



DOI 10.28925/2663-4023.2026.35.1431

УДК 004.056:004.8

Бойко Вадим Віталійович

магістрант

Харківський національний економічний університет імені Семена Кузнеця, Харків, Україна

ORCID:0009-0002-6270-8541

vadyb.boiko.1@hneu.net

Шаповалова Олена Олександрівна

к.т.н., доцент, доцент каф. КІТ

Харківський національний економічний університет імені Семена Кузнеця, Харків, Україна

ORCID:0000-0003-4566-6634

olena.shapovalova@hneu.net

Долгова Наталя Геннадіївна

к.т.н., доцент,

доцент кафедри кібербезпеки та інформаційних технологій,

Харківський національний економічний університет імені Семена Кузнеця, Харків, Україна

ORCID ID: 0000-0002-8950-8200

natalya.dolgova@hneu.net

ПРОЄКТУВАННЯ АРХІТЕКТУРИ БЕЗПЕЧНОЇ АВТОНОМНОЇ МУЛЬТИАГЕНТНОЇ СИСТЕМИ ТЕСТУВАННЯ НА ПРОНИКНЕННЯ НА БАЗІ ВЕЛИКИХ МОВНИХ МОДЕЛЕЙ

Анотація. У статті проведено комплексний аналіз методів та підходів до проєктування архітектури безпечної мультіагентної системи автономного тестування на проникнення, що функціонує на базі великих мовних моделей. Актуальність дослідження зумовлена необхідністю подолання розриву між ресурсомістким та повільним аудитом інформаційної безпеки вручну та повністю автоматизованою, але небезпечною роботою класичних сканерів, неконтрольоване використання яких на виробництві створює критичні ризики відмови в обслуговуванні (DoS) або виходу тестування за дозволені межі. Під час аналізу останніх публікацій визначено, що існуючі рішення на базі штучного інтелекту (наприклад, PentAGI або Penligent) потерпають від специфічних загроз, таких як каскадні збої, підробка ідентичності (identity spoofing) та групові галюцинації мовних моделей. Теоретичним підґрунтям роботи виступає концепція інтеграції нейромережевого підходу та криптографічного захисту операцій, згідно з якою великі мовні моделі виконують виключно функції семантичного управління контекстом, стратегічного планування та маршрутизації завдань. Безпосереднє виконання команд та генерація доказів делегується реальним спеціалізованим інструментам, що виключає ризик штучної фабрикації артефактів. Методика дослідження передбачає проєктування бекенд-інфраструктури у форматі модульного моноліту з використанням мови TypeScript та фреймворку Nest.js на базі архітектурних патернів CQRS і Saga. Для управління процесом розгортання атаки задіяно фреймворк LangGraph, що формалізує аудит у вигляді графів станів із чітким розподілом завдань (розвідка, пошук вразливостей, верифікація) між незалежними агентами. Основним результатом дослідження є розроблений механізм суворої ізоляції інструментарію: усі деструктивні утиліти, включаючи Metasploit, nmap та OWASP ZAP, запускаються в одноразових Docker-контейнерах на базі Kali Linux. Комунікація між оркестратором та ізольованим середовищем відбувається виключно через стандартні потоки вводу/виводу за специфікацією Model Context Protocol (MCP) без відкриття зовнішніх мережеских портів. Для забезпечення контролю над процесом впроваджено систему динамічного переривання (Approval Gating), яка підтримує три рівні автономності (Manual, Semi-auto, Full-auto) та дозволяє фахівцю узгоджувати високоризиковані дії. З метою гарантування незмінності доказової бази реалізовано криптографічне підписання отриманих артефактів за алгоритмом Ed25519. У висновках доведено, що запропонована архітектура усуває проблему надмірної агентності штучного інтелекту, створюючи надійну проміжну ланку між цільовою системою та експертом. Подальші дослідження будуть



спрямовані на масштабування інфраструктури для забезпечення можливості паралельного тестування значної кількості хостів.

Ключові слова: тестування на проникнення; мультиагентна система; великі мовні моделі; інформаційна безпека; графі станів; ізольоване середовище; Model Context Protocol; криптографічний захист, штучний інтелект.

