим доступу: https://scispace.com/pdf/smart-contract-vulnerability-detection-using-graph-neural-40egxpqrcq.pdf

2. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. [Електронний ресурс]. Режим доступу: https://arxiv.org/abs/2002.08155

3. Combining Graph Neural Networks with Expert Knowledge for Smart Contract Vulnerability Detection. [Електронний ресурс]. Режим доступу: https://messi-q.github.io/projects/papers/tkde/tkde.pdf

4. GraphSA: Smart Contract Vulnerability Detection Combining Graph Neural Networks and Static Analysis. [Електронний ресурс]. Режим доступу: https://www.researchgate.net/publication/374309511_GraphSA_Smart_Contract_Vulnerability_Detection_Combining_Graph_Neural_Networks_and_Static_Analysis

УДК 004.056

Степанюк О.В., Залізняк В.В., Касянчук М.М.

Західноукраїнський національний університет

АРХІТЕКТУРА ОБЧИСЛЮВАЛЬНОГО КОМПЛЕКСУ З БАГАТОРІВНЕВИМ КОНТРОЛЕМ ДОСТУПУ

Вступ. Актуальність розробки архітектури обчислювального комплексу з багаторівневим контролем доступу (КД) зумовлена зростаючими вимогами до захисту інформації в умовах інтенсивної цифровізації та переходу до розподілених обчислювальних середовищ [1]. Поява нових кіберзагроз вимагає впровадження гнучких та стійких архітектур, здатних запобігати несанкціонованому доступу навіть у разі часткового порушення безпеки. Багаторівневий КД дозволяє розмежовувати права на різних рівнях, що суттєво зменшує ризики внутрішніх і зовнішніх порушень [2]. Таким чином, дослідження та розробка архітектури обчислювального комплексу з багаторівневим КД є актуальною науковою та практичною задачею.

Мета. Метою даної роботи є розробка архітектури обчислювального комплексу з багаторівневим КД, що спрямована на підвищення стійкості інформаційних систем, мінімізацію ризиків порушення безпеки та забезпечення надійного функціонування критичних цифрових інфраструктур.

1. Архітектура обчислювального комплексу з багаторівневим контролем доступу

Стосовно систем обробки даних проблема КД зводиться, перш за все, до КД до різних інформаційних систем: операційної системи, баз даних, системи електронного документообігу тощо, до технічних засобів обробки даних та інших об'єктів. В останні десятиліття ця проблема ще більше загострилася в зв'язку з інтенсивним розвитком інтернету, що призвело до появи великої кількості нових видів загроз – мережевих. Ці види загроз виявилися істотно більш небезпечними, що породило великий сплеск у розвитку різних засобів і механізмів мережевого захисту комп'ютерних систем. Політична громадськість багатьох країн гостро ставить питання про забезпечення повного державного контролю над мережевими трафіками національного рівня, розроблення своїх програмно–апаратних засобів обробки даних.

Таким чином, об'єктами доступу, що становлять інтерес для зловмисних дій, є інформаційні системи об'єкта захисту, технічні засоби обробки даних, а також безпосередньо процеси обробки даних. Сучасна система доступу до обчислювальних систем включає велику кількість засобів контролю, ідентифікації та верифікації користувачів (рисунок 1).

Вимоги до забезпечення КД та умови функціонування можуть бути сформульовані наступним чином. Обчислювальний комплекс із системою захищеного КД по комп'ютерних мережах забезпечує можливість доступу з будь–якого пристрою, який використовує користувач.

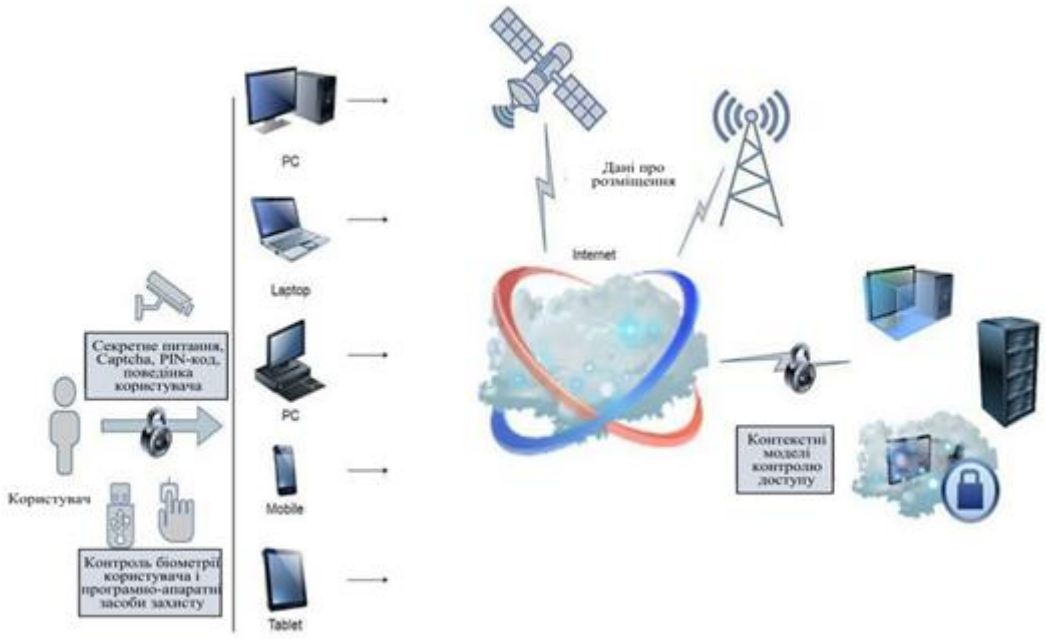


Рисунок 1 – Архітектура обчислювального комплексу з багаторівневим КД

При цьому відбувається ідентифікація користувача з використанням різних засобів біометрії (наприклад, відбиток пальця, вбудовані або зовнішні камери), в процесі доступу використовуються програмні засоби верифікації (секретне питання, PIN–коди, Captcha). В даний час розвиваються системи, пов'язані з аналізом поведінки користувачів (наприклад, реакції користувачів на запитання доступу чи поведінкові моделі). Елементом КД є облік локації користувача, ці дані можуть бути порівняні з даними у профілі користувача. Для доступу до сервісів існує рівень КД, що забезпечується контекстними моделями. Такими моделями можуть бути політики, побудова моделей за журналами подій, відповідним групам користувачів, а також спеціалізовані методики, що належать до предметної області сервісу – медичні, банківські, освітні, які мають свої спеціалізовані протоколи та особливості доступу На рівні обчислювальних ресурсів – серверів, віртуальних машин, баз даних, здійснюється апаратний КД.

Таким чином, змістовно процес доступу до обчислювальної системи описується відношенням певної групи суб'єктів до заданої сукупності ресурсів, до яких ці суб'єкти хочуть мати доступ. Для опису, формалізації, аналізу або синтезу системи КД необхідно, перш за все, описати три основні його компоненти: суб'єктів, що беруть участь у системі КД; ресурси; об'єкти, які є предметом інтересів суб'єктів.

Зазначимо, що у процесі КД ставиться завдання забезпечення безпеки всіх необхідних властивостей (або, узагальнено усієї обчислювальної системи) даних обмеженого доступу.

