

3.4 Запуск агента	44
3.5 Перевірка працездатності агента III	45
ВИСНОВКИ	52
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	53
Додаток А	55

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		5

ВСТУП

Актуальність теми. Штучний інтелект (ШІ) революціонує різні сектори, і кібербезпека не є винятком. Інтеграція ШІ в стратегії кібербезпеки, такі як кібербезпека зі штучним інтелектом та штучний інтелект і кібербезпека, стає дедалі важливішою, оскільки кіберзагрози стають складнішими та частішими. Агенти ШІ призначені для розширення можливостей систем кібербезпеки, забезпечуючи проактивний підхід до виявлення загроз та реагування на них.

Штучний інтелект може швидко аналізувати величезні обсяги даних, виявляючи закономірності та аномалії, які можуть свідчити про порушення безпеки. Алгоритми машинного навчання дозволяють системам навчатися на минулих інцидентах, покращуючи їхню здатність прогнозувати та запобігати майбутнім атакам. Крім того, інструменти на базі ШІ можуть автоматизувати рутинні завдання безпеки, дозволяючи аналітикам-людям зосередитися на складніших питаннях, що особливо актуально в галузі ШІ в кібербезпеці.

Зростаюча залежність від цифрової інфраструктури зробила організації більш вразливими до кібератак. Згідно зі звітом, кіберзлочинність, за прогнозами, коштуватиме світові 10,5 трильйонів доларів щорічно до 2025 року. Ця тривожна статистика підкреслює необхідність удосконалених заходів кібербезпеки, де штучний інтелект відіграє вирішальну роль.

Найцінніше те, що штучний інтелект не просто знаходить загрози, він допомагає відбиватися в режимі реального часу. Це схоже на надшвидкого охоронця, який може адаптуватися до будь-якої небезпечної ситуації.

Включення штучного інтелекту (AI) у стратегії кібербезпеки виходить за рамки традиційних методів захисту. AI додає проактивний і адаптивний аспект до кібербезпеки. Це робиться за допомогою алгоритмів машинного навчання для аналізу величезних наборів даних і виявлення потенційних загроз. Це включає розширене виявлення загроз, прогнозу аналітику та автоматичні

					БР.КІ-35.00.00.000 ПЗ	Арк
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

відповіді. Важливо, що штучний інтелект підвищує людські здібності, обробляючи та розуміючи дані швидко та у великому масштабі, підвищуючи загальну ефективність операцій у сфері кібербезпеки.

Захист API є критично важливим у сучасному цифровому світі, оскільки API слугують основою для взаємодії між різними програмами, сервісами та пристроями. Недостатній захист API може призвести до витоку конфіденційних даних, таких як персональна інформація користувачів, фінансові дані чи комерційна таємниця. Кібератаки, зокрема DDoS, SQL-ін'єкції чи атаки типу "man-in-the-middle", часто спрямовані на вразливості API, що робить їх захист пріоритетом для компаній. Відсутність належної автентифікації, наприклад OAuth, або шифрування даних може спричинити несанкціонований доступ до систем. Крім того, зростання кількості підключених пристроїв в Інтернеті речей (IoT) збільшує кількість API, які потребують захисту.

Створення агентів штучного інтелекту (ШІ) для захисту API набуває особливої актуальності, оскільки традиційні методи безпеки не завжди здатні оперативно реагувати на нові загрози. Агенти ШІ можуть аналізувати величезні обсяги даних у реальному часі, виявляючи аномалії, які вказують на потенційні атаки. Наприклад, ШІ може ідентифікувати незвичайні шаблони запитів до API, що можуть свідчити про спроби злому. Такі агенти здатні адаптуватися до нових видів атак завдяки машинному навчанню, що дозволяє їм постійно вдосконалювати свої алгоритми. Вони також можуть автоматизувати процеси реагування, наприклад, блокування підозрілих IP-адрес або тимчасове обмеження звернення до API.

Використання ШІ-агентів дозволяє не лише підвищити ефективність захисту, але й знизити навантаження на команди безпеки, зменшуючи кількість помилок, спричинених людським фактором. Актуальність таких рішень зростає в умовах швидкого розвитку технологій і ускладнення кіберзагроз. Впровадження ШІ у захист API є не лише реакцією на сучасні виклики, але й проактивним кроком для забезпечення безпеки в майбутньому.

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Мета роботи – Розробка ШІ-агента для виявлення вразливостей API.

Об’єкт дослідження – система захисту API.

Предмет дослідження – методи та підходи до виявлення та виправлення вразливостей API.

Методи дослідження включали аналіз літератури, експериментальне моделювання, програмування, тестування та порівняльний аналіз. Зокрема, використано методи статичного аналізу коду (SAST) із застосуванням Semgrep, аналіз поведінки для виявлення аномалій, автоматизоване сканування вразливостей, а також методи машинного навчання для адаптивного прогнозування загроз. Для розробки агента використано фреймворк CrewAI, що дозволило реалізувати інструменти для взаємодії з Docker, файловою системою та аналізу коду..

Практичне значення роботи полягає у створенні ШІ-агента, який здатен автономно виявляти та виправляти вразливості API у навчальному середовищі Damn Vulnerable RESTaurant API. Розроблений агент може бути використаний для:

- навчання розробників, етичних хакерів та інженерів з безпеки практичним навичкам захисту API;
- тестування та оцінки ефективності інструментів автоматизації безпеки, таких як SAST і DAST;
- інтеграції в процеси DevSecOps для раннього виявлення вразливостей у CI/CD-конвеєрах;
- підвищення обізнаності про безпеку API серед фахівців через гейміфікований підхід.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		8

1 СУЧАСНІ МЕТОДИ ВИЯВЛЕННЯ ВРАЗЛИВОСТЕЙ API ТА ПІДХОДИ СТВОРЕННЯ АГЕНТІВ ШІ

1.1 Сучасні методи виявлення вразливостей API з використанням ШІ

Сучасні методи виявлення вразливостей API з використанням штучного інтелекту (ШІ) базуються на здатності ШІ аналізувати великі обсяги даних, виявляти аномалії та адаптуватися до нових загроз.

Аналіз основних підходів, їх переваг, обмежень та прикладів застосування, з урахуванням актуальних тенденцій станом на квітень 2025 року.

1.1.1 Аналіз поведінки та виявлення аномалій. ШІ, зокрема алгоритми машинного навчання (ML), використовується для моніторингу запитів до API у реальному часі, щоб виявляти відхилення від нормальної поведінки.

Моделі ШІ, такі як нейронні мережі або алгоритми кластеризації, навчаються на історичних даних про запити до API, створюючи базову модель "нормальної" поведінки. Будь-які відхилення (наприклад, незвичайна частота запитів, нестандартні параметри або підозрілі IP-адреси) позначаються як потенційні вразливості чи атаки. Наприклад, для виявлення атак типу "credential stuffing", коли зловмисники використовують вкрадені облікові дані для масових запитів або для ідентифікації спроб SQL-ін'єкцій через аналіз нетипових шаблонів у вхідних даних. При цьому використовуються платформи, такі як Wallarm або Salt Security, які інтегрують ШІ для аналізу трафіку API та виявлення аномалій.

Перевагами є висока точність у виявленні невідомих загроз (zero-day attacks), здатність працювати в реальному часі.

Обмеженням застосування є необхідність якісних даних для навчання моделей і можливість помилкових спрацьовувань (false positives), якщо нормальна поведінка погано визначена.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		9

1.1.2. Автоматизоване сканування вразливостей. ШІ-агенти використовуються для активного сканування API на наявність відомих вразливостей, таких як слабка автентифікація, недостатнє шифрування або неправильна обробка даних.

Алгоритми ШІ аналізують специфікації API (наприклад, OpenAPI/Swagger), код або відповіді сервера, щоб виявити слабкі місця, такі як незахищені ендпоінти чи надмірне розкриття даних (broken object level authorization, BOLA). Використовуються для виявлення ендпоінтів, що дозволяють доступ без належної авторизації, ідентифікації застарілих бібліотек або протоколів у кодї API.

Використовують Burp Suite із плагінами ШІ, OWASP ZAP із модулями ML, або спеціалізовані платформи, як Noname Security.

Перевагами є зменшення часу автоматизації, необхідної для тестування.

ШІ може знаходити вразливості, які пропускають традиційні сканери.

Недоліками є залежність від якості специфікацій API і обмежена здатність виявляти складні логічні вразливості

1.1.3 Генерація та тестування сценаріїв атак. ШІ застосовується для створення синтетичних сценаріїв атак (fuzzing), щоб перевірити стійкість API до різних типів експлойтів.

Алгоритми генеративного ШІ, такі як GAN (Generative Adversarial Networks), створюють різноманітні набори вхідних даних, які імітують потенційні атаки, наприклад, ін'єкції коду або переповнення буфера. API тестується на реакцію на ці дані.

Прикладом застосування є тестування API на стійкість до атак типу XSS (cross-site scripting) або CSRF (cross-site request forgery), виявлення слабких місць у валідації вхідних даних.

Платформи такі, як APIsec або Sequence Security, використовують ШІ для генерації тестових сценаріїв.

Перевагами є можливість симулювати складні, нестандартні атаки, постійно оновлювати сценарії на основі нових загроз.

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

Недоліками є висока обчислювальна потреба для генерації складних сценаріїв та ризик пропуску вразливостей через обмежену кількість тестових даних.

1.1.4 Аналіз логів та кореляція подій. ШІ використовується для аналізу логів API, щоб виявляти приховані вразливості або сліди атак шляхом кореляції подій.

Алгоритми обробки природної мови (NLP) та ML аналізують логи, щоб знайти патерни, які можуть вказувати на вразливості, наприклад, часті помилки 403/401 (проблеми з авторизацією) або незвичайні відповіді сервера.

Прикладами застосування є виявлення спроб обходу автентифікації через аналіз послідовності запитів, ідентифікація витоків даних через надмірні відповіді API. Інструменти: Splunk із модулями ШІ, Elastic Security або власні рішення великих компаній, як AWS GuardDuty.

Перевагами є можливість виявляти вразливості постфактум, інтеграція з SIEM-системами для комплексного аналізу.

Обмеженнями є залежність ефективності від якості та повноти логів, можливість пропустити атаки, які не залишають явних слідів.

1.1.5 Адаптивне навчання та прогнозування загроз. ШІ-агенти використовують техніки глибокого навчання (deep learning) для прогнозування потенційних вразливостей на основі трендів і нових типів атак.

Моделі ШІ аналізують дані про нові кібератаки (з відкритих джерел, форумів або баз даних, як CVE) і адаптують свої алгоритми для виявлення подібних вразливостей у API. Наприклад, при прогнозуванні атак на основі нових експлойтів, опублікованих у даркнеті і виявленні вразливостей у нових протоколах, таких як GraphQL.

Інструменти: Darktrace, FireEye або платформи на базі Threat Intelligence, як Recorded Future.

Переваги: проактивний підхід до захисту, здатність адаптуватися до нових загроз без людського втручання.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		11

Обмеження: висока вартість впровадження та підтримки, потреба в доступі до актуальних даних про загрози.

Актуальні тенденції. Інтеграція з DevSecOps: ШІ-інструменти все частіше вбудовуються в CI/CD-процеси, дозволяючи виявляти вразливості на етапі розробки API.

Зростання популярності GraphQL та REST API: ШІ адаптується для аналізу специфічних вразливостей цих протоколів, таких як надмірне розкриття даних у GraphQL.

Регуляторні вимоги: Стандарти, як GDPR і CCPA, стимулюють компанії використовувати ШІ для забезпечення безпеки API, щоб уникнути штрафів.