ВСТУП

Використання великих мовних моделей (Large Language Models, LLM) розширює можливості автоматизації тестування на проникнення, дозволяючи поєднувати пошук вразливостей із плануванням, аналізом результатів та вибором подальших дій. Традиційне тестування потребує значних часових ресурсів і залучення кваліфікованих фахівців, зокрема для виконання повторюваних операцій зі сканування мережесервісів, аналізу конфігурацій та документування результатів. Водночас використання автоматизованих інструментів без належного контролю може створювати ризики для цільової інфраструктури, зокрема внаслідок надмірно інтенсивного сканування, виконання небезпечних команд або виходу за встановлені межі тестування. Застосування мультиагентного підходу дає змогу розподілити завдання між спеціалізованими агентами та організувати їхню взаємодію в межах єдиного процесу перевірки захищеності інформаційних систем. Однак розширення автономності таких агентів потребує відповідних архітектурних рішень для керування їхніми діями, контролю допустимості операцій, ізоляції інструментального середовища та забезпечення цілісності отриманих результатів.

Питання інтеграції агентного штучного інтелекту в наступальну кібербезпеку останні роки активно досліджується [1]. Еволюція засобів автоматизації, яка починалась з лінійних скриптів, зараз сконцентрувала увагу на складних мультиагентних архітектурах, серед яких виділяють Single-Agent ReAct, Planner-Executor, Specialized Roles, Swarm, MCP-based та Native Skills [2]. Існуючі відкриті рішення, такі як багатоагентна система PentAGI, активно використовують семантичну векторну пам'ять для управління контекстом [3], проте страждають від високого рівня хибнопозитивних спрацювань (від 10% до 30%) через роботу за задалегідь запрограмованою логікою [4]. Інші платформи, наприклад, Penligent, зосереджені на глибокій верифікації вразливостей через імплементацію TTP (Techniques, Tactics, and Procedures) [5], але є закритими SaaS-продуктами, що створює ризики конфіденційності для корпоративних даних [6].

Крім того, сучасні дослідження підтверджують, що використання мультиагентних систем генерує нові специфічні вразливості, зокрема каскадні збої (cascading failures), підробку ідентичності (identity spoofing) та групові галюцинації (group hallucination) [7]. Незважаючи на широке висвітлення цих загроз, невирішеною частиною загальної проблеми залишається відсутність комплексного аналізу методів, які б дозволили спроектувати таку архітектуру, що гарантовано нівелює ризики, пов'язані з використанням ШІ, завдяки ізоляції інструментарію, механізмам переривання (approval gating) та криптографічному підтвердженню артефактів [8].

Метою статті є розроблення та обґрунтування архітектури безпечної автономної мультиагентної системи тестування на проникнення на основі великих мовних моделей, що передбачає розподіл функцій між спеціалізованими агентами, координацію їхньої взаємодії, керування рівнями автономності, контроль виконання потенційно небезпечних операцій, ізоляцію інструментального середовища та забезпечення цілісності отриманих результатів.

ТЕОРЕТИЧНІ ОСНОВИ ДОСЛІДЖЕННЯ

Сучасні автономні системи кібербезпеки все частіше спираються на концепцію з інтеграцією великих мовних моделей (LLM). Основною метою побудови таких систем є перехід від використання ШІ як основного генератора тексту, артефактів тощо до його застосування як когнітивного ядра в межах автономної агентної архітектури.

Великі мовні моделі – це нейромережеві системи, побудовані на архітектурі трансформерів, які навчаються на великих масивах текстових даних для розуміння природної мови, виявлення семантичних зв'язків та передбачення наступних токенів у послідовності [9]. Для вирішення прикладних завдань, таких як тестування на проникнення, використання виключно LLM є обмеженим, оскільки вони не мають доступу до реального середовища, схильні до галюцинацій та обмежені розміром контекстного вікна.

Для подолання цих обмежень у теорії ШІ застосовується концепція інтелектуальних агентів. Сучасні дослідження визначають агентний ШІ як поєднання мовної моделі з трьома критичними компонентами:

- надання моделі доступу до виклику зовнішніх API або утиліт;

- збереження контексту за допомогою короткострокової пам'яті у межах сесії та довгострокової семантичної або векторної пам'яті;
- здатність розбивати комплексне завдання на підзадачі.

На ранніх етапах розвитку агентних систем широко застосовувалась лінійна парадигма планування, зокрема підхід ReAct (Reason + Act), який об'єднує процеси міркування та виконання дій у єдиний цикл [10]. Проте лінійні ланцюжки не завжди є достатньо ефективними для складних багатокрокових завдань через втрату контексту та неможливість гнучкого повернення на попередні етапи у разі помилки.

Для розв'язання цієї проблеми у сучасному оркеструванні агентами використовують графи станів (State Graphs). У сучасних наукових роботах, таких як дослідження архітектури Graph of Thoughts (GoT), доведено, що моделювання процесів LLM у вигляді орієнтованих графів значно підвищує здатність системи вирішувати комплексні завдання порівняно з лінійними методами [11].