Під суб'єктом розуміється не лише фізична чи юридична особа, яка бажає скористатися певним програмно–апаратним чи інформаційним ресурсом в інформаційній системі, а також інформаційні процеси, програмно–апаратні засоби, які можуть отримати доступ до обчислювальної системи. Таким чином, під суб'єктом розуміється будь–який об'єкт або процес, який може здійснити певну дію в інформаційній системі з використанням процесорних пристроїв

системи. Приклади суб'єктів програмно–апаратного типу: операційна система; антивірусні та мережеві засоби; інформаційні системи конкретного призначення (редактори, СУБД, системи електронного документообігу, системи для обслуговування бізнес–процесів, зокрема, 1С тощо).

Під об'єктом, що здійснює доступ, розуміються будь–які поіменовані елементи інформаційної системи, до яких може отримати доступ хоча б один з процесорних пристроїв: периферійне та мережеве обладнання; файли; носії інформації; зовнішні пристрої різного призначення. Більше того, одні об'єкти доступу можуть бути частиною інших об'єктів (якщо вони поіменовані та доступні для процесорів); наприклад, флешка та файли, записані на ній є об'єктами доступу, такими ж є база даних і окремі записи, що містяться в ній, міжмережевий екран та окремі його параметри, доступні для процесорів.

Доступ описує безпосередньо набір тих дій, які може здійснювати даний суб'єкт над цим об'єктом. Таким чином, доступ прив'язаний до пари «суб'єкт–об'єкт» і являє собою набір дозволених (санкціонованих) дій, які може вчиняти суб'єкт над об'єктом. Основними видами доступів є читання та запис даних. Багато інших видів дій з даними можуть з деяким ступенем умовності зведені до цих видів доступу. Наприклад, видалення файлу може розглядатися як запис порожніх даних у файл – у цьому випадку порожній файл ототожнюється з віддаленим файлом, що в цілому не завжди коректно. Далі, додавання даних у файл ототожнено із записом даних. Однак, якщо дані структуровані (наприклад, у базах даних), то подібне додавання вимагає попереднього формування структурного скелета нового запису, що вже не вкладається у схему просто запису даних у файл. Тому до перерахованих видів дій, які можуть входити в доступ, додаються також видалення об'єкта (файлу, пристрою з переліку тощо), додавання об'єкта (записи, пристрої), активації об'єкта (запуску програми, включення технічного пристрою), модифікація (заміна), блокування, контроль права власності (включаючи авторизацію, підтвердження авторства чи авторських прав). До складу можливих видів доступу можуть бути включені інші більш специфічні дії.

Процес формування та реалізації доступів для кожної пари «суб'єкт–об'єкт» із заданих переліків суб'єктів та об'єктів, а також порядок зміни цих доступів у процесі функціонування інформаційної системи становить основний зміст політики управління доступом.

Висновок. Розроблено архітектуру обчислювального комплексу з багаторівневим контролем доступу.

Перелік використаних джерел.

1. Gil–Garcia J.R., Flores–Zúñiga M.Á. Towards a comprehensive understanding of digital government success: Integrating implementation and adoption factors. Government Information Quarterly. 2020. Vol.37. No. 4. P. 101518.

2. Cai F., He J., Ali Zardari Z., Han S. Distributed management of permission for access control model. Journal of Intelligent & Fuzzy Systems. 2020. Vol. 38. No. 2. P. 1539–1548.

УДК 004.6

Вадим ХМЕЛИК

Західноукраїнський національний університет

ДОСЛІДЖЕННЯ АРХІТЕКТУРИ ОПЕРАЦІЙНОГО ЦЕНТРУ БЕЗПЕКИ

Вступ. Операційний центр безпеки (Security Operations Center, SOC) є ключовим елементом системи кіберзахисту, який забезпечує централізоване виявлення, аналіз і реагування на загрози. Однак використання комерційних рішень для побудови SOC може бути економічно недоцільним, особливо для навчальних закладів, малих підприємств і дослідницьких лабораторій. Тому актуальним є створення SOC на основі інструментів з відкритим кодом, що дозволяє досягти високого рівня функціональності при мінімальних витратах.

Мета: розробка архітектури SOC на основі опен–сорс інструментів, що забезпечує виявлення, аналіз, реагування та вдосконалення стану кібербезпеки інформаційних систем.

1. Дослідження функціональних можливостей SOC

SOC є ключовим елементом інформаційної безпеки організації оскільки забезпечує комплексний моніторинг, аналіз та реагування на кіберзагрози, виконуючи основні функції, що наведені на рисунку 1 [1].

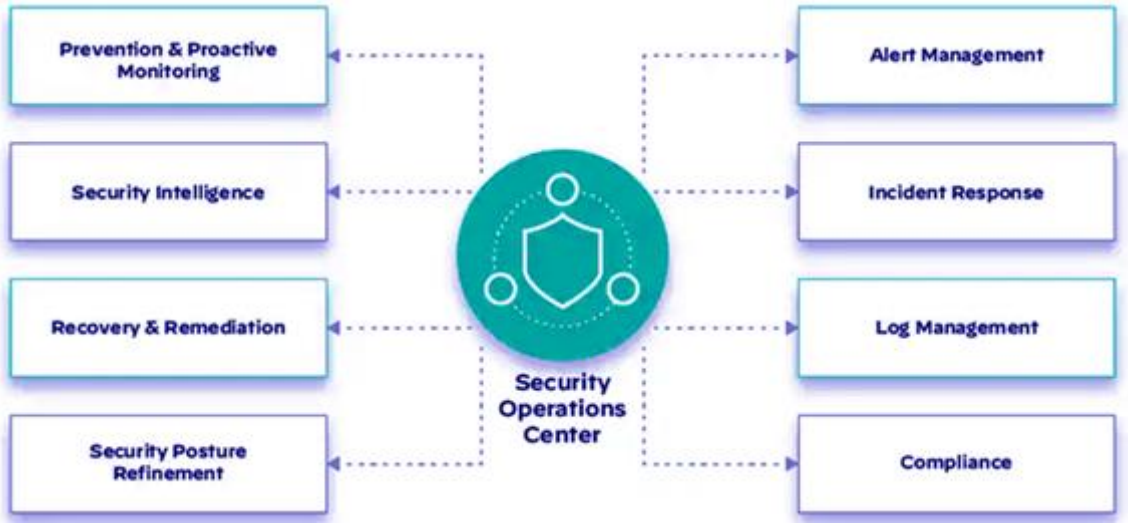


Рисунок 1 – Функції SOC

Функція моніторингу безпеки передбачає здійснення спостереження за мережею, системами та додатками з метою виявлення підозрілих активностей, аномалій та потенційних загроз. Складові моніторингу включають:

– профілактичний моніторинг – регулярна перевірка систем та мереж для запобігання потенційним загрозам;

– проактивний моніторинг – виявлення прихованих атак і ранніх ознак зловмисної активності.

– Функція розвідка безпеки абезпечує збір та аналіз даних про потенційні кіберзагрози та включає

– виявлення та аналіз інцидентів – збір даних про потенційні кіберзагрози з використанням систем виявлення вторгнень (IDS/IPS), логів та

аналітики для ідентифікації і класифікації інцидентів;

– управління загрозами та вразливостями – реалізує ідентифікацію потенційних уразливостей, планування патч–менеджменту та впровадження превентивних заходів.

Розвідка безпеки допомагає SOC передбачати нові загрози та виявляти слабкі місця в системах організації, забезпечуючи проактивний захист.