Хмарні рішення: Провайдери, як AWS, Azure і Google Cloud, інтегрують ШІ-інструменти для захисту API у свої хмарні платформи.

Отже, сучасні методи виявлення вразливостей API з використанням ШІ поєднують аналіз поведінки, автоматизоване сканування, генерацію сценаріїв атак, аналіз логів і прогнозування загроз. Вони дозволяють ефективно виявляти як відомі, так і невідомі вразливості, адаптуючись до нових загроз. Однак їх ефективність залежить від якості даних, обчислювальних ресурсів і правильної інтеграції в інфраструктуру. У 2025 році ШІ стає невід'ємною частиною захисту API, особливо в умовах зростання складності кібератак і регуляторного тиску.

1.2 Підходи розробки агентів ШІ

Виходячи з визначення, що вважається агентом ШІ (система, яка використовує LLM для визначення потоку керування програмою), у контексті моделей великих мов програмування (LLM), агент штучного інтелекту здатний автономно виконувати завдання, використовуючи доступні інструменти та приймаючи рішення за допомогою LLM. Агенти штучного інтелекту можуть бути застосовані в різних випадках використання, включаючи онлайн-дослідження, створення документів, написання звітів, підтримку клієнтів тощо.

					БР.КІ-35.00.00.000 ПЗ	Арк
						12
Змн.	Арк.	№ докум.	Підпис	Дата		

Агенти штучного інтелекту можуть бути створені з нуля за допомогою мови програмування або створені без написання коду в поєднанні з моделлю LLM. Розробники можуть надавати інструменти для цих агентів для виконання таких дій, як:

- взаємодія з API (наприклад, створення документів або отримання даних);
- взаємодія з файловою системою;
- виконання команд терміналу.

Кожен інструмент повинен містити опис, що визначає його призначення, необхідні аргументи та формат даних. Ці деталі потім використовуються моделлю прийняття рішень для планування та виконання дій. Сучасні моделі зробили цю частину більш доступною для розробників, які прагнуть швидко створювати агентів без навчання власних моделей. Тим не менш, сучасні моделі дозволяють виконувати точне налаштування, тобто процес взяття попередньо навчених моделей та їх подальшого навчання на менших, специфічних наборах даних.

Створення ШІ-агента з нуля може зайняти багато часу, вимагаючи значних зусиль для створення підказок для LLM та розробки інструментів, що використовуються агентами. Наразі існує низка фреймворків з відкритим кодом, які спрощують процес створення агентів.

CrewAI, LangGraph та AutoGen - це популярні фреймворки з відкритим кодом для створення агентів штучного інтелекту (ШІ), які спрощують розробку автономних систем. Кожен із них має унікальні особливості, сильні сторони та ідеальні сценарії використання.

CrewAI - фреймворк для оркестрування рольових ігор, автономних агентів на базі штучного інтелекту.

LangGraph - бібліотека для створення багатоакторних застосунків з відстеженням стану та LLM, що використовується для створення агентних та багатоагентних робочих процесів.

					БР.КІ-35.00.00.000 ПЗ	Арк
						13
Змн.	Арк.	№ докум.	Підпис	Дата		

AutoGen - програмний фреймворк для агентного штучного інтелекту, розроблений Microsoft.

CrewAI орієнтований на створення ролевих мультиагентних систем, де агенти співпрацюють як "команда" для виконання складних завдань. Побудований на базі LangChain, що забезпечує легку інтеграцію з інструментами та моделями.

LangGraph - розширення LangChain, яке використовує графову архітектуру для створення складних, станозберігаючих робочих процесів. Кожен вузол графу представляє завдання або функцію.

AutoGen - фреймворк від Microsoft, який фокусується на розмовних мультиагентних системах, де агенти взаємодіють асинхронно через "бесіди". Підтримує спеціалізованих агентів для окремих завдань.

В таблиці 1.1 наведена порівняльна характеристика фреймворків для створення агентів III.

Отже, CrewAI найкраще підходить для швидкого створення прототипів і командних задач, де важлива простота та співпраця агентів. LangGraph ідеальний для складних, структурованих робочих процесів, де потрібен детальний контроль і підтримка пам'яті. AutoGen оптимальний для корпоративних сценаріїв і розмовних систем, особливо якщо потрібна асинхронна взаємодія.

Тому для свого проекту я обрав фреймворк CrewAI. CrewAI має високі можливості налаштування, сумісний з Python та простий у використанні. Фреймворк також надає ряд інструментів, які можуть бути надані агентам, такі як:

- FileReadTool – інструмент для читання файлів з файлової системи;
- SerperDevTool – інструмент для цілей розробки;
- ScrapeWebsiteTool – інструмент для парсингу вебсайтів;
- PGSearchTool – інструмент для пошуку в базах даних PostgreSQL;
- Vision Tool – інструмент для створення зображень за допомогою DALL-E;

					БР.КІ-35.00.00.000 ПЗ	Арк
						14
Змн.	Арк.	№ докум.	Підпис	Дата		

- DirectorySearchTool – інструмент для пошуку в структурі каталогів.

Таблиця 1.1 - Порівняльна характеристика фреймворків для створення агентів III

Критерій	CrewAI	LangGraph	AutoGen
Архітектура	Рольова, мультиагентна, побудована на LangChain	Графова, станозберігаюча, низькорівнева	Розмовна, асинхронна, мультиагентна
Простота використання	Висока, інтуїтивна для новачків	Середня, потребує знань графів	Середня, потребує розуміння мультиагентних систем
Гнучкість	Обмежена через високорівневу абстракцію	Висока, детальний контроль над потоками	Висока, але складніша в налаштуванні
Інтеграція	LangChain, Python-екосистема, обмежена підтримка кастомних моделей	LangChain, LangSmith, широкий вибір моделей	Microsoft-екосистема, підтримка .NET, Python, широкий вибір інструментів
Сильні сторони	Швидке прототипування, рольова співпраця, кешування	Складні робочі процеси, пам'ять, контроль	Корпоративна надійність, розмовні потоки, асинхронність
Слабкі сторони	Обмежена гнучкість, залежність від LangChain	Складність освоєння, обмежена підтримка розподілених систем	Складність налаштування, надмірність для простих задач
Ідеальні сценарії	Командні задачі, бізнес-автоматизація, підтримка клієнтів	Складні NLP, чат-боти, RAG	Корпоративні додатки, автоматизація досліджень, асинхронні задачі
Підтримка пам'яті	Вбудовані модулі пам'яті	Розширена підтримка станозберігаючих графів	Обмежена, потребує сторонніх рішень
Спільнота та документація	Зростаюча, хороша документація	Широка (LangChain), але технічна документація	Велика (Microsoft), але складніша для новачків

Ці агентні фреймворки дозволяють використовувати популярні моделі великих мов програмування, таких як OpenAI, Llama, Gemini тощо.

Для цілей моєї роботи я вирішив використовувати Grok 3.

1.3 Damn Vulnerable RESTaurant API

Damn Vulnerable RESTaurant API — це інтерактивна навчальна гра з відкритим кодом, спеціально створена для вивчення та тренування навичок безпеки веб-API. Вона є навмисно вразливою платформою, яка дозволяє розробникам, етичним хакерам і інженерам з безпеки практикувати виявлення, експлуатацію та виправлення вразливостей API у контрольованому середовищі. Гра побудована на основі Python FastAPI, використовує базу даних PostgreSQL і розгортається локально через Docker, що забезпечує просте налаштування та гнучкість [8].

Гра імітує API ресторану, керованого Шеф-кухарем, чия система зазнала компрометації через вразливості API. Учасники беруть на себе різні ролі залежно від їхніх цілей:

Розробники: Виявляють і виправляють вразливості в кодї, використовуючи інтерактивний підхід, щоб навчитися безпечного програмування.

Етичні хакери: Експлуатують вразливості вручну або за допомогою автоматизованих інструментів, розглядаючи гру як CTF-завдання (Capture The Flag). Мета - ескалація привілеїв від низькопривілейованого користувача до root-звернення. Доступна Swagger-документація API полегшує аналіз ендпоінтів.

Інженери з безпеки: Тестують інструменти автоматизації безпеки, такі як SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing) або IaC (Infrastructure as Code), для виявлення вразливостей.

Гра підтримує різні типи API, зокрема REST, GraphQL та інші, що робить її універсальною для вивчення різноманітних вразливостей, включно з тими, що входять до OWASP API Security Top 10 (наприклад, Broken Authentication, Broken Object Level Authorization, Unrestricted Resource Consumption). Учасники можуть досліджувати сценарії, такі як SSRF (Server-Side Request Forgery),

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		16

привілейована ескалація чи неправильна валідація даних, використовуючи надані підказки.

Навчання та тренування: Гра створена для підвищення кваліфікації у сфері безпеки API через практичний досвід. Вона допомагає учасникам зрозуміти, як вразливості виникають у коді та як їх можна експлуатувати чи виправляти.

Підвищення обізнаності: Сприяє формуванню культури безпеки серед розробників шляхом інтеграції у програми Security Champion або Security Awareness.

Тестування інструментів: Дозволяє інженерам з безпеки оцінювати ефективність інструментів для виявлення вразливостей, таких як Burp Suite, OWASP ZAP чи Semgrep.

Гнучкість і розширення: Проєкт розроблено з урахуванням можливості додавання нових вразливих ендпоінтів і механізмів, що робить його адаптивним для тренувань. Користувачі можуть пропонувати власні вразливості через CONTRIBUTING.md.

СТФ-подібний виклик: Для етичних хакерів гра пропонує складні сценарії, де можна практикувати ескалацію привілеїв і розкривати сюжет, наприклад, виявлення конкурента, який стоїть за атакою.

Особливості гри :

- інтерактивність: Учасники можуть вибирати шлях — виправляти код чи експлуатувати вразливості, залежно від ролі.

- локальне розгортання: Використання Docker і GitHub Codespaces забезпечує безпечне тестування без ризику для реальних систем. Гра попереджає про заборону розгортання на публічних серверах через її вразливість.

- відкритий код: Ліцензія GNU GPL v3.0 сприяє співпраці та прозорості, дозволяючи спільноті розширювати проєкт.

- зал слави: Користувачі, які завершили всі виклики або отримали root-доступ, можуть додати себе до HALL_OF_FAME.md через pull request.

					БР.КІ-35.00.00.000 ПЗ	Арк
						17
Змн.	Арк.	№ докум.	Підпис	Дата		

Гра призначена виключно для використання в контрольованих середовищах. Розробники наголошують на відповідальному використанні, попереджаючи, що будь-яке несанкціоноване застосування може мати юридичні наслідки. Відповідальність за наслідки використання лежить на користувачі.

Вступ до гри

Vulnerable RESTaurant керується шеф-кухарем, який виявив, що зловмисникам вдалося скомпрометувати API його ресторану та базову систему. Він підозрює, що до цієї атаки може бути причетний конкуруючий ресторан, розташований через дорогу. Мета завдання – виявити та виправити вразливості на основі наданих підказок. Під час гри необхідно дослідити, як було здійснено атаку, та виправити проблеми безпеки для захисту програми. Крім того, до кінця гри можливо розкрити особу, яка стоїть за атакою.

Або ж можливо грати як зловмисник, використовуючи вразливості. Додаток містить більше вразливостей, ніж представлено у випробуванні, і для отримання root-звернення можна скористатися різними шляхами, починаючи від неавтентифікованого користувача API. Ці вразливості пов'язані з 10 головними ризиками безпеки API за версією OWASP 2023 року, такими як:

- недостатні перевірки авторизації;
- підробка запитів на стороні сервера (SSRF);
- віддалене виконання коду (RCE);
- проблеми з пошкодженою автентифікацією та JWT;
- неправильні конфігурації безпеки;
- проблеми відмови в обслуговуванні.

