МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«____»_____ 2023 р.

На правах рукопису
УДК 004.056.5:510.22(043.3)

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА ВИЩОЇ ОСВІТИ ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Система захисту даних в багатокористувацькій грі

Виконавець: Андрій ЦЮП'ЯШУК
Науковий керівник: к.т.н., доцент Анна ІЛЬЄНКО
Консультант розділу «Охорона навколишнього середовища»: к.т.н., доцент Тетяна ДМИТРУХА
Нормоконтролер: к.т.н., доцент Анна ІЛЬЄНКО

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Магістр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«__» _____ 2023 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи
здобувача вищої освіти Цюп'яшука Андрія Сергійовича

1. Тема: *Система захисту даних в багатокористувацькій грі*
затверджена наказом ректора від «15» вересня 2023 р. № 1814/ст.
2. Термін виконання: з 16.10.2023 р. по 31.12.2023 р.
3. Вихідні дані: проаналізувати існуючі системи та методики забезпечення інформаційної безпеки в багатокористувацьких іграх; на основі аналізу виділити вхідні і вихідні параметри, завдяки яким можливо провести порівняння існуючих систем, виявлення їх переваг і недоліків; розробити методику, алгоритм та програмне забезпечення системи захисту даних.
4. Зміст пояснювальної записки: аналіз існуючих систем та методик захисту даних в багатокористувацьких іграх; розробка методики захисту даних користувачів у багатокористувацькій грі; розробка програмного забезпечення запропонованої системи, верифікація отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ з/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	16.10.2022	*Виконано*
2.	Аналіз літературних джерел	20.10.2022	*Виконано*
3.	Обґрунтування вибору рішення	22.10.2022	*Виконано*
4.	Збір інформації	24.10.2022	*Виконано*
5.	Дослідження сучасних систем і методик захисту даних у багатокористувацьких іграх	27.10.2022	*Виконано*
6.	Розробка методики та структури системи захисту даних користувачів в багатокористувацькій грі	10.11.2022	*Виконано*
7.	Розробка алгоритму та програмного забезпечення системи захисту даних в багатокористувацькій грі	19.11.2022	*Виконано*
8.	Оформлення презентації	16.12.2022	*Виконано*
9.	Отримання рецензій від рецензента	22.12.2022	*Виконано*

6. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона навколишнього середовища	Дмитруха Т.І.		

7. Дата видачі завдання: «16» жовтня 2023 р.

Здобувач вищої освіти Андрій Цюп’яшук

(підпис, дата)

Керівник кваліфікаційної роботи Анна ІЛЬЄНКО

(підпис, дата)

РЕФЕРАТ

Дипломна робота на тему: «Система захисту даних в багатокористувацькій грі» складається зі вступу, основної частини, що містить 4 розділи, 3 висновки до кожного розділу, загального висновку, 2 додатків та списку використаної літератури. Загальний обсяг роботи – 115 сторінок. Робота містить 8 рисунків та 4 таблиці. Список використаних джерел включає 26 джерел.

Метою дипломної роботи є розробка системи захисту даних в багатокористувацькій грі.

У кваліфікаційній роботі розглянуті питання щодо сучасних методів захисту даних користувачів у багатокористувацьких іграх.

Проведені дослідження базуються на сучасних методах побудови захищених інформаційних мереж, методах шифрування і зберігання даних користувачів.

Реалізація власного методу за допомогою різних видів шифрування та встановлення анти-чіт логіки дозволяє показати адекватність роботи системи і доцільність її використання у багатокористувацьких іграх.

Запропоновані методи дозволяють забезпечити надійність даних користувача та захист мережевої передачі даних.

Ключові слова: МЕРЕЖА, ШИФРУВАННЯ, БАГАТОКОРИСТУВАЦЬКА ГРА, АНТИ-ЧІТ

З М І С Т

ВСТУП

Актуальність теми. Сучасні комп'ютерні технології та інтернет-ігри стали невід’ємною частиною життя сучасного суспільства, забезпечуючи розвагу, спілкування та важливий засіб відпочинку для мільйонів користувачів по всьому світу. Однак із зростанням популярності інтернет-ігор виникає дедалі більше викликів, пов'язаних із захистом особистих даних гравців. Системи захисту даних у багатокористувацьких іграх стають важливим аспектом їх розробки та експлуатації.

Сучасні механізми захисту даних стали об'єктом постійних досліджень та розробки, спрямованих на забезпечення конфіденційності, цілісності та доступності особистих даних користувачів. Ігрова індустрія зростає з неймовірною швидкістю, і з нею зростають і вимоги до захисту інформації, яку користувачі надають під час гри. Захист особистих даних стає завданням, яке вимагає від розробників інноваційних рішень та постійного вдосконалення технологічних підходів.

Мета роботи - розробка системи захисту даних в багатокористувацькій грі.

Об’єкт дослідження – процедура шифрування внутрішньо-ігрових даних та даних при їх передачі на сервер, процес моніторингу і скринінгу даних

Предмет дослідження – методи шифрування, автентифікації та авторизації.

Методи. Проведені дослідження базуються на сучасних методах побудови захищених інформаційних мереж, методах застосування шифрування, методи і підходи спостереження та логування атак.

Виходячи з мети, завданням даної дипломної роботи є:

- провести дослідження сучасних рішень щодо захисту даних в багатокористувацькій грі;
- розробити систему захисту даних в багатокористувацькій грі;
- провести тестування системи та оцінку ефективності реалізованої системи захисту даних в багатокористувацькій грі.

Наукова новизна. Розроблено систему захисту даних у багатокористувацькій грі, за рахунок використання підходів щодо шифрування

внутрішньо-ігрових даних і даних при передачі на сервер в поєднанні з системою спостереження та логування атак, що дозволило покращити базові системи збереження даних користувача, пришвидшити збереження в порівнянні до готових рішень, а також створити розширювану систему, яка адаптується під потреби розробника.

Практична цінність роботи полягає у реалізації системи захисту даних у багатокористувацькій грі з використанням мови програмування C#, графічного редактора Unity, що робить її мультиплатформенною та здатною захищати мережеві даних і впровадити захист від зовнішніх ін’єкцій.

Апробація результатів. Цюп'яшук А.С. АНАЛІЗ СУЧАСНИХ МЕХАНІЗМІВ ЗАХИСТУ ДАНИХ В БАГАТОКОРИСТУВАЦЬКІЙ ГРІ // Modern problems of science, education and society: VIII Міжнародна науково-практична конференція, 9-11 жовтня 2023 р.: тези доповіді. – К., 2023. – С.354-357.

РОЗДІЛ 1
СУЧАСНІ МЕХАНІЗМИ ЗАХИСТУ ДАНИХ

1.1. Сучасний стан забезпечення безпеки інформації

У сучасному цифровому світі інформація є важливим ресурсом, який потрібно захищати від різних загроз і загроз для її конфіденційності, цілісності та доступності. Цей розділ дипломної роботи розглядає сучасний стан забезпечення безпеки інформації, включаючи ключові аспекти, тенденції та виклики, що виникають у сфері інформаційної безпеки.

Ключові поняття

Перш ніж розглядати сучасний стан безпеки інформації, слід визначити деякі ключові поняття:

Інформаційна безпека - це процес захисту інформації від несанкціонованого доступу, модифікації або втрати для забезпечення її конфіденційності, цілісності та доступності.

Загрози - це потенційні події або дії, які можуть призвести до порушення безпеки інформації. Загрози можуть бути технічними, природними або людськими.

Вразливості - це слабкі місця або дефекти в інформаційних системах, які можуть бути використані зловмисниками для виконання атак та порушень безпеки.

Заходи забезпечення - це технічні та організаційні заходи, які призначені для запобігання загрозам і використовуються для забезпечення безпеки інформації.

Стан сучасної інформаційної безпеки

У сучасному світі спостерігається стійке зростання кількості та складності інформаційних загроз. Це включає в себе кібератаки, віруси, розповсюдження шкідливих програм та інші форми кіберзлочинності. Загрози можуть бути

спрямовані на корпорації, урядові установи, громадян і навіть національну безпеку.

За останні десятиліття обсяг і важливість даних значно зросли. Велика кількість конфіденційної інформації, особистих даних і комерційної інформації зберігається в цифровому форматі, що робить їх привабливими цілями для зловмисників.

Загальні тренди у забезпеченні безпеки інформації надзвичайно важливі, оскільки вони визначають динаміку та напрямок розвитку цієї сфери. Нижче розглянемо деякі більш докладно:

Зростання кількості та складності загроз

Зараз ми спостерігаємо зростання якісної та кількісної складності загроз інформаційній безпеці. Це включає в себе такі аспекти:

- Розширеність кіберзлочинності: Зловмисники стають більш витонченими та ресурсозатратними у своїх атаках, використовуючи різноманітні технології, такі як шифрування, анонімність та вимагаючи великі викупи від жертв.
- Активність державних акторів: Деякі держави займаються кібервійськами діяльностями, які можуть містити кібершпигунство, кіберсаботаж і витоки конфіденційної інформації.
- Розширення атак на підприємства та критичну інфраструктуру: Великі компанії, уряди та критичні системи, такі як енергетика та транспорт, стають мішенню для атак, що може призвести до серйозних наслідків.

Збільшення обсягу даних

В цифрову епоху обсяги даних швидко зростають. Це створює додаткові виклики у забезпеченні безпеки:

- Збільшення пов'язаних з даними ризиків: Захист великих обсягів даних вимагає більшої уваги до зберігання, передачі і обробки інформації.
- Особисті дані та конфіденційність: Захист особистих даних користувачів є пріоритетом для багатьох організацій через зростання суворих регуляторних вимог.

- Захист від витоків даних: Витоки даних можуть завдати непоправної шкоди репутації інституції та спричинити фінансові втрати.

Регулювання та вимоги щодо конфіденційності даних

- Загальний регламент з охорони даних (GDPR): Європейський Союз встановив суворі стандарти щодо захисту особистих даних, накладаючи на компанії вимоги збереження конфіденційності інформації.
- Країнозалежне регулювання: Різні країни встановлюють власні правила та вимоги щодо інформаційної безпеки, що ускладнює міжнародні операції.
- Підвищення вимог щодо звітності: Законодавці вимагають від організацій більшої прозорості та звітності щодо інцидентів інформаційної безпеки.

Загальні тренди у забезпеченні безпеки інформації створюють складну і динамічну обстановку, в якій організації та фахівці з інформаційної безпеки повинні залишатися завжди на висоті. Розуміння цих трендів допомагає розробити більш ефективні стратегії та заходи для захисту інформації в сучасному світі цифрових технологій.

Незважаючи на досягнення у сфері інформаційної безпеки, існують ряд викликів і перспектив, які потребують уваги:

- Адаптація до нових загроз: Зловмисники постійно розвивають нові методи атак, тому організаціям потрібно постійно адаптуватися та підтримувати свої заходи безпеки на актуальному рівні.
- Збільшення обсягу та складності даних: Зростаючий обсяг та складність даних створюють виклики для їх безпеки та захисту від втрати чи несанкціонованого доступу.
- Глобальний характер загроз: Кіберзагрози мають глобальний характер, і їх вплив може розповсюджуватися на всі частини світу. Співпраця між країнами та галузями є важливою для забезпечення безпеки інформації.

Розвиток технологій та інновації

1. Штучний інтелект (ШІ): ШІ відіграє значущу роль у забезпеченні безпеки інформації, аналізуючи великі обсяги даних для виявлення аномалій та

потенційних загроз. Автоматизовані системи на основі ШІ можуть швидко реагувати на кібератаки та мінімізувати ризики.

2. Квантові обчислення: Розвиток квантових обчислень може створити нові можливості для криптографії та шифрування, одночасно вносячи нові виклики у забезпеченні безпеки. Розробка квантово-стійких методів стає важливим завданням.
3. Блокчейн-технології: Блокчейн може використовуватися для забезпечення безпеки даних та автентифікації користувачів. Він створює надійні та необхідні умови для зберігання даних, які не піддаються модифікації.
4. Інформаційна безпека в окремих секторах
5. Корпоративна інформаційна безпека: Компанії і підприємства повинні розвивати інтегровані програми забезпечення безпеки для захисту корпоративних даних та інтелектуальної власності.
6. Сфера охорони здоров'я: Зростаюча кількість медичних даних, що зберігаються в цифровому форматі, потребує посилення заходів безпеки для захисту конфіденційності пацієнтів.
7. Фінансовий сектор: Банки та фінансові установи вкладають значні зусилля у забезпечення безпеки інформації, оскільки вони стають головними цілями для кіберзлочинців.

Сучасний стан забезпечення безпеки інформації вимагає постійного вдосконалення та адаптації до зростаючих загроз і вимог. Забезпечення конфіденційності, цілісності та доступності інформації залишається важливим завданням для організацій і суспільства загалом. Сучасні технології, такі як ШІ, блокчейн та квантові обчислення, відкривають нові можливості для забезпечення безпеки, але вони також вимагають постійного вдосконалення та адаптації. Співпраця між галузями та країнами, регулювання та освіта грають важливу роль у забезпеченні інформаційної безпеки. Навчання та підвищення обізнаності користувачів також є важливими чинниками у забезпеченні безпеки інформації в сучасному світі.

1.2. Класифікація загроз та вразливостей інформації

У світі, де інформація стала ключовим ресурсом, захист інформаційної безпеки стає надзвичайно важливим завданням для організацій та індивідів. Для ефективного управління інформаційною безпекою важливо спершу розробити систематичний підхід до класифікації загроз та вразливостей. У цьому розділі дипломної роботи ми розглянемо різні аспекти класифікації загроз та вразливостей інформації, використовуючи приклади для кращого розуміння.

Класифікація загроз інформації

1. За походженням:

- Зовнішні загрози: Це загрози, які виникають ззовні інформаційної системи. Наприклад, хакерські атаки, віруси, шкідливі програми. Приклад: розповсюдження шкідливого ПЗ через електронну пошту.
- Внутрішні загрози: Це загрози, які виникають в середовищі організації або від її співробітників. Наприклад, недбале ставлення до безпеки, недостатні знання персоналу. Приклад: витік конфіденційної інформації співробітником.

2. За характером атак:

- Активні атаки: Це атаки, при яких загроза спрямована на систему з метою злому, втрати даних або завдання шкоди. Приклад: SQL-ін'єкція для вилучення даних з бази даних.
- Пасивні атаки: У таких атаках зловмисники намагаються отримати доступ до інформації, не руйнуючи систему, наприклад, перехоплення пакетів у мережі. Приклад: аналіз мережевого трафіку для отримання конфіденційних даних.

3. За метою атаки:

- Кіберзлочинні атаки: Основною метою таких атак є фінансовий виграш, крадіжка інформації або інші кримінальні дії. Приклад: вимагання викупу

після шифрування файлів шкідливим програмним забезпеченням (рансомвар).

- Кібершпигунство: Тут атаки спрямовані на отримання конфіденційної інформації для політичних, військових або комерційних цілей. Приклад: кібершпигунство для отримання важливих військових секретів.

Класифікація вразливостей інформаційних систем

1. За джерелом вразливостей:

- Технічні вразливості: Пов'язані з програмними помилками, слабкими налаштуваннями систем, а також іншими технічними аспектами. Приклад: вразливість у веб-додатку, яка дозволяє виконати незаконний SQL-запит.
- Людські вразливості: Вразливості, які виникають через недбале ставлення або недостатню освіченість персоналу. Приклад: викладення паролів на видному місці.

2. За ступенем важливості:

- Критичні вразливості: Такі вразливості мають великий потенціал для завдання шкоди інформаційній системі та організації. Приклад: вразливість, яка дозволяє зловмисникові отримати повний доступ до сервера.
- Некритичні вразливості: Вразливості, які менш важливі або не мають великого потенціалу для завдання серйозної шкоди. Приклад: вразливість, яка дозволяє вимкнути малозначну функціональність системи.

3. За можливістю використання:

- Активні вразливості: Вразливості, які можна легко використовувати для атаки. Приклад: вразливість, яка дозволяє виконати віддалену команду на сервері.
- Пасивні вразливості: Вразливості, які не завжди можна ефективно використовувати для атаки, але вони можуть стати джерелом ризику в майбутньому. Приклад: відкритий доступ до певних портів на мережевому обладнанні.

Класифікація загроз та вразливостей інформації є важливим інструментом для розуміння та управління ризиками інформаційної безпеки. Зазначені

класифікаційні аспекти допомагають організаціям розробляти більш ефективні стратегії та заходи забезпечення безпеки для захисту інформації в умовах зростаючого обсягу кіберзагроз та вразливостей. Практичні приклади підкреслюють актуальність цих класифікаційних критеріїв у сучасному світі інформаційних технологій.

Вразливості і загрози в багатокористувацьких іграх

Вразливості і загрози для багатокористувацьких ігор можуть мати серйозний вплив на ігровий досвід гравців та репутацію гри. Ось деякі приклади таких вразливостей і загроз:

1. Вразливості гри:

- Вразливості в мережевій архітектурі: Недостатня захищеність мережевого стеку гри може призвести до маніпуляцій, таких як DDoS-атаки (розподілений відмова в обслуговуванні) або перехоплення пакетів даних, що може призвести до обману гравців або збоїв у грі.
- Вразливості безпеки клієнтської сторони: Вразливість в клієнтській програмі гри може бути використана для читерства або внесення змін у гру, які призводять до недопущених переваг або спотворень.
- Застосунки третіх сторін: Використання додатків або модифікацій, розроблених третіми сторонами, може створити вразливості або надати несанкціоновані переваги гравцям.

2. Загрози для інформаційної безпеки:

- Маніпуляція геймплеєм: Зловмисники можуть змінювати параметри гри, щоб отримати переваги або завдати шкоду іншим гравцям. Наприклад, швидше ніж зазвичай переміщення персонажа або нескінченна кількість ресурсів.
- Читерство та хакінг: Гравці можуть використовувати шахрайські програми, щоб обходити правила гри та отримувати переваги. Наприклад, використання "aimbot" у шутерах, щоб автоматично наводити мету на противників.

- Фішинг і обман: Зловмисники можуть створювати фейкові веб-сайти або обманювати гравців, щоб отримати їхні облікові дані або особисту інформацію.
- Віруси та шкідливе програмне забезпечення: Завантаження нелегальних копій гри або модифікацій може призвести до зараження комп'ютера вірусами або шкідливим програмним забезпеченням.

3. Антиборотьба та захист:

- Античит системи: Гри можуть використовувати античит системи для виявлення та блокування читерів та недопущення неправомірних дій гравців.
- Шифрування мережевого трафіку: Захищені мережеві протоколи та шифрування можуть допомогти у запобіганні перехопленню чутливої інформації.
- Постійне оновлення: Регулярні оновлення гри та виявлення та виправлення вразливостей можуть покращити безпеку гри та зменшити кількість атак.

Узагальнюючи, ігри, особливо багатокористувацькі, стикаються з багатьма загрозами та вразливостями, які можуть вплинути на ігровий досвід гравців. Розуміння цих загроз та вразливостей та вжиття відповідних заходів забезпечення безпеки є важливим для збереження цілісності гри та задоволення гравців.

1.3. Огляд сучасних рішень щодо захисту даних в багатокористувацькій грі

Більшість великих ігрових гігантів мають свої власні системи захисту даних в іграх. Кожна з компаній розроблювала такі методи спеціально під свої ігрові рушії і свої механіки в іграх. Здається, що такі системи мають бути дійсно хорошими, в них все ж є велика кількість недоліків. В цьому розділі буде

розглянуто основні варіанти сучасних рішень щодо захисту даних в багатокористувацьких іграх.

Valve Anti-Cheat

Valve Anti-Cheat (VAC) - це система античиту, розроблена компанією Valve Corporation для виявлення і вилучення читерів і шахраїв у багатьох іграх, які використовують платформу Steam. ВАС є важливим інструментом для забезпечення чесної гри і підтримки інтегритету геймплею в іграх, таких як Counter-Strike: Global Offensive (CS:GO), Dota 2 та інші.