Функція відновлення та виправлення після інциденту SOC забезпечує відновлення нормальної роботи систем і усунення наслідків атаки, включаючи відновлення даних, виправлення конфігурацій і впровадження патчів.

Вдосконалення стану безпеки – дана функція передбачає постійне підвищення рівня захисту інформаційних систем організації. SOC аналізує слабкі місця, впроваджує превентивні заходи та удосконалює процеси безпеки. Забезпечує можливості аналізу та генерації звітів про стан безпеки, тренди інцидентів та оцінка ефективності впроваджених заходів безпеки.

Керування сповіщеннями – функція, що забезбечує обробку сигналів від систем безпеки (IDS, IPS, SIEM) дозволяє своєчасно реагувати на потенційні загрози та включає пріоритизацію та фільтрацію подій для ефективного реагування.

Функція реагування на інциденти передбачає виконання заходів для локалізації, ізоляції та усунення загроз, включаючи карантин шкідливого ПЗ, блокування атак або відновлення систем.

Керування журналами забезпечує збір та аналіз логів дозволяє відстежувати активність користувачів та систем. Ця функція є ключовою для виявлення аномальної поведінки та подій безпеки, а також для проведення подальшого аналізу інцидентів.

Відповідність (Compliance) – функція SOC, що забезпечує дотримання організацією вимог нормативних актів і стандартів (наприклад, GDPR, PCI–DSS, HIPAA), зокрема включає регулярні аудити, створення звітів та підтримку необхідних процедур контролю.

2. Інструменти SOC з відкритим кодом

Після визначення основних функцій SOC важливим етапом є вибір відповідних інструментів для їх реалізації. На даний час організаціями застосовуються різноманітні програмні рішення для забезпечення моніторингу, аналізу інцидентів, реагування на загрози та управління безпекою. Особливо актуальними є open source інструменти, які дозволяють створювати ефективний SOC без значних фінансових витрат, одночасно зберігаючи високу гнучкість та масштабованість [2].

В таблиці 1 наведені основні категорії інструментів з відкритим кодом, їх реалізацій та призначення у контексті SOC.

Поєднання чітко визначених функцій SOC та спеціалізованих інструментів дозволяє організації забезпечити ефективний моніторинг, проактивне виявлення загроз, швидке реагування на інциденти та підтримку нормативної відповідності. Використання опен–сорс рішень забезпечує гнучкість, масштабованість та можливість інтеграції різних компонентів SOC у єдину систему.

Таблиця 1 – Інструменти з відкритим кодом для реалізації SOC

Категорія	Інструменти	Призначення
SIEM (Security Information and Event Management)	Wazuh, OSSIM, ELK Stack (Elasticsearch, Logstash, Kibana), Graylog, Splunk Free	Збір, нормалізація та аналіз логів, кореляція подій, створення дашбордів та звітів
IDS/IPS	Snort, Suricata, Zeek (Bro), OSSEC	Виявлення мережевих атак, аномалій та підозрілих активностей
Антивірус / Malware Detection	ClamAV, YARA, інтеграція з VirusTotal API, LMD (Linux Malware Detect)	Виявлення шкідливого програмного забезпечення та аналіз загроз
Моніторинг мережі	Nagios, Zabbix, Prometheus, Icinga, Netdata	Системний та мережевий моніторинг, оповіщення про відхилення та аномалії
Управління інцидентами	TheHive, Cortex, RTIR (Request Tracker for Incident Response)	Автоматизація реагування на інциденти, розслідування та управління завданнями
Хмарна безпека / Threat Intelligence	MISP (Malware Information Sharing Platform), OpenCTI	Обмін, кореляція та аналіз інформації про загрози, інтеграція з зовнішніми джерелами Threat Intelligence

3. Архітектура SOC на основі інструментів з відкритим кодом

Архітектура SOC з використанням відкритих інструментів передбачає інтеграцію декількох компонентів для комплексного захисту. Метою запропонованої архітектури (рисунок 2) є забезпечення централізованого збору, нормалізації, кореляції та аналізу подій безпеки з подальшим автоматизованим і ручним реагуванням, зручним візуальним поданням і збереженням доказової інформації для розслідування інцидентів та звітності. Архітектура спроектована з урахуванням принципів модульності, масштабованості, відмовостійкості та можливості інтеграції нових компонентів.

1. Джерела даних:
   – Логи серверів, мережевих пристроїв, кінцевих вузлів.
   – Події систем безпеки (антивіруси, IDS/IPS).
   – Загрози з відкритих баз (Threat Intelligence).
2. Збір та нормалізація даних:
   – Використання Wazuh Agent, Logstash, Fluentd для централізації та нормалізації логів.
3. Кореляція та аналіз подій:
   – SIEM–системи (Wazuh, OSSIM) обробляють дані, визначають аномалії та потенційні загрози.
4. Виявлення загроз:
   – IDS/IPS (Snort, Suricata) аналізують мережевий трафік

– Антивірусні рішення (ClamAV, YARA) перевіряють файли та процеси на вузлах.

5. Реагування на інциденти:

– TheHive та Cortex автоматизують процес розслідування та реагування.

– Active Response в Wazuh дозволяє ізолювати або видаляти загрози.

6. Візуалізація та звітність:

– Kibana та Grafana забезпечують дашборди, аналітичні панелі та історичні звіти.