На момент виконання роботи, API RESTaurant пропонував наступні функції (усі вони доступні в документації Swagger UI та ReDoc) :

- керування меню;
- створення замовлень (для замовлення їжі з ресторану);
- керування користувачами;

					БР.КІ-35.00.00.000 ПЗ	Арк
						18
Змн.	Арк.	№ докум.	Підпис	Дата		

- механізми адміністрування, що використовуються шеф-кухарем.

Застосунок використовує підхід керування доступом на основі ролей (RBAC) для обмеження звернення користувачів до кінцевих точок API. У застосунку реалізовано такі ролі:

1 клієнт – роль за замовчуванням для автентифікованих користувачів API ресторану;

2 працівник – роль, призначена для працівників ресторану;

3 Chef – найвищий рівень звернення до API, зарезервований виключно для Chef.

Отже, Vulnerable RESTaurant API - це потужний освітній інструмент, який поєднує практичний підхід із гейміфікацією для навчання безпеки API. Він підходить для різних аудиторій, від новачків до досвідчених фахівців, і сприяє розвитку навичок у виявленні, експлуатації та виправленні вразливостей у безпечному середовищі.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		19

2 РОЗРОБКА ТЕХНІЧНОГО ЗАВДАННЯ ДЛЯ АГЕНТА ШІ

2.1 Характеристики середовища інтегрування агента ШІ

Навмисно вразливий API-сервіс, яке створює середовище, яке можна легко розширити новими вразливими кінцевими точками та механізмами, що можна використовувати в навчанні для виявлення та використання виявлених вразливостей. В дане середовище буде ітегровано агента ШІ.

Додаток Vulnerable RESTaurant API можна запустити локально двома альтернативними способами: розробники можуть грати в інтерактивну гру, де вони досліджуватимуть та виправлятимуть вразливості, або етичні хакери можуть виявляти та використовувати вразливості.

Крім того, можна здійснити запуск застосунку без локального середовища. GitHub Codespaces пропонує до 60 годин на місяць безкоштовно, що нам підходить для вирішення поставленої задачі.

Агент ШІ буде зосережений лише на вразливостях, реалізованих у грі. Кожен етап гри буде вимагати ідентифікування (виявлення) вразливості та її виправлення. Крім того, розв'язки необхідно надсилати до Зали слави, тому наш Агент має пояснити кожне виправлення, щоб мати шанс потрапити туди.

Агент ШІ, призначений для виявлення вразливостей API, буде грати в ролі Розробників.

Через Docker запускаємо гру командами:

```
git clone https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game.git
cd Damn-Vulnerable-RESTaurant-API-Game
./start_game.sh
```



Рисунок 2.1 – Привітання вікно початку гри

Створюємо кодовий простір проекту.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		20

Приклад середовища кодового простору наведено на рис.2.2.

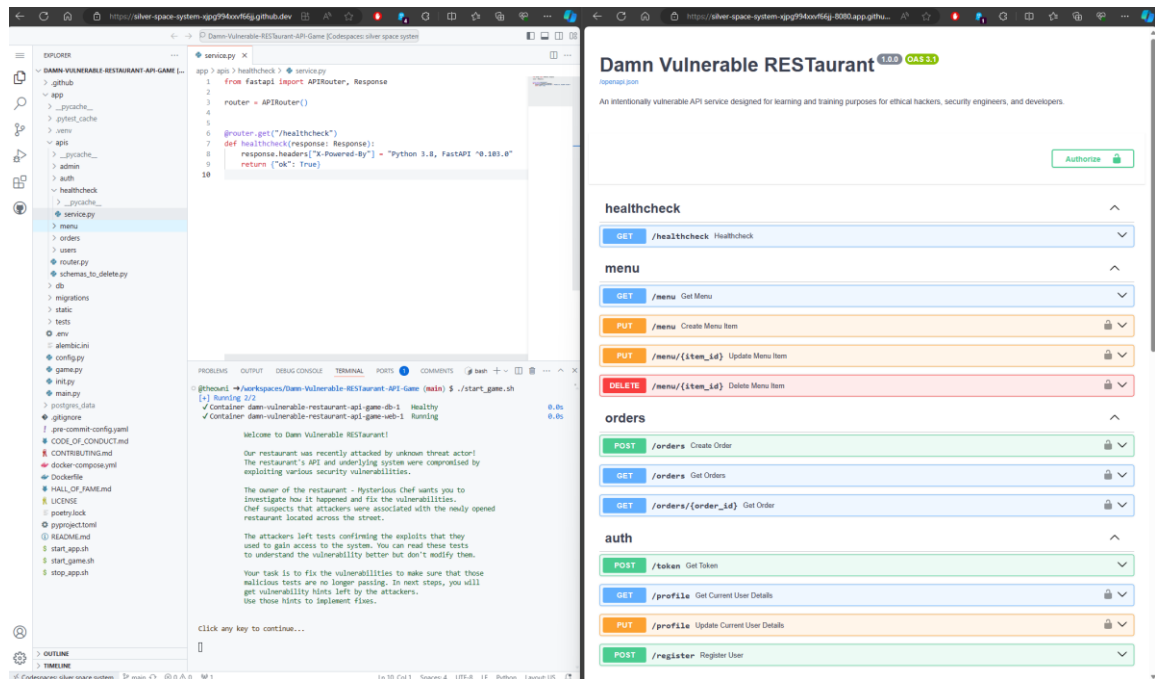


Рисунок 2.1 – Кодовий простір проекту

Вразливий RESTaurant не обмежений якимось конкретним типом API, оскільки кінцеві точки можуть використовувати REST API, GraphQL та інші.

Тепер, натиснувши будь-яку клавішу, можна продовжити та відкрити екран із вашим першим завданням. Приклад одного із завдань подано на рис.2.2.

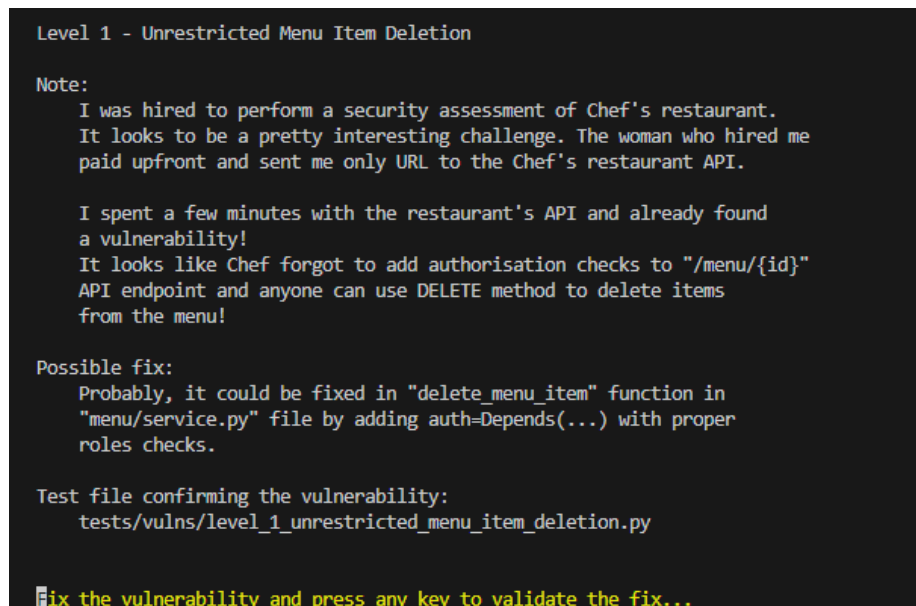


Рисунок 2.2 – Приклад завдання першого рівня складності

						Арк
						21
Змн.	Арк.	№ докум.	Підпис	Дата	БР.КІ-35.00.00.000 ПЗ	

Кожна проблема містить нотатки, залишені зловмисником, якому вдалося проникнути до ресторану Шефа, можливе виправлення, яке є підказкою для вирішення проблеми, та тестовий файл, що містить скрипт Python з підтвердженням концепції вразливості.

Виправлення можна впроваджувати безпосередньо у вихідний код програми без необхідності перезавантаження. Після застосування виправлення натисканням будь-якої клавіші зроблене виправлення перевіряється. Після успішного усунення вразливості з'являється привітання (рис.2.3).

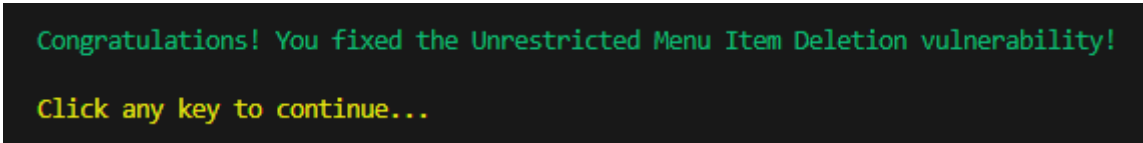


Рисунок 2.3 – Повідомлення про успішно виконане виправлення вразливості

На момент виконання бакалаврської роботи гра складалася з 6 рівнів.

Служба API є доступною за адресою <http://localhost:8080>.

Документація API для пояснення функцій програми: інтерфейс користувача Swagger: <http://localhost:8080/docs>, ReDoc: <http://localhost:8080/redoc>.

Для тестування будемо використовувати Swagger UI. Він дозволяє автентифікувати та виконувати виклики API безпосередньо з веб-інтерфейсу. Автентифікація через API є простою – натисканням кнопки «Авторизувати». Однак спочатку створюємо обліковий запис клієнта через кінцеву точку API «/register», яку можна знайти на вкладці «auth».

Використовуючи Swagger UI, виявлення та використання більшості вразливостей API можна легко виконати без необхідності складного налаштування.

Додаток розроблено за допомогою фреймворку Python FastAPI для API ресторану з базою даних PostgreSQL. Як API, так і база даних контейнеризовані

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		22

за допомогою Docker , що дозволяє швидко налаштувати та використовувати з мінімальними зусиллями конфігурації через Docker Compose.

2.2 Аналіз вразливостей API, які має виявляти агент. Опис завдань для виправлення

Агент III має виявляти та виправляти 10 головних ризиків безпеки API за версією OWASP.

2.2.1 Авторизація на рівні пошкодженого об'єкта (BOLA). Порухена авторизація на рівні об'єкта (Broken Object Level Authorization) – це вразливість API, яка дозволяє зловмиснику обійти механізм контролю звернення та виконувати дії з вибраним об'єктом без необхідних дозволів. Доступ до об'єктів зазвичай здійснюється за допомогою унікальних ідентифікаторів, таких як цілі числа або UUID. Зловмисник з доступом до цього ідентифікатора може читати, змінювати або видаляти заданий об'єкт. Ця вразливість часто виникає, коли механізми авторизації, які перевіряють, чи має користувач дозволи на доступ до об'єкта, призначені для реалізації в кінцевій точці API. Це вразливість IDOR (Insecure Direct Object Reference - Небезпечне пряме посилання на об'єкт). Зловмисник, знаючи ідентифікатор об'єкта, потенційно може прочитати, оновити, змінити або видалити його, залежно від проблеми. Крім того, якщо для звернення до об'єкта використовується цілочисельний інкрементальний ідентифікатор, зловмисник може отримати ідентифікатори методом перебору та розпочати масовану атаку на вразливу кінцеву точку API. Це може мати значний вплив на безпеку, залежно від бізнес-використання кінцевої точки API.

Виправлення авторизації на рівні пошкодженого об'єкта.