Основні принципи роботи Valve Anti-Cheat:

- Система евристичного аналізу: VAC використовує алгоритми аналізу та визначення нечесних дій, що можуть включати в себе використання читів, хаків, ботів і іншого шахрайського програмного забезпечення. Вона шукає підозрілі дії та зміни в грі, які не мають законних пояснень.
- Порівняння з відомими читами: VAC володіє базою даних з відомими шахрайськими програмами і читами. Під час гри він може порівнювати активні процеси та файли гравця з цими базами даних, щоб виявити співпадіння.
- Затримка бану: ВАС може затримувати бани на деякий час, перед тим як вони будуть негайно введені. Це може сприяти виявленню більше читерів, оскільки вони можуть продовжувати грати в гру, не знаючи, що їх дії зафіксовані.
- Регулярні оновлення та аналіз даних: VAC постійно оновлюється, і вся інформація про гравців, яка використовується для визначення шахрайської активності, відсилається до серверів Valve для аналізу. Це дозволяє постійно покращувати систему.
- Безпека як джерело незалежності: VAC намагається виявляти програмне забезпечення, яке може змінювати гру або шкодити іншим гравцям. Важливо відзначити, що система не перевіряє інші програми, що запущені на комп'ютері гравця, за винятком тих, які прямо взаємодіють з грою.

Якщо гравець використовує шахрайське програмне забезпечення або чити, їх акаунт може бути позначено як "VAC-banned," і вони можуть бути виключені з онлайн-гри впродовж тривалого періоду або назавжди, залежно від серйозності порушень.

Переваги Valve Anti-Cheat (VAC):

- Ефективність: VAC виявляє багато типів шахрайського програмного забезпечення та читів, завдяки чому воно дуже ефективне у виявленні та вилученні читерів.
- Автоматизація: VAC працює в автоматичному режимі, а це означає, що велика кількість читерів може бути виявлена і покарана без активної участі адміністраторів.
- Затримка банів: VAC може затримувати бани на деякий час, перед тим як вони будуть негайно введені. Це дозволяє збирати більше інформації про читерів та відслідковувати їхню активність.
- Стабільність ігрових серверів: Боротьба з читерством допомагає зберігати стабільність інфраструктури гри, що поліпшує геймплей для чесних гравців.

Недоліки Valve Anti-Cheat (VAC):

- Затримка оновлень: Поряд з його перевагами, затримка в впровадженні банів може дозволити деяким читерам продовжувати грати безкарно протягом певного часу.
- Вразливості системи: Як будь-яка програма, VAC може мати вразливості. Якщо зловмисники знайдуть спосіб обхід VAC або використовувати недоліки, це може створити проблеми для чесних гравців.
- Відсутність прозорості: Valve не розголошує подробиці щодо того, як саме працює VAC або які конкретно дії ведуть до бану. Це може викликати сумніви іноді, і гравці можуть скаржитися на некоректне вилучення банів.
- Ложно-позитивні випадки: Інколи VAC може помилково визначати гравців як читерів і вводити бани невинним людям, що призводить до негативних відгуків та обурення гравців.

У цілому, Valve Anti-Cheat є важливою системою для забезпечення інтегрітету геймплею, проте він не є ідеальним і має свої переваги та недоліки. Valve постійно працює над покращеннями системи, але боротьба з читерством залишається складною задачею в ігровому світі. [1]

Riot Vanguard

Riot Vanguard - це античит-система, розроблена компанією Riot Games для гри Valorant і інших ігор цієї компанії. Ця система призначена для виявлення та блокування читерів та шахраїв, які можуть спотворювати інтегрітет гри і створювати некоректний геймплей для інших гравців. Ось як працює Riot Vanguard:

- Рівень доступу як системний рівень: Riot Vanguard працює на рівні ядра операційної системи (керівництво Kernel) і запускається разом з системою при завантаженні комп'ютера. Це дає йому високий рівень доступу до комп'ютера та дозволяє виявляти некоректну активність на рівні операційної системи.
- Спостереження за процесами: Vanguard перевіряє активні процеси на комп'ютері гравця і аналізує їх діяльність, щоб виявити можливу шахрайську або відомо недозволену діяльність, таку як спроби модифікувати гру або втручання в процеси гри.
- Підписи і визначення відомих читів: Vanguard користується базою даних з відомими читами і шахрайськими програмами. Він порівнює активні процеси і файли гравця з цими підписами, щоб виявити відповідність і вжити відповідних заходів.
- Детекція заборонених програм: Vanguard може виявляти наявність і активність програм, які вважаються забороненими або потенційно шахрайськими, такі як чити, спеціальні макроси або програми для злому.
- Шифрування трафіку: У деяких іграх, як Valorant, Riot Vanguard також може відслідковувати трафік мережі, щоб виявити можливі ознаки шахрайської діяльності або спроб обходу античит-системи.

- Постійні оновлення і моніторинг: Riot постійно оновлює Vanguard, додаючи нові функції та вдосконалюючи систему для боротьби з новими видами читів і шахрайського програмного забезпечення.

Важливо відзначити, що Riot Vanguard запускається тільки під час гри в ігри компанії Riot, і він припиняє роботу, коли гра закривається. Такий підхід допомагає забезпечити безпеку та конфіденційність гравців поза грою.

Переваги Riot Vanguard:

- Ефективність в боротьбі з читерством: Vanguard є високоефективною системою для виявлення та блокування читерів та інших видів шахрайської активності. Вона допомагає зберегти чесний геймплей і покращити ігровий досвід для чесних гравців.
- Системний рівень доступу: Vanguard запускається на рівні ядра операційної системи, що надає йому високий рівень доступу та забезпечує більш ефективну роботу для виявлення шахрайських програм і читів.
- Активні оновлення і моніторинг: Riot постійно оновлює Vanguard і вносить в нього нові функції та покращення, щоб боротися зі змінами у сфері читерства та шахрайства.
- Система реагування на швидкі зміни: Vanguard може швидко реагувати на нові види читів і шахрайського програмного забезпечення, що дозволяє компанії Riot Games вживати відповідних заходів.

Недоліки Riot Vanguard:

- Спротив та обурення гравців: Введення античит-системи на рівні ядра операційної системи спричинює обурення та опор приватності з боку деяких гравців. Вони вважають, що такий високий рівень доступу може порушити їхні права.
- Можливість помилкових вилучень: Іноді Vanguard може помилково визначати програми або процеси як шахрайські, що може призвести до недоречних банів для чесних гравців.

- Недоступність на інших операційних системах: Riot Vanguard підтримує лише Windows, що виключає користувачів інших операційних систем, таких як macOS або Linux, від гри в Valorant.
- Можливість обхіду системи: Зловмисники постійно намагаються знайти способи обхіду системи, і іноді це може бути успішним. Riot повинен постійно реагувати на нові методи обходу.

У цілому, Riot Vanguard є ефективною системою для боротьби з читерством у грі Valorant, але вона має свої недоліки та обмеження, які потрібно враховувати при її використанні. [2]

Easy Anti-Cheat

Easy Anti-Cheat (EAC) - це античит-система, яка використовується в багатьох онлайн-іграх для виявлення та блокування шахрайської активності, такої як використання читів та іншого недозволеного програмного забезпечення. Вона має свою особливу архітектуру та принципи роботи:

- Перевірка гри в режимі реального часу: Easy Anti-Cheat постійно моніторить гру в режимі реального часу, спостерігаючи за діями гравців і перевіряючи їх на предмет можливого шахрайства.
- Сканування пам'яті: EAC сканує оперативну пам'ять комп'ютера гравця, щоб виявити відомі чити, шахрайські програми або модифікації гри. Якщо відбувається втручання в пам'ять гри, це може бути виявлено та заблоковано.
- Перевірка файлів гри: EAC перевіряє цілісність файлів гри, щоб виявити будь-які зміни, які можуть бути здійснені читами або шахрайськими програмами. Якщо файли гри були змінені або пошкоджені, це може бути сприйнято як підозріле діяння.
- Перевірка системи: EAC перевіряє комп'ютер гравця на предмет встановленого шахрайського програмного забезпечення, небажаних додатків або процесів, які можуть впливати на гру.
- Взаємодія з серверами гри: EAC взаємодіє з серверами гри, обмінюючись інформацією про виявлені шахрайські дії. Якщо гравець виявляється

використовувати недозволені програми або чити, ця інформація може бути надіслана на сервери гри для подальших дій.

- Заборона і вилучення читерів: Якщо EAC виявляє шахрайську активність або недозволене програмне забезпечення, воно може призвести до заборони аккаунта гравця або навіть до вилучення гри з комп'ютера.

Easy Anti-Cheat є популярною античит-системою, яка використовується в багатьох іграх для забезпечення чесної гри та захисту від читерства. Вона поєднує в собі різні методи виявлення шахрайської активності і стежить за гравцями під час гри, щоб запобігти нечесному геймплею.

Переваги Easy Anti-Cheat (EAC):

- Ефективність виявлення читерства: EAC виявляє багато видів шахрайської активності, такі як використання читів, шахрайських програм і модифікацій гри. Це допомагає зберегти інтегрітет геймплею і забезпечити чесність гри.
- Системна незалежність: EAC працює на рівні програмного коду, що дозволяє йому бути незалежним від конкретної гри або платформи. Він може бути використаний в різних іграх та на різних операційних системах.
- Регулярні оновлення: Розробники EAC постійно вносять оновлення і покращення, щоб боротися зі швидко змінюючимся ландшафтом читерства. Це допомагає системі залишатися ефективною впродовж часу.
- Мінімальний вплив на продуктивність: EAC спроектований так, щоб мінімізувати вплив на продуктивність гри і комп'ютера гравця. Він працює в фоновому режимі та споживає мінімальні ресурси.

Недоліки Easy Anti-Cheat (EAC):

- Ложно-позитивні випадки: Інколи EAC може помилково виявляти гравців як читерів або надавати фальшиві позитиви. Це може призвести до незаслужених заборон та негативного впливу на гравців.
- Недоступність для ігор на інших платформах: EAC може бути обмежено на певних платформах або для конкретних типів ігор. Це може обмежувати використання системи для розробників ігор на інших платформах або для ігор, які вимагають специфічних рішень з безпеки.

- Спротив приватності: Як і в інших античит-системах, EAC збирає інформацію про гравців та їх комп'ютери для виявлення шахрайської активності. Деякі гравці можуть вважати це порушенням приватності.
- Завжди існує ризик обходу: Зловмисники постійно шукають способи обійти EAC та інші античит-системи, і іноді це може бути успішним.

У цілому, Easy Anti-Cheat є важливою системою для боротьби з читерством у багатьох іграх, проте вона має свої переваги і недоліки, які розробники і гравці повинні брати до уваги при її використанні. [3]

BattlEye

BattlEye - це античит-система, яка використовується в багатьох онлайн-іграх для виявлення та блокування шахрайської активності, такої як використання читів та іншого недозволеного програмного забезпечення. Вона працює на основі ряду технік і має наступні основні принципи роботи:

- Моніторинг процесів гри: BattlEye постійно моніторить активні процеси гри, які виконуються на комп'ютері гравця під час гри. Вона відстежує дії гравців та перевіряє їх на предмет незвичайної або підозрілої активності.
- Перевірка пам'яті і файлів: BattlEye сканує оперативну пам'ять комп'ютера та файли гри для виявлення можливого шахрайського програмного забезпечення або модифікацій. Вона може перевіряти цілісність гри та виявляти зміни, які можуть бути результатом недозволеної активності.
- Аналіз зловмисного програмного забезпечення: BattlEye має базу даних з відомими читами та шахрайськими програмами. Вона порівнює активні процеси та файли гравця з цими відомими підписами, щоб виявити можливі співпадіння.
- Прослуховування мережевого трафіку: BattlEye може слідкувати за мережевим трафіком гравця, щоб виявити можливі ознаки шахрайської діяльності або спроб обходу античит-системи.
- Заборона і вилучення читерів: Якщо BattlEye виявляє недозволену активність або шахрайське програмне забезпечення, вона може призвести до заборони аккаунта гравця або навіть до вилучення гри з комп'ютера.

BattlEye працює в реальному часі під час гри і використовує різні техніки для виявлення інформації, яка може свідчити про незаслужену вигоду чи недозволену активність гравця. Ця система включає в себе захисні заходи для забезпечення інтегрітету гри і заборони читерства.

Зловмисники постійно шукають способи обійти BattlEye і інші античит-системи, і розробники системи постійно оновлюють її для боротьби зі змінами у сфері читерства та шахрайства.

Переваги BattlEye:

- Ефективність виявлення читерства: BattlEye відома своєю високою ефективністю виявлення і блокування шахрайської активності та використання читів. Вона допомагає забезпечити чесний геймплей для всіх гравців.
- Запобігання модифікаціям гри: BattlEye допомагає запобігти модифікаціям гри, які можуть негативно впливати на ігровий процес і давати перевагу деяким гравцям.
- Стабільність гри: Боротьба з читерством за допомогою BattlEye сприяє підтримці стабільності гри, оскільки вона зменшує можливість зловмисного втручання в геймплей і мережеву активність.
- Оновлення та підтримка: BattlEye постійно оновлюється і підтримується, щоб залишатися ефективною у боротьбі зі змінами у сфері читерства і шахрайства.

Недоліки BattlEye:

- Ложно-позитивні випадки: Інколи BattlEye може помилково визначати гравців як читерів або вносити надлишкові обмеження, що може викликати негативний вплив на чесних гравців.
- Обмежена доступність: BattlEye може бути обмежено на певних платформах або для конкретних типів ігор, що обмежує можливість його використання для розробників ігор на інших платформах.

- Завжди існує ризик обходу: Зловмисники постійно шукають способи обійти BattlEye і інші античит-системи, і іноді це може бути успішним.
- Вплив на продуктивність: Деякі гравці відзначають, що BattlEye може впливати на продуктивність комп'ютера та гри, особливо під час завантаження.

У цілому, BattlEye є важливою системою для боротьби з читерством та підтримки чесного геймплею в онлайн-іграх. Проте вона також має свої недоліки та обмеження, які розробники і гравці повинні враховувати при використанні. [4]

Anti-Cheat Toolkit

Anti-Cheat Toolkit - це набір інструментів та бібліотек для розробників ігор, які допомагають створювати власні системи захисту від читерства і недозволеної активності в онлайн-іграх. Anti-Cheat Toolkit не є окремою античит-системою, але надає інструменти для створення власних методів виявлення та блокування читерства. Цей набір інструментів може містити різноманітні функції, проте основні принципи його роботи включають наступне:

- Моніторинг гри в реальному часі: Anti-Cheat Toolkit відстежує активний геймплей гравців в реальному часі. Це включає в себе спостереження за діями гравців, взаємодією з грою та обміном даними між клієнтом і сервером гри.
- Перевірка пам'яті і файлів: Набір інструментів може сканувати оперативну пам'ять і файли гри на предмет можливих змін, модифікацій або незвичайних дій. Це допомагає виявляти читерство, яке може впливати на гру.
- Аналіз графічних параметрів і інших даних: Anti-Cheat Toolkit може аналізувати графічні налаштування та інші параметри гри для виявлення незвичайних або недозволених змін, які можуть вказувати на читерство.
- Перевірка мережевої активності: Він може аналізувати мережеву активність гравців і перевіряти її на предмет аномальностей або незвичайної поведінки, яка може свідчити про шахрайську активність.

- Захист від модифікацій клієнта: Anti-Cheat Toolkit допомагає захищати клієнт гравця від недозволених модифікацій, що можуть бути використані для отримання переваги.
- Ведення журналу та звітність: Набір інструментів може створювати журнали та звіти про підозрілу активність, які можуть бути використані для аналізу та подальших заходів.

Anti-Cheat Toolkit надає розробникам ігор можливість налаштувати та інтегрувати власну систему захисту від читерства, враховуючи особливості своєї гри. Вона дозволяє боротися з різними видами шахрайської активності та виявляти чити та інші недозволені програми, що можуть впливати на ігровий процес.

Переваги Anti-Cheat Toolkit:

- Гнучкість і налаштовуваність: Anti-Cheat Toolkit надає розробникам ігор можливість створити систему захисту, яка відповідає особливостям їхньої гри. Вони можуть налаштовувати правила та алгоритми виявлення читерства під свої потреби.
- Контроль над безпекою гри: Розробники можуть відстежувати та контролювати безпеку своєї гри, не покладаючися на зовнішні античит-системи. Це дозволяє їм активно реагувати на нові загрози та шахрайські методи.
- Система захисту під власним брендом: Розробники можуть створити систему захисту з власним брендом, що може підсилити довіру гравців та покращити імідж гри.
- Можливість інтеграції з іншими інструментами: Anti-Cheat Toolkit може бути інтегрована з іншими інструментами розробки гри для покращення загального процесу розробки та захисту.

Недоліки Anti-Cheat Toolkit:

- Час і ресурси розробки: Створення власної системи захисту від читерства вимагає значних зусиль і ресурсів. Це може бути дорогоцінним та часоємним завданням.
- Потребує постійного оновлення: Загрози читерства постійно змінюються, і Anti-Cheat Toolkit потребує постійного оновлення та адаптації для боротьби з новими методами читерства.
- Ризик ложно-позитивів і відмов: Неправильна настройка або реалізація системи може призводити до ложнопозитивних випадків, коли чесні гравці помилково отримують заборони або обмеження.
- Завантаження на сервери гри: Використання Anti-Cheat Toolkit може збільшувати завантаження на сервери гри та зменшувати продуктивність гри.

Загалом, Anti-Cheat Toolkit надає розробникам гнучкість і контроль над захистом від читерства в їхній грі, але вона також вимагає великих зусиль і ресурсів для розробки та підтримки. Розробники повинні обережно враховувати переваги та недоліки цього підходу перед вирішенням, чи використовувати його в своїй грі. [5]

Таблиця 1.1.

Зведені дані порівняння готових рішень захисту багатокористувацьких ігор

	VAC	Riot Vanguard	BattlEye	EAC	Anti-Cheat Toolkit
Підтримка OS					
Windows	+	+	+	+	+
Linux	+	-	+	+	-
iOS/MacOS	+	-	+	+	+
Рівень спостереження					
Системний	+	+	+	+	+

Продовження таблиці 1.1.

Мережевий	+	+	+	+	-
Відкритість	-	-	-	+	+
Гнучкість налаштування	-	-	-	+	+
Вплив на продуктивність	-	+	+	+	-
Регулярність оновлення	-	+	-	+	-
Легкість інтеграції	-	-	-	+	+
Можливість використання під власним брендом	-	-	-	+	+
Ведення звітності	+	+	+	-	-

1.4. Висновки до розділу

1. В пункті 1.1. було проведено аналіз сучасного стану забезпечення безпеки інформації. Було розглянуто основні виклики та інновації в сфері безпеки.
2. В пункті 1.2. було проведено класифікацію загроз та вразливостей інформації. Під час аналізу було виділено основні загрози, їх приклади та вплив на інформаційну безпеку. Також було проаналізовано загрози та вразливості багатокористувацьких ігор.
3. В пункті 1.3. було проведено порівняльний аналіз готових рішень із забезпечення інформаційної безпеки в багатокористувацьких іграх. За результатами аналізу було створено зведену таблицю порівняння.

РОЗДІЛ 2
ОСНОВИ МЕТОДІВ ЗАХИСТУ У БАГАТОКОРИСТУВАЦЬКІЙ ГРІ

2.1. Методи захисту даних у багатокористувацькій грі

2.1.1. Криптографічні методи в захисті даних в іграх.