Граф станів є абстрактною математичною моделлю, що складається з множини вершин (вузлів або станів) та спрямованих ребер (переходів між станами). У мультиагентних системах кожен вузол графа репрезентує ізольованого спеціалізованого агента, а ребра визначають логічні умови передачі управління від одного агента до іншого.

Використання графів станів вирішує проблему нездатності ШІ динамічно оцінювати складність завдань. На відміну від лінійного виконання, графова архітектура забезпечує [11]:

- можливість системи багаторазово повертатися до попереднього вузла, якщо поточних даних недостатньо для експлуатації вразливості;
- кожен вузол отримує лише ті дані, які необхідні для його функції, що запобігає перевантаженню контекстного вікна та знижує ризик галюцинацій;
- граф може бути зупинений на будь-якому ребрі для очікування зовнішнього тригера, наприклад, дозволу перед переходом до стану підвищеного ризику.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Методологічною основою даного дослідження є поєднання методів системного аналізу, архітектурного моделювання та експериментального проектування ПЗ. Зважаючи на високий рівень непередбачуваності, притаманний використанню великих мовних моделей у завданнях наступальної кібербезпеки, класичні емпіричні підходи до автоматизації є недостатніми. Тому основним методом розв'язання поставленої наукової проблеми обрано побудову архітектури, яка апаратно та програмно обмежує деструктивні можливості ШІ.

Для реалізації запропонованої моделі застосовано комплексний підхід до проектування інфраструктури за шаблоном «модульний моноліт» (modular monolith) [14]. Вибір цієї методології розробки зумовлений необхідністю забезпечення інкапсуляції логічних доменів (оркестрація, управління агентами, обробка подій) у єдиній кодовій базі. Це дозволяє уникнути значних накладних витрат на мережеву взаємодію між мікросервісами, що є критично важливим для підтримання високої швидкості обробки поточних даних під час сканування цільових мереж. Крім того, методика дослідження передбачає імплементацію принципів Zero-Trust (нульової довіри) на рівні взаємодії між когнітивним ядром ШІ та середовищем виконання утиліт.

На рисунку 1 зображено загальну структуру та потоки даних розробленої архітектури.

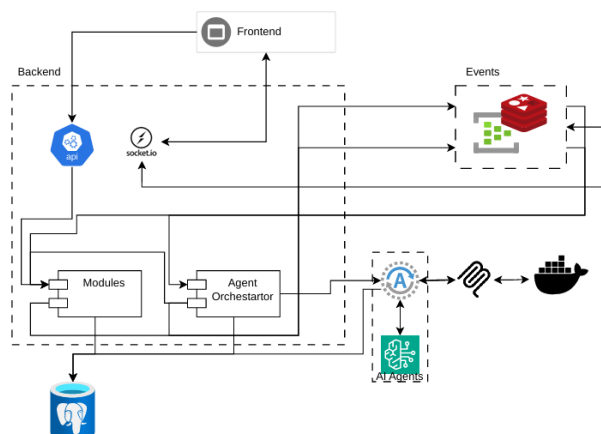




Рис. 1. Структурна схема мультиагентної системи тестування на проникнення

Архітектура складається з п'яти основних взаємодіючих рівнів та блоку зовнішніх сервісів, що забезпечує високу модульність, безпеку виконання та масштабованість. Основними елементами розробленої архітектури є:

- клієнтська частина (Frontend);
- серверна частина (Backend);
- брокер подій (Event Broker);
- підсистема ШІ (AI Agents);
- сховища даних (Datastores).

Клієнтська частина забезпечує користувацький інтерфейс, який надсилає запити до серверної частини через API та отримує зворотній зв'язок у режимі реального часу за допомогою технології Socket.io.

Серверна частина представлена модульним монолітом на фреймворку Nest.js, що виступає центральним вузлом, який координує всі процеси та контролює правила безпеки. До нього входять наступні елементи:

- REST контролери для отримання запитів, авторизації та маршрутизації;
- Socket.io для відправки та обробки подій від брокера та користувача;
- модулі, які обробляють бізнес логіку системи та керують процесом проведення пентесту, інтеграціями та авторизацією;
- agent-orchestration містить сервіси, які відповідають за управління графами агентів для кожного хосту.

Брокер подій використовує брокер повідомлень Redis, що забезпечує асинхронний зв'язок, маршрутизацію подій та управління чергами між модулями.

AI Agents є блоком, який приймає аналітичні рішення та формує команди для проведення тестування на проникнення [12]. Для гарантування безпеки ці команди передаються в ізольований Docker-контейнер. Підсистема ШІ складається з:

- графу послідовності виконання фаз;
- інтеграції з провайдером LLM;
- Throwaway Kali containers з реальними інструментами.