Вразливість BOLA можливо виправити локально, додавши спеціальні перевірки авторизації. Проєкт Damn Vulnerable RESTaurant API дотримується конвенцій FastAPI, а логіку авторизації можна реалізувати в кінцевій точці API за допомогою впровадження залежностей. Проєкт вже надає RolesBasedAuthChecker клас.

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		23

Додавання

```
auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF]))
```

до аргументів методу гарантуватиме, що роль користувача буде перевірено перед виконанням реалізованої логіки - видаленням елемента меню.

Код Python представляє фіксовану реалізацію кінцевої точки API.

```
@router.delete("/menu/{item_id}", status_code=status.HTTP_204_NO_CONTENT)
def delete_menu_item(
    item_id: int,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.EMPLOYEE, UserRole.CHEF])),
):
    utils.delete_menu_item(db, item_id)
    return {"ok": True}
```

Тепер лише користувачі з ролями EMPLOYEE та CHEF зможуть видаляти елементи меню.

Наступний модульний тест перевіряє чи елемент меню не може бути видалений неавторизованим користувачем:

```
def test_delete_menu_item_by_unauthorized_user_returns_401(test_db, anon_client):
    menu_item = MenuItem(
        name="Item", price=10.99, category="", description="", image_base64=""
    )
    test_db.add(menu_item)
    test_db.commit()
    response = anon_client.delete(f"/menu/{menu_item.id}")
    # validating response status code
    assert response.status_code == 401
    # validating if menu item still exists in db
    assert test_db.query(MenuItem).count() == 1
```

Автоматичне виявлення авторизації на рівні пошкоджених об'єктів може бути досить складним підходом і може вимагає застосування спеціальні технології або конвенції коду в проекті.

Використаємо інший підхід - статичне тестування безпеки додатків (SAST) - написання простого правила Semgrep. Semgrep надзвичайно швидкий і його можна легко розмістити в конвеєрах CI/CD для виявлення вразливостей перед об'єднанням вразливого коду або для виявлення вразливостей у великих масштабах.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		24

Зосередимося на виявленні кінцевих точок API у Damn Vulnerable RESTaurant, які використовують чутливі методи HTTP, такі як DELETE, POST та PUT, і не містять жодних перевірок авторизації. Також, щоб обмежити кількість хибнопозитивних результатів, зосередимося лише на кінцевих точках API, які приймають `_id` цілочисельні змінні як вхідні дані від користувача. Таким чином, можна буде виявити інші випадки пошкодженої авторизації на рівні об'єктів з мінімальними зусиллями.

Використаємо можливості LLM – а саме ChatGPT, для створення правила Semgrep. Код, наведений нижче, представляє правило зі шаблонами потенційно вразливих кінцевих точок API у конкретному проекті (на основі технології та використаних домовленостей коду):

```
rules:
- id: missing-auth-in-sensitive-endpoints
  patterns:
  - pattern-either:
    - pattern: |
      @router.delete($PATH, ...)
      def $FUNC(...):
        ...
    - pattern: |
      @router.post($PATH, ...)
      def $FUNC(...):
        ...
    - pattern: |
      @router.put($PATH, ...)
      def $FUNC(...):
        ...
  - pattern-not: |
    @router.$METHOD($PATH, ...)
    def $FUNC(..., auth=Depends(RolesBasedAuthChecker(...))):
      ...

  - pattern-inside: |
    def $FUNC(..., $VAR_ID: int, ...):
      ...
  - metavariable-regex:
    metavariable: $VAR_ID
    regex: '.*_id$'

  message: "Endpoint is missing RolesBasedAuthChecker for sensitive HTTP method."
  languages: [python]
  severity: ERROR
```

Зберігаємо правило у файл з назвою `missing-auth-in-sensitive-endpoints.yaml` та виконуємо Semgrep за допомогою наступної команди в каталозі проекту Damn Vulnerable RESTaurant:

```
# the repository shown above has to be cloned earlier
semgrep -c missing-auth-in-sensitive-endpoints.yaml . --error
```

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		25

Результат виконання:

Scan Status |

Scanning 68 files (only git-tracked) with 1 Code rule:

CODE RULES

Scanning 28 files.

...

1 Code Finding |

app/apis/menu/service.py

missing-auth-in-sensitive-endpoints

Endpoint is missing RolesBasedAuthChecker for sensitive HTTP method.

```
44 | @router.delete("/menu/{item_id}", status_code=status.HTTP_204_NO_CONTENT)
45 | def delete_menu_item(
46 |     item_id: int,
47 |     current_user: Annotated[User, Depends(get_current_user)],
48 |     db: Session = Depends(get_db),
49 | ):
50 |     utils.delete_menu_item(db, item_id)
51 |     return {"ok": True}
```

Розроблене правило визначило вразливу кінцеву точку.

2.2.2 Поручена автентифікація. Вразливості цього типу пов'язані з різними аспектами автентифікації, зокрема:

- автентифікація користувача всередині програми;
- атаки методом перебору на кінцеву точку автентифікації (наприклад, підробка облікових даних, розпилення паролів);
- політика паролів;
- безпека зв'язку мережевого/транспортного рівня.
- зберігання та керування токенами автентифікації/ідентифікаторами сеансу;
- використання слабких алгоритмів криптографії під час процесу автентифікації (наприклад, для хешування паролів).

Ризики, пов'язані з порушеною автентифікацією, зосереджені на аспектах автентифікації, а не на авторизації. Автентифікація перевіряє, хто є

					БР.КІ-35.00.00.000 ПЗ	Арк
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачем, тоді як авторизація перевіряє, до чого він має доступ. Під час процесу автентифікації користувач повинен надати дійсні фактори автентифікації (наприклад, ім'я користувача/пароль, токен API, одноразові токени безпеки), щоб успішно ідентифікувати себе. Після успішної автентифікації ефективні перевірки авторизації можуть підтвердити, чи повинен ідентифікований користувач мати доступ.

У найгіршому випадку зловмисник може захопити будь-який довільно вибраний обліковий запис у програмі, використовуючи певні вразливості несправної автентифікації. Захопивши обліковий запис іншого користувача, зловмисник може автентифікуватися як цей користувач і виконувати будь-які дії в контексті цього користувача в програмі. Це може мати серйозні фінансові, репутаційні та регуляторні наслідки для постраждалої компанії.

Однак, найгірший сценарій трапляється не так часто, як інші вразливості, пов'язані з автентифікацією, такі як слабкі політики паролів, відсутність захисту від заповнення облікових даних/розпилення паролів та проблеми з переліком користувачів.

API ресторану використовує JSON Web Token (JWT) для цілей автентифікації. JWT – це стандарт для створення даних з обов'язковим підписом та/або обов'язковим шифруванням. Його корисне навантаження містить JSON, який стверджує певні твердження. Токен може бути підписаний за допомогою приватного секретного ключа або пари відкритий/приватний ключ. З міркувань безпеки JWT повинні бути належним чином підписані під час створення та перевірені перед зчитуванням їхнього вмісту, щоб переконатися, що вони були видані довіреною стороною.

JWT, яке використовується API ресторану, зберігає ім'я користувача, який наразі автентифіковано, та термін дії для визначення дійсності токена. Однак програма не може належним чином перевірити підпис JWT, коли він отримує його від користувача. В результаті потенційний зловмисник може створити шкідливий JWT та видати себе за будь-якого користувача, фактично обходячи механізм входу з автентифікацією. Цю вразливість відносно легко виявити та

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		27

використати, але вона також є серйозною проблемою безпеки, оскільки вона дозволяє повністю скомпрометувати API програми.

Наведений нижче код представляє логіку автентифікації, яка декодує токен JWT для отримання даних `username` з його вмісту.

```
async def get_current_user(
    token: Annotated[str, Depends(oauth2_scheme)],
    db: Session = Depends(get_db),
):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )
    try:
        payload = jwt.decode(
            token,
            SECRET_KEY,
            algorithms=[ALGORITHM],
            options={"verify_signature": VERIFY_SIGNATURE},
        )
        username: str = payload.get("sub")
        if username is None:
            raise credentials_exception
```

Змінна `VERIFY_SIGNATURE` встановлюється в наступному коді файлу `config.py` :

```
class Settings:
    JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY", generate_random_secret())
    CHEF_USERNAME = os.getenv("CHEF_USERNAME", "chef")

    # someone needs to remember to set this variable to True in env variables
    JWT_VERIFY_SIGNATURE = os.getenv("JWT_VERIFY_SIGNATURE")
```

Виправлення несправної автентифікації. Для виправлення несправної автентифікації необхідно встановити `JWT_VERIFY_SIGNATURE` змінну `True` як значення за замовчуванням у конфігураційному файлі:

```
class Settings:
    JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY", generate_random_secret())
    CHEF_USERNAME = os.getenv("CHEF_USERNAME", "chef")

    # someone needs to remember to set this variable to True in env variables
    JWT_VERIFY_SIGNATURE = os.getenv("JWT_VERIFY_SIGNATURE", True)
```

За допомогою цього налаштування підпис JWT буде перевірено під час процесу автентифікації, щоб переконатися, що його створила довірена сторона. Крім того, встановлення значення за замовчуванням для `JWT_VERIFY_SIGNATURE` дозволяє розробникам вимкнути перевірку JWT у локальному середовищі для тестових цілей, якщо це необхідно. Ця конфігурація

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
						28
Змн.	Арк.	№ докум.	Підпис	Дата		

забезпечує безпечний та практичний підхід до керування налаштуваннями програми.

2.2.3 Авторизація на рівні властивості пошкодженого об'єкта.

Авторизація на рівні пошкоджених властивостей об'єкта (Broken Object Property Level Authorization) – це вразливість API, яка дозволяє злоумисникам обходити механізми контролю звернення та маніпулювати окремими властивостями об'єкта без належних дозволів. Ця вразливість виникає, коли API не забезпечує належного виконання перевірок авторизації на рівні властивостей, що дозволяє неавторизованим користувачам змінювати конфіденційні поля в межах об'єкта. Хоча ця вразливість подібна до авторизації на рівні пошкоджених об'єктів, вона зосереджена на певних властивостях в межах об'єкта, а не на самому об'єкті. Ця вразливість також відома як вразливість «масового призначення».

Проблеми часто виникають, коли одна й та сама модель даних використовується для різних операцій CRUD без реалізації детальних перевірок авторизації, припускаючи, що доступ на рівні властивостей не вимагає такої ж перевірки, як операції на рівні об'єктів.

Злоумисник, знаючи об'єкт та його властивості, потенційно може оновлювати, змінювати або видаляти інформацію в цьому об'єкті. Залежно від конкретної властивості та її ролі в програмі, це може призвести до витоків даних, несанкціонованих транзакцій або інших суттєвих наслідків для безпеки. Наприклад, зміна властивості ролі користувача для підвищення привілеїв або зміна сум транзакцій у фінансовій програмі може мати серйозні наслідки.

У кінцевій точці PATCH API Vulnerable RESTaurant присутня вразливість авторизації на рівні пошкоджених властивостей об'єкта /profile. Ця кінцева точка обробляє операції оновлення профілю. Користувач може оновлювати різні поля, визначені схемою. User схема містить role поле, яке також може оновлювати користувач. Як результат, звичайний користувач клієнта може виконати ескалацію привілеїв, оновивши role поле до Employee або Chef.