Криптографічні методи захисту в іграх відіграють ключову роль у забезпеченні безпеки та надійності ігрового середовища. Основні ролі цих методів включають:

- Конфіденційність даних: Криптографічне шифрування важливо для захисту конфіденційних даних гравців, таких як особиста інформація та облікові записи. Захищений обмін даними між гравцями та серверами гри забезпечує конфіденційність особистої інформації.
- Цілісність даних: Цифрові підписи та хеш-функції використовуються для перевірки цілісності ігрових даних. Це допомагає виявляти будь-які зміни чи втручання в ігрові файли чи комунікацію між клієнтом і сервером.
- Аутентифікація та Авторизація: Криптографічні методи допомагають впевнитися в аутентичності гравців під час входу в гру та забезпечують безпечний обмін інформацією для авторизації доступу до різних ресурсів гри.
- Боротьба з обманом (Anti-Cheat):

Використання криптографічних алгоритмів для виявлення шахрайських методів та програмного забезпечення. Це допомагає уникати обману та підтримує чесну гру для всіх гравців.

- Захист від атак:

Криптографічні методи допомагають у захисті від різних видів атак, таких як атаки на витік даних, атаки на збережені дані та інші загрози ігровому середовищу.

- Управління ключами:

Ефективне управління ключами грає важливу роль у забезпеченні безпеки криптографічних систем. Безпечне зберігання та передача ключів важливо для запобігання несанкціонованому доступу.

- Захист від внутрішніх загроз:

Криптографічні методи можуть допомагати в запобіганні внутрішнім загрозам, таким як витік ігрової інформації від працівників або атаки від інсайдерів.

Усі ці аспекти сприяють створенню безпечного, надійного та справедливого ігрового середовища, яке є важливим для залучення гравців та збереження їхнього довір'я до гри.

2.2.2. Використання хеш-функцій для забезпечення цілісності даних.

В багатокористувацьких іграх важливо забезпечити цілісність даних, оскільки цілісність ігрової інформації є критичним для уникнення можливих атак та забезпечення чесної та стабільної гри для всіх гравців. Використання хеш-функцій є ефективним методом для досягнення цієї мети. Ось як це може виглядати:

- Хешування даних гравців: Перед збереженням чутливих даних гравців (таких як паролі, інформація про обліковий запис) в базі даних, вони можуть бути хешовані. Коли гравець входить в гру, введені ним дані також хешуються та порівнюються із збереженим хешем.
- Хешування ігрових файлів: Файли гри, що містять ігрові дані, можуть бути хешовані для перевірки їхньої цілісності. Це може включати текстури, аудіофайли, конфігураційні файли тощо. Якщо будь-які файли модифікуються або пошкоджуються, їхні хеш-значення змінюються.
- Перевірка цілісності патчів і оновлень: Перед застосуванням патчів або оновлень гри, можна обчислити хеш-значення цих файлів і порівняти їх із заздалегідь визначеними значеннями. Це забезпечить, що патчі або оновлення не були змінені чи пошкоджені під час завантаження.
- Античит-заходи: Хеш-функції можуть бути використані для створення хеш-сум для деяких ігрових елементів або параметрів. Ці значення можуть

періодично відправлятися на сервер для перевірки їхньої валідності та виявлення можливих модифікацій чи шахрайства.

- Хешування ідентифікаторів сесій: При обміні даними між клієнтом і сервером, ідентифікатори сесій або токени можуть бути хешовані для забезпечення безпеки та уникнення можливих атак на ці дані.
- Контроль цілісності мережевого трафіку: Хеш-функції можна використовувати для перевірки цілісності мережевого трафіку між клієнтами та серверами гри, щоб уникнути атак на перехоплення чи модифікацію даних.

Застосування хеш-функцій у багатокористувацьких іграх допомагає зменшити ризик змін та модифікацій важливих ігрових елементів, що сприяє створенню надійного та безпечного ігрового середовища.

2.1.3. Методи шифрування програмного коду гри.

Шифрування програмного коду гри - це важлива складова заходів забезпечення безпеки, яка спрямована на ускладнення або унеможливлення доступу до вихідного коду гри незадовго до чи під час виконання гри. Це може захищати від незаконного копіювання, реверс-інженерії та інших атак. Ось кілька методів шифрування програмного коду гри:

- Обфускація коду: Цей метод використовується для ускладнення зрозуміння вихідного коду, змінюючи його структуру та назви змінних та функцій без зміни його логіки. Обфускація може використовувати різні техніки, такі як заміна назв функцій та змінних на невідомі або скорочені варіанти, вставка непотрібного коду та інші прийоми для заплутування коду.
- Шифрування строк: Строки, такі як константи чи інші чутливі дані, можуть бути зашифровані, щоб ускладнити їхнє зрозуміння. Під час виконання гри вони розшифровуються динамічно. Це може унеможливити просте аналізування коду шляхом перегляду строк у вихідному коді.
- Шифрування всього вихідного коду: Застосування алгоритмів шифрування для всього вихідного коду гри. Код розшифровується перед виконанням

програми. Це ускладнює процес зворотного інженерінгу, оскільки вихідний код непрозорий, доки не буде розшифрований.

- Використання віртуальних машин: Використання віртуальних машин для виконання коду гри. Код компілюється в інструкції віртуальної машини, а потім виконується на віртуальній машині, яка може бути обернено інженерована тільки в складних умовах.
- Поділ коду на модулі: Розділення програмного коду на модулі або бібліотеки та завантаження їх динамічно. Це може ускладнити аналіз і виявлення суттєвих частин коду.
- Динамічне завантаження ресурсів: Заміна статичних ресурсів (таких як текстури, звуки) на динамічно завантажувані. Такий підхід може використовувати шифрування для захисту ресурсів від простого доступу.

Завдання полягає в тому, щоб вибрати комбінацію зазначених методів, яка найефективніше відстоюватиме ваш код від неправомірного втручання та реверс-інженерії. У той же час, слід пам'ятати, що абсолютної безпеки не існує, і всі заходи можуть бути подолані з досвідом і ресурсами.

2.3.4. Ефективність різних методів шифрування для уникнення атак.

Ефективність різних методів шифрування великою мірою залежить від контексту та використання. Існує ряд різних методів шифрування, і кожен з них має свої переваги та обмеження. Ось кілька методів та їхній потенційний вплив на уникнення атак:

- Симетричне шифрування:

  Переваги: Ефективне для шифрування великих обсягів даних, швидке.

  Обмеження: Вимагає безпечного обміну ключами, що може бути складним для здійснення в розподілених системах.
- Асиметричне шифрування:

  Переваги: Не вимагає обміну секретним ключем, більш безпечне для відкритого обміну ключами.

  Обмеження: Може бути менш ефективним для шифрування великих обсягів даних.

- Хеш-функції:

  Переваги: Швидкі та ефективні для перевірки цілісності, невід'ємні (однаковий вхід завжди видає один і той же хеш).

  Обмеження: Чутливі до зіткнень (два різних вхідні дані можуть виробляти один і той же хеш).

- TLS/SSL шифрування:

  Переваги: Забезпечує захист трафіку між клієнтом та сервером, використовує комбінацію симетричного та асиметричного шифрування.

  Обмеження: Потребує налаштувань та підтримки інфраструктури.

- Квантова криптографія:

  Переваги: Теоретично надійна перед квантовими обчислювачами, здатна виявляти неправомірний доступ.

  Обмеження: Технологічно складна та на даний момент досить експериментальна.

- Шифрування з використанням блокчейн-технологій:

  Переваги: Децентралізована та стійка до модифікацій.

  Обмеження: Масштабованість та ефективність для великих обсягів даних.

- Шифрування з використанням хмарних технологій:

  Переваги: Забезпечує зручний доступ та управління ключами, може забезпечити високу доступність.

  Обмеження: Залежність від третіх сторін, питання щодо конфіденційності даних.

Найбільш ефективний метод шифрування залежить від конкретного застосування, рівня безпеки, потреб користувачів та потенційних загроз. Зазвичай, комбінація різних методів (зокрема, застосування "захисту в глибину") може забезпечити більш високий рівень безпеки.

Таблиця 2.1.

Зведені дані порівняння різних методів шифрування

	Симетричне шифрування	Асиметричне шифрування	Хеш-функції	TLS/SSL	Блокчейн-технології
Швидкість шифрування	+	-	+	-	-
Потрібна наявність ключа	+	-	+	+	-
Якість безпеки	+	-	+	+	+
Захист мережевої передачі даних	-	-	-	+	+
Децентралізованість	-	+	-	-	+
Залежність від третіх сторін	+	+	+	+	-
Масштабованість	+	+	-	+	-
Технологічно складна для реалізації	-	-	-	+	+

2.2. Теоретичні основи захисту даних в багатокористувацьких середовищах

2.2.1 Розгляд атак на основі перехоплення даних в грі.

Атаки на основі перехоплення даних у грі є серйозною загрозою для безпеки та конфіденційності гравців та геймдевелоперів. Ці атаки можуть включати

перехоплення комунікації між гравцем і сервером, витягання конфіденційної інформації, такої як паролі, або зміну гейм-даних для отримання неправомірних переваг. Ось деякі типи атак на основі перехоплення даних у грі та способи захисту:

- Man-in-the-Middle атаки: Атака, при якій зловмисник вставляється між комунікаційними каналами між гравцем і сервером, перехоплюючи та можливо змінюючи дані.
- Атаки на аутентифікацію: Зловмисник намагається взломати або перехопити інформацію про аутентифікацію гравця. Для прикладу можуть використовуватись такі технології:
    - Брутфорс: Зловмисник випробовує всі можливі комбінації логінів та паролів.
    - Фішинг: Використання фальшивих веб-сайтів чи листів електронної пошти для отримання авторизаційних даних.

  Для захисту від таких атак можуть використовуватись наступні методи:
    - Використання сильних паролів та двоетапної верифікації.
    - Регулярна зміна паролів та використання антибрутфорс заходів.
    - Витягання конфіденційної інформації:
- Витягання конфіденційної інформації: Зловмисник намагається отримати конфіденційну інформацію, таку як особисті дані гравців, номери кредитних карт тощо. Для прикладу:
    - Імперсонація: Зловмисник видається за довірену особу для отримання конфіденційної інформації.
    - Соціальна інженерія: Використання маніпуляцій та обману для отримання інформації.

  Для захисту від подібних атак допоможуть наступні методи:
    - Шифрування конфіденційної інформації у споживача та під час передачі.
    - Посилення обізнаності гравців стосовно соціальної інженерії.

- Атаки на гейм-протокол: Зловмисник намагається змінити гейм-протокол для отримання неправомірних переваг у грі. Прикладом таких атак є:
    - Підміна пакетів: Зміна або інтерсепція пакетів, які взаємодіють між гравцем і сервером.
    - Використання читів: Модифікація клієнтського коду для отримання переваг у грі.

  Для захисту від подібних атак варто розглядати наступні методи:
    - Виявлення та відхилення модифікованих клієнтських сторін.
    - Використання цифрових підписів для перевірки подій у грі.
- Атаки на локальні сховища гравця: Зловмисник намагається отримати доступ до локальних сховищ гравців для витягання чутливої інформації або зміни гейм-даних. Для прикладу приведено кілька варіацій атак на локальні сховища:
    - Експлуатація вразливостей: Використання вразливостей у програмному забезпеченні для незаконного доступу.
    - Кейлоггери: Встановлення програм, які реєструють натискання клавіш для отримання паролів.

  Для захисту від подібних атак варто розглянути наступні методи:
    - Шифрування локальних сховищ гравців.
    - Використання антивірусного програмного забезпечення та антикейлоггерів.
- Використання читів і ботів: Зловмисники можуть використовувати чити та боти для отримання неправомірних переваг та порушення балансу гри.

  Прикладами таких атак є наступні:
    - Автоматизація: Використання програм для автоматизації дій гравця.
    - Модифікація гейм-файлів: Зміна гейм-файлів для отримання переваг.

  Для захисту від атак з використанням читів і ботів варто розглянути наступні методи:

- Використання систем античиту та регулярне оновлення для виявлення та блокування читів і ботів.
- Суворі політики використання сторонніх програм.

- Соціальна інженерія: Зловмисники використовують маніпуляції та обман для отримання конфіденційної інформації від гравців чи співробітників геймдевелоперів.

До цих атак відносяться наступні:

- Фішинг: Використання підроблених веб-сайтів або електронних листів для отримання інформації.
- Відома атака: Зловмисник вивчає поведінку та звички гравців для ефективного обману.

Для захисту від подібних атак варто брати до уваги наступні методи:

- Посилення обізнаності користувачів щодо соціальної інженерії та фішинг-атак.
- Встановлення політик безпеки та здійснення аудиту соціальної інженерії.

2.2.2. Захист від Man-in-the-Middle атак у геймінгу.

Man-in-the-Middle (MitM), або атака "чоловік посередині", є видом кібератаки, при якій зловмисник вставляється в комунікаційний канал між двома сторонами. Зловмисник стає "посередником" і може перехоплювати, блокувати або навіть змінювати передавані дані між сторонами без їхнього відома чи згоди.

Основні характеристики атак Man-in-the-Middle включають:

- Перехоплення даних: Зловмисник може отримувати доступ до передаваної інформації між двома сторонами. Це може включати текстові повідомлення, паролі, особисті дані, фінансові дані та інші конфіденційні відомості.
- Зміна даних: Зловмисник може модифікувати передавані дані, щоб впливати на зміст або результати обміну інформацією між сторонами. Наприклад, він може змінювати суму грошей у фінансовому переказі або повідомлення у чаті.

- Підробка аутентифікації: Зловмисник може виглядати для обох сторін, наче він є легітімним комунікаційним партнером. Він може використовувати різні методи для імітації аутентичності, такі як фішинг або підробка сертифікатів безпеки.
- Диверсія данних: Зловмисник може вводити додаткові дані в комунікаційний потік, які не були відправлені або отримані оригінальними сторонами.

До атак MitM можна використовувати різні методи, включаючи:

- ARP Spoofing: Зловмисник надсилає фальшиві ARP-пакети для поверхні в мережі, змушуючи інші пристрої вважати, що атакуючий пристрій є легітімним шлюзом.
- DNS Spoofing: Зміна відповідей DNS-запитів для перенаправлення трафіку на контрольовані зловмисником сервери.
- Вибірковий прослуховувач: Використання спеціалізованого обладнання для перехоплення трафіку в бездротових мережах.

MitM-атаки можуть бути використані для здійснення різних злочинів, таких як крадіжка ідентифікаційних даних, фінансові махінації, або навіть для впливу на процеси великої організації чи уряду. Захист від атак MitM включає в себе застосування шифрування, безпечних методів аутентифікації, моніторинг мережевого трафіку та використання безпекових протоколів.

Захист від атак Man-in-the-Middle (MitM) у геймінгу є критично важливим, оскільки ці атаки можуть призвести до втрати конфіденційної інформації та порушення безпеки геймплейу. Ось деякі заходи безпеки, які можна вжити для захисту від MitM-атак у геймінговому середовищі:

- Використання шифрування трафіку: Встановлення шифрованого з'єднання, наприклад, за допомогою протоколу HTTPS для веб-геймінгу чи шифрування мережевого трафіку для онлайн ігор.
- Застосування протоколів, які забезпечують захист від перехоплення, наприклад, TLS/SSL.

- Використання відповідних методів аутентифікації: Застосування сильних методів аутентифікації для перевірки ідентичності користувачів. Впровадження двоетапної верифікації, щоб зробити атаки на паролі більш складними.
- Використання цифрових підписів: Використання цифрових підписів для підтвердження автентичності гейм-клієнтів та серверів. Захист важливих гейм-даних від змін або підробки за допомогою цифрових підписів.
- Захист Від ARP: Моніторинг та виявлення атак на ARP (Address Resolution Protocol) для попередження вставлення зловмисника в середину мережі. Використання ARP-безпеки та обмеження динамічних ARP-відповідей.
- Створення внутрішньої бездротової мережі: Встановлення внутрішньої бездротової мережі для геймінгу, щоб зменшити ризик неконтрольованого підключення до невідомих мереж. Застосування безпеки WPA3 для захисту внутрішньої бездротової мережі.
- Захист локальних сховищ геймерів: Використання шифрування для локальних сховищ гравців, де зберігаються конфіденційні дані, такі як паролі та особисті дані. Регулярна перевірка та оновлення захисту локальних сховищ.
- Посилення безпеки мережевих портів: Застосування технологій, таких як портиал входу (port knocking) або використання віртуальних приватних мереж (VPN), щоб забезпечити додатковий захист від недозволених підключень.
- Активний моніторинг та виявлення аномалій: Використання систем виявлення інтранет-загроз для виявлення аномальної активності в мережі. Реагування на невідомі або підозрілі пристрої, які можуть бути спробами MitM-атак.
- Інструкції для гравців: Надання рекомендацій гравцям щодо безпечного підключення до мережі та обрання надійних мережевих налаштувань. Посилення обізнаності гравців щодо потенційних загроз та методів їх уникнення.

- Оновлення та Патчі: Регулярне оновлення програмного забезпечення гри та застосування патчів для виправлення вразливостей, що можуть використовуватися для MitM-атак.

Ці заходи разом створюють більшу стійкість проти MitM-атак у геймінговому середовищі та забезпечують більшу безпеку гравців та геймдевелоперів.

2.2.3. Використання SSL/TLS протоколів для безпечної комунікації.

Використання SSL/TLS протоколів у багатокористувацьких іграх є важливим елементом для забезпечення безпеки комунікації між клієнтами та серверами. SSL (Secure Sockets Layer) та його еволюційний нащадок TLS (Transport Layer Security) є криптографічними протоколами, які гарантують шифрування та захист від перехоплення чи модифікації даних під час передачі через мережу.

Основні переваги використання SSL/TLS у багатокористувацьких іграх включають:

- Шифрування даних:

  SSL/TLS забезпечує шифрування даних, яке ускладнює перехоплення та читання інформації, переданої між гравцями та серверами.

  Захист особистих даних, таких як логіни, паролі, а також іншої конфіденційної інформації, від несанкціонованого доступу.

- Аутентифікація:

  SSL/TLS включає процес аутентифікації, що дозволяє гравцям та серверам впевнитися в тому, що вони спілкуються саме з тим пристроєм або сервером, який вони очікують.

  Захист від атак Man-in-the-Middle, які можуть виникнути у відсутності аутентифікації.

- Цілісність даних:

  SSL/TLS гарантує цілісність переданих даних, забезпечуючи, що вони не були змінені або пошкоджені під час передачі.

  Запобігання модифікації гейм-даних або інших важливих повідомлень.

- Довіреність та сертифікація:

  SSL/TLS використовує сертифікати для підтвердження автентичності сервера чи клієнта, що зменшує ризик атак, які базуються на підробці або фішингу.

  Гарантує, що гравці спілкуються та обмінюються інформацією лише з довіреними серверами.

- Захист від перехоплення токенів автентифікації:

  SSL/TLS мінімізує ризик перехоплення токенів автентифікації, таких як сесійні ключі, що можуть використовуватися для входу в гру.

  Збереження безпеки авторизаційних даних гравців та уникнення несанкціонованого доступу.

- Захист від атак "Багатокористувацького Спостереження":

  SSL/TLS сприяє запобіганню атакам, спрямованим на спостереження та аналіз трафіку між гравцями та серверами.

  Збереження конфіденційності гейм-даних та уникнення витоку інформації.

Використання SSL/TLS протоколів у багатокористувацьких іграх є невід'ємною частиною загального заходу захисту, який допомагає забезпечити конфіденційність, цілісність та аутентичність комунікації, що є критично важливим у галузі геймінгу.

2.2.4 Аналіз ризиків і заходів безпеки в реальному часі.

Багатокористувацькі ігри, які працюють в режимі реального часу, стикаються з різноманітними ризиками, пов'язаними з безпекою. Ці ризики можуть включати технічні загрози, атаки на гейм-сервери, маніпулювання гейм-протоколами, а також соціально-інженерні аспекти. Ось аналіз деяких ризиків та заходів безпеки:

- Перехоплення трафіку і атаки Man-in-the-Middle (MitM):

  Ризики:

  - Зміна або перехоплення гейм-даних.
  - Втрата конфіденційної інформації гравців.