Сховища даних використовують реляційну базу даних PostgreSQL для збереження конфігурацій, історії дій, станів графів агентів та історію про користувачів і тестування.

Цикл взаємодії елементів проходить через декілька фаз і відбувається наступним чином:

- під час ініціалізації оператор через веб-інтерфейс надсилає HTTPS REST-запит до бекенду на створення нового тестування (із зазначенням цілі та рівня автономності);
- доменний модуль зберігає запис у PostgreSQL і публікує подію через брокер подій Redis Streams;
- модуль agent-orchestration зчитує цю подію і запускає виконання графа індивідуально для кожного дозволеного хоста.

Під час кожної фази вузол графа розгортає контейнер Kali Linux, де взаємодія відбувається через сокет Docker Engine та протокол MCP. За необхідності LLM генерує параметри для утиліт, після чого реальний інструмент виконує дію.

Якщо дія є ризикованою, наприклад, запуск експлойта, граф зупиняється. Стан графа записується у PostgreSQL, а через Redis Streams надсилається подія. RealtimeGateway в реальному часі доставляє цей запит до браузера оператора через WebSocket.

Для збереження доказів усі виведення реальних інструментів та знайдені артефакти повертаються до доменного модуля, де підписуються і зберігаються в БД.

Після успішного завершення обходу всіх вузлів графа система звертається до БД, агрегує підписані знахідки та генерує фінальний PDF-звіт.

Основна ідея дослідження полягає в реалізації концепції інтеграції нейромережевого підходу та криптографічного захисту операцій, що дозволяє мінімізувати обмеження, наявні в генеративних моделях. При спробах самостійно генерувати докази вразливостей або технічні артефакти великі мовні моделі (LLM) у кібербезпеці виявляються схильними до галюцинацій [12]. Архітектура, що пропонується, спирається на принцип чіткого розділення обов'язків таким чином, що нейромережевий компонент (LLM) використовується виключно для семантичного міркування, планування та маршрутизації завдань, а символічний компонент (ізолювані сканери та утиліти) забезпечує детерміноване виконання команд та отримання об'єктивних фактів. Таким чином, жодні докази чи артефакти не генеруються моделлю штучно та усі висновки ґрунтуються на реальних результатах



взаємодії
цільовим середовищем.

3

Основною моделювання дій атакуючого виступає глобальна база знань MITRE ATT&CK, яка забезпечує стандартизовану таксономію тактик та технік (TTP) [13].

Управління контекстом у мультиагентній системі ґрунтується на математичному апараті графів станів (State Graphs). На відміну від лінійних парадигм, до яких відноситься ReAct, що швидко втрачають контекст через обмежені вікна токенів, графова архітектура дозволяє утримувати глобальне бачення цільової інфраструктури [10].

Для мінімізації ризиків експлуатації самих ШІ-агентів система навчена на класифікації загроз OWASP Top 10 for LLM [14]. Особлива увага приділяється запобіганню вразливостей типу Excessive Agency (надмірна агентність), коли ШІ може отримати доступ до деструктивних функцій [6]. Це вирішується через впровадження принципу повної медіації та архітектури Zero-Trust на рівні міжсервісної комунікації, де кожна дія агента проходить багаторівневу валідацію на відповідність дозволеним межам тестування. Реалізація запропонованого підходу виконана з застосуванням комплексного підходу до проєктування програмної архітектури типу «модульний моноліт» [14].

Серверну частину розроблено мовою TypeScript у строгому режимі (strict mode) з використанням фреймворку Nest.js (версія 10). Такий вибір забезпечує високу надійність типізації даних між клієнтом та сервером, а також підтримує інверсію управління (Dependency Injection) для гнучкого масштабування. З огляду на необхідність обробки тривалих процесів сканування без блокування основного потоку виконання, архітектуру побудовано на основі патернів CQRS (Command Query Responsibility Segregation) і Saga.

Для роботи з даними було розроблено структуру, яка містить наступні модулі:

- база даних;
- ORM;
- шина подій та кешування.

Для реалізації бази даних обрана PostgreSQL (версія 16), що використовується як основне сховище для конфігурацій, результатів та довготривалих станів графів агентів через механізм PostgresSaver.

Для організації безпечної взаємодії з базою даних як ORM використовується Prisma ORM, що виключає ризики SQL-ін'єкцій на програмному рівні.

Шина подій та кешування реалізовано за допомогою Redis (версія 7) з підтримкою Redis Streams, що забезпечує асинхронний зв'язок між мікросервісними модулями, а система BullMQ забезпечує управління чергами тривалих завдань.