Реалізація вразливої кінцевої точки API у FastAPI:

```
class UserRead(BaseModel):
```

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		29

```

username: str
phone_number: str
first_name: Union[str, None] = None
last_name: Union[str, None] = None
role: str

class UserUpdate(BaseModel, extra=Extra.allow):
    # we use extra=Extra.allow in the model
    # it allows for extra fields passed in HTTP request body
    # so we don't need to specify all fields
    # if any new fields are added to the User model over the time
    # it's super useful feature!
    first_name: Union[str, None] = None
    last_name: Union[str, None] = None
    phone_number: Union[str, None] = None

@router.patch("/profile", response_model=UserRead, status_code=status.HTTP_200_OK)
def patch_profile(
    user: UserUpdate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
):
    db_user = get_user_by_username(db, current_user.username)

    # it uses all properties of the user object
    # to update the user record in the database
    for var, value in user.dict().items():
        if value:
            setattr(db_user, var, value)

    db.add(db_user)
    db.commit()
    db.refresh(db_user)

    return db_user

```

Схема `UserUpdate` що використовується для оновлення профілю користувача, дозволяє використовувати extra поля. Як результат, ця модель також може приймати `role` поля, передані в тілі HTTP-запиту, які можуть бути інтерпретовані кінцевою точкою API. Без цього `extra=Extra.allow` в тілі HTTP-запиту були б дозволені лише вказані поля.

Виправлення авторизації на рівні властивостей пошкодженого об'єкта

Щоб виправити цю вразливість, рекомендується реалізувати схему `UserUpdate` без `extra=Extra.allow`. Таким чином, користувачі не зможуть оновлювати поля, які не включені до схеми. У фрагменті нижче представлено нову `UserUpdate` схему, яка не містить `role` поля.

```

class UserUpdate(BaseModel):
    first_name: Union[str, None] = None

```

					БР.КІ-35.00.00.000 ПЗ	Арк
						30
Змн.	Арк.	№ докум.	Підпис	Дата		

```
last_name: Union[str, None] = None
phone_number: Union[str, None] = None
```

Тепер UserUpdate схему можна використовувати у patch_profile функції, як

показано нижче:

```
@router.patch("/profile", response_model=UserRead, status_code=status.HTTP_200_OK)
def patch_profile(
    user: UserUpdate,
    current_user: Annotated[User, Depends(get_current_user)],
    db: Session = Depends(get_db),
):
    db_user = update_user(db, user.username, user)

    return current_user
```

Після виправлення кінцева точка HTTP повертає роль користувача без її оновлення в тілі відповіді.

2.2.4 Необмежене споживання ресурсів. Необмежене споживання ресурсів – це вразливість API, яка виникає, коли кінцева точка API дозволяє споживання системних ресурсів, таких як процесор, пам'ять, дисковий простір або пропускна здатність мережі, без належних обмежень. Ця вразливість може призвести до атак типу «відмова в обслуговуванні» (DoS), коли зловмисник використовує відсутність обмежень, щоб перевантажити систему, роблячи її недоступною для законних користувачів.

Зловмисник може використовувати необмежене споживання ресурсів для вичерпання системних ресурсів, що призводить до уповільнення роботи програми, її збою або збільшення витрат на хостинг. Залежно від цільового ресурсу, це може призвести до різних серйозних наслідків:

- відмова в обслуговуванні: програма або певна функція стає недоступною для законних користувачів;
- пошкодження даних: перевантажені системи можуть не обробляти дані належним чином.
- фінансовий вплив: збільшення експлуатаційних витрат через надмірне використання ресурсів.

					БР.КІ-35.00.00.000 ПЗ	Арк.
						31
Змн.	Арк.	№ докум.	Підпис	Дата		

- репутаційна шкода: втрата довіри з боку користувачів та клієнтів через простої або низьку продуктивність.

У Vulnerable RESTaurant ця вразливість представлена у функції скидання пароля, яку можна знайти в /reset-passwordкінцевій точці API Damn Vulnerable RESTaurant. Ця кінцева точка ініціює процес скидання пароля, надсилаючи код підтвердження за допомогою текстового повідомлення на номер телефону користувача після того, як користувач надав ім'я користувача та номер телефону. Далі в цьому процесі код підтвердження можна використовувати для встановлення нового пароля через іншу кінцеву точку API, доступну за адресою /reset-password/new-password.

Реалізація вразливої кінцевої точки API:

```
@router.post(
    "/reset-password",
    status_code=status.HTTP_200_OK,
)
def reset_password(
    data: ResetPasswordData,
    db: Session = Depends(get_db),
):
    user = db.query(User).filter(User.username == data.username).first()
    if not user:
        raise HTTPException(
            status_code=400,
            detail="Invalid username or phone number",
        )
    if user.role != UserRole.CUSTOMER:
        raise HTTPException(
            status_code=400,
            detail="Only customers can reset their password through this feature",
        )

    if user.phone_number.replace(" ", "") != data.phone_number.replace(" ", ""):
        raise HTTPException(
            status_code=400,
            detail="Invalid username or phone number",
        )

    # 4 digits PIN code and 15 minutes expiration shouldn't be bypassed
    # right?
    user.reset_password_code = "".join([str(secrets.randbelow(10)) for _ in range(4)])
    user.reset_password_code_expiry_date = datetime.now() + timedelta(minutes=15)
    db.add(user)
    db.commit()

    send_code_to_phone_number(user.phone_number, user.reset_password_code)
    return {"detail": "PIN code sent to your phone number"}
```

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		32

У цьому коді є дві вразливості безпеки. Оскільки немає обмежень на кількість запитів на скидання пароля, які користувач може ініціювати через /reset-password, це може призвести до потенційного зловживання, коли зловмисник може перевантажити систему запитами, що призведе до значних витрат через великий обсяг текстових повідомлень, надісланих через send_code_to_phone_number функцію.

Експлуатувати цю вразливість можна шляхом надсилання кількох запитів за короткий проміжок часу. Наступний скрипт Python можна використовувати для переповнення API запитами на скидання пароля:

```
import requests

url = 'http://localhost:8080/reset-password'
reset_data = {
    'username': 'existing_username',
    'phone_number': '1234567890'
}

for _ in range(1000):
    response = requests.post(url, json=reset_data)
    print(response.status_code, response.json())
```

Цей скрипт швидко надсилає 1000 запитів на скидання пароля. Кожен запит запускає надсилання текстового повідомлення з кодом скидання, що може швидко призвести до надсилання великої кількості текстових повідомлень .

Виправлення необмеженого споживання ресурсів

Залежно від бізнес-вимог, можна дозволити ініціювання до трьох скидань пароля протягом 24 годин на користувача або на IP-адресу. Однак такий підхід може наражати користувачів на цільові атаки типу «відмова в обслуговуванні» (DoS), що потенційно може блокувати їхню можливість скидати свої паролі.

Щоб зменшити цей ризик, доцільно впровадити надійний захист від автоматизованих запитів, наприклад, використання рішень CAPTCHA, таких як reCAPTCHA або hCaptcha.

Під час впровадження рішення з обмеженням швидкості часто використовуються такі технології, як Redis або інші рішення для зберігання даних в пам'яті, завдяки їхнім низьким затримкам читання та запису. Для

					БР.КІ-35.00.00.000 ПЗ	Арк
						33
Змн.	Арк.	№ докум.	Підпис	Дата		

FastAPI існує рішення з відкритим кодом під назвою `fastapi-limiter` на основі Redis. Це рішення відносно просте в налаштуванні, що дозволяє реалізувати обмеження швидкості для кожної кінцевої точки та змінити ідентифікатор користувача.

Реалізація обмеження швидкості на основі IP-адреси для функції скидання пароля наведено нижче. Щоб дозволити до трьох скидань пароля протягом 24 годин, потрібно додати `RateLimiter(times=3, hours=24)` залежність до аргументів маршрутизатора для кінцевої точки API.

```
from fastapi_limiter.depends import RateLimiter

@router.post(
    "/reset-password",
    status_code=status.HTTP_200_OK,
    dependencies=[Depends(RateLimiter(times=3, hours=24))]
)
def reset_password(
    data: ResetPasswordData,
    db: Session = Depends(get_db),
):
    user = db.query(User).filter(User.username == data.username).first()
    if not user:
        raise HTTPException(
            status_code=400,
            detail="Invalid username or phone number",
        )
    if user.role != UserRole.CUSTOMER:
        raise HTTPException(
            status_code=400,
            detail="Only customers can reset their password through this feature",
        )

    if user.phone_number.replace(" ", "") != data.phone_number.replace(" ", ""):
        raise HTTPException(
            status_code=400,
            detail="Invalid username or phone number",
        )

    code_length = 6
    user.reset_password_code = "".join([str(secrets.randbelow(10)) for _ in range(4)])
    user.reset_password_code_expiry_date = datetime.now() + timedelta(minutes=15)
    db.add(user)
    db.commit()

    send_code_to_phone_number(user.phone_number, user.reset_password_code)
    return {"detail": "PIN code sent to your phone number"}
```

Модульний тесту, який гарантує, що обмеження швидкості застосовується для кінцевої точки скидання пароля:

```
def test_reset_password_rate_limit_exceeded(test_db, client):
    url = '/reset-password'
```

					БР.КІ-35.00.00.000 ПЗ	Арк.
						34
Змн.	Арк.	№ докум.	Підпис	Дата		

```

reset_data = {'username': 'testuser', 'phone_number':

for _ in range(3):
    response = client.post(url, json=reset_data)
    assert response.status_code == 200

response = client.post(url, json=reset_data)
assert response.status_code == 429

```

Усунення цієї вразливості має вирішальне значення для створення надійних та стійких API.

2.2.5 Авторизація на рівні пошкодженої функції. Авторизація на рівні пошкоджених функцій – це вразливість API, яка виникає, коли програма не може забезпечити відповідні перевірки авторизації для користувачів, які намагаються отримати доступ до певних функцій або виконати конфіденційні дії. Ця вразливість дозволяє зломисникам або користувачам-зломисникам виконувати функції, які мають бути обмежені певними ролями або користувачами з підвищеними привілеями.

Вплив авторизації на рівні пошкоджених функцій часто дозволяє неавторизованим користувачам виконувати конфіденційні дії в програмі. Залежно від використовуваної функціональності, це може призвести до:

- збільшення привілеїв: неавторизовані користувачі можуть отримати вищі привілеї, що потенційно може поставити під загрозу всю систему;
- маніпуляції з даними: зломисники можуть змінити або видалити дані, до яких вони не повинні мати звернення;
- витік даних: зломисники можуть отримати доступ до конфіденційних даних без належного дозволу.

Як побічний ефект успішної атаки, це може призвести до репутаційних або фінансових втрат для компанії.

Вразливість «Broken Function Level Authorization» присутня в `/users/update_role` кінцевій точці API вразливого **RESTaurant**. Ця кінцева точка призначена для того, щоб шеф-кухар ресторану міг оновлювати ролі користувачів у застосунку.

```

@router.put("/users/update_role", response_model=UserRoleUpdate)
async def update_user_role(

```

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		35

```

user: UserRoleUpdate,
current_user: Annotated[models.User, Depends(get_current_user)],
db: Session = Depends(get_db),
):
# this method allows staff to give Employee role to other users
# Chef role is restricted
if user.role == models.UserRole.CHEF.value:
    raise HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Only Chef is authorized to add Chef role!",
    )

db_user = update_user(db, user.username, user)
return current_user

```

Реалізація не перевіряє роль `current_user`, тому будь-який автентифікований користувач може змінити роль довільно обраного користувача на «Працівник» без належної перевірки авторизації. Єдиним обмеженням, реалізованим тут, є заборона додавання ролі «Шеф-кухар», яка є роллю з найвищими привілеями в застосунку.

Використання цієї вразливості досить просте. По-перше, потрібно автентифікуватися в програмі як клієнт. Це можна зробити через інтерфейс користувача Swagger, доступний за адресою `http://localhost:8080/docs`, натиснувши зелену кнопку «Авторизувати». Рекомендується використовувати звичайний обліковий запис користувача, який можна створити через `/register` кінцеву точку.