  Заходи безпеки:

- Використання SSL/TLS для шифрування гейм-трафіку.
- Захист від атак ARP Spoofing та інших методів MitM.

- Атаки на сервер і відмова в обслуговуванні (DoS/DDoS):

  Ризики:
  - Завади у роботі гейм-сервера через перевантаження запитами.
  - Перерви в гейм-сесіях для гравців.

  Заходи безпеки:
  - Використання систем захисту від DDoS.
  - Масштабування інфраструктури для роботи зі зростанням трафіку.

- Маніпулювання гейм-протоколами і читами:

  Ризики:
  - Можливість зміни гейм-даних та отримання неправомірних переваг.
  - Зменшення балансу гри через використання читів.

  Заходи безпеки:
  - Використання цифрових підписів для перевірки подій у грі.
  - Системи античиту та регулярні оновлення для виявлення та блокування читів.

- Фішинг та обман гравців:

  Ризики:
  - Втрата облікових записів гравців.
  - Розповсюдження шкідливих програм через фішингові атаки.

  Заходи безпеки:
  - Посилення освіти гравців щодо фішингу та методів захисту.
  - Двоетапна аутентифікація та використання безпекових підходів до управління обліковими записами.

- Виток конфіденційної інформації:

  Ризики:
  - Втрата особистих даних гравців через атаки на бази даних.
  - Зловживання конфіденційною інформацією гравців.

  Заходи безпеки:

- Захист баз даних за допомогою шифрування та регулярних аудитів безпеки.
- Забезпечення відповідності з правилами GDPR та інших законодавчих актів щодо захисту особистих даних.

- Атаки на клієнтську сторону (exploits):

  Ризики:
  - Використання вразливостей клієнтського програмного забезпечення для введення шкідливих кодів.
  - Зміна локальних гейм-даних на пристроях гравців.

  Заходи безпеки:
  - Регулярні оновлення гейм-клієнтів та застосування патчів для виправлення вразливостей.
  - Використання антивірусного програмного забезпечення та антиексплоітів.

- Атаки соціальної інженерії:

  Ризики:
  - Обман гравців та здобуття їхніх облікових даних.
  - Розповсюдження шкідливих вірусів через соціальні мережі гри.

  Заходи безпеки:
  - Посилення освіти гравців щодо соціальної інженерії.
  - Використання систем фільтрації та блокування небажаних вмістів.

- Забезпечення безпеки гравців:

  Ризики:
  - Невірна поведінка гравців, включаючи булінг, образи та інші форми онлайн-агресії.
  - Ризики пов'язані з особистою безпекою гравців.

  Заходи безпеки:
  - Встановлення правил поведінки та антибулінгових політик.
  - Можливість звітувати про порушення та взаємодія з модераторами.

- Моніторинг та реагування в реальному часі:

Ризики:

- Недостатній моніторинг може призвести до несприятливих наслідків для гравців та безпеки гри.
- Затримка у виявленні та реагуванні на інциденти може посилити їхні наслідки.

Заходи безпеки:

- Використання систем виявлення аномалій та автоматизованих систем реагування на інциденти.
- Регулярні навчання та тренування персоналу для швидкого та ефективного втручання.

Безпека в багатокористувацьких іграх є комплексним завданням, і вимагає поєднання технічних, організаційних та освітніх заходів для ефективного управління ризиками та забезпечення безпеки гравців та інфраструктури гри.

2.2.5 Оцінка можливостей виявлення та відхилення атак під час гри.

Оцінка можливостей виявлення та відхилення атак під час гри включає в себе використання різних технічних, організаційних та аналітичних засобів для виявлення аномалій та реагування на них. Нижче подано кілька ключових елементів, які впливають на ефективність цього процесу:

- Системи виявлення аномалій:

  Можливості:

  - Використання систем виявлення аномалій для моніторингу гейм-трафіку та виявлення атипових зразків поведінки гравців чи атак на сервер.
  - Використання машинного навчання для автоматизації виявлення аномалій.

  Виклики:

  - Розробка досконалих моделей для виявлення аномалій у гейм-середовищі може бути складною через велику різноманітність можливих сценаріїв.

- Аналіз гейм-даних:

Можливості:

- Використання аналітики гейм-даних для виявлення аномальних статистичних показників, що можуть вказувати на атаки або несправедливі переваги.
- Застосування інструментів бізнес-аналітики для визначення аномалій у споживанні ресурсів гравцями.

Виклики:

- Обробка великої кількості гейм-даних та виявлення суттєвих аномалій може вимагати потужних обчислювальних ресурсів та комплексних алгоритмів.

- Системи античиту:

  Можливості:

  - Використання спеціалізованих систем античиту для виявлення та блокування зловживань та читів.
  - Впровадження систем моніторингу інтегритету гейм-клієнтів та гейм-серверів.

  Виклики:

  - Постійне оновлення систем античиту для боротьби з новими методами обхіду захисту.

- Моніторинг мережі та серверів:

  Можливості:

  - Системи моніторингу мережі для виявлення аномальної активності та несправностей.
  - Використання технік "baseline" для визначення типового зразка поведінки гравців та виявлення аномалій.

  Виклики:

  - Забезпечення низької латентності при великому обсязі мережевого трафіку.

- Організаційні заходи:

  Можливості:

- Створення ефективних процедур відгуку на інциденти та навчання персоналу щодо реагування на атаки.
- Впровадження політик безпеки та вимог щодо безпеки для розробників та адміністраторів гри.

Виклики:

- Забезпечення взаємодії між розробниками, адміністраторами та командами безпеки.

- Взаємодія з гравцями:

Можливості:

- Використання засобів спілкування з гравцями для отримання зворотного зв'язку стосовно аномальної поведінки чи зловживань.
- Створення каналів для подання скарг щодо потенційних атак.

Виклики:

- Розробка ефективних систем фідбеку та забезпечення конфіденційності гравців, які подають заявки.

Ефективність виявлення та відхилення атак в багатокористувацьких іграх залежить від комплексного підходу, що включає в себе технічні, організаційні та аналітичні засоби, а також співпрацю між різними сторонами, такими як розробники, адміністратори, безпекові експерти та самі гравці.

2.2.6 Важливість аналізу поведінки гравців для виявлення підозрілих активностей.

Аналіз поведінки гравців виявляється надзвичайно важливим для забезпечення безпеки та виявлення підозрілих активностей в багатокористувацьких іграх. Ось деякі ключові аспекти, які підкреслюють важливість аналізу поведінки гравців:

- Виявлення шаблонів та аномалій: Аналіз поведінки гравців допомагає виявляти типові шаблони поведінки, а також аномалії в порівнянні з ними. Автоматичне виявлення аномальних активностей може вказувати на потенційні атаки чи зловживання.

- Визначення нормальної поведінки: Спостереження та аналіз поведінки гравців дозволяє визначити, що вважається "нормальною" поведінкою у конкретній грі чи середовищі. Це стає відправною точкою для виявлення невідомих або непритаманних зразків поведінки.
- Виявлення читів та зловживань: Багатокористувацькі ігри часто стикаються з використанням читів та неправомірних програм. Аналіз поведінки дозволяє виявляти надзвичайні досягнення чи дії, які можуть свідчити про використання зловживань.
- Боротьба із зловживанням та агресією: Виявлення агресивної або неетичної поведінки гравців може допомогти в попередженні конфліктів та забезпеченні позитивного геймплею. Це особливо важливо у відносинах з спільнотою гравців.
- Антиспам та захист від флуду: Виявлення надмірного спаму чи флуду в чатах може зменшити негативний вплив такої діяльності на гравців та гру в цілому. Аналіз патернів спаму дозволяє вчасно виявляти та втручатися.
- Захист від афер та обману: Виявлення підозрілих обманниць та афер може допомогти в збереженні конфіденційності та безпеки гравців. Аналіз несподіваної зміни в ставках або грошових транзакціях може вказувати на шахрайство.
- Автоматизована система виявлення аномалій: Розробка автоматизованих систем виявлення аномалій дозволяє виявляти підозрілі активності у реальному часі. Можливість автоматично реагувати на аномалії підвищує ефективність заходів безпеки.
- Безпека та захист від гравців: Забезпечення безпеки гравців та їх особистих даних важливо для збереження довіри до гри та розвитку гейм-середовища. Виявлення підозрілих активностей дозволяє уникати витоку особистої інформації та інших ризиків.
- Сприяння позитивному геймплею: Виявлення агресивної чи неетичної поведінки сприяє створенню позитивного та безпечного геймплею для всіх

гравців. Аналіз поведінки дозволяє виявляти та негайно реагувати на недоречні дії гравців.

- Виявлення знань та вмінь гравців: Аналіз поведінки допомагає виявляти реальні знання та вміння гравців, що може бути важливим для підтримки справедливого та конкурентного геймплею. Розробка систем, які враховують рівень навичок гравців, покращує гейм-досвід.

Аналіз поведінки гравців не лише виявляє підозрілі активності, але й допомагає вдосконалювати геймплей, створюючи безпечне та задовільне гейм-середовище. Заходи безпеки, які базуються на аналізі поведінки гравців, стають необхідним компонентом в сучасних багатокористувацьких іграх.

2.3. Теоретичні основи інтеграції заходів захисту в багатокористувацьку гру

2.3.1. Впровадження систем моніторингу та виявлення порушень.

Впровадження системи моніторингу та виявлення порушень у багатокористувацьку гру є важливим етапом для забезпечення безпеки, чесності та розвитку гри. Далі наведено кілька кроків та рекомендацій стосовно розробки системи моніторингу та виявлення порушень у багатокористувацькій грі:

- Аналіз ідентифікації користувачів. Використовуйте надійні методи аутентифікації, такі як двофакторна аутентифікація, щоб ускладнити несанкціонований доступ. Запроваджуйте систему унікальних ідентифікаторів для кожного користувача.
- Моніторинг дій користувачів: Записуйте дії користувачів, щоб мати можливість аналізувати їхню поведінку. Стежте за підозрілими або надто активними аккаунтами.

- Аналіз гравців та виявлення шаблонів: Розробляйте алгоритми для виявлення аномальної поведінки або використання читів. Використовуйте машинне навчання для виявлення шаблонів порушень.
- Обробка скарг та звітів:

Забезпечте можливість гравцям легко повідомляти про підозрілу або нечесну діяльність. Створюйте механізми для швидкого аналізу та реагування на скарги.

- Активний моніторинг заходів безпеки: Постійно оновлюйте систему моніторингу відповідно до нових загроз та методів обходу захисту. Використовуйте тестування на проникнення для виявлення потенційних вразливостей.
- Покарання та запобігання: Запроваджуйте ефективні санкції для тих, хто порушує правила гри. Проводьте регулярні аудити та перевірки, щоб запобігти порушенням.
- Спільнота та прозорість: Залучайте гравців до процесу виявлення порушень та розробки правил. Надавайте інформацію про заходи, які ви приймаєте для забезпечення чесності гри.
- Документація та навчання: Надайте користувачам доступ до правил та процедур виявлення порушень. Проводьте навчання для гравців щодо наслідків порушення правил.

Важливо враховувати, що система моніторингу повинна бути гнучкою та адаптованою до специфіки вашої гри, і регулярно оновлюватися відповідно до змін в гральному середовищі та нових методів обходу захисту.

2.3.2. Розгляд принципів інтеграції систем моніторингу безпеки у геймінгові платформи.

Інтеграція систем моніторингу безпеки у геймінгові платформи вимагає комплексного підходу для забезпечення надійності, чесності та безпеки гравців. Одним із ключових принципів є впровадження потужних систем ідентифікації та аутентифікації, які дозволяють ефективно визначати користувачів і захищати їхні облікові записи від несанкціонованого доступу.

Важливим аспектом є моніторинг активності гравців під час гри, збір і аналіз даних про їхню поведінку. Це дозволяє виявляти аномалії та підозрілі дії, що можуть свідчити про використання читів або іншу нелегітимну активність.

Для боротьби з читами і нелегітимним програмним забезпеченням, важливо впроваджувати алгоритми виявлення, які вчаться на основі попередніх випадків та аналізу шаблонів порушень. Це допомагає ефективно реагувати на нові види шахрайства та забезпечує постійне оновлення системи моніторингу.

Покладання акценту на взаємодію з гравцями та створення механізмів зворотнього зв'язку також важливо. Вислуховування їхніх скарг, аналіз звітів про порушення та швидка реакція на них дозволяє підтримувати довіру гравців до платформи.

Необхідно регулярно оновлювати систему моніторингу, дотримуючись принципу активного вдосконалення та адаптації до нових викликів та загроз безпеці в гральній індустрії. Також важливо забезпечувати прозорість щодо заходів безпеки, спілкування з гравцями та надання навчання з питань безпеки для підвищення обізнаності користувачів.

2.3. Висновки до розділу

В пункті 2.1. було проведено аналіз та порівняння основних математичних методів захисту даних у багатокористувацькій грі. Було наведено порівняльну таблицю з основними аспектами у виборі методів шифрування для багатокористувацького середовища.

В пункті 2.2. було проведено теоретичний аналіз основних загроз для багатокористувацького середовища, різні аспекти атак та способи їх запобігання.

В пункті 2.3. було наведено теоретичну інформації щодо впровадження систем моніторингу та безпеки у багатокористувацьку гру. Ці дані допоможуть створити більш гнучку і налаштовану систему.

РОЗДІЛ 3
РОЗРОБКА СИСТЕМИ ЗАХИСТУ ДАНИХ У БАГАТОКОРИСТУВАЦЬКІЙ ГРІ

3.1. Опис середовища розробки

3.1.1. Опис ігрового рушія

Unity — це ігровий рушій (game engine), що використовується для розробки ігор та інших інтерактивних додатків. Основні характеристики та компоненти Unity включають:

- Графічний рушій:
  - Unity надає потужний графічний рушій, який підтримує рендеринг 2D та 3D графіки.
  - Має широкий набір ефектів, освітлення, тіней, та інших графічних можливостей.
- Крос-платформеність: Одна з ключових переваг Unity - це можливість розробляти ігри для різних платформ, таких як Windows, macOS, Linux, Android, iOS, WebGL, Xbox, PlayStation та інші.
- Мови програмування: Unity підтримує кілька мов програмування, таких як C#, JavaScript та Boo. Основним і рекомендованим варіантом є C#.
- Редактор Unity: Має інтуїтивний графічний редактор, який дозволяє вам створювати і редагувати сцени, об'єкти, матеріали та інші ресурси.
- Фізика та анімація: Unity включає вбудовані системи фізики та анімації, що дозволяють створювати реалістичні рухи та поведінку об'єктів.
- Активна спільнота та ресурси: Unity має велику та активну спільноту розробників, а також багато онлайн-ресурсів, таких як форуми, уроки та документація.

- Розширюваність: Unity можна легко розширювати за допомогою додаткових плагінів та активів, які дозволяють вам додавати нові функції та можливості.
- Система компонентів: Багато роботи в Unity базується на концепції компонентів, що робить розробку більш модульною та гнучкою.

Unity застосовується в ігровій індустрії, а також в інших галузях, таких як віртуальна реальність, доповнена реальність, тренажери та симулятори. Він продовжує вдосконалюватися і оновлюватися, надаючи розробникам потужний інструмент для творчості та інновацій.

Unity являється обгрунтованим вибором для розробки багатокористувацьких ігор з приводу кількох ключових факторів. Крос-платформеність, яку пропонує Unity, є критичною для успіху в багатокористувацьких іграх, де гравці можуть користуватися різними пристроями та операційними системами. Це забезпечує широкий охоплення аудиторії та полегшує взаємодію гравців.

Мова програмування C#, яка є основною для Unity, володіє високим рівнем гнучкості та ефективності. Це робить можливим реалізацію складних багатокористувацьких функцій та підтримку стабільного та швидкого обміну даними між гравцями.

Графічний редактор Unity є інтуїтивно зрозумілим інструментом для розробників, дозволяючи легко працювати з візуальними елементами гри та оптимізувати інтерфейс користувача для максимального зручення гравців.

Спільнота Unity є живою та активною, що забезпечує не лише навчання та підтримку, але й можливість обміну досвідом з іншими розробниками. Розширюваність Unity за допомогою плагінів дозволяє долучити до проекту додаткові функції та інструменти, що робить його гнучким інструментом для втілення різноманітних ідей у багатокористувацьких іграх.

Photon PUN 2

Photon PUN (Photon Unity Networking) 2 є розширенням для Unity, яке надає потужний інструментарій для розробки багатокористувацьких ігор. Основні риси та характеристики Photon PUN 2 включають:

- Мережева архітектура: Photon PUN 2 пропонує потужну мережеву архітектуру, що дозволяє легко створювати багатокористувацькі ігри з високою продуктивністю.
- Синхронізація об'єктів: Забезпечує ефективну синхронізацію стану гри між клієнтами та сервером, щоб забезпечити консистентність для усіх гравців.
- Лобі та кімнати: Photon дозволяє легко створювати лобі, в яких гравці можуть обирати кімнати для гри, та автоматично знаходити оптимальне підключення.
- Підтримка платформ: Photon PUN 2 підтримує різні платформи, такі як Windows, macOS, Android, iOS, WebGL, Xbox, PlayStation, що дозволяє розробникам досягати більшого аудиторійного охоплення.
- Працює з Photon Cloud або власним сервером: Можливість використовувати хмарний сервіс Photon Cloud або налаштовувати власний сервер для керування мережевими сеансами.
- Розширені функції мережевого геймплею: Photon PUN 2 надає розширені можливості для мережевого геймплею, такі як авторитети об'єктів, індикація затримок та інші функції, що сприяють якості гри.

Photon PUN 2 дозволяє розробникам швидко і легко створювати багатокористувацькі ігри, зосереджуючись на геймплейі та іншій творчій роботі, оскільки він вже надає потужні інструменти для мережевого взаємодії.

Unity чи Unreal Engine

Unity відзначається кількома перевагами, які можуть бути важливими для певних розробників. По-перше, Unity сприяє швидшому вивченню та пристосуванню, що особливо корисно для початківців та інді-розробників. Легший цикл розробки та доступність готових рішень сприяють швидшому старту проекту. Крім того, Unity часто використовується для менших проектів та експериментів, де важлива швидкість випуску та гнучкість в розробці. Використання мови програмування C# також може бути перевагою для тих, хто вже володіє цією мовою. Усе це робить Unity привабливим вибором для тих, хто шукає простоту в розробці та швидкий випуск продукту.

Таблиця 3.1.

Зведені дані порівняння ігрових рушіїв Unity та Unreal Engine

	Unity	Unreal Engine
Легкість вивченя	+	-
Швидкий процес розробки	+	-
Наявність широкого вибору ресурсів	+	-
Активність спільноти	+	+
Розширюваність програм	+	-
Крос-платформенність	+	+
Детальна фізика та графіка	-	+
Можливість розробки власних систем захисту	+	-

3.1.2 Опис мови програмування

C# — це об'єктно-орієнтована, типізована, строго статична мова програмування, розроблена компанією Microsoft. Вона входить до сімейства мов .NET Framework і є однією з основних мов для розробки додатків для платформи .NET.

Декілька ключових аспектів мови C#:

- Синтаксис: C# має чистий та легко засвоюваний синтаксис, який нагадує синтаксис інших мов програмування, таких як C++ або Java. Це робить його доступним для програмістів з різним рівнем досвіду.

- Об'єктно-орієнтована: Мова C# базується на принципах об'єктно-орієнтованого програмування (ООП), що дозволяє розробникам використовувати концепції класів та об'єктів для організації коду.
- Система типів: C# використовує строгу систему типів, що означає, що типи змінних повинні бути визначені в момент їхнього створення, і це підвищує безпеку та надійність коду.
- Можливості асинхронного програмування: Мова C# має вбудовану підтримку асинхронного програмування, що дозволяє зручно реагувати на асинхронні події та оптимізує роботу з багатозадачністю.
- Платформенна незалежність: Через використання платформи .NET, програми, написані на C#, можуть бути виконані на різних операційних системах, що сприяє переносимості коду.
- Велика бібліотека класів: C# володіє обширною бібліотекою класів, що входить до .NET Framework, що дозволяє розробникам швидко та ефективно взаємодіяти з різними операційними системами та сервісами.