Для побудови багатоагентної логіки було застосовано фреймворки LangChain та LangGraph [15]. LangGraph дозволяє створювати індивідуальні графи виконання для кожного цільового хоста з нативною підтримкою переривань, що є критично важливим для реалізації функціоналу очікування дозволу оператора (Approval Gating).

Методика проведення безпосереднього тестування на проникнення передбачає повну ізоляцію шкідливого коду. Для цього було імплементовано наступний механізм:

- система через Docker Engine API (dockercode) розгортає одноразові (throwaway) контейнери на базі Kali Linux [16];
- запускаються професійні утиліти (Metasploit, OWASP ZAP) всередині розгорнутих контейнерів;
- комунікація між бекендом та ізольованим середовищем здійснюється виключно через стандартні потоки вводу/виводу за специфікацією Model Context Protocol (MCP), без відкриття мережових портів, що унеможливорює зворотнє інфікування платформи.

Задля забезпечення достовірності отриманих результатів, передбачається криптографічне підписання кожного видобутого артефакту (файлу, результату сканування тощо) унікальним ключем Ed25519, який належить конкретному агенту-виконавцю. Це дозволяє сформувати надійну послідовність доказів (evidence chain), стійких до фальсифікацій.

У ході застосування запропонованого підходу спроектовано та формалізовано архітектуру інтелектуальної мультиагентної системи, яка імітує діяльність професійної групи фахівців з кібербезпеки шляхом розподілу ролей між спеціалізованими агентами. Розроблена архітектура завдяки впровадженню графів станів вирішує проблему нездатності динамічно оцінювати складність завдань, яка притаманна існуючим рішенням, таким як PentAGI [4].



Цільова інфраструктура проходить всі етапи життєвого циклу тестування з залученням інформації з бібліотеки LangGraph, яка розгортає окремих граф виконання для кожного виявленого хоста. Процес розділено на логічні фази, за кожну з яких відповідає незалежний вузол (агент), що спеціалізується на конкретному класі завдань. Розподіл обов'язків між агентами наведено у таблиці 1.

Таблиця 1

Розподіл обов'язків між вузлами багатоагентного графа

Назва вузла (агента)	Характеристика та інструментарій
ReconnaissanceNode	Відповідає за етап збору інформації без активної експлуатації. Розгортає ізольований контейнер, запускає мережеві утиліти (nmap, dig) та формує масив перевірених фактів про ціль.
VulnSearchNode	Аналітичний вузол, що використовує зібрані дані для запитів до реальних баз вразливостей (NVD, Shodan) з метою зіставлення сервісів із відомими CVE.
VulnCheckNode	Вузол перевірки, що підтверджує можливість експлуатації вразливостей через протокол MCP без виконання деструктивного коду.
PrivEscNode	Спеціалізований агент для підвищення привілеїв. Викликає модулі локального підвищення привілеїв (msf_local_privesc), фіксує результати і знищує контейнер після завершення.
WebAppScanNode	Запускає OWASP ZAP Automation Framework для сканування виявлених HTTP/HTTPS сервісів. Штучно обмежує інтенсивність атак (Medium strength) для запобігання DoS.
ConsiderPivots	Вузол без власного контейнера, який аналізує доступність нових цілей у мережі (CIDR) та оцінює доцільність розширення вектора атаки.
ReporterNode	Фінальний вузол, який консолідує артефакти та генерує структурований PDF-звіт після завершення тестування всіх хостів.

Критичним результатом проєктування є впровадження механізму Sandboxed Tooling. Для запобігання зворотньому інфікуванню самої системи агент-ШІ позбавлений можливості напряму виконувати команди в операційній системі хоста. Замість цього розроблено модуль оркестрації контейнерів, в межах якого виконується наступний код, який дозволяє створювати ізольовані контейнери та виконувати в них команди:

```
const container = await docker.createContainer({
  Image: 'kali-linux-pentest-base',
  Cmd: ['/bin/sh'],
  AttachStdin: true,
  AttachStdout: true,
  AttachStderr: true,
  NetworkMode: 'none'
});
await container.start();
```

Даний підхід демонструє, що взаємодія між керуючим бекендом (Nest.js) та середовищем виконання (Kali Linux) здійснюється виключно через стандартні потоки вводу/виводу за специфікацією MCP. Це унеможливує identity spoofing та каскадні мережеві атаки, які є характерними вразливостями для розподілених ШІ-систем [10].