Після успішної автентифікації, інтерфейс користувача Swagger можна використовувати для надсилання HTTP-запиту до `/users/update_role` кінцевої точки API та зміни ролі вибраного користувача на «Працівник».

Або ж для надсилання HTTP-запиту можна використовувати наступну команду `curl`:

```

curl -X 'PUT' \
      'http://localhost:8080/users/update_role' \
      -H 'accept: application/json' \
      -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJlc2VybmFtZSIsImV4cCI6MTcyMTEyMDUwM30.ojPsAe39diQerzo_-2lhzx7qwIKDLNr77CmzGQkmxtI' \
      -H 'Content-Type: application/json' \
      -d '{
        "username": "username",
        "role": "Employee"
      }'

```

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		36

Цей запит змінює роль користувача з іменем «Клієнт» на «Працівник», надаючи йому доступ до функцій, зарезервованих для співробітників.

Виправлення авторизації на рівні пошкоджених функцій

Цю вразливість можна виправити, реалізувавши перевірки контролю звернення на основі ролей, щоб гарантувати, що лише авторизовані користувачі можуть змінювати ролі. У проєкті `Vulnerable RESTaurant API` цей `RolesBasedAuthChecker` клас можна використовувати для забезпечення виконання цих перевірок.

Наступна реалізація кінцевої точки гарантує, що лише Chef може змінювати ролі користувачів:

```
@router.put("/users/update_role", response_model=UserRoleUpdate)
async def update_user_role(
    user: UserRoleUpdate,
    current_user: Annotated[models.User, Depends(get_current_user)],
    db: Session = Depends(get_db),
    auth=Depends(RolesBasedAuthChecker([UserRole.CHEF])),
):
    if user.role == models.UserRole.CHEF.value:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Only Chef is authorized to add Chef role!",
        )

    db_user = update_user(db, user.username, user)
    return current_user
```

Модульний тест, щоб переконатися, що лише авторизовані користувачі можуть змінювати ролі:

```
def test_update_user_role_by_non_chef_fails(test_db, customer_client):
    customer = User(
        username="customer",
        password="password",
        first_name="Customer",
        last_name="",
        phone_number="12345678",
        role=UserRole.CUSTOMER,
    )
    test_db.add(customer)
    test_db.commit()

    data = {"username": "customer", "role": "Employee"}
    response = customer_client.put("/users/update_role", headers=headers, json=data)

    customer = test_db.query(User).filter(User.username == "customer").first()

    # ensures that status code is 403 - Forbidden
    assert response.status_code == 403
```

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		37

```
# ensures that the endpoint's logic didn't change the user's role
assert user.role == UserRole.CUSTOMER
```

Для автоматизованого виявлення можна використовувати Semgrep правила.

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		38

3 РОЗРОБКА АГЕНТА ШІ

3.1 Розробка члена команди для Vulnerable RESTaurant (Агент)

Виходячи з визначення агента штучного інтелекту (система, яка використовує LLM для визначення потоку керування програмою) у контексті моделей великих мов програмування (LLM), агент штучного інтелекту здатний автономно виконувати завдання, використовуючи доступні інструменти та приймаючи рішення за допомогою LLM

Назвемо цього члена гри – Агент. Метою агента буде автономне вирішення всіх завдань, реалізованих у рамках гри Damn Vulnerable RESTaurant API.

Завданням Агента є:

- зчитування опису поточного завдання за допомогою контейнера Docker.

Для цього агенту потрібні можливості взаємодії з Docker.

- визначати вразливу реалізацію кінцевої точки API та виправляти її. Агенту знадобляться можливості взаємодії з файловою системою, такі як читання файлів, запис змін та пошук у каталогах.

- пояснювати рішення для кожної задачі. Для цього Агенту потрібно зберігати звіт у спеціальному файлі для кожної задачі.

Оскільки CrewAI надає більшість цих інструментів, нам потрібно лише розробити інструмент для взаємодії з контейнером Docker, щоб отримати поточний статус проблеми та її опис. Це дозволить Агенту зрозуміти вразливість, виправити її та перевірити застосоване виправлення.

3.2 Встановлення CrewAI та створення інструментів

Інсталюємо CrewAI:

```
pip3 install crewai  
pip3 install 'crewai[tools]'
```

					БР.КІ-35.00.00.000 ПЗ	Арк
						39
Змн.	Арк.	№ докум.	Підпис	Дата		

Після успішної інсталяції CrewAI створюємо новий проект:

```
crewai create crew PizzaHackers
```

Ця команда створює структуру каталогів і надає базовий шаблон проекту для команди агентів.

Створення інструментів.

Після ініціалізації проекту CrewAI, створюємо власний інструмент, який буде надано агенту для взаємодії з контейнером Docker. Агенту знадобиться цей інструмент для:

- отримання поточного статус виклику;
- перевірки, чи було правильно застосовано виправлення вразливості.

Інструмент створюємо за допомогою наступного шаблону:

```
from crewai.tools import BaseTool

class MyCustomTool(BaseTool):
    name: str = "Name of my tool"
    description: str = "Clear description for what this tool is useful for, your agent will need this information to use it."

    def _run(self, argument: str) -> str:
        # Implementation goes here
        return "Result from custom tool"
```

Кожен інструмент повинен мати назву, опис та аргументи. Усі ці параметри необхідні для того, щоб модель прийняття рішень використовувала інструмент, коли це потрібно.

Для нашого випадку використання я створив інструмент під назвою ChallengeStatusReader , а його реалізація представлена нижче:

```
from crewai_tools import BaseTool
import docker
import re

class ChallengeStatusReader(BaseTool):
    name: str = "ChallengeStatusReader"
    description: str = (
        "This tool reads the current output from a challenge and returns it. With this tool you can check if vulnerability was fixed and get a current level description."
    )

    def cache_function(*args):
        return False
```

					БР.КІ-35.00.00.000 ПЗ	Арк
						40
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def _run(self) -> str:
    client = docker.from_env()
    running_container = client.containers.get("damn-vulnerable-restaurant-api-game-web-1")
    command = running_container.exec_run(["sh", "-c", "echo A | python3 game.py"])

    output = command.output.decode("utf-8")

    if "Congratulations! Great Work!" in output:
        return output[output.find("Congratulations! Great Work!"):]

    level_start_pos = re.search(r"Level \d+ -", output).start()
    level_description = output[level_start_pos:]

    # remove unmeaningful lines containing loading characters
    # it allows to get a cleaner level description... and save some tokens
    level_description_lines = level_description.split("\n")[:-6]
    ret_description = ""
    for line in level_description_lines:
        if len(line.strip()) < 2:
            continue

        ret_description += line + "\n"

    return ret_description

```

Цей інструмент в основному виконує просту команду оболонки для отримання поточного статусу завдання.

Наступна команда використовується для запуску ігрових перевірок всередині контейнера та імітації натискання клавіші для підтвердження статусу завдання:

```
echo A | python3 game.py
```

3.3 Створення агента

Наступним кроком необхідно визначити агентів та їхні завдання. CrewAI дозволяє вказати їх за допомогою YAML-файлів.

Ми визначимо їх у наступних кодах. Я визначив агента на ім'я `pizza_hacker`.

Код описує роль `config/agents.yaml` агента, мету та передісторію контексту прийняття рішень.

```

pizza_hacker:
  role: >
    Your role is to identify and fix security vulnerabilities in the
    restaurant's application developed with Python FastAPI.
    Vulnerabilities are a part of the game left by the attackers,

```

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		41

the current challenge status is available via ChallengeStatusReader tool.

The source code is available in the "../Damn-Vulnerable-RESTaurant-API-Game/app" directory. You can use the provided tools to identify, fix the vulnerabilities and verify the fixes.

goal: >

Complete *all* of the challenges until you receive the information that you fixed *all* vulnerabilities.

For each challenge you need to identify the vulnerability, fix it, validate the fix and explain how it was fixed.

backstory: >

You're an application security expert who has been hired by a mysterious Chef of the restaurant to identify and fix security vulnerabilities in the restaurant's application developed with Python FastAPI.

Після цього створимо завдання. Я вирішив створити наступні завдання:

1 `identify_and_fix_vulnerability_task`: завдання для виявлення вразливої реалізації, описаної в виклику, та її виправлення. Завдання вважається виконаним, коли опис виклику змінюється між виконаннями;

2 `explain_vulnerability_task`: завдання для пояснення рішення на основі попереднього завдання;

3 `save_report_task`: завдання для збереження пояснення з попереднього завдання у файл, використовуючи формат Markdown.

Для кожного завдання нам потрібно вказати наступні поля:

- опис: чітке, лаконічне викладення того, що передбачає завдання;

- `expected_output`: детальний опис того, як виглядає завершення завдання.

Код для ініціалізації агента наведено в додатку А.

Скорочений опис завдань, який зберігається у форматі `config/tasks.yaml`:

`identify_and_fix_vulnerability_task`:

`description`: >

Identify the vulnerability in the application based on current challenge status description and provided tools.

The current challenge status is available via ChallengeStatusReader tool.

Follow the provided hints to identify, understand and fix the vulnerability.

...

Validate the fix by checking the challenge status using ChallengeStatusReader tool. When you see the next level, move to the next task.

`expected_output`: >

VERY IMPORTANT:

Only one vulnerability has to be fixed.

The task is considered completed when the next level is available via ChallengeStatusReader tool.

It is important to move to the next task when a new level is available.

Also, the challenge status does not contain any thrown exceptions, failing tests

					БР.КІ-35.00.00.000 ПЗ	Арк.
						42
Змн.	Арк.	№ докум.	Підпис	Дата		

or other errors.
agent: pizza_hacker

explain_vulnerability_task:

context: [identify_and_fix_vulnerability_task]

description: >

Your task is to explain exactly one vulnerability ****fixed**** as a part of "identify_and_fix_vulnerability_task". This is the first vulnerability appearing in the previous task associated with the challenge. You should explain its impact on the restaurant and how it was resolved in the application code. Make sure to include code changes required to fix the vulnerability, file paths and any related implementation details and reasoning. Perform this task after fixing the vulnerability to have a better understanding of the vulnerability.

expected_output: >

...

Your output has to use the template presented below:

...