Мова C# широко використовується для розробки десктопних застосунків, веб-додатків, ігор та інших програм. Вона постійно розвивається, додаючи нові функції та покращення, що робить її популярною серед розробників.

У Unity мова програмування C# є основною та рекомендованою мовою для розробки. C# використовується для написання скриптів, що керують поведінкою об'єктів, реалізації геймплейу, логіки і взаємодії між різними елементами гри. Ця мова програмування взаємодіє з ігровим движком Unity, забезпечуючи розробникам зручний інструментарій для творчості та реалізації ідей у своїх проектах. Розширені можливості мови C# дозволяють легко створювати складні системи, реалізовувати штучний інтелект, обробляти введення гравців та взаємодіяти з іншими частинами ігрового середовища в Unity.

Використання мови програмування C# у Unity має кілька переваг:

- Легкість вивчення: C# має чистий та легко засвоюваний синтаксис, що полегшує вивчення для початківців та новачків.

- Інтеграція з Unity: C# є офіційною мовою для скриптів у Unity, що означає глибоку інтеграцію та підтримку з боку движка.
- Широкі можливості розробки: З використанням C# у Unity можна легко реалізовувати різноманітні функції, від логіки геймплею до реалізації штучного інтелекту.
- Велика спільнота та ресурси: C# використовується в багатьох галузях програмування, що сприяє величезній та активній спільноті розробників, яка ділиться знаннями та допомагає у вирішенні проблем.
- Спрощена асинхронна розробка: Вбудована підтримка асинхронного програмування у C# полегшує роботу з асинхронними операціями та подіями.
- Висока продуктивність: C# дозволяє реалізувати продуктивний код, що важливо для оптимізованих та швидких ігрових додатків.
- Крос-платформеність: Код, написаний на C# для Unity, може бути легко перенесений між різними платформами, забезпечуючи крос-платформеність гри.

Загалом, використання C# у Unity надає зручний та потужний інструмент для розробки ігор, дозволяючи розробникам швидко та ефективно втілювати свої ідеї.

3.2 Основний функціонал системи захисту даних в багатокористувацькій грі

3.2.1 Шифрування внутрішньо-ігрових даних

Шифрування внутрішньо-ігрових даних важливе для забезпечення безпеки та унеможливлення несанкціонованого доступу до конфіденційної інформації. Цей процес забезпечує захист від можливих атак та витоків даних, а також дозволяє зберігати важливу інформацію, таку як паролі або ігровий прогрес, у безпечному стані. Шифрування внутрішньо-ігрових даних включає в себе застосування

різноманітних шифрувальних алгоритмів та практик, щоб забезпечити конфіденційність та цілісність інформації в межах гри.

В розробленому проєкті представлено велику кількість даних, які потрібно шифрувати задля того, щоб зловмисники не змогли змінити їх через файли програми. Зазвичай всі дані в грі зберігаються в звичайному JSON форматі у вигляді змінних, таких як int, float, string і т.д.

Для прикладу розглянемо звичайний json файл, який був збережений грою без ніякого захисту.

JSON

```
{
  "Coins": "100",
  "Experience": "325.50",
  "Username": "Test",
  "UserID": "123456"
}
```

Зловмисник може з легкістю маніпулювати даними свого акаунту, використовуючи внутрішні дані, завантажені на пристрій. Це перша і основна проблема, яку потрібно вирішити.

Обфускація - це процес ускладнення коду програми або даних з метою утруднення реверс-інжинірингу, тобто забезпечення труднощів у розумінні або відтворенні вихідного коду. Це стандартна практика у сфері програмування та розробки програмного забезпечення для підвищення безпеки та унеможливлення несанкціонованого доступу чи модифікації програмного коду.

Основні аспекти обфускації включають:

- Перейменування: Зміна імен змінних, функцій і класів на більш неочевидні та складні для розуміння.
- Заміна констант: Заміна числових та рядкових констант на інші значення для утруднення розуміння логіки програми.

- Вставлення непотрібного коду: Додавання непотрібного коду, який не впливає на функціональність, але збільшує обсяг та ускладнює аналіз.
- Абстракція умов: Заміна зрозумілих умов та логічних виразів більш складними або вигаданими.
- Шифрування рядків: Шифрування текстових рядків, таких як рядки, що містять повідомлення про помилки чи ключі.
- Видалення зайвого мета-інформації: Вилучення зайвої мета-інформації, яка може бути корисною для реверс-інжинірингу, наприклад, видалення імен методів чи класів.

Обфускація не забезпечує абсолютної безпеки, і великі обфусковані програми все ще можуть бути розібрані. Однак вона створює додаткові труднощі для тих, хто намагається аналізувати або модифікувати вихідний код.

Для основних змінних в грі, які потрібно зберігати локально на пристрої користувача була написана система з використанням класів ObscuredInt, ObscuredFloat, ObscuredString. Це класи які наслідують базову логіку вказаних типів даних, але відмінність у тому що при кожній зміні значення відбувається шифрування.

Для реалізації цього було створено інтерфейс IObscured, який відповідав за шифрування і дешифрування. Тобто даний інтерфейс має два методи:

- Encrypt();
- Decrypt();

Також було створено допоміжний клас ObscuredHelper який створює сесійний ключ, за допомогою якого відбувається шифрування. Даний клас має наступні методи:

- GenerateKey();
- RandomizeCryptoKey();

Для прикладу, гравець виграв матч, йому нарахувало 25 монет. Без даний засобів шифрування його JSON файл матиме наступний вигляд:

```
{
  "Coins": "25",
  "UserName": "Test"
}
```

Зловмисник може зайти до себе в файли програми і просто змінити дане значення, яке автоматично буде підтягнуте кодом при запуску гри. В разі використання Obscured інтерфейсу, ми можемо зашифрувати наші дані.

В такому випадку при зарахуванні монет вони будуть зашифровані кодом програми:

```
public class Test
{
    int coins = 0;

    public void GetReward()
    {
        var reward = 25;
        coins += reward;
        SaveLoadSystem.Data.Coins = reward;
    }
}

public class SaveLoadSystem.Data
{
    public ObscuredInt Coins;

    public void Save()
    {
    }
}
```

В даному випадку перед тим як зберегти значення монет в користувача, вони проходять шифрування. Obscured класи побудовані таким чином, що при зміні значення, автоматично генерується ключ і шифрування відбувається повністю автоматично.

Після такого шифрування JSON файл матиме наступний вигляд:

JSON

```
{
  "ghI918TDWtSuobrtCjNBUw": {},
  "U1JfQRyCHKzYCf+/5eR7+w": {}
}
```

Зловмисник не знатиме що саме йому потрібно міняти і не зможе змінити дані всередині гри.

HoneyPot

Honeypot - це вид кібербезпеки, що представляє собою фальшивий об'єкт чи систему, призначений для виявлення, вивчення чи відстеження кіберзлочинців. Honeypot може бути як програмним, так і апаратним засобом, розташованим в мережі з метою привертання уваги потенційних зловмисників.

Основні характеристики honeypot включають:

- Фальшиві дані: Honeypot намагається видавати себе за реальну систему чи ресурс, ізолюючи його від решти реальних активів мережі.
- Захоплення даних: Honeypot може записувати та аналізувати всю взаємодію зловмисників, зокрема, їхні спроби атак та методи.
- Виявлення загроз: За допомогою honeypot можна виявляти нові та еволюціонуючі загрози, адже вони привертають атаки, які можуть вказувати на нові та не відомі види атак.
- Навчання та аналіз: Використання honeypot дозволяє вивчати методи та тактику зловмисників, а також покращувати стратегії кіберзахисту на основі отриманих даних.
- Попередження перед атакою: Honeypot може слугувати попередженням перед реальними атаками, дозволяючи вчасно реагувати та захищати реальні системи.

Хоча honeypot може бути корисним інструментом для виявлення та вивчення загроз, важливо враховувати його вплив на безпеку мережі, оскільки неправильна конфігурація може створити нові вразливості.

Для реалізації honeypot було використано фальшиву змінну з типом ObscuredString, яка зберігається разом з всіма даними користувача у одному JSON файлі. Для перевірки була написана система HoneyPotChecker, яка при запуску гри дешифрує значення змінної honeypot, якщо це значення було штучно змінене, програма буде знати про те, що зловмисник намагався штучно підмінити дані користувача.

Вигляд JSON файлу без залучення шифрування:

JSON

```
{
  "Coins": "25",
  "UserID": "Test",
  "HoneyPot": "HoneyPot"
}
```

Вигляд JSON файлу в зашифрованому форматі:

JSON

```
{
  "IidnL9io83NpnpxSYVee7A": {},
  "1xteGhnjvI6u9lOAZwdKpg": {},
  "32jEM3tVWbTlyDb8o8mRQZhXhmnyWumtFPMlfcShLhY": {}
}
```

В разі бажання зловмисником підмінити дані, він не знатиме яку колонку в JSON потрібно змінювати, тому буде пробувати кожну з них. В разі зміни колонки HoneyPot, програма зрозуміє що дані було пошкоджено.

Система моніторингу і скринінгу даних

Для того щоб обробляти завантаження файлів під час запуску гри і перевіряти їх на наявність змін було створено систему моніторингу InjectionDetector. Ця система управляє honeypot, всіма Obscured змінними і перевіряє їх на наявність змін. При спробі зловмисника підмінити дані, дана система це помітить при запуску гри і запустить процес, прописаний для випадків зміни даних. У даному випадку це видалення акаунту користувача.

Також дана система періодично перевіряє гру під час роботи програми на виявлення зовнішніх ін’єкцій.

"Зовнішні ін'єкції" в іграх вказують на використання зовнішніх програм або модифікацій для втручання в процес виконання гри та зміни її стану або поведінки. Це може включати в себе різні види втручань, такі як:

- Чити та Хаки: Використання сторонніх програм для отримання переваг у грі, таких як нескінченне здоров'я, безмежна амуніція, швидкість чи невидимість.
- Модифікація файлів гри: Зміна або заміна оригінальних файлів гри для внесення змін у графіку, звук, текстури чи інші аспекти гри.
- Боти та Автоматизовані скрипти: Використання програм або скриптів для автоматизації виконання завдань або отримання ресурсів у грі без активної участі гравця.
- Маніпуляції з мережею: Втручання в мережевий трафік гри для зміни даних, що передаються між клієнтом та сервером, з метою отримання переваг.

Ці зовнішні ін'єкції можуть створювати проблеми для ігор та їхніх розробників, оскільки вони порушують баланс гри та руйнують враження від гри для інших гравців. Розробники намагаються впроваджувати заходи безпеки та анти-чіти для уникнення такого типу втручань і забезпечення чесного та справедливого геймплею.

Дана система налаштована таким чином, що в ній прописано базові параметри геймлею, які не можуть відхилятися від норм під час «чистого» геймплею. Тобто якщо користувач не використовує сторонні програми, чити і т.д. програма не бачить у ньому потенційного зловмисника. В разі, якщо користувач робить зовнішню ін’єкцію, програма помітить зміни в базових налаштуваннях, таких як швидкість пересування гравця, ненормальне нарахування внутрішньоігрової валюти. Систему можна налаштувати на різні види запобігання таким діям. Якщо говорити про зовнішні ін’єкції, то у разі їх використання програма може просто закриватися для гравця, який використав їх, або ж наприклад видаляти акаунт.

3.2.2. Шифрування передачі даних на сервер

Для реалізації багатокористувацької частини гри було виділено окремий сервер та сервіс для керування гравцями та базами даних.

Playfab

PlayFab - це хмарна платформа для розробки та управління іграми. Вона надає різноманітні сервіси для розробників ігор, що дозволяють зосередитися на створенні цікавих ігор, а не на аспектах інфраструктури та управління. Основні можливості PlayFab включають:

- Управління гравцями: Реєстрація та управління гравців, збереження їхніх профілів, статистики та досягнень.
- Сервери: Потужність для розгортання та управління власними серверами для масштабованої багатокористувацької гри.
- Економіка та магазини: Системи внутрішньої економіки, включаючи магазини, валюти та можливості монетизації.
- Аналітика: Засоби для збору та аналізу даних гри, які допомагають розробникам зрозуміти поведінку гравців та вдосконалити геймплей.
- Багатокористувацькі сервіси: Розширені можливості для реалізації Багатокористувацьких функцій, таких як змагання, класифікації та соціальні можливості.

PlayFab спрощує ряд завдань, з якими розробники ігор можуть стикатися під час створення та управління іграми, дозволяючи їм більше уваги приділяти творчості та вдосконаленню геймплею.

Авторизація і реєстрація

PlayFab надає зручний механізм для реєстрації та авторизації гравців у грі. Розробники можуть використовувати PlayFab SDK для інтеграції цих функціональностей, щоб ефективно керувати обліковими записами гравців та забезпечити безпеку.

Для реєстрації гравця через PlayFab, розробники можуть використовувати відповідний метод, який дозволяє створити новий обліковий запис гравця та зберегти необхідну інформацію.

Авторизація в PlayFab зазвичай включає в себе використання токенів або інших механізмів для перевірки валідності гравця під час входу в гру. Розробники можуть використовувати методи авторизації PlayFab SDK для здійснення цього процесу.

Важливо зазначити, що в реалізації цих операцій розробники повинні дотримуватися найкращих практик забезпечення безпеки, зокрема, передавати дані зашифровано та слідкувати за правильністю перевірки ідентифікації гравців.

Для реєстрації користувачів було обрано оптимальний метод для невеликих ігр – використання ідентифікаційного номеру (ID) пристрою.

SystemInfo.deviceUniqueIdentifier

При першому вході в гру, система автоматично реєструє гравця за допомогою його айді на сервері, де в подальшому буде зберігати всі дані.

Unity Web Requests

В Unity, взаємодія з віддаленими ресурсами часто виконується за допомогою HTTP-запитів. Для цьогослід використовувати клас UnityWebRequest. Цей клас дозволяє створювати та відправляти HTTP-запити, обробляти відповіді та взаємодіяти з віддаленими ресурсами.

Основний процес виглядає приблизно так:

- Створення запиту

```
using UnityEngine;
using UnityEngine.Networking;
string url = "https://example.com/api/data";
UnityWebRequest request = UnityWebRequest.Get(url);
```

- Відправлення запиту

```
yield return request.SendWebRequest();
```

- Перевірка та обробка відповіді

```
if (request.result == UnityWebRequest.Result.Success)
{
   Debug.Log("Request successful! " + request.downloadHandler.text);
}
else
{
      Debug.LogError("Request failed: " + request.error);
}
```

Приклад POST запиту: CryptoConstHolder

```
string url = "https://example.com/api/post";
List<IMultipartFormSection> formData = new List<IMultipartFormSection>();
formData.Add(new
MultipartFormDataSection("field1=value1&field2=value2"));

UnityWebRequest request = UnityWebRequest.Post(url, formData);
yield return request.SendWebRequest();
```

Для роботи з відправкою і отриманням запитів було створено декілька відповідних класів.

APIConstHolder

Цей клас відповідає за зберігання констант з кінцевими точками запитів, для того щоб вони не були підмінені у разі ін’єкції.

Приклад застосування:

```
public const string RequestURL = "https://game-api.vercel.app/";
```

WebRequestManager

Цей клас відповідає за створення веб реквестів та отримання і обробку результатів.

```
IEnumerator PostRequest(string body, Action callback = null)
    {
        using(UnityWebRequest www = UnityWebRequest.Post(CryptoConstHolder.RequestURL,
body))
        {
            www.SetRequestHeader("Content-Type", "text/plain");
            yield return www.SendWebRequest();
            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.Log(www.error);
                RequestError.Instance.Show(www.downloadHandler.text);
            }
            else
            {
```

```
            Debug.Log(www.result);
            Debug.Log(www.downloadHandler.text);
            callback?.Invoke();
        }
    }
```

В даному прикладі продемонстровано роботу запитів. З APIConstHolder береться кінцева точка відправлення запиту. Також є можливість добавити Action на випадок успішного запиту, для того, щоб програма виконувала якийсь функціонал при отриманні успішного запиту.

AES Шифрування

AES (Advanced Encryption Standard) - це симетричний алгоритм шифрування, який використовується для захисту конфіденційності інформації. Введений Національним Інститутом Стандартів і Технологій США (NIST) в 2001 році, AES став прийнятим стандартом у багатьох країнах та для багатьох типів застосувань.

Основні характеристики AES:

- Симетричний Алгоритм: AES використовує один і той же ключ для як шифрування, так і розшифрування даних. Такий підхід називається симетричним, оскільки ключ може бути використаний в обох напрямках.
- Розмір Блоку та Ключа: AES працює з блоками даних розміром 128 біт. Ключ може бути 128, 192 або 256 біт.
- Кілька Раундів: Шифрування даних у AES включає низку раундів залежно від розміру ключа: 10 раундів для 128-бітного ключа, 12 раундів для 192-бітного ключа і 14 раундів для 256-бітного ключа.
- Змішування та Заміна: Кожен раунд AES включає в себе операції змішування та заміни байтів, що сприяє стійкості шифру до криптоаналізу.
- Стійкість: AES вважається дуже стійким до різних видів атак, включаючи атаки brute force, differential, та linear cryptanalysis.

AES широко використовується в сучасних системах шифрування, включаючи застосування в інформаційній безпеці, мережевих протоколах, протоколах забезпечення безпеки Wi-Fi, SSL/TLS та інших технологіях.

Під час розробки багатокористувацької гри не можна залишати серверну частину незахищену. Для цього було використано AES шифрування. Всі запити на сервер, та ті які відправляються із сервера користувачу підлягають шифруванні і дешифруванню. Ключ шифрування зберігається на окремо виділеному сервері, задля безпеки і захищеності від ін'єкцій з боку користувача. Кожен запит, який надсилається у Web Request проходить декілька раундів шифрування перед відправкою на сервер. Серверна сторона в свою чергу розшифровує даний запис і робить зміни.

Також в даній системі можна використовувати унікальний пароль, який теж буде зберігатися на сервері. Клієнт гри буде конвертувати його в байтовий код і робити з нього ключ для шифрування.

```
private static byte[] ConvertToKeyBytes(SymmetricAlgorithm algorithm, string password)
        {
            algorithm.GenerateKey();
            var keyBytes = Encoding.UTF8.GetBytes(password);
            var validKeySize = algorithm.Key.Length;
            if (keyBytes.Length != validKeySize)
            {
                var newKeyBytes = new byte[validKeySize];
                Array.Copy(keyBytes, newKeyBytes, Math.Min(keyBytes.Length, newKeyBytes.Length));
                keyBytes = newKeyBytes;
            }
            return keyBytes;
        }
```

За допомогою даного методу може бути згенерований ключ для шифрування. Кількість раундів може вибиратись довільно програмою, також ця кількість може оброблятись шифратором певну кількість разів і відправлятися в зашифрованому виді на сервер, в такому разі процес захисту даних буде більш ефективним.

Використання клієнтського шифрування для створення і приєднання до ігор

Багатокористувацькі ігри вимагають створення кімнат та лоббі для того, щоб створити гру. Гравці повинні мати змогу чесно приєднуватись до кімнат в залежності від їх типу. Наприклад, якщо в грі є 2 режими – один з грою на віртуальну валюту, інший – за реальну, треба розподілити цих гравців і не дати змогу зловмиснику зайти в іншу кімнату. Для цього було використано HashTable.

HashTable — це структура даних, яка реалізує асоціативний масив або словник, де дані зберігаються у вигляді пари "ключ-значення". У багатьох мов програмування, таких як Java, C#, Python та інші, існують вбудовані класи або бібліотечні модулі для роботи з хеш-таблицями.