Для розв'язання проблеми безпечного використання ШІ в промислових середовищах, в архітектуру закладено механізм динамічного переривання графа (interrupt). Дії системи класифікуються за рівнем ризику, а оператору надається можливість обрати один із трьох режимів автономності:

1. manual;
2. semi-auto;
3. full-auto.

У manual режимі кожна дія, рутинне сканування зокрема, вимагає авторизації або надання детальної інформації від людини-оператора.

Рутинні операції в semi-auto режимі виконуються автоматично, проте система обов'язково призупиняється та вимагає схвалення перед виконанням ризикованих дій (експлуатація вразливостей, горизонтальне переміщення між хостами).



У full-auto режимі система функціонує без втручання людини, керуючись наперед заданими лімітами безпеки. У цьому режимі повністю заблоковані деструктивні дії, що можуть призвести до компрометації або знищення інфраструктури клієнта.

Спілкування платформи з користувачем забезпечується в режимі реального часу за допомогою технології Socket.IO, що дозволяє миттєво доставляти запити на підтвердження дій (Approval Gating).

Під час збору доказової бази має бути задіяно спеціалізованого агента, що гарантує цілісність артефактів. Кожна успішна команда чи видобутий файл криптографічно підписується унікальним ключем Ed25519, створюючи надійний ланцюжок доказів. Це вирішує проблему декларативності результатів, яка є вразливим місцем у наявних аналогах. Крім того, перед формуванням фінального PDF-звіту через бібліотеку Puppeteer, система пропускає дані через механізм маскування (redaction pass), автоматично приховуючи чутливі дані клієнта (паролі, ключі API, приватні ключі).

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Проведений аналіз підходів до побудови автономних систем тестування на проникнення на основі великих мовних моделей дозволив визначити основні вимоги до організації мультиагентної взаємодії та безпечного виконання інструментальних операцій. За результатами дослідження розроблено архітектуру мультиагентної системи, у якій великі мовні моделі використовуються для аналізу отриманої інформації та планування подальших дій. Безпосереднє виконання команд покладається на спеціалізовані інструменти тестування, що працюють в ізольованому контейнерному середовищі.

Для координації роботи спеціалізованих агентів у запропонованій архітектурі передбачено використання графів станів, які визначають послідовність виконання завдань і дозволяють зберігати контекст тестування. Керування діями системи здійснюється з урахуванням обраного рівня автономності (Manual, Semi-auto, Full-auto), а для операцій, що потребують дозволу фахівця, передбачено механізм Approval Gating. Взаємодію з інструментами тестування запропоновано організувати через протокол Model Context Protocol (MCP) із використанням одноразових Docker-контейнерів. Окремим архітектурним рішенням є застосування алгоритму Ed25519 для криптографічного підписання доказових артефактів, що дозволяє передбачити можливість перевірки їхньої цілісності та походження.

Основним результатом роботи є архітектурне рішення, у якому функції інтелектуального планування відокремлено від безпосереднього виконання інструментальних операцій. Такий підхід дозволяє передбачити контроль дій автономних агентів без відмови від автоматизації процесу тестування на проникнення. Поєднання механізмів керування автономністю з ізоляцією середовища виконання спрямоване на зменшення ризиків неконтрольованого використання інструментів тестування. Водночас практична ефективність запропонованих рішень потребує експериментального підтвердження.

Подальші дослідження передбачають завершення програмної реалізації системи та перевірку її функціонування в контрольованому середовищі. Особливу увагу доцільно приділити оцінюванню коректності взаємодії агентів і роботи механізмів контролю автономності. Наступним етапом може стати дослідження можливостей масштабування архітектури для одночасного виконання кількох тестувань, а також оптимізація паралельної роботи агентів з метою зменшення взаємного блокування під час виконання завдань.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cankurt, S. (2026, 9 квітня). AI pentesting agents 2026: The rise of 39+ tools tested. AppSec Santa. <https://appsecsanta.com/research/ai-pentesting-agents-2026>
2. Veeam Software. (2026, 24 лютого). Veeam introduces agent commander to confront agentic AI risk at enterprise scale. <https://www.veeam.com/company/press-release/veeam-introduces-agent-commander-to-confront-agentic-ai-risk-at-enterprise-scale.html>
3. GitHub - vxcontrol/pentagi: Fully autonomous AI Agents system capable of performing complex penetration testing tasks. GitHub. <https://github.com/vxcontrol/pentagi>
4. Feedback from internal security team assessment — Is this a fair evaluation of pentagi's current capabilities vxcontrol/pentagi Discussion #217. GitHub. <https://github.com/vxcontrol/pentagi/discussions/217>
5. Penligent AI pentesting tool | agentic penetration testing for security teams. Penligent. <https://www.penligent.ai/>
6. Shannon AI pentesting tool vs penligent, what security engineers should actually compare in 2026. Penligent Security Blog – AI-Driven Hacking Tutorials, Exploit PoCs & Cybersecurity Research.