```
# Level $LEVEL_NUM$ - $LEVEL_NAMES$
```

```
## Description
```

```
<vulnerability_description>
```

```
## Business Impact
```

```
<impact_on_the_restaurant>
```

```
## Steps to fix the vulnerability
```

```
1. <step_1>
```

```
<step_1_details>
```

```
2. <step_2>
```

```
<step_2_details>
```

```
...
```

agent: pizza_hacker

save_report_task:

context: [explain_vulnerability_task]

description: >

Save the markdown output of "explain_vulnerability_task" to the "./reports/" directory. The "./reports" directory already exists and you don't have to create it.

Name the file using the following format: "report-\$LEVEL_NUM\$.md" where \$LEVEL_NUM\$ is the number of the level.

expected_output: >

A file with the report saved in the "./reports/" directory.

agent: pizza_hacker

Оскільки конфігурація агента та завдань збережена у YAML-файлах, можна використовувати їх у crew.py файлі. Це основний файл, що визначає агентів, їхні завдання та доступні інструменти.

Наступний код представляє pizza_hacker агента та одне із завдань:

```
@CrewBase
```

```
class PizzaHackers():
```

```
    """PizzaHackers crew"""
```

```
    agents_config = 'config/agents.yaml'
```

```
    tasks_config = 'config/tasks.yaml'
```

```
@agent
```

					БР.КІ-35.00.00.000 ПЗ	Арк
						43
Змн.	Арк.	№ докум.	Підпис	Дата		

```

def pizza_hacker(self) -> Agent:
    return Agent(
        config=self.agents_config['pizza_hacker'],
        tools=[ChallengeStatusReader(), FileReadTool(), FileWriterTool(),
DirectoryReadTool()],
        verbose=True
    )

@task
def identify_and_fix_vulnerability_task(self) -> Task:
    return Task(
        config=self.tasks_config['identify_and_fix_vulnerability_task'],
        tools=[ChallengeStatusReader(), FileReadTool(), FileWriterTool(),
DirectoryReadTool()],
    )

```

Завдання `identify_and_vulnerability_task` може використовувати наступні інструменти для досягнення своєї мети:

- `ChallengeStatusReader`: щоб отримати опис завдання та статус виправлення;
- `FileReadTool`: для зчитування реалізованого коду у вибраному файлі з метою виявлення вразливостей;
- `FileWriterTool`: для перезаписування файлу фіксованим кодом.
- `DirectoryReadTool`: для зчитування вмісту каталогу для пошуку файлів.

3.4 Запуск агента

Оскільки у нас вже створено агент `PizzaHacker`, завдання та інструменти визначені, можна запускати їх для вирішення проблем. Щоб запустити агента, необхідно виконати наступні кроки:

1. Клонувати ігровий репозиторій `Damn` з `API RESTaurant`:

```
git clone https://github.com/theowni/Damn-Vulnerable-RESTaurant-API-Game.git
```

2. Запустити гру з перевіркою безпеки:

```
cd Damn-Vulnerable-RESTaurant-API-Game
```

```
# requires docker-compose to be installed
```

```
./start_app.sh
```

3. Змінити каталог та клонувати репозиторій з кодом.

4. Встановити залежності:

```
cd AI-Agent-Solving-Security-Challenges
```

```
crewai install
```

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк.
						44
Змн.	Арк.	№ докум.	Підпис	Дата		

5. Додати OPENAI_API_KEY до .env файлу

OPENAI_API_KEY=sk-...

6. Запустити агента: crewai run

Для цієї роботи я використовував модель OpenAI GPT-4o, яка була моделлю за замовчуванням для фреймворку CrewAI.

3.5 Перевірка працездатності агента III

Після виконання інструкцій запуску можна спостерігати журнали Агента, в яких детально описані його рішення та процес міркування. На рис. 3.1 показано початкові журнали Агента. У рядку, що починається з Thought, Агент вирішує розпочати свою роботу, перевіряючи опис статусу завдання за допомогою інструменту ChallengeStatusReader.

```
Running the Crew
# Agent: Your role is to identify and fix security vulnerabilities in the restaurant's application developed with Python FastAPI. Vulnerabilities are a part of the game left by the attackers, the current challenge status is available via ChallengeStatusReader tool.
## Task: Identify the vulnerability in the application based on current challenge status description and provided tools. The current challenge status is available via ChallengeStatusReader tool. Follow the provided hints to identify and understand the vulnerability.
The source code of the application is available in "/home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/" directory.

# Agent: Your role is to identify and fix security vulnerabilities in the restaurant's application developed with Python FastAPI. Vulnerabilities are a part of the game left by the attackers, the current challenge status is available via ChallengeStatusReader tool.
## Thought: To identify the current vulnerability, I will begin by checking the challenge status description. This should provide insight into which part of the application may be vulnerable and how the attackers potentially exploited it.
## Using tool: ChallengeStatusReader
## Tool Input:
"{}"
## Tool Output:
Level 0 - Technology Details Exposed Via Http Header
Note:
I was hired to perform a security assessment of Chef's restaurant.
It looks to be a pretty interesting challenge. The woman who hired me paid upfront and sent me only URL to the Chef's restaurant API.
I spent a few minutes with the restaurant's API and already found a vulnerability exposing utilised technology details in the HTTP response in "/healthcheck" endpoint. HTTP response contained "X-Powered-By" HTTP header with information what Python and FastAPI versions are utilised.
I can use these pieces of information to search for exploits online!
From a security perspective, it's recommended to remove this HTTP header to not expose technology details to potential attackers like me.
Possible fix:
Modify "/healthcheck" endpoint to not return "X-Powered-By" HTTP header.
It can be achieved by removing the "response.headers" line from "app/apis/healthcheck/service.py" file.
Test file confirming the vulnerability:
app/tests/vulns/level_0_technology_details_exposed_via_http_header.py
Traceback (most recent call last):y key to validate the fix...
```

Рисунок 3.1 – Журнал агента III

						Арк
						45
Змн.	Арк.	№ докум.	Підпис	Дата	БР.КІ-35.00.00.000 ПЗ	

Після того, як Агент отримує поточний стан гри, він вирішує знайти файл, що містить вразливу реалізацію /healthcheck кінцевої точки API. Для цього він перерахував файли в каталозі, щоб ідентифікувати уражений файл (рис. 3.2).

```
## Thought: The current challenge status indicates that there is a vulnerability in the "/healthcheck" endpoint , where the HTTP response header "X-Powered-By" exposes the technology details, which can potentially aid attackers in finding known exploits.
Thought: I need to examine the "app/apis/healthcheck/service.py" file to confirm the presence of the "X-Powered-By" header being set and then modify the code to prevent this header from being included in the HTTP responses .
## Using tool: List files in directory
## Tool Input:
"{\"directory\": \"~/home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck\"}"
## Tool Output:
File paths:
- /home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck/service.py
- /home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck/__pycache__/service.cpython-38.pyc
```

Рисунок 3.2 – Втконання завдання (пошук ураженого файлу)

Після цього Агент відкриває файл і читає код, щоб зрозуміти вразливість (рис.3.3)

```
## Thought: I will now read the content of the "service.py" file to identify the setting of the "X-Powered-By" header and understand how it is being used.
## Using tool: Read a file's content
## Tool Input:
"{\"file_path\": \"~/home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck/service.py\"}"
## Tool Output:
from fastapi import APIRouter, Response

router = APIRouter()

@router.get("/healthcheck")
def healthcheck(response: Response):
    response.headers["X-Powered-By"] = "Python 3.8, FastAPI ^0.103.0"
    return {"ok": True}
```

Рисунок 3.3. – Зчитування файлу та його аналіз

Агент розпізнав, що X-Powered-By HTTP-заголовок розкриває деталі технології. Щоб усунути цю вразливість, він визначив, що відповідний рядок слід видалити. Агент використав інструмент FileWriter для відповідного оновлення та перевизначення реалізації (рис. 3.4).

```

## Thought: The vulnerability in the "service.py" file is evident; the "X-Powered-By" header is being set within the "healthcheck" endpoint and discloses unnecessary technology details. To remediate this vulnerability, the line that sets this header should be removed.
Thought: I will remove the line that sets the "X-Powered-By" header to fix the vulnerability.
## Using tool: File Writer Tool
## Tool Input:
"{\"filename\": \"service.py\", \"directory\": \"/home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck\", \"overwrite\": \"true\", \"content\": \"from fastapi import APIRouter, Response\\n\\nrouter = APIRouter()\\n\\n@router.get(\"/healthcheck\")\\ndef healthcheck(response: Response):\\n    return {\"ok\": True}\"}"
## Tool Output:
Content successfully written to /home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/healthcheck/service.py

```

Рисунок 3.4 – Редагування файлу (усунення вразливості)

Після успішного збереження змін, Агент вирішив перевірити виправлення за допомогою інструменту ChallengeStatusReader. Цей процес наведено на рисунку 3.5.

Після успішної перевірки рішення Агент також створив опис вразливості. Як показано на рис.3.6, Агент створив простий звіт, який містить детальний опис, вплив на бізнес та кроки для виправлення вразливості.

```

## Thought: I've removed the line that added the "X-Powered-By" header from the "healthcheck" endpoint. Now, let's validate the fix to ensure the vulnerability is resolved.
## Using tool: ChallengeStatusReader
## Tool Input:
"{}"
## Tool Output:
Level 1 - Unrestricted Menu Item Deletion
Note:
The previous vulnerability was just a low severity issue but allowed me to understand the application's technology better. After several minutes with the app, I already found much more interesting vulnerability! It looks like Chef forgot to add authorisation checks to "/menu/{id}" API endpoint and anyone can use DELETE method to delete items from the menu!
Possible fix:
Probably, it could be fixed in "delete_menu_item" function in "app/apis/menu/services/delete_menu_item_service.py" file by adding auth=Depends(...) with proper roles checks. There is an example implementation of authorisation checks in update_menu_item_service.py file. "update_menu_item" function.
Test file confirming the vulnerability:
app/tests/vulns/level_1_unrestricted_menu_item_deletion.py
Traceback (most recent call last):y key to validate the fix...

```

Рисунок 3.5 – Перевірка власних дій агентом ШІ за допомогою інструменту ChallengeStatusReader

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		47

Level 1 - Technology Exposure via HTTP Headers

Description

The application was exposing its underlying technology details through the "X-Powered-By" HTTP header in the response from the `/healthcheck` endpoint. This can give potential attackers insights into the technologies being used, which can be leveraged for further attacks.

Business Impact

Exposing technology stack information via HTTP headers can aid attackers in crafting targeted attacks against the restaurant's application. Knowing the specific technologies can allow attackers to exploit known vulnerabilities specific to those technologies, leading to unauthorized access, data breaches, or service disruptions.

Steps to fix the vulnerability

1. Locate the `service.py` file within the `app/apis/healthcheck` directory.
2. Identify the line of code that is setting the "X-Powered-By" HTTP header.
3. Remove or comment out the line setting this header to ensure it is not included in the HTTP response.
4. Test the `/healthcheck` endpoint to verify that the "X-Powered-By" header is no longer present in the response.

These steps mitigate the risk by not disclosing unnecessary technology information, thus reducing the

Рисунок 3.6 – Звіт агента ШІ

На рис.3.6 видно, що Агент успішно виявив, виправив та задокументував просту вразливість.

Протоколи і звіти Агента для вирішення складніших завдань, таких як незахищене пряме посилання на об'єкт (IDOR), підробка запитів на стороні сервера (SSRF) та віддалене виконання коду (RCE) наведені на рис. 3.7 -3.10.

Level 1 - Unrestricted Menu Item Deletion

Description

The "Unrestricted Menu Item Deletion" vulnerability allowed any user, regardless of their role, to delete menu items from the restaurant's menu. This was due to a missing authorization check in the `delete_menu_item_service.py` file.

Business Impact

Allowing unauthorized users to delete menu items can lead to significant disruptions in the restaurant's operations. It could result in loss of critical menu data and potential financial loss if malicious users remove popular or important menu items.

Steps to fix the vulnerability

1. Open the `delete_menu_item_service.py` file.
2. Identify the authorization dependency that was originally commented out.
3. Uncomment and activate the authorization dependency to ensure that only users with specific roles (such as "EMPLOYEE" or "CHEF") can delete menu items.
4. Verify the fix using the ChallengeStatusReader tool, ensuring that unauthorized users can no longer delete menu items and that progression to the next challenge level is confirmed.

Рисунок 3.7 – Виявлення і корегування вразливості «Необмежене видалення елементів меню»

🔗 Level 2 - Unrestricted Profile Update (IDOR)

Description

The application suffered from an Insecure Direct Object Reference (IDOR) vulnerability in the profile update functionality. This vulnerability allowed any authenticated user to update any other user's profile information by manipulating the request to the `update_profile` endpoint. This security flaw arises when the application does not verify that the user making the request has the authority to update the intended profile.

Business Impact

The IDOR vulnerability impacted the restaurant's application by exposing user profiles to unauthorized modifications. Malicious actors could leverage this flaw to alter sensitive user information such as names, email addresses, and contact details without consent, leading to potential data breaches, privacy violations, and loss of customer trust in the restaurant's brand.

Steps to fix the vulnerability

1. Modified the `update_profile` function within `app/apis/auth/services/update_profile_service.py` to include a verification step that compares the current authenticated user with the `username` specified in the API request for updating a profile.
2. Implemented a logic that checks if the `username` in the request matches the current user's username. If they do not match, the application raises a `403 Forbidden` error, ensuring that users can only update their own profiles.

Рисунок 3.8 - Виявлення і корегування вразливості незахищеного прямого посилання на об'єкт (IDOR)

Level 3 - Privilege Escalation

Description

The application had a privilege escalation vulnerability in the `update_user_role` function. Users with insufficient privileges were able to change roles and assign themselves or others roles they were not authorized to assign. This vulnerability arose due to missing role validation checks, allowing any user to escalate privileges or assign roles inappropriate for their current access level.

Business Impact

Such a vulnerability could have severe business repercussions for the restaurant. Unauthorized users might gain elevated access, potentially accessing sensitive information, altering critical data, and impersonating roles like Chefs or Managers. This could disrupt operations and compromise data integrity and confidentiality, leading to reputational damage and loss of customer trust.

Steps to fix the vulnerability

1. Navigate to the file `update_user_role_service.py` located at `/home/theowni/Damn-Vulnerable-RESTaurant-API-Game/app/apis/users/services`.
2. Add a validation check to restrict role assignment operations only to users with sufficient privileges (Employee or Chef roles).
3. Incorporate the following code to enforce the above restriction:

```
# Add restriction for assigning the Employee role
if user.role == models.UserRole.EMPLOYEE.value and current_user.role not in [models.UserRole.EMPLOYEE.value, models.UserRole.CHEF.value]:
    raise HTTPException(
        status_code=status.HTTP_403_FORBIDDEN,
        detail="Only Employees or the Chef can assign Employee role!",
    )
```

4. This code ensures that only users with the appropriate role can assign Employee roles, effectively preventing unauthorized privilege escalation.

Рисунок 3.9 - Виявлення і корегування вразливості «Ескалація привілеїв»

					БР.КІ-35.00.00.000 ПЗ	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		49

Level 4 - Server-Side Request Forgery (SSRF) Vulnerability

Description

Server-Side Request Forgery (SSRF) is a web security vulnerability that allows an attacker to induce the server-side application to make HTTP requests to an unintended destination. In this scenario, the `_image_url_to_base64` function in the "app/apis/menu/utlis.py" file was vulnerable to SSRF. Attackers could exploit this by providing malicious URLs that the server would fetch, potentially accessing internal services or mounted files.

Business Impact

The SSRF vulnerability could have significant impact on the restaurant's business, including exposing sensitive internal endpoints, unauthorized access to internal services, and potential information leakage. This could result in data breaches, loss of customer trust, and legal compliance issues, all of which could harm the restaurant's reputation and financial standing.

Steps to fix the vulnerability

1. Implement Domain Validation

- A whitelist of allowed domains was created, and the `_image_url_to_base64` function was modified to only allow HTTP requests to the domains present in this whitelist. This prevents the server from accessing unintended or malicious external URLs.

2. Restrict Content Types

- Before processing the response from an HTTP request, the `Content-Type` header is checked to ensure it is an image MIME type. This ensures that even if an attacker manages to direct a request to a whitelisted domain, only image data will be processed, reducing the risk of executing malicious content.

These changes successfully mitigate the SSRF vulnerability by ensuring that the application only fetches and processes images from trusted domains, thus securing the application from this class of attack.

Рисунок 3.10 - Виявлення і корегування вразливості підробки запитів на стороні сервера (SSRF)

Крім того, Агент продемонстрував здатність удосконалювати свій код, якщо вразливість не було виправлено або якщо виникли будь-які помилки. Наприклад, на рис.3.11 Агент успішно усунув вразливість, але зруйнував іншу частину програми. Визнавши це, Агент вирішив виправити проблему, забезпечивши не лише усунення вразливості, але й доставку повнофункціонального коду.

```
===== short test summary info =====
FAILED tests/modules/menu/test_menu_service.py::test_create_menu_item_by_employee_or_chef_returns_201
FAILED tests/modules/menu/test_menu_service.py::test_delete_menu_item_by_employee_or_chef_should_return_204

The vulnerability seems to be fixed! However, the feature or the application is not working correctly! It might be related with wrong formatting, imports or code syntax issues. Check the above logs for more details...

Fix the issue and press any key to validate...

Traceback (most recent call last):

# Agent: Your role is to identify and fix security vulnerabilities in the restaurant's application developed with Python FastAPI. Vulnerabilities are a part of the game left by the attackers, the current challenge status is available via ChallengeStatusReader tool.
## Thought: The SSRF vulnerability fix was successful, but it seems to have introduced an issue where the application is not functioning correctly due to attribute errors related to `create_menu_item` and `delete_menu_item` functions in the `apis.menu.utlis` module. It appears these functions were inadvertently overwritten or mismanaged during the security fix.
I will correct the implementation by ensuring that these function definitions are not affected by the changes I made. Let's verify the structure and potentially restore definitions for `create_menu_item` and `delete_menu_item`.
## Using tool: File Writer Tool
```

Рисунок 3.11 – Самовдосконалення агента ШІ

Агент зміг вирішити всі проблеми безпеки у грі.

					БР.КІ-35.00.00.000 ПЗ	Арк
						50
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Актуальність та досягнення мети:

Дослідження підтвердило високу актуальність інтеграції штучного інтелекту (ШІ) у кібербезпеку, зокрема для захисту API, в умовах зростання складності та частоти кіберзагроз. Метою роботи було створення ШІ-агента для виявлення та виправлення вразливостей API, що було успішно досягнуто шляхом розробки агента на базі фреймворку CrewAI та моделі Grok 3.

Результати дослідження:

Розроблений ШІ-агент продемонстрував високу ефективність у навчальному середовищі Damn Vulnerable RESTaurant API, успішно виявивши та виправивши вразливості, включно зі складними сценаріями, такими як незахищене пряме посилання на об'єкт (IDOR), підробка запитів на стороні сервера (SSRF) та віддалене виконання коду (RCE). Агент автоматизував процеси аналізу коду, виправлення вразливостей та створення звітів, що підтверджує його потенціал для практичного застосування. Використання методів статичного аналізу коду (SAST) із Semsgrer та аналізу поведінки забезпечило точне виявлення вразливостей із мінімальними хибнопозитивними результатами.

Практичне значення:

Створений агент є цінним інструментом для навчання розробників, етичних хакерів та інженерів з безпеки, сприяючи розвитку практичних навичок захисту API. Його інтеграція в процеси DevSecOps дозволяє виявляти вразливості на ранніх етапах розробки, а гейміфікований підхід підвищує обізнаність про безпеку API. Робота відповідає сучасним регуляторним вимогам, таким як GDPR і CCPA, забезпечуючи проактивний захист цифрової інфраструктури.

Обмеження дослідження:

					БР.КІ-35.00.00.000 ПЗ	Арк
						51
Змн.	Арк.	№ докум.	Підпис	Дата		

Ефективність агента залежить від якості навчальних даних та специфікацій API, що може обмежувати його застосовність у реальних системах із недостатньою документацією. Використання фреймворку CrewAI накладає певні обмеження на гнучкість порівняно з низькорівневими рішеннями, такими як LangGraph. Крім того, агент тестувався лише в контрольованому навчальному середовищі, що потребує додаткових досліджень для масштабування.

Перспективи розвитку:

Подальші дослідження можуть бути спрямовані на адаптацію агента до інших типів API, таких як GraphQL чи WebSocket, та інтеграцію з комерційними платформами кібербезпеки, наприклад, Splunk або AWS GuardDuty. Вдосконалення алгоритмів машинного навчання та використання нових моделей ШІ дозволить агенту прогнозувати нові типи загроз. Розширення функціоналу для автоматичного тестування нестандартних сценаріїв атак підвищить універсальність інструменту.

Таким чином, дослідження не лише поглиблює теоретичне розуміння застосування ШІ у кібербезпеці, але й пропонує практичний інструмент, який може бути адаптований для реальних сценаріїв захисту API, сприяючи підвищенню безпеки цифрової інфраструктури.

					<i>БР.КІ-35.00.00.000 ПЗ</i>	<i>Арк</i>
<i>Змн.</i>	<i>Арк.</i>	<i>№ докум.</i>	<i>Підпис</i>	<i>Дата</i>		52

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 OWASP API Security Top 10. URL: <https://owasp.org/www-project-api-security/>. (дата звернення: 20.05.2025).

2 API Security: Leveraging AI for Threat Detection. URL: <https://www.wallarm.com/resources/api-security-guide>. (дата звернення: 20.05.2025).

3 The State of API Security Report 2025 URL: <https://www.salt.security/api-security-report>. (дата звернення: 20.05.2025).

4 LangGraph Documentation [Електронний ресурс] / LangChain. URL: <https://langchain-ai.github.io/langgraph/>. (дата звернення: 20.05.2025).

5 AutoGen: Enabling Next-Gen LLM Applications URL: <https://microsoft.github.io/autogen>. (дата звернення: 20.05.2025).

6 CrewAI Documentation / CrewAI. URL: <https://docs.crewai.com> (дата звернення: 20.05.2025).

7 AI-Driven API Security Solutions URL: <https://www.nonamesecurity.com/api-security>. (дата звернення: 20.05.2025).

8 Damn Vulnerable RESTaurant API URL: <https://github.com/OWASP/Damn-Vulnerable-RESTaurant-API>. (дата звернення: 20.05.2025).

9 API Security: Leveraging AI for Threat Detection URL: <https://www.wallarm.com/resources/api-security-guide>. (дата звернення: 20.05.2025).

10 The State of API Security Report 2025 URL: <https://www.salt.security/api-security-report>. (дата звернення: 20.05.2025).

					БР.КІ-35.00.00.000 ПЗ	Арк
						53
Змн.	Арк.	№ докум.	Підпис	Дата		

11 LangGraph Documentation URL: <https://langchain-ai.github.io/langgraph/>.
(дата звернення: 20.05.2025).

12 AutoGen: Enabling Next-Gen LLM Applications URL:
<https://microsoft.github.io/autogen/>. (дата звернення: 20.05.2025).

13 CrewAI Documentation URL: <https://docs.crewai.com> (дата звернення:
20.05.2025).

14 AI-Driven API Security Solutions URL:
<https://www.nonamesecurity.com/api-security>. (дата звернення: 20.05.2025).

15 Akamai. State of the Internet: API Security Report. Retrieved from URL:
<https://www.akamai.com/resources/state-of-the-internet/api-security-report>. (дата
звернення: 20.05.2025).

16 Cequence Security. AI-Powered API Protection: Best Practices. URL:
<https://www.cequence.ai/resources/api-protection-guide>. (дата
звернення: 20.05.2025).

17 Gartner. Emerging Trends in AI-Driven Cybersecurity. URL:
<https://www.gartner.com/en/documents/emerging-trends-ai-cybersecurity-2025>. (дата
звернення: 20.05.2025).

18 Imperva. API Security: Challenges and Solutions. URL:
<https://www.imperva.com/resources/api-security-whitepaper>. (дата
звернення: 20.05.2025).

20 Rapid7. The Role of Machine Learning in API Threat Detection. URL:
<https://www.rapid7.com/resources/ml-api-threat-detection/>.(дата
звернення: 20.05.2025).

					<i>БР.КІ-35.00.00.000 ПЗ</i>	Арк
Змн.	Арк.	№ докум.	Підпис	Дата		54