Основні характеристики HashTable:

- Хеш-функція: Хеш-таблиці використовують хеш-функції для перетворення ключа у індекс масиву (бакету), де відбувається зберігання значення. Ефективність хеш-функції важлива для рівномірного розподілу ключів по бакетах.
- Бакети: Кожний індекс масиву відомий як бакет. У кожному бакеті може бути збережено багато значень. Колізії, тобто випадки, коли два різні ключі відображаються в один і той же бакет, вирішуються різними стратегіями (наприклад, методом ланцюгів або відкритого хешування).
- Ефективність: В середньому, операції вставки, вилучення та пошуку у HashTable мають часову складність O(1), але в найгіршому випадку може бути O(n), де n - кількість ключів.
- Підтримка операцій: Зазвичай HashTable підтримує такі операції, як вставка, вилучення та пошук за ключем.

- Розширення: Часто реалізації HashTable автоматично збільшують розмір масиву, коли відбувається переповнення, щоб зберегти ефективність.

Приклад використання в C#:

```
Hashtable hashtable = new Hashtable();

// Додавання пари ключ-значення
hashtable.Add("key1", "value1");

// Отримання значення за ключем
Console.WriteLine(hashtable["key1"]);

// Вилучення за ключем
hashtable.Remove("key1");
```

Хеш-таблиці дуже ефективні для операцій пошуку та вставки в середньому випадку, але важливо правильно обирати хеш-функцію та управляти колізіями для забезпечення їхньої ефективності.

Для прикладу, гравець попадає в загальне лоббі з іншими гравцями, обирає режим гри, в списку з доступними режимами йому буде відображатись лише ігри з його обраним режимом. Якщо ж наш гравець – зловмисник, він може за допомогою ін’єкції змінити під час сеансу режим гри і йому відображатимуться не його режими. Але в випадку використання HashTable його буде додано в таблицю з його кімнатами за допомогою унікального ідентифікатору і ніяка ін’єкція це не зможе змінити.

3.2.3 Дешифрування отриманих від сервера даних на клієнтській стороні

Дешифрування даних на стороні клієнта є більш важкою задачею, ніж шифрування і передавання даних, тому що ключ для дешифрування знаходиться саме на серверній частині і для повноцінного забезпечення цілісності даних потрібно використати більш дієві методи.

Багатопотокове дешифрування

"Багатопотокове дешифрування" це процес розшифрування інформації, який використовує багато паралельних потоків обчислень. Це може бути корисно в сучасних комп'ютерних системах з багатьма ядрами процесора, де кожен потік може працювати над частинкою дешифрування незалежно.

В контексті криптографії або інформаційної безпеки, багатопотокове дешифрування може використовуватися для прискорення процесу розкриття шифру. Такий підхід особливо ефективний при використанні спеціалізованих обчислювальних ресурсів, таких як графічні процесори (GPU) або спеціалізовані апаратні засоби.

Також для дешифрування можуть використовуватися не лише багатопотокові обчислення, але й інші методи оптимізації, такі як векторизація, розпаралелювання, оптимізації алгоритмів і так далі, залежно від конкретної реалізації та вимог проекту.

Крім того, що багатопотокове дешифрування знижує шанс атаки на інформацію, яка передається, це також впливає на продуктивність системи, яка робить дешифрування. Сервери зазвичай мають потужні процесори, які здатні дуже швидко дешифрувати і шифрувати інформації, а от пристрої користувача можуть різнитися і потрібно сприяти оптимізації процесу.

Багатопотокове дешифрування може бути корисним з кількох причин:

- Прискорення часу обробки: Застосування багатопотоковості дозволяє використовувати одночасно кілька обчислювальних ресурсів для дешифрування інформації. Це може призвести до значного зменшення часу, необхідного для завершення процесу дешифрування.
- Ефективне використання багатоядерних процесорів: У сучасних комп'ютерах часто використовують багатоядерні процесори. Багатопотокове дешифрування дозволяє розподіляти завдання між різними ядрами, що призводить до ефективнішого використання обчислювального потенціалу системи.
- Покращення продуктивності в обчислювальних задачах: У випадку великих обсягів даних або вимогливих за шифруванням алгоритмів багатопотокове

дешифрування може покращити продуктивність системи, дозволяючи одночасно обробляти кілька частин інформації.

- Покращення відгуку системи: У випадках, коли шифрована інформація використовується в реальному часі, наприклад, у відеоіграх чи потокових сервісах, багатопотокове дешифрування може допомогти забезпечити плавний ігровий процес чи стрімінг, зменшуючи затримки через більш швидке оброблення даних.
- Ефективне використання апаратного забезпечення: Застосування спеціалізованих обчислювальних пристроїв, таких як графічні процесори (GPU) чи апаратні пристрої для шифрування/дешифрування, може покращити ефективність багатопотокового дешифрування, оскільки ці пристрої часто розроблені з урахуванням паралельних обчислень.

Усі ці аспекти сприяють покращенню продуктивності та забезпеченню ефективного використання обчислювальних ресурсів для завдань дешифрування.

3.2 Тестування

3.3.1 Тестування системи захисту внутрішньо-ігрових даних

Для тестування системи захисту внутрішньо-ігрових даних потрібно спробувати змінити дані з пристрою.

Для цього було зроблено декілька спроб різними способами.

Зміна даних вручну

Для початку було проведену зміну даних без захисту в ігровому середовищі. Для цього потрібно перейти в регістр платформи, на якій встановлена гра. В моєму випадку це Windows. Для цього запускаємо в командному рядку команду **regedit** і відкриваємо регістр даних користувача. Далі переходимо в **HKEY_CURRENT_USER/SOFTWARE/Unity/UnityEditor/<Назва гри>** і знаходимо потрібні нам дані.

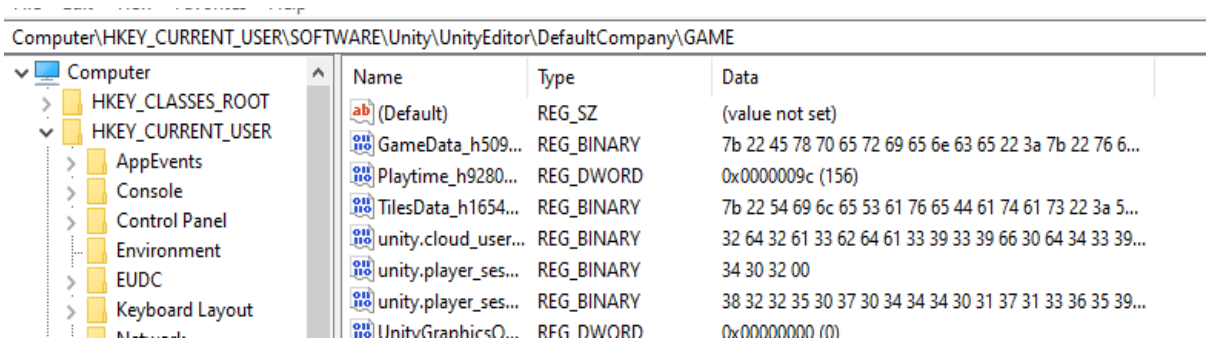

Рис. 3.1. Регістр даних користувача в системі

Як можна побачити, дані легко знаходяться, що не поставить зловмиснику великої проблеми. Далі ми переходимо, наприклад у вкладку GameData і змінюємо значення параметрів.

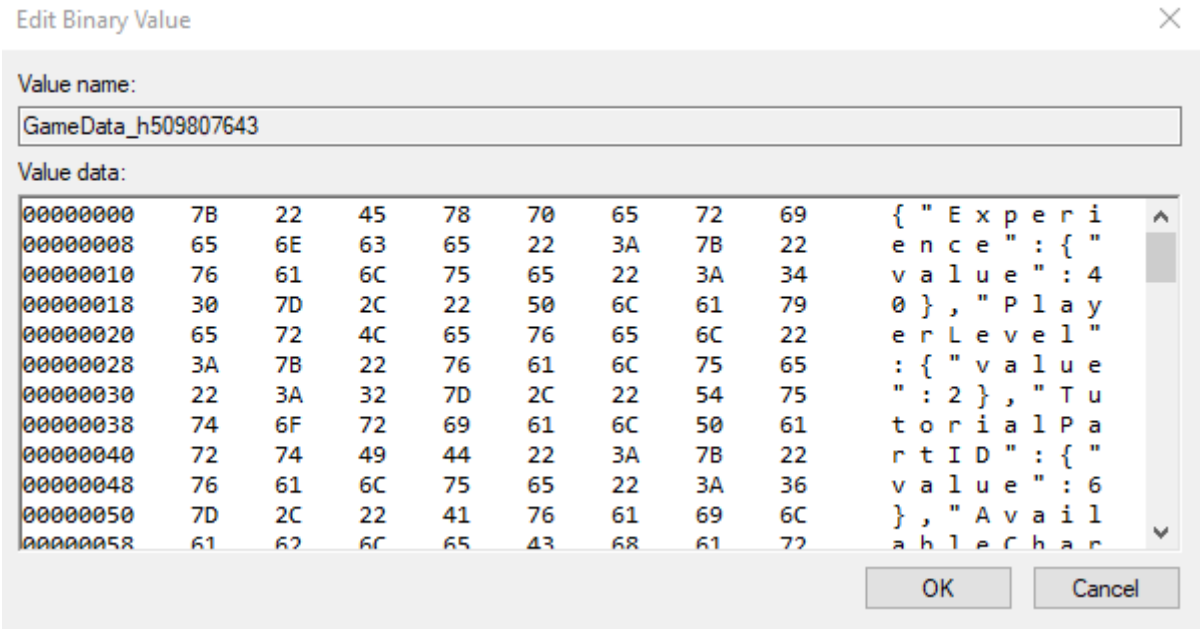

Рис. 3.2. Приклад даних користувача в розшифрованому вигляді

Як можемо бачити, змінити дані не є проблемою взагалі. Любий користувач може змінити такі значення як Experience, PlayerLevel і тд.

Далі спробуємо зробити таку ж дію, але з встановленими обфускованими змінними і подивимось, наскільки процес зміни даних стане важчим для зловмисника.

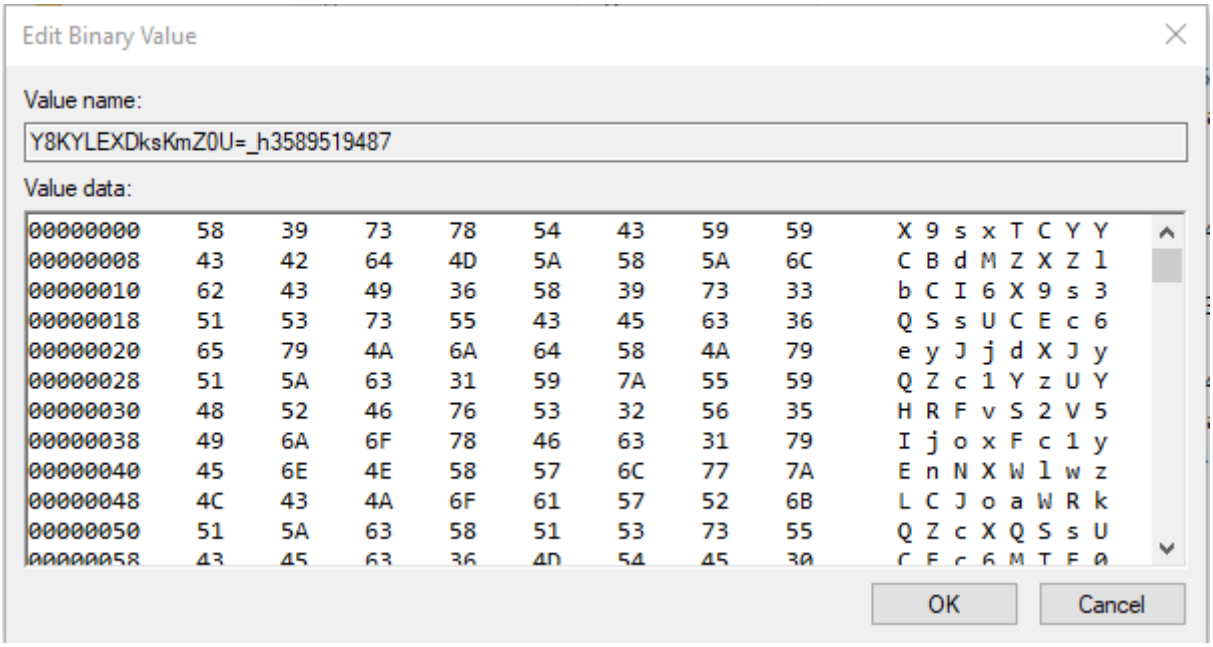

Рис. 3.3. Приклад даних користувача в зашифрованому вигляді

Як можна побачити, всі дані знаходяться в зашифрованому вигляді і зловмиснику немає як змінити їх. У випадку зміни любого рядка, дані буде пошкоджено і гра не буде запускатись, тому бачимо, що обфускація працює належним чином.

HoneyPot

Існує ймовірність що зловмисник вгадає рядок, де можна змінити дані користувача і скористається цим. Для забезпечення від даного сценарію було введено декілька honeypot в ігрове середовище. Це невеликі файли з даними, які ніяким чином не впливають на геймплей, але динамічно змінюються, мають зашифрований вигляд і таким чином можуть привабити зловмисника, як потенційно важливі дані користувача. Для перевірки було створено декілька honeypot.

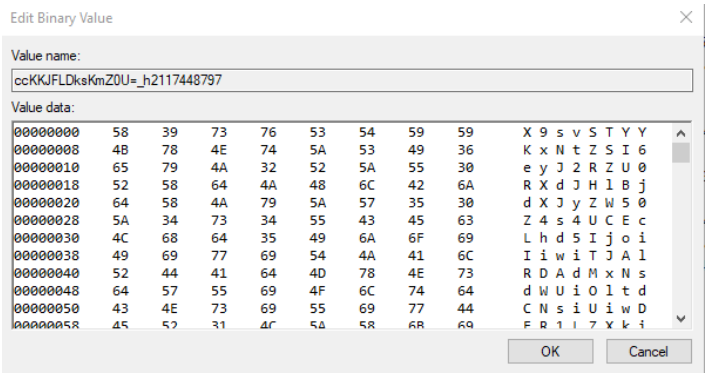

Рис. 3.4. Приклад використання HoneyPot

На вигляд це звичайні дані користувача, але зловмисник не матиме змогу дізнатись що це лише приманка, яка визначить, що він намагався змінити дані користувача. Після зміни даних, було перейдено в гру і ось результат, який отримає зловмисник, який змінить дані в honeypot.

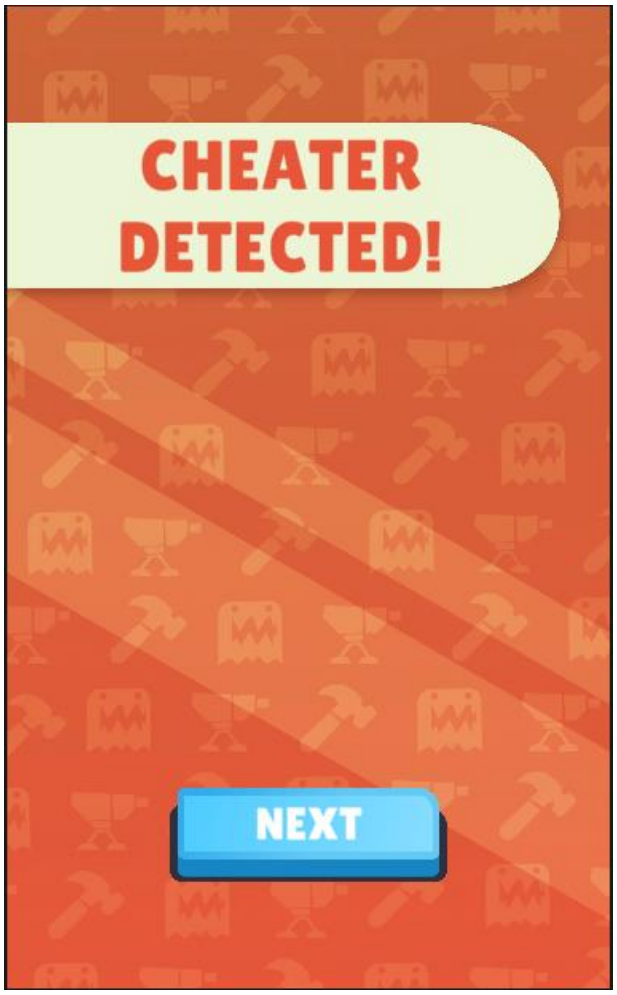

Рис. 3.5. Приклад реагування системи на зміну даних користувача

3.3.2 Тестування системи передачі даних на сервер

Тестування безпеки системи передачі даних на сервер включає в себе ряд дій для визначення і виправлення потенційних уразливостей, які можуть використовуватися для несанкціонованого доступу, атак або витоку конфіденційної інформації. Ось деякі основні типи тестів безпеки, які можуть бути використані:

- Тестування перехоплення трафіку (Traffic Interception Testing):

  Мета: Перевірка вразливостей передачі даних через незахищені канали.

Як проводити: Спроби перехопити та аналізувати трафік між клієнтом і сервером для виявлення можливих загроз безпеці.

- Тестування вразливостей введення (Input Vulnerability Testing):

Мета: Виявлення можливостей введення некоректних або шкідливих даних, які можуть викликати атаки вривання.

Як проводити: Введення спеціально сформованих даних через різні введені точки та вивчення реакції системи.

- Тестування вразливостей автентифікації та авторизації (Authentication and Authorization Vulnerability Testing):

Мета: Перевірка недоліків у процесах автентифікації та авторизації, які можуть дозволити несанкціонований доступ.

Як проводити: Спроби обхід автентифікації, випробовування неправильних ролей користувачів та інші техніки.

- Тестування вразливостей безпеки сесій (Session Security Vulnerability Testing):

Мета: Визначення можливих уразливостей у керуванні сесіями та захисті інформації про сесії.

Як проводити: Спроби перехопити, взламати або викрасти ідентифікатори сесій та їхні дані.

- Тестування витоку інформації (Data Leakage Testing):

Мета: Виявлення можливих точок витоку конфіденційної інформації під час передачі даних.

Як проводити: Моделювання ситуацій, що можуть призводити до витоку даних, та вивчення, як система реагує на такі ситуації.

Варто зазначити, що тестування безпеки є неперервним процесом, і його слід проводити регулярно для забезпечення високого рівня захисту системи передачі даних.

Важливо підкреслити, що ви можете тестувати захист передачі даних на сервер лише на власних серверах або на серверах з дозволу власників системи. Без

відповідних дозволів проведення тестів на системах, до яких ви не маєте доступу, є незаконним і недопустимим.

- Тестування шифрування трафіку: Спробуйте використовувати інструменти, такі як Wireshark або Burp Suite, для перехоплення та аналізу трафіку. Перевірте, чи дані передаються у зашифрованому вигляді, наприклад, через протокол HTTPS.

  Для цього тестування було використано Wireshark. Під час геймплею було проведено аналіз портів і передачі даних, для того, щоб визначити, чи є вразливості.