- <https://www.penligent.ai/hackinglabs/shannon-ai-pentesting-tool-vs-penligent-what-security-engineers-should-actually-compare-in-2026/>
7. Hammar, K. (2026). Multiagent LLM systems for security operations. *IEEE Security & Privacy*, 2–10. <https://doi.org/10.1109/msec.2026.3713252>
 8. TrinityGuard: A unified framework for safeguarding multi-agent systems. *arXiv.org e-Print archive*. <https://arxiv.org/html/2603.15408v1>
 9. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6). <https://doi.org/10.1007/s11704-024-40231-1>
 10. Wang, X., Bai, H., Huang, S., Wu, G., Yang, D., Wang, S., Liao, J., Zhang, Y., Liu, M., & Liu, L. (2026). A survey on synergizing large language models and knowledge graphs for reasoning. *Engineering Applications of Artificial Intelligence*, 182, 116132. <https://doi.org/10.1016/j.engappai.2026.116-132>
 11. Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., & Hoefler, T. (2024). Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16), 17682–17690. <https://doi.org/10.1609/aaai.v38i16.29720>
 12. Negnevitsky, M. (2025). The rise of autonomous AI agents: Automating complex tasks. *International Journal of Artificial Intelligence for Science (IJAI4S)*, 1(2). <https://doi.org/10.63619/ijai4s.v1i2.007>
 13. Jo, Y., Choi, O., You, J., Cha, Y., & Lee, D. H. (2022). Cyberattack models for ship equipment based on the MITRE att&ck framework. *Sensors*, 22(5), 1860. <https://doi.org/10.3390/s22051860>
 14. LLM06:2025 excessive agency. OWASP Gen AI Security Project. <https://genai.owasp.org/llmrisk/llm062025-excessive-agency/>
 15. LangChain overview - Docs by LangChain. Docs by LangChain. <https://docs.langchain.com/oss/python/langchain/overview>
 16. Kali linux | penetration testing and ethical hacking linux distribution. Kali Linux. <https://www.kali.org/>

**Vadim Boyko**

Master's student

Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine

ORCID:0009-0002-6270-8541

*vadym.boiko.1@hneu.net***Olena Shapovalova**

Ph.D. (Tech.), Associate Professor,

Associate Professor of the Department of Cybersecurity and Information Technology,

Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine

ORCID:0000-0003-4566-6634

*olena.shapovalova@hneu.net***Natalya Dolgova**

Ph.D. (Tech.), Associate Professor,

Associate Professor of the Department of Cybersecurity and Information Technology,

Simon Kuznets Kharkiv National University of Economics, Kharkiv, Ukraine

ORCID ID: 0000-0002-8950-8200

*natalya.dolgova@hneu.net***ANALYSIS OF METHODS FOR DESIGNING THE ARCHITECTURE OF A SECURE MULTI-AGENT SYSTEM FOR AUTONOMOUS PENETRATION TESTING BASED ON LARGE LANGUAGE MODELS**

Abstract. The article provides a comprehensive analysis of methods and approaches for designing the architecture of a secure multi-agent system for autonomous penetration testing based on large language models. The relevance of the study is driven by the need to bridge the gap between resource-intensive, slow manual information security audits and the dangerous full automation of classic scanners, the uncontrolled use of which in production environments poses critical risks of Denial of Service (DoS) or exceeding the authorized scope. An analysis of recent publications revealed that existing AI-based solutions (e.g., PentAGI or Penligent) suffer from specific threats such as cascading failures, identity spoofing, and group hallucinations of language models. The theoretical foundation of the work is the concept of integrating a neural network approach and cryptographic protection of operations, according to which large language models exclusively perform the functions of semantic context management, strategic planning, and task routing. The direct execution of commands and the generation of evidence are delegated to real specialized tools, eliminating the risk of artificial artifact fabrication. The research methodology involves designing the backend infrastructure as a modular monolith using TypeScript and the Nest.js framework based on CQRS and Saga architectural patterns. To manage the attack deployment process, the LangGraph framework is utilized, formalizing the audit as state graphs with a clear distribution of tasks (reconnaissance, vulnerability search, verification) among independent agents. The main result of the research is the developed mechanism for strict tool isolation: all destructive utilities, including Metasploit, nmap, and OWASP ZAP, are launched in disposable Kali Linux-based Docker containers. Communication between the orchestrator and the isolated environment occurs solely through standard input/output streams according to the Model Context Protocol (MCP) specification, without exposing external network ports. To ensure control over the process, a dynamic interruption system (Approval Gating) has been implemented, supporting three levels of autonomy (Manual, Semi-auto, Full-auto) and allowing the specialist to approve high-risk actions. To guarantee the immutability of the evidence base, cryptographic signing of the obtained artifacts using the Ed25519 algorithm is implemented. The conclusions prove that the proposed architecture eliminates the problem of excessive artificial intelligence agency, creating a reliable intermediary layer between the target system and the expert. Future research will focus on scaling the infrastructure to enable parallel testing of a significant number of hosts.