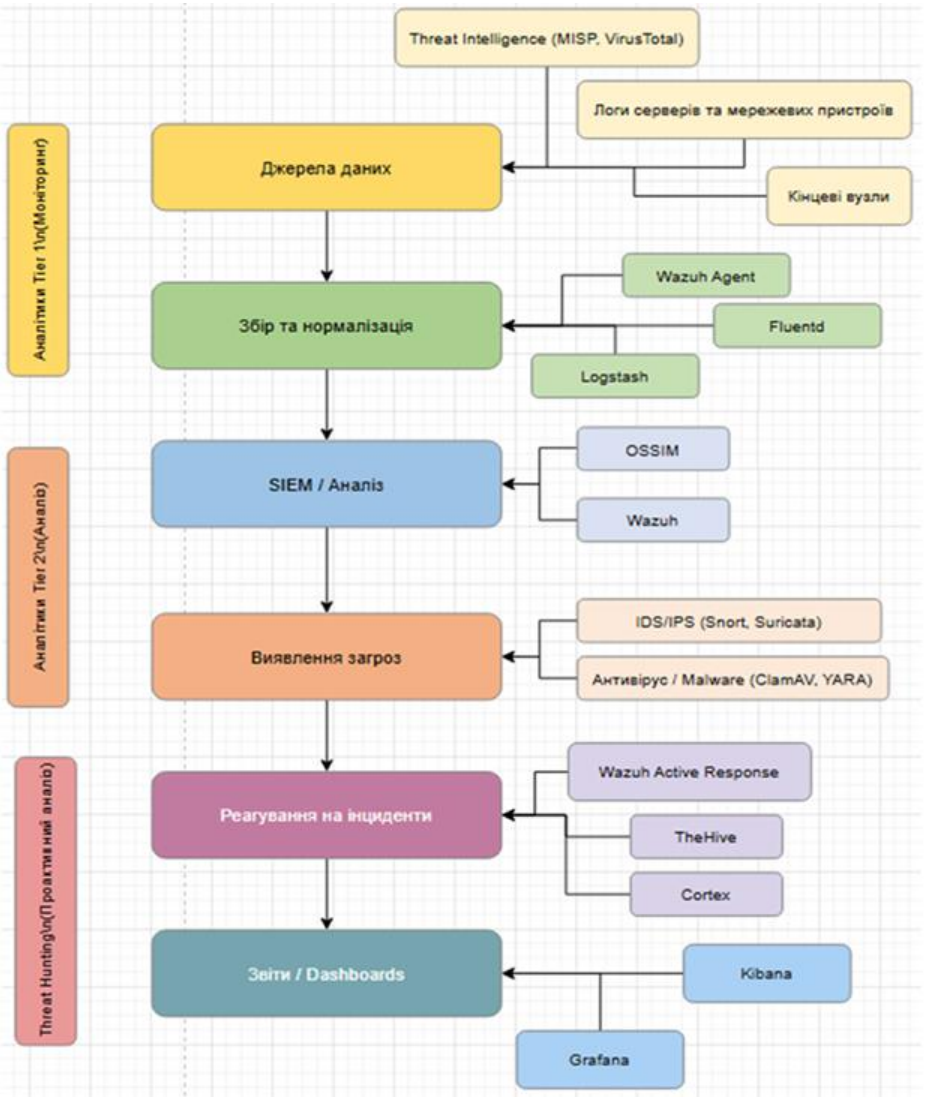


Рисунок 2 – Архітектура SOC на основі інструментів з відкритим кодом

Висновок. Така архітектура дозволяє побудувати SOC без значних фінансових витрат, використовуючи лише відкриті рішення, при цьому забезпечуючи комплексний захист корпоративної інфраструктури.

Перелік використаних джерел.

1. Що таке SOC та навіщо він потрібен. [Електронний ресурс].– Режим доступу: https://gigatrans.ua/ua/news/chto–takoe–soc–i–zachem–on–nuzhen

2. 10 Open–Source SOC tools. [Електронний ресурс].– Режим доступу: https://www.wiz.io/academy/open–source–soc–tools

УДК 004.056.5

Максим ЧУХНІЙ, Андрій ВЕЛЕЩУК

Західноукраїнський національний університет

СУЧАСНІ ЗАГРОЗИ БЕЗПЕКИ ВЕБ–ДОДАТКІВ

Вступ. Веб–додатки відіграють ключову роль у забезпеченні комунікації, обміну даними та наданні послуг. Однак разом із їхнім розвитком зростає і кількість кіберзагроз, які ставлять під ризик конфіденційність, цілісність і доступність інформації.

Актуальність проблеми обумовлена постійною еволюцією методів атак і зростаючою складністю веб–архітектур. У цьому контексті особливо важливо вивчати сучасні загрози безпеки веб–додатків і способи їх нейтралізації.

Метою аналізу є виявлення та класифікація сучасних загроз безпеки веб–додатків, а також оцінка їхнього впливу на функціонування систем і захист даних користувачів. Особлива увага приділяється методам виявлення вразливостей і розробці ефективних засобів протидії кіберзагрозам.

1. SQL ін'єкції

Розглянемо типові вразливості, яким піддаються багато веб–застосунків. Як і належить, атаки класу «Ін'єкції» займають провідну позицію рейтингу OWASP Top 10, зустрічаючись практично повсюдно і будучи дуже різноманітними в реалізації. Уразливості подібного класу починаються SQL–ін'єкціями, у різних його варіаціях, і закінчуючи RCE – віддаленим виконанням коду.

Схему проведення атаки приведено на рисунку 1.

SQLi: http://example.com/?id=1' union select 1,2,version(),4

RCE: http://example.com/search.php?q=;+cat+/etc/passwd

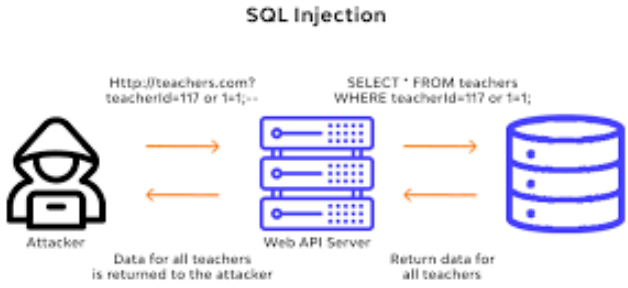


Рисунок 1. Схема виконання SQL–ін'єкції

2. XSS вразливості

Міжсайтовий скриптинг – вразливість, що наразі зустрічається куди ріже, ніж раніше, якщо вірити рейтингу OWASP Top 10, але незважаючи на це не стала менш небезпечною для веб–додатків та користувачів. Особливо для користувачів, адже атака XSS націлена саме на них. У загальному випадку зловмисник впроваджує скрипт у веб–додаток, який спрацьовує для кожного користувача, який відвідав шкідливу сторінку.

Схема експлуатації XSS вразливостей приведена на рисунку 2.

http://example.com/?search=<script>alert('xss')</script>

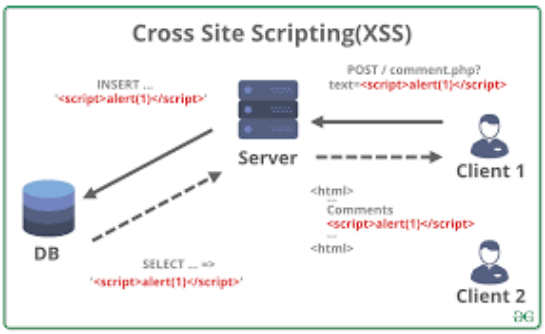


Рисунок 2. Схема експлуатації XSS вразливостей

3. Вразливості LFI/RFI

Уразливості даного класу дозволяють зловмисникам через браузер включати локальні та віддалені файли на сервері у відповідь від веб–програми. Цей пролом є там, де відсутня коректна обробка вхідних даних, якою може маніпулювати зловмисник, інжектувати символи типу path traversal і включати інші файли з веб–сервера.

Схема експлуатації вразливості LFI/RFI приведена на рисунку 3.

http://example.com/?search=/../../../../../../etc/passwd

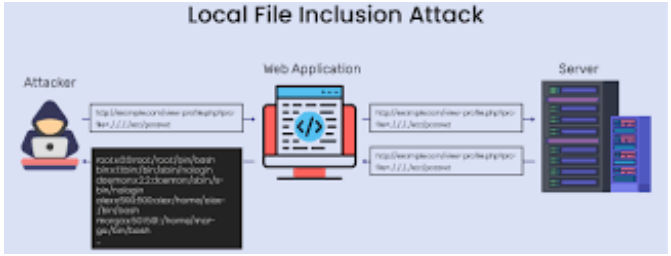


Рисунок 3. Експлуатація вразливості LFI/RFI

4. Атаки через JSON та XML

Веб–програми та API, що обробляють запити у форматі JSON або XML, також схильні до атак, оскільки такі формати мають свої недоліки.

JSON (JavaScript Object Notation) – це полегшений формат обміну даними, що використовується для зв'язку між програмами. Він схожий на XML, але простіше та краще підходить для обробки за допомогою JavaScript. Багато веб–додатків використовують цей формат для обміну даними між собою та серіалізації/десеріалізації даних. Деякі веб–програми також використовують JSON для збереження важливої інформації, наприклад даних користувача. Зазвичай використовується в RESTful API та додатках AJAX.