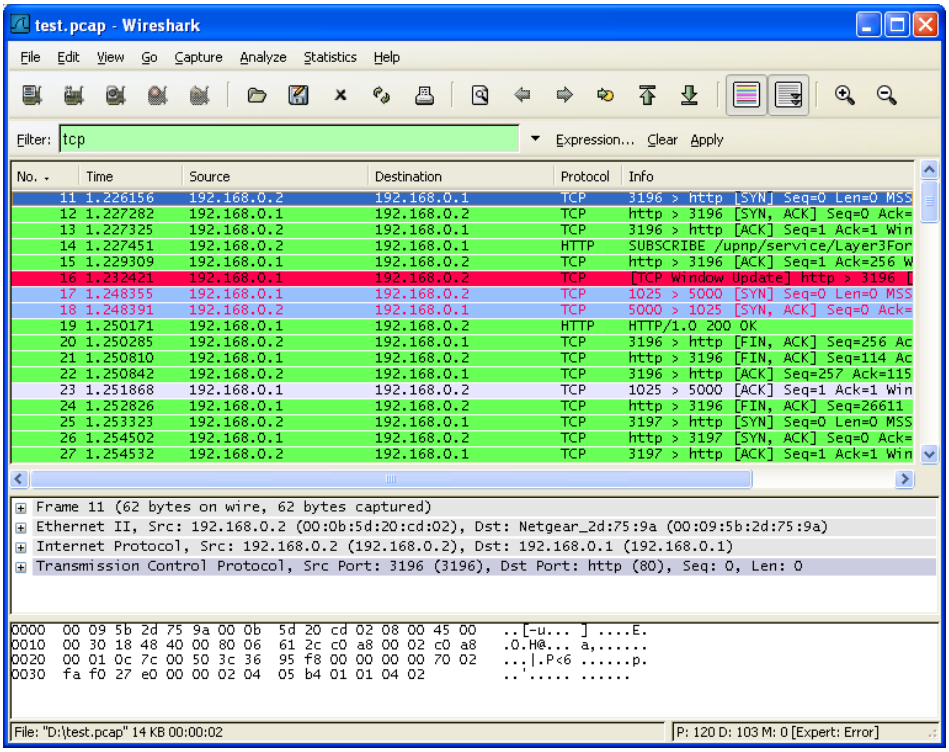

Рис. 3.6. Використання WireShark для перевірки мережевого трафіку гри

Як можна побачити, майже всі дані та порти є захищеними, лише декілька передач даних були незахищеними, але після ретельного аналізу виявилось, що це передача сокетних даних самого геймплею, яка відбувається кожну секунду і ніяк не повпливає на геймплей під час спроб її перехопити.

- Тестування автентифікації та авторизації: Спробуйте невдалий вхід або атаку на авторизацію для перевірки, як система реагує на неправильні дані автентифікації та чи виключається неправомірний доступ.

Як вже зазначалось, логін для користувачів при реєстрації встановлюється за допомогою унікального ідентифікатору пристрою користувача. Він зберігається у внутрішніх даних користувача і є зашифрованим і захищеним honeypot. Уявімо ситуацію, коли зловмисник через root права змінює собі унікальний ідентифікатор і намагається зайти в гру під іншим логіном.

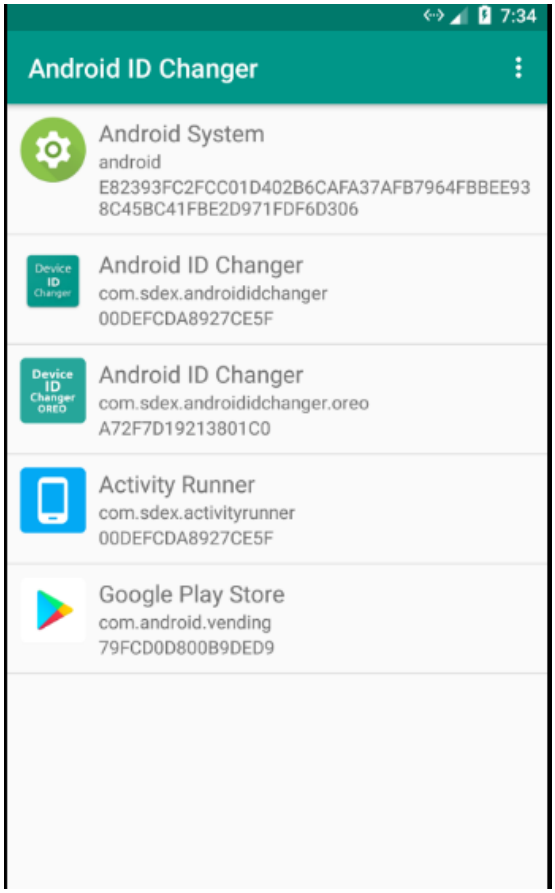

Рис. 3.7. Приклад використання сторонніх програм для зміни ідентифікатора пристрою

Дана програма дозволяє тимчасово змінити унікальний ідентифікатор для відображення у програмах.

При запуску гри, система звіряє захищений айді, який збережений в даних користувача з айді пристрою. В разі розбіжностей система не пускатиме користувача в гру і надішле на сервер повідомлення з інформацією про несанкціонований вхід, де модератори зможуть прийняти піри безпеки.

Рис. 3.8. Реагування системи на зміну ідентифікатора пристрою

3.3 Оцінка ефективності та порівняльні таблиці запропонованого рішення з готовими рішеннями

Після аналізу та тестування запропонованих рішень щодо захисту багатокористувацької гри, можна прийти до висновку, що даний захист є цілком надійним від спроб зміни внутрішньоігрової логіки, зміни даних, або перехоплення даних, які відправляються на сервер. Даної системи вистачить, щоб забезпечити «чистий» геймплей невеликій багатокористувацькій грі.

Після аналізу якості шифрування та його відповідності стандартам безпеку, не було виявлено великих вразливостей стосовно захисту даних користувача.

Аналіз передачі трафіку виявився досить непоганим, а запропоноване рішення цілком захищає невелику кількість передачі трафіку, що підходить для невеликих багатокористувацьких ігор.

Також, запропонована система авторизації та реєстрації не дасть змогу зловмисникам скористатись чужим акаунтом та буде запобігати ботам у грі.

Система підготована до всіх спроб несанкціонованих доступів та змін даних і в ній прописано всі виключення, які дозволяють швидко і якісно реагувати на атаки.

Таблиця 3.1.

Зведені дані порівняння запропонованого рішення з готовими рішенями захисту багатокористувацьких ігор

	Запропоноване рішення	Riot Vanguard	BattlEye	EAC	Anti-Cheat Toolkit
Підтримка OS					
Windows	+	+	+	+	+
Linux	+	-	+	+	-
iOS/MacOS	+	-	+	+	+
Рівень спостереження					
Системний	+	+	+	+	+
Мережевий	+	+	+	+	-
Відкритість	+	-	-	+	+
Гнучкість налаштування	+	-	-	+	+
Вплив на продуктивність	-	+	+	+	-

Продовження таблиці 3.1.

Легкість інтеграції	-	-	-	+	+
Можливість використання під власним брендом	+	-	-	+	+
Ведення звітності	+	-	-	-	-
Наявність захисту мережевих даних	+	+	+	-	-
Захист від зовнішніх ін’єкцій	+	+	-	-	+
Прописані виключення для запобігання атакам	+	-	-	-	-
Можливість кастомізації виключень	+	-	-	-	+

3.5 Висновки до розділу

В пункті 3.1. було проведено аналіз ігрових рушіїв, було визначено їх плюси та мінуси стосовно розробки багатокористувацької гри а також визначено найкращий варіант. Також було проведено аналіз доступних мов програмування для ігрового рушія та було обрано оптимальний варіант з найкращими запропонованими рішеннями.

В пункті 3.2 було описано основний функціонал системи безпеки даних у розробленому рішенні. Було описано порядок інтеграції рішення, плюси та мінуси

використання різних способів шифрування та захисту передачі даних між сервером та клієнтом.

В пункті 3.3 було проведено тестування готового рішення. Під час тестування було перевірено різні типи атака, починаючи від ручної зміни даних користувача і закінчуючи впливом на мережевий трафік гри. Було проведено аналіз тестів і визначено переваги та недоліки готового рішення.

В пункті 3.4. було проведено аналіз і порівняння розробленого рішення з готовими рішеннями. Було наведено всі недоліки та переваги всіх рішень.

РОЗДІЛ 4
ОХОРОНА НАВКОЛИШНЬОГО СЕРЕДОВИЩА

Завданням законодавства про охорону навколишнього природного середовища є регулювання відносин у галузі охорони, використання і відтворення природних ресурсів, забезпечення екологічної безпеки, запобігання і ліквідації негативного впливу господарської та іншої діяльності на навколишнє природне середовище, збереження природних ресурсів, генетичного фонду живої природи, ландшафтів та інших природних комплексів, унікальних територій та природних об'єктів, пов'язаних з історико-культурною спадщиною.

Сучасне природоохоронне законодавство визначається рядом завдань і принципів, які спрямовані на збереження та раціональне використання природних ресурсів, охорону біорізноманіття і забезпечення сталого розвитку. Загальні завдання природоохоронного законодавства можуть включати:

- Збереження біорізноманіття:
    - Захист екосистем, включаючи ліси, водні басейни, ґрунти та інші природні об'єкти.
    - Заохочення створення та управління природоохоронними територіями, такими як національні парки, заповідники і біосферні резервати.
- Забезпечення сталого використання природних ресурсів:
    - Регулювання видобутку і використання водних, лісових, мінеральних та інших природних ресурсів з метою запобігання їх надмірного вичерпання.
    - Встановлення принципів відновлюваності та сталого використання природних ресурсів.
- Боротьба з забрудненням та іншими формами негативного впливу:
    - Регулювання викидів забруднюючих речовин у повітря, воду і ґрунт.
    - Встановлення стандартів якості довкілля та нормативів для підприємств і промислових об'єктів.
- Захист вразливих видів і екосистем:

- Розробка програм і заходів з охорони рідкісних та загрожених видів.
- Забезпечення регулярного моніторингу та оцінки стану природних об'єктів.

- Залучення громадськості та сприяння екологічній освіті:
  - Забезпечення участі громадськості в прийнятті рішень з питань природоохорони.
  - Здійснення заходів з підвищення екологічної свідомості та освіти.

Ці завдання можуть варіюватися в залежності від конкретних умов кожної країни чи регіону. Важливо, щоб природоохоронне законодавство було ефективним, а його виконання контролювалося і сприялося сталому розвитку та збереженню природи.

Сучасне природоохоронне законодавство є важливим інструментом для управління взаємовідносинами між суспільством та природною середою. Воно визначає рамки для сталого використання та охорони природних ресурсів, забезпечуючи гармонію між потребами сучасного суспільства та необхідністю збереження екологічної цілісності.

Одним із основних завдань природоохоронного законодавства є стимулювання сталого розвитку, де економічний розвиток здійснюється, не ведучи до надмірного використання ресурсів чи знищення природних екосистем. Це означає визначення параметрів використання природних ресурсів, які не перевищують їх природної здатності до відновлення.

Однак законодавство також має вирішувати завдання охорони природної різноманітності, що стає все більш уразливою через антропогенний тиск. Це може включати в себе створення охоронних зон, резерватів та механізмів контролю за експлуатацією природних екосистем.

Закони також повинні враховувати проблему забруднення довкілля, регулюючи видобуток та викиди забруднюючих речовин у повітря, воду та ґрунт. Вони створюють стандарти якості довкілля для забезпечення здоров'я населення та збереження природного середовища.

Дуже важливим елементом є залучення громадськості до процесу розробки та виконання природоохоронних заходів. Співпраця з громадськістю сприяє підвищенню екологічної свідомості, створенню екологічно відповідального суспільства та забезпеченню більш ефективної реалізації заходів з охорони навколишнього середовища.

Враховуючи сучасні виклики, такі як зміни клімату та екологічні кризи, природоохоронне законодавство постійно вдосконалюється для забезпечення адаптації до нових умов та збереження природних ресурсів для майбутніх поколінь.

У контексті інформаційних технологій (ІТ) сучасне природоохоронне законодавство має ряд завдань, спрямованих на раціональне використання технологій для збереження природи та зменшення негативного впливу на довкілля. Основні завдання в айті сфері включають:

- Цифровий моніторинг довкілля: Впровадження технологій для постійного моніторингу стану природних ресурсів, включаючи водні та лісові ресурси, які дозволяють отримувати точні та актуальні дані.
- Геоінформаційні системи (ГІС): Використання ГІС для аналізу просторових даних і визначення оптимальних місць для створення охоронних зон або резерватів.
- Електронна звітність та документообіг: Забезпечення ефективного збору, обробки та аналізу електронних звітів та документів, пов'язаних з природоохоронними заходами.
- Розробка екологічних додатків: Створення мобільних застосунків та онлайн-інструментів, які допомагають громадянам брати участь у природоохоронних ініціативах та сприяють екологічній свідомості.
- Контроль викидів та забруднення: Використання сучасних технологій для контролю та відстеження викидів забруднюючих речовин у повітря, воду та ґрунт.

- Розвиток інноваційних рішень: Підтримка ініціатив, спрямованих на розробку та впровадження новітніх технологій для зменшення впливу людської діяльності на навколишнє середовище.
- Кібербезпека природоохоронних систем: Захист інформаційних систем, що використовуються для моніторингу та управління природоохоронними заходами, від кіберзагроз та несанкціонованого доступу.
- Електронна освіта та інформаційна освіта: Поширення інформації про екологічні питання через онлайн-курси та інші електронні ресурси для підвищення екологічної свідомості громадян та фахівців в галузі ІТ.

ВИСНОВКИ

Підсумовуючи викладене в даній роботі, можна стверджувати, що захист даних користувачів в багатокористувацьких іграх відіграє все більшу роль. Багатокористувацькі ігри щоденно набувають все більшої популярності і користувачі все більше стикаються із зловмисниками в іграх. Це може бути звичайне читерство, але випадки доходять до викрадення персональних даних і навіть коштів гравця. Зрозуміло, що великі компанії виробляють готові рішення для своїх ігор, вони мають безліч ресурсів для захисту своїх користувачів, але всі системи вони мають у закритому доступі, що ставить звичайних розробників у глухий кут, де вони не мають нормального готового рішення для захисту своєї багатокористувацької гри.

Відповідно, на основі даних міркувань, було проаналізовано основні види атак на багатокористувацькі ігри і розглянуто головні методи для виявлення і боротьби з атаками на дані користувачів в багатокористувацьких іграх.

Дані методи було протестовано, показано результати захисту даних на пристрої користувача а також у мережі. Також було імплементовано системи спостереження та логування атак, для того, щоб у розробників була можливість адаптуватися до вразливостей та приймати відповідні рішення.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Valve Anti Cheat [Online]. – Available: https://en.wikipedia.org/wiki/Valve_Anti-Cheat (in English)
2. Riot Vanguard [Online]. – Available: https://www.gamechampions.com/en/blog/valorant-anti-cheat-vanguard/ (in English)
3. Easy-Anti Cheat [Online]. – Available: https://www.easy.ac/en-us/#:~:text=Easy%20Anti%2DCheat%20is%20the,of%20hybrid%20anti%E2%80%93cheat%20mechanisms. (in English)
4. BattleEye Anti-Cheat [Online]. – Available: https://www.battleye.com/about/ (in English)
5. Anti-Cheat toolkit [Online]. – Available: https://codestage.net/uas/actk/#:~:text=Anti%2DCheat%20Toolkit%20(ACTk),anti%2Dcheat%20expertise%20and%20experience.https://ap.uu.edu.ua/article/262 (in English)
6. Multiplayer Games Basics [Online]. – Available: https://vokigames.com/ua/vstup-do-istoriyi-gejmdyzajnu-chastyna-4-bagatokorystuvaczki-igry-ta-chastkova-struktura/ (in Ukrainian)
7. Data Protection [Online]. – Available: https://www.eset.com/ua/about/newsroom/blog/data-protection/opasnoye-razvlecheniye-pochemu-sleduyet-otkazatsya-ot-piratskikh-igr/ (in English)
8. Game Cheating [Online]. – Available: https://uk.wikipedia.org/wiki/%D0%A7%D0%B8%D1%82_%D0%B2_%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD_%D1%96%D0%B3%D1%80%D0%B0%D1%85 (in English)
9. Data security [Online]. – Available: https://iitd.com.ua/news/shho-take-informacijna-bezpeka-pidpriiemstva-ta-jaki-osnovni-zasadi-zahistu-danih-isnujut/ (in Ukrainian)

10. Basics of data security implementation [Online]. – Available: https://core.ac.uk/download/pdf/81588059.pdf (in English)
11. Грищук Р. Джерела первинних даних для розроблення шаблонів потенційно небезпечних кібератак / Р. Грищук, В. Охрімчук, В. Ахтирцева // Захист інформації. – 2016. – № 18(1). – С. 21-29.
12. Корпань Я.В. Класифікація загроз інформаційній безпеці в комп'ютерних системах при віддаленій обробці даних / Я.В. Корпань // Реєстрація, зберігання і обробка даних. – 2015. – № 17(2). – С. 39-46.
13. Cybersecurity in Games [Online]. – Available: https://gurt.org.ua/articles/34602/ (in English)
14. Cyberattacks and their prevention [Online]. – Available: https://www2.deloitte.com/ua/uk/pages/risk/articles/beneath-the-surface-of-a-cyberattack.html (in English)
15. Game development baiscs [Online]. – Available: http://www.dy.nayka.com.ua/pdf/1_2022/4.pdf (in Ukrainian)
16. QuantSecurity basics [Online]. – Available: https://futuro.in.ua/news/307-yak-kvantovi-kompyutery-mozhut-zruynuvaty.html (in Ukrainian)
17. Blockchain techiques in cybersecurity [Online]. – Available: https://academy.binance.com/uk/articles/what-is-blockchain-and-how-does-it-work (in English)
18. Blockchain security in gamedev [Online]. – Available: https://www.h-x.technology/ua/blog-ua/how-secure-internet-of-things-with-blockchain-ua (in English)
19. Cybersecurity in Games [Online]. – Available: https://goal-int.org/ponyattya-ta-zmist-kibershpigunstva/ (in English)
20. What is a DDoS attack? [Online]. – Available: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-a-ddos-attack (in English)

21. What is Anti-Cheat? [Online]. – Available: https://www.schellman.com/blog/cybersecurity/what-is-anti-cheat (in English)
22. Best Anti-Cheat software [Online]. – Available: https://www.makeuseof.com/best-anti-cheat-software/ (in English)
23. How to secure mobile games? [Online]. – Available: https://www.linkedin.com/advice/0/how-do-you-secure-your-mobile-game-skills-mobile-technology (in English)
24. Franiatte M. Anti-Cheating Note and Solution: Being Competitive in Black Ops II Multiplayer / Michael Franiatte. – Brussels: Michael Franiatte, 2014. – 236 c. – (1).
25. Gregory J. Game Engine Architecture / Jason Gregory. – United States: CRC Press, 2014. – 1052 c. – (2).
26. Problems with Security in Games [Online]. – Available: https://ap.uu.edu.ua/article/262 (in Ukrainian)