Keywords: penetration testing; multi-agent system; large language models; information security; state graphs; isolated environment; Model Context Protocol; cryptographic protection.



REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Cankurt, S. (2026, 9 April). AI pentesting agents 2026: The rise of 39+ tools tested. AppSec Santa. <https://appsecsanta.com/research/ai-pentesting-agents-2026>
2. Veeam Software. (2026, 24 February). Veeam introduces agent commander to confront agentic AI risk at enterprise scale. <https://www.veeam.com/company/press-release/veeam-introduces-agent-commander-to-confront-agentic-ai-risk-at-enterprise-scale.html>
3. GitHub - vxcontrol/pentagi: Fully autonomous AI Agents system capable of performing complex penetration testing tasks. GitHub. <https://github.com/vxcontrol/pentagi>
4. Feedback from internal security team assessment — Is this a fair evaluation of pentagi's current capabilities vxcontrol/pentagi Discussion #217. GitHub. <https://github.com/vxcontrol/pentagi/discussions/217>
5. Penlagent AI pentesting tool | agentic penetration testing for security teams. Penlagent. <https://www.penlagent.ai/>
6. Shannon AI pentesting tool vs penlagent, what security engineers should actually compare in 2026. Penlagent Security Blog – AI-Driven Hacking Tutorials, Exploit PoCs & Cybersecurity Research. <https://www.penlagent.ai/hackinglabs/shannon-ai-pentesting-tool-vs-penlagent-what-security-engineers-should-actually-compare-in-2026/>
7. Hammar, K. (2026). Multiagent LLM systems for security operations. IEEE Security & Privacy, 2–10. <https://doi.org/10.1109/msec.2026.3713252>
8. TrinityGuard: A unified framework for safeguarding multi-agent systems. arXiv.org e-Print archive. <https://arxiv.org/html/2603.15408v1>
9. Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., & Wen, J. (2024). A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6). <https://doi.org/10.1007/s11704-024-40231-1>
10. Wang, X., Bai, H., Huang, S., Wu, G., Yang, D., Wang, S., Liao, J., Zhang, Y., Liu, M., & Liu, L. (2026). A survey on synergizing large language models and knowledge graphs for reasoning. Engineering Applications of Artificial Intelligence, 182, 116132. <https://doi.org/10.1016/j.engappai.2026.116-132>
11. Besta, M., Blach, N., Kubicek, A., Gerstenberger, R., Podstawski, M., Gianinazzi, L., Gajda, J., Lehmann, T., Niewiadomski, H., Nyczyk, P., & Hoefler, T. (2024). Graph of thoughts: Solving elaborate problems with large language models. Proceedings of the AAAI Conference on Artificial Intelligence, 38(16), 17682–17690. <https://doi.org/10.1609/aaai.v38i16.29720>
12. Negnevitsky, M. (2025). The rise of autonomous AI agents: Automating complex tasks. International Journal of Artificial Intelligence for Science (IJAI4S), 1(2). <https://doi.org/10.63619/ijai4s.v1i2.007>
13. Jo, Y., Choi, O., You, J., Cha, Y., & Lee, D. H. (2022). Cyberattack models for ship equipment based on the MITRE att&ck framework. Sensors, 22(5), 1860. <https://doi.org/10.3390/s22051860>
14. LLM06:2025 excessive agency. OWASP Gen AI Security Project. <https://genai.owasp.org/llmrisk/llm062025-excessive-agency/>
15. LangChain overview - Docs by LangChain. Docs by LangChain. <https://docs.langchain.com/oss/python/langchain/overview>
16. Kali linux | penetration testing and ethical hacking linux distribution. Kali Linux. <https://www.kali.org/>

Отримано редакцією журналу / Received: 12.09.26

Прорецензовано / Revised: 24.09.26

Схвалено до друку / Accepted: 26.10.26



This work is licensed under Creative Commons Attribution-noncommercial-sharealike 4.0 International License.