JSON найчастіше асоціюється з API, проте часто використовується навіть у звичайних і добре відомих веб–додатках. Наприклад, редагування матеріалів WordPress проводиться безпосередньо через відправку запитів у форматі JSON:

POST /index.php?rest_route=%2Fwp%2Fv2%2Fposts%2F12&_locale=user HTTP/1.1

Host: wordpress.example.com

...

%Інші заголовки%

...

{"id":12,"title":"test title","content":"test body","status":"publish"}

Проста ін'єкція JSON на стороні сервера може бути виконана в PHP таким чином:

Сервер зберігає дані користувача у вигляді рядка JSON, включаючи тип облікового запису.

Ім'я користувача та пароль приймаються безпосередньо з введення користувача без очищення;.

Рядок JSON формується за допомогою простої конкатенації:

```
$json_string = '{"account":"user","user":"'.$_GET['user'].'","pass":"'.$_GET['pass'].'"}'
```

Зловмисник додає дані до свого імені користувача:

john%22,%22account%22:%22administrator%22

Результуючий рядок JSON:

```
{
        "account":"user",
        "user":"john",
        "account":"administrator",
        "pass":"password"
}
```

При читанні збереженого рядка парсер JSON (json_decode) виявляє два account–записи і бере останній, надаючи права адміністратора користувачеві john.

Проста ін'єкція JSON на стороні клієнта може бути виконана таким чином:

Рядок JSON такий самий, як у наведеному вище прикладі;

Сервер отримує рядок JSON із ненадійного джерела;

Клієнт аналізує рядок JSON, використовуючи eval:

```
var result = eval("("+json_string+")");
document.getElementById("#account").innerText = result.account;
document.getElementById("#user").innerText = result.name;
document.getElementById("#pass").innerText = result.pass;
```

Значення account:

```
user"});alert(document.cookie);({"account":"user
```

Функція eval виконує alert.

Виконання призводить до XSS та отримання document.cookie. Приклад застосування вразливості приведено на рисунку 4.

Рисунок 4. Схема виконання JSON Injection

Захоплення JSON – атака, яка в певному сенсі схожа на підробку міжсайтових запитів (CSRF), при якій зловмисник намагається перехопити дані JSON, надіслані веб–додатку з веб–сервера.

Атакуючий створює шкідливий веб–сайт і вбудовує скрипт у свій код, який намагається отримати доступ до даних JSON від цільової веб–програми. Користувач, що взаємодіє з цільовим інтернет–ресурсом, буває шкідливий сайт (наприклад, за рахунок прийомів соціальної інженерії). Оскільки політика однакового походження (SOP) дозволяє включати та виконувати JavaScript із будь–якого сайту в контексті будь–якого другого сайту, користувач отримує доступ до даних JSON. Шкідливий сайт перехоплює дані JSON. Схематично це зображено на рисунку 5.

Рисунок 5. Схема виконання JSON Hijacking

Висновок. У результаті проведеного аналізу встановлено, що веб–додатки залишаються вразливими до широкого спектра сучасних загроз, серед яких найпоширенішими є SQL–ін'єкції, міжсайтові скриптові атаки (XSS), атаки типу CSRF, а також вразливості, пов'язані з неправильною автентифікацією та управлінням сесіями. Постійна еволюція кіберзлочинних методів вимагає від розробників і фахівців з кібербезпеки регулярного оновлення знань та використання сучасних засобів захисту. Важливим чинником забезпечення безпеки є впровадження системного підходу до тестування та моніторингу веб–додатків. Таким чином, підвищення рівня безпеки можливе лише за умови комплексного підходу, що поєднує технічні, організаційні та освітні заходи.

Перелік використаних джерел.

1. Rasal L.A., Attar V.Z. Web browser architecture proposal with local agent, International Conference on Intelligent Agent & Multi–Agent Systems, Chennai, India, 2009, pp. 1–5, doi: 10.1109/IAMA.2009.5228079.

2. Min B., Varadharajan V. Rethinking Software Component Security: Software Component Level Integrity and Cross Verification. The Computer Journal, vol. 59, no. 11, pp. 1735–1748, Nov. 2016, doi: 10.1093/comjnl/bxw047.

3. Gamboa H., Fred A.L.N., Jain A.K. Webbiometrics: User Verification Via Web Interaction," 2007 Biometrics Symposium, Baltimore, MD, USA, 2007, pp. 1–6, doi: 10.1109/BCC.2007.4430552.

4. Li.J., Li H. Evolution of Application Security based on OWASP Top 10 and CWE/SANS Top 25 with Predictions for the 2025 OWASP Top 10, International Conference on Inventive Computation Technologies (ICICT), Kirtipur, Nepal, 2025, pp. 1178–1183, doi: 10.1109/ICICT64420.2025.11004742.

5. Choiriyah A., Qomariasih N. Security Analysis on Websites Belonging to the Health Service Districts in Indonesia Based on the Open Web Application Security Project (OWASP) Top 10 2021, International Conference on Information Technology and Computing (ICITCOM), Yogyakarta, Indonesia, 2023, pp. 267–272, doi: 10.1109/ICITCOM60176.2023.10442816.

УДК 004.056.5:004.738.5:336.74

Роман ЩИПАНСЬКИЙ, Бабала ЛЮДМИЛА

Західноукраїнський національний університет

ТЕОРЕТИЧНІ ОСНОВИ БЕЗПЕКИ БЛОКЧЕЙН–ТЕХНОЛОГІЙ ТА КРИПТОВАЛЮТНИХ ТРАНЗАКЦІЙ

Всуп. Блокчейн–технологія стала фундаментальною основою сучасної цифрової економіки, забезпечуючи децентралізоване зберігання даних та проведення криптовалютних транзакцій. Однак, незважаючи на високий рівень криптографічного захисту, блокчейн–системи залишаються вразливими до різноманітних кіберзагроз, що потребує комплексного дослідження методів забезпечення їх безпеки.

Мета дослідження – проаналізувати теоретичні основи безпеки блокчейн–технологій, виявити основні вразливості та систематизувати сучасні методи захисту криптовалютних транзакцій.

1. Фундаментальні принципи блокчейн–технологій

Блокчейн являє собою розподілену базу даних, що складається з послідовно пов'язаних блоків, кожен з яких містить хеш попереднього блоку, забезпечуючи незмінність даних. Криптографічну основу становлять хеш–функції SHA–256 (Bitcoin) та Keccak–256 (Ethereum), які гарантують цілісність інформації [1]. Механізми консенсусу відіграють ключову роль у забезпеченні узгодженості транзакцій. Proof of Work (PoW) забезпечує високу безпеку через розв'язання складних обчислювальних задач, але потребує значних енергетичних витрат. Proof of Stake (PoS) зменшує енергоспоживання на основі принципу підтвердження частки володіння, однак створює ризики централізації [2].

Архітектурні особливості та масштабування. Структура блокчейну включає повні вузли (full nodes), що зберігають увесь ланцюг блоків, та легкі вузли (light nodes) для економії ресурсів (рисунок 1).