Фрагмент вихідного коду програмного засобу

```
using Common;
 using Detectors;
 using System;
 using UnityEngine;
 using System.Runtime.InteropServices;
 using UnityEngine.Serialization;
 using Utils;
 [Serializable]
 public partial struct ObscuredFloat : IObscuredType
 {
#if UNITY_EDITOR
        public string migratedVersion;
#endif
        [SerializeField] internal int currentCryptoKey;
        [SerializeField] internal int hiddenValue;
        [SerializeField] [FormerlySerializedAs("hiddenValue")]
#pragma warning disable 414
        private ACTkByte4 hiddenValueOldByte4;
#pragma warning restore 414

        [SerializeField] internal float fakeValue;
        [SerializeField] internal bool fakeValueActive;
        [SerializeField] internal bool inited;

        private ObscuredFloat(float value)
        {
```

```
                currentCryptoKey = GenerateKey();
                hiddenValue = Encrypt(value, currentCryptoKey);
                hiddenValueOldByte4 = default;

    #if UNITY_EDITOR
                fakeValue = value;
                fakeValueActive = true;
                migratedVersion = null;
    #else
                var detectorRunning = ObscuredCheatingDetector.ExistsAndIsRunning;
                fakeValue = detectorRunning ? value : 0f;
                fakeValueActive = detectorRunning;
    #endif
                inited = true;
            }
            public static int Encrypt(float value, int key)
            {
                return FloatIntBytesUnion.XorFloatToInt(value, key);
            }
            public static float Decrypt(int value, int key)
            {
                return FloatIntBytesUnion.XorIntToFloat(value, key);
            }
            public static int MigrateEncrypted(int encrypted, byte fromVersion = 0, byte toVersion = 2)
            {
                return FloatIntBytesUnion.Migrate(encrypted, fromVersion, toVersion);
            }
```

```
        public static ObscuredFloat FromEncrypted(int encrypted, int key)
        {
            var instance = new ObscuredFloat();
            instance.SetEncrypted(encrypted, key);
            return instance;
        }
        public static int GenerateKey()
        {
            return RandomUtils.GenerateIntKey();
        }

        private static bool Compare(float f1, float f2)
        {
            var epsilon = ObscuredCheatingDetector.ExistsAndIsRunning ?
                ObscuredCheatingDetector.Instance.floatEpsilon :
float.Epsilon;
            return NumUtils.CompareFloats(f1, f2, epsilon);
        }


        public int GetEncrypted(out int key)
        {
            if (!inited)
                Init();

            key = currentCryptoKey;
            return hiddenValue;
        }
```

```
public void SetEncrypted(int encrypted, int key)
{
	inited = true;
	hiddenValue = encrypted;
	currentCryptoKey = key;

	if (ObscuredCheatingDetector.ExistsAndIsRunning)
	{
		fakeValueActive = false;
		fakeValue = InternalDecrypt();
		fakeValueActive = true;
	}
	else
	{
		fakeValueActive = false;
	}
}
public float GetDecrypted()
{
	return InternalDecrypt();
}
public void RandomizeCryptoKey()
{
	var decrypted = InternalDecrypt();
	currentCryptoKey = GenerateKey();
	hiddenValue = Encrypt(decrypted, currentCryptoKey);
}

private float InternalDecrypt()
```

```
            {
                if (!inited)
                {
                    Init();
                    return 0;
                }

    #if ACTK_OBSCURED_AUTO_MIGRATION
                if (hiddenValueOldByte4.b1 != 0 ||
                   hiddenValueOldByte4.b2 != 0 ||
                    hiddenValueOldByte4.b3 != 0 ||
                    hiddenValueOldByte4.b4 != 0)
                {
                    var union = new FloatIntBytesUnion {b4 =
hiddenValueOldByte4};
                    union.b4.Shuffle();
                    hiddenValue = union.i;

                    hiddenValueOldByte4.b1 = 0;
                    hiddenValueOldByte4.b2 = 0;
                    hiddenValueOldByte4.b3 = 0;
                    hiddenValueOldByte4.b4 = 0;
                }
    #endif

                var decrypted = Decrypt(hiddenValue, currentCryptoKey);
                if (ObscuredCheatingDetector.ExistsAndIsRunning &&
fakeValueActive && !Compare(decrypted, fakeValue))
                {
    #if ACTK_DETECTION_BACKLOGS
```

```
                    Debug.LogWarning(ObscuredCheatingDetector.LogPrefix +
"Detection backlog:\n" +
                                     $"type: {nameof(ObscuredFloat)}\n" +
                                     $"decrypted: {decrypted}\n" +
                                     $"fakeValue: {fakeValue}\n" +
                                     $"epsilon:
{ObscuredCheatingDetector.Instance.floatEpsilon}\n" +
                                     $"compare:          {Compare(decrypted,
fakeValue)}");
        #endif

         ObscuredCheatingDetector.Instance.OnCheatingDetected(this,          decrypted,
fakeValue);
                }

                return decrypted;
            }

            private void Init()
            {
                currentCryptoKey = GenerateKey();
                hiddenValue = Encrypt(0, currentCryptoKey);
                fakeValue = 0;
                fakeValueActive = false;
                inited = true;
            }

            //! @cond

            #region obsolete
```

```
        [Obsolete("This API is redundant and does not perform any actions. It will be removed in future updates.")]
        public static void SetNewCryptoKey(int newKey) {}

        [Obsolete("This API is redundant and does not perform any actions. It will be removed in future updates.")]
        public void ApplyNewCryptoKey() {}

        [Obsolete("Please use new Encrypt(value, key) API instead.", true)]
        public static int Encrypt(float value) { throw new Exception(); }

        [Obsolete("Please use new Decrypt(value, key) API instead.", true)]
        public static float Decrypt(int value) { throw new Exception(); }

        [Obsolete("Please use new FromEncrypted(encrypted, key) API instead.", true)]
        public static ObscuredFloat FromEncrypted(int encrypted) { throw new Exception(); }

        [Obsolete("Please use new GetEncrypted(out key) API instead.", true)]
        public int GetEncrypted() { throw new Exception(); }

        [Obsolete("Please use new SetEncrypted(encrypted, key) API instead.", true)]
        public void SetEncrypted(int encrypted) {}

        #endregion

        //! @endcond
```

```
[StructLayout(LayoutKind.Explicit)]
internal struct FloatIntBytesUnion
{
    [FieldOffset(0)] private float f;
    [FieldOffset(0)] internal int i; // need to be internal for ACTK_OBSCURED_AUTO_MIGRATION
    [FieldOffset(0)] internal ACTkByte4 b4; // need to be internal for ACTK_OBSCURED_AUTO_MIGRATION

    public static int Migrate(int value, byte fromVersion, byte toVersion)
    {
        var u = FromInt(value);

        if (fromVersion < 2 && toVersion == 2)
            u.b4.Shuffle();

        return u.i;
    }

    internal static int XorFloatToInt(float value, int key)
    {
        return FromFloat(value).Shuffle(key).i;
    }

    internal static float XorIntToFloat(int value, int key)
    {
        return FromInt(value).UnShuffle(key).f;
    }
```

```
            private static FloatIntBytesUnion FromFloat(float value)
            {
                return new FloatIntBytesUnion { f = value};
            }

            private static FloatIntBytesUnion FromInt(int value)
            {
                return new FloatIntBytesUnion { i = value};
            }

            private FloatIntBytesUnion Shuffle(int key)
            {
                i ^= key;
                b4.Shuffle();

                return this;
            }

            private FloatIntBytesUnion UnShuffle(int key)
            {
                b4.UnShuffle();
                i ^= key;

                return this;
            }
        }
    }

public static class AESEncryptor
    {
```

```
        /// <summary>
        /// A class containing AES-encrypted text, plus the IV value required to decrypt it (with the correct password)
        /// </summary>
        public struct AESEncryptedText
        {
            public string IV;
            public string EncryptedText;
        }

        /// <summary>
        /// Encrypts a given text string with a password
        /// </summary>
        /// <param name="plainText">The text to encrypt</param>
        /// <param name="password">The password which will be required to decrypt it</param>
        /// <returns>An AESEncryptedText object containing the encrypted string and the IV value required to decrypt it.</returns>
        public static AESEncryptedText Encrypt(string plainText, string password)
        {
            using (var aes = Aes.Create())
            {
                aes.GenerateIV();
                aes.Key = ConvertToKeyBytes(aes, password);

                var textBytes = Encoding.UTF8.GetBytes(plainText);

                var aesEncryptor = aes.CreateEncryptor();
                var encryptedBytes = aesEncryptor.TransformFinalBlock(textBytes, 0, textBytes.Length);
```

```
                    return new AESEncryptedText
                    {
                        IV = Convert.ToBase64String(aes.IV),
                        EncryptedText = Convert.ToBase64String(encryptedBytes)
                    };
                }
            }

            /// <summary>
            /// Decrypts an AESEncryptedText with a password
            /// </summary>
            /// <param name="encryptedText">The AESEncryptedText object to decrypt</param>
            /// <param name="password">The password to use when decrypting</param>
            /// <returns>The original plainText string.</returns>
            public static string Decrypt(AESEncryptedText encryptedText, string password)
            {
                return Decrypt(encryptedText.EncryptedText, encryptedText.IV, password);
            }

            /// <summary>
            /// Decrypts an encrypted string with an IV value password
            /// </summary>
            /// <param name="encryptedText">The encrypted string to be decrypted</param>
```

```
        /// <param name="iv">The IV value which was generated when the text was encrypted</param>
        /// <param name="password">The password to use when decrypting</param>
        /// <returns>The original plainText string.</returns>
        public static string Decrypt(string encryptedText, string iv, string password)
        {
            using (Aes aes = Aes.Create())
            {
                var ivBytes = Convert.FromBase64String(iv);
                var encryptedTextBytes = Convert.FromBase64String(encryptedText);

                var decryptor = aes.CreateDecryptor(ConvertToKeyBytes(aes, password), ivBytes);
                var decryptedBytes = decryptor.TransformFinalBlock(encryptedTextBytes, 0, encryptedTextBytes.Length);

                return Encoding.UTF8.GetString(decryptedBytes);
            }
        }

        // Ensure the AES key byte-array is the right size - AES will reject it otherwise
        private static byte[] ConvertToKeyBytes(SymmetricAlgorithm algorithm, string password)
        {
            algorithm.GenerateKey();
```

```
                var keyBytes = Encoding.UTF8.GetBytes(password);
                var validKeySize = algorithm.Key.Length;

                if (keyBytes.Length != validKeySize)
                {
                    var newKeyBytes = new byte[validKeySize];
                    Array.Copy(keyBytes, newKeyBytes,
Math.Min(keyBytes.Length, newKeyBytes.Length));
                    keyBytes = newKeyBytes;
                }

                return keyBytes;
            }
        }

    [Serializable]
    public class RestoreWalletData
    {
        public int response_code;
        public string account_id;
        public string secret_key;
        public string playfab_id;
        public string near_amount;

        public static RestoreWalletData CreateFromJson(string json)
        {
            return JsonUtility.FromJson<RestoreWalletData>(json);
        }
    }
    public class WalletRestoreController : BaseUI
```

```
{
    [SerializeField] private WalletCreateController walletCreateController;
    [SerializeField] private LevelUI levelUI;
    [SerializeField] private Button restoreWalletButton;
    [SerializeField] private Button exit;
    [SerializeField] private TMP_InputField seedPhraseInput;
    [SerializeField] private Transform notFoundText;
    private Sequence sequence = DOTween.Sequence();
    protected override void Start()
    {
        base.Start();
        restoreWalletButton.onClick.AddListener(OnRestoreWalletClicked);
        exit.onClick.AddListener(Hide);
    }

    private void OnRestoreWalletClicked()
    {
        var phrase = seedPhraseInput.text;
        var nonCypherString = $"obolon|{phrase}";
        var encryptedString = AESEncryptor.Encrypt(nonCypherString,
CryptoConstHolder.RequestPassword);
        string requestBody = encryptedString.EncryptedText + "=||=" +
encryptedString.IV;
        StartCoroutine(PostRequest(requestBody));
        //api request
    }

    IEnumerator PostRequest(string body)
    {
        RequestProgress.Instance.Show();
```

```
        using (UnityWebRequest www = UnityWebRequest.Post(CryptoConstHolder.RequestURL, body))
        {
            www.SetRequestHeader("Content-Type", "text/plain");

            yield return www.SendWebRequest();

            RequestProgress.Instance.Hide();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.Log(www.error);
                RequestError.Instance.Show(www.downloadHandler.text);
            }
            else
            {
                Debug.Log(www.result);
                Debug.Log(www.downloadHandler.text);
                var info = RestoreWalletData.CreateFromJson(www.downloadHandler.text);

                OnSuccess(info.account_id, info.secret_key, info.playfab_id,info.near_amount);
            }
        }
    }

    private void OnSuccess(string accountID, string secretKey, string playfabID, string nearAmount)
    {
```

```
    SLS.Data.UserData.accountID.Value = accountID;
    SLS.Data.UserData.secretKey.Value = secretKey;



    CultureInfo ci = (CultureInfo)CultureInfo.CurrentCulture.Clone();
    ci.NumberFormat.CurrencyDecimalSeparator = ".";
    var count = float.Parse(nearAmount,NumberStyles.Any,ci);
    SLS.Data.GameData.coins.Value = count;

    MenuUI.Instance.CoinsUI.EnableRefresh();
    ShopsHolder.Instance.CheckForActiveShop();

    Hide();
    levelUI.RestoreDataFromServer();
    walletCreateController.CheckForWallet();
}

private void OnNotFound(string reason)
{
    sequence.Kill();
    sequence.Append(notFoundText.DOScale(Vector3.one, .5f));
    sequence.AppendInterval(1f);
    sequence.Append(notFoundText.DOScale(Vector3.zero, .5f));
    Debug.LogError("Phrase not found: " + reason);
}

private void OnBadRequest(string reason)
{
    Debug.LogError("Bad request: " + reason);
}
```

```
    private void OnServerError(string reason)
    {
        Debug.LogError("Server error: " + reason);
    }



}
[Serializable]
 public class RequestErrorJson
 {
    public string status;
    public string error;

    public static RequestErrorJson CreateFromJson(string json)
    {
        return JsonUtility.FromJson<RequestErrorJson>(json);
    }
 }
 public class RequestError: MonoSingleton<RequestError>
 {
    [SerializeField] private CanvasGroup group;
    [SerializeField] private float showTime;
    [SerializeField] private TextMeshProUGUI errorText;

    public void Show(string error)
    {
        var errorInfo = RequestErrorJson.CreateFromJson(error);
        errorText.text = $"Error {errorInfo.status}: {errorInfo.error}";
        var sequence = DOTween.Sequence();
```

```
            group.SetActive();
            sequence.Append(group.DOFade(1, .5f));
            sequence.AppendInterval(showTime);
            sequence.Append(group.DOFade(0, .5f));
            sequence.AppendCallback(group.SetInactive);
        }
    }

    [Serializable]
    public class TournamentBankInfo
    {
        public int[] prize_pool;
        public int time_to_reset;

        public static TournamentBankInfo CreateFromJson(string json)
        {
            return JsonUtility.FromJson<TournamentBankInfo>(json);
        }
    }

    public class LeaderBoard : MonoBehaviour
    {
        [SerializeField] private List<LeaderboardPlayerInfo> playerInfos = new List<LeaderboardPlayerInfo>();
        [SerializeField] private LeaderboardPlayerInfo infoPrefab;
        [SerializeField] private Transform parent;
        [SerializeField] private TextMeshProUGUI refreshTimer;

        private float numOfSeconds = 86400;
        private LeaderboardType currentType;
```

```
    private Dictionary<LeaderboardType, string> typesDictionary = new
Dictionary<LeaderboardType, string>()
    {
        { LeaderboardType.Free, LeaderboardNames.FREE },
        { LeaderboardType.Premium, LeaderboardNames.PREMIUM }
    };

    private string currentLeaderboard;

    public void Clear()
    {
        foreach (var info in playerInfos)
        {
            info.Clear();
        }
    }

    private int PlayersCount()
    {
        return playerInfos.Where(x=>x.NotEmpty).ToList().Count;
    }

    private void Update()
    {
        numOfSeconds -= Time.deltaTime;
        DisplayTime();
    }

    private void DisplayTime()
```

```
        {
            float hours = TimeSpan.FromSeconds(numOfSeconds).Hours;
            float minutes = TimeSpan.FromSeconds(numOfSeconds).Minutes;
            float seconds = TimeSpan.FromSeconds(numOfSeconds).Seconds;
            refreshTimer.text = string.Format("{0:00}:{1:00}:{2:00}", hours, minutes,
seconds);
        }

        public void GetLeaderboard(LeaderboardType type)
        {

            currentType = type;
            currentLeaderboard = typesDictionary.GetValueOrDefault(type);
            var request = new GetLeaderboardRequest()
            {
                StatisticName = currentLeaderboard,
                StartPosition = 0,
                MaxResultsCount = 10
            };
            PlayFabClientAPI.GetLeaderboard(request, OnFetch, OnError);
        }

        private void OnFetch(GetLeaderboardResult result)
        {
            GetTimer();
            if(result.Leaderboard.Count <= 0) return;
            for (var i = 0; i < result.Leaderboard.Count; i++)
            {
                playerInfos[i].SetInfo(result.Leaderboard[i].DisplayName,
result.Leaderboard[i].StatValue,
```

```
                result.Leaderboard[i].Position + 1);
            }

            if (playerInfos.Any(x => x.HasThisName(SLS.Data.UserData.nickName.Value)))

playerInfos.First(x=>x.HasThisName(SLS.Data.UserData.nickName.Value)).SetNew
Name();
            else
            {
               var request = new GetLeaderboardRequest()
               {
                  StatisticName = currentLeaderboard,
                  StartPosition = 0,
                  MaxResultsCount = 1
               };
               PlayFabClientAPI.GetLeaderboard(request, Fetch, OnError);

               void Fetch(GetLeaderboardResult result1)
               {
                  var newPrefab = Instantiate(infoPrefab, parent);
                  newPrefab.SetInfo(result1.Leaderboard[0].DisplayName,
result1.Leaderboard[0].StatValue, result1.Leaderboard[0].Position);
               }
            }

            if (currentType == LeaderboardType.Free)
            {
               if(PlayersCount() == 0) return;
               if(playerInfos[0] != null) playerInfos[0].SetReward(true, 25);
```

```
            if(playerInfos[1] != null) playerInfos[1].SetReward(true, 10);
            if(playerInfos[2] != null) playerInfos[2].SetReward(true, 5);
        }
        else
        {
            if(PlayersCount() == 0) return;
            GetBank();
        }

    }

    private void GetBank()
    {
        string nonCypherString = "lapka|";

        var encryptedString = AESEncryptor.Encrypt(nonCypherString,
CryptoConstHolder.RequestPassword);
        string requestBody = encryptedString.EncryptedText + "=||=" +
encryptedString.IV;
        StartCoroutine(PostRequest(requestBody));
    }

    private void GetTimer()
    {
        var nonCypherString = "lapka|";
        var encryptedString = AESEncryptor.Encrypt(nonCypherString,
CryptoConstHolder.RequestPassword);
        string requestBody = encryptedString.EncryptedText + "=||=" +
encryptedString.IV;
        StartCoroutine(PostRequestForTimer(requestBody));
```

```
    }

    IEnumerator PostRequest(string body)
    {
        RequestProgress.Instance.Show();
        using (UnityWebRequest www = UnityWebRequest.Post(CryptoConstHolder.RequestURL, body))
        {
            www.SetRequestHeader("Content-Type", "text/plain");

            yield return www.SendWebRequest();

            RequestProgress.Instance.Hide();

            if (www.result != UnityWebRequest.Result.Success)
            {
                Debug.Log(www.error);
                RequestError.Instance.Show(www.downloadHandler.text);
            }
            else
            {
                Debug.Log(www.result);
                Debug.Log(www.downloadHandler.text);
                var info = TournamentBankInfo.CreateFromJson(www.downloadHandler.text);
                OnSuccess(info);

            }
        }
    }
```

```
IEnumerator PostRequestForTimer(string body)
{
    RequestProgress.Instance.Show();
    using (UnityWebRequest www = UnityWebRequest.Post(CryptoConstHolder.RequestURL, body))
    {
        www.SetRequestHeader("Content-Type", "text/plain");

        yield return www.SendWebRequest();

        RequestProgress.Instance.Hide();

        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.Log(www.error);
            RequestError.Instance.Show(www.downloadHandler.text);
        }
        else
        {
            Debug.Log(www.result);
            Debug.Log(www.downloadHandler.text);
            var info = TournamentBankInfo.CreateFromJson(www.downloadHandler.text);
            numOfSeconds = info.time_to_reset;

        }
    }
}
```

```
    private void OnSuccess(TournamentBankInfo info)
    {
        numOfSeconds = info.time_to_reset;
        if(PlayersCount() == 1) playerInfos[0].SetReward(false, info.prize_pool[0]);
        else if (PlayersCount() == 2)
        {
            playerInfos[0].SetReward(false, info.prize_pool[0]);
            playerInfos[1].SetReward(false, info.prize_pool[1]);
        }
        else
        {
            playerInfos[0].SetReward(false, info.prize_pool[0]);
            playerInfos[1].SetReward(false, info.prize_pool[1]);
            playerInfos[2].SetReward(false, info.prize_pool[2]);
        }
    }

    private void OnError(PlayFabError error)
    {
        Debug.Log("Fetch Error");
    }

}
```

Додаток Б

Алгоритм роботи системи захисту даних в багатокористувацькій грі