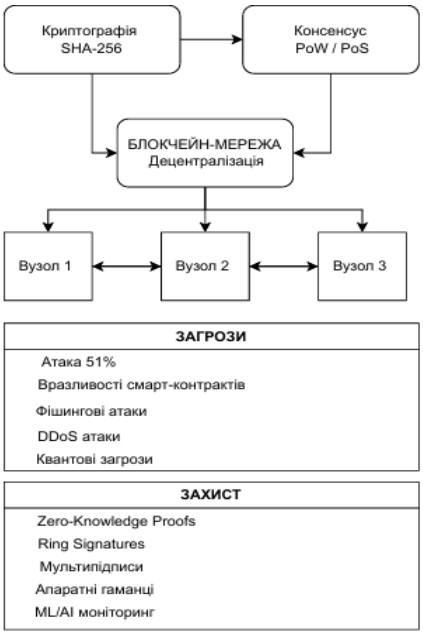


Рисунок 1 – Архітектура безпеки блокчейн–систем

Проблема масштабованості вирішується через впровадження шардингу – розділення мережі на менші частини для паралельної обробки транзакцій, та Layer 2 рішення (Lightning Network, zk–Rollups) [3].

Серед основних загроз виділяються: атака 51% (контроль над більшістю обчислювальної потужності мережі), вразливості смарт–контрактів (як у випадку інциденту з DAO, де було втрачено понад 50 млн доларів), фішингові атаки на приватні ключі користувачів, DDoS–атаки для перевантаження мережі, та потенційна загроза з боку квантових обчислень для існуючих криптографічних алгоритмів [4].

Сучасні методи захисту криптовалютних транзакцій. Забезпечення безпеки досягається через комплексний підхід [4]:

– криптографічні методи: асиметричне шифрування, цифрові підписи (ECDSA, EdDSA), Zero–Knowledge Proofs для підтвердження достовірності без розкриття інформації;

– методи анонімізації: CoinJoin, Ring Signatures, Stealth Addresses, протоколи конфіденційності (MimbleWimble);

– організаційні заходи: мультипідписи, апаратні гаманці, механізми розподіленого управління ключами (Shamir's Secret Sharing, Social Recovery Wallets). Розроблено удосконалений алгоритм розслідування кіберінцидентів з інтеграцією ML/AI аналізу загроз, що включає чотири послідовні етапи: реєстрацію з автоматизованим визначенням пріоритету, аналіз загрози з використанням машинного навчання, реагування через спеціалізовані інструменти нейтралізації, та закриття з аналізом першопричин і оновленням бази знань.

Висновки. Дослідження показало, що безпека блокчейн–систем вимагає багаторівневого підходу, що поєднує технічні (удосконалення протоколів, стійкість смарт–контрактів, криптографічні методи) та організаційні заходи (навчання користувачів, стандарти безпеки, механізми швидкого реагування). Впровадження ML/AI технологій у процеси моніторингу та реагування на інциденти створює адаптивну систему безпеки, здатну еволюціонувати паралельно з розвитком загроз. Виявлені тенденції вказують на необхідність подальшого вдосконалення методів захисту, особливо в контексті розвитку квантових обчислень та штучного інтелекту.

Перелік використаних джерел

1. Nakamoto S. Bitcoin: A Peer–to–Peer Electronic Cash System. 2008. [Ектронний ресурс].– Режим доступу: https://bitcoin.org/bitcoin.pdf

2. Zheng Z., Xie S., Dai H., Chen X., Wang H. Blockchain challenges and opportunities: A survey. International Journal of Web and Grid Services. 2018. Vol. 14. No. 4. P. 352–375.

3. Poon J., Dryja T. The Bitcoin Lightning Network: Scalable Off–Chain Instant Payments. 2016. [Ектронний ресурс].– Режим доступу: https://lightning.network/lightning–network–paper.pdf

4. Conti M., Kumar E. S., Lal C., Ruj S. A Survey of Security Threats in Blockchain Systems. ACM Computing Surveys. 2022. Vol. 55. No. 3. P. 1–36.

УДК 681.51

Якименко Н., Слободян В., Якименко Ю., Хомяк Р.

Західноукраїнський національний університет

МЕТОДОЛОГІЯ КІЛЬКІСНОГО МОДЕЛЮВАННЯ КІБЕРРИЗИКІВ ДЛЯ ПІДТРИМКИ УПРАВЛІНСЬКИХ РІШЕНЬ В ОРГАНІЗАЦІЯХ

Вступ. Сучасні цифрові платформи, корпоративні інформаційні системи та мережеві сервіси перетворилися на критично важливі елементи діяльності організацій, що призвело до зростання залежності бізнесу від стійкості IT–інфраструктури. Одночасно із цим масштаби та складність кібератак суттєво збільшились, а економічні наслідки інцидентів почали вимірюватися мільйонами доларів. У таких умовах питання кількісного оцінювання кіберризиків набуває ключового значення, оскільки традиційні якісні методи (“високий–середній–низький”) не дають можливості виміряти реальні можливі втрати, порівняти альтернативні засоби захисту та обґрунтувати інвестиції [1].

Обмеженість якісних підходів пов’язана з їхньою суб’єктивністю та неврахуванням складних імовірнісних сценаріїв. Крім того, кіберзбитки мають асиметричний характер: більшість інцидентів є маловитратними, однак поодинокі атаки (витік даних, компрометація сервера, злом облікових записів адміністратора) можуть формувати «важкий хвіст» розподілу втрат. Саме тому сучасні моделі ризиків потребують використання статистичних розподілів, математичного апарату симуляцій та гібридного підходу до формування сценаріїв [2].

Розроблена методологія поєднує історичні дані, експертні оцінки та математичне моделювання, що дозволяє створювати реалістичну криву ризику, необхідну для управлінських рішень.

Мета: Розробка методології кількісного моделювання кіберризиків для підтримки управлінських рішень в організаціях.

1 Концептуальні основи моделювання кіберризиків на основі даних про інциденти

Методологія базується на формуванні математичної моделі збитків від кіберінцидентів, де загальна вартість події розглядається як сума її окремих компонент. Формально це описується як:

$$C = C_{invest} + C_{rep} + C_{conf} + C_{IP} + C_{bus}, \quad (1)$$

де C_{invest} – витрати на розслідування,

C_{rep} – втрати репутації,

C_{conf} – збитки від компрометації конфіденційних даних,

C_{IP} – шкода інтелектуальній власності,

C_{bus} – втрати, пов’язані з порушенням бізнес–процесів.

Оскільки кожен компонент має стохастичний характер, він розглядається як випадкова величина з певним розподілом [3]. Для моделювання використовуються логнормальний, експоненційний або Парето–розподіли залежно від типу інциденту. Наприклад, витрати на компрометацію даних

описуються логнормальним розподілом:

$$C_{conf} \sim Lognormal(\mu, \sigma), \quad (2)$$

що відображає природну асиметрію втрат.

Частота появи інцидентів моделюється як пуассонівський процес:

$$N(t) \sim Poisson(\lambda t), \quad (3)$$

де λ – середня частота інцидентів за одиницю часу.

Композицію річних збитків визначає випадкова сума:

$$L = \sum_{i=1}^{N} C_i, \quad (4)$$

що є класичною моделлю сукупного збитку у задачах ризик–аналізу. Внаслідок стохастичності як частоти, так і величини втрат загальний розподіл річного збитку не має аналітичного розв'язку, тому використовується метод Монте–Карло.

У симуляціях генерується 10–100 віртуальних років діяльності організації, кожен із яких включає випадкову кількість інцидентів та їхню випадкову інтенсивність. Результатом є крива ризику – залежність імовірності перевищення певного рівня втрат: P(L>x), яка дозволяє визначати ключові показники, зокрема: Value–at–Risk (*VaR*) $VaR_{0.95}=\inf\{x:P(L\leq x)\geq 0.95\}$, та Expected Shortfall (*ES*) $ES_{0.95}=E[L|L>VaR_{0.95}]$.

Ці показники дозволяють прогнозувати не лише середній рівень збитків, а й очікувані максимальні втрати в гірших сценаріях.

2. Експериментальні результати

Методика була протестована на модельному наборі даних, що включає понад 20 000 записів про кіберінциденти різних типів. За результатами калібрування встановлено, що веб–атаки є найпоширенішими (38% від усіх подій), але найбільші збитки пов'язані з витоком даних, де максимальні втрати в окремих випадках перевищували 250 тис. доларів.

Симуляція 50 річних сценаріїв показала такі результати:

Середні річні очікувані збитки (Mean Annual Loss): MAL=1.84 млн. дол.

95% VaR (максимальний збиток у 95% сценаріїв): $VaR_{0.95}$=3.21 млн.дол.

Expected Shortfall (очікувані збитки в найгірших 5% років): $ES_{0.95}$=4.57 млн. дол.

Після моделювання ефективності засобів кіберзахисту (MFA, WAF, аналітика поведінки, регулярні тренінги персоналу) встановлено, що комбіновані заходи дозволяють:

- зменшити частоту інцидентів на 27%;
- знизити середню вартість інциденту на 11%;
- скоротити річні очікувані збитки приблизно на 32%, тобто: MALпісля≈1.25 млн. дол.

Ці результати демонструють здатність методології чітко визначати економічну ефективність заходів безпеки, що робить її дієвим інструментом для бюджетування та планування кіберзахисту.

Висновки. Запропонована методологія кількісного моделювання кіберризиків забезпечує комплексний підхід до аналізу можливих втрат

організації, поєднуючи статистичні моделі, сценарне прогнозування та симуляційні методи. Вона дозволяє враховувати як часті маловитратні інциденти, так і рідкісні події з катастрофічними наслідками, формувати реалістичні криві ризику та визначати ефективність заходів безпеки на основі кількісних показників. Експериментальні дослідження підтверджують, що застосування методології дає змогу суттєво підвищити точність управління кіберризиками, покращити розподіл ресурсів та забезпечити обґрунтованість управлінських рішень.

Перелік використаних джерел.

1. Albina Orlando Cyber Risk Quantification: Investigating the Role of Cyber Value at Risk. *Risks*, 2021, Vol. 9, Issue 10, 184, pp. 1–12.

2. Bentley M., Stephenson A., Toscas P., Zhu Z. A Multivariate Model to Quantify and Mitigate Cybersecurity Risk. *Risks*, 2020, Vol. 8, Issue 2, Article 61, pp. 1–20.

3. Dzhamtyrova R., Maple C. Dynamic cyber risk estimation with Competitive Quantile Autoregression. *Data Mining and Knowledge Discovery*, 2022, Vol. 36, pp. 513–536.

УДК 004.056

Наталія ЯЦКІВ, Аліна МИКОЛАЙСЬКА

Західноукраїнський національний університет

МОДЕЛЬ ОЦІНКИ КІБЕРРИЗИКІВ У ХМАРНИХ СЕРВІСАХ

Вступ. Активне впровадження хмарних сервісів у бізнес–процеси організацій різних масштабів зумовлює різке зростання залежності активів від інфраструктури сторонніх провайдерів. Концентрація даних і обчислювальних ресурсів у хмарі, розподіленість компонентів, використання API та автоматизованих DevOps–процесів створюють специфічний профіль кіберризиків, що суттєво відрізняється від класичних IT–середовищ. Існуючі рамки управління ризиками (NIST SP 800–30/800–37, ISO/IEC 27005, ENISA Cloud Computing Risk Assessment, CSA Cloud Controls Matrix) задають загальні принципи і процеси, але потребують адаптації до контексту конкретних хмарних сервісів, моделей розгортання та розподілу відповідальності. Це обумовлює необхідність розробки спеціалізованої моделі оцінки ризиків, яка враховує особливості архітектури хмарних рішень, типів сервісів (IaaS/PaaS/SaaS), юридичних вимог, динаміки навантажень та можливостей безперервного моніторингу [1].

Мета. Розробити формалізовану модель оцінки ризиків у хмарних сервісах, що дозволяє системно і відтворювано визначати рівень кіберризику для різних сценаріїв використання хмари та обґрунтовувати вибір заходів захисту.

1. Методологічні засади побудови моделі

У якості методологічної основи моделі доцільно використати процесний підхід ISO/IEC 27005 до управління ризиками інформаційної безпеки (ідентифікація, аналіз, оцінка, обробка, моніторинг і перегляд ризиків) у поєднанні з NIST SP 800–30/800–37, що деталізують етапи оцінки ризику та вбудовують їх у життєвий цикл інформаційних систем. Для хмарного контексту додаються рекомендації ENISA щодо аналізу переваг і ризиків хмарних технологій та практики CSA Cloud Controls Matrix (CCM), яка надає структурований каталог контролів для IaaS, PaaS і SaaS [2].

Ключовою концепцією моделі є представлення ризику як функції від трьох груп параметрів:

– характеристик активів (цінність, критичність, залежність від хмари);

– параметрів загроз і вразливостей (ймовірність виникнення інцидентів, рівень експозиції, ефективність існуючих контролів);

– контексту хмарної архітектури (тип сервісу, модель розгортання, модель спільної відповідальності, географія та юрисдикція розміщення даних). Модель повинна забезпечити можливість як якісної (категорії «низький–середній–високий»), так і кількісної/напівкількісної (бальна шкала, інтегральний індекс) оцінки ризиків.

2. Структура моделі оцінки ризиків

Запропоновану модель доцільно будувати багаторівневою, виділяючи щонайменше три рівні.

Рівень бізнес–процесів – аналіз впливу відмови або компрометації хмарних сервісів на виконання ключових бізнес–функцій, фінансові показники, репутацію, відповідність нормативним вимогам.

Рівень хмарних сервісів – оцінка ризиків для конкретних сервісів (облікові записи, сховища, віртуальні машини, контейнерні платформи, бази даних, serverless–функції тощо).

Рівень технічної інфраструктури та контролів – урахування стану мережевих, криптографічних, ідентифікаційно–авторизаційних, моніторингових та інших засобів безпеки, у т.ч. специфічних для хмари (IAM–політики, security–групи, KMS, WAF, CASB).

Для кожного хмарного сервісу вводиться набір атрибутів: тип моделі (IaaS/PaaS/SaaS), модель розгортання (публічна, приватна, гібридна, multi–cloud), категорія даних (персональні, фінансові, комерційна таємниця), вимоги до конфіденційності, цілісності та доступності. На основі цих атрибутів формується профіль ризику сервісу, який визначає релевантний набір загроз (наприклад, витік даних через неправильно налаштоване сховище, атаки на API, компрометація облікових записів адміністратора, вразливості в ланцюгу постачання, збої провайдера).

3. Етапи процесу оцінки ризиків у моделі

1. Підготовка та визначення контексту. На цьому етапі визначаються межі оцінки (організаційний підрозділ, система, проєкт), перелік хмарних сервісів та провайдерів, регуляторні вимоги (GDPR, національне законодавство, галузеві стандарти), а також ролі й відповідальність сторін (клієнт, хмарний провайдер, інтегратор). Частина інформації може бути зібрана на основі опитувальника CSA CAIQ, що формалізує питання до провайдера [3].

2. Ідентифікація активів, загроз і вразливостей. Формується реєстр активів, пов’язаних із хмарою (дані, сервіси, облікові записи, ключі шифрування, журнали подій). Для кожної категорії активів визначаються можливі загрози (технічні, організаційні, юридичні) на основі ENISA–каталогів ризиків та типових сценаріїв атак на хмару. Вразливості класифікуються за джерелом: конфігураційні помилки, недоліки процесів, людський фактор, залежність від третьої сторони, слабкі або відсутні контролі.

3. Аналіз і розрахунок ризиків. Для кожного сценарію ризику оцінюється ймовірність реалізації P та величина впливу I на конфіденційність, цілісність та доступність, а також на юридичні та наслідки відповідності. На практиці модель може використовувати напівкількісний підхід:

$$R = P \cdot I \cdot W,$$

де W – ваговий коефіцієнт, що враховує критичність бізнес–процесу або особливі вимоги до даних (наприклад, персональні дані, дані платіжних карток). Значення P та I задаються на шкалі (наприклад, 1–5 або 1–10), а результат нормується до уніфікованої шкали ризику (низький, середній, високий, критичний). У більш складному варіанті можливе використання методів багатокритеріальної оптимізації або нечіткої логіки для точнішого урахування невизначеності [4].

4. Оцінка ефективності існуючих контролів. На цьому кроці ризики коригуються з урахуванням реалізованих заходів безпеки, зіставлених із доменами CSA CCM (ідентичність та доступ, шифрування, безпека віртуалізації, логування та моніторинг, управління змінами тощо). Для кожного контролю встановлюється рівень зрілості/ефективності, що дозволяє переходити від «вихідного» (інгерентного) ризику до «залишкового».

5. Пріоритизація та формування плану обробки ризиків. Ризики ранжуються за значенням інтегрального показника R, додатково враховується можливість їхнього агрегування на рівні бізнес–процесів і хмарних сервісів. Для пріоритетних ризиків визначаються сценарії обробки (зменшення, уникнення, передавання, прийняття) відповідно до ISO/IEC 27005 та ISO/IEC 27001, а також перелік рекомендованих технічних і організаційних заходів.

6. Безперервний моніторинг та перегляд оцінок. Оскільки хмарні середовища є високо динамічними, модель передбачає регулярне оновлення оцінок ризику на основі даних моніторингу (журнали доступу, події безпеки, результати сканування вразливостей, звіти провайдера). Інтеграція з SIEM/SOAR–системами та хмарними засобами моніторингу дозволяє автоматизувати частину розрахунків та оперативно реагувати на зміну профілю загроз.

4. Формалізація ризику та індекс хмарної безпеки

Для практичної реалізації моделі доцільно запровадити індекс хмарної безпеки для кожного сервісу або бізнес–процесу. Такий індекс може мати вигляд зваженої суми окремих компонентів ризику:

$$R_{cloud} = \sum_j w_j \cdot R_j,$$

де R_j – ризик за окремим сценарієм (наприклад, витік даних, відмова сервісу, порушення відповідності;

w_j – вагові коефіцієнти, що відображають пріоритети організації (наприклад, більша вага для ризиків, пов'язаних із персональними даними або безперервністю бізнесу).

Таке формулювання дозволяє агрегувати ризики з різних джерел та відображати їх у єдиному показнику, що зручно для представлення керівництву.

На основі значень R_{cloud} визначаються «діапазони дій»:

– зелений рівень – ризик прийнятний, достатньо підтримувати поточні контролі та здійснювати моніторинг;

– жовтий рівень – потрібна оптимізація конфігурацій і додаткові компенсуючі заходи;

– червоний рівень – необхідні невідкладні технічні та організаційні зміни, можлива зміна моделі використання хмари або провайдера.

5. Інтеграція моделі з нормативними рамками та практикою

Розроблювана модель повинна бути сумісною з існуючими стандартами та рамками, щоб її результати могли використовуватися в системах менеджменту інформаційної безпеки (ISMS) і звітності для аудиту. По–перше, вона має

підтримувати узгодження з ISO/IEC 27001/27002, ISO/IEC 27017 (контролі для хмарних сервісів) та ISO/IEC 27018 (захист персональних даних у хмарі). По–друге, показники та категорії ризику повинні легко відображатися на домени CSA CCM і опитувальники CAIQ, що спрощує оцінку постачальників та участь у програмах на кшталт CSA STAR [5].

Крім того, модель має враховувати підходи ENISA до оцінки ризиків у хмарі, де ризики ранжуються за ймовірністю й впливом та супроводжуються рекомендаціями щодо заходів безпеки для різних типів користувачів (малі/середні підприємства, великі організації, постачальники послуг). Це дозволяє забезпечити порівнюваність результатів оцінки з європейськими практиками та полегшує комунікацію з регуляторами й партнерами.

Висновки. Хмарні сервіси формують специфічний профіль кіберризиків, обумовлений мультитенантністю, розподіленою архітектурою, високим рівнем автоматизації та розподілом відповідальності між провайдером і клієнтом; тому використання лише загальних підходів до управління ризиками без спеціалізованої адаптації є недостатнім.

Розроблена модель оцінки ризиків у хмарних сервісах базується на міжнародно визнаних стандартах (ISO/IEC 27005, NIST SP 800–30/800–37) і рекомендаціях ENISA та CSA, але доповнюється багаторівневою структурою (бізнес–процеси – хмарні сервіси – технічні контролі) та механізмами формалізованого розрахунку інтегрального показника ризику.

Описаний процес оцінки ризиків (визначення контексту, ідентифікація активів, загроз і вразливостей, аналіз і розрахунок ризиків, урахування ефективності контролів, пріоритизація та безперервний моніторинг) забезпечує відтворюваність результатів, підтримує як якісний, так і напівкількісний аналіз і може бути інтегрований у існуючі ISMS–процеси.

Запровадження індексу хмарної безпеки та узгодження моделі з рамками ISO/IEC та ENISA дозволяє організаціям не лише оцінювати поточний рівень ризику, але й обґрунтовано приймати рішення щодо вибору/зміни хмарних провайдерів, оптимізації конфігурацій сервісів і планування заходів із підвищення кіберстійкості хмарної інфраструктури.

Перелік використаних джерел.

1. Force, Joint Task. "Risk management framework for information systems and organizations". NIST Special Publication 800 (2018): 37.

2. Cloud Computing Risk Assessment. [Електронний ресурс].- Режим доступу: https://www.enisa.europa.eu/publications/cloud–computing–risk–assessment

3. Cloud Controls Matrix and CAIQ v4. [Електронний ресурс].- Режим доступу: https://cloudsecurityalliance.org/artifacts/cloud–controls–matrix–v4

4. Information Security. NIST Special Publication 800–30. [Електронний ресурс].- Режим доступу: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800–30r1.pdf

5. Ali, T., Al–Khalidi, M., & Al–Zaidi, R. (2024). Information security risk assessment methods in cloud computing: Comprehensive review. Journal of Computer Information Systems, 1